# Web WPI

## Web API简介

Web API是新添加到ASP.NET平台上的，能够快捷地创建Web服务，以便对HTTP客户端提供API。

它建立在常规MVC框架应用程序的基础之上，但不属于MVC框架部分。其核心是ASP.NET平台的一部分，因而能够用于其他类型的Web应用程序，或作为独立的Web服务引擎。

建立Web API特性的基础是对MVC框架应用程序添加一种特殊的控制器——API控制器，它有两个明显的特征：

1. 动作方法返回的是模型对象，而不是ActionResult类型（或派生类型）的对象——模型对象会被编码成JSON格式
2. 动作方法是根据请求中所使用的HTTP方法来选择的

API控制的作用是提供传递Web数据的服务，并不支持视图、布局，也不支持一直用来为浏览器显示生产HTML的任何其他特性。API控制器能够支持任何可以使用Web的客户端，最常见的用法是对Web应用程序的Ajax请求提供服务。

所以，简单一句话Web API就是为客户端提供Web服务。

为了实现这种Web服务，客户端需要做以下两件事：

第一：“要求服务”——客户端通常要使用JavaScript（一般是jQuery或渐进式Ajax库）向服务端发送要求服务的请求，一般为Ajax请求。

第二：“处理服务”——对Web API返回的数据进行处理（一般用JavaScript处理），并显示结果。

所以，API控制器很明确地将项目中那些与数据有关的动作和那些与视图相关的动作分离开了，同时也使得创建通用目的的Web API变得快速而简单。

## 创建Web API程序

在创建MVC项目时可以直接使用Web API模板，但这里打算使用基本模板，直接使用Web API只是自动添加了一些便利的一般性代码，而通过创建基本模板，然后一步一步添加所需内容，更容易弄清楚其中“真相”。

示例项目的名称为：WebServices，其他相应的模型对象、存储库、控制器和视图当然也都需要，还有更重要的API控制器，这些都会在下面小节一一介绍。

### 建立模型和存储库

模型类：Reservation.cs

using System.ComponentModel.DataAnnotations;

namespace WebServices.Models

{

public class Reservation

{

public int ReservationId { get; set; }

public string ClientName { get; set; }

public string Location { get; set; }

}

}

对于该模型类不需要太多的关注其含义，将注意力集中在Web API即可。下面是一些存储相关的接口和简单实现：

接口：IReservationRepository.cs

using System.Collections.Generic;

namespace WebServices.Models

{

public interface IReservationRepository

{

IEnumerable<Reservation> GetAll();

Reservation Get(int id);

Reservation Add(Reservation item);

void Remove(int id);

bool Update(Reservation item);

}

}

存储类：ReservationRepository.cs，该类除了实现上述接口，还定义了一些内存中的模型对象样本，目的只是为了简化项目，实际项目中还是要与数据库进行交互的。

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace WebServices.Models

{

public class ReservationRepository : IReservationRepository

{

private List<Reservation> data = new List<Reservation>

{

new Reservation { ReservationId = 1, ClientName = "Adam", Location = "London" },

new Reservation { ReservationId = 2, ClientName = "Steve", Location = "New York" },

new Reservation { ReservationId = 3, ClientName = "Jacqui", Location = "Paris" }

};

private static ReservationRepository repo = new ReservationRepository();

public static IReservationRepository getRepository()

{

return repo;

}

public IEnumerable<Reservation> GetAll()

{

return data;

}

public Reservation Get(int id)

{

var matches = data.Where(r => r.ReservationId == id);

return matches.Count() > 0 ? matches.First() : null;

}

public Reservation Add(Reservation item)

{

item.ReservationId = data.Count + 1;

data.Add(item);

return item;

}

public void Remove(int id)

{

Reservation item = Get(id);

if (item != null)

{

data.Remove(item);

}

}

public bool Update(Reservation item)

{

Reservation storedItem = Get(item.ReservationId);

if (storedItem != null)

