Настройка маршрутизации

Исходный код проекта

По умолчанию URL, используемые для доступа к веб-формам в приложении ASP.NET Framework соответствуют именам файлов. Таким образом, запрос следующего URL:

```
http://localhost:9021/Pages/Listing.aspx
```

приведет к обработке файла Listing.aspx из папки Pages и возвращению результирующей HTML-разметки клиентскому браузеру. Если вы переместили файл Listing.aspx в папку по имени Store, то для получения того же самого контента придется запрашивать такой URL:

```
http://localhost:9021/Store/Listing.aspx
```

Преимущество этой модели связано с простотой и легкостью ее понимания. Однако она не всегда удобна — изменение местоположения файла веб-формы означает обновления поиска и обновления всех файлов, которые ссылались на этот файл веб-формы, что представляет собой утомительный и чреватый ошибками процесс. Например, ранее мы ссылались на местоположение файла Listing.aspx при создании страничных ссылок:

```
<div class="pager">
        <%
            for (int i = 1; i <= MaxPage; i++)
            {
                Response.Write(
                    String.Format("<a href='/Pages/Listing.aspx?page={0}' {1}>
                        i, i == CurrentPage ? "class='selected'" : "", i));
            }
        %>
</div>
```

Наш проект довольно прост, но в сложном приложении ссылок подобного рода может оказаться очень много. Нахождение их и аккуратное обновление занимает время, к тому же требует тщательного тестирования, подтверждающего, что все ссылки были найдены. Обычного поиска и замены будет недостаточно, т.к. обычно обнаруживается, что пути к страницам генерируются разнообразными способами, которые редко удается найти настолько просто, как показанные выше. (Типичный подход предусматривает жесткое кодирование папки и расширения файла со вставкой имени файла из переменной.)

Отображение URL на пути также создает проблемы у пользователей, которые часто создают закладки для определенных URL. После переименования или перемещения файла, URL, сохраненный в закладках у пользователя, перестанет работать.

Нам необходимо что-то более надежное и гибкое — и таковым является решение с **маршрутизацией URL**. Маршрутизация позволяет создать абстракцию между URL, которые приложение поддерживает, и файлами веб-форм, к которым эти URL относятся. В этой статье мы продемонстрируем его базовую конфигурацию, чтобы вы могли видеть, как оно работает.

Создание класса конфигурации маршрутизации

Конфигурация маршрутизации должна быть установлена при запуске приложения ASP.NET Framework, чтобы URL, которые планируется поддерживать, были определены еще до поступления первого клиентского запроса. Соглашение о конфигурации запуска предполагает создание в папке App_Start класса, содержащего метод настройки, и затем вызов этого метода в глобальном классе приложения Global.asax.

Соглашение об именовании для файлов классов в папке App_Start выглядит как <средство>Config.cs, поэтому мы должны создать новый файл класса по имени RouteConfig.cs с содержимым, представленным в примере ниже:

```
using System;
using System.Web.Routing;

// namespace GameStore.App_Start
namespace GameStore
{
    public class RouteConfig
    {
        public static void RegisterRoutes(RouteCollection routes)
        {
            routes.MapPageRoute(null, "", "~/Pages/Listing.aspx");
            routes.MapPageRoute(null, "list", "~/Pages/Listing.aspx");
        }
    }
}
```

Обратите внимание, что пространство имен, добавленное Visual Studio в файл RouteConfig.cs, закомментировано. По умолчанию класс RouteConfig будет помещен в пространство имен GameStore.App_Start — в этом нет ничего плохого, но мы предпочитаем держать классы конфигурации в пространстве имен GameStore. Мы строго придерживаемся таких предпочтений при кодировании, хотя и не можем сформулировать разумные причины своего поведения.

