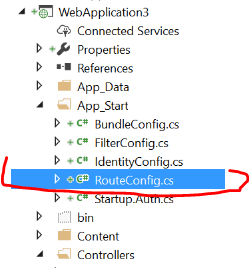
|  |
| --- |
| ` |
| ASP.NET MVC Learning |
|  |
|  |
| **William Liu** |
|  |
| **5/25/2014** |

1. MVC Routing: MVC 路由机制

Folder: App\_Start 里有个文件： RouteConfig.cs



using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using System.Web.Routing;

namespace WebApplication3

{

public class RouteConfig

{

public static void RegisterRoutes(RouteCollection routes)

{

routes.IgnoreRoute("{resource}.axd/{\*pathInfo}");

routes.MapRoute(

name: "Default",

url: "{controller}/{action}/{id}",

defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }

);

}

}

}

定义了一个静态方法 RouteConfig.RegisterRoutes(RouteCollection routes)

在全局入口文件： Global.asax 里 Application\_Start() 事件里

namespace WebApplication3

{

public class MvcApplication : System.Web.HttpApplication

{

protected void Application\_Start()

{

AreaRegistration.RegisterAllAreas();

FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);

RouteConfig.RegisterRoutes(RouteTable.Routes);

BundleConfig.RegisterBundles(BundleTable.Bundles);

}

}

}

注意： RouteTable.Routes 是一个公开的静态对象，它是RouteCollectionss，是继承自RouteBase 的Route对象的集合。所以可以在任何地方访问到它

设置路由表：通过 RouteCollection.MapRoute & RouteTable.Routes.MapRoute()

routes.MapRoute(

name: "Default",

url: "{controller}/{action}/{id}",

defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional },

constraints: new { controller = "Home" }

);

routes.MapRoute(

name: "myaccount",

url: "{controller}/{action}/{page}",

defaults: new { controller = "Students", action = "Index", page = "50" },

constraints: new

{

page = @"\d+"

}

);

路由对比：

默认行为： 如果网址路径能够在网站实体目录查找到相对应得实体文档， 就会自动略过所有网址路由对比。

可以关闭此默认行为： RouteTable.Routes.RouteExistingFiles = true;

在整个请求生命周期内会有两个主要的全局对象存在：Request Response

Request.RequestContext

RouteTable.Routes.GetVirtualPath(

Request.RequestContext,

new RouteValueDictionary(

new {

controller = "myaccount",

action = "about",

page="test",

name = "William Liu",

year = DateTime.Now

}

)

).VirtualPath

GetVirtualPath : 会通过获取当前路由表的所有路径来进行比对， 如果比对失败

GetVirtualPath 会返回 null ， 则不能调用 VirtualPath 否则出错

如何自定义物理文件到 URL 的路由：

routes.MapPageRoute("mypage", "my/myhtml", "~/views/my.html");

routes.MapRoute(

name: "myaccount",

url: "{controller}/{action}/{page}",

defaults: new { controller = "Home", action = "Index", page = "50" },

constraints: new

{

page = @"\d+"

}

);

如果出现错误， .html 没有注册， 需要修改 web.config

<system.web>

<compilation debug="true" targetFramework="4.5.2" >

<buildProviders >

<add extension=".htm" type="System.Web.Compilation.PageBuildProvider" />

<add extension=".html" type="System.Web.Compilation.PageBuildProvider"/>

</buildProviders>

</compilation>

</system.web>

路由模块的各种对象：以下是常用的有关路由的对象

RouteCollection

* RouteExistingFiles
* MapPageRoute
* MapRoute
* GetVirtualPath
* GetRouteData

RouteTable

* Routes
* RouteTable.Routes.GetRouteData

Request

* RequestContext
* RequestContext.RouteData
* Request.RequestContext.RouteData.Values
* Request.RequestContext.HttpContext

RouteBase

* GetVirtualPath
* GetRouteData

RouteData

* Route ( RouteBase )
* RouteHandler
* DataTokens (RouteValueDictionary)
* Values (RouteValueDictionary)

RouteValueDictionary : IDictionary<string, object>, ICollection<KeyValuePair<string, object>>s

public class RouteValueDictionary :

IDictionary<string, object>,

ICollection<KeyValuePair<string, object>>,

IEnumerable<KeyValuePair<string, object>>,

IEnumerable

实例化赋予初始值

RouteValueDictionary rva = new RouteValueDictionary{ { "id", 300 }, { "name", "William" } };

构造函数赋予初始值

RouteValueDictionary rvb = new RouteValueDictionary( new {id=400, name="Goody", age=38} );

RouteValueDictionary

* Add( string key, object value )

RouteValueDictionary 可以使用KeyValuePair<string, object> 来遍历所有项目

foreach(KeyValuePair<string, object>el in Model)

{

<li> @el.Key - @el.Value </li>

}

IDictionary<TKey, TValue> 的使用

IDictionary<int, string> sta = new Dictionary<int, string> { { 100, "hell" }, { 200, "good" } };

IDictionary<int, string> stb = new Dictionary<int, string>();

KeyValuePair<int, string> st1 = new KeyValuePair<int, string>(300, "will");

sta.Add(st1);

foreach(KeyValuePair<int, string> el in sta)

{

Console.WriteLine("Dictionary Key:{0} Value:{1}", el.Key, el.Value);

}

// 不支持以下的构造函数，

IDictionary<string, object> stb = new Dictionary<string, object>( new {id=2000, name="will" } );

IDictionary<TKey, TValue>

* Add(TKey key, TValue value)
* Add(KeyValuePair<TKey, TValue> kvp)

NameValueCollection

NameValueCollection nvc = new NameValueCollection {

{ "name", "William" },

{ "name", "Tommy" },

{ "age", "43" }

};

nvc.Add("class", "English");

nvc.Add("name", "Susan");

nvc.Set("age", "68");

bool hasKey = nvc.HasKeys();

string value = nvc.AllKeys.Contains("name")? nvc.Get("name") : "";

string values = nvc.AllKeys.Contains("name") ? string.Join("|", nvc.GetValues("name")) : "";

Console.WriteLine("found: {0}", value);

Console.WriteLine("founds: {0}", values);

for (int i=0; i<nvc.Count; i++)

{

Console.WriteLine(string.Format("key:{0} value:{1} values:{2}",

nvc.GetKey(i) ,

nvc.Get(i),

string.Join(" \* ", nvc.GetValues(i)) )

);

}

// NameValueCollection 不是继承自 IDictionary, 所以不能使用 KeyValueCollections

foreach(KeyValuePair<string,string> el in nvc) - 这种用法是错误的

{

Console.WriteLine("Key:{0} value:{1}", el.Key, el.Value);

}

1. VieswData , ViewBag, TempData, ViewModel
   1. ViewData – 继承自ViewDataDictionary 继承自 Idictionary<string, object>

ViewData[“key”] – 是以 object 的封箱形式保存数据,可以存放各种的数据类型

ViewData – 只会存在这次的HTTP要求中，并不象Session可以将数据带到下一个HTTP请求

ViewData["myinfo"] = "This is your information";

ViewData["myarr"] = new int[] { 11, 13, 21 };

IList<person> stu = new List<person>

{

new person { id=100, name="Tommy Li", age = 68 },

new person { id = 200, name = "Susan Yu", age = 35 },

new person { id=300, name = "Peter Hui", age = 15 }

};

ViewData["hisarr"] = stu;

在view 里使用 ViewData, 需要unboxing 拆箱使用

String: @ViewData["myinfo"]<br />

My Arrary: @string.Join(",", **(int[])**ViewData["myarr"])<br />

Peson List:

@foreach (WebApp4.Controllers.person el

in (ViewData["personList"] as **IList<WebApp4.Controllers.person>**) )

{

<li>@el.id @el.name @el.age</li>

}

* 1. ViewBag

继承自 DynamicViewDataDictionary 动态ViewData字典。 唯一和 ViewData的区别

就是动态类型.

ViewData["Message"] = "Hello World";

ViewBag.Message = "Hello World";

而者可以互换使用, 但是还是有区别：

ViewData["key one"] = "Hello World"; - 可以通过

ViewBag.key one = "Hello World"; - 不可以通过编译

Msg11: @Html.TextBox("msg11", ViewData["msg11"]) – 可以通过

Msg22: @Html.TextBox("msg22", ViewBag.msg22) - 出错 – ViewBag是动态类型不能用于绑定

Msg22: @Html.TextBox("msg22", (string)ViewBag.msg22) - 强制转化以后没有问题

* 1. TempData - 类型是 TempDataDictionary

TempData 是使用 Session来保存信息， 而且只保存一次访问,

也就是说， TempData设置以后， 只要没有被读取过， 可以在其他页面访问。

记住：只要被读取一次即失效， 不管是本页面，还是跳转的页面。

* 1. Model - ViewData.Model 是 ViewData的一个非常有用的扩展属性

ViewData.ModelMetadata ViewData.ModelState

ViewData.Model - 通用object 类型，

在 View 里： 访问 @ViewData.Model 或者直接 @Model

如何拆箱unboxing: @model datatype

public ActionResult Index()

{

ViewData.Model = new List<product>

{

new product {id=100, name="Apple" },

new product {id=200, name="Banana" }

};

return View();

}

或者直接传递给 View()

public ActionResult Index()

{

var prodList = new List<product>

{

new product {id=100, name="Apple" },

new product {id=200, name="Banana" }

};

return View(prodList);

}

@using WebApp6.Models

@model IList<product> 注意不能带分号；

<ul class="list-group">

@foreach (product el in Model)

{

<li class="list-group-item">@el.id @el.name</li>

}

</ul>

1. View()

// Summary:

// Creates a System.Web.Mvc.ViewResult object using the view name and master-page

// name that renders a view to the response.

//

// Parameters:

// viewName:

// The name of the view that is rendered to the response.

//

// masterName:

// The name of the master page or template to use when the view is rendered.

//

// Returns:

// The view result.

protected internal ViewResult View();

protected internal ViewResult View(IView view);

protected internal ViewResult View(string viewName);

protected internal ViewResult View(object model);

protected internal virtual ViewResult View(IView view, object model);

protected internal ViewResult View(string viewName, object model);

protected internal ViewResult View(string viewName, string masterName);

protected internal virtual ViewResult View(string viewName, string masterName, object model);

1. Controller: 可以参考Controller 提供了什么属性和方法可以用于访问，如：RouteData, Request, Response, ControllerContext, Session, ViewData(ViewBag) , ModelState
2. Razor 语法：
   1. 命名空间的使用：

@using WebApp4.Controllers

@using WebApp4.Models

@using WebApp4.Controllers; - 可以带分号，也可以不用带分号

@using WebApp4.Models;

@{

using WebApp4.Controllers;

using WebApp4.Models;

}

这样写出错， 必须单独使用 @

* 1. 对于常用的命名空间：可以放在 Views 目录下的 web.config 里.

放在web.config 里，在 cshtml 页面里无法使用 intellisense 的功能

<system.web.webPages.razor>

<host factoryType="System.Web.Mvc.MvcWebRazorHostFactory, System.Web.Mvc,

Version=5.2.3.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" />

<pages pageBaseType="System.Web.Mvc.WebViewPage">

<namespaces>

<add namespace="System.Web.Mvc" />

<add namespace="System.Web.Mvc.Ajax" />

<add namespace="System.Web.Mvc.Html" />

<add namespace="System.Web.Optimization"/>

<add namespace="System.Web.Routing" />

<add namespace="WebApp4" />

<add namespace="WebApp4.Models" />

</namespaces>

</pages>

</system.web.webPages.razor>

如下：无法识别和使用 intellisense

@foreach(product el in (IList<product>)ViewData["prod"] )

{

<li>@el.id @el.name</li><br />

}

所以需要使用

@using WebApp6.Models;

@foreach(product el in (IList<product>)ViewData["prod"] )

{

<li>@el.id @el.name</li><br />

}

* 1. Razor 语法

Now: @DateTime.Now<br />

* + 1. C# 代码片段

@{

var items = new List<string> { "AAA", "BBB", "CCC" };

}

List of String Length: @items.Count() <br />

@{ int count = 0;}

@foreach(string s in items)

{

count++;

<li>Item : @count - @s </li>

}

* + 1. 常见的错误: 避免二义使用 @(C#variable)

@{

string myString = "My String Here";

}

<span style="color:blue;font-size:32px;">My String:@myString.Models</span>

想输出：My String Here.Models 则出错

My String:@(myString).Models 正确的做法是:@(myString)

* + 1. 输出 @ ， 使用两个 @@ 即代表原样输出 @ 字符

This is good:<br /><br />

@william liu @susan su

Error：

Line 55: <br />

Line 56: This is good:<br /><br />

Line 57: @william liu @susan su

This is good:<br /><br />

@@william liu @@susan su

则输出：  
 This is good:  
 @william liu @susan su

* + 1. 电子邮件的处理

@{

string emailDomain = "hotmail";

}

<span style="color:blue;font-size:32px;">My Email: william\_lwh@emailDomain.com</span>

对于电子邮件会自动识别: [william\_lwh@emailDomain.com](mailto:william_lwh@emailDomain.com)

@{

string emailDomain = "hotmail";

}

<span style="color:blue;font-size:32px;">My Email: william\_lwh@@@(emailDomain).com</span>

电子邮件输出: [william\_lwh@hotmail.com](mailto:william_lwh@hotmail.com) @@ 输出一个 @; @(emailDomain) 输出变量

<span style="color:blue;font-size:32px;">Length: [string@emailDomain.Length</span](mailto:string@emailDomain.Length%3c/span)>

会输出成电子邮件：Length: [string@emailDomain.Length](mailto:string@emailDomain.Length)

Length: [string@(emailDomain.Length)](mailto:string@(emailDomain.Length)) 则输出：Length: string7

* + 1. HTML 编码: 默认是自动编码， 如果需要无编码输出则使用 Html.Raw(string)

ViewData["text"] = "<span style='color:red;font-size:32px;'>Good Test Text</span>";

return View();

@ViewData["text"]

输出： <span style='color:red;font-size:32px;'>Good Test Text</span>

HTML: &lt;span style=&#39;color:red;font-size:32px;&#39;&gt;Good Test

Text&lt;/span&gt;

@Html.Raw(ViewData["text"])

输出：Good Test Text

HTML: <span style='color:red;font-size:32px;'>Good Test Text</span>

* + 1. 混合代码和纯文本

@foreach(string s in items)

{

count++;

Hello Item: @count - @s 出错

}

<span>Hello Item: @count - @s</span> - 自动判断，不出错

@{ int count = 0;}

@foreach(string s in items)

{

count++;

<span>Hello Item: @count - @s</span> -

@:How are you??