{

storedItem.ClientName = item.ClientName;

storedItem.Location = item.Location;

return true;

}

else

{

return false;

}

}

}

}

### 添加Home控制器

在实际项目中是可以混合使用常规控制器和API控制器的，而且如果希望支持HTML客户端，混合使用就很有必要了。

Home控制器——一个不为视图传递任何模型对象的简单控制器，目的是为了配合后面实现完全的Web服务方式，具体内容如下：

using System.Web.Mvc;

namespace WebServices.Controllers

{

public class HomeController : Controller

{

public ActionResult Index()

{

return View();

}

}

}

### 添加视图及CSS

将下面这些CSS样式添加到/Content/Site.css文件中：

table {

margin: 10px 0;

}

th {

text-align: left;

}

.nameCol {

width: 100px;

}

.locationCol {

width: 100px;

}

.selectCol {

width: 30px;

}

.display {

float: left;

border: thin solid black;

margin: 10px;

padding: 10px;

}

.display label {

display: inline-block;

width: 100px;

}

Index.cshtml视图：

@{

ViewBag.Title = "Index";

}

@section scripts{

<script src="~/Scripts/jquery.unobtrusive-ajax.js"></script>

}

<div id="summaryDisplay" class="display">

<h4>Reservations</h4>

<table>

<thead>

<tr>

<th class="selectCol"></th>

<th class="nameCol">Name</th>

<th class="locationCol">Location</th>

</tr>

</thead>

<tbody id="tableBody">

<tr><td colspan="3">The data is loading</td></tr>

</tbody>

</table>

<div>

<button id="refresh">Refresh</button>

<button id="add">Add</button>

<button id="edit">Edit</button>

<button id="delete">Delete</button>

</div>

</div>

<div id="addDisplay" class="display">

<h4>Add New Reservation</h4>

@{

AjaxOptions addAjaxOpts = new AjaxOptions

{

// 这里将放入各种选项

};

}

@using (Ajax.BeginForm(addAjaxOpts))

{

@Html.Hidden("ReservationId", 0)

<p><label>Name:</label>@Html.Editor("ClientName")</p>

<p><label>Location:</label>@Html.Editor("Location")</p>

<button type="submit">Submit</button>

}

</div>

<div id="editDisplay" class="display">

<h4>Edit Reservation</h4>

<form id="editForm">

<input id="editReservationId" type="hidden" name="ReservationId" />

<p><label>Name:</label><input id="editClientName" name="ClientName" /></p>

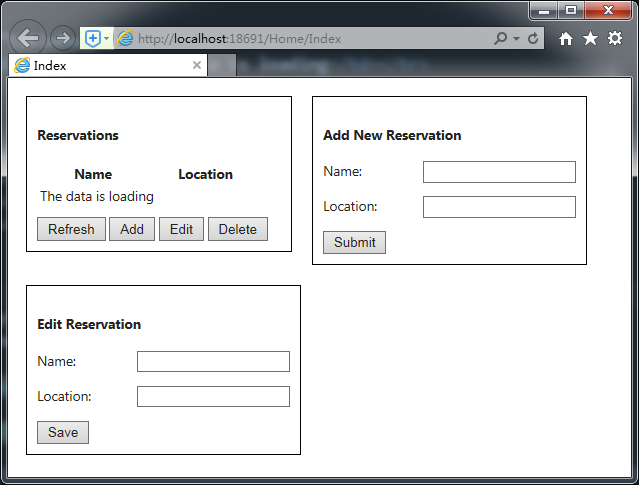
<p><label>Location:</label><input id="editLocation" name="Location" /></p>

</form>

<button id="submitEdit" type="submit">Save</button>

</div>

对于该视图，目前三个HTML片段内容都会显示出来，后面将通过JavaScript代码对其可见性进行控制。现在先来感受一下该视图的效果：



当完成改程序时，只有Reservation片段会显示给用户。后面将从服务器加载模型数据，并填充到table元素中。

Add New Reservation片段含有一个启用了渐进式Ajax的form，它是用来将数据回递给服务器，以便在存储库中创建新的Reservation对象。其中使用的AjaxOptions对象还未设置任何选项，后面等示例项目其余部分完成后，将会进行完善。

