ГЛАВА 34

Ресурсы и локализация

В связи с тем, что все большее число компаний выходят на интернациональные рынки через Интернет, для успешной работы приложений необходима поддержка различных региональных стандартов. Платформа .NET Framework обладает интегрированной инфраструктурой, которая позволяет создавать интернациональные приложения.

По сути дела, CLR (Common Language Runtime — общеязыковая исполняющая среда) поддерживает механизм упаковки и развертывания ресурсов для приложений любого типа. CLR и библиотека базовых классов платформы .NET Framework имеют несколько классов, предназначенных для управления ресурсами и доступа к ним в приложениях. Эти классы принадлежат пространству имен `System.Resources` и `System.Globalization`.

В этой главе мы поговорим обо всех подробностях работы с ресурсами в приложениях ASP.NET и о создании интернациональных приложений ASP.NET на основе внедренных ресурсов и комплексной поддержки локализации.

Ресурсы в приложениях .NET

Обычно в приложениях используется большое количество изображений и строк для пиктограмм панели инструментов или меню, надписей меню и надписей меток. Попытка изменить эти строки и изображения приведет к неудовлетворительным результатам, если вы поместите их прямо в исходный код. Чтобы иметь возможность изменять эти строки и изображения в программе самым легким из возможных способов, не пересматривая весь исходный код с целью их поиска, вы можете поместить их в отдельные файлы и изменять только в одном месте.

Платформа .NET Framework и CLR обеспечивают всестороннюю поддержку этого подхода посредством внедренных *ресурсов*. Каждую строку, изображение и данные другого типа, которые следует хранить отдельно от исходного кода, можно поместить в отдельные файлы ресурсов. Обычно эти ресурсы компилируются в бинарные файлы самим приложением. Таким образом, они автоматически развертываются вместе с приложением, и никаких дополнительных действий по их развертыванию выполнять не требуется.

Ресурсы используются, прежде всего, при локализации. С их помощью можно определять значения свойств элементов управления (например, можно определить текст для метки — элемент управления `Label`) в различных файлах ресурсов — по одному значению для каждого регионального стандарта, поддерживаемого приложением. Каждый из этих файлов ресурсов содержит строки (пары “ключ-значение”) для локализованных свойств элемента управления, переведенные на язык соответствующего региона. Во время выполнения CLR загружает эти ресурсы из соответствующих файлов внедренных ресурсов и применяет их к свойствам элементов управления.

Нужно сказать, что ресурсы используются не только для целей локализации. В Windows-приложениях ресурсы применяются также для пиктограмм панелей инструментов, меню и состояния с той целью, чтобы не развертывать их отдельно с приложением (это не является обычной практикой для веб-приложений). Ресурсы можно использовать также в специальных классах инсталлятора для внедрения дополнительных сценариев развертывания (например, сценарии для создания базы данных или модификации каталогов COM+), для того чтобы не развертывать их отдельно в рамках всего процесса развертывания.

Определения ресурсов обычно хранятся в файлах `.resx`. Файлы ресурсов `.resx` представляют собой обычные XML-файлы, содержащие либо строковые значения, либо ссылки на внешние файлы. Строки и ссылки на файлы компилируются как внедренные ресурсы в бинарные файлы приложения. Добавить ресурсы в свой проект можно, если щелкнуть правой кнопкой мыши в окне веб-сайта или на проекте в окне Solution Explorer (Проводник решений) и выбрать команду Add New Item (Добавить новый пункт). В появившемся диалоговом окне Add New Item (Добавить новый пункт) нужно выбрать пункт Resource File (Файл ресурса).

Следующий пример показывает фрагмент образца файла `.resx`. Файл содержит простые строки и ссылку на внешний файл изображения.

```
<root>
   <resheader name="resmimetype">
      <value>text/microsoft-resx</value>
   </resheader>
   <resheader name="version">
      <value>2.0</value>
   </resheader>
   ...
   <data name="Binary Code Sm"
            type="System.Resources.ResXFileRef, System.Windows.Forms">
      <value>
         Binary Code Sm.png;System.Drawing.Bitmap, System.Drawing,
         Version=2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a
      </value>
   </data>
   <data name="LegendAge">
      <value xml:space="preserve">Age</value>
   </data>
   <data name="LegendFirstname">
      <value xml:space="preserve">Firstname</value>
   </data>
   <data name="LegendLastname">
      <value xml:space="preserve">Lastname</value>
   </data>
</root>
```

На заметку! Это лишь фрагмент файла `.resx`. Visual Studio обычно автоматически генерирует большое количество комментариев и схему XML для файла ресурсов в одном XML-файле.

Скомпилировать файлы как внедренные ресурсы приложения можно и другим способом: нужно добавить их в проект и выбрать значение `EmbeddedResource` для свойства `BuildAction` в окне Properties (Свойства) файла. Однако этот способ пригоден только для Windows-проектов, таких как библиотеки классов, Windows-службы или приложения Windows Forms. Для приложений ASP.NET должен применяться способ, который будет рассматриваться в этой главе. В любом случае, вы сможете затем обращаться к ресурсам программно посредством класса `ResourceManager`.

На заметку! Ресурсы можно добавлять в приложение как текстовые файлы. Эти текстовые файлы должны состоять из пар "ключ-значение", по одной паре в строке, в формате `ключ = значение`. Их можно компилировать с помощью инструмента `resgen.exe` в двоичный формат с расширением `.resources`. Файлы `.resources` впоследствии можно будет добавлять в свое приложение в качестве внедренных ресурсов (`BuildAction=EmbeddedResource`). С другой стороны, поскольку этот способ управления ресурсами на данный момент устарел, мы не рекомендуем использовать его.

При рассмотрении следующего образца веб-приложения вы узнаете о том, как можно использовать ресурсы для заполнения свойств элементов управления и получения некоторой другой информации, например, о шаблонах документов для генерации отчетов. Веб-приложение потребует от пользователя ввести свое имя, фамилию и возраст. На основании этой информации приложение генерирует фрагмент HTML с помощью XML и XSLT. Для этой цели в ресурсах приложения сохраняются два шаблона — один шаблон представляет структуру XML, используемую в качестве основы для таблицы стилей XSLT, а другой — саму таблицу стилей XSLT.

На заметку! XSLT — это акроним XSL Transformations. Он используется для преобразования структур XML в другие структуры XML или языки, основанные на дескрипторах, такие как HTML. В этой книге мы не будем вдаваться в подробности XSLT. Если вы хотите узнать больше об XSLT, обратитесь к документации MSDN по адресу `http://msdn2.microsoft.com/en-us/library/ms256069.aspx`.

Чтобы реализовать этот пример, мы собираемся использовать очень простую страницу ASP.NET, как показано в следующем фрагменте кода:

```
<%@ Page Language="C#" AutoEventWireup="true"
         CodeFile="Default.aspx.cs" Inherits="Default_aspx" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
   "http://www.w3.org/TR/xhtml11/DTD/xhtml11.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:Label ID="LegendFirstname" runat="server" Text="Label" />
       <asp:TextBox ID="TextFirstname" runat="server"></asp:TextBox><br />
       <asp:Label ID="LegendLastname" runat="server" Text="Label"/>
       <asp:TextBox ID="TextLastname" runat="server"></asp:TextBox>
       <br />
       <asp:Label ID="LegendAge" runat="server" Text="Label"/>
       <asp:TextBox ID="TextAge" runat="server"></asp:TextBox><br />
       <asp:Button ID="GenerateAction" runat="server"
                   Text="Generate Document"
                   OnClick="GenerateAction_Click" /><br />
       <asp:Substitution ID="DocumentSubstitute"
                         runat="server"
                         MethodName="SubstituteHtml" />
    </div>
    </form>
</body>
</html>
```

Как можно видеть, метки в предыдущем коде разметки вообще не инициализируются со значащим текстом для конечного пользователя. Более того, у нас имеется элемент управления `<asp:Substitution />`, который мы используем для отображения результатов преобразования с помощью таблицы стилей XSLT на основании документа XML. Он получает результаты посредством метода, указанного в свойстве `MethodName`. Реализацию этого метода вы увидите далее в этой главе. Обратите внимание на то, что мы не используем элемент управления `<asp:Xml>`, так как для преобразования он требует объект `XslTransform`, который не используется со времени выхода версии .NET Framework 3.5. Об этом мы поговорим чуть позже. Наконец, все строки, а также шаблоны XML и XSLT, внедряются как ресурсы. После того как файл ресурсов сборки будет добавлен в проект, Visual Studio предложит редактор ресурсов для модификации содержимого файла `.resx`. С его помощью вы сможете добавить в свой проект строки и файл любого другого типа, как показано на рис. 34.1.

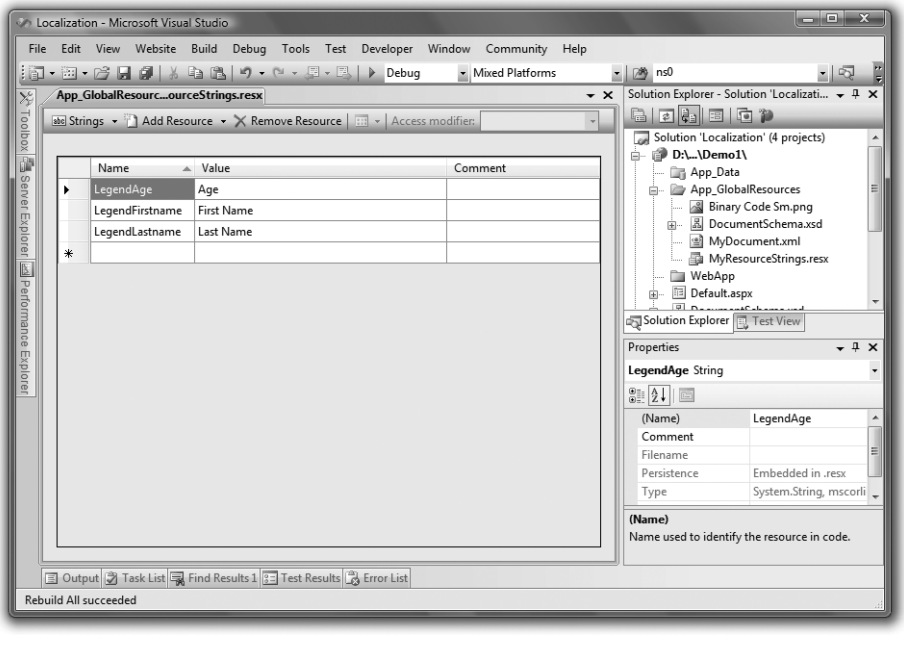

Рис. 34.1. Редактор ресурсов Visual Studio 2008

Первая кнопка в панели инструментов редактора ресурсов показывает тип ресурсов, с которыми вы работаете в настоящий момент. Как видно на рис. 34.1, текущее представление показывает строковые ресурсы. Если щелкнуть на кнопке, вы сможете переключаться между разными представлениями, такими как изображения или другие файлы. Добавлять строковые ресурсы можно посредством ввода значений в последней строке сетки, как показано на рис. 34.1. Если вам понадобится добавить другие типы ресурсов, нужно будет использовать другую кнопку в панели инструментов редактора ресурсов (кнопка Add Resource (Добавить ресурс) на рис. 34.1). С помощью команды Add Existing File (Добавить существующий файл) меню Add Resource (Добавить ресурс) вы можете добавить в свой проект любой внешний файл любого формата, хранящийся на диске. В зависимости от того, какую модель приложения вы используете, Visual Studio применяет разные подходы для добавления этих файлов в ваш проект.

Если использовать модель веб-сайта, представленную в Visual Studio 2005, то эти файлы будут автоматически добавляться в один из специальных каталогов ресурсов веб-сайта ASP.NET (например, `App_GlobalResources`) и компилироваться как внедренные ресурсы в итоговые бинарные файлы во время динамической компиляции при первом запросе.

Когда используются проекты веб-приложений, то файл ресурсов по умолчанию добавляется непосредственно в папку вашего Web-приложения, в которой вы открыли контекстное меню. Наконец, содержимое файла ресурсов добавляется в качестве ресурсов в итоговую сборку, скомпилированную посредством Visual Studio в каталог `Bin` веб-приложения. Дополнительные файлы, такие как изображения, тоже можно добавлять с помощью команды Add Existing File меню Add Resource в редакторе ресурсов, как было сказано в начале этого абзаца. Однако при использовании проектов веб-приложений они добавляются в подкаталог `Resources` вашего веб-приложения и компилируются в результирующий бинарный файл.

Теперь давайте посмотрим на специальные папки ASP.NET для управления ресурсами. По сути, при наличии этих папок Visual Studio и ASP.NET различают глобальные ресурсы и локальные ресурсы. *Глобальные ресурсы* доступны из любой страницы приложения, а *локальные ресурсы* существуют на одной странице и доступны только из нее. Глобальные ресурсы хранятся в специальном каталоге `App_LocalResources` для каждой страницы. Файлы ресурсов, добавленные в каталог `App_LocalResources`, связываются со своей страницей посредством их имени в таком формате: `<Имя_вашей_страницы>.aspx.resx`.

Однако эти папки применяются по умолчанию только тогда, когда вы используете модель веб-сайта. При использовании шаблона проекта веб-приложения Visual Studio выбирает другой подход, о чем было сказано в предыдущем абзаце. Подход, используемый по умолчанию для проектов веб-приложения, всегда создает ресурсы, доступные глобально. Естественно, вы можете использовать также и специальные папки ASP.NET, такие как `App_GlobalResources` или `App_LocalResources` в проектах веб-приложений.

На рис. 34.2 показана текущая компоновка приложения, которое мы используем для изучения основ работы с ресурсами — на этом рисунке приложение работает без использования внедренных ресурсов.

Как можно видеть на рис. 34.2, надписи меток (элементов управления `Label`) не инициализированы. Сейчас мы займемся добавлением кода, необходимого для инициализации текстовых свойств и генерирования простого документа. Рассматривая этот пример, вы узнаете о различных способах доступа к внедренным ресурсам. Более того, вы увидите, что ресурсы могут использоваться для решения различных задач, хотя, конечно, локализация является главной их целью.