<text>I am fine , @count thank you </text>

}

在代码块中输出文本：

1. 文本可以是被 HTML tag 包括， 会自动识别变量与文本
2. 输出纯文本可以使用 @：

@:How @count are you?? i don't know item @s

How 1 are you?? i don't know item AAA

这样混合也可以

1. 可以把文本使用<text></text> 包括， 里面可以包含变量和表达式
   * 1. 注释： @\* …. \*@
     2. 泛型方法必须使用括号, 调用无返回值的函数 @()

First Product Name: @( Model.FirstOrDefault<product>().name )<br />

@{ MyFunc.tellme3(10000); }

* + 1. @helper 辅助方法

@helper 辅助方法： 不用回传任何返回值， 只能输出 HTML 片段

@helper ShowPrice(int prc) {

if (prc <= 0)

{

@:Price not allow 0

}

else

{

@:Price is @prc

}

}

Price1: @ShowPrice(100) <br />

Price2: @ShowPrice(-10) <br />

将@helper 辅助方法 共享给所有的 View 使用：

1. 必须放在目录 App­\_Code 下
2. 可以在目录 App­\_Code 下再创建自己的目录
3. 调用方法 ： @目录名.文件名.helper方法名( [params] )

App\_Code\Money\UIJSON.cshtml

@helper ShowJSON(string tag, string a) {

string str = string.Format("<{0}>{1}</{0}>", tag, a);

<text>@str</text>

}

App\_Code\Money\UIMoney.cshtml

@helper ShowPrice(int prc) {

if(prc <=0 )

{

@:Price less than 0 , not allowed @prc

}

else

{

@:Price is good : @prc;

}

}

@helper ShowDate(int hours) {

DateTime dt = DateTime.Now;

dt.AddHours(hours);

<text>Today after @hours is @dt</text>

}

JSON Str: @Money.UIJSON.ShowJSON("HOMER", "SHIFU")<br />

DateTime: @Money.UIMoney.ShowDate(32)<br />

Price1: @Money.UIMoney.ShowPrice(100) <br />

Price2: @Money.UIMoney.ShowPrice(-10) <br />

Price3: @ShowPrice(200) <br /> - 这是View里定义的本地

Price4: @ShowPrice(-20) <br /> - 这是View里定义的本地

* + 1. @functions 自定义函数: 函数是纯 C# 语法， 不能混合Razor 语法

@functions {

public int add(int a, int b)

{

return a + b;

}

public string showString(string a)

{

return "<span style='color:red;font-size:32px'>" + a + "</span>"; - 会被 Encode

}

public IHtmlString showHTML(string a)

{

string ret = "<span style='color:red;font-size:32px'>" + a + "</span>";

return new HtmlString(ret);

}

public void showMe(string a)

{

string ret = "this is string:" + a;

}

}

Functions:<br />

Add: @add(100, 235)<br />

String: @showString("Hello World")<br />

HTML: @showHTML("Hello World")<br />

Show: @{ showMe("Hello World"); }<br />

输出：

Functions:  
Add: 335  
String: <span style='color:red;font-size:32px'>Hello World</span>  
HTML: Hello World  
Show:

@functions 共享给所有的View

1. 必须放在目录 App­\_Code 下
2. 可以在目录 App­\_Code 下再创建自己的目录
3. @functions 里面的方法必须是 静态方法 : public static funcName( .. ) { c# code }
4. @functions 里的方法必须是纯 C# 代码， 不能混合Razor 语法
5. 如果要输出 HTML , 返回 IhtmlString return new HtmlString(ret);
6. 如果要输出HTML, 也可以返回MvcHtmlString, return new MvcHtmlString(html);
7. 调用方法 ： @目录名.文件名. function方法名( [params] )

App\_Code\Money\MyFunc.cshtml

@functions {

public static int substract(int a, int b)

{

return a - b;

}

}

App\_Code\Money\UIJSON.cshtml

可以把 @helper @functions 放在一个文件里

@helper ShowJSON(string tag, string a) {

string str = string.Format("<{0}>{1}</{0}>", tag, a);

<text>@str</text>

}

@functions {

public static int subto(int a, int b)

{

return a - b;

}

}

App\_Code\Money\UIMoney.cshtml

@helper ShowDate(int hours) {

DateTime dt = DateTime.Now;

dt.AddHours(hours);

<text>Today after @hours is @dt</text>

}

@functions {

public static IHtmlString addPrice(int a)

{

string ret = string.Format("<span style='color:blue;font-size:2em;'>Price:{0} + Tax 12% = {1}</span>", a, a \* 1.12);

return new HtmlString(ret);

}

}

Functions:<br />

Date: @Money.UIMoney.ShowDate(42)<br />

Substract: @Money.MyFunc.substract(100, 32)<br />

price: @Money.UIMoney.addPrice(200)<br />

Subto: @Money.UIJSON.subto(200, 135)<br />

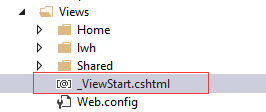
<br />

输出：

Functions:  
Date: Today after 42 is 7/26/2016 8:04:02 PM   
Substract: 68  
price: Price:200 + Tax 12% = 224  
Subto: 65

1. Layout 布局
   1. 全局布局 : 对所有的 View

在 ~/Views/\_ViewStart.cshtml 文件



默认情况下， 任何 View 页面载入之前都会先载入执行这个文件：

这个文件的内容如下：设置主版布局为默认的 Shared 目录下的 \_Layout.cshtml

@{

Layout = "~/Views/Shared/\_Layout.cshtml";

}

默认的主版布局 \_Layout.cshtml 文件内容如下：

它是一个HTML 框架结构：如 head , 载入CSS, body , @RenderBody()

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>@ViewBag.Title - My ASP.NET Application</title>

@Styles.Render("~/Content/css")

@Scripts.Render("~/bundles/modernizr")

<script type="text/javascript">

var appInsights=window.appInsights||function(config){

function r(config){t[config]=function(){var i=arguments;t.queue.push(function(){t[config].apply(t,i)})}}var t={config:config},u=document,e=window,o="script",s=u.createElement(o),i,f;for(s.src=config.url||"//az416426.vo.msecnd.net/scripts/a/ai.0.js",u.getElementsByTagName(o)[0].parentNode.appendChild(s),t.cookie=u.cookie,t.queue=[],i=["Event","Exception","Metric","PageView","Trace","Ajax"];i.length;)r("track"+i.pop());return r("setAuthenticatedUserContext"),r("clearAuthenticatedUserContext"),config.disableExceptionTracking||(i="onerror",r("\_"+i),f=e[i],e[i]=function(config,r,u,e,o){var s=f&&f(config,r,u,e,o);return s!==!0&&t["\_"+i](config,r,u,e,o),s}),t

}({

instrumentationKey:"25411d81-3b44-4b90-9058-83b389bc514e"

});

window.appInsights=appInsights;

appInsights.trackPageView();

</script>

</head>

<body>

<div class="navbar navbar-inverse navbar-fixed-top">

<div class="container">

<div class="navbar-header">

<button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

@Html.ActionLink("Application name", "Index", "Home", new { area = "" }, new { @class = "navbar-brand" })

</div>

<div class="navbar-collapse collapse">

<ul class="nav navbar-nav">

<li>@Html.ActionLink("Home", "Index", "Home")</li>

<li>@Html.ActionLink("About", "About", "Home")</li>

<li>@Html.ActionLink("Contact", "Contact", "Home")</li>

</ul>

</div>

</div>

</div>

<div class="container body-content">

@RenderBody()

<hr />

<footer>

<p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>

</footer>

</div>

@Scripts.Render("~/bundles/jquery")

@Scripts.Render("~/bundles/bootstrap")

@RenderSection("scripts", required: false)

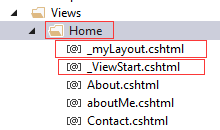
</body>

</html>

* 1. 局部Layout : 对该Controller下的所有 View

任何与Controller同名的Views子目录下也能出现同样的 \_ViewStart.cshtml 文件

这样就可以让不同的Controller预设载入不同的主版页面



@{

Layout = "\_myLayout.cshtml";

}

* 1. View页面指定 Layout: 只对该 View 页面有效

@{

ViewBag.Title = "View";

Layout = "~/Views/lwh/\_myLay1.cshtml";

}

Layout = “xxxx”; 可以写在页面的任何位置，不一定是开头。这和页面执行的顺序有关， 和赋值无关。

* 1. View 页面的执行顺序：

1. 全局\_ViewStart： ~/Views/\_ViewStart.cshtml
2. 局部\_ViewStart: ~/Views/Home/\_ViewStart.cshtml
3. View 页面： 执行完该页面， 再去查看 Layout 的值
4. 如果在 View 页面中，设置了 Layout , 则覆盖前面\_ViewStart里设置
5. 执行 Layout 主版页面， 并通过 @RenderBody()合成， 一起输出HTML
6. View 的执行全程提供了 ViewContext 对象来访问跟View 有关的全局变量，包括：RouteData, Request, ViewData(ViewBag) … s

基于上面的执行顺序， 变量与HTML 的传递 :

1. 全局\_ViewStart 与 局部\_ViewStart 里定义的局部变量是彼此不能访问的。

全局\_ViewStart:

@{

Layout = "~/Views/Shared/\_Layout.cshtml";

string gdata = "This is Global View Data: ";

}

hello this gdata: @gdata<br />

局部\_ViewStart :

@{

Layout = "\_myLayout.cshtml";

string ldata = "this is controller data: " + gdata; - 不能访问

}

This is Controller ViewStart: @ldata<br />

1. 全局\_ViewStart 与 局部\_ViewStart 里输出的HTML 将是View 页面内容的一部分， 而且是最先输出。 仍然属于 @RenderBody()里的内容. 虽然可以这么构建HTML , 但是我们不能这么做，因为违反关注点分离原则。
2. 全局\_ViewStart 与 局部\_ViewStart 里都可以使用 ViewData, ViewBag 等数据

主要通过 ViewContext.ViewData, ViewContext.ViewBag 来访问, 不能直接使用

ViewData , ViewBag, 只能通过 ViewContext

@using WebApp6.Models;

@{

Layout = "~/Views/Shared/\_Layout.cshtml";

string gdata = "This is Global View Data:";

}

hello this gdata: @gdata<br />

Global ViewStart Product List:<br />

<ul>

@foreach(product el in (List<product>)ViewContext.ViewData.Model) {

<li>@el.id -:- @el.name</li>

}

</ul>

@using WebApp6.Models;

@{

Layout = "\_myLayout.cshtml";

string ldata = "this is controller data: ";

}

This is Controller ViewStart: @ldata<br />

Controller ViewStart Product List:<br />

<ul>

@foreach (product el in (List<product>)ViewContext.ViewData.Model)

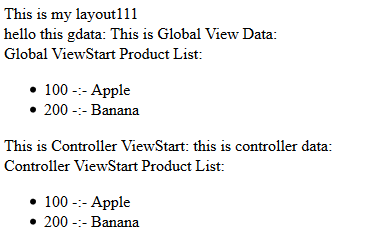
{

<li>@el.id -:- @el.name</li>

}

</ul>

输出到页面：



1. 全局ViewStart 和 局部ViewStart 里可以更改，添加数据到 ViewData, ViewBag

ViewContext.ViewData["gdata"] = "Global ViewStart Data Test";

ViewContext.ViewBag.gdata1 = "Global ViewBag";

ViewContext.ViewData["ldata"] = "Controller ViewStart Data Test";

ViewContext.ViewBag.gdata1 = "Local ViewBag OverWrite global";

输出：

Global Data: Global ViewStart Data Test   
Local Data: Controller ViewStart Data Test   
Global Bag: Local ViewBag OverWrite global

Global ViewStart:

((List<product>)ViewContext.ViewData.Model).Add(

new product { id = 300, name = "Orange" }

);

Controller ViewStart:

((List<product>)ViewContext.ViewData.Model).Add(

new product { id = 400, name = "Grape"}

);

Global ViewStart Product List:

* 100 -:- Apple
* 200 -:- Banana
* 300 -:- Orange

This is Controller ViewStart: this is controller data:   
Controller ViewStart Product List:

* 100 -:- Apple
* 200 -:- Banana
* 300 -:- Orange
* 400 -:- Grape
  1. View 页面与Layout 页面的执行顺序： 先执行完 View 页面以后， 再根据Layout变量的值，去执行 Layout 页面， 通过@RenderBody() 把View的内容混合输出。

基于以上的执行顺序： View页面修改的值， 会影响到Layout页面可见。Layout页面的修改不会影响到View页面.

View 页面：

@model List<product>

@{

Model.Add(new product { id = 800, name = "Blue Berry" });

ViewData["getMeData"] = "GetMe Test Data";

}

Layout 页面：

@model List<WebApp6.Models.product>

@{

Model.Add(new product { id=600, name="Plum" });

ViewData["layData"] = "This is layout data";

}

输出结果：



* 1. Layout 主版页面通常是共用于一组 View 页面， 所以不应该使用View页面所接收到的数据，也不应该使用这些接收的数据来逻辑构建主版页面。 因为各个View页面所接收到的数据是变化多端的。

* 1. Layout 主版页面 @RenderBody() 必须有且只有一个
  2. Layout 主版页面@RenderSection 可以没有或者有一个以上

Layout 页面

@model List<WebApp6.Models.product>

@{

Model.Add(new product { id=600, name="Plum" });

ViewData["layData"] = "This is layout data";

}

<!DOCTYPE html>

<html>

<head>

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

<title>@ViewBag.Title</title>

@**RenderSection**("js\_section")

</head>

<body>

<br />

<span style="color:red;font-size:2em;">Layout 页面</span><br />

@**RenderSection**("wait\_div")

This is View 页面：<br />

<div>

@**RenderBody**()

</div>

@**RenderSection**("load\_com")

mylay11 layout data: @ViewData["layData"] <br />

mylay11 getMe data: @ViewData["getMeData"]

</body>

</html>

<html>

<body>

@RenderBody()

@RenderSection("scripts", required: false)

</body>

</html>

@section scripts {

<script type="text/javascript">alert('hello');</script>

}

View 页面:

@model List<product>

@{

Layout = "~/Views/lwh/\_myLay1.cshtml";

Model.Add(new product { id = 800, name = "Blue Berry" });

ViewData["getMeData"] = "GetMe Test Data";

}

<div>

@Html.ActionLink("Back to List", "Index")

</div>

<span style="color:red;font-size:2em;">getMe.cshtml View 页面</span>

<br />

View Page product list:<br />

<ul id="productlist">

@foreach(product el in Model)

{

<li>@el.id -\*- @el.name</li>

}

</ul>

@section js\_section {

<script type="text/javascript"

src="~/jquery/myplugin/jquery.lwh.common.js"></script>

<link type="text/css"

href="~/jquery/myplugin/css/light/jquery.lwh.loading.css"

rel="stylesheet" />

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

alert("document is ready; product list folowing");

</script>

}

@section wait\_div {

<div style="display:block; width:300px; height:300px;

background-color:blue;">

This is wait div

</div><br />

}

@section load\_com {

<div style="display:block; width:300px; height:300px;

background-color:orange;">

This is Loading Zone

</div><br />

}

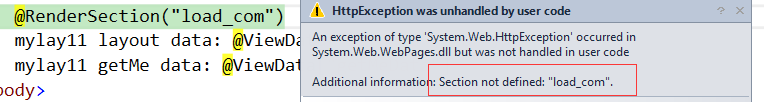
<script>

alert("getMe is ready; product list folowing");

</script>



@**RenderSection**("load\_com") 在View页面没有定义，则会出错：



如何解决：

* 1. required : false 即不是必须

@RenderSection("load\_com", required: false)

* 1. 也可以通过判断来输出: 如果View有定义则使用，没有使用主板定义

应用场合： 先子后父

@if (IsSectionDefined("load\_com"))

{

@RenderSection("load\_com");

}

else

{

@:You can define default one in the layout page!