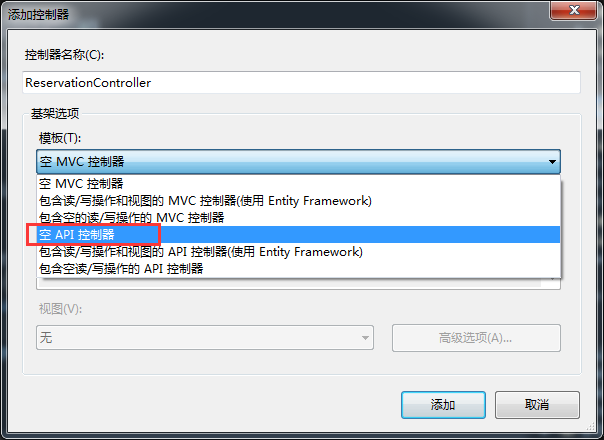
Edit Reservation片段是用来编辑Reservation对象的。这里没有采用Ajax的form，将打算直接使用jQuery的Ajax支持。

## 创建API控制器

现在示例项目准备的差不多了，来看看该如何添加API控制器吧。

在解决方案资源管理器中右击“Controllers”文件夹，在菜单中选择“添加”->“控制器”，将控制器名改为：ReservationController，并选择“空API控制器”，如图：





下面是该模板创建的API控制器的默认内容：

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Net.Http;

using System.Web.Http;

namespace WebServices.Controllers

{

public class ReservationController : **ApiController**

{

}

}

这个控制器的基类与其他常规的不一样，使用的是ApiController，其命名空间为System.Web.Http. ApiController，它实现的是IHttpController接口（*书中提到的是IController接口，难道是微软后来调整了接口？*）。下面是为该控制器添加的一些功能实现：

using System.Collections.Generic;

using System.Web.Http;

using WebServices.Models;

namespace WebServices.Controllers

{

public class ReservationController : ApiController

{

**IReservationRepository repo = ReservationRepository.getRepository();**

**public IEnumerable<Reservation> GetAllReservation()**

**{**

**return repo.GetAll();**

**}**

**public Reservation GetReservation(int id)**

**{**

**return repo.Get(id);**

**}**

**public Reservation PostReservation(Reservation item)**

**{**

**return repo.Add(item);**

**}**

**public bool PutReservation(Reservation item)**

**{**

**return repo.Update(item);**

**}**

**public void DeleteReservation(int id)**

**{**

**repo.Remove(id);**

**}**

}

}

提示：这里使用了ReservationRepository.getRepository静态方法获取IReservationRepository。如果在控制器中创建一个新的ReservationRepository对象，那么在创建控制器的新实例对API请求进行服务时，所作的修改不会反映到发送给客户端的数据中。这是因为示例代码使用了内存存储的方式，如果使用的是持久化数据存储的实际存储库，则不会发生这种情况。

**测试API控制器**

如果使用了IE这样的浏览器（本人使用的是IE11），导航到/api/reservation，将会提示保存或打开一个包含以下JSON数据的文件：

[{"ReservationId":1,"ClientName":"Adam","Location":"London"},{"ReservationId":2,"ClientName":"Steve","Location":"New York"},{"ReservationId":3,"ClientName":"Jacqui","Location":"Paris"}]

