# URL和Ajax辅助器方法

## 创建基本的链接和URL

在我们介绍链接或URL之前先做一些准备，我们这部分要介绍的知识将要使用的项目就是之前建立的HelperMethods项目，现在需要先为其添加一个Person控制器，并在其中定义一个Person模型对象的结合，具体如下：

using HelperMethods.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace HelperMethods.Controllers

{

public class PersonController : Controller

{

private Person[] personData =

{

new Person{FirstName = "Adam",LastName = "Freeman",Role = Role.Admin},

new Person{FirstName = "Steven",LastName = "Sanderson",Role = Role.Admin},

new Person{FirstName = "Jacqui",LastName = "Griffyth",Role = Role.User},

new Person{FirstName = "John",LastName = "Smith",Role = Role.User},

new Person{FirstName = "Anne",LastName = "Jones",Role = Role.Guest}

};

public ActionResult Index()

{

return View();

}

public ActionResult GetPeople()

{

return View(personData);

}

public ActionResult GetPeople(string selectedRole)

{

if (selectedRole == null || selectedRole == "All")

{

return View(personData);

}

else

{

Role selected = (Role)Enum.Parse(typeof(Role), selectedRole);

return View(personData.Where(p => p.Role == selected));

}

}

}

}

将会使用的CSS如下（添加在/Content/Site.css，在/Views/Shared/\_Layout.cshtml文件中有一个引用此文件的link元素）：

…

table, td, th {

border: thin solid black;

border-collapse: collapse;

padding: 5px;

background-color: lemonchiffon;

text-align: left;

margin: 10px 0;

}

div.load {

color: red;

margin: 10px 0;

font-weight: bold;

}

div.ajaxLink {

margin-top: 10px;

margin-right: 5px;

float: left;

}

…

创建链接或URL是视图的常见功能，它能够帮用户导航至程序的其他部分。下表总结了一些常见的HTML辅助器，大家需要知道使用这些辅助器方法生成链接和URL有个好处就是：输出来自路由配置，这意味着路由的改变会自动反映在链接和URL中。

|  |  |
| --- | --- |
| 描述 | 示例 |
| 相对于程序的URL | Url.Content ("∼/Content/Site.css")  输出：  /Content/Site.css |
| 链接到指定的动作/控制器  （*输出链接*） | Html.ActionLink("My Link", "Index", "Home")  输出：  <a href "/">My Link</a> |
| 动作URL  （*输出URL*） | Url.Action ("GetPeople", "People")  输出：  /People/GetPeople |
| 使用路由数据的URL | Url.RouteUrl(new {controller = "People", action = "GetPeople"})  输出：  /People/GetPeople |
| 使用路由数据的链接 | Html.RouteLink("My Link", new {controller = "People", action = "GetPeople"})  输出：  <a href "/People/GetPeople">My Link</a> |
| 链接到指定路由 | Html.RouteLink("My Link", "FormRoute", new {controller = "People", action = "GetPeople"})  输出：  <a href "/app/forms/People/GetPeople">My Link</a> |

后面我们创建一个视图来观察一下上述表格中的辅助器方法的使用及其效果：

@{

ViewBag.Title = "Index";

}

<h2>Basic Links &Aacute; URLs</h2>

<table>

<thead><tr><th>Helper</th><th>Output</th></tr></thead>

<tbody>

<tr>

<td>Url.Content("~/Content/Site.css")</td>

<td>@Url.Content("~/Content/Site.css")</td>

</tr>

<tr>

<td>Html.ActionLink("My Link", "Index", "Home")</td>

<td>@Html.ActionLink("My Link", "Index", "Home")</td>

</tr>

<tr>

<td>Url.Action("GetPeople", "People")</td>

<td>@Url.Action("GetPeople", "People")</td>

</tr>

<tr>

<td>Url.RouteUrl(new { controller = "People", action = "GetPeople" })</td>

<td>@Url.RouteUrl(new { controller = "People", action = "GetPeople" })</td>

</tr>

<tr>

<td>Html.RouteLink("My Link", new {controller = "People", action = "GetPeople"})</td>

<td>@Html.RouteLink("My Link", new { controller = "People", action = "GetPeople" })</td>

</tr>

<tr>

<td>Html.RouteLink("My Link", "FormRoute", new {controller = "People", action = "GetPeople"})</td>

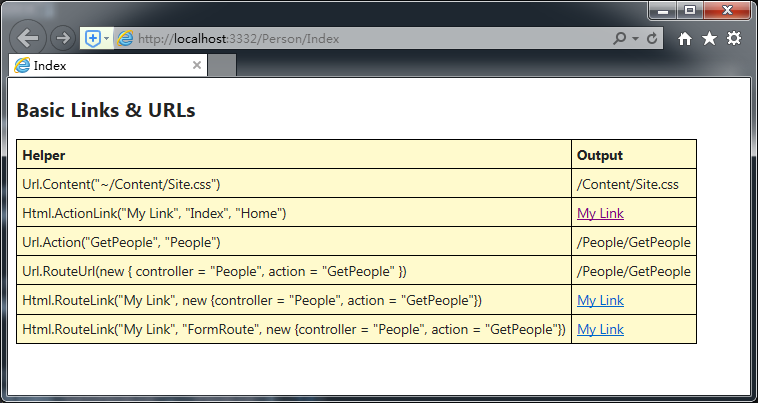
<td>@Html.RouteLink("My Link", "FormRoute", new { controller = "People", action = "GetPeople" })</td>