}

1. 模型绑定Model Binding

<form action="#" method="get">

user: <input type="text" name="user" value="William Liu" />

<br />

name: <input type="text" name="uname" value="U1111" />

<input type="text" name="uname" value="U2222" />

<br />

Lang: <input type="checkbox" name="lang" value="English" />English

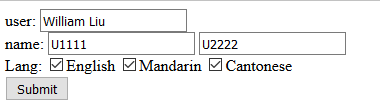
<input type="checkbox" name="lang" value="Mandarin" />Mandarin

<input type="checkbox" name="lang" value="Cantonese" />Cantonese

<br />

<input type="submit" value="Submit" />

</form>



如果get 的URL:

http://www.dev.com/test.html?user=William+Liu&uname=U1111&uname=U2222&lang=English&lang=Mandarin&lang=Cantonese#

<form method="post" action="/Home/About">

<div class="row">

<div class="col-md-1">User</div>

<div class="col-md-2">

<input type="text" name="user\_name" value="@TempData["User"]" />

</div>

<div class="col-md-1 col-lg-offset-1">Spouse</div>

<div class="col-md-2">

<input type="text" name="spouse\_name" value="@TempData["Spouse"]" />

<input type="text" name="spouse\_name" value="@TempData["Spouse"]" />

</div>

</div>

<div class="row">

<div class="col-md-1">Birth</div>

<div class="col-md-2">

<input type="date" required name="birthdate" value="@TempData["bdate"]" />

</div>

<div class="col-md-1 col-lg-offset-1">Salary</div>

<div class="col-md-2">

<input type="number" name="salary" value="@TempData["income"]" />

</div>

</div>

<input type="submit" name="btnSubmit" class="btn btn-info" value="Submit Me" />

</form>

我们在表单里定义了 4 个输入元素：user\_name, spouse\_name, birthdate, salary

* 1. 传统表单模式

public ActionResult About()

{

TempData["Params"] = Request.Form.Count;

TempData["User"] = Request.Form["user\_name"];

TempData["Spouse"] = Request.Form["spouse\_name"];

TempData["bdate"] = Request.Form["birthdate"];

TempData["income"] = Request.Form["salary"];

ViewData["Params"] = Request.Form.Count;

ViewData["User"] = Request.Form["user\_name"];

ViewData["Spouse"] = Request.Form["spouse\_name"];

ViewData["bdate"] = Request.Form["birthdate"];

ViewData["income"] = Request.Form["salary"];

ViewData["language"] = Request.Form["language"];

ViewData["btn"] = Request.Form["btnSubmit"];

return View();

}

<div class="row">

<div class="col-md-4">Params: @ViewData["Params"]</div>

</div>

<div class="row">

<div class="col-md-2">User</div>

<div class="col-md-2">@ViewData["User"]</div>

<div class="col-md-2">Spouse</div>

<div class="col-md-2">@ViewData["Spouse"]</div>

</div>

<div class="row">

<div class="col-md-2">BirthDate</div>

<div class="col-md-2">@ViewData["bdate"]</div>

<div class="col-md-2">Salary</div>

<div class="col-md-2">@ViewData["income"]</div>

</div>

<div class="row">

<div class="col-md-2">Language</div>

<div class="col-md-6">@ViewData["language"]</div>

</div>

<div class="row">

<div class="col-md-2">Submit</div>

<div class="col-md-2">@ViewData["btn"]</div>

</div>

通过：Request.Form 来获取 Form Post 的值：

* + 1. Input输入元素必须要设置name=“key” 只有 name属性的元素才会被传递出去
    2. 按钮button 或者 submit， 只有点击才会生成 name = value 键值对
    3. 对于重名元素的值， 是使用逗号, 分隔 如： checkbox, text(多个同名)



也可以通过FormCollection(继承自NameValueCollection)

public ActionResult About(FormCollection pform, string btnSubmit)

{

TempData["Params"] = pform.Count;

TempData["User"] = pform["user\_name"];

TempData["Spouse"] = pform["spouse\_name"];

TempData["bdate"] = pform["birthdate"];

TempData["income"] = pform["salary"];

ViewData["Params"] = pform.Count;

ViewData["User"] = pform["user\_name"];

ViewData["Spouse"] = pform["spouse\_name"];

ViewData["bdate"] = pform["birthdate"];

ViewData["income"] = pform["salary"];

ViewData["language"] = pform["language"];

ViewData["btn"] = pform["btnSubmit"];

ViewData["btn1"] = btnSubmit;

return View();

}

* 1. 简单模型绑定

public ActionResult About(

string user\_name,

string spouse\_name,

DateTime birthdate,

decimal salary,

string language,

string btnSubmit)

{

TempData["Params"] = 0;

TempData["User"] = user\_name;

TempData["Spouse"] = spouse\_name;

TempData["bdate"] = birthdate;

TempData["income"] = salary;

ViewData["Params"] = 0;

ViewData["User"] = user\_name;

ViewData["Spouse"] = spouse\_name;

ViewData["bdate"] = birthdate;

ViewData["income"] = salary;

ViewData["language"] = language;

ViewData["btn"] = Request.Form["btnSubmit"];

ViewData["btn1"] = btnSubmit;

return View();

}

注意对于简单绑定：

1. 对于重名的元素checkbox, 只提取被选择的第一值
2. 或者多个重名text, 只输出第一个元素的值， 不管元素值是否为空， 为空则输出为空
3. 如果想正确获取重名元素的所有值。 可以使用数组 c# dateType[]

public ActionResult About(

string user\_name,

string[] spouse\_name,

DateTime birthdate,

decimal salary,

string[] language,

string btnSubmit)

ViewData["Spouse"] = spouse\_name;

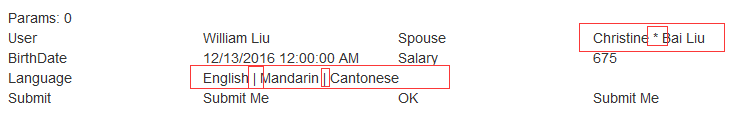
ViewData["language"] = language;

<div class="col-md-2">Spouse</div>

<div class="col-md-2">@string.Join(" \* ", (string[])ViewData["Spouse"])</div>

<div class="col-md-2">Language</div>

<div class="col-md-6">@string.Join(" | ", (string[])ViewData["language"])</div>



在 Action方法上的参数名称和Form里输入元素的名称一致即可。

会通过 DefaultModelBinder类别将 Form 或者 QueryString 传来的数据进行处理,

* 1. 复杂模型绑定

可以通过DefaultModelBinder将Form的输入元素映射到非常复杂的 .Net 类别

称之为“复杂模型”或者“模型”.该模型可能是一个List<T> 或者是包含多个属

性的自定义类型。

1. Form

<form method="post" name="frm" action="/Home/About">

<div class="row">

<div class="col-md-1">User</div>

<div class="col-md-2"> - **注意 user\_name 大小写是不同的**

<input type="text" name="user\_Name" value="@TempData["User"]" />

<input type="text" name="User\_name" value="@TempData["User"]" />

<input type="text" name="User\_Name" value="@TempData["User"]" />

</div>

<div class="col-md-1 col-lg-offset-1">Spouse</div>

<div class="col-md-2">

<input type="text" name="spouse\_name" value="@TempData["Spouse"]" />

<input type="text" name="spouse\_name" value="@TempData["Spouse"]" />

</div>

</div>

<div class="row">

<div class="col-md-1">Birth</div>

<div class="col-md-2">

<input type="date" required name="birthdate" value="@TempData["bdate"]" />

</div>

<div class="col-md-1 col-lg-offset-1">Salary</div>

<div class="col-md-2">

<input type="number" name="salary" value="@TempData["income"]" />

</div>

</div>

<div class="row">

<div class="col-md-1">Language</div>

<div class="col-md-1">

<input type="checkbox" name="language" value="English" /> English

</div>

<div class="col-md-1">

<input type="checkbox" name="language" value="Mandarin" /> Mandarin

</div>

<div class="col-md-1">

<input type="checkbox" name="language" value="Cantonese" /> Cantonese

</div>

</div>

<input type="submit" name="btnSubmit" class="btn btn-info" value="Submit Me" />

</form>

1. 定义 .Net 类型

public class User

{

public string[] USER\_NAME { get; set; }

public string[] spouse\_name { get; set; }

public DateTime birthdate { get; set; }

public decimal salary { get; set; }

public string[] language { get; set; }

}

注意：

1. 对于象checkbox, text ( 多个同名) 可以使用数组来接收传来的值
2. 大小写不敏感，from : user\_name 可以映射 C# Obj.USER\_NAME
3. Action 方法上的参数指定类型

public ActionResult About(User postUser)

{

return View(postUser);

}

可以通过DefaultModelBinder 会将Form传进来的 KeyValueCollection，

来和User 的所有属性匹配。 所以名字约定很重要，必须同名

1. 在View 里输出传进来的User，是放在 ViewData.Model里

@using WebApp7.Models

@model User - 把 Model 强类型化，可以享受Intellisense的方便

@{

ViewBag.Title = "About";

}

<h2>@ViewBag.Title.</h2>

<h3>@ViewBag.Message</h3>

<p>Use this area to provide additional information.</p>

<br />

<div class="row">

<div class="col-md-2">User</div>

<div class="col-md-2">@**string.Join**(" \* ", Model.USER\_NAME)</div>

<div class="col-md-2">Spouse</div>

<div class="col-md-2">@**string.Join**(" \* ", Model.spouse\_name)</div>

</div>

<div class="row">

<div class="col-md-2">BirthDate</div>

<div class="col-md-2">@Model.birthdate</div>

<div class="col-md-2">Salary</div>

<div class="col-md-2">@Model.salary</div>

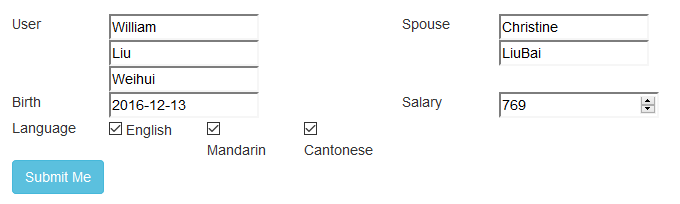
</div>

<div class="row">

<div class="col-md-2">Language</div>

<div class="col-md-6">@**string.Join**(" | ", Model.language)</div>

</div>





* 1. 多个复杂模型绑定

<form method="post" name="frm" action="/Home/About">

<div class="row">

<div class="col-md-1">User</div>

<div class="col-md-2">

<input type="text" name="user\_Name" value="@TempData["User"]" />

<input type="text" name="User\_name" value="@TempData["User"]" />

<input type="text" name="User\_Name" value="@TempData["User"]" />

</div>

<div class="col-md-1 col-lg-offset-1">Spouse</div>

<div class="col-md-2">

<input type="text" name="spouse\_name" value="@TempData["Spouse"]" />

<input type="text" name="spouse\_name" value="@TempData["Spouse"]" />

</div>

</div>

<div class="row">

<div class="col-md-1">Birth</div>

<div class="col-md-2">

<input type="date" required name="birthdate" value="@TempData["bdate"]" />

</div>

<div class="col-md-1 col-lg-offset-1">Salary</div>

<div class="col-md-2">

<input type="number" name="salary" value="@TempData["income"]" />

</div>

</div>

<div class="row">

<div class="col-md-1">Language</div>

<div class="col-md-1">

<input type="checkbox" name="language" value="English" /> English

</div>

<div class="col-md-1">

<input type="checkbox" name="language" value="Mandarin" /> Mandarin

</div>

<div class="col-md-1">

<input type="checkbox" name="language" value="Cantonese" /> Cantonese

</div>

</div>

<div class="row">

<div class="col-md-1">Suite</div>

<div class="col-md-1">

<input type="text" name="suite" value="504" />

</div>

<div class="col-md-1">

<input type="text" name="street" value="6540 Burlinton Ave." />

</div>

<div class="col-md-1">

<input type="text" name="city" value="Burnaby" />

</div>

</div>

<input type="submit" name="btnSubmit" class="btn btn-info" value="Submit Me" />

</form>

分成两部分： User 和 Address

public class User

{

public string[] USER\_NAME { get; set; }

public string[] spouse\_name { get; set; }

public DateTime birthdate { get; set; }

public decimal salary { get; set; }

public string[] language { get; set; }

}

public class Address

{

public int Suite { get; set; }

public string Street { get; set; }

public string City { get; set; }

}

}

Action 方法上：

public ActionResult About(User postUser, Address paddr)

{

ViewBag.Address = paddr;

return View(postUser);

}

可以通过DefaultModelBinder 会将Form传进来的 KeyValueCollection,

来和User , Address 来匹配映射

由于 ViewData.Model 只有一个， 所以要借助 ViewBag 来传递 Address 到View

@using WebApp7.Models

@model User

@{

Address **Addr** = ViewBag.Address;

ViewBag.Title = "About";

}

<h2>@ViewBag.Title.</h2>

<h3>@ViewBag.Message</h3>

<p>Use this area to provide additional information.</p>

<br />

<div class="row">

<div class="col-md-2">User</div>

<div class="col-md-2">@string.Join(" \* ", Model.USER\_NAME)</div>

<div class="col-md-2">Spouse</div>

<div class="col-md-2">@string.Join(" \* ", Model.spouse\_name)</div>

</div>

<div class="row">

<div class="col-md-2">BirthDate</div>

<div class="col-md-2">@Model.birthdate</div>

<div class="col-md-2">Salary</div>

<div class="col-md-2">@Model.salary</div>

</div>

<div class="row">

<div class="col-md-2">Language</div>

<div class="col-md-6">@string.Join(" | ", Model.language)</div>

</div>

<div class="row">

<div class="col-md-2">Address</div>

<div class="col-md-6">

@string.Join(", ", **Addr**.Suite, **Addr**.Street, **Addr**.City )

</div>

</div>



* 1. 绑定到同一个类型的多个模型
     1. 同样的内容， 有多个Form, 或者有多个副本， 不一定是多个Form

<form method="post" name="frm" action="/Home/About">

@\* Form 1 content \*@

<div class="row">

<div class="col-md-1">User</div>

<div class="col-md-2">

<input type="text" name="**form1**.user\_Name" value="@TempData["User"]" />

<input type="text" name="**form1**.User\_name" value="@TempData["User"]" />

<input type="text" name="**form1**.User\_Name" value="@TempData["User"]" />

</div>

<div class="col-md-1 col-lg-offset-1">Spouse</div>

<div class="col-md-2">

<input type="text" name="**form1**.spouse\_name" value="@TempData["Spouse"]" />

<input type="text" name="**form1**.spouse\_name" value="@TempData["Spouse"]" />

</div>

</div>

<div class="row">

<div class="col-md-1">Birth</div>

<div class="col-md-2">

<input type="date" required name="**form1**.birthdate" value="@TempData["bdate"]" />

</div>

<div class="col-md-1 col-lg-offset-1">Salary</div>

<div class="col-md-2">

<input type="number" name="**form1**.salary" value="@TempData["income"]" />

</div>

</div>

<div class="row">

<div class="col-md-1">Language</div>

<div class="col-md-1">

<input type="checkbox" name="**form1**.language" value="English" /> English

</div>

<div class="col-md-1">

<input type="checkbox" name="**form1**.language" value="Mandarin" /> Mandarin

</div>

<div class="col-md-1">

<input type="checkbox" name="**form1**.language" value="Cantonese" /> Cantonese

</div>

</div>

@\* Form 2 content \*@

<div class="row">

<div class="col-md-1">User</div>

<div class="col-md-2">

<input type="text" name="**form2**.user\_Name" value="@TempData["User"]" />

<input type="text" name="**form2**.User\_name" value="@TempData["User"]" />

<input type="text" name="**form2**.User\_Name" value="@TempData["User"]" />

</div>

<div class="col-md-1 col-lg-offset-1">Spouse</div>

<div class="col-md-2">

<input type="text" name="**form2**.spouse\_name" value="@TempData["Spouse"]" />

<input type="text" name="**form2**.spouse\_name" value="@TempData["Spouse"]" />

</div>

</div>

<div class="row">

<div class="col-md-1">Birth</div>

<div class="col-md-2">

<input type="date" required name="**form2**.birthdate" value="@TempData["bdate"]" />

</div>

<div class="col-md-1 col-lg-offset-1">Salary</div>

<div class="col-md-2">

<input type="number" name="**form2**.salary" value="@TempData["income"]" />

</div>

</div>

<div class="row">

<div class="col-md-1">Language</div>

<div class="col-md-1">

<input type="checkbox" name="**form2**.language" value="English" /> English

</div>

<div class="col-md-1">

<input type="checkbox" name="**form2**.language" value="Mandarin" /> Mandarin

</div>

<div class="col-md-1">

<input type="checkbox" name="**form2**.language" value="Cantonese" /> Cantonese

</div>

</div>

@\* Address content \*@

<div class="row">

<div class="col-md-1">Suite</div>

<div class="col-md-1">

<input type="text" name="suite" value="504" />

</div>

<div class="col-md-1">

<input type="text" name="street" value="6540 Burlinton Ave." />

</div>

<div class="col-md-1">

<input type="text" name="city" value="Burnaby" />

</div>

</div>

<input type="submit" name="btnSubmit" class="btn btn-info" value="Submit Me" />

</form>

有两组一样名称的输入元素， 我们只能把名称使用前缀加以区别：

如：

<input type="text" name="**form1**.user\_Name" value="" />

<input type="text" name="**form2**.user\_Name" value="" />

* + 1. Action 方法上的参数

public ActionResult About(User form1, User form2, Address paddr)

{

ViewBag.User = form1;

ViewBag.Address = paddr;

return View(form2);

}

通过命名约定， 分别使用三个模型来接收传来的表单数据：

User form1, User form2, Address paddr

* + 1. 输出到 View页面

@using WebApp7.Models

@model User

@{

Address **Addr** = ViewBag.Address;

User **tuser** = ViewBag.User;

ViewBag.Title = "About";