如果是其他浏览器（如Google Chrome）将会得到XML数据。

<ArrayOfReservation>

<Reservation>

<ClientName>Adam</ClientName>

<Location>London</Location>

<ReservationId>1</ReservationId>

</Reservation>

<Reservation>

<ClientName>Steve</ClientName>

<Location> New York </Location>

<ReservationId>2</ReservationId>

</Reservation>

<Reservation>

<ClientName> Jacqui </ClientName>

<Location> Paris </Location>

<ReservationId>3</ReservationId>

</Reservation>

</ArrayOfReservation>

从数据中可以看出有两件有意思的事：

1. 对/api/reservation的请求得到的是全部模型对象及其属性的列表，从此可以看出它调用了Reservation控制器的GetAllReservation方法。
2. 不同的浏览器接收不同的数据格式——JSON格式或XML格式。

之所以可能得到不同的数据格式，是因为Web API会使用包含在请求中的HTTP的Accept报头，根据不同的客户端发送相应的数据格式的数据。

IE的报头：Accept: text/html, application/xhtml+xml, \*/\*。其内容说明了浏览器将首先选择text/html格式的数据，其次是application/xhtml+xml，\*/\*表示如果前面两种不可用，则可以是其他任何数据类型。对于Web API支持JSON和XML两种格式的数据，但将优先选择JSON格式的数据。

Google Chrome的报头：Accept:text/html,application/xhtml+xml, **application/xml**;q 0.9,\*/\*;q 0.8。高亮部分说明了浏览器将可以优先接收application/xml数据，因此，Web API向其返回了XML数据。

## 理解API控制器工作原理

API控制器拥有自己的路由配置，是在/App\_Start/WebApiConfig.cs配置的。从默认的配置中可以看出，其中使用了不同于常规MVC路由中使用的类。Web API被作为了独立的ASP.NET特性来实现，且可以将它独立于MVC框架之外使用。下面是默认的API路由配置：

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web.Http;

namespace WebServices

{

public static class WebApiConfig

{

public static void Register(HttpConfiguration config)

{

config.Routes.MapHttpRoute(

name: "DefaultApi",

routeTemplate: "api/{controller}/{id}",

defaults: new { id = RouteParameter.Optional }

);

}

}

}

静态方法Register在系统启动时由Global.asax中的Application\_Start方法调用，并注册。该方法接收一个HttpConfiguration对象，通过该对象的Routes属性提供了对路由的访问。路由是使用MapHttpRoute方法来创建的。从配置中也可以看出，它与常规MVC路由还有个区别，就是它没有action片段变量。

### 理解API控制器的动作选择

默认路由有一个静态api片段，还有controller和id两个片段，其中id片段变量是可以先的。但是它没有action片段，所以，正是从这一点可以看出API控制器形成的行为。

那这样的该如何决定使用哪一个动作方法呢？实际上是接收到一条与Web API路由匹配时，通过形成该请求的HTTP方法来决定的。

ApiController类通过路由可以指定它需要的是哪一个目标控制器，并使用HTTP方法查找合适的动作方法。

在创建API控制器动作方法时有一个约定：以它所支持的HTTP方法为前缀，然后是它所操作模型的某种参考。当然，如果其他动作方法，但只要其中包含用以形成该请求的HTTP方法名也是可以的。

对于示例而言，如果是GET请求，则将会在GetAllReservation方法和GetReservation方法之间进行选择，具体决定使用哪一个，控制器则会对参数进行考察，给予优先的是接收并使用了路由变量的。如：在请求/api/reservation时，没有controller以外的其他路由变量，所以选择的是GetAllReservation方法。

在控制器中还为其他动作制定了目标，如：POST、DELETE、Put。像Reservation控制器这样的这种风格是“表现式状态传输（Representation State Transfer，即REST）”风格的Web API的基础，经常被称为RESTful服务（REST化服务），在这种服务中，某个操作是通过URL和请求该URL的HTTP方法相结合的方式来指定。

### 将HTTP方法映射到动作方法

使用HTTP方法得出动作方法的方式虽然很好，但也有它的问题，比如PutReservation就不如UpdateReservation更为自然，更能明确动作与存储库之间的关系。这里建议使用另一种方式进行处理：创建一种不同的控制器。在有些情况下，希望通过API提供的方法与存储的功能是偏离的，因而拥有独立的API控制器会更易于管理。