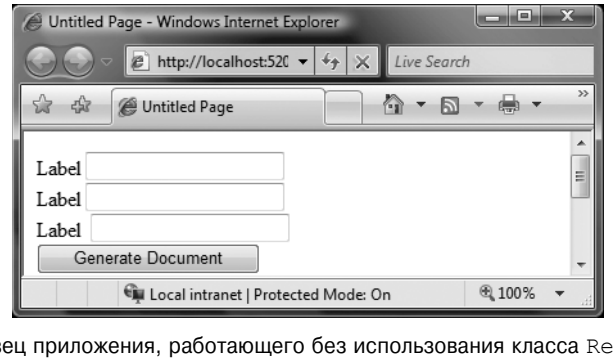

Рис. 34.2. Образец приложения, работающего без использования класса `ResourceManager`

По сути, вы можете обращаться к внедренным ресурсам посредством класса, сгенерированного Visual Studio. Этот класс генерируется на основании информации, хранящейся в файле .resx. Это означает, что когда вы будете создавать или модифицировать ресурсы с помощью редактора ресурсов, Visual Studio внедрит ресурсы в приложение и автоматически создаст строго типизированный класс, необходимый для доступа к этим ресурсам с помощью свойств (имена свойств будут происходить из имен ресурсов, выбираемых вами в редакторе ресурсов). Таким образом, каждая запись в файле ресурсов будет представлять сгенерированный класс, а имя класса будет происходить из имени созданного вами файла ресурсов.

Следующий фрагмент кода демонстрирует, как можно обращаться к ресурсам из ранее добавленного файла MyResourceStrings.resx. С учетом этого, именем класса строго типизированного ресурса является MyResourceStrings, а сам класс содержит три свойства определенных ранее ресурсов.

```
protected void Page_Load(object sender, EventArgs e)
{
    // Простые строковые ресурсы.
    LegendFirstname.Text = Resources.MyResourceStrings.LegendFirstname;
    LegendLastname.Text = Resources.MyResourceStrings.LegendLastname;
    LegendAge.Text = Resources.MyResourceStrings.LegendAge;
}
```

На заметку! Далее, в разделе "Локализация веб-приложений", вы увидите, каким образом Visual Studio 2008 и ASP.NET обеспечивают наилучшую поддержку локализации надписей и других свойств элементов управления. В этой же части главы мы просто хотим продемонстрировать вам низкоуровневый API-интерфейс для управления любым типом ресурсов, а не только локализацию ресурсов.

Внутренне сгенерированный класс использует экземпляр класса ResourceManager, принадлежащий пространству имен System.Resources. Доступ к экземпляру этого класса возможен через свойство ResourceManager сгенерированного класса. Внутренне процедуры свойств для доступа к внедренным ресурсам являются обычными оболочками вызовов одного из методов GetXxx (т.е. GetString или GetStream) данного экземпляра ResourceManager. Например, ресурс, доступ к которому осуществляется с помощью сгенерированной процедуры свойства Resource.MyResourceStrings.LegendAge, также доступен через Resources.MyResourceStrings.ResourceManager.GetString("LegendAge"), как показано на рис. 34.3.

Свойство ResourceManager сгенерированного класса создает экземпляр ResourceManager автоматически, как показано в следующем фрагменте кода.

Рис. 34.3. Методы класса ResourceManager

Естественно, сделать это может любой из вас.

```
ResourceManager ResMgr = new ResourceManager("Resources.MyResourceStrings",
Assembly.GetAssembly(typeof(Resources.MyResourceStrings)));
```

Первый параметр определяет базовое имя ресурсов, которые необходимо загрузить с помощью класса `ResourceManager`. Второй параметр определяет сборку, в которую будут скомпилированы ресурсы. Если ресурсы компилируются в сборку, выполняющую этот код, то достаточно будет воспользоваться методом `GetExecutingAssembly()` класса `System.Reflection.Assembly`. По большому счету это справедливо для библиотек классов или классических Windows-приложений. В ASP.NET делается это немного иначе, поскольку в этой платформе сборки генерируются и компилируются автоматически. Внутренне инфраструктура ASP.NET действительно создает различные сборки для кода страницы и глобальных ресурсов. Имя же динамически сгенерированной сборки является, по сути, неизвестным, поскольку оно определяется еще и инфраструктурой. Таким образом, использование метода `GetExecutingAssembly()` ничего не даст. Как вариант, использование метода `GetAssembly` с описанием типа сгенерированного класса ресурса является возможным вариантом создания специального экземпляра класса `ResourceManager`, так как этот тип находится в сборке, созданной ASP.NET для внедренных ресурсов, содержащихся в этом классе.

Теперь давайте перейдем к следующему разделу — генерирование простого отчета на основе XML-файла и таблицы стилей XSLT, внедренной непосредственно в вашу страницу посредством элемента управления `<asp:Substitution>`, который расположен на вашей странице. Для этой цели потребуется создать файл XML и таблицу стилей XSLT, что можно сделать с помощью любого понравившегося вам редактора. Можно использовать Visual Studio — нужно щелкнуть правой кнопкой мыши на вашем веб-сайте (или на папке внутри веб-сайта), выбрать команду Add New Item (Добавить новый элемент), а затем в диалоговом окне Add New Item выбрать XML File and XSLT File (Файлы XML и XSLT).

Файл XML хранит базовую информацию, которую мы ввели в наших текстовых полях в очень простой структуре XML — вы можете сохранить ее в своем веб-приложении в виде файла `XmlTemplate.xml` в любой папке; ниже показан пример того, как это можно сделать:

```
<PersonTemplate>
  <Firstname></Firstname>
  <Lastname></Lastname>
  <Age></Age>
</PersonTemplate>
```

Содержимое XML-файла определяет инструкции по преобразованию предыдущего XML-файла в действительный HTML-код, который можно внедрить в вашу страницу с помощью элемента управления `<asp:Substitution>` (в предыдущем примере он назывался `SubstituteHtml`). Данная таблица стилей выглядит следующим образом:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
              xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <h1>Person Report</h1>
    <p>
      Этот отчет был сгенерирован с целью демонстрации использования ресурсов.
      Он получает входные данные из файла XML, который хранится в ресурсах
      как шаблон, а также преобразовывает содержимое в приятно оформленный
      отчет (надеемся, что он действительно приятно оформлен).
    </p>
```

```
    <table>
      <tr>
        <td><b>Firstname:</b></td>
        <td>
          <xsl:value-of select=".//Firstname" />
        </td>
      </tr>
      <tr>
        <td><b>Lastname:</b></td>
        <td>
          <xsl:value-of select=".//Lastname"/>
        </td>
      </tr>
      <tr>
        <td><b>Age:</b></td>
        <td>
          <xsl:value-of select=".//Age" />
        </td>
      </tr>
    </table>
  </xsl:template>
</xsl:stylesheet>
```

Для данного примера сохраните файл `ReportTemplate.xslt` в любой папке в веб-приложении. Теперь у вас есть все файлы. Чтобы сделать эти файлы доступными в виде внедренных ресурсов и посредством класса `ResourceManager`, который вы использовали ранее для ваших строк меток, вам необходимо добавить их с помощью редактора ресурсов, показанного на рис. 34.1. Просто откройте редактор ресурсов, дважды щелкнув на файле `.resx`, который вы добавили перед этим (файл `MyResourceStrings.resx` в папке `App_GlobalResources`, как показано на рис. 34.1), после чего в меню Add Resource (Добавить ресурс) выберите команду Add Existing File (Добавить существующий файл). Затем перейдите к ранее созданным файлам `XmlTemplate.xml` и `ReportTemplate.xslt` и добавьте их в ваши глобальные ресурсы. На рис. 34.4 показаны файлы, добавленные вами в ресурсы.

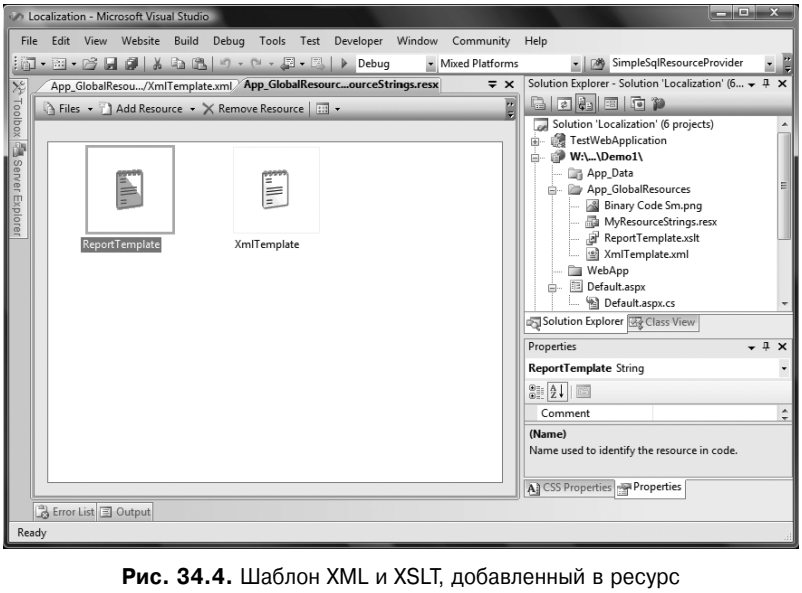

Рис. 34.4. Шаблон XML и XSLT, добавленный в ресурс

Эти ресурсы, основанные на файлах, можно найти, переключившись к представлению Files (Файлы) в редакторе ресурсов. Это можно сделать, щелкнув на первой кнопке в панели инструментов редактора ресурсов, о чем уже было сказано в этом разделе. После того как файл ресурсов будет сохранен, вы сможете получить доступ к содержимому файла посредством следующего сгенерированного класса ресурсов:

```
string XmlTemplate = Resources.MyResourceStrings.XmlTemplate;
string XslTemplate = Resources.MyResourceStrings.ReportTemplate;
```

Поскольку оба файла являются текстовыми, они возвращаются в виде строк из сгенерированного класса ресурсов. Если вы добавите бинарные файлы, такие как изображения, сгенерированный класс вернет их в виде битовых массивов. Вы можете также использовать ранее представленный метод `GetStream()` класса `ResourceManager` для доступа к этим ресурсам посредством потоков (применять потоки будет удобно, если вы захотите создать экземпляры классов, такие как `System.Drawing.Image`, непосредственно из ресурса).

Теперь, когда у вас имеются оба ресурса, вы можете написать код для генерирования отчета, который будет внедрен в вашу страницу посредством элемента управления `DocumentSubstitute` `<asp:Substitution>`, добавленного на страницу в начале этого раздела. В основном вы будете загружать шаблон XML в дерево XML DOM с помощью класса `System.Xml.XmlDocument`. Затем вам нужно будет записать значения для имени, фамилии и возраста из текстовых полей в XML-документ с помощью запросов XPath посредством метода `SelectSingleNode()` из `XmlDocument`.

Это означает, что у вас есть готовый к использованию XML-документ, и вы можете загрузить таблицу стилей XSLT в экземпляр `System.Xml.Xsl.XslCompiledTransform`.

На заметку! В предыдущих версиях .NET Framework существовал класс `XslTransform`, который использовался для выполнения преобразований XSLT в вашем коде. Этот класс и сейчас существует для поддержания обратной совместимости, хотя он был исключен из версии .NET Framework 3.5. Вместо него нужно использовать класс `XslCompiledTransform`, который был впервые представлен в версии .NET Framework 2.0. Этот класс динамически создает скомпилированный класс для выполнения преобразования в фоновом режиме при загрузке документа XSLT, что позволяет существенно повысить производительность преобразований XSLT.

Объект `XslCompiledTransform` выполняет преобразование XSLT из вашей базовой структуры XML в HTML на основе вашей таблицы стилей XSLT. К сожалению, элемент управления `<asp:Xml>` пока что не поддерживает `XslCompiledTransform`. Таким образом, вам придется выполнять преобразование вручную и сохранять результаты в кэше. После этого элемент управления `<asp:Substitution>` получит свое содержимое посредством вызова метода, указанного в его свойстве `MethodName`, который, в свою очередь, получит трансформированный HTML из кэша и вернет его элементу управления в виде строки.

```
protected void GenerateAction_Click(object sender, EventArgs e)
{
    // Теперь получаем XML-файл и шаблон XSLT
    // для генерирования отчета из ресурсов.
    string XmlTemplate = Resources.MyResourceStrings.XmlTemplate;
    string XslTemplate = Resources.MyResourceStrings.ReportTemplate;
    // Загружаем XmlTemplate в DOM и инициализируем его свойства.
    XmlDocument doc = new XmlDocument();
    doc.LoadXml(XmlTemplate);
    doc.SelectSingleNode(".//Firstname").InnerText = TextFirstname.Text;
    doc.SelectSingleNode(".//Lastname").InnerText = TextLastname.Text;
    doc.SelectSingleNode(".//Age").InnerText = TextAge.Text;
```

```
    // Подготавливаем XmlTextReader для загрузки таблицы стилей
    // XSLT, а затем создаем объект XslCompiledTransform
    // для преобразований XSLT.
    XmlTextReader ReaderForXsl = new XmlTextReader(
                                 new StringReader(XslTemplate));
    XslCompiledTransform transform = new XslCompiledTransform();
    transform.Load(ReaderForXsl);