}

<h2>@ViewBag.Title.</h2>

<h3>@ViewBag.Message</h3>

<p>Use this area to provide additional information.</p>

<br />

<div class="row">

<div class="col-md-2">User</div>

<div class="col-md-2">@string.Join(" \* ", Model.USER\_NAME)</div>

<div class="col-md-2">Spouse</div>

<div class="col-md-2">@string.Join(" \* ", Model.spouse\_name)</div>

</div>

<div class="row">

<div class="col-md-2">BirthDate</div>

<div class="col-md-2">@Model.birthdate</div>

<div class="col-md-2">Salary</div>

<div class="col-md-2">@Model.salary</div>

</div>

<div class="row">

<div class="col-md-2">Language</div>

<div class="col-md-6">@string.Join(" | ", Model.language)</div>

</div>

<div class="row">

<div class="col-md-2">User</div>

<div class="col-md-2">@string.Join(" \* ", **tuser**.USER\_NAME)</div>

<div class="col-md-2">Spouse</div>

<div class="col-md-2">@string.Join(" \* ", **tuser**.spouse\_name)</div>

</div>

<div class="row">

<div class="col-md-2">BirthDate</div>

<div class="col-md-2">@**tuser**.birthdate</div>

<div class="col-md-2">Salary</div>

<div class="col-md-2">@**tuser**.salary</div>

</div>

<div class="row">

<div class="col-md-2">Language</div>

<div class="col-md-6">@string.Join(" | ", **tuser**.language)</div>

</div>

<div class="row">

<div class="col-md-2">Address</div>

<div class="col-md-6">@string.Join(", ", **Addr**.Suite, **Addr**.Street, **Addr**.City )</div>

</div>

* 1. 显式模型绑定

当Action 操作方法有指定参数，模型绑定会隐式地工作。 也可以使用控制器中的 UpdateModel 和 TryUpdateModel方法来显式地调用模型绑定

public ActionResult About() - 不带任何参数

{

User form1 = new User();

UpdateModel(form1,"form1"); - 必须指定前缀

User form2 = new User();

UpdateModel(form2, "form2"); - 必须指定前缀

Address paddr = new Address();

UpdateModel(paddr);

ViewBag.User = form1;

ViewBag.Address = paddr;

return View(form2);

}

可以带参数或者部分参数：

public ActionResult About(User form1, User form2)

{

UpdateModel(form1,"form1");

UpdateModel(form2, "form2");

Address paddr = new Address();

UpdateModel(paddr);

ViewBag.User = form1;

ViewBag.Address = paddr;

return View(form2);

}

对简单类型不起作用？

String: <input type="text" name="test\_string" value="Hello World" />

string test\_string = string.Empty;

UpdateModel(test\_string);

ViewBag.test = test\_string;

使用UpdateModel() 如果不成功(如数据验证不成功)， 会抛出异常.

使用try {} catch {}

public ActionResult About(User form1, User form2)

{

Address paddr = new Address();

try

{

UpdateModel(form1, "form1");

form1.spouse\_name += " Valid";

UpdateModel(form2, "form2");

form2.spouse\_name += " Valid";

UpdateModel(paddr);

paddr.City += " Valid";

}

catch

{

form1.spouse\_name += " InValid";

form2.spouse\_name += " InValid";

paddr.City += " InValid";

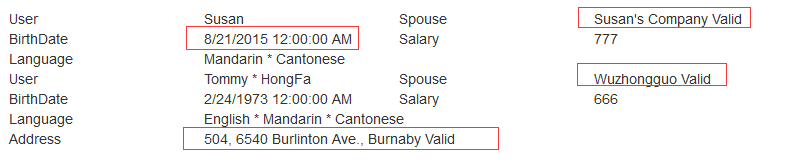
}

ViewBag.User = form1;

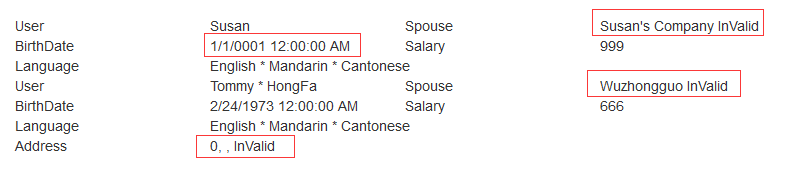
ViewBag.Address = paddr;

return View(form2);

}



只是把日期输入错误，并没有影响到其他的， 我们发现并没有全盘否定



我们调整一下顺序：

public ActionResult About(User form1, User form2)

{

Address paddr = new Address();

try

{

UpdateModel(paddr); - Address First

paddr.City += " Valid";

UpdateModel(form1, "form1");

form1.spouse\_name += " Valid";

UpdateModel(form2, "form2");

form2.spouse\_name += " Valid";

}

catch

{

form1.spouse\_name += " InValid";

form2.spouse\_name += " InValid";

paddr.City += " InValid";

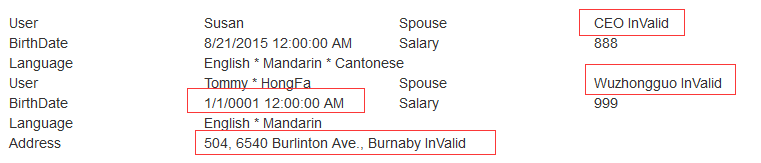
}

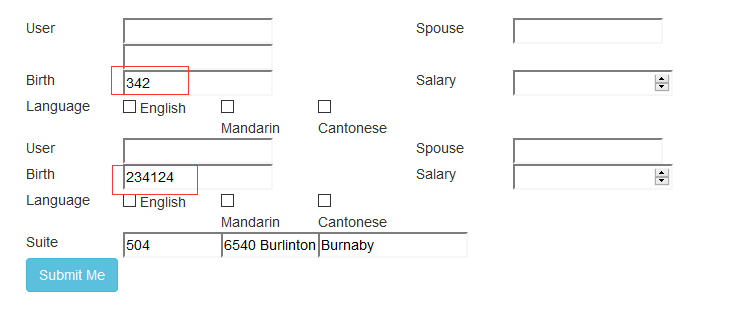
ViewBag.User = form1;

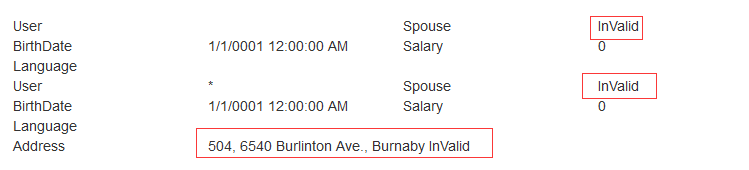
ViewBag.Address = paddr;

return View(form2);

}







public ActionResult About(User form1, User form2)

{

Address paddr = new Address();

try

{

UpdateModel(paddr);

if(ModelState.IsValid)

paddr.City += " Valid";

else

paddr.City += " Update InValid";

UpdateModel(form1, "form1");

if (ModelState.IsValid)

form1.spouse\_name += " Valid";

else

form1.spouse\_name += " Update InValid";

UpdateModel(form2, "form2");

if (ModelState.IsValid)

form2.spouse\_name += " Valid";

else

form2.spouse\_name += " Update InValid";

}

catch

{

form1.spouse\_name += " InValid";

form2.spouse\_name += " InValid";

paddr.City += " InValid";

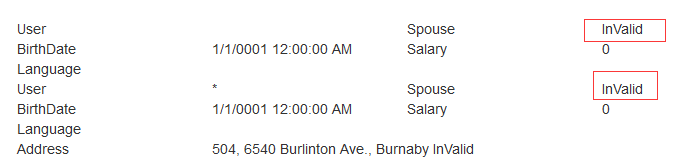
}

ViewBag.User = form1;

ViewBag.Address = paddr;

return View(form2);

}



if (ModelState.IsValid) 判断无效， 并不被执行

总结：UpdateModel() 很奇怪， 是否是全部映射完以后再抛出错误。 而且 ModelState.IsValid 是全局判断，是在Model传入Action操作前就已经做好验证和判断，哪个是无效和错误的。而不是在执行UpdateModel() 那一刻在判断

使用TryUpdateModel()

public ActionResult About(User form1, User form2)

{

if (TryUpdateModel(form1, "form1"))

{

form1.spouse\_name += " Valid";

} else

{

form1.spouse\_name += " InValid";

}

if (TryUpdateModel(form2, "form2"))

{

form2.spouse\_name += " Valid";

}

else

{

form2.spouse\_name += " InValid";

}

Address paddr = new Address();

if (TryUpdateModel(paddr))

{

paddr.City += " Valid";

}

else

{

paddr.City += " InValid";

}

ViewBag.User = form1;

ViewBag.Address = paddr;

return View(form2);

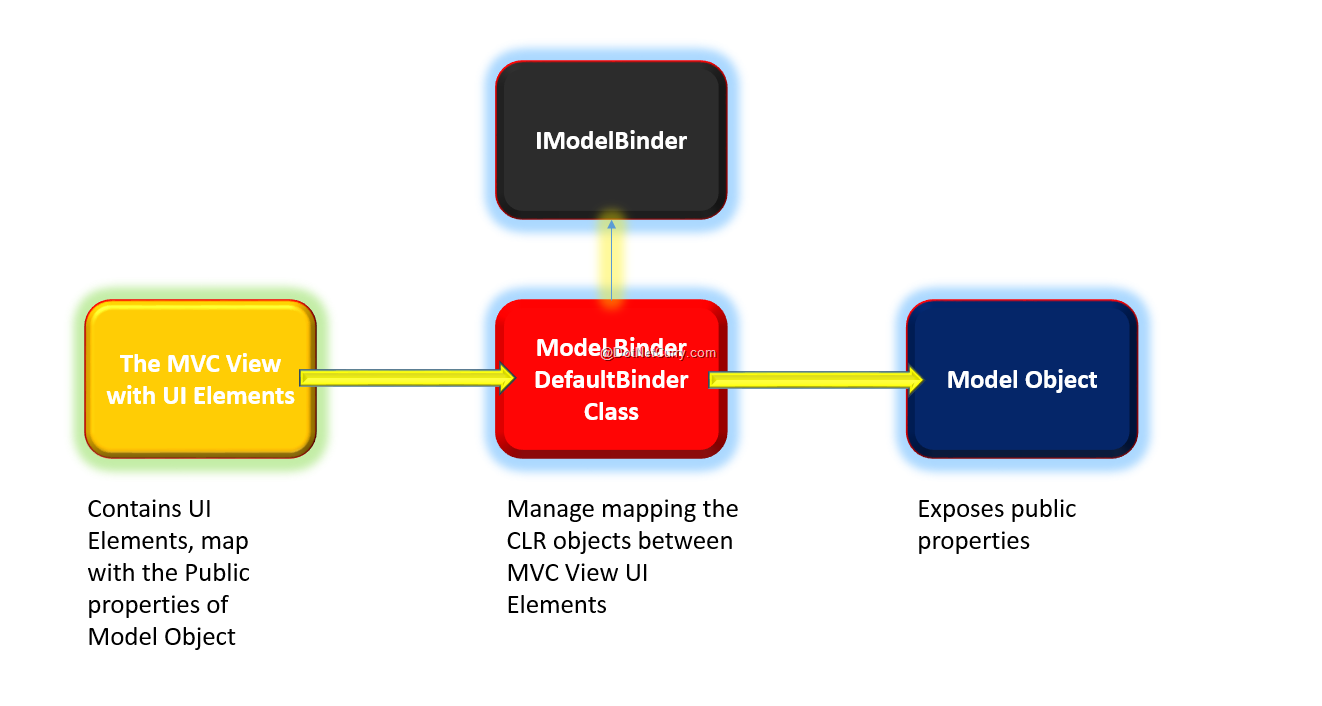
}

TryUpdateModel() – 返回True/False 不抛出异常

成功与否不取决于某个 TryUpdateModel() 是否成功， 而取决于全局的

ModelState.IsValid 是否通过

* 1. 自定义模型绑定



* + 1. 定义好数据模型类

public class Address

{

public int Suite { get; set; }

public string Street { get; set; }

public string City { get; set; }

public string ShipAddress { get; set; }

}

* + 1. 自定义模型绑定器：实施ImodelBinder

public class AddressBinder : IModelBinder

{

public object BindModel(ControllerContext controllerContext,

ModelBindingContext bindingContext)

{

HttpRequestBase request = controllerContext.HttpContext.Request;

int ss = 0;

int.TryParse(request.Form["suite"], out ss);

string st = request. Form ["Street"];

string cc = request.Form["City"];

string ad = string.Format("Suite#{0}, Street:{1}, City:{2}", ss, st, cc);

return new Address

{

Suite = ss,

Street = st,

City = cc,

ShipAddress = ad

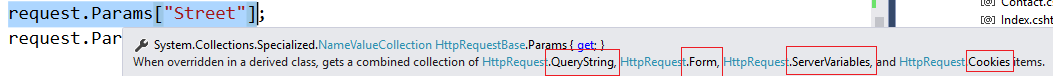
};

}

}

注意：

* 如果是获取form post 过来的数据：request.Form["xx"]
* 如果是获取method get 过来的数据：request.QueryString["xxx"]
* 如果是获取组合数据：request.Params["Street"], 它会按顺序从  
  QueryString, Form, ServerVariables, Cookies 里获取数据，用逗号隔开



* + 1. 全局使用 custom ModelBinder, 可以在 Application\_Start() 里注册

ModelBinders.Binders.Add(typeof(Address), new AddressBinder());

* + 1. 由于是全局注册过，直接使用即可：

public ActionResult About(User form1, User form2, Address paddr)

<div class="row">

<div class="col-md-2">Address</div>

<div class="col-md-6">@string.Join(", ", Addr.Suite, Addr.Street, Addr.City )</div>

</div>

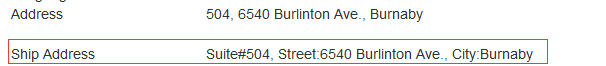
<br />

<div class="row">

<div class="col-md-2">Addr</div>

<div class="col-md-6">@Addr.ShipAddress</div>

</div>



* + 1. 如果不是全局注册，也可以单独使用

Parameter: [ModelBinder(typeof(AddressBinder))] Address paddr

public ActionResult About( User form1,

User form2,

[ModelBinder(typeof(AddressBinder))] Address paddr)

{

ViewBag.User = form1;

ViewBag.Address = paddr;

return View(form2);

}

* + 1. 自定义ModelBinder 处理多个情况：

public class CustomBinder : IModelBinder

{

public object BindModel( ControllerContext controllerContext,

ModelBindingContext bindingContext)

{

HttpRequestBase request = controllerContext.HttpContext.Request;

// 情况一

if (bindingContext.ModelType==typeof(Address))

{

int ss = 0;

int.TryParse(request.Form["suite"], out ss);

string st = request.Form["Street"];

string cc = request.Form["City"];

string ad = string.Format("Suite#{0}, Street:{1}, City:{2} Model:{3}-{4}-{5}-{6}", ss, st, cc, bindingContext.ModelName, bindingContext.ModelState.IsValid, bindingContext.ModelType, bindingContext.Model);

return new Address

{

Suite = ss,

Street = st,

City = cc,

ShipAddress = ad

};

}

// 情况二

if(bindingContext.ModelType==typeof(User))

{

string userstr = string.IsNullOrEmpty(request.Form["User\_name"])?"Unknown": request.Form["User\_name"];

string[] users = userstr.Split(new char[] { ',' });

return new User

{

USER\_NAME = users,

spouse\_name = "Bind Spouse",

birthdate = DateTime.Now,

salary = 100,

language = new string[] { "EN", "CN", "FR" }

};

}

return null;

}

}

public ActionResult About(User form1,

[ModelBinder(typeof(CustomBinder))] User form2,

[ModelBinder(typeof(CustomBinder))] Address paddr

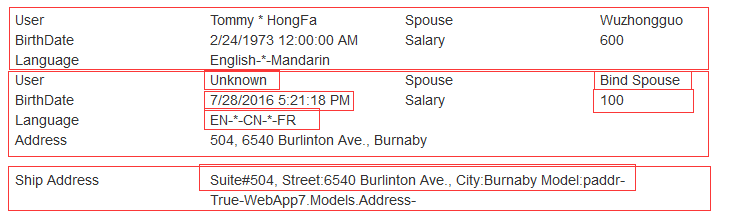
) {

ViewBag.User = form2;