System.Web.Http命名空间包含了一组注解属性，可以用来指定一个动作所使用的HTTP方法，如：

using System.Collections.Generic;

using System.Web.Http;

using WebServices.Models;

namespace WebServices.Controllers

{

public class ReservationController : ApiController

{

IReservationRepository repo = ReservationRepository.getRepository();

public IEnumerable<Reservation> GetAllReservation()

{

return repo.GetAll();

}

public Reservation GetReservation(int id)

{

return repo.Get(id);

}

**[HttpPost]**

public Reservation **CreateReservation**(Reservation item)

{

return repo.Add(item);

}

**[HttpPut]**

public bool **UpdateReservation**(Reservation item)

{

return repo.Update(item);

}

public void DeleteReservation(int id)

{

repo.Remove(id);

}

}

}

## 编写使用Web API的JavaScript代码

到现在，是该考虑如何使用JavaScript代码使用所创建的这个Web API的时候了。示例将使用jQuery来操作Index.cshtml视图所渲染的HTML元素，并处理将要形成的Ajax请求。——注意，这是指形成Ajax请求以及对该请求的形成和结果进行处理两项工作。

### 基本功能

首先创建一个/Scripts/Home文件夹，然后在其中添加一个新的脚本：Index.js——这遵循了按视图组织脚本的约定。最后，在Index.cshtml视图文件的script小节中添加一个script元素，以便载入JavaScript代码，下面是加入该视图中的代码：

@section scripts{

<script src="~/Scripts/jquery.unobtrusive-ajax.js"></script>

**<script src="~/Scripts/Home/Index.js"></script>**

}

Index.js的功能代码：

function selectView(view) {

$('.display').not('#' + view + "Display").hide();

$('#' + view + "Display").show();

}

function getData() {

$.ajax({

type: "GET",

url: "/api/reservation",

success: function (data) {

$('#tableBody').empty();

for (var i = 0; i < data.length; i++) {

$('#tableBody').append('<tr><td><input id="id" name="id" type="radio">'

+ 'value="' + data[i].ReservationId + '"/></td>'

+ '<td>' + data[i].ClientName + '</td>'

+ '<td>' + data[i].Location + '</td></tr>');

}

$('input:radio')[0].checked = "checked";

selectView("summary");

}

});

}

$(document).ready(function () {

selectView("summary");

getData();

$("button").click(function (e) {

var selectedRadio = $('input:radio:checked')

switch (e.target.id) {

case "refresh":

getData();

break;

case "delete":

break;

case "add":

selectView("add");

break;

case "edit":

selectView("edit");

break;

case "submitEdit":

break;

}

});

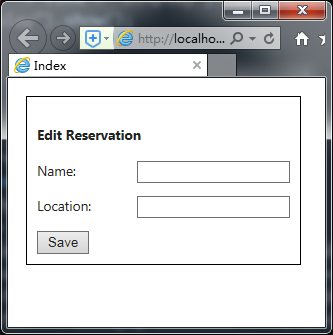
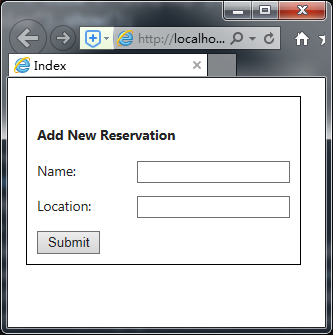
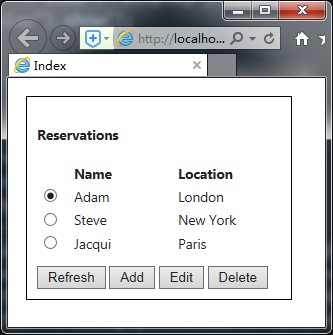
});