    // Теперь выполняем преобразование вручную с помощью объекта
    // XslCompiledTransform. Для этой цели мы упаковываем XML-содержимое,
    // которое необходимо преобразовать, в XmlTextReader, а искомое
    // содержимое — в XmlTextWriter (нам необходимо использовать
    // StringReader и StringWriter, т.к. напрямую они не поддерживаются).
    StringReader sr = new StringReader(doc.OuterXml);
    XmlTextReader xtr = new XmlTextReader(sr);
    StringWriter sw = new StringWriter();
    XmlTextWriter xtw = new XmlTextWriter(sw);
    transform.Transform(xtr, xtw);
    Cache["Content"] = sw.ToString();
}

public static string SubstituteHtml(HttpContext context)
{
    if (context.Cache["Content"] != null)
    {
        return (string)context.Cache["Content"];
    }
    else
    {
        return "no content generated, yet!";
    }
}
```

После того как вы добавите этот код к обработчику события кнопки `GenerateAction` образца страницы, показанной ранее в этой главе, вы сможете запустить приложение и посмотреть на результаты. В конце концов, ресурсы позволяют вам проложить некоторое подобие обходного пути между статическими ресурсами и вашим приложением. Вместо того чтобы напрямую жестко кодировать строковые ресурсы в вашем коде, или даже формировать шаблоны (как, например, таблица стилей XSLT в предыдущем примере), или путевую информацию для таких шаблонов (в качестве альтернативного варианта показанному выше подходу), вы просто помещаете их в заменяемые ресурсы. И что самое замечательное, ASP.NET автоматически получает доступ к нужным ресурсам для подходящих языков. Например, если в предыдущем примере вы предоставите второй шаблон XSLT и файл `.resx` для другого региона, ASP.NET автоматически будет использовать шаблон XSLT, предназначенный для входящего региона в ваших ресурсах, не требуя от вас писать какой-либо специальный код. Это очень элегантный и умный подход для обеспечения содержимого, не привязанного к какому-либо определенному региону, начиная от простых строк и заканчивая изображениями или любым другим типом ресурсов, специфических для региона. И это как раз та тема, которой будут посвящены следующие разделы этой главы!

Локализация веб-приложений

Инфраструктура, представленная в первой части этой главы, обеспечивает фундамент для локализации любого типа приложения .NET, включая Windows-приложения, библиотеки классов, службы и, естественно, веб-приложения.

Прежде чем перейти к изучению технической стороны локализации приложений, мы обсудим главные проблемы, связанные с этим процессом. На рис. 34.5 показаны некоторые проблемы, которые могут возникнуть при локализации веб-приложения.

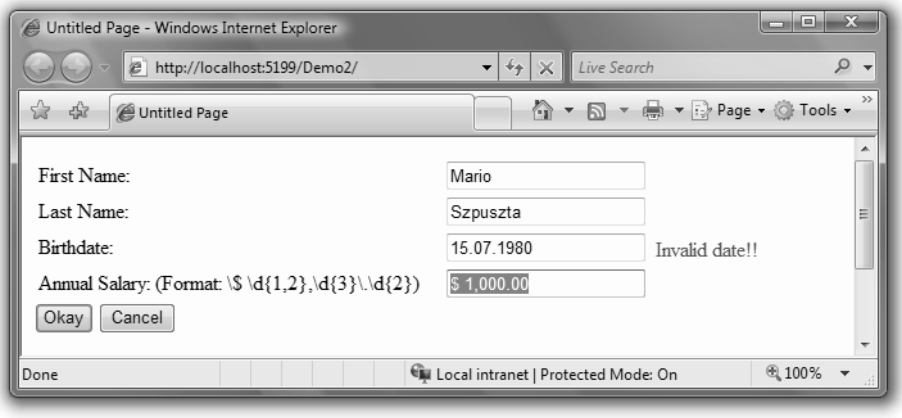

Рис. 34.5. Образец приложения, показывающий проблемы при локализации

Несмотря на то что этот пример является довольно простым, он демонстрирует некоторые из основных проблем, которые могут возникнуть при локализации веб-приложений. Самой простой проблемой, естественно, является локализация строк для надписей меток и кнопок. Также вы должны локализовать формат обозначения иностранных денежных единиц, дат и времени. Для этой цели вы должны локализовать выражение проверки элемента управления проверкой достоверности.

Наконец, при разборе информации в коде для сохранения ее в базе данных или для других действий по обработке нужно учитывать формат дат, времени и обозначения денежных единиц. Все эти значения могут быть различными и поэтому должны настраиваться для каждого отдельного региона.

Локализация и CLR

Обычно ресурсы создаются для каждого региона, который должно поддерживать приложение. В примере, приведенном ранее, вы узнали о том, как создаются эти ресурсы для региона, выбранного по умолчанию, поскольку мы не определяли никакой информации, связанной с тем или иным регионом. Эти ресурсы используются общеязыковой исполняющей средой (CLR) — последним участником процесса локализации.

CLR определяет поведение по поиску ресурсов, специфических для данного региона. Исходя из этого, каждый набор ресурсов должен определять базовое имя, заданное в первой части имени файла ресурсов. Вторая часть имени, которая была опущена в первом примере этой главы, определяет регион. Если “региональная” часть в имени не будет указана, то ресурсы, определенные в файле ресурсов, будут использоваться в качестве ресурсов по умолчанию. Например, если базовым именем внедренного файла ресурса является `MyResourceString.resx`, то именем, специфическим для данного региона, будет `MyResourceStrings.en-US.resx`.

Как показано на рис. 34.6, CLR определяет иерархию регионов и языков. (Чтобы посмотреть весь список регионов и их иерархий, найдите описание класса `CultureInfo` в документации MSDN.)

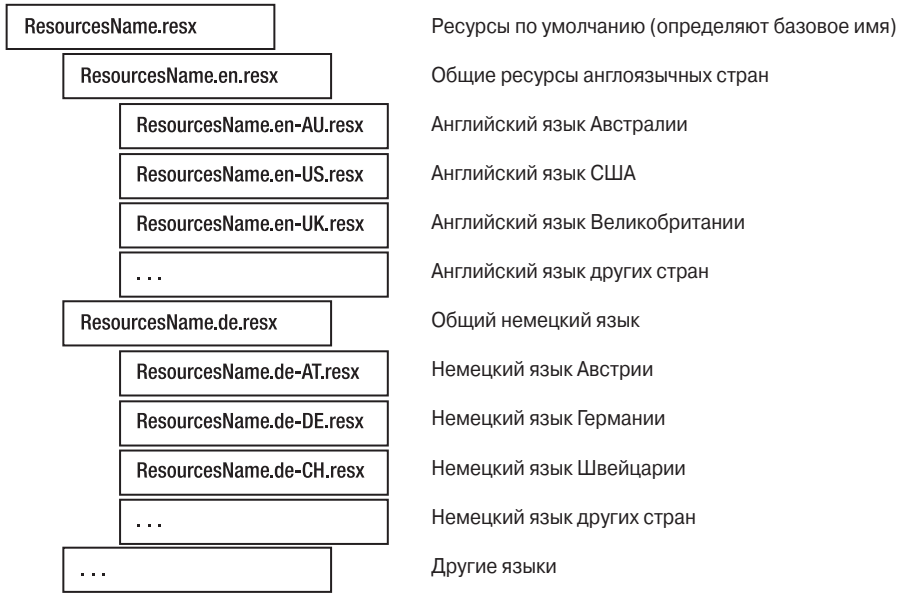

Рис. 34.6. Иерархия регионов, определяемая CLR

CLR автоматически находит ресурсы, специфические для данного региона, на основании этих иерархий. По сути, она пытается найти наиболее соответствующий набор ресурсов для текущих настроек региона. Общие ресурсы региона всегда используются в качестве запасного варианта на тот случай, если ресурсы определенного региона не будут доступными. Более того, если для данного региона в приложении нет внедренных ресурсов, CLR автоматически использует ресурсы по умолчанию, внедренные в приложение. Если ресурсы по умолчанию не включают значение для запрошенного ресурса, CLR генерирует исключение.

При загрузке ресурсов с помощью класса `ResourceManager` всегда определяется только базовое имя. Следовательно, если в проект были добавлены файлы ресурсов с именами `MyResourceStrings.resx` и `MyResourceStrings.en-US.resx`, то при создании экземпляра класса `ResourceManager` будет использоваться только базовое имя, независимо от региона. Например:

```
ResourceManager ResMgr = new ResourceManager("Resources.MyResourceStrings",
              Assembly.GetAssembly(typeof(Resources.MyResourceStrings)));
```

На заметку! ASP.NET автоматически присваивает префикс `Resources` именам ресурсов. Так, базовое имя для (глобальных) ресурсов, добавляемых в проект веб-приложения, всегда будет иметь формат `Resources.MyResource_Имя_Файла`. Префикс `Resources` можно опускать, например, при добавлении ресурсов в библиотеку классов.

Определять файлы ресурсов, специфических для данного региона, не обязательно. CLR может загрузить ресурсы, специфические для данного региона, из файлов ресурсов на основании "текущего" региона (или культуры). А каким является этот текущий регион? HTTP определяет элемент HTTP-заголовка, позволяющий браузеру посылать информацию, специфическую для данного региона, от клиента серверу. На основании этой информации можно создать экземпляр класса `CultureInfo`, как показано в следующем разделе.

Класс CultureInfo

Любая информация, специфическая для данного региона, хранится в экземпляре класса CultureInfo. Этот класс принадлежит пространству имен System.Globalization. Он позволяет запрашивать информацию следующего рода: имя региона, форматы его дат, форматы его чисел, форматы его денежной единицы. Следующий код создает экземпляр класса CultureInfo на основании языка, посланного браузером.

```
protected void Page_Load(object sender, EventArgs e)
{
  CultureInfo ci;
  if ((Request.UserLanguages != null) && (Request.UserLanguages.Length > 0))
  {
     ci = new CultureInfo(Request.UserLanguages[0]);
     System.Threading.Thread.CurrentThread.CurrentUICulture = ci;
  }
  else
  {
     ci = System.Threading.Thread.CurrentThread.CurrentUICulture;
  }
  StringBuilder MessageBuilder = new StringBuilder();
  MessageBuilder.Append("Current culture info: ");
  MessageBuilder.Append("<BR>");
  MessageBuilder.AppendFormat("-) Name: {0}", ci.Name);
  MessageBuilder.Append("<BR>");
  MessageBuilder.AppendFormat("-) ISO Name: {0}", ci.ThreeLetterISOLanguageName);
  MessageBuilder.Append("<BR>");
  MessageBuilder.Append("-) Currency Symbol: " + ci.NumberFormat.CurrencySymbol);
  MessageBuilder.Append("<BR>");
  MessageBuilder.Append("-) Long Date Pattern: " + ci.DateTimeFormat.LongDatePattern);
  LegendCI.Text = MessageBuilder.ToString();
}
```

Объект Request содержит свойство UserLanguages. Это свойство включает информацию о языке, которую послал браузер в HTTP-заголовке серверу. Однако отправка информации о регионе посредством HTTP-заголовка является необязательной и поэтому может быть не доступной для веб-приложения. В этой ситуации можно определить регион, используемый по умолчанию, в файле web.config, как показано далее:

```
<system.web>
   <!-- ... Другие параметры конфигурации ... -->
   <globalization enableClientBasedCulture="true"
                  culture="de-DE"
                  uiCulture="de-DE"/>
   <!-- ... Другие параметры конфигурации ... -->
</system.web>
```

Параметры настройки, определенные в этом элементе конфигурации, автоматически посылаются свойствам Thread.CurrentThread.CurrentUICulture и Thread.CurrentThread.CurrentCulture. CLR использует эти свойства для поиска соответствующей даты во внедренных ресурсах, а также для управления функциями форматирования вроде ToString() или Parse других типов региона, таких как DateTime или Decimal.

Таким образом, если браузер посылает информацию, специфическую для данного региона, от клиента, вы должны переопределить параметры настроек, как было показано в предыдущем примере кода в начале события Page_Load. Свойство CurrentCulture влияет на поведение функций форматирования. Например, при вызове DateTime.Now.ToString или DateTime.Now.Parse будет использоваться формат даты, основанный на свойстве CurrentCulture потока. То же самое касается и форматов чисел.

С другой стороны, класс ResourceManager использует свойство CurrentCulture для поиска ресурсов, специфических для данного региона. Это означает, что при получении запроса от браузера ASP.NET автоматически инициализирует свойство CurrentCulture экземпляра System.Threading.Thread.CurrentThread. На основании этого свойства CurrentUICulture, а также на основании иерархии регионов, определяемой CLR (показана на рис. 34.6), класс ResourceManager автоматически получает локализованные ресурсы от соответствующих внедренных ресурсов при вызове одного из методов GetXxx. В табл. 34.1 представлены различные примеры этого поведения, если для приложения присутствуют следующие файлы ресурсов.

- MyResources.resx — ресурсы по умолчанию со значениями для FirstName, LastName, Age и DocumentName.
- MyResources.en.resx — ресурсы по умолчанию для англоязычных регионов со значениями для Firstname, Lastname и Age.
- MyResources.de.resx — ресурсы по умолчанию для регионов немецкого языка со значениями Firstname, Lastname и Age.
- MyResources.de-DE.resx — ресурсы для немецкого языка Германии со значениями для Firstname и Lastname.
- MyResources.de-AT.resx — ресурсы для немецкого языка Австрии со значениями для Firstname и Lastname.

Таблица 34.1. Примеры поведения ResourceManager в различных ситуациях

CurrentUICulture	Вызов метода	Результат
en-US	GetString("Firstname")	Используется значение Firstname ресурсов MyResources.en.resx, поскольку в приложении отсутствуют ресурсы для английского языка США.
en-GB	GetString("Firstname")	Используется значение Firstname ресурсов MyResources.en.resx, поскольку в приложении отсутствуют ресурсы для английского языка Великобритании.
en-US	GetString("DocName")	Используется значение DocumentName ресурсов MyResources.resx, поскольку в файле ресурсов не определены значения для английского языка для ключа DocName.
de-DE	GetString("Firstname")	Используется значение Firstname ресурсов MyResources.de-DE.resx.
de-DE	GetString("Age")	Используется значение Age ресурсов MyResources.de.resx, поскольку значение не определено в файле MyResources.de-DE.resx.
de-AT	GetString("Lastname")	Используется значение Lastname ресурсов MyResources.de-AT.resx.
de-AT	GetString("DocumentName")	Используется значение DocumentName ресурсов MyResources.resx, поскольку этого значения нет в ресурсах MyResources.de-AT.resx и MyResources.de.resx.

Локальные ресурсы для одной страницы

Классы, с которыми мы работали до настоящего времени, обеспечивают базовую инфраструктуру для локализации .NET-приложений любого типа. В версиях ASP.NET 1.*x* эту инфраструктуру нужно было использовать для локализации содержимого ваших элементов управления вручную.

К счастью, с выходом версии ASP.NET 2.0 и Visual Studio 2005 все изменилось, и теперь они поддерживают веб-разработчика в локализации в той же мере, что и Windows Forms с самого начала. Поскольку ASP.NET 3.5 основана на версии ASP.NET 2.0, базовая инфраструктура осталась той же, что и в прежней версии — ASP.NET 3.5 и Visual Studio 2008 не могут похвастаться какими-либо серьезными изменениями. Чтобы локализовать страницу, достаточно выбрать в меню Tools (Сервис) команду Generate Local Resources (Сгенерировать локальные ресурсы) — но только вы должны знать, что при этом необходимо, чтобы ваша страница была открыта в окне проектирования; этот вариант не поддерживается в окне кода или раздельном окне (при котором отображается и код, и визуальный дизайн) в Visual Studio 2008. Visual Studio сгенерирует в папке `App_LocalResources` файл ресурсов, который будет включать значения для каждого элемента управления страницы, открытой на данный момент в окне конструктора. На рис. 34.7 показаны ресурсы, сгенерированные для приведенного ранее примера.

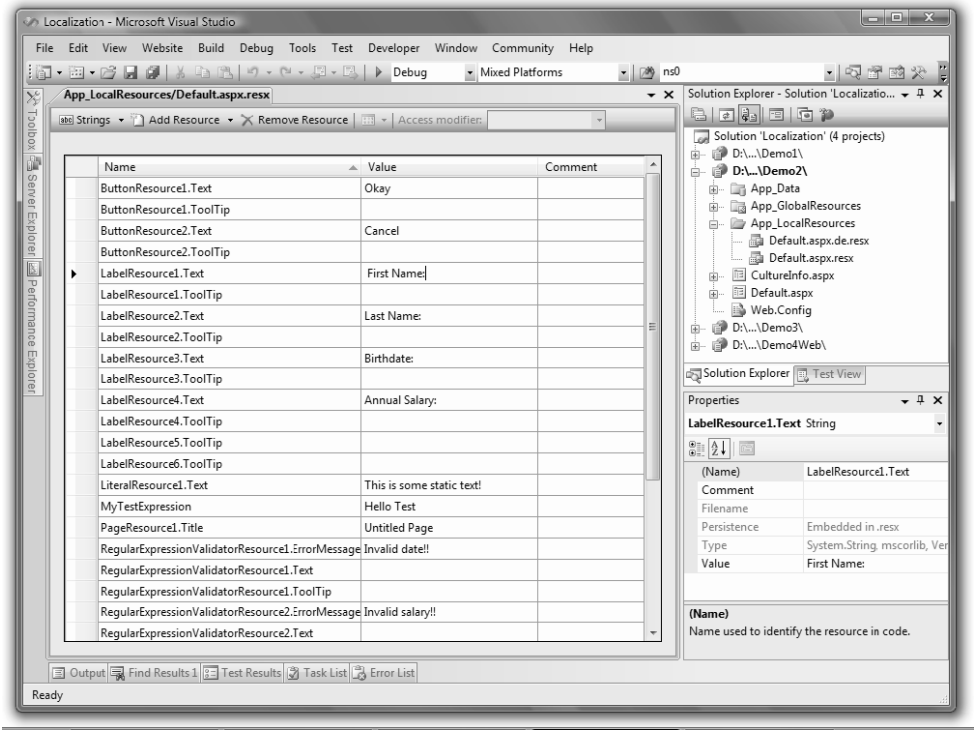

Рис. 34.7. Сгенерированные ресурсы для приложения, представленного на рис. 34.5

Visual Studio генерирует ресурсы для нескольких свойств каждого элемента управления. Ресурсы всегда имеют префикс, обозначающий имя элемента управления и суффикс, обозначающий имя свойства. Visual Studio автоматически генерирует ресурсы по

умолчанию только для тех элементов управления, которые находятся на странице. Все последующие ресурсы, специфические для данного региона, вам придется добавлять вручную, копируя сгенерированные ресурсы и присваивая им соответствующее имя (например, `Default.aspx.en-US.resx`).

Инструментальное средство генерации ресурсов создает запись для каждого свойства, отмеченного с помощью атрибута `[Localizable]` в элементе управления. Таким образом, если вы хотите создать специальный локализуемый элемент управления, нужно отметить все локализуемые свойства с помощью этого атрибута. Например:

```
[Localizable(true)]
public string MyProperty
{
    get { ... }
    set { ... }
}
```

Если скопировать созданные ранее ресурсы и сохранить их в файле `Default.aspx.de.resx`, то в приложение будут добавлены ресурсы, специфические для региона немецкого языка. После этого среда времени выполнения сможет инициализировать свойства элемента управления на основании свойства `CurrentUICulture` потока с использованием строк, содержащихся в файле внедренных ресурсов для данного региона. На рис. 34.8 показан адаптированный файл ресурсов, на рис. 34.9 — результаты просмотра с локальными настройками `German`, а на рис. 34.10 — результаты просмотра с локальными настройками `English`.

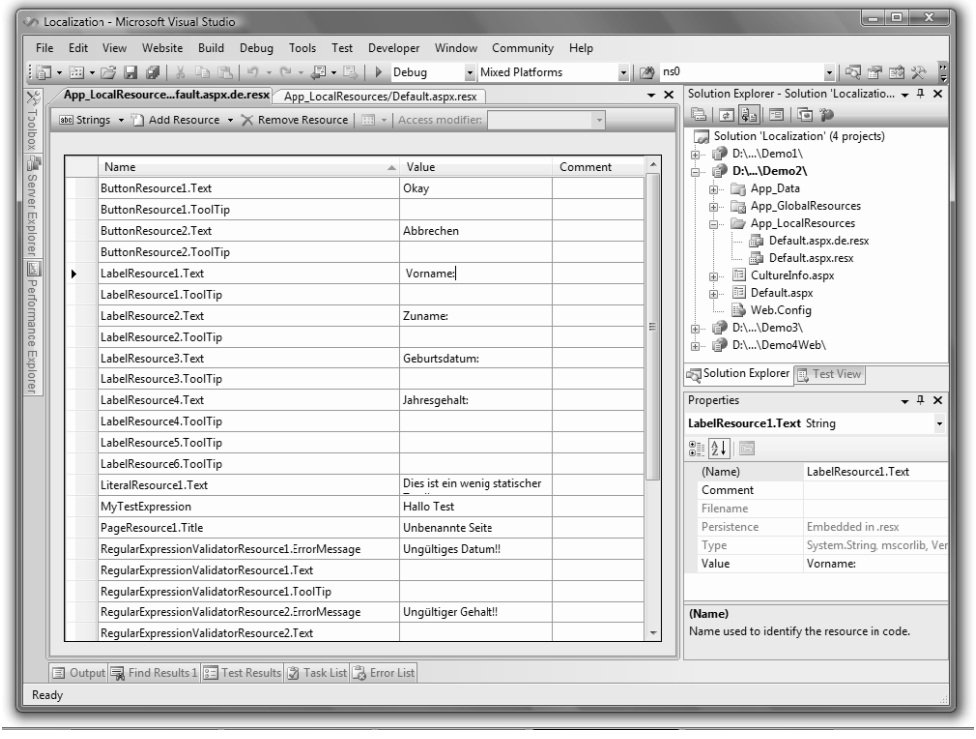

Рис. 34.8. Файл дополнительных ресурсов для другого региона

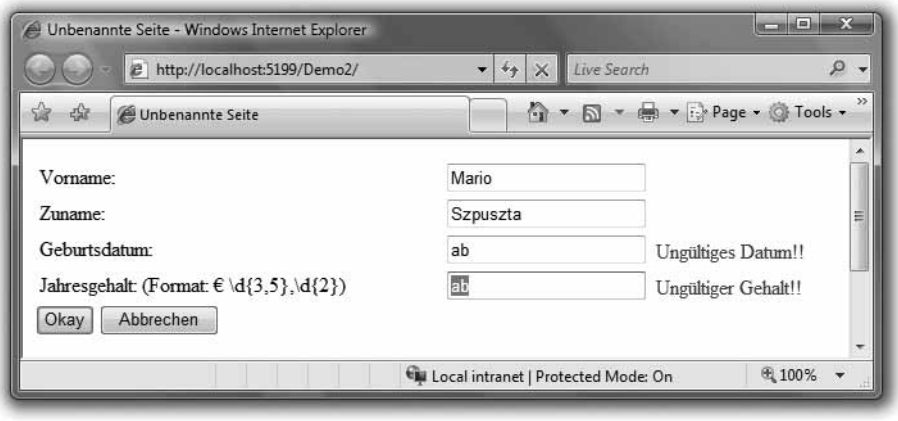

Рис. 34.9. Просмотр с локальными настройками `German`

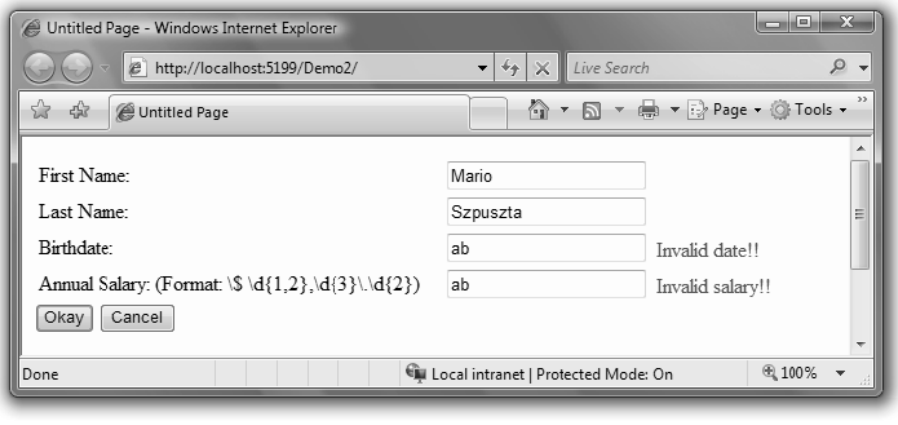

Рис. 34.10. Просмотр с локальными настройками `English`

Кроме генерации файла ресурсов, Visual Studio изменила исходный код страницы. Для каждого локализуемого (`[Localizable]`) свойства каждого элемента управления, расположенного на странице, было добавлено выражение локализации, как показано в следующем фрагменте кода:

```
<asp:Label ID="LegendFirstname" runat="server" Text="Firstname:"
        meta:resourcekey="LabelResource1"></asp:Label>
```

Выражения локализации идентифицируются посредством атрибута `meta:resourceKey` дескриптора. При разборе страницы среда времени исполнения перебирает все элементы управления и генерирует необходимый код для получения ресурса с помощью класса `ResourceManager`. Декларативные назначения свойств остаются незатронутыми и будут отображены в режиме проектирования. К сожалению, имена, которые Visual Studio присваивает ключам ресурсов (например, `Label1Resource1`), не являются производными от значения идентификатора элемента управления. Они генерируются из имени типа элемента управления и локализованного свойства этого элемента управления, и они включают одно число, которое увеличивается на единицу для каждого экземпляра элемента управления от данного типа (т.е. `LabelResource1.Property`,

LabelResource2.Property, LabelResource3.Property для меток). Вместе с "имя-тип" число, сгенерированное для каждого экземпляра этого типа, а также свойство уникально идентифицируют ресурс.

Выражение локализации в предыдущем фрагменте кода является так называемым неявным выражением локализации. *Неявные выражения локализации* напоминают сокращения для ключей ресурсов, включенных во внедренные ресурсы для страницы. Они должны быть выполнены в соответствии с соглашениями об именовании, используемыми в Visual Studio для генерирования ресурсов (например, Ключ_Ресурса.Имя_Свойства). Неявные выражения локализации просто определяют базовый ключ ресурса для внедренного ресурса без имени свойства. Имена свойства происходят из второй части имени.

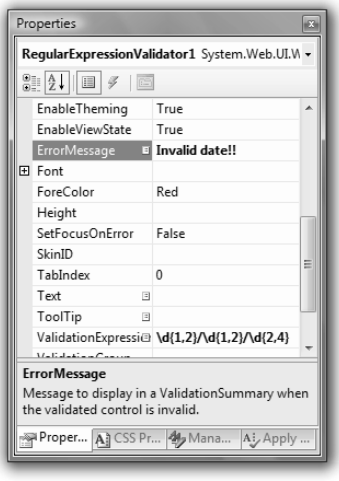

Рис. 34.12. Локализованные атрибуты, отмеченные в окне Properties (Свойства)

Таким образом, вы можете использовать неявные выражения локализации только для локализуемых ([Localizable]) свойств. Более того, их нельзя применять для глобальных ресурсов приложений. Другой способ привязки свойств элементов управления к ресурсам заключается в использовании *явных выражений локализации*. Они обеспечивают большую гибкость за счет привязки любого свойства элемента управления к внедренным ресурсам, и работают с глобальными ресурсами приложений. Более подробно о явных выражениях локализации мы поговорим в следующем разделе, где будут рассматриваться подробности глобальных ресурсов.

Если внимательно посмотреть на сгенерированные файлы ресурсов на рис. 34.7 и 34.8, то станет ясно, что работа до конца еще не сделана. Хотя элемент управления RegularExpressionValidator присутствует в сгенерированных ресурсах, выражение проверки не включено, поскольку оно не отмечено атрибутом [Localizable]. Однако проверка даты рождения и годового заработка должна осуществляться на основании региональных настроек пользователя, просматривающего страницу, поскольку посетителю из США будет удобно добавлять дату рождения в привычном для себя формате (то же самое можно сказать и в отношении посетителей из Германии, Австрии и других стран).

Таким образом, мы должны сделать еще кое-что, дабы завершить работу по локализации приложения. По сути, для локализации проверки двух текстовых полей можно воспользоваться двумя способами. Первый из них состоит в том, чтобы автоматически генерировать регулярное выражение проверки на основании объекта CultureInfo, созданного для региона пользователя. Второй способ заключается в добавлении записи во внедренные ресурсы для выражения проверки. Поскольку мы решили рассмотреть работу явных выражений локализации, мы остановимся на втором способе.

Во-первых, нужно добавить во внедренные ресурсы две новых записи, которые будут содержать регулярное выражение для проверки данных, введенных пользователем — даты рождения и годового заработка. Чтобы сделать это, выполните, пожалуйста, следующее.

1. Откройте файлы локальных ресурсов для разных регионов вашей текущей страницы. Если вы работаете со страницей Default.aspx, как в предыдущих разделах этой главы, то это будут файлы Default.aspx.resx и Default.aspx.de.resx, как показано на рис. 34.8.

2. Добавьте новую запись `RegularExpressionValidatorResource1.DateFormat` в каждый из этих файлов; одна из них будет содержать регулярное выражение для региона US English (в файле `Default.aspx.resx`), а другая будет содержать регулярное выражение для региона German (в файле `Default.aspx.de.resx`) со значением `\d{2}\.\d{2}\.\d{4}`.
3. Теперь добавьте новую запись `RegularExpressionValidatorResource2.SalaryFormat` в каждый из файлов ресурсов. Значением для нее в файле `Default.aspx.resx` должно быть регулярное выражение, представляющее действительный формат заработной платы для США (`\$ \d{1,2},\d{3}\.\d{2}`), а в файле `Default.aspx.de.resx` — регулярным выражением, представляющим европейский формат заработной платы (`\d{3,5},\d{2}`). На рис. 34.11 показано добавленное содержимое для стандартного файла ресурсов `Default.aspx.resx` с выражениями для добавленного региона US English.
4. Сохраните оба файла ресурсов, чтобы сделать ресурсы доступными для ваших страниц.

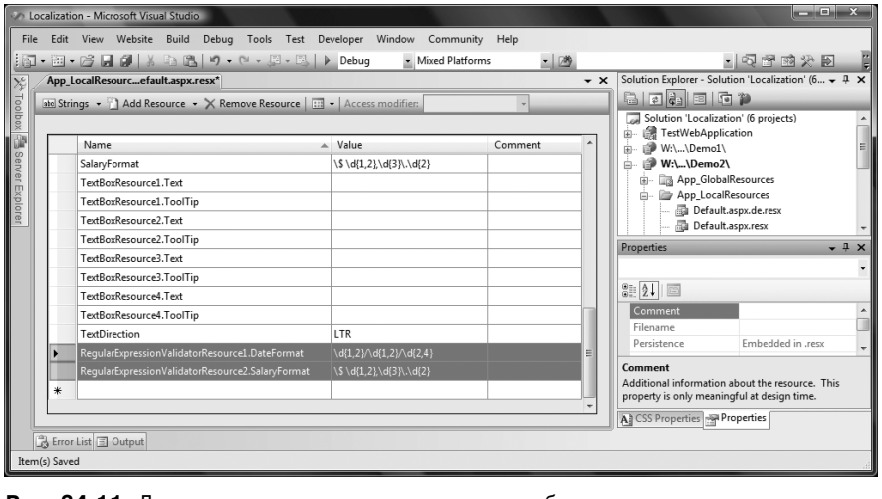

Рис. 34.11. Дополнительные значения ресурсов, добавленные в локальные ресурсы для файла `Default.aspx`

После этого нужно будет изменить определение этих элементов управления следующим образом (предполагая, что записями ресурсов являются `RegularExpressionValidatorResource1.DateFormat` и `RegularExpressionValidatorResource2.SalaryFormat`):