ViewBag.Address = paddr;

return View(form1);

}



可以看到 form2 , paddr 完全按照自定义逻辑来映射数据

* + 1. 默认的DefaultModelBinder: 影响全局

public class **mydefBinder**: DefaultModelBinder

{

public override object BindModel( ControllerContext controllerContext,

ModelBindingContext bindingContext)

{

if(bindingContext.ModelType == typeof(Address))

{

return new Address

{

Suite = 999,

Street = "Unknown Street",

City = "Lost City",

ShipAddress = "Nowhere"

};

}

if(bindingContext.ModelType == typeof(User))

{

return new User

{

USER\_NAME = new string[] { "Default", "User" },

spouse\_name = "Default spouse",

language = new string[] { "Def en", "Def cn" },

birthdate = DateTime.Now.Subtract(new TimeSpan(100, 100, 100)),

salary = 11111

};

}

return base.BindModel(controllerContext, bindingContext);

}

}

protected void Application\_Start()

{

ModelBinders.Binders.DefaultBinder = new **mydefBinder**();

}



自定义的ModelBinder 优先级比默认的DefaultModelBinder 高

* + 1. 提供者 IModelBinderProvider

步骤：  
1、通过实现IModelBinder接口，可以创建自定义模型绑定器。  
2.注册自定义模型绑定器有三种方法。  
 方法一：在应用程序的Global.asax的Application\_Sart()方法注册自定义的模型绑定器。  
 ModelBinders.Binders.Add(

typeof(Address),

new AddressModelBinder()

);

方法二：创建模型绑定器提供器。

这个对有多个类型进行操作的的自定义模型绑定器或有许多提供器要维护，这个是个灵活的方式）

1. 通过实现IModelBinderProvider接口来创建一个模型绑定器提供器。这个接口的定义如下：

public interface IModelBinderProvider {

IModelBinder GetBinder(Type modelType); }

}  
  
 2）这个自定义的模型绑定器提供器也需要到应用程序中注册。当然也是在Global.asax的

Application\_Start方法中注册。

假如我有这么个自定义模型绑定器提供器：  
   
 public class CustomModelBinderProvider:IModelBinderProvider

{

public IModelBinder GetBinder(Type modelType)

{

return modelType==typeof(Address)?new AddressModerBrinder():null;

}

}  
  
 那么我要注册CustomeModelBinderProvider 模型绑定器提供器，就应该这样：  
 protected void Application\_Start()

{

AreaRegistration.RegisterAllAreas();

ModelBinderProviders.BinderProviders.Add(

new CustomModelBinderProvider()

);

RouteConfig.RegisterRoutes(RouteTable.Routes);

}

* + 1. 提供者IModelBinderProvider实例

第一步： 定义好ModelBinder

public class **AddressBinder** : IModelBinder

{

public object BindModel( ControllerContext controllerContext,

ModelBindingContext bindingContext)

{

HttpRequestBase request = controllerContext.HttpContext.Request;

if (bindingContext.ModelType==typeof(Address))

{

int ss = 0;

int.TryParse(request.Form["suite"], out ss);

string st = request.Form["Street"];

string cc = request.Form["City"];

string ad = string.Format("Suite#{0}, Street:{1}, City:{2} Model:{3}-{4}-{5}-{6}", ss, st, cc, bindingContext.ModelName, bindingContext.ModelState.IsValid, bindingContext.ModelType, bindingContext.Model);

return new Address

{

Suite = ss,

Street = st,

City = cc,

ShipAddress = ad

};

}

return null;

}

}

public class **UserBinder** : IModelBinder

{

public object BindModel( ControllerContext controllerContext,

ModelBindingContext bindingContext)

{

HttpRequestBase request = controllerContext.HttpContext.Request;

if (bindingContext.ModelType == typeof(User))

{

string userstr = string.IsNullOrEmpty(request.Form["User\_name"]) ? "Unknown" : request.Form["User\_name"];

string[] users = userstr.Split(new char[] { ',' });

return new User

{

USER\_NAME = users,

spouse\_name = "Bind Spouse",

birthdate = DateTime.Now,

salary = 100,

language = new string[] { "EN", "CN", "FR" }

};

}

return null;

}

}

第二步： 定义好ModelBinderProvider

很简单：根据不同的类型，返回不同的ModelBinder

public class **myModelProvider** : IModelBinderProvider

{

public IModelBinder GetBinder(Type modelType)

{

IModelBinder ret = null;

if (modelType == typeof(Address)) ret = new **AddressBinder**();

if (modelType == typeof(User)) ret = new **UserBinder**();

return ret;

}

}

第三步： Application\_Start() 注册提供者

protected void Application\_Start()

{

AreaRegistration.RegisterAllAreas();

FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);

RouteConfig.RegisterRoutes(RouteTable.Routes);

BundleConfig.RegisterBundles(BundleTable.Bundles);

ModelBinderProviders.BinderProviders.Add(new **myModelProvider**());

}

测试：

public ActionResult About(User form1, User form2, Address paddr)

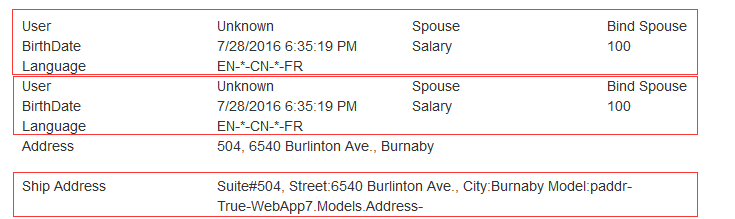
{

ViewBag.User = form2;

ViewBag.Address = paddr;

return View(form1);

}



1. 表单Form和HTML 辅助方法
   1. Action and Method : action=”url” method=”post|get”

Default : http.get url: relative path or absolute path

通常： get 用于读取操作，url 带有键值对， 这样可以保留url 使用

Post 用于更新， 创建，删除操作， 要预防重复提交。

* 1. URL Path 和 Server Path
     1. RouteTable.Routes.GetVirtualPath(requestContext, RouteValueDictionary)

此函数是构建新的虚拟路劲:

@{

var context = this.ViewContext.RequestContext;

var urlvalue = new RouteValueDictionary {

{ "controller", "Home"},

{ "action", "About"}

};

var path = RouteTable.Routes.GetVirtualPath(context,urlvalue);

var urlvalue1 = new RouteValueDictionary

{

{"controller","Sales" },

{"idd", 200 }

};

var path1 = RouteTable.Routes.GetVirtualPath(context, urlvalue1);

var urlvalue2 = new RouteValueDictionary

{

{"action", "Shop" }

};

var path2 = RouteTable.Routes.GetVirtualPath(context, urlvalue2);

}

URL: @path.VirtualPath<br />

URL1: @path1.VirtualPath<br />

URL2: @path2.VirtualPath<br />

Path: @Server.MapPath(path.VirtualPath)<br />

URL: /Home/About  
URL1: /Sales?idd=200 - 没有 Action, 除了”controller”, “action” 以外， 其他都是 QueryString 内容  
URL2: /Home/Shop - 如果没有controller, 则使用当前的 controller   
Path: D:\Projects\VS2015\WebApp7\WebApp7\Home\About

* + 1. Server.MapPath()

Path0: @Server.MapPath("good/day/hello.cshtml")<br />

Path1: @Server.MapPath("/Hello/World/day.html")<br />

Path2: @Server.MapPath("~/Doc/Video/hello")<br />

Path3: @Server.MapPath("~/Doc of Root/Video Of Downtown/hello World")<br />

Path0: D:\Projects\VS2015\WebApp7\WebApp7\Home\good\day\hello.cshtml  
Path1: D:\Projects\VS2015\WebApp7\WebApp7\Hello\World\day.html  
Path2: D:\Projects\VS2015\WebApp7\WebApp7\Doc\Video\hello  
Path3: D:\Projects\VS2015\WebApp7\WebApp7\Doc of Root\Video Of Downtown\hello World

* 1. Html.BeginForm() Form辅助方法

@{ Html.BeginForm(); }

<div class="row">

<div class="col-md-1">User</div>

<div class="col-md-2">

<input type="text" name="user\_Name" value="@Model.USER\_NAME" />

</div>

<div class="col-md-1 col-lg-offset-1">Spouse</div>

<div class="col-md-2">

<input type="text" name="spouse\_name" value="@Model.spouse\_name" />

</div>

</div>

<div class="row">

<div class="col-md-1">Birth</div>

<div class="col-md-2">

<input type="date" name="birthdate" value="@Model.birthdate" />

</div>

<div class="col-md-1 col-lg-offset-1">Salary</div>

<div class="col-md-2">

<input type="number" name="salary" value="@Model.salary" />

</div>

</div>

<input type="submit" name="btnSubmit" class="btn btn-info" value="Submit Me" />

@{Html.EndForm();}

http://localhost:21371/Home/Index

<form method="post" action="/Home/Index">

http://localhost:21371/

<form method="post" action="/">

http://localhost:21371/HOME

<form method="post" action="/HOME">

http://localhost:21371/HOME?user\_name=GoogleMe

<form method="post" action="/HOME?user\_name=GoogleMe">

默认和地址栏里的地址一样，去除<http://hostname>

Method: 默认是 Post , 而<form> 默认是 get

另外一种写法：

@using (Html.BeginForm()) {

<div class="row">

<div class="col-md-1">User</div>

<div class="col-md-2">

<input type="text" name="user\_Name" value="@Model.USER\_NAME" />

</div>

<div class="col-md-1 col-lg-offset-1">Spouse</div>

<div class="col-md-2">

<input type="text" name="spouse\_name" value="@Model.spouse\_name" />

</div>

</div>

<div class="row">

<div class="col-md-1">Birth</div>

<div class="col-md-2">

<input type="date" name="birthdate" value="@Model.birthdate" />

</div>

<div class="col-md-1 col-lg-offset-1">Salary</div>

<div class="col-md-2">

<input type="number" name="salary" value="@Model.salary" />

</div>

</div>

<input type="submit" name="btnSubmit" class="btn btn-info" value="Submit Me" />

}

1. Html.BeginForm()

Html.BeginForm()

<form method="post" action="**/Home/Index**"> - 默认是当前页面

1. Html.BeginForm(object routeValues)

Html.BeginForm(new { idd = 200, street = "Burlington" })

<form method="post" action="**/Home/About**?idd=200&street=Burlington"> - 默认是当前页面

Html.BeginForm(new {idd=200, uid=300, controller="**sales**",

action="**detail**", foo="Hello World" })

<form method="post" action="**/sales/detail**?idd=200&uid=300&foo=Hello%20World">

如果遇到 controller, action 将转化为路由里的controller, action

1. Html.BeginForm(RouteValueDictionary)

Html.BeginForm(new RouteValueDictionary {

{ "controller", "sales" },

{ "action", "detail" },

{ "user\_name", "Good Man" }

})

<form method="post" action="/sales/detail?user\_name=Good%20Man">

Form[“user\_name”] 优先级比 QueryString[“user\_name”] 要高，即是Form[“user\_name”]为空

1. Html.BeginForm(actionName, controllerName)

Html.BeginForm("detail","sales")

<form method="post" action="/sales/detail">

1. Html.BeginForm(actionName, controllerName, FormMethod)

Html.BeginForm("detail","sales",FormMethod.**Get**)

<form method="**get**"  action="/sales/detail">

支持Named Parameter: 所以也可以这么写

Html.BeginForm(method: FormMethod.Get, controllerName:"sales", actionName:"detail")

1. Html.BeginForm(actionName, controllerName, object routeValues)

Html.BeginForm( "**about**", "**home**",

new {idd=200, uid=300, controller="**sales**", action="**detail**", foo="Hello World" })

<form method="post" action="**/home/about**?idd=200&uid=300&foo=Hello%20World">

当 routeValues里也有controller, action, 则优先级低,

即使actionName=””, controllerName=”” 为空字符串也优先级高

但是如果: actionName=null, controllerName=null 则使用 routeValues里的值

Html.BeginForm(null, null,

new {idd=200, uid=300, controller="sales", action="detail", foo="Hello World" })

<form method="post" action="/sales/detail?idd=200&uid=300&foo=Hello%20World">

1. Html.BeginForm(actionName, controllerName, RouteValueDictionary)

Html.BeginForm("**detail**",null, new RouteValueDictionary {

{ "controller","**google**" },

{"action","**search**" },

{"idd", 200 },

{"bdate", DateTime.Now }

})

<form method="post" action="/**google**/**detail**?idd=200&bdate=07%2F29%2F2016%2012%3A05%3A02">

原理同上：controllerName, actionName 即使是空字符串，只要不为null, 优先级高

如果为null, 则使用 RouteValueDictionary 里的值

1. Html.BeginForm(actionName, controllerName, FormMethod, object htmlAttributes)

Html.BeginForm( "detail","sales",FormMethod.Get,

new { target="\_blank", style="color:red;", @class="myform", my\_data\_id=100 })

<form  class="myform"  target="\_blank"  style="color:red;"  my-data-id="100"

method="get"  action="/sales/detail">

注意：

htmlAttributes 实际上就是HTML 的属性， 由于C# 变量不能使用 – 减号， 所以必须使用下划线 \_ , 下划线对应到属性则变成 - 。 如：**my\_data\_id** => **my-data-id**

C# class 是保留字， html class 必须使用 @class 如： @class=”myform”

1. Html.BeginForm( actionName, controllerName, FormMethod,

IDictionary<string,object> htmlAttributes)

@{

IDictionary<string, object> attr = new Dictionary<string, object>

{

{"class", "myform111" },

{"style", "color:blue;font-size:1.5em" },

{"my\_data\_id", 2999 }

};

}

@using (Html.BeginForm("detail","sales",FormMethod.Get, attr) ) {

}

<form  class="myform111"  style="color:blue; font-size:1.5em" my\_data\_id="2999"

method="get" action="/sales/detail">

IDictionary<string, object> 的好处是：属性的键是字符串，可以避免C#语法的保留字，减号缺陷

1. Html.BeginForm(actionName, controllerName, object routeValues, FormMethod)
2. Html.BeginForm(actionName, controllerName, RouteValueDictionary, FormMethod)
3. Html.BeginForm(actionName, controllerName, routeValues, FormMethod,

object htmlAttributes)

1. Html.BeginForm(actionName, controllerName, RouteValueDictionary, FormMethod,

IDictionary<string,object> htmlAttributes)

1. 13 的规律是 object & object; Dictionary & Dictionary

@{ Ajax.BeginForm("Download", "Home",

new AjaxOptions { UpdateTargetId = "downloadid" },

new { name = "dwnfrm" } ); }

@{ Html.EndForm(); }

如果参数给错了， 会出现URL 错误 Length=4 , 正好是 Home的长度

<http://localhost:3469/Home/Download?Length=4>

正确写法： 要按照方法参数的约定

@{ Ajax.BeginForm("Download", new { Controller = "Home" },

new AjaxOptions { UpdateTargetId = "downloadid" },

new { name = "dwnfrm" } ); }

@{ Html.EndForm(); }

* 1. Html.TextBox()
     1. Html.TextBox( ) - 数据自动绑定原理

Title: @Html.TextBox("Title")<br />

UName: @Html.TextBox("user\_name")<br />

Title: <input id="Title" type="text" value="**Home Page**" name="Title">

<br>

Name: <input id="user\_name" type="text" value="" name="user\_name">

解析：

1. TextBox的name 不仅是 html input 的name 名称， 也是用于数据模型自动绑定
2. 如何自动绑定？由controller 传进来的无非就是 ViewData, ViewBag, ViewData.Model, TempData
3. 自动绑定: 优先 ViewData(ViewBag) 然后才是ViewData.Model, TempData 不参与