脚本中定义了三个函数：

1. selectView用于修改class为display的div元素的可见性，以便只有一组元素显示出来。
2. getData使用jQuery的Ajax支持，以便对/api/reservation形成GET请求，返回的JSON格式对象数组用于添加视图表格的行，以替换“data is loading”提示字符。表中每一行都含有一个单项按钮，用于让用户选择一个Reservation对象，进行编辑或删除。
3. 最后一个函数是传递给jQuery的ready函数的。其中调用了selectView函数，以便只显示summaryDisplay元素的内容，并调用getData函数加载/api/reservation地址上的数据。

除此之外，还建立了一个事件处理函数，它可以根据id标签属性的值区分是点击的哪一个按钮，以便触发相应的请求。Refresh按钮触发的是getData方法，以加载从服务器得到的数据，而对Edit和Add按钮的相应是调用selectView函数，以显示创建和编辑模型对象所需要的元素。

下面看看具体效果吧：



### 添加编辑新预约的支持

现在对前面的脚本添加一个编辑的支持：

$(document).ready(function () {

selectView("summary");

getData();

$("button").click(function (e) {

var selectedRadio = $('input:radio:checked')

switch (e.target.id) {

case "refresh":

getData();

break;

case "delete":

break;

case "add":

selectView("add");

break;

case "edit":

**$.ajax({**

**type: "GET",**

**url: "/api/reservation/" + selectedRadio.attr('value'),**

**success: function (data) {**

**$('#editReservationId').val(data.ReservationId);**

**$('#editClientName').val(data.ClientName);**

**$('#editLocation').val(data.Location);**

**selectView("edit");**

**}**

**});**

break;

case "submitEdit":

break;

}

});

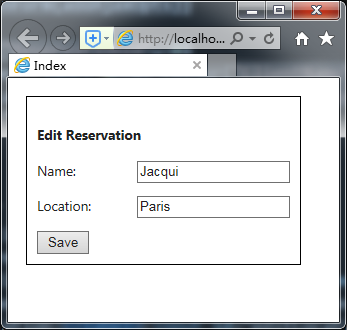
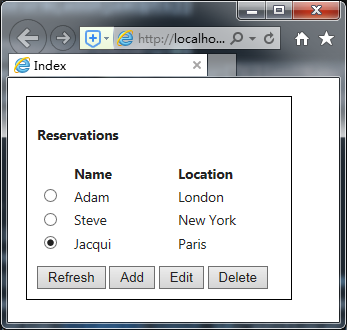
});

在getData函数中创建table的各个表行时，使用各个Reservation对象的ReservationId属性值，以设置其单选按钮元素的值，如：

<input name="id" id="id" type="radio" value="3">

当用户点击Edit按钮时，找出已选的单选按钮，并使用其value标签属性的值构建发送给服务器的请求URL。如果用户已经选择上述单选按钮，那么便请求/api/reservation/3的地址。然后通知jQuery，用户需要的是一个GET请求，并需要的是一个该URL好HTTP方法的组合，该组合以Reservation控制器中GetReservation动作方法为目标。

用取回的JSON数据来设置页面的editDisplay片段中各个input元素的值，然后调用selectView函数将它们显示给用户：



为了能够保存修改结果，需要填写对id为submitEdit的按钮进行处理的case块，如：

$(document).ready(function () {

selectView("summary");

getData();

$("button").click(function (e) {

var selectedRadio = $('input:radio:checked')

switch (e.target.id) {

case "refresh":

getData();

break;

case "delete":

break;

case "add":

selectView("add");

break;

case "edit":

$.ajax({

type: "GET",

url: "/api/reservation/" + selectedRadio.attr('value'),

success: function (data) {

$('#editReservationId').val(data.ReservationId);

$('#editClientName').val(data.ClientName);

$('#editLocation').val(data.Location);

selectView("edit");

}

});

break;

case "submitEdit":

**$.ajax({**

**type: "PUT",**

**url: "/api/reservation/" + selectedRadio.attr('value'),**

**success: function (result) {**

**if (result) {**

**var cells = selectedRadio.closest('tr').children();**

**cells[1].innerText = $('#editeClientName').val();**

**cells[2].innerText = $('#editLocation').val();**

**selectView("summary");**

**}**

**}**

**});**

break;