```
<asp:RegularExpressionValidator
       ID="RegularExpressionValidator1" runat="server"
       ControlToValidate="BirthdateText"
       ErrorMessage="Invalid date!!"
       ValidationExpression=
           '<%$ Resources: RegularExpressionValidatorResource1.DateFormat %>'
       meta:resourcekey="RegularExpressionValidatorResource1">
</asp:RegularExpressionValidator>
<asp:RegularExpressionValidator
       ID="RegularExpressionValidator2" runat="server"
       ControlToValidate="SalaryText"
       ErrorMessage="Invalid salary!!"
       ValidationExpression=
```

```
            '<%$ Resources: RegularExpressionValidatorResource2.SalaryFormat %>'
        meta:resourcekey="RegularExpressionValidatorResource2">
</asp:RegularExpressionValidator>
```

Легко заметить, предыдущий элемент управления проверкой достоверности содержит некоторый статический текст — в данном случае это `ErrorMessage`. Это не должно вас беспокоить. Так как элемент управления проверкой имеет атрибут `meta:resourcekey`, элемент управления проигнорирует статический текст, и среда исполнения получит свои данные из сгенерированных ресурсов. До тех пор, пока элемент управления будет иметь атрибут `meta:resourcekey`, он будет игнорировать статический текст и считывать всю информацию из внедренных, локализованных ресурсов. В случае `ValidationExpression` вы должны будете использовать явные выражения проверки, поскольку для этого свойства не предусмотрена автоматическая локализация. Общий формат для явных выражений локализации имеет следующий синтаксис:

```
<%$ Resources: [Ключ_Приложения, ] Ключ_Ресурса %>
```

Ключ приложения обозначает общие ресурсы приложений, и поэтому может быть опущен (что и сделано в предыдущем примере) при доступе к локальным ресурсам. На рис. 34.12 показано, что локализованные свойства обозначаются специальными пиктограммами в окне Properties (Свойства). Выражения локализации сами усиливают новый механизм выражения, включенный в ASP.NET.

Вы можете редактировать выражения для всех свойств с помощью нового редактора выражений. В верхней части окна Properties (Свойства) содержится новое свойство Expressions (Выражения). Если щелкнуть на кнопке с изображением троеточия, откроется редактор выражений, как показано на рис. 34.13. Свойство `ClassKey` определяет параметр `ApplicationKey` для явного выражения локализации, а другое свойство задает ключ ресурса. Фактически оно определяет имя класса, сгенерированного для глобальных ресурсов.

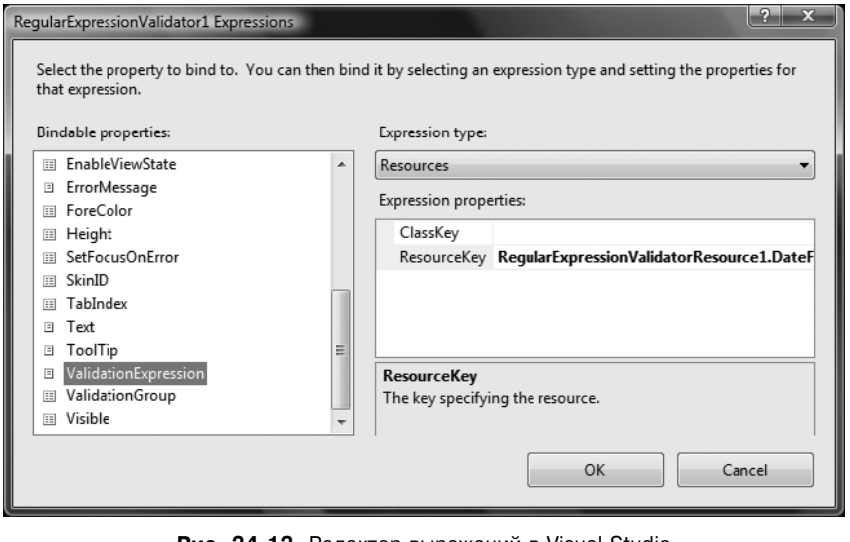

Рис. 34.13. Редактор выражений в Visual Studio

Совместное использование ресурсов страницами

Генерация локальных ресурсов для одной страницы может привести к дублированию строк ресурсов или другой информации ресурсов. Поэтому имеет смысл разделять ресурсы между страницами посредством глобальных ресурсов.

Инструментальная поддержка разделяемых ресурсов в контексте генерирования и привязки ресурсов к элементам управления осуществляется не на таком уровне, как для локальных ресурсов страниц. В первом примере этой главы вы могли видеть, как используются глобальные ресурсы приложений, которые можно разделять между множеством страниц. Если вы захотите добавить ресурсы сборки в проект, Visual Studio спросит вас о том, нужно ли помещать эти ресурсы в каталог глобальных ресурсов. Если ресурсы помещаются в этот каталог, они будут доступны из всех страниц.

Поскольку используемые ранее выражения проверки форматов даты и форматов чисел определенно подходят для повторного использования в нескольких страницах приложения, то лучше всего поместить их в файл глобальных ресурсов. Для этого потребуется сначала добавить новый файл глобальных ресурсов, а затем добавить в эти ресурсы значения, как показано на рис. 34.14.

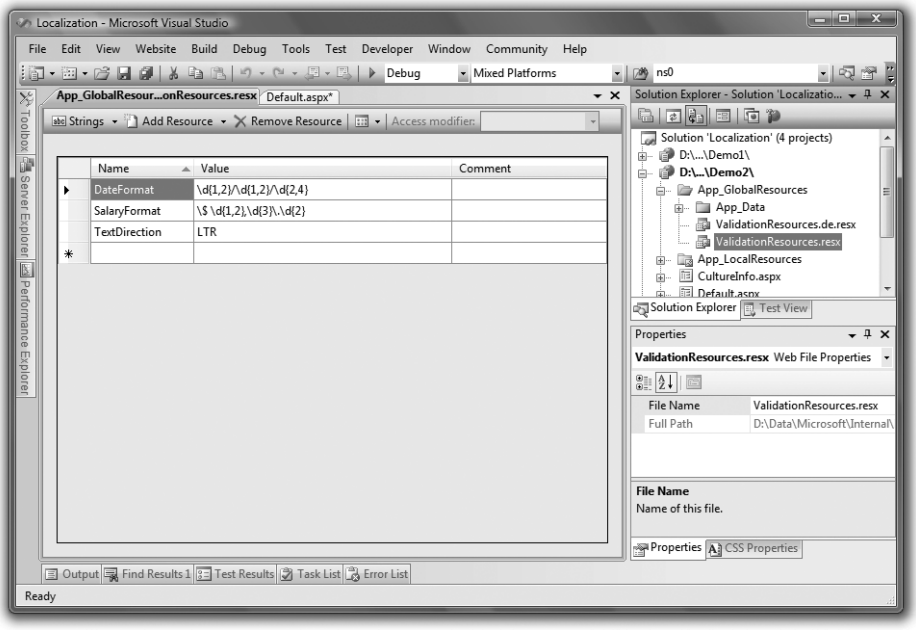

Рис. 34.14. Глобальные ресурсы, добавленные в приложение

Теперь нужно принять явные выражения локализации для двух элементов управления проверкой достоверности. Для этого необходимо изменить имя и добавить параметр `ApplicationKey`. В связи с тем, что ранее вы присвоили файлу глобальных ресурсов имя `ValidationResources.resx`, это имя и будет являться значением для свойства `ApplicationKey` (без расширения `.resx`).

```
<asp:RegularExpressionValidator ControlToValidate="BirthdateText"
    ErrorMessage="Invalid date!!" ID="RegularExpressionValidator1"
    runat="server"
    ValidationExpression='<%$ Resources:ValidationResources, DateFormat %>'
    meta:resourcekey="RegularExpressionValidatorResource1" />
```

Другое существенное отличие между локальными и глобальными ресурсами заключается в программном способе доступа к этим ресурсам. Несмотря на то что доступ к глобальным ресурсам может осуществляться строго типизированным образом через сгенерированную страницу классов ресурсов, доступ к локальным ресурсам должен осуществляться с помощью метода `GetLocalResourceObject`.

```
this.GetLocalResourceObject("LabelResource1.Text")
```

На заметку! Выражения локализации и другие типы выражений работают с элементами управления только с атрибутом `runat="server"`. Их обработка на стороне сервера ложится на плечи инфраструктуры локализации ASP.NET. Следовательно, если вы хотите локализовать элементы управления HTML с помощью явных или неявных выражений локализации, потребуется добавить атрибут `runat="server"`.

По сути, Visual Studio генерирует ресурсы для каждого локализуемого (`[Localizable]`) свойства каждого элемента управления страницы или пользовательского элемента управления, открытого в окне конструктора, в уже описанном формате именования `Ключ_Ресурса.Имя_Свойства`. Если свойство элемента управления уже привязано к ресурсу с помощью явного выражения локализации, то генерация ресурсов для этого свойства будет полностью опущена. Поэтому, чтобы локализовать остальные свойства, для элемента управления включен атрибут `meta:resourceKey`.

Между прочим, Visual Studio автоматически генерирует запись ресурса для заголовка страницы и добавляет атрибут `meta:resourceKey` в директиву `Page`. Вы можете применить явные выражения локализации к директиве `Page`, как показано ниже:

```
<%@ Page Language="C#" ... Culture="auto"
                       UICulture="auto"
                       meta:resourcekey="PageResource1" %>
<%@ Page Language="C#" ... Culture="auto"
                       UICulture="auto"
        Title='<%$ Resources:ValidationResources, DateFormat %>' %>
```

В первом примере свойства страницы, такие как заголовок, неявным образом привязываются к локальным ресурсам, а во втором примере свойство `Title` страницы привязывается к глобальному ресурсу с именем `DateFormat`, хранящемуся в классе `ValidationResources`.

Более того, вы можете определить настройки региона по умолчанию и пользовательского интерфейса на уровне страницы в дополнение к конфигурации в продемонстрированном ранее файле `web.config`.

Локализация статического текста

Мы обсудили, как использовать элементы управления HTML с атрибутом `runat="server"` только для локализации. Поскольку их обработка осуществляется на стороне сервера, локализация свойств этих элементов управления осуществляется довольно просто. А что можно сказать относительно статического текста и направлений чтения текста?

Ответ на этот вопрос очень прост. ASP.NET 2.0 (а значит, и ASP.NET 3.5) включает новый элемент управления, основанный на хорошо известном элементе управления `Literal`, который был добавлен в библиотеку элементов управления. Этот новый элемент управления называется `Localize` и он должен охватывать статический текст, как показано в следующем фрагменте кода:

```
<asp:Localize runat=server
    meta:resourcekey="LiteralResource1"> is some static text!</asp:Localize>
```

Если этот элемент управления будет охватывать некоторый текст, то текстовая часть страницы будет автоматически включена в процесс генерирования ресурсов подобно любому другом элементу управления. Главное отличие между элементами управления `Literal` и `Localize` связано с поведением конструктора. Содержимое элемента управления `Literal` нельзя редактировать в конструкторе, а вот текст, охватываемый элементом управления `Localize`, можно редактировать подобно любому другому текстовому содержимому страницы.

Направления чтения текста

В заключение, вам нужен способ, позволяющий определять направления чтения текста в интернациональных приложениях, поскольку в одних регионах чтение производится слева направо, а в других — наоборот.

Для этой цели можно воспользоваться парой элементов управления ASP.NET, таких как элементы управления `Panel` и `WebPart`. Следовательно, имеет смысл определить свойство в глобальных или локальных ресурсах для направления чтения текста и применять явные выражения локализации, чтобы настроить свойство направления чтения этих элементов управления. Чтобы задать это свойство прямо в корневом элементе вашего HTML-файла, используйте атрибут `runat="server"` для самого дескриптора `<html>`, после чего примените к нему явные выражения локализации, как показано в следующем фрагменте кода:

```
<html runat="server"
      dir='<%$ Resources:ValidationResources, TextDirection %>'
      xmlns="http://www.w3.org/1999/xhtml" >
...
</html>
```

Динамическое переключение локали

По умолчанию ASP.NET усиливает заголовок HTTP языка пользователя, чтобы определить, какой язык следует использовать для локализации текущей страницы. Однако часто разработчики веб-сайтов желают предоставить своим пользователям возможность выбирать язык, с которым они хотят работать, в списке доступных языков. Это бывает очень удобно, когда пользователь не является владельцем компьютера, за которым он работает, и, поэтому, он не имеет возможности изменить региональные настройки в компьютере или браузере. В такой ситуации пользователю, естественно, хочется просматривать ваше веб-приложение на том языке, который выберет именно он.

В действительности, ASP.NET предлагает вам простой способ динамического переключения языка, используемый для любой операции, связанной с локализацией. Единственное, что вам нужно понять в этом отношении, это то, что локализация самой страницы и каких-либо функций, которые предлагает базовая библиотека классов (такие как `DateTime.Parse()` или `DateTime.ToString()`), зависит от свойств `Culture` и `UICulture` класса `Page`. Единственной трудной задачей является настройка этих свойств в подходящий момент жизненного цикла страницы. Чтобы сделать эту задачу как можно легче, базовый класс `System.Web.UI.Page` позволяет вам переопределять метод, который вызывается инфраструктурой в подходящее время. Этим методом является `InitializeCulture`. Вам нужно лишь задать свойства `Culture` и `UICulture` класса `Page`. Эти свойства напрямую сопоставляются со свойствами `CurrentCulture` и `CurrentUICulture` в `CurrentThread` — статическим свойством класса `System.Threading.Thread` — и возвращает экземпляр активному в данный момент времени потоку. Следовательно, понятно, что свойство `Culture` страницы управляет поведением методов, связанных с регионами, в классах (например, `DateTime.Parse()`), в то время как свойство `UICulture` влияет на локализацию элементов управления, добавленных на страницу.

Ниже показан пример кода, который отображает страницу, содержащую метку заголовка, раскрывающийся список DropDownList с доступными языками и кнопку для переключения языков. Версия региона по умолчанию для соответствующего файла ресурсов (Default.aspx.resx) показана на рис. 34.15.