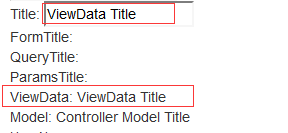
Title: @Html.TextBox("Title")<br />

FormTitle: @Request.Form["Title"]<br />

ViewData: @ViewData["Title"]<br />

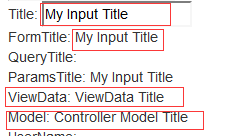
Model: @Model.Title<br />

页面初始化载入时：



TextBox 数据自动绑定原则：

1. 由于TextBox本身是输入元素input, 所以它会先绑定本身post 出去的值



1. ViewData[“xxx”] ( ViewBag.xxx ) 的优先级别比 Model.xxx 的高

如果页面初始化载入时，如上图可以看见 ViewData 的级别高，

1. TempData 不参与自动数据绑定，可以不理会

多个同名TextBox

Title: @Html.TextBox("Title")<br />

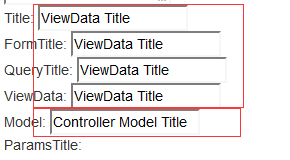
FormTitle: @Html.TextBox("Title",Request.Form["Title"])<br />

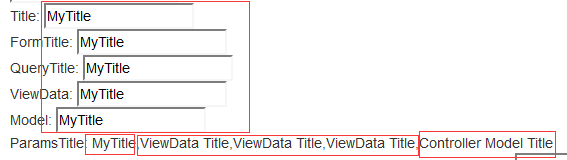
QueryTitle: @Html.TextBox("Title", Request.QueryString["Title"])<br />

ViewData: @Html.TextBox("Title", ViewData["Title"])<br />

Model: @Html.TextBox("Title", Model.Title)<br />

ParamsTitle: @Request.Params["Title"]<br />





所以同名的TextBox 全部绑定一样的值就是第一个Post 出去的值

也就是第一个同名元素post 出去的值，不管你TextBox 强制指定的值，强制指定的值是初始化时使用的默认值。

@Request.Params["Title"] 合并顺序是： QueryString, Form, SeverVariables, Cookies

1. 绑定到对象的属性上： userObj.User\_Name

TextBox User\_Name: @Html.TextBox("user\_name")<br />

<br />

User.User\_Name: @Html.TextArea("User.user\_name")<br />

public ActionResult Index(User userObj)

{

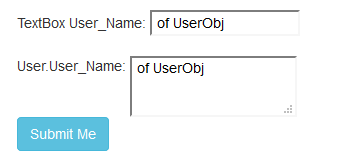
userObj.USER\_NAME += " of UserObj";

ViewData["User"] = userObj;

return View(userObj);

}

初始化页面时， 两个值是一样的





提交以后： @Html.TextBox("user\_name") 如前所述优先绑定post 的值

而@Html.TextArea("User.user\_name") 绑定的是ViewData["User"]对象的属性

userObj.USER\_NAME 属性全大写，User.user\_name 全小写，由于不是直接使用对象，

所以只是使用关键字搜索， 大小写不敏感。

这也和之前讲过的一样原理：

<input type="text" name="user1.user\_name" value="" />

<input type="text" name="user2.user\_name" value="" />

public ActionResult Index(User user1, User user2)

{ … }

* + 1. Html.TextBox( name )
    2. Html.TextBox( name , object value )
    3. Html.TextBox( name , object value, object HtmlAttributes )

@Html.TextBox("user\_name", null,

new {id="my\_user\_name", style="color:red;font-size:2em;" })

<input id="my\_user\_name" type="text" value=" of UserObj"

style="color:red;font-size:2em;" name="user\_name">

初始化时，虽然 value=null , 但是 Model.USER\_NAME 起作用

* + 1. Html.TextBox( name , object value, IDictionary<string,object> htmlAttributes)

原理同 Html.BeginForm()一样

* + 1. Html.TextBox( name , object value, string format)

format因为在HTML input 元素输入的类型都是字符型，

所以这里的format都是针对C# 字符串类型的输出格式定义 toString( format ) 类似

@Html.TextBox("birthdate", null, "{0:MMMM dd, yyyy}")

@Html.TextBox("birthdate", null, "{0:yyyy/MM/dd hh:mm}")

@Html.TextBox("SALARY", null, "{0:0,000,000}")



* + 1. Html.TextBox( name , object value, string format, object HtmlAttributes)
    2. Html.TextBox( name , object value, string format,

IDictionary<string,object> htmlAttributes)

其他的参数同上，在此不累述

* 1. Html.Label() & Html.DisplayName()
     1. Html.Label( string expression ) – expression 是试图去自动匹配

ViewData["User\_Name"] = "My name";

@Html.Label("user\_name") - 不会找到 ViewData

<label for="user\_name">user\_name</label>

public ActionResult Index(User userObj)

{

return View(userObj);

}

@Html.Label("user\_name") - 会去Model里找到 user\_name 对应的属性

<label for="user\_name">**USER\_NAME**</label> 注意 USER\_NAME 是Model的属性名

public class User

{

[Display(Name ="用户名")]

public string USER\_NAME { get; set; }

}

<label for="user\_name">用户名</label> - 会自动使用 Display(Name=””)

* + 1. Html.Label( expression, string labelText )

@Html.Label("user\_name", "优先用户")

<label for="user\_name">优先用户</label>

labelText – 优先级高

* + 1. Html.Label( expression, object HtmlAttributes )

@Html.Label("user\_name", new { style="color:red;font-weight:700;" } )

<label style="color:red;font-weight:700;" for="user\_name">用户名</label>

* + 1. Html.Label( expression, IDictionary<string,object> htmlAttributes)

@Html.Label("user\_name", new Dictionary<string,object> {

{ "**style**","color:red;font-weight:700;" },

{"**class**", "label\_css" }

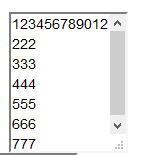
})

<label **class**="label\_css" **style**="color:red;font-weight:700;" for="user\_name">用户名</label>

* + 1. Html.Label( expression, string labelText, object HtmlAttributes )
    2. Html.Label( expression, string labelText, object HtmlAttributes )
    3. Html.DisplayName(expression) – 输出纯文本没有<label>, 也没有labelText, htmlAttributes
  1. Html.TextArea() 和 TextBox()类似
     1. Html.TextArea(string name, string value, htmlAttributes ) – 6 个重载
     2. Html.TextArea(name, string value, int rows, int cols, object htmlAttributes )

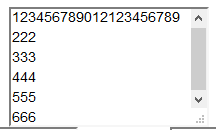
@Html.TextArea("user\_name", null, 6,12,null)

<textarea id="user\_name" rows="6" name="user\_name" cols="12"></textarea>



@Html.TextArea("user\_name", null, 6,12,new { style = "width:200px;height:120px;" })

<textarea id="user\_name" style="width:200px;height:120px;" rows="6" name="user\_name" cols="12"></textarea>



* 当我们指定了 width , height 则内容按照width, height 来填充
* 如果没有指定width, height 则 textarea 的宽和高，由 rows, cols,还有字体大小来决定textarea 的大小

* + 1. Html.TextArea(name, string value, int rows, int cols, IDictionary htmlAttributes )
  1. Html.Password()
     1. Html.Password(name, object value, htmlAttributes) – 4 个重载

@Html.Password("user\_name")

<input id="user\_name" type="password"  name="user\_name">

* 1. Html.Hidden()
     1. Html.Hidden(name, object value, htmlAttributes) – 4 个重载

@Html.Hidden("birthdate", DateTime.Now)

<input id="birthdate" type="hidden" value="7/29/2016 10:39:35 PM" name="birthdate" data-val-date="The field birthdate must be a date." data-val="true">

* 1. Html.ValidationSummary()
     1. Html.ValidationSummary()

@Html.ValidationSummary(false, "There are some errors", new { style="color:red;"}, "div")



<div class="validation-summary-errors" style="color:red;" data-valmsg-summary="true">

<div>There are some errors</div>

<ul>

<li>The value 'sdfasdf10:44:31 PM' is not valid for birthdate.</li>

</ul>

</div>

* 1. Html.DropDownList()
     1. Html.DropDownList(name)

@Html.DropDownList("shipAddress")

传进来的是 SelectList 或者IEnumerable<SelectListItem> 才可以

public ActionResult Index(User userObj)

{

IList<SelectListItem> addrList = new List<SelectListItem>

{

new SelectListItem { Value="100", Text="AAAA AAA" },

new SelectListItem { Value="200", Text="BBBB BBB" },

new SelectListItem { Value="255", Text="CCC CCC" }

};

// 选择 200 – BBBB BBB

SelectListItem selItem = addrList.Where(p => p.Value == "200").FirstOrDefault();

if (selItem != null) selItem.Selected = true;

ViewData["shipAddress"] = addrList;

return View(userObj);

}



<select id="shipAddress" name="shipAddress">

<option value="100">AAAA AAA</option>

<option value="200" selected="selected">BBBB BBB</option>

<option value="255">CCC CCC</option>

</select>

当我们提交 就会生成键值对 shipAddress=200

注意不是提交整个List, 虽然名字一样，因为传进来是List , 提交出去就是选

择的值

DropDownList 数据源来自于实体模型数据

技术要点：将IEnumerable<Entity> 生成SelectList指定键值字段，文本字段

public ActionResult Index(User userObj)

{

IList<Address> addrList = new List<Address>

{

new Address { id=100, Street="AAAA AAA" },

new Address { id=200, Street="BBBB BBB" },

new Address { id=255, Street="CCC CCC" }

};

SelectList addrSelectList = new SelectList(addrList, "id", "Street", **200**);

ViewData["shipAddress"] = addrSelectList;

return View(userObj);

}

@Html.DropDownList("shipAddress", "Please Select")



<select id="shipAddress" name="shipAddress">

<option value="">Please Select</option>

<option value="100">AAAA AAA</option>

<option value="**200**" selected="selected">BBBB BBB</option>

<option value="255">CCC CCC</option>

</select>

DropDownList 传递IEnumerable<Entity> 到View，在View里生成 SelectList

public ActionResult Index(User userObj)

{

HashSet<Address> addrList = new HashSet<Address>

{

new Address { id=100, Street="AAAA AAA" },

new Address { id=200, Street="BBBB BBB" },

new Address { id=255, Street="CCC CCC" },

new Address { id=300, Street="DDD DDDD" },

new Address { id=400, Street="EEE EEEE" }

};

ViewData["shipAddress"] = addrList;

return View(userObj);

}

@Html.DropDownList("salary",

new SelectList( (IEnumerable<Address>)ViewData["shipAddress"], "id", "street", 300) )

(IEnumerable<Address>)ViewData["shipAddress"] 记得 object拆箱(IEnumerable<Address>)



* + 1. Html.DropDownList(name, string optionLabel)

@Html.DropDownList("shipAddress", "Please Select")

<select id="shipAddress" name="shipAddress">

<option value="">Please Select</option> - 生成一个空值提示

<option value="100">AAAA AAA</option>

<option value="200" selected="selected">BBBB BBB</option>

<option value="255">CCC CCC</option>

</select>



所以接收方必须是： int? 可以为nullable , 否则出错

public ActionResult About(int? shipAddress)

{

ViewData["shipID"] = shipAddress;

return View();

}

* + 1. Html.DropDownList(name, IEnumerable<SelectListItem> selectList)

name – 是 HTML select 元素的名称，用于模型绑定的名称

selectList – 是生成列表的数据清单

和之前Html.DropDownList(name) 默认是同名不一样，现在可以指定其他名称

public ActionResult Index(User userObj)

{

HashSet<Address> addrList = new HashSet<Address>

{

new Address { id=100, Street="AAAA AAA" },

new Address { id=200, Street="BBBB BBB" },

new Address { id=255, Street="CCC CCC" },

new Address { id=300, Street="DDD DDDD" }

};

SelectList addrSelectList = new SelectList(addrList, "id", "Street", 200);

ViewData["**shipAddress**"] = addrSelectList;

//ViewData["myAddress"] = 300;

return View(userObj);

}

@Html.DropDownList("**myAddress**", ViewData["**shipAddress**"] as SelectList)

// 一下默认被选中的是200, 前提是没有指定 myAddress的值

ViewData["myAddress"] = null; 为null 默认被选中的是 200



<select id="**myAddress**" name="myAddress">

<option value="">Please Select</option>

<option value="100">AAAA AAA</option>

<option value="200" selected="selected">BBBB BBB</option>

<option value="255">CCC CCC</option>

<option value="300">DDD DDDD</option>

</select>

如果

ViewData["shipAddress"] = addrSelectList;

ViewData["myAddress"] = 300;

return View(userObj);

300 才是绑定的值， 也就是列表的选定项目



如果没有默认选定项目，也没有绑定数据值或者值为null,

也没有optionLabel, 则第一个项目出现在选择框里，但是状态不是selected

<select id="myAddress" name="myAddress">

<option value="100">AAAA AAA</option> - 注意状态不是 selected

<option value="200">BBBB BBB</option>

<option value="255">CCC CCC</option>

<option value="300">DDD DDDD</option>

</select>



**常见的错误**：当我们使用IList<SelectListItem> 时

public ActionResult Index(User userObj)

{

IList<SelectListItem> addrList = new List<SelectListItem>

{

new SelectListItem { Value="100", Text="AAAA AAA" },

new SelectListItem { Value="200", Text="BBBB BBB" },

new SelectListItem { Value="255", Text="CCC CCC", Selected =true },

new SelectListItem { Value="300", Text="DDD DDD" },

new SelectListItem { Value="400", Text="EEE EEE" }

};

ViewData["shipAddress"] = addrList;

return View(userObj);

}

@Html.DropDownList("salary", ViewData["shipAddress"] as SelectList)

当我们试图绑定给 数据模型的属性 salary 时， 页面会出错：

## *There is no ViewData item of type 'IEnumerable<SelectListItem>' that has the key 'salary'.*

原因是：

因为IList<SelectListItem> 是默认连接到ViewData[“shipAddress”]

所以使用同名绑定则没有问题：

@Html.DropDownList("shipAddress", ViewData["shipAddress"] as SelectList)



总结：

想使用不同名的绑定， 不能使用IList<SelectListItem>

可以使用Ilist<Entity>, Ienumerable<Entity>, 然后生成 SelectList(). 通过SelectList 构造函数，可以指定键值字段， 文本字段，以及默认选择项。

虽然DropDownList 是单选， 但是模型绑定可以是数组如： int[], string[]

IList<SelectListItem> glist = new List<SelectListItem>

{

new SelectListItem

{

Value = "male",

Text = "Male",

Selected = true

},

new SelectListItem

{

Value = "female",

Text = "Female",

Selected = true

},

new SelectListItem

{

Value = "transex",

Text = "TransSex",

Selected = true

}

};

SelectList slist = new SelectList(glist, "value", "text" , "female");

ViewData["sexlist"] = slist;

return View(ff);

@Html.DropDownList("**gender**", ViewData["**sexlist**"] as SelectList )

注意：如果使用SelectList 可以使用不同的名称，否则如果使用IList<SelectListItem>

则必须ViewData[xxx] 和 DropDownList(“xxx”, ViewData[xxx] as IEnumerable<SelectListItem>) 必须同名

* + 1. Html.DropDownList(name, selectList, optionLabel)
    2. Html.DropDownList(name, selectList, object htmlAttributes )

@Html.DropDownList("myAddress", ViewData["shipAddress"] as SelectList,

new {id="addr\_id", style="color:blue;font-size:2em;"})



<select id="addr\_id" style="color:blue;font-size:2em;" name="myAddress">

<option value="100">AAAA AAA</option>

<option value="200">BBBB BBB</option>

<option value="255">CCC CCC</option>

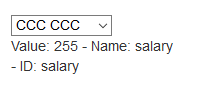
<option value="300">DDD DDDD</option>

</select>

* + 1. Html.DropDownList(name, selectList, Idictionary<s , o> htmlAttributes )
    2. Html.DropDownList(name, selectList, optionLabel, object htmlAttributes )
    3. Html.DropDownList(name, selectList, optionLabel, Idictionary<s , o> htmlAttributes)

@Html.DropDownList("salary", ViewData["shipAddress"] as SelectList, new { id = "addrid" })<br />

Value: @Html.Value("salary") - Name: @Html.Name("salary") - ID: @Html.Id("salary")



Html.Value() Html.Name(), Html.Id() 更改id 不起作用？还是name的值

DropDownList 可以变成单选的ListBox, 设置 size 属性

@Html.DropDownList("salary", ViewData["shipAddress"] as SelectList,

new { style = "width:200px;height:160px;", size=20 })