}

});

});

注意这里在对id为submitEdit的按钮进行处理时请求的动作方法为UpdateReservation（书中此处是PutReservation，但这在前面已经修改了，改为使用HttpPut注解属性进行映射的UpdateReservation动作方法了，但这并不是重点，重点是其参数类型），其参数为一个Reservation对象，但由于API控制器同样可以受益于模型绑定，所以，不必对其进行数据转换。

### 添加删除预约的支持

下面是添加的删除功能：

case "delete":

$.ajax({

type: "DELETE",

url: "/api/reservation/" + selectedRadio.attr('value'),

success: function (data) {

selectedRadio.closest('tr').remove();

}

});

break;

注意，作为示例，这里仅删除了表格中的行，在实际项目中还需要考虑从服务器数据库中将其删除，这样才能符合实际情况。

### 添加创建预约的支持

对于创建新的预约的功能，打算采用不同的做法——使用MVC框架的渐近式Ajax表单特性。所以，此处使用的是Index.cshtml视图，并对其中的AjaxBeginForm辅助器方法的参数对象进行了配置：

提示：如果希望对所有请求都使用Ajax表单，或希望在一个只支持GET和POST方法的浏览器中使用REST化的服务，可以使用Html.HttpMethodOverride辅助器，为表单添加一个隐藏的元素，API控制器将会对该元素进行解析，并使用目标动作方法。此时，只能重新POST请求，但它可以作为一种备用技术，特别是针对旧式浏览器。

<div id="addDisplay" class="display">

<h4>Add New Reservation</h4>

@{

AjaxOptions addAjaxOpts = new AjaxOptions

{

**OnSuccess = "getData",**

**Url = "/api/reservation"**

};

}

@using (Ajax.BeginForm(addAjaxOpts))

{

@Html.Hidden("ReservationId", 0)

<p><label>Name:</label>@Html.Editor("ClientName")</p>

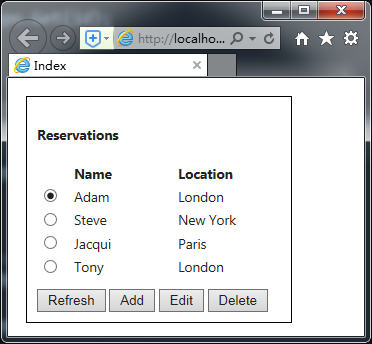
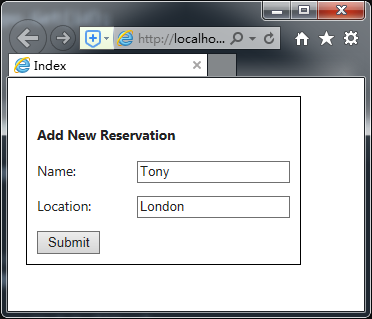
<p><label>Location:</label>@Html.Editor("Location")</p>

<button type="submit">Submit</button>

}

</div>

这个form将默认使用POST方法进行递交，而且不必动态地建立URL，因为PostReservation（应该为CreateReservation，这个和UpdateReservation方法一样是在后来改为使用相应的注解属性进行映射的，因此都使用了新的方法名。CreateReservation方法使用的是HttpPost注解属性）动作方法并不依靠片段变量参数。下图是最终效果：



现在已经介绍完如何实现Web API程序了。可以看出单纯创建API的过程包括，创建一个派生于ApiController的新控制器，以及创建其动作方法。这些动作方法的名称对应于以它们为目标的HTTP方法（或使用System.Web.Http命名空间下的注解属性进行映射）。

*注：在书中还介绍了“部署”相关的知识，这里我主要是为了记录学习MVC4框架的相关知识，就不对其进行编写学习笔记了。同样对于书中的实际真是示例项目——SportsStore也不做笔记记录了。*