```
<%@ Page Language="C#" AutoEventWireup="true"
         CodeFile="Default.aspx.cs" Inherits="_Default"
         Culture="auto" UICulture="auto"
         meta:resourcekey="PageResource1" %>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
   <title>Testing Dynamic Localization</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
   <h1>
      <asp:Label runat="server" ID="TitleLabel"
                 meta:resourcekey="TitleLabelResource1" />
   </h1>
   <asp:Label runat="server" ID="SelectionLegend"
              meta:resourcekey="SelectionLegendResource1" /><br />
   <asp:DropDownList runat="server" ID="LanguageList"
              meta:resourcekey="LanguageListResource1" Width="166px">
   <asp:ListItem Text="German" Value="de-at"
              meta:resourcekey="ListItemResource1" />
   <asp:ListItem Text="English" Value="en-us"
              meta:resourcekey="ListItemResource2" />
   </asp:DropDownList>
   <br />
   <asp:Button runat="server" ID="SwitchCommand"
               meta:resourcekey="SwitchCommandResource1"
               OnClick="SwitchCommand_Click" />
   </div>
   </form>
</body>
</html>
```

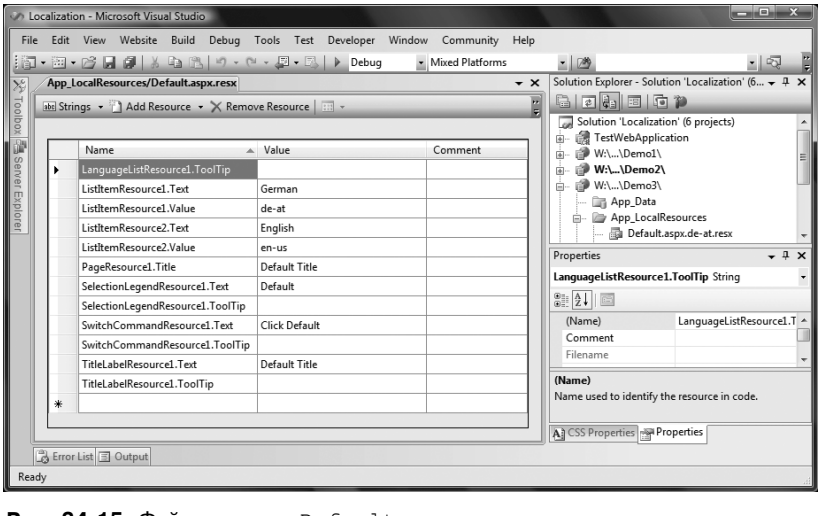

Рис. 34.15. Файл ресурсов Default.aspx.resx для предыдущего примера

Как можно видеть, все эти элементы управления имеют ключ ресурса, связанный с ними для локализации. Обратите внимание на то, что свойства `Culture` и `UICulture` тоже задаются в директиве `Page`. Визуальный дизайнер определяет значение по умолчанию `auto`. Это означает, что страница получает необходимую информацию для локализации из языков пользователя, отправленных в заголовке HTTP. Когда пользователь щелкает на кнопке, приложение должно изменить локали в соответствии с языком, выбранным в раскрывающемся списке `LanguageList`. Это можно сделать следующим образом:

```
public partial class _Default : System.Web.UI.Page
{
    private const string LanguageSessionKey = "CurrentLanguage";
    protected override void InitializeCulture()
    {
        base.InitializeCulture();
        if (Session[_Default.LanguageSessionKey] != null)
        {
            // Задаем текущий регион для страницы.
            Page.UICulture = Session[_Default.LanguageSessionKey].ToString();
            Page.Culture = Session[_Default.LanguageSessionKey].ToString();
        }
    }
    protected void SwitchCommand_Click(object sender, EventArgs e)
    {
        if (LanguageList.SelectedIndex >= 0)
        {
            Session[_Default.LanguageSessionKey] = LanguageList.SelectedValue;
            Response.Redirect(Request.Url.ToString());
        }
    }
}
```

В файле отделенного кода для страницы вам нужно переопределить метод `InitializeCulture`, который вызывается инфраструктурой для инициализации любой информации, необходимой для локализации. Во-первых, метод вызывает реализацию базового класса. Во-вторых, базовая реализация инициализирует свойства локализации, основываясь на настройках директивы `Page` страницы ASP.NET. После этого мы проверяем в нашей переопределенной реализации, была ли добавлена специальная настройка языка в состояние сеанса. Если это так, мы назначаем настройку языка из состояния сеанса для свойств `Page.Culture` и `Page.UICulture`. Строка, содержащаяся в состоянии сеанса, должна быть действительной комбинацией кода языка и кода страны в формате `[код_языка]-[код_страны]`. Ранее в табл. 34.1 были перечислены примеры действительных комбинаций.

Специальные поставщики ресурсов

Локализация является той областью, в которой команда разработчиков ASP.NET проделала действительно серьезную работу с момента выхода версии ASP.NET 2.0, если проводить сравнение с версиями ASP.NET 1.*x*. Локализация хорошо интегрирована в среду разработки, а инфраструктура является чрезвычайно гибкой. Однако этого все равно недостаточно. Одним из наиболее часто задаваемых вопросов относительно локализации является следующий: “Можете ли вы хранить ресурсы в другом месте, например в базе данных?” Это особенно касается крупных команд разработчиков — для таких сценариев идеальным вариантом является центральное хранилище для локализованных ресурсов. Естественно, в большинстве крупных проектов локализацией зани-

маются специальные команды. В этом случае было бы здорово, если бы разработчики и дизайнеры могли создавать свои страницы как обычно, и применять Visual Studio 2005, как это было показано по ходу этой главы, для создания ресурсов локализации. Однако вместо того чтобы хранить и извлекать эти ресурсы (особенно строковые ресурсы, которые используются для перевода) из файлов `.resx`, их нужно брать из центральной базы данных.

К счастью, ASP.NET имеет чрезвычайно гибкую структуру. Буквально каждая часть инфраструктуры может быть подвергнута расширению (в этом вы сможете убедиться во время прочтения главы 18, когда мы будем рассматривать `VirtualPathProvider`). И это справедливо также для системы локализации ASP.NET. Как и многое другое в ASP.NET, инфраструктура ресурсов и локализации состоит из нескольких слоев, как показано на рис. 34.16.

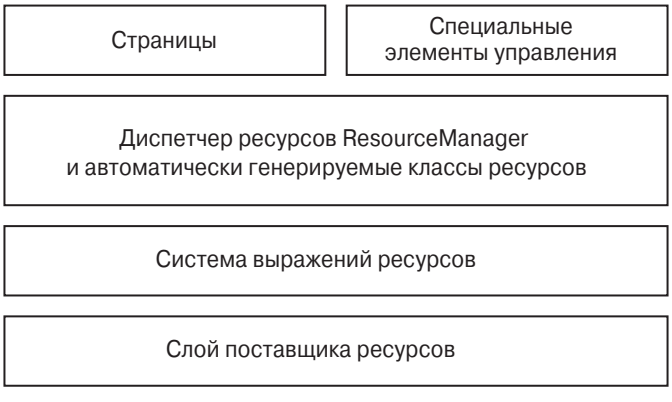

Рис. 34.16. Архитектура системы ресурсов и локализации

Вы можете расширить систему ресурсов и локализации на любом уровне данной многослойной модели (до настоящего момента вы видели лишь два верхних уровня). Самый нижний уровень — слой поставщика ресурсов — отвечает за чтение и запись ресурсов из выбранного вами источника. По умолчанию поставщик читает и записывает информацию в файлы `.resx`, которые или хранятся в файловой системе, или скомпилированы в качестве внедренных ресурсов в исполняемые файлы вашего приложения. Если вы хотите хранить ресурсы в другом месте либо предоставить комбинацию файлов `.resx` и выбранного вами хранилища, то вам придется использовать этот слой.

Реализация `ResourceProvider`

Как правило, в процессе настройки уровня поставщика ресурсов вам придется работать с двумя наборами компонентов. Существует группа компонентов, которые отвечают за работу всех частей исполняющей среды локализации, таких как извлечение и возврат ресурсов из хранилища; другая группа компонентов интегрируется с Visual Studio 2005 для автоматического генерирования ресурсов на основании содержимого веб-страниц в вашем решении. На рис. 34.17 показаны классы, включенные в процесс создания слоя поставщика специальных ресурсов.

Давайте рассмотрим слой исполняющей среды до того, как приступать к обсуждению интеграции времени проектирования. Настоящая работа появляется в реализации `ResourceProvider` — это класс, который реализует интерфейс `IResourceProvider` пространства имен `System.Web.Compilation`. Управление `ResourceProvider` осуществляется посредством реализации `ResourceProviderFactory`.

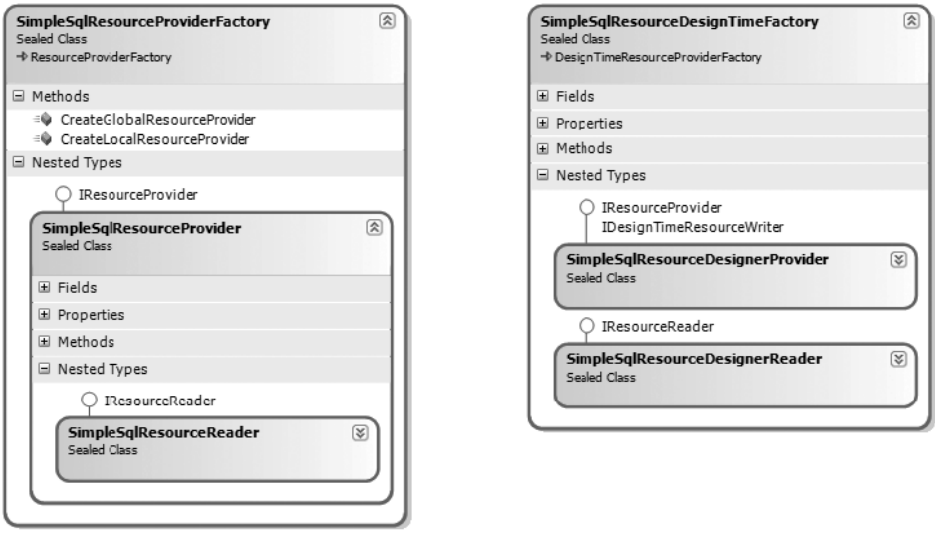

Рис. 34.17. Классы, необходимые для реализации слоя поставщика специальных ресурсов

Подробное рассмотрение классов `ResourceProviderFactory` будет предложено в разделе "Реализация `ResourceProviderFactory`" далее в этой главе.

```
private sealed class SimpleSqlResourceProvider : IResourceProvider
{
   object IResourceProvider.GetObject(string resourceKey, CultureInfo culture)
   {
      // ...
   }
   IResourceReader IResourceProvider.ResourceReader
   {
      get { /* ... */ }
   }
}
```

Обратите внимание на то, что данный класс отмечен идентификатором `private`, поскольку фабрика возвращает его как `ResourceProvider` — интерфейс, который этот класс реализует. Сам класс никогда не должен работать вне поставщика ресурсов, чтобы сохранить любые зависимые части конкретной реализации и позволить зависимостям существовать только на уровне интерфейса `IResourceProvider`.

ASP.NET создает экземпляр этого класса для каждой страницы (вернее, для каждого виртуального пути) вашего веб-приложения. Это означает, что один экземпляр этого класса отвечает за управление всеми ресурсами одной страницы. Интерфейс `IResourceProvider` требует, чтобы вы реализовали два метода. Первый из них, `GetObject`, ASP.NET вызывает каждый раз, когда производится запрос локализованного ресурса. Он передает уникальный ключ для ресурса, а также `CultureInfo` региона, для которого должен быть получен объект. Свойство `ResourceReader` возвращает экземпляр объекта `IResourceReader`, который позволяет ASP.NET перечислить все доступные ключи ресурсов для текущей страницы. Это означает, что при локализации страницы ASP.NET сначала получает набор доступных ресурсов для данной страницы через свойство `ResourceReader`. После этого она вызывает метод `GetObject` только для тех элементов управления, которые имеют атрибут `meta:resourcekey` и запись в ранее полученном экземпляре `IResourceReader`.

Внимание! Крайне важно, чтобы свойство `ResourceReader` вашей реализации `IResourceProvider` возвращало полный список доступных ключей ресурсов для страницы. Не пытайтесь выполнять фильтрацию ресурсов, загруженных в это свойство для текущего региона, или похожую операцию. К сожалению, Visual Studio 2005 ведет себя как-то странно, когда список ресурсов, возвращенный этим методом, оказывается неполным.

Как уже упоминалось, ASP.NET создает экземпляр реализации `IResourceProvider` для каждой страницы в вашем веб-приложении. Естественно, при хранении ресурсов в базе данных вам понадобятся строка подключения и некоторая разновидность вспомогательного класса для чтения и записи информации в базу данных. Для следующего примера предположим, что вы создали класс `SimpleSqlResourceDatabaseHelper` с помощью следующего интерфейса:

```
internal class SimpleSqlResourceDatabaseHelper
{
    public SimpleSqlResourceDatabaseHelper(string connString)
    {
        // Создаем команды и объекты подключения.
        // ...
    }
    // Ресурсы будут возвращать ListDictionary с локализованными
    // строками ресурсов. ListDictionary — это часть пространства имен
    // System.Collections.Specialized. Таким образом, вам необходимо
    // импортировать это пространство имен.
    public ListDictionary GetResources(string cultureName, string virtualPath)
    {
        // Выполняем команду SELECT и возвращаем результаты
    }
    public void AddResource(string virtualPath, string name,
                            string value, string cultureName)
    {
        // Выполняем команды UPDATE/INSERT.
    }
}
```

Это класс инкапсулирует весь набор классов ADO.NET и доступа к базе данных. Метод `AddResource` добавляет или обновляет ресурсы в базе данных, а метод `GetResource` читает все ресурсы для определенного виртуального пути и региона. Поскольку реализация этого класса является простой логикой базы данных и не представляет собой ничего особенного, мы пропустим ее и продолжим обсуждение реализаций поставщика ресурсов (более подробно об ADO.NET можно прочитать в главах 7 и 8). Теперь давайте посмотрим на реализацию `IResourceProvider` для нашего экземпляра.

