# 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People控制器，并在其中定义一个Person模型对象的集合，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 PeopleController: 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>

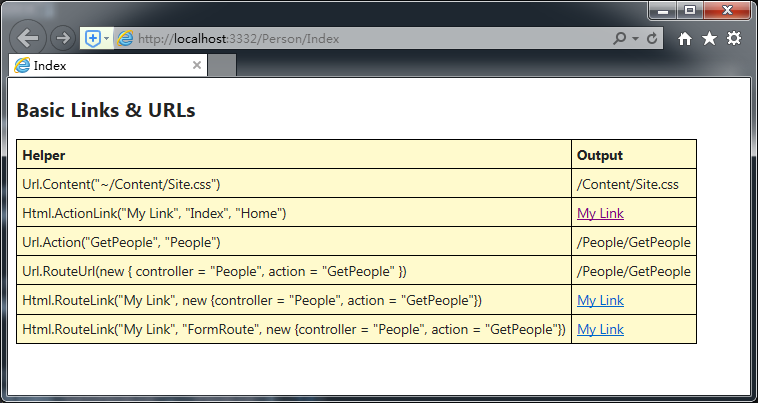
<td>@Html.RouteLink("My Link", "FormRoute", new { controller = "People", action = "GetPeople" })</td>

</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>

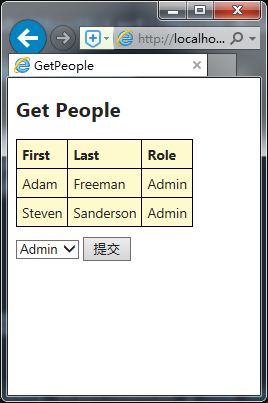
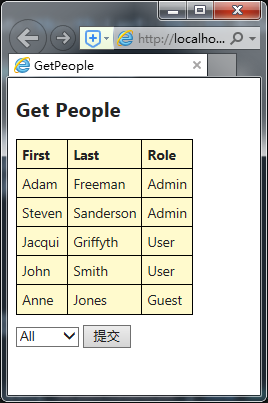
@Html.DropDownList("selectedRole", new SelectList(new[] { "All" }.Concat(Enum.GetNames(typeof(Role)))))

<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

前面已经做好了创建渐进式Ajax特性的准备，后面我们通过以等价Ajax逐步替换常规的同步表单的方式实现一个Ajax表单来了解渐进式Ajax特性的工作原理。

### 准备控制器

这里需要实现的是当点击提交按钮时，只有HTML的table元素中的数据被更新。因此，就需要我们首先重构People控制器中的动作方法，以便通过一个子动作仅获取想要的数据：

using HelperMethods.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace HelperMethods.Controllers

{

public class PeopleController : 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 PartialViewResult GetPeopleData(string selectedRole = "All")**

**{**

**IEnumerable<Person> data = personData;**

**if (selectedRole != "All")**

**{**

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

**data = personData.Where(p => p.Role == selected);**

**}**

**return PartialView(data);**

**}**

**public ActionResult GetPeople(string selectedRole = "All")**

**{**

**return View((object)selectedRole);**

**}**

}

}

此时，我们删去了GetPeople动作方法的HttpPost版本，仅保留了一个GetPeople动作方法。并新增了GetPeopleData动作，用于选择需要显示的Person对象，并传递给PartialView方法，以便生成所需的表格行。

好了，再为GetPeopleData动作方法创建一个新的分部视图文件即可（视图文件路径：/Views/People/GetPeopleData.cshtml），该分部视图仅负责生成填充表格的tr元素：

@using HelperMethods.Models

@model IEnumerable<Person>

@foreach (Person p in Model)

{

<tr>

<td>@p.FirstName</td>

<td>@p.LastName</td>

<td>@p.Role</td>

</tr>

}

更新一下/Views/People/GetPeople.cshtml视图：

@using HelperMethods.Models

@model string

@{

ViewBag.Title = "GetPeople";

}

<h2>Get People</h2>

<table>

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

<tbody>

**@Html.Action("GetPeopleData", new { selectedRole = Model })**

</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>

}

### 创建Ajax表单

目前虽然仍是一个同步表单，但已经在控制器中将功能分离开了，使得可以通过GetPeopleData动作只请求表格的行。

现在来看看如何使用Ajax请求这一目标动作：

@using HelperMethods.Models

@model string

@{

ViewBag.Title = "GetPeople";

**AjaxOptions ajaxOpts = new AjaxOptions**

**{**

**UpdateTargetId = "tableBody"**

**};**

}

<h2>Get People</h2>

<table **id="tableBody"**>

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

<tbody>

@Html.Action("GetPeopleData", new { selectedRole = Model })