* 1. Html.ListBox()

工作原理和DropDownList 一样， 不同的是没有 optionLabel , 支持多选。其他原则一样

* + 1. Html.ListBox(name, selectList, htmlAttributes) – 4 个重载方法

public class User

{

[Display(Name ="用户名")]

public string USER\_NAME { get; set; }

public string homer { get; set; }

public string spouse\_name { get; set; }

public DateTime? birthdate { get; set; }

public **int[]** salary { get; set; } - 多选必须是数组, 否则出错

public string Title { get; set; }

}

public ActionResult Index(User userObj)

{

HashSet<Address> addrList = new HashSet<Address>

{

new Address { id=100, Street="AAAA AAA" },

new Address { id=200, Street="BBBB BBB" },

new Address { id=255, Street="CCC CCC" },

new Address { id=300, Street="DDD DDDD" },

new Address { id=400, Street="EEE EEEE" }

};

SelectList addrSelectList = new SelectList(addrList, "id", "Street" ,

new int[] { 200, 300 });

addrSelectList.Where(p => p.Value == "200").Single().Selected = true;

addrSelectList.Where(p => p.Value == "300").Single().Selected = true;

ViewData["shipAddress"] = addrSelectList;

return View(userObj);

}

@Html.ListBox("salary", ViewData["shipAddress"] as SelectList, new { style = "width:200px;height:160px;" }) <br />

Html.:

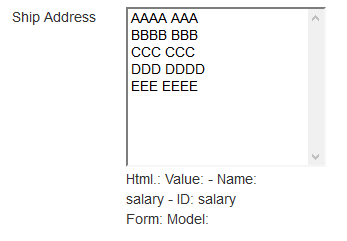
Value: @Html.Value("salary") –

Name: @Html.Name("salary") –

ID: @Html.Id("salary")<br />

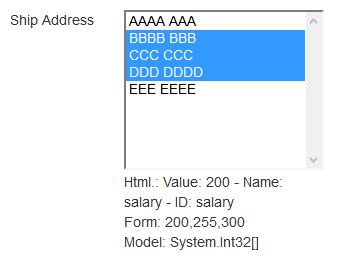
Form: @Request.Form["salary"]

Model: @Model.salary



虽然初始数据为null, 但是默认选项并没有被选中。不知为何？

选择并提交, 输出如下：



<select id="salary" style="width:200px;height:160px;" name="salary" multiple="multiple">

<option value="100">AAAA AAA</option>

<option value="200" selected="selected">BBBB BBB</option>

<option value="255" selected="selected">CCC CCC</option>

<option value="300" selected="selected">DDD DDDD</option>

<option value="400">EEE EEEE</option>

</select>

* + 1. Html.ListBox() – 预选

public ActionResult Index(User userObj)

{

HashSet<Address> addrList = new HashSet<Address>

{

new Address { id=100, Street="AAAA AAA" },

new Address { id=200, Street="BBBB BBB" },

new Address { id=255, Street="CCC CCC" },

new Address { id=300, Street="DDD DDDD" },

new Address { id=400, Street="EEE EEEE" }

};

SelectList addrSelectList = new SelectList(addrList, "id", "Street",

new int[] { 200, 300 });

addrSelectList.Where(p => p.Value == "200").Single().Selected = true;

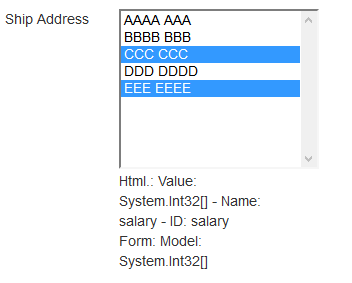
addrSelectList.Where(p => p.Value == "300").Single().Selected = true;

ViewData["shipAddress"] = addrSelectList;

userObj.salary = new int[] { 255, 400 }; - 这个会起到预选的作用

return View(userObj);

}



* 1. Html.CheckBox()
     1. Html.CheckBox(name, isChecked, htmlAttributes) – 6 个重载方法
     2. Html.CheckBox() 没有Value, Value 都是 true

public bool salary { get; set; }

数据模型必须是bool, 否则如下提到的错误

@Html.CheckBox("salary", true) Good Salary?

<input id="salary" type="checkbox" value="true" name="salary"

data-val-required="The salary field is required." data-val="true" checked="checked">

<input type="hidden" value="false" name="salary">

我们会奇怪的看到， CheckBox() 方法会产生两个input 元素：

除了 checkbox 外，还有一个同名hidden元素。为什么？

因为在HTML规范中， 只提交被选中的复选框的值。没有选中的就不会生成

键值提交， 也就是说， 如果复选框没有选中( false的情况) 是不会提交的。

所以通过同名的hidden 就可以保证 false 的情况被提交

即使我们选中提交以后：

<input id="salary" type="checkbox" value="true" name="salary"

data-val-required="The salary field is required." data-val="true"

**checked="checked"**>

<input type="hidden" value="false" name="salary">

注意： hidden永远是false 值，只是用来保证没有选中的情况，提交false

http://localhost:21371/?salary=true&salary=false&btnSubmit=Submit+Me

而选中被提交，我们会看到提交了两个salary=true&salary=false

优先取值true

* + 1. Html.CheckBox() 强行指定Value, 会遇到问题难以解决

public int salary { get; set; }

@Html.CheckBox("salary", false, new { value = 3000 } ) Good Salary?

<input id="salary" type="checkbox" value="3000" name="salary" data-val-required="The salary field is required." data-val-number="The field salary must be a number." data-val="true">

当我点击提交以后，会出现：

## *String was not recognized as a valid Boolean.*

**Description:** An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.   
  
**Exception Details:** System.FormatException: String was not recognized as a valid Boolean.  
  
**Source Error:**

|  |
| --- |
| Line 61: <div class="col-md-2">  Line 62: Customer: @Html.checkboxList("customer", 200)<br />  Line 63: @Html.CheckBox("salary", false, new { value = 3000 } ) Good Salary?<br /> |

* + 1. Html.CheckBox()方法只能适合处理 bool 类型的数据模型，

而不适合处理多值多选的情况

* 1. Html.RadioButton()
     1. Html.RadioButton(name, object value, isChecked, htmlAttributes) – 6个重载

Html.RadioButton(name, object value)

Html.RadioButton(name, object value, isChecked)

Html.RadioButton(name, object value, object htmlAttributes)

Html.RadioButton(name, object value, Idictionary<s,o> htmlAttributes)

Html.RadioButton(name, object value, isChecked, object htmlAttributes)

Html.RadioButton(name, object value, isChecked, IDictionary<s,o> htmlAttributes)

* + 1. Html.RadioButton()

public int salary { get; set; } - 必须和RadioButton Value相兼容的类型

@Html.RadioButton("salary",100) 100<br />

@Html.RadioButton("salary", 200) 200<br />

@Html.RadioButton("salary", 250, true) 250<br />

@Html.RadioButton("salary", 300, true) 300<br />

@Html.RadioButton("salary", 400) 400<br />

<input id="salary" type="radio" value="100" name="salary"

data-val-required="The salary field is required."

data-val-number="The field salary must be a number." data-val="true">

100

<br>

<input id="salary" type="radio" value="200" name="salary">

200

<br>

<input id="salary" type="radio" value="250" name="salary" checked="checked">

250

<br>

<input id="salary" type="radio" value="300" name="salary" checked="checked">

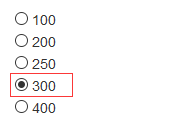
300

<br>

<input id="salary" type="radio" value="400" name="salary">

400

<br>



虽然设置了两个被选中，但是因为RadioButton是单选的，所以最后一个被选中

* + 1. Html.RadioButton() 预设值的问题

public ActionResult Index(User userObj)

{

userObj.salary = **200**;

return View(userObj);

}

@Html.RadioButton("salary",100) 100<br />

@Html.RadioButton("salary", 200) 200<br />

@Html.RadioButton("salary", **250**, true) 250<br />

@Html.RadioButton("salary", **300**, true) 300<br />

@Html.RadioButton("salary", 400) 400<br />

<input id="salary" type="radio" value="100" name="salary"

data-val-required="The salary field is required."

data-val-number="The field salary must be a number." data-val="true">

100

<br>

<input id="salary" type="radio" value="**200**" name="salary" **checked="checked"**>

200

<br>

<input id="salary" type="radio" value="**250**" name="salary" **checked="checked**">

250

<br>

<input id="salary" type="radio" value="**300**" name="salary" **checked="checked**">

300

<br>

<input id="salary" type="radio" value="400" name="salary">

400

<br>

我们发现都被选上了， 但是最后一个才会显示选中。

所以尽量不要在RadioButton() 方法里预设选中。

注意：这些预设值是在初始化载入时起作用， 一旦提交以后，这些预设就不存在了。

* 1. Html.ValidationMessage()
     1. Html.ValidationMessage(modelName, errMessage, htmlAttributes, tage) – 12个
     2. Html.ValidationMessage(modelName, errMessage)

@Html.ValidationMessage("birthdate")

<span  class="field-validation-error"

data-valmsg-replace="true"

data-valmsg-for="birthdate">

The value 'DSFASD 12:06:07 PM' is not valid for birthdate.

</span>

@Html.ValidationMessage("birthdate","Birth date is invalid!,yyyy-mm-dd")

<span class="field-validation-valid" data-valmsg-replace="false" data-valmsg-for="birthdate">

Birth date is invalid!,yyyy-mm-dd

</span>

@Html.ValidationMessage("birthdate","Birth date is invalid!,yyyy-mm-dd", "s")

<s class="field-validation-error" data-valmsg-replace="false" data-valmsg-for="birthdate">

Birth date is invalid!,yyyy-mm-dd

</s>

**使用HTML 辅助方法输出超链接**

* 1. Html.ActionLink ()
     1. Html.ActionLink() – 10个重载方法

1. Html.ActionLink(linkText, actionName)
2. Html.ActionLink(linkText, actionName, [o]routeValues) - object
3. Html.ActionLink(linkText, actionName, [d]routeValues) – Dicitonary
4. Html.ActionLink(linkText, actionName, [o]routeValues, [o]htmlAttributes)
5. Html.ActionLink(linkText, actionName, [d]routeValues, [d]htmlAttributes)
6. Html.ActionLink(linkText, actionName, controllerName)

7) Html.ActionLink(linkText, actionName, controllerName,

[o]routeValues, [o]htmlAttributes)

8) Html.ActionLink(linkText, actionName, controllerName,

[d]routeValues, [d]htmlAttributes)

1. Html.ActionLink(linkText, actionName, controllerName,

protocal, hostName, fragment

[o]routeValues, [o]htmlAttributes)

1. Html.ActionLink(linkText, actionName, controllerName,

protocal, hostName, fragment

[d]routeValues, [d]htmlAttributes)

严格遵守方法的参数类型约定

* + 1. Html.ActionLink() 应用实例

@Html.ActionLink("Link to Sales", "about", "home",

new RouteValueDictionary { { "idd", 100 }, { "user", "Will+liu" } },

new Dictionary<string, object> { { "style", "color:red;font-weight:700;font-size:2em;" } }

)

<a  style="color:red;font-weight:700;font-size:2em;"

href="/home/about?idd=100&user=Will%2Bliu">

Link to Sales

</a>

需要注意的是：

* object routeValues , object htmlAttributes
* IDictionary<string, object> routeValues, IDictionary<string, object> htmlAttributes

以上是不能混合使用的， 否则生成很奇怪的超链接, 如下例子

@Html.ActionLink("Link to Sales", "about", "home",

new { idd=100 , user="Will liu", bdate=DateTime.Now }, - object

new Dictionary<string, object>{{ "style", "color:red;font-weight:700;font-size:2em;"}}

* Dictionary<string, object>

)

<a href="/home/about?idd=100&user=Will%20liu&bdate=07%2F31%2F2016%2012%3A54%3A55" values="System.Collections.Generic.Dictionary`2+ValueCollection[System.String,System.Object]" keys="System.Collections.Generic.Dictionary`2+KeyCollection[System.String,System.Object]" count="1" comparer="System.Collections.Generic.GenericEqualityComparer`1[System.String]">Link to Sales</a>

正确做法：

@Html.ActionLink("Link to Sales", "about", "home",

new { idd=100 , user="Will liu", bdate=DateTime.Now }, - object

new { style= "color:red;font-weight:700;font-size:2em;" } - object

)

<a style="color:red;font-weight:700;font-size:2em;"

href="/home/about?idd=100&user=Will%20liu&bdate=07%2F31%2F2016%2012%3A56%3A48">

Link to Sales

</a>

* + 1. Html.ActionLink() 链接到外部域名

@Html.ActionLink("Link to Sales", "about", "home", "https", "www.sohu.com", "product\_1",

new { idd=100 , user="Will liu", bdate=DateTime.Now },

new { style= "color:red;font-weight:700;font-size:2em;" }

)

<a  style="color:red;font-weight:700;font-size:2em;"

href="**https**://**www.sohu.com**/**home**/**about**?**idd=100&user=Will%20liu&bdate=07%2F31%2F2016%2013%3A02%3A11**#**product\_1**">Link to Sales</a>

* 1. Html.RouteLink()

虽然RouteLink 和 ActionLink 遵从相同的模式，但是RouteLink只接受路由名称，而不能接受控制器名称和操作名称

* + 1. Html.RouteLink() - 11 个重载方法

1. Html.RouteLink(linkText, [o]routeValues)
2. Html.RouteLink(linkText, [d]routeValues)

@Html.RouteLink("route link for me", new { action = "about", idd = 100, name = "willliu" })

<a href="/company/Home/about?idd=100&name=willliu">route link for me</a>

我们来探讨一下链接是如何生成的？

路由表：

routes.MapRoute(

name: "realtor",

url: "realtor/{controller}/{action}/{id}",

defaults: new { controller = "Sales", action = "Index", id = UrlParameter.Optional }

);

routes.MapRoute(

name: "myroute",

url: "company/{controller}/{action}/{id}",

defaults: new {controller = "Sales", action = "Index", id = UrlParameter.Optional }

);

routes.MapRoute(

name: "Default",

url: "{controller}/{action}/{id}",

defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional

);

Route Table: <br />

@foreach(Route el in RouteTable.Routes)

{

<span>Url: @el.Url</span><br />

}

<br /><br />

Action Link:

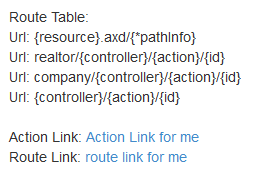
@Html.ActionLink("Action Link for me", "about", "home")

<br />

Route Link:

@Html.RouteLink("route link for me", new {action="about",idd=100,name="willliu"})

<br />



Action Link:

<a href="/**realtor**/home/about">Action Link for me</a>

<br>

Route Link:

<a href="/**realtor**/Home/about?idd=100&name=willliu">route link for me</a>

<br>

ActionLink先查找路由表第一个注册的：也就是上面"**realtor**/{controller}/{action}/{id}"

然后把 controller, action 替换，最终生成 "/**realtor**/home/about"

RouteLink也是一样，只是没有提供controller, 则使用当前的controller

如果我们输入：[http://localhost:21371/**Sales**/About](http://localhost:21371/Sales/About)

Action Link:

<a href="/**realtor**/home/about">Action Link for me</a> - controller, action 都固定

<br>

Route Link:

<a href="/**realtor**/**Sales**/about?idd=100&name=willliu">route link for me</a>

* Controller 没有设置所以取当前的Sales, Action 固定 about

Action Link:

@Html.ActionLink("Action Link for me","detail")

<br />

Route Link:

@Html.RouteLink("route link for me", new { idd = 100, name = "willliu" })

不指定controller, 但是在link里都会自动添加当前的controller.