```
private sealed class SimpleSqlResourceProvider : IResourceProvider
{
    private string _virtualPath;
    private SimpleSqlResourceDatabaseHelper _db = null;
    private ListDictionary _resCache = null;
    internal SimpleSqlResourceProvider(string virtualPath, string connString)
    {
        _virtualPath = System.IO.Path.GetFileName(virtualPath);
        _db = new SimpleSqlResourceDatabaseHelper(connString);
    }
    // ...
}
```

Как вы уже могли видеть ранее в этом разделе, интерфейс `IResourceProvider` не определяет ни одного метода, поддерживающего параметр для передачи виртуального пути веб-приложения, для которого создается поставщик. Поэтому ваша реализация `IResourceProvider` должна обеспечивать конструктор, в котором ваша собственная фабрика будет иметь возможность передавать путь виртуального каталога веб-приложения в реализацию `IResourceProvider`. Ваша реализация должна будет хранить виртуальный путь веб-приложения, переданный посредством конструктора в переменной экземпляра, чтобы иметь возможность находить подходящие ресурсы для веб-приложения в базовом хранилище. Реализацию этого принципа вы сможете увидеть во фрагменте кода, который будет представлен вслед за следующим фрагментом кода.

Таким образом, в конструкторе вашего класса `SimpleSqlResourceProvider` вам необходимо инициализировать виртуальный путь и создать экземпляр ранее упоминавшегося вспомогательного класса базы данных. Чтобы не усложнять это пример, вы просто работаете с именем страницы, а не с полным виртуальным путем локализируемого ресурса. Для кэширования ресурсов, загружаемых из базы данных, в своей реализации вы будете использовать член `_resCache`. Для этой цели вам необходимо реализовать некоторые вспомогательные методы. Во-первых, вам понадобится метод для извлечения ресурсов из хранилища и добавления их в словарь кэша в вашей реализации `IResourceProvider` (в нашем примере это `SimpleSqlResourceProvider`):

```
private ListDictionary GetResources(string cultureName)
{
    if (_resCache == null)
        _resCache = new ListDictionary();
    // Регион по умолчанию будет добавлен
    // с ключом string.empty в локальный кэш
    if (cultureName == null)
        cultureName = string.Empty;
    ListDictionary dict;
    if (!_resCache.Contains(cultureName))
    {
        dict = _db.GetResources(cultureName, _virtualPath);
        _resCache.Add(cultureName, dict);
    }
    else
    {
        dict = (ListDictionary)_resCache[cultureName];
    }
    return dict;
}
```

`GetResources` создает экземпляр `ListDistionary` для вашего кэша, если это еще не было сделано. Словарь `_resCache` будет содержать количество экземпляров `ListDictionary`. Это будут те словари ресурсов, которые содержат значения для локализации. Обычно корневой кэш (в данном случае это `_resCache`) будет содержать два словаря ресурсов — один для нейтрального региона, а другой для региона, выбранного в данный момент. Поэтому метод проверит, существует ли в кэше словарь для запрошенного региона. Если словарь существует, метод возвращает его из кэша. В противном случае он производит чтение ресурсов в базе данных с помощью `SimpleSqlResource DatabaseHelper`, созданного в конструкторе, а затем добавляет полученный словарь в локальный кэш, откуда и возвращает словарь. Теперь вы можете продолжить реализацию ранее представленного метода `GetObject`:

```
object IResourceProvider.GetObject(string resourceKey, CultureInfo culture)
{
    // Извлекаем регион, активный в данный момент времени.
    string cultureName = null;
    if (culture == null)
    {
        CultureInfo currentUICulture = CultureInfo.CurrentUICulture;
        if (currentUICulture != null)
        {
            cultureName = currentUICulture.Name;
        }
    }
    else
    {
        cultureName = culture.Name;
    }

    // Теперь получаем словарь для извлеченного региона.
    ListDictionary dict = GetResources(cultureName);

    // Возвращаем словарь, если он существует для запрошенного региона.
    // В противном случае вы просто получите словарь для нейтрального региона.
    if (dict.Contains(resourceKey))
        return dict[resourceKey];
    else
    {
        dict = GetResources(null);
        if (dict.Contains(resourceKey))
            return dict[resourceKey];
    }

    // Не было найдено ни одного словаря.
    return null;
}
```

Прежде чем метод `GetObject` получит запрошенный словарь ресурсов, он попытается выяснить, какой регион выбран в данный момент. Если регион был передан в метод, он получает словарь для переданного региона. В противном случае он использует свойство `CurrentUICulture` класса `CultureInfo`, чтобы получить регион, активный в данный момент времени (текущий регион для активного потока). Затем он получает словарь ресурсов для определенного региона с помощью представленного ранее метода `GetResources`. Этот словарь содержит значение для запрошенного ключа ресурсов. Если для запрошенного региона нет словаря ресурсов, он извлекает словарь для нейтрального региона и из него извлекает значение для запрошенного ресурса. Наконец, в вашей реализации `ResourceProvider` не достает одного свойства — `ResourceReader`:

```
IResourceReader IResourceProvider.ResourceReader
{
    get
    {
        return new SimpleSqlResourceReader(GetResources(null));
    }
}
```

Теперь вам осталось вернуть экземпляр `IResourceReader` в методе-получателе свойства на основании ресурсов по умолчанию. А для того чтобы вернуть `IResourceReader`, вам нужно его еще и реализовать. Как показано ниже, делается это довольно просто:

```
private sealed class SimpleSqlResourceReader : IResourceReader
{
   private IDictionary _resources;
   internal SimpleSqlResourceReader(IDictionary dict)
   {
      _resources = dict;
   }
   void IResourceReader.Close()
   {
      // Никаких действий...
   }
   IDictionaryEnumerator IResourceReader.GetEnumerator()
   {
      return _resources.GetEnumerator();
   }
   IEnumerator System.Collections.IEnumerable.GetEnumerator()
   {
      return _resources.GetEnumerator();
   }
   void IDisposable.Dispose()
   {
      // Никаких действий...
   }
}
```

Читатель ресурсов просто просматривает доступные ресурсы в словаре. Поскольку вы уже храните ресурсы в экземплярах `ListDictionary`, которые являются реализациями `IDictionary`, просмотр содержимого является ни чем иным, как возвращением нумератора в переданном словаре. Это можно сделать, просто вызвав метод `GetEnumerator()` словаря, как было показано в предыдущем фрагменте кода. Несмотря на то что это была трудная часть работы, вам предстоит сделать еще кое-что. До настоящего времени вы реализовали класс `ResourceProvider` со всеми необходимыми частями. А как ASP.NET будет активизировать `ResourceProvider`? Каким образом ASP.NET вообще будет знать о вашем классе `ResourceProvider`? Ответ на эти вопросы вы найдете, если внимательно изучите классы `ResourceProviderFactory`.

Реализация `ResourceProviderFactory`

`ResourceProviderFactory` отвечает за создание экземпляров реализаций `IResourceProvider`. Как таковая, фабрика — это точка входа в ваш специальный поставщик `ResourceProvider`. Фабрика — это просто класс, являющийся наследником `ResourceProviderFactory`, который состоит из двух методов, перечисленных ниже. Имейте в виду, что класс содержит ранее созданный класс `IResourceProvider` как вложенный класс (в нашем случае это `SimpleSqlResourceProvider`).

```
public sealed class SimpleSqlResourceProviderFactory
                          : ResourceProviderFactory
{
   public override IResourceProvider CreateLocalResourceProvider(
                                        string virtualPath)
   {
      virtualPath = System.IO.Path.GetFileName(virtualPath);
      return new SimpleSqlResourceProvider(virtualPath,
         ConfigurationManager.ConnectionStrings["ResDB"].ConnectionString);
   }
   public override IResourceProvider CreateGlobalResourceProvider(
                                                 string className)
```

```
    {
        // Не реализован в данном примере.
        // Реализация может выглядеть почти так же, как и для
        // локальных ресурсов. Единственное отличие заключается в том,
        // что данная реализация может использоваться для глобальных
        // ресурсов, поэтому она заменяет содержимое App_GlobalResources,
        // тогда как реализация локального ресурса заменяет
        // содержимое App_LocalResources.
        return null;
    }
    private sealed class SimpleSqlResourceProvider : IResourceProvider
    {
        // ...
        // Реализация IResourceProvider.
        // См. предыдущие фрагменты кода.
        // ...
    }
}
```

`ResourceProviderFactory` должен реализовать два метода: один для создания поставщика глобальных ресурсов, а другой для создания поставщика локальных ресурсов. Поставщики глобальных ресурсов используются для управления ресурсами для всех страниц и частей веб-приложения. Реализация, применяемая по умолчанию для глобальных ресурсов, хранит локализованные объекты в файлах `.resx` в папке `App_GlobalResources` приложения. Управление локальными ресурсами осуществляется отдельно для каждой страницы. Для данного примера вы будете реализовывать только локальные ресурсы (принципы реализации глобального поставщика `ResourceProvider` являются такими же, за исключением того, что для фильтрации этих ресурсов используется не виртуальный путь, а `classKey`, определяющий корневое имя ресурса).

ASP.NET вызывает метод `CreateLocalResourceProvider`, когда она первый раз начинает локализацию страницы (экземпляры `IResourceProvider` кэшируются в объекте `HttpApplication`). Таким образом, в реализации этого метода вам нужно лишь создать экземпляр ранее реализованного класса `SimpleSqlResourceProvider` и вернуть его. И, в конечном счете, вам нужно зарегистрировать `ResourceProviderFactory` в файле `web.config` приложения следующим образом (предполагается, что перед этим были созданы все классы в пространстве имен `Apress.Localization`):

```
<system.web>
    <globalization
        resourceProviderFactoryType=
            "Apress.Localization.SimpleSqlResourceProviderFactory" />
</system.web>
```

Теперь ASP.NET создает класс `SimpleSqlResourceProviderFactory`, когда первый раз будет запускать приложение (фабрика тоже кэшируется в `HttpApplication`). Каждый раз, когда страница будет впервые запускаться, и ее нужно будет локализовать, для нее будет вызываться фабрика, чтобы создать локальный `ResourceProvider` для этой страницы. После этого `ResourceProvider` будет применяться для получения ресурсов для всех элементов управления с атрибутом `meta:resourcekey`, связанным с ним.

Внимание! Прежде чем вы приступите к тестированию вашего поставщика, вы должны будете реализовать поддержку времени проектирования, о чем будет сказано в следующем разделе этой главы. Без этой поддержки вам придется вручную вводить ресурсы в вашу базу данных, иначе `IResourceProvider` не будет работать должным образом, так как он не сможет найти ресурсы.

Специальные и поставщики времени проектирования ResourceProviders

Сейчас приложение готово к получению ресурсов из базы данных. Но вам ведь нужно еще как-то добавить эти ресурсы в саму базу данных — и, естественно, необходимо иметь возможность делать это прямо в Visual Studio. Вашей целью должна быть поддержка разработчика из самой среды Visual Studio посредством хорошо известных способов добавления ресурсов в эту базу данных. К счастью, инфраструктура поставщика ресурсов из ASP.NET поддерживает вас и здесь. Вам нужно всего лишь создать `ResourceProvider` и `ResourceProviderFactory` для поддержки времени проектирования. Реализация этих классов является такой же, как и реализация поставщиков и фабрик для среды времени выполнения. Помимо реализации `ResourceProvider` и фабрики, необходимо реализовать `IDesignTimeResourceWriter` в вашей реализации `ResourceProvider`. Реализация `IDesignTimeResourceWriter` отвечает за запись ресурсов обратно в базу данных (или в хранилище данных, в зависимости от того, что вы выберете). Об этом новом интерфейсе вы узнаете позже в этом разделе, когда мы будем говорить о реализации `IResourceProvider` для поддержки времени проектирования. А пока что давайте разберемся с фабричным классом поставщика для поддержки времени проектирования. В основном, он выглядит следующим образом:

```
public sealed class SimpleSqlResourceDesignTimeFactory
                : System.Web.UI.Design.DesignTimeResourceProviderFactory
{
   private IServiceProvider _serviceProvider = null;
   private SimpleSqlResourceDesignerProvider _localProvider = null;
   private SimpleSqlResourceDesignerProvider LocalProvider
   {
      get
      {
         if (_localProvider == null) {
            _localProvider = new
                SimpleSqlResourceDesignerProvider(_serviceProvider);
      }
      return _localProvider;
      }
   }
   public override IResourceProvider CreateDesignTimeGlobalResourceProvider(
                    IServiceProvider serviceProvider, string classKey)
   {
      return null;
   }
   public override IResourceProvider CreateDesignTimeLocalResourceProvider(
                                  IServiceProvider serviceProvider)
   {
      _serviceProvider = serviceProvider;
      return LocalProvider;
   }
   public override IDesignTimeResourceWriter
      CreateDesignTimeLocalResourceWriter(IServiceProvider serviceProvider)
   {
      _serviceProvider = serviceProvider;
      return LocalProvider;
   }
   private sealed class SimpleSqlResourceDesignerProvider
                : IResourceProvider, IDesignTimeResourceWriter
   {
```

```
        // ...
        // Реализация локального поставщика и писателя ресурсов.
    }
}
```

`DesignTimeResourceProviderFactory` требует поддержки большего количества фабричных методов, чем `ResourceProviderFactory` для среды времени выполнения. Помимо фабричных методов для создания поставщиков глобальных и локальных ресурсов ему нужна поддержка фабричного метода для создания `ResourceWriter` для локальных ресурсов. `ResourceWriter` отвечает за добавление ресурсов в базу данных, а также за генерирование уникальных ключей для этих ресурсов. В любом случае эти фабричные методы получают ссылку на среду разработки в качестве параметра. Эта ссылка на среду разработки устанавливается посредством экземпляров `IServiceProvider`. Вы можете представлять себе `IServiceProvider` как интерфейс обратной связи с Visual Studio. Если вам нужно прочитать параметры конфигурации из вашего файла `web.config` или содержимое из страниц, параметры проекта и т.п., вы можете сделать это посредством методов и свойств передаваемого параметра `serviceProvider`.

Реализация поставщика времени проектирования

Реализация `ResourceProvider` очень похожа на реализацию версии времени выполнения. В предыдущем примере класс, реализующий `IResourceProvider`, реализует еще один интерфейс — `IDesignTimeResourceWriter`. Этот интерфейс требует от вас реализовать методы для генерации ключей для новых ресурсов, добавляя новые ресурсы в ваш словарь ресурсов и передавая эти ресурсы обратно в базу данных.