</tbody>

</table>

**@using (Ajax.BeginForm("GetPeopleData", ajaxOpts))**

{

<div>

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

<button type="submit">提交</button>

</div>

}

其实，MVC框架支持Ajax表单的核心在于Ajax.BeginForm辅助器方法，它可以接收一个AjaxOptions对象。很多人喜欢在视图开始出以Razor代码块的形式创建该对象，但如果愿意，直接在调用Ajax.BeginForm时内联地创建也并非不可。

AjaxOptions对象拥有一组常用属性，用来配置以何种形式向服务器发送异步请求，以及对取回的数据做哪些处理，下表列举了这些属性：

|  |  |
| --- | --- |
| 属性 | 描述 |
| Confirm | 在形成Ajax请求之前，设置显示给用户的确认窗口中的消息 |
| HttpMethod | 设置用来形成请求的HTTP方法——必须是GET或Post |
| InsertionMode | 指定从服务器接受的内容以何种方式插入到HTML。有三种选择：InsertAfter、InsertBefore和Replace（默认值） |
| LoadingElementId | 指定HTML元素的ID，这是执行Ajax请求期间要显示的HTML元素 |
| LoadingElementDuration | 指定动画的持续时间，用于显露由LoadingElementId指定的元素 |
| UpdateTargetId | 设置HTML元素的ID，从服务器接收的内容将被插入到该元素中 |
| Url | 设置所请求的服务器端URL |

AjaxOptions还有一些属性，能够指定请求生命周期不同阶段的回调。后面将重点介绍。

### 理解渐进式Ajax的工作机制

在调用Ajax.BeginForm辅助器方法时，用AjaxOptions对象指定的选项被转换成运用于form元素的标签属性，下面是一段生成的form元素结果：

<form id="form0" action="/People/GetPeopleData" method="post" **data-ajax-update="#tableBody" data-ajax-mode="replace"** data-ajax="true">

…

</form>

当浏览器加载GetPeopleData.cshtml视图渲染的HTML页面时，jquery.unobtrusive-ajax.js库中的JavaScript会扫描这些HTML元素，通过考察data-ajax标签属性为true的元素，能够识别出这是一个Ajax表单。

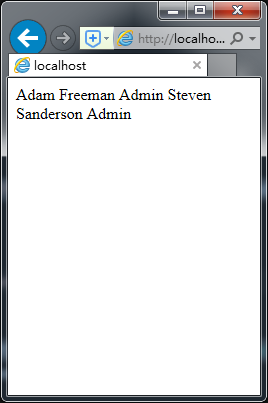
其他以data-ajax开头的标签属性含有用AjaxOptions类指定的值。这些配置选项被用于配置jQuery，jQuery具有对Ajax请求进行管理的内建支持。

提示：并不是必须使用MVC框架的渐进式Ajax支持。还有其他办法，包括直接使用jQuery。但是，建议选用一种技术，最好坚持下去，不要在同一个视图中将MVC框架的渐进式Ajax支持与其他技术混合使用，这可能导致一些不适宜的交互影响。比如：可能会复制或丢弃Ajax请求。

## 设置Ajax选项

### 确保优雅降级

前面已经实现了Ajax的异步请求，但这么做也有一个问题，那就是如果用户禁用JavaScript（或使用不支持JavaScript的浏览器），就不能正常工作了，此时，提交表单的一个结果就是使得浏览器放弃当前的HTML页面，并用目标动作方法返回的片段代替，如下图：



我们可以通过使用AjaxOptions.Url属性，以便指定异步请求的目标URL作为Ajax.BeginForm方法的参数，而不是以动作名称作为其参数的方式解决这个问题：

@using HelperMethods.Models

@model string

@{

ViewBag.Title = "GetPeople";

AjaxOptions ajaxOpts = new AjaxOptions

{

UpdateTargetId = "tableBody",

**Url = Url.Action("GetPeopleData")**

};

}

<h2>Get People</h2>

<table id="tableBody">

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

<tbody>

@Html.Action("GetPeopleData", new { selectedRole = Model })

</tbody>

</table>

**@using (Ajax.BeginForm(ajaxOpts))**

{

<div>

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

<button type="submit">提交</button>

</div>

}

通过上面的这种用法，会产生这样的效果：如果未启用JavaScript，则创建一个回递给原始动作方法的form元素，如：

<form id="form0" **action="/People/GetPeople"** method="post" **data-ajax-url="/People/GetPeopleData"** data-ajax-update="#tableBody" data-ajax-mode="replace" data-ajax="true">