Класс RouteConfig применяется для определения новой схемы URL для приложения GameStore. Параметр routes, передаваемый методу RegisterRoutes(), представляет собой объект **RouteCollection**. Его метод MapPageRoute() используется для создания маршрутов. Маршрут сообщает среде ASP.NET Framework, каким образом обрабатывать URL, который не соответствует файлу веб-формы .aspx на диске.

Существует немало способов настройки конфигурации маршрутизации для приложения ASP.NET Framework, но многие из них требуют подробных объяснений. Здесь мы создали конфигурацию маршрутизации, опирающуюся на базовые возможности. Показанные выше два оператора в методе RegisterRoutes() создают новую схему URL, представляющую собой набор URL, которые могут применяться для указания на файлы веб-форм в приложении.

Если приложение выполняется на порту 9021 машины localhost, например, то приведенные выше операторы добавляют поддержку следующих URL:

```
http://localhost:9021/
http://localhost:9021/list
```

Оба эти URL приводят к тому, что для обработки запроса будет использоваться файл веб-формы ~\Pages\Listing.aspx.

По-прежнему необходима возможность перехода на определенные страницы

информации о товарах; для этого не рекомендуется использовать строки запросов URL, которые мало того, что неуклюжи, так еще и вышли из моды. Желательно поддерживать URL, включающие в себя страницу товаров, например:

```
http://localhost:9021/list/2
```

Такая разновидность URL относится к дешифруемому или композиционному стилю. Нам нравится применять этот тип URL, когда только возможно, поскольку при его непосредственном редактировании он выглядит яснее и проще для пользователя, чем строки запросов. (Вас может удивить, насколько много пользователей веб-приложений предпочитают вводить URL вручную.) Чтобы получить то, что нужно, необходимо добавить в метод RegisterRoutes() новый оператор, как показано в примере ниже:

```
using System;
using System.Web.Routing;

// namespace GameStore.App_Start
namespace GameStore
{
    public class RouteConfig
    {
        public static void RegisterRoutes(RouteCollection routes)
        {
            routes.MapPageRoute(null, "list/{page}", "~/Pages/Listing.aspx");
            routes.MapPageRoute(null, "", "~/Pages/Listing.aspx");
            routes.MapPageRoute(null, "list", "~/Pages/Listing.aspx");
        }
    }
}
```

Конструкция {page} называется *переменной сегмента маршрутизации*. Она позволяет захватить часть URL и пользоваться ею при обработке запроса. Когда ASP.NET Framework получает URL следующего вида:

```
http://localhost:9021/list/2
```

запрос обрабатывается с применением страницы Listing.aspx, а вдобавок создается переменная по имени page, которой присваивается значение 2. Вскоре мы покажем, как получать доступ к значениям переменных маршрутизации.

Порядок определения маршрутов играет важную роль, поэтому мы добавили

новый маршрут в качестве первого оператора метода RegisterRoutes(). Мы полагаемся на устанавливаемый здесь порядок маршрутов, когда дело доходит до генерации новых страничных ссылок позже.

Обновление глобального класса приложения

При запуске приложения необходимо вызвать метод RouteConfig.RegisterRoutes(), что требует применения глобального класса приложения Global.asax. В примере ниже показано обновленное содержимое файла Global.asax.cs, в котором вызывается метод, конфигурирующий маршрутизацию, и удалены методы, ненужные в данный момент:

```
using System;
using System.Web.Routing;

namespace GameStore
{
    public class Global : System.Web.HttpApplication
    {

        protected void Application_Start(object sender, EventArgs e)
        {
            RouteConfig.RegisterRoutes(RouteTable.Routes);
        }
    }
}
```

В классе System.Web.RouteTable определено статическое свойство Routes, предоставляющее объект RouteCollection, который необходим для конфигурирования, проводимого в методе Application_Start() глобального класса приложения.