```
private sealed class SimpleSqlResourceDesignerProvider
                  : IResourceProvider, IDesignTimeResourceWriter
{
    public SimpleSqlResourceDesignerProvider(IServiceProvider provider)
    {
        _provider = provider;
        _db = new SimpleSqlResourceDatabaseHelper(...);
    }
    object IResourceProvider.GetObject(string resourceKey, CultureInfo culture)
    {
        // Извлекаем словарь ресурсов как в GetObject
        // из представленного ранее класса SimpleSqlResourceProvider.
    }
    IResourceReader IResourceProvider.ResourceReader
    {
        get
        {
            // Получаем словарь ресурсов, как в классе
            // SimpleSqlResourceProvider, представленном ранее.
        }
    }
    string IDesignTimeResourceWriter.CreateResourceKey(
                        string resourceName, object obj)
    {
        // Генерируем новый ключ для ресурса. В качестве ключа может
        // быть произвольная строка, но она должна быть уникальной.
    }
    void IResourceWriter.AddResource(string name, byte[] value)
    {
        // Добавляем в словарь ресурс массива byte[].
    }
```

```
    void IResourceWriter.AddResource(string name, object value)
    {
        // Добавляем объект (сериализуемый) в словарь.
    }
    void IResourceWriter.AddResource(string name, string value)
    {
        // Добавляем строковый ресурс в словарь.
    }
    void IResourceWriter.Close()
    {
        // Если необходимо, закрываем соединения.
    }
    void IResourceWriter.Generate()
    {
        // Просматриваем словарь ресурсов и
        // сохраняем каждую запись в базе данных.
    }
    void IDisposable.Dispose()
    {
        // Закрываем все, что использовалось.
    }
}
```

Во время разработки вам нужно просто сохранить ссылку на `IServiceProvider`, переданную конструктору, и создать экземпляр вспомогательного класса базы данных, который использовался ранее в этом примере. Интересной частью является то, что при создании экземпляра вашего класса базы данных вам нужно соответствующим образом инициализировать связующие строки. Но в связи с тем, что с этим классом вы работаете в режиме проектирования, вы не можете просто получить доступ к конфигурационному файлу, как это делаете обычно. Вам нужно пройти через `IServiceProvider`, как показано ниже:

```
private IServiceProvider _provider;
private SimpleSqlResourceDatabaseHelper _db;
private ListDictionary _resCache = null;
public SimpleSqlResourceDesignerProvider(IServiceProvider provider)
{
    _provider = provider;
    _db = new SimpleSqlResourceDatabaseHelper(
              CreateConnectionString(provider));
}
private const string DatabaseLocationKey = "ResDB";
private string CreateConnectionString(IServiceProvider serviceProvider)
{
    if (serviceProvider == null)
    {
        return System.Configuration.ConfigurationManager.AppSettings[
                                                  DatabaseLocationKey];
    }
    else
    {
        IWebApplication webApp =
            (IWebApplication)serviceProvider.GetService(
                              typeof(IWebApplication));
        return webApp.OpenWebConfiguration(
                true).ConnectionStrings.ConnectionStrings
                        [DatabaseLocationKey].ConnectionString;
    }
}
```

Поскольку вы знаете, что вы занимаетесь разработкой веб-приложения, то, работая с проектом веб-сайта, вам необходимо запросить у `IServiceProvider` службу `IWebApplication`, которая размещена в IDE-среде Visual Studio 2005. Эта служба обеспечивает доступ ко всему, что имеет отношение к проектам веб-приложения в Visual Studio (например, свойства проекта, файлы в проекте и открытые в данный момент дизайнеры), поэтому она позволяет вам открывать веб-конфигурацию для получения связующих строк с базой данных посредством `OpenWebConfiguration`.

Как видите, ваш класс `SimpleSqlResourceDesignerProvider` тоже хранит словарь `ListDictionary`, называемый `_resCache`, который похож на ту реализацию, которую вы недавно создали. Это словарь переносит в кэш все отличные словари ресурсов для разных регионов запрошенной страницы. Реализации `GetObject` и `ResourceReader` подобны реализациям, которые вы уже видели в классе `SimpleSqlResourceProvider`, за исключением того, что они постоянно работают с регионом, выбираемым по умолчанию.

```
object IResourceProvider.GetObject(string resourceKey,
                                   CultureInfo culture)
{
    // Всегда используем регион по умолчанию.
    string cultureName = string.Empty;
    // Получаем объект ресурса.
    ListDictionary dict = GetResources(cultureName);
    if (dict.Contains(resourceKey))
        return dict[resourceKey];
    else
    {
        dict = GetResources(null);
        if (dict.Contains(resourceKey))
            return dict[resourceKey];
    }
    return null;
}
IResourceReader IResourceProvider.ResourceReader
{
    get
    {
        return new SimpleSqlResourceDesignerReader(GetResources(null));
    }
}
```

Однако если думать о получении ресурсов для страницы, в конечном счете, вам нужно будет знать о виртуальном пути, в котором вы работаете. Не забывайте о реализации метода `GetResources`, которая ранее использовалась методом `GetObject` и свойством `ResourceReader`. В вашей предыдущей реализации для среды времени выполнения (`SimpleSqlResourceProvider`) вы использовали переменную-член `virtualPath` класса, чтобы получать ресурсы для активной в текущий момент времени страницы. Это похоже на реализацию поставщика времени проектирования, в чем вы сможете убедиться, посмотрев на следующий фрагмент кода. Реализация `IResourceReader`, используемая в свойстве `ResourceReader`, является такой же, как и реализация `SimpleSqlResourceReader`, представленная ранее.

```
private ListDictionary GetResources(string cultureName)
{
    if (_resCache == null)
        _resCache = new ListDictionary();
    if (cultureName == null)
        cultureName = string.Empty;
    ListDictionary dict;
```

```
    if (!_resCache.Contains(cultureName))
    {
        dict = _db.GetResources(cultureName,
                    GetVirtualPath(_provider));
        _resCache.Add(cultureName, dict);
    }
    else
    {
        dict = (ListDictionary)_resCache[cultureName];
    }
    return dict;
}
```

Чтобы извлечь необходимые ресурсы для текущей страницы из базы данных, вам нужен виртуальный путь (или, по крайней мере, имя страницы в этом примере). В среде времени проектирования, для получения имени активной в данный момент времени страницы вам понадобится другой подход. Это можно продемонстрировать на примере реализации метода `GetVirtualPath`, который уже использовался ранее.

```
private string GetVirtualPath(IServiceProvider provider)
{
    System.ComponentModel.Design.IDesignerHost host =
        (IDesignerHost)provider.GetService(typeof(IDesignerHost));
    WebFormsRootDesigner rootDesigner =
        host.GetDesigner(host.RootComponent) as WebFormsRootDesigner;
    return System.IO.Path.GetFileName(rootDesigner.DocumentUrl);
}
```

И снова `IServiceProvider` является ответом на эту загадку. Он предоставляет доступ к службе дизайнера (она называется `IDesignerHost`). С помощью этой службы вы можете иметь доступ к активному в данный момент времени дизайнеру, открытому в Visual Studio. Когда нужно будет локализовать страницу, это всегда будет дизайнер веб-страницы. Посредством дизайнера веб-страницы вы получаете доступ к полному имени файла, который открыт в данный момент времени в дизайнере. Естественно, это позволяет вам извлекать имя открытой в данный момент времени страницы, чтобы получить подходящие ресурсы из базы данных и записать их обратно в нее.

Оставшаяся реализация очень проста. Вам нужно добавить ресурсы в ваш кэшированный словарь (переменная-член `_resCache` в вашем классе) и записать кэшированные ресурсы обратно в базу данных. Это делается следующим образом.

```
string IDesignTimeResourceWriter.CreateResourceKey(
                    string resourceName, object obj)
{
    // Теперь добавляем словарь для указанного объекта.
    // Кроме этого, мы генерируем ключ, который еще не используется.
    int counter = 1;
    string objTypeName = obj.GetType().Name;
    string keyBaseName = objTypeName + "Resource" + resourceName;
    counter = GetNextKeyIndex(keyBaseName, counter);
    return string.Format("{0}{1}", keyBaseName, counter);
}
void IResourceWriter.AddResource(string name, byte[] value)
{
    if (Resources.Contains(name))
        Resources[name] = value;
    else
        Resources.Add(name, value);
}
```

```
void IResourceWriter.AddResource(string name, object value)
{
   if (Resources.Contains(name))
      Resources[name] = value;
   else
      Resources.Add(name, value);
}
void IResourceWriter.AddResource(string name, string value)
{
   if (Resources.Contains(name))
      Resources[name] = value;
   else
      Resources.Add(name, value);
}
void IResourceWriter.Close()
{
   // Если необходимо, закрываем соединения.
}
void IResourceWriter.Generate()
{
   // Получаем текущий виртуальный путь.
   string vPath = GetVirtualPath(_provider);
   // Производим запись обратно в базу данных.
   foreach (object k in Resources.Keys)
   {
      // Для простоты, мы имеем дело просто со строками
      // в нашем простом примере реализации.
      _db.AddResource(vPath, k.ToString(),
            Resources[k].ToString(), null);
   }
}
```

Методы, которые вы видели в предыдущем коде, Visual Studio вызывает главным образом тогда, когда вы выбираете команду Generate Local Resource (Генерировать локальный ресурс) в меню Tools (Сервис). Для каждого локализуемого свойства каждого элемента управления на странице, Visual Studio сначала вызывает метод `CreateResourceKey` для генерирования уникального ключа, используемого в атрибуте `meta:resourcekey`, а также в базе данных для идентификации ресурса. Впоследствии он вызывает один из методов `AddResource` (в зависимости от типа свойства), передавая ранее сгенерированный ключ для добавления ресурса в словарь ресурсов. В заключение, когда среда Visual Studio произведет перебор всех элементов управления и локализованных свойств, она вызывает метод `Generate` писателя ресурсов для записи элементов ресурса обратно в базу данных. Предыдущая реализация снова использует ранее представленный вспомогательный класс `SimpleSqlResourceDatabaseHelper`, чтобы добавить записи в базу данных. Доступ к словарю ресурсов осуществляется через свойство `Resources` класса — это сделано ради удобства. Генерирование ключа происходит в методе `GetNextKeyIndex`, который вызывается посредством `CreateResourceKey`. Недостающие части показаны в следующем фрагменте кода.

```
private int GetNextKeyIndex(string key, int counter)
{
   // Переходим к существующим элементам и проверяем, используется ли ключ.
   foreach (string k in Resources.Keys)
   {
      if (k.IndexOf(string.Format("{0}{1}", key, counter)) >= 0)
      {
         counter = GetNextKeyIndex(key, counter + 1);
```

```
        }
    }
    return counter;
}
private ListDictionary _Resources = null;
private ListDictionary Resources
{
    get
    {
        if (_Resources == null)
        {
            _Resources = _db.GetResources("none",
                      GetVirtualPath(_provider));
            if (_Resources == null)
                _Resources = new ListDictionary();
        }
        return _Resources;
    }
}
```

Метод `GetNextKeyIndex` просто перебирает все записи ресурсов в словаре, чтобы проверить, используется ли уже выбранный ключ. Если он используется, метод увеличивает значение счетчика на единицу, чтобы модифицировать суффикс имени для ключа, а затем снова пересматривает список существующих ресурсов, чтобы проверить, используется ли он. Свойство `Resources` просто применяется для удобства доступа к ресурсам нейтрального региона выбранной в данный момент времени страницы. Оно проверяет, были ли уже созданы эти ресурсы. Если это не подтверждается, оно извлекает ресурсы из текущего виртуального пути методом `GetRecources`, который вы уже видели в этой главе.

Регистрация поставщика времени проектирования

В завершение вам необходим способ, посредством которого вы сможете сообщить Visual Studio о новом поставщике времени проектирования, который вы только что создали. На этот раз, мы не будем создавать запись в файле `web.config`. Как правило, поставщик ресурсов времени проектирования тесно связан с поставщиком времени выполнения. Таким образом, мост между этими двумя мирами построен на атрибуте, применяемом к `ResourceProviderFactory`, который использовался во время выполнения. Этот атрибут сообщает Visual Studio о `DesignTimeResourceProviderFactory`.

```
[
    DesignTimeResourceProviderFactoryAttribute
    (
        typeof(SimpleSqlResourceDesignTimeFactory)
    )
]
public sealed class SimpleSqlResourceProviderFactory : ResourceProviderFactory
{
    // ...
}
```

Благодаря этому атрибуту и регистрации `ResourceProviderFactory` в файле `web.config`, Visual Studio знает, что существует `ResourceProviderFactory` для времени проектирования. Таким образом, она использует `ResourceDesignTimeFactory` для создания классов поставщика ресурсов и писателя ресурсов, реализованных в предыдущих двух разделах для чтения ресурсов, добавления новых ресурсов и записи этих ресурсов обратно в хранилище. Теперь, когда у вас имеются поставщики времени выполнения

и времени проектирования, вы полностью подготовили почву для работы с ресурсами. Разработчики могут создавать свои страницы привычным способом и выбирать команду Generate Local Resources в меню Tools. Таким способом будут создаваться новые записи ресурсов в центральной базе данных, чтобы остальные разработчики, работающие в команде, могли иметь доступ к этим записям. Более того, отдельная команда разработчиков, которая отвечает за перевод, сможет иметь доступ к этой базе данных (например, с помощью простого Web-интерфейса) для перевода и обновления ресурсов для всех регионов. Само собой, каждый член команды будет автоматически получать любые изменения переведенных ресурсов, так как центральная база данных может использоваться со специальными поставщиками ресурсов.

На заметку! Хранить ресурсы в центральной базе данных интересно и полезно. Однако для повышения производительности мы рекомендуем хранить любые ресурсы в сопроводительных скомпилированных сборках для решения, поскольку доступ к внедренным ресурсам осуществляется гораздо быстрее, чем к базе данных. Неплохо использовать базу данных во время разработки и иметь готовые специальные шаги компоновки. Во время выполнения этих шагов будет производиться чтение ресурсов из базы данных, чтобы их можно было компилировать в виде внедренных ресурсов в сборки для окончательной компоновки.

Резюме

В этой главе рассматривались основы создания интернациональных Web-приложений с помощью ASP.NET. Во-первых, вы узнали, как платформа .NET управляет ресурсами для приложений, и как можно обращаться к этим ресурсам программно. Более того, оказалось, что эти ресурсы полезны не только для локализации, но и для других целей. Например, они могут быть полезны для внедренных шаблонов по умолчанию, используемых для отчетов (как запасной вариант на тот случай, если в других каталогах не окажется шаблонов), для дополнительных сценариев установки или для XML-фрагментов, используемых в вашем приложении.

Во-вторых, вы узнали, как CLR управляет ресурсами, являющимися специфическими для данного региона. CLR выбирает внедренные ресурсы на основании настройки `CultureInfo` в свойстве `Thread.CurrentUICulture`. CLR включает механизм обхода (часто называемый “hub and spoke”) для выявления ресурсов: поиск производится от наиболее совпадающих до более общих ресурсов региона в определенной иерархии регионов. Если нужный регион не будет найден, то для локализации будет использоваться регион, установленный в приложении по умолчанию.

В-третьих, вы узнали о том, каким образом ASP.NET и Visual Studio обеспечивают поддержку в локализации Web-приложений с помощью локальных ресурсов страниц и разделяемых ресурсов приложений. Вы могли узнать о том, как осуществляется доступ к этим ресурсам программным и декларативным образом через неявные и явные выражения локализации. Вы узнали также о способе динамического переключения локалей во время выполнения, что позволяет вам предоставить пользователю возможность выбрать свой язык.

Наконец, было показано, как расширяемая инфраструктура ASP.NET, доступная с момента выхода версии 2.0, позволяет создавать специальные поставщики ресурсов, включая настройку поставщиков времени проектирования.