…

</form>

如果启用了JavaScript，则渐进式Ajax库会以data-ajax-url标签属性为目标URL，该属性引用了子动作。如果禁用了JavaScript，则浏览器会使用常规表单的递交技术，其目标URL取自action属性，它会回递给生成完整HTML页面的动作方法。

### 在Ajax请求期间给用户提供反馈

由于使用Ajax进行请求时，请求的动作是在后台进行的，所以这也造成了用户无法了解正在发生的事情。但是，可以使用AjaxOptions.LoadingElementId和AjaxOptions.LoadingElementDuration属性通知用户此刻正在执行的请求。如：

@using HelperMethods.Models

@model string

@{

ViewBag.Title = "GetPeople";

AjaxOptions ajaxOpts = new AjaxOptions

{

UpdateTargetId = "tableBody",

Url = Url.Action("GetPeopleData"),

// 指定 HTML 元素的 ID，这是执行 Ajax 请求期间要显示的 HTML 元素

LoadingElementId = "loading",

// 指定动画持续的时间，用于显露由 LoadingElementId 指定的元素

LoadingElementDuration = 1000 };

}

<h2>Get People</h2>

**<div id="loading" class="load" style="display:none">**

**<p>数据加载中...</p>**

**</div>**

<table id="tableBody">

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

<tbody>

@Html.Action("GetPeopleData", new { selectedRole = Model })

</tbody>

</table>

@using (Ajax.BeginForm(ajaxOpts))

{

<div>

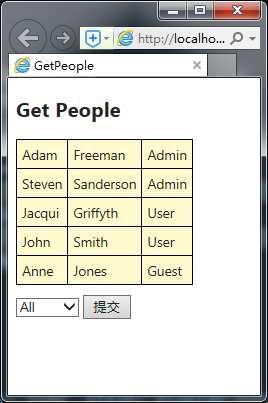
@Html.DropDownList("selectedRole", new SelectList(new[] { "All" }.Concat(Enum.GetNames(typeof(Role)))))

<button type="submit">提交</button>

</div>

}

效果如图：



…

@{

ViewBag.Title = "GetPeople";

AjaxOptions ajaxOpts = new AjaxOptions

{

UpdateTargetId = "tableBody",

Url = Url.Action("GetPeopleData"),

// 指定 HTML 元素的 ID，这是执行 Ajax 请求期间要显示的 HTML 元素

LoadingElementId = "loading",

// 指定动画持续的时间，用于显露由 LoadingElementId 指定的元素

LoadingElementDuration = 1000,

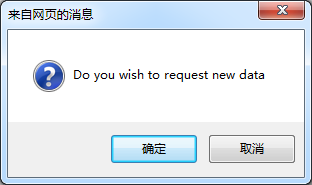
Confirm = "Do you wish to request new data"

};

}

…

上面的修改补充了一个提示，它会在用户每次提交表单时显示（当然，实际项目中还是要慎重使用这一功能——它可能惹恼用户哦~）：



## 创建Ajax链接

除了表单外，Ajax还有异步执行的a元素。它十分类似于Ajax表单的工作方式，下面是在GetPeople.cshtml视图中添加的Ajax连接：

…

<div>

**@foreach (string role in Enum.GetNames(typeof(Role)))**

**{**

**<div class="ajaxLink">**

**@Ajax.ActionLink(role, "GetPeopleData",**

**new { selectedRole = role },**

**new AjaxOptions { UpdateTargetId = "tableBody" })**

**</div>**

**}**

</div>

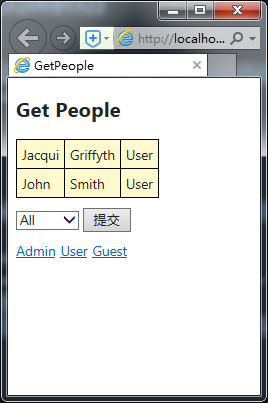
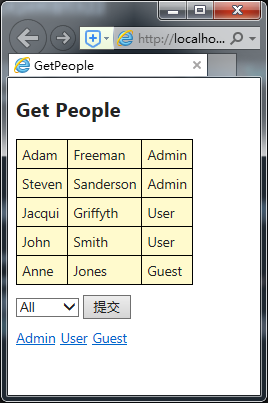
…

下面是这段代码产生的HTML的a元素片段：

<a **href="/People/GetPeopleData?selectedRole=Admin"** data-ajax-update="#tableBody" data-ajax-mode="replace" data-ajax="true">Admin</a>

注意，上面已对href进行了加粗，这是因为由于路由配置中没有selectedRole变量条目，因此，其标签属性生成的URL用查询字符串方式指定了链接所表示的角色。