Использование переменных маршрутизации

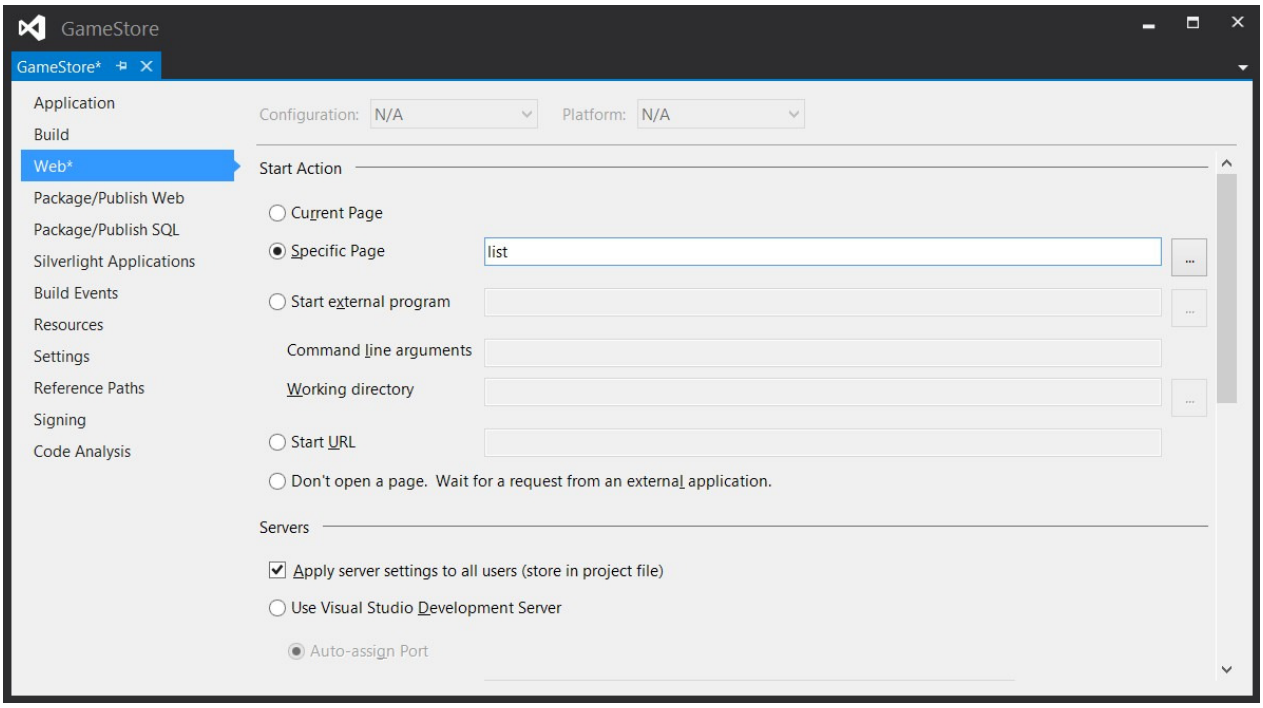
Мы обновили содержимое файла отделенного кода \Pages\Listing.aspx.cs, реализовав проверку переменных маршрутизации с целью выяснения, захвачено ли значение page:

```
using System;
using System.Collections.Generic;
using GameStore.Models;
using GameStore.Models.Repository;
using System.Linq;

namespace GameStore.Pages
{
    public partial class Listing : System.Web.UI.Page
    {
        private Repository repository = new Repository();
        private int pageSize = 4;

        protected int CurrentPage
        {
            get {
                int page;
                page = GetPageFromRequest();
                return page > MaxPage ? MaxPage : page;
            }
        }

        protected int MaxPage
        {
            get {
                return (int)Math.Ceiling((decimal)repository.Games.Count() / p
            }
        }

        private int GetPageFromRequest()
        {
            int page;
            string reqValue = (string)RouteData.Values["page"] ??
                Request.QueryString["page"];
            return reqValue != null && int.TryParse(reqValue, out page) ? page
        }

        protected IEnumerable<Game> GetGames()
        {
            return repository.Games
                .OrderBy(g => g.GameId)
                .Skip((CurrentPage - 1) * pageSize)
```