</tr>

</tbody>

</table>

最终运行效果如：



## 使用MVC的渐进式Ajax

Ajax即Asynchronous JavaScript and XML（异步JavaScript与XML）的缩写。相对于其XML部分，其异步部分更为重要，且用途更广。MVC框架实现了对其支持，这也方便了我们的开发。

### 创建同步表单视图

下面先对GetPeople动作方法创建视图，内容如下：

@using HelperMethods.Models

@model IEnumerable<Person>

@{

ViewBag.Title = "GetPeople";

}

<h2>Get People</h2>

<table>

<thead><tr><th>First</th><th>Last</th><th>Role</th></tr></thead>

<tbody>

@foreach (Person p in Model)

{

<tr>

<td>@p.FirstName</td>

<td>@p.LastName</td>

<td>@p.Role</td>

</tr>

}

</tbody>

</table>

@using (Html.BeginForm())

{

<div>

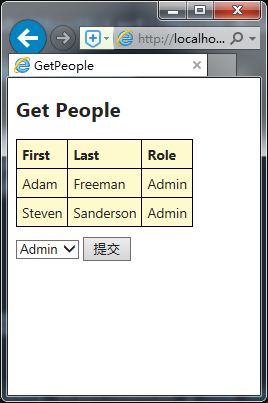
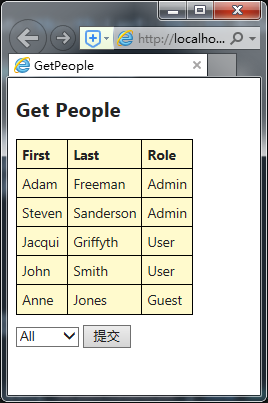
@Html.DropDownList("selectedRole", new SelectList(new[] { "All" }.Concat(Enum.GetNames(typeof(Role)))))

<button type="submit">提交</button>

</div>

}

效果如下图：



这是一个HTML表单基本限制的一个简单示例，它会在提交时重载整个页面。这在真实环境中将会带来很大的代价，而且，在页面重载的时候还不能执行其他操作，我们只能等待最终展示完成。所以，也就需要一个能够异步加载的方法了，后面就来看看这一技术的使用。

### 为渐进式Ajax准备项目

若要使用渐进式Ajax特性的，需要在程序中两个地方配合创建。

第一：在/Web.config文件（项目根文件夹中的那个）中，configuration/appSettings元素含有一个用于UnobtrusiveJavaScriptEnabled属性的项，必须将其设置为true（MVC框架为创建新项目是默认也会是一个true值），如：

<configuration>

…

<appSettings>

<add key="webpages:Version" value="2.0.0.0" />

<add key="webpages:Enabled" value="false" />

<add key="PreserveLoginUrl" value="true" />

<add key="ClientValidationEnabled" value="true" />

**<add key="UnobtrusiveJavaScriptEnabled" value="true" />**

</appSettings>

…

</configuration>

第二：添加对实现渐进式Ajax功能的jQuery JavaScript库的引用。我们在开发时可以在单个视图中引用该库，但更常用的一种方式是在布局文件中进行引用，以便实现在使用该布局的视图中能够通用。下面是在/Views/Shared/\_Layout.cshtml文件添加了两个对JavaScript库的引用：

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width" />

<title>@ViewBag.Title</title>

<link href="~/Content/Site.css" rel="stylesheet" />

<style type="text/css">

label {

display: inline-block;

width: 100px;

}

.dateElem {

margin: 5px;

}

h2 > label {

width: inherit;

}

.editor-label, .editor-field {

float: left;

}

.editor-label, .editor-label label, .editor-field input {

height: 20px;

}

.editor-label {

clear: left;

}

.editor-field {

margin-left: 10px;

margin-top: 10px;

}

input[type=submit] {

float: left;

clear: both;

margin-top: 10px;

}

.column {

float: left;

margin: 10px;

}

</style>

**<script src="~/Scripts/jquery-1.8.2.min.js" type="text/javascript"></script>**

**<script src="~/Scripts/jquery.unobtrusive-ajax.min.js" type="text/javascript"></script>**

</head>

<body>

@RenderBody()

</body>

</html>

当使用Basic模板选项创建MVC项目时，Visual Studio会将这些以script元素进行引用的文件添加到项目的Scripts文件夹中。（注意，这里添加的是min版本的js文件，这种版本是压缩版，在部署到服务器上时使用这种版本可以增加网站的加载效率，但是开发的时候建议使用普通版，这样方便阅读源代码。）

## 创建渐进式Ajax表单