下图是点击这些链接的效果：



**确保链接的优雅降级**

Ajax链接与Ajax表单一样，都会遇到如果浏览器不支持或禁用JavaScript时的问题，解决办法还是使用Ajax.Url属性来指定Ajax请求的URL，同时这里还要对Ajax.ActionLink辅助器方法指定GetPeople动作：

…

<div>

@foreach (string role in Enum.GetNames(typeof(Role)))

{

<div class="ajaxLink">

@Ajax.ActionLink(role, "**GetPeople**",

new { selectedRole = role },

new AjaxOptions

{

UpdateTargetId = "tableBody",

**Url = Url.Action("GetPeopleData", new { selectedRole = role })**

})

</div>

}

</div>

…

独立的AjaxOptions能够为每个链接的Url属性指定一个不同的值，并为非JavaScript浏览器提供优雅降级支持。

## 使用Ajax回调

下表为AjaxOptions中定义的一组回调属性：

|  |  |  |
| --- | --- | --- |
| 属性 | jQuery事件 | 描述 |
| OnBegin | beforeSend | 在发送请求之前立即调用 |
| OnComplete | complete | 请求成功时调用 |
| OnFailure | error | 请求失败时调用 |
| OnSuccess | success | 请求已完成时调用，不管请求是否成功或失败 |

每一个AjaxOptions回调属性都与jQuery库所支持的一个Ajax事件相关联。下面就使用一个实例展示一下这些回调属性查看Ajax请求的进展：

@using HelperMethods.Models

@model string

@{

ViewBag.Title = "GetPeople";

AjaxOptions ajaxOpts = new AjaxOptions

{

UpdateTargetId = "tableBody",

Url = Url.Action("GetPeopleData"),

// 指定 HTML 元素的 ID，这是执行 Ajax 请求期间要显示的 HTML 元素

LoadingElementId = "loading",

// 指定动画持续的时间，用于显露由 LoadingElementId 指定的元素

LoadingElementDuration = 1000,

Confirm = "Do you wish to request new data"

};

}

**<script type="text/javascript">**

**function OnBegin() {**

**alert("This is the OnBegin Callback");**

**}**

**function OnSuccess(data) {**

**alert("This is the OnSuccess Callback: " + data);**

**}**

**function OnFailure(request, error) {**

**alert("This is the OnFailure Callback" + error);**

**}**

**function OnComplete(request, status) {**

**alert("This is the OnComplete Callback" + status);**

**}**

**</script>**

<h2>Get People</h2>

<div id="loading" class="load" style="display:none">

<p>数据加载中...</p>

</div>

<table id="tableBody">

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

<tbody>

@Html.Action("GetPeopleData", new { selectedRole = Model })

</tbody>

</table>

@using (Ajax.BeginForm(ajaxOpts))

{

<div>

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

<button type="submit">提交</button>

</div>

}

<div>

@foreach (string role in Enum.GetNames(typeof(Role)))

{

<div class="ajaxLink">

@Ajax.ActionLink(role, "GetPeople",

new { selectedRole = role },

new AjaxOptions

{

UpdateTargetId = "tableBody",

Url = Url.Action("GetPeopleData", new { selectedRole = role }),

**OnBegin = "OnBegin",**

**OnFailure = "OnFailure",**

**OnSuccess = "OnSuccess",**

**OnComplete = "OnComplete"**

})

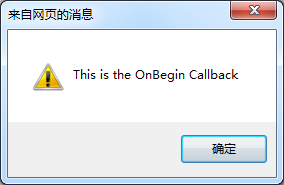
</div>

}

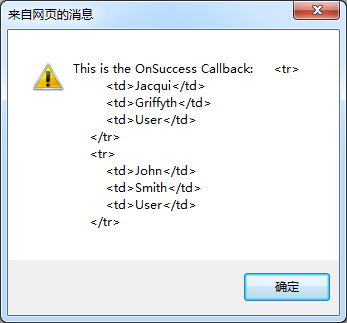
</div>

上面代码中定义了四个函数以便使用表中所述回调属性，但我们只需要说明问题即可，因此将逻辑进行了简化——仅仅输出一些消息。这样我们随便点击一个连接就可以看到这一过程了（Ajax请求的进展过程）：

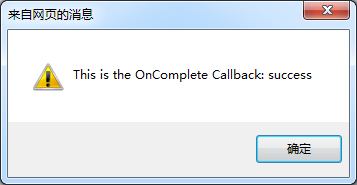
1、OnBegin



2、OnSuccess



3、OnComplete



清楚了Ajax请求的过程，可以在实际项目中利用这一过程中对应的函数可以实现我们想做的任何事情，其中最常见也最有用的做法之一就是处理JSON数据，这在以后的学习中可以见到。

## 使用JSON