Значения переменных маршрутизации извлекаются через коллекцию `RouteData.Values` в вспомогательном методе `GetPageFromRequest()`. В этом примере производится попытка получить значение переменной `page`. Поскольку нет никакой гарантии, что из `URL` будет извлечено значение для переменной маршрутизации, необходимо аккуратно обрабатывать значения `null`. Наша старая схема `URL` по-прежнему работает, так что при отсутствии доступной переменной маршрутизации мы проверяем свойства `Request.QueryString`.

Тестирование конфигурации маршрутизации

Выберите пункт `GameStore Properties` (Свойства `GameStore`) в меню `Project` (Проект) среды `Visual Studio` и щелкните на вкладке `Web` (Веб). Именно здесь `Visual Studio` фиксирует страницу, на которую браузер переходит при запуске приложения. Ваш выбор был сохранен как `\Pages\Listing.aspx`, когда вы указывали его в качестве стартовой страницы. Удостоверьтесь, что выбран переключатель `Specific Page` (Указанная страница) и введите в поле справа значение `list`, как показано на рисунке ниже:



Запустите приложение, выбрав пункт Start Debugging (Начать отладку) в меню Debug (Отладка). Браузер перейдет на указанный вами URL вида /list. Для навигации на определенную страницу товаров к URL понадобится добавить /2 или /3. (Однако не пользуйтесь пока страничными ссылками, т.к. они по-прежнему работают со старой схемой URL; мы очень скоро это исправим.)

Генерирование маршрутизируемых ссылок

Старая схема URL все еще работает, и это хорошо, т.к. мы используем старые URL в страничных ссылках. Чтобы полностью принять URL, определенные в конфигурации маршрутизации, понадобится изменить способ создания страничных ссылок в файле \Pages\Listing.aspx:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Listing.aspx.cs" Inh
    MasterPageFile="~/Pages/Store.Master" %>
<%@ Import Namespace="System.Web.Routing" %>

<asp:Content ContentPlaceHolderID="bodyContent" runat="server">
    <div id="content">
        <%
            foreach (GameStore.Models.Game game in GetGames())
            {
                Response.Write(String.Format(@"
                        <div class='item'>
                            <h3>{0}</h3>
                            {1}
                            <h4>{2:c}</h4>
                        </div>",
                    game.Name, game.Description, game.Price));
            }
        %>
    </div>
    <div class="pager">
        <%
            for (int i = 1; i <= MaxPage; i++)
            {
                string path = RouteTable.Routes.GetVirtualPath(null, null,
                    new RouteValueDictionary() { { "page", i } }).VirtualPath;
                Response.Write(
                    String.Format("<a href='{0}' {1}>{2}</a>",
                        path, i == CurrentPage ? "class='selected'" : "", i));
            }
        %>
    </div>
</asp:Content>
```

Если мы просто жестко закодируем новую схему URL в файлах веб-форм, то не получим никаких реальных преимуществ — сами URL улучшатся, но все равно придется отслеживать и изменять любые ссылки, когда понадобится модифицировать схему маршрутизации.

Вместо этого мы генерируем необходимые URL с использованием метода RouteTable.Routes.GetVirtualPath(). Код, который должен быть сгенерирован для страничных ссылок, несколько громоздок, т.к. в конфигурации маршрутизации приложения может присутствовать немало сложностей. В данный момент достаточно знать, что переделанный фрагмент кода генерирует страничные ссылки в форме http://localhost:9021/list/2.

Обычно оператор, генерирующий ссылку из маршрута, будет помещаться в класс отделенного кода — он оставлен внутри фрагмента кода, только чтобы упростить пример. Существуют другие способы генерации ссылок с использованием конфигурации маршрутизации, часть из которых больше подходят для применения во фрагментах кода.