Action Link:

<a href="/**realtor**/**Home**/detail">Action Link for me</a>

<br>

Route Link:

<a href="/**realtor**/**Home**?idd=100&name=willliu">route link for me</a>

如果我们输入: http://localhost:21371/Sales/**About**

Action Link:

<a href="/**realtor**/Sales/detail">Action Link for me</a>

<br>

Route Link:

<a href="/**realtor**/Sales/**About**?idd=100&name=willliu">route link for me</a>

1. Html.RouteLink(linkText, routeName)

凡是带有routeName 参数的方法，都是ActionLink所不没有的

Route Table:  
Url: {resource}.axd/{\*pathInfo}  
Url: realtor/{controller}/{action}/{id}  
Url: company/{controller}/{action}/{id}  
Url: {controller}/{action}/{id}

Action Link:

@Html.ActionLink("Action Link for me","detail")

<br />

Route Link:

@Html.RouteLink("route link for me", "myroute")

Action Link:

<a href="/realtor/Home/detail">Action Link for me</a>

<br>

Route Link:

<a href="/**company**/Home">route link for me</a> - company/{controller}/{action}/{id}

<br>

**注意：**

1. 虽然没有提供controller，但是并没有使用默认的Sales，而是使用当前Home
2. RouteLink 可以提供routeName 来使用指定的路由表， 这是ActionLink 所没有的
3. Html.RouteLink(linkText, [o]routeValues, [o]htmlAttributes)
4. Html.RouteLink(linkText, [d]routeValues, [d]htmlAttributes)

@Html.RouteLink("route link for me",

new { idd = 100, action = "about", controller = "school" },

new { style = "color:blue;font-size:2em;" } )

<a style="color:blue;font-size:2em;" href="**/realtor/school/about**?idd=100">route link for me</a>



1. Html.RouteLink(linkText, routeName, [0]routeValues)
2. Html.RouteLink(linkText, routeName, [d]routeValues)
3. Html.RouteLink(linkText, routeName, [0]routeValues, [o]htmlAttributes)
4. Html.RouteLink(linkText, routeName, [d]routeValues, [d]htmlAttributes)

@Html.RouteLink("route link for me","myroute" ,

new { idd = 100, action = "about", controller = "school" },

new { style = "color:blue;font-size:2em;" } )

Route Link:

<a style="color:blue;font-size:2em;" href="/company/school/about?idd=100">route link for me</a>



1. Html.RouteLink( linkText, routeName,

protocol, hostName, fragment,

[o]routeValues, [o]htmlAttributes)

1. Html.RouteLink( linkText, routeName,

protocol, hostName, fragment,

[d]routeValues, [d]htmlAttributes)

@Html.RouteLink("route link for me", "myroute",

"https", "www.mydomain.com", "product111",

new { idd = 100, action = "about", controller = "school" },

new { style = "color:blue;font-size:2em;" } )

Route Link:

<a  style="color:blue;font-size:2em;"

href="**https://www.mydomain.com/**company/school/about?idd=100**#product111**">

route link for me</a>

* 1. Html.Partial() & Html.RenderPartial()
     1. Html.Partial() – 4 个重载方法

Html.Partial(partialViewName)

Html.Partial(partialViewName, model)

Html.Partial(partialViewName, ViewData)

Html.Partial( partialViewName, object model, ViewData)

* + 1. Html.Partial() 和 Html.RenderPartial() 的区别

非常相似，RenderPartail() 不是返回字符串， 而是直接写入响应流

@Html.Partial() - 可以通过代码表达式来输出

@{ Html.RenderPartial(); } – 只能放在代码块里调用，分号必须；

Partial:<br />

@Html.Partial("myview")

<br />

Render Partial: <br />

@{ Html.RenderPartial("myview"); }

Partial view: myview.cshtml

在partial view 可以直接使用页面的数据模型

<span style="color:red;font-size:2em;">

This is a partial View

@ViewBag.myAddress

@Model.USER\_NAME

</span><br />

* + 1. Html.Partial()

ViewData["myAddress"] = 300;

public class User

{

[Display(Name ="用户名")]

public string USER\_NAME { get; set; }

public string homer { get; set; }

public string spouse\_name { get; set; }

public DateTime? birthdate { get; set; }

public int salary { get; set; }

public string **Street** { get; set; }

}

Partial View: myview.cshtml

<span style="color:red;font-size:2em;">

This is a partial View

@ViewBag.myAddress -

@ViewBag.good -

@Model.Street

</span><br />

@{

Address addr = new Address

{

id = 110,

Street = "Burlingtion",

City = "burnaby",

Suite = 504

};

}

@Html.Partial("myview", addr,

new ViewDataDictionary(this.ViewData) { { "good", "goodday" } } )

* 传递新的 Model,并且扩展现有的ViewData

<br />

@Html.Partial("myview", Model, ViewData)

* 如果传递现有的Model, ViewData， 则可以省略， 默认会传进去

<br />

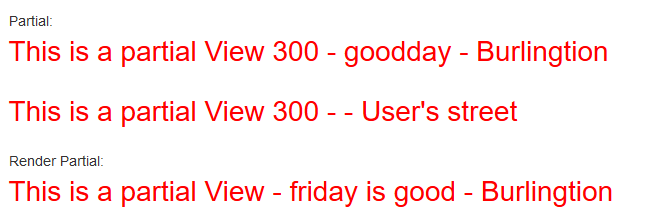
Render Partial: <br />

@{ Html.RenderPartial("myview", addr,

new ViewDataDictionary { { "GOOD", "friday is good" } }); }

* 传递全新的 ViewData

<br />



* 1. Html.Action() & Html.RenderAction()
     1. Html.Action(actionName, controllerName, routeValues) - 6个重载

Html.Action() 是必须经过 Controller->Model->View最后输出

所以通常是返回 partial view

提示：

当前页面的Model, ViewData 不能被共享使用， Html.Action ()所加载的页面是独立的数据模型，必须由它所在的controller 来提供数据模型.

* + 1. Html.Action() 也可以加载 View页面或者Partial View 页面， 加载完

插入到当前语句的位置

Home/About Page:<br />

@Html.Action("about","home")

* 没有将主板Layout 加载, 因为和页面是使用同一个主板文件

<br />

<br />

Sales detail View:<br />

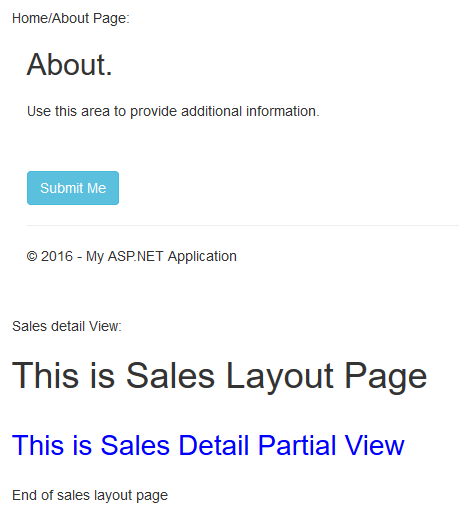
@Html.Action("detail", "sales")

* 连同主板Layout 一起加载进来

<br />

<br />

上面的都不是partial view



About.cshtml

@{

ViewBag.Title = "About";

Layout = "\_myLayout.cshtml";

}

<h2>@ViewBag.Title.</h2>

<h3>@ViewBag.Message</h3>

<p>Use this area to provide additional information.</p>

<br />

@using (Html.BeginForm(new { idd = 200, street = "Burlington" }))

{

@ViewBag.shipID

<br />

<input type="submit" name="btnSubmit" class="btn btn-info" value="Submit Me" />

}

\_myLayout.cshtml

<!DOCTYPE html>

<html>

<head>

<title>@Page.Title</title>

@RenderSection("head", required: false)

</head>

<body>

<h1>This is My Layout Page</h1>

<br />

@RenderBody()

<br />

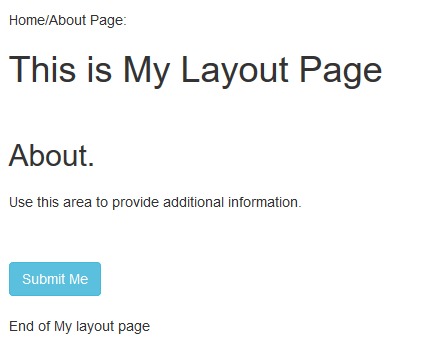
End of My layout page

</body>

</html>

Home/About Page:<br />

@Html.Action("about","home")



此时连同主板Layout一起输出

* + 1. [ChildActionOnly] 和 ControllerContext.IsChildAction

[ChildActionOnly]

public ActionResult detail()

{

Address addr = new Address

{

id = 333,

Street = "Canda Way",

child = ControllerContext.IsChildAction

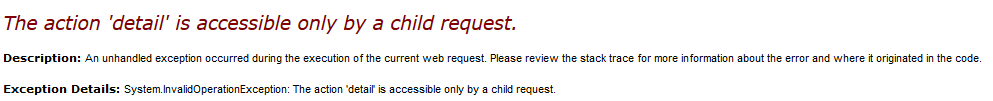
};

return PartialView(addr);

}

如果是由 Action() 或者RenderAction() 调用，则为True, 通过 URL调用则为False

[ChildActionOnly] 设置了这个属性， 通过URL 来调用，会出错



[ActionName("callDetail")]

public ActionResult detail()

{

Address addr = new Address

{

id = 333,

Street = "Canda Way",

child = ControllerContext.IsChildAction

};

return PartialView("detail", addr); - 记得一定要指明detail 否则跑去找callDetail

}

@Html.Action("callDetail", "sales")

<br />

<br />

@Html.ActionLink("Sales Detail", "callDetail", "sales")

都必须使用新的名称

* 1. Url辅助方法
     1. Url.Action() – 和ActionLink 非常相似，但是不返回锚点， 也就没

有htmlAttributes 参数, 返回字符串

* + 1. Url.Content() – 把相对路径转为URL, 这个方法尤其网站是使用Application

时候会有用处.

* + 1. Url.RouteUrl() – 和RouteLink相似， 但不返回锚点

@Url.RouteUrl("myroute",new { controller = "sales", action = "detai" })

/company/sales/detail

* 1. Html.BeginRouteForm() – 主要可以指定路由表来生成URL

@{Html.BeginRouteForm("myroute");}

<form method="post" action="/company/Home">

1. 表单Form和HTML 强类型辅助方法
   1. Html.LabelFor() & HTML.TextBoxFor()

public class User

{

[Key]

public int user\_id { get; set; }

[Display(Name ="用户名")]

public string USER\_NAME { get; set; }

[DisplayName("配偶姓名")]

public string spouse\_name { get; set; }

public DateTime? birthdate { get; set; }

public int salary { get; set; }

public int? home { get; set; }

}

@{Html.BeginForm();}

<br />

@Html.Label("user\_name") : @Html.TextBox("user\_name")<br />

@Html.LabelFor(m=>m.USER\_NAME) : @Html.TextBoxFor(m=>m.USER\_NAME)<br />

You input user: @Request.Form["user\_name"] Model: @Model.USER\_NAME<br />

<br />

@Html.Label("spouse\_name"): @Html.TextBoxFor(m=>m.spouse\_name)<br />

You input Spouse: @Request.Form["spouse\_name"] Model: @Model.spouse\_name<br />

<input type="submit" name="btnSubmit" class="btn btn-info" value="Submit Me" />

@{ Html.EndForm(); }

public ActionResult Index(User userObj)

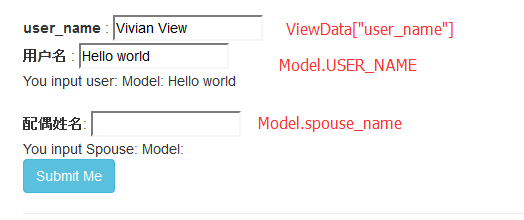
{

ViewData["User\_Name"] = "Vivian View";

userObj.USER\_NAME = "Hello world";

return View(userObj);

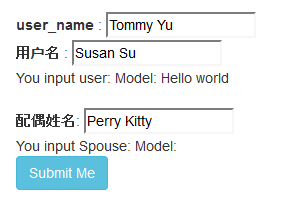
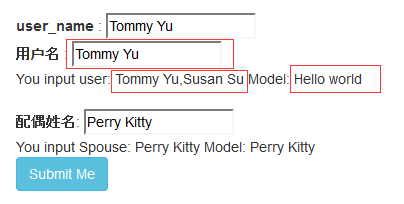
}



使用强类型的对于初始化数据是有好处的, 可以不受其他 ViewData 同名模型影响，准确绑定。

但是当我们提交表单时：

如上例子： 由于存在两个同名“user\_name”，只有第一个元素会起决定作用

提交以后

我们可以看到， 虽然 Model.USER\_NAME 一直是固定赋值“Hello World”， 但是提交以后则取值于提交的数据

View <Input value> -> Submit -> Controller (修改模型的值 change value) -> View <Input value>

* + 1. Html.LabelFor() – 会使用属性值[Display(Name ="")]或者 [DisplayName("配偶姓名")]
    2. Html.DisplayNameFor() - 显式纯文本的模型名称，或者是属性名称
    3. Html.TextAreaFor()
    4. Html.PasswordFor()
    5. Html.CheckBoxFor()
    6. Html.RadioButtonFor()
    7. Html.DropDownListFor()
    8. Html.ListBoxFor()
    9. Html.HiddenFor()
    10. Html.IdFor()
    11. Html.NameFor()
    12. Html.ValueFor()
    13. Html.DisplayTextFor() – 显式纯文本的模型数据值

不受Form 提交的影响，controller 里修改的值，也会反应到View的输出

* + 1. Html.ValidationMessageFor()

同非强类型的方法基本一致, 唯一就是准确绑定

@{Html.BeginForm();}

<br />

@Html.Label("user\_name","My User") : @Html.TextBox("user\_name")<br />

@Html.DisplayText("user\_name")<br />

@Html.DisplayNameFor(m => m.USER\_NAME) : @Html.TextBoxFor(m => m.USER\_NAME)<br />

@Html.DisplayTextFor(m => m.USER\_NAME)<br />

@Html.Value("user\_name") - @Html.ValueFor(m=>m.USER\_NAME)<br />

You input user: @Request.Form["user\_name"] Model: @Model.USER\_NAME<br />

<br />

@Html.Label("spouse\_name"): @Html.TextBoxFor(m=>m.spouse\_name)<br />

You input Spouse: @Request.Form["spouse\_name"] Model: @Model.spouse\_name<br />

@Html.DisplayText("spouse\_name") - @Html.DisplayTextFor(m=>m.spouse\_name)<br />

<input type="submit" name="btnSubmit" class="btn btn-info" value="Submit Me" />

@{ Html.EndForm(); }

public ActionResult Index(User userObj)

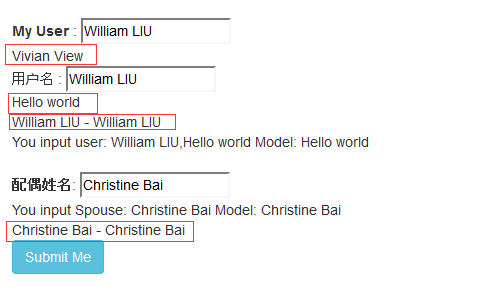
{

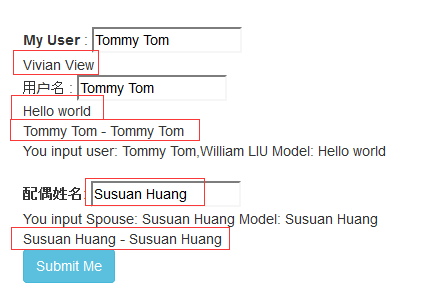
ViewData["User\_Name"] = "Vivian View";

userObj.USER\_NAME = "Hello world";

return View(userObj);

}





* 1. Html.DisplayFor() & Html.EditorFor()

首先参考字段的Metadata元数据来推断输出的格式和input type

主要是参考：[DataType(DataType.EmailAddress)] , 如果没有设置则参考C# 的类型

public class User

{

[Key]

public int user\_id { get; set; }

[Display(Name ="用户名")]

public string USER\_NAME { get; set; }

[Display(Name ="Email Addres")]

[DataType(DataType.EmailAddress)]

public string email { get; set; }

[Display(Name = "生日快乐")]

[DataType(DataType.Date)]

public DateTime birthdate { get; set; }

public DateTime hiredate { get; set; }

[DataType(DataType.Currency)]

public int salary { get; set; }

public int? home { get; set; }

}

@{Html.BeginForm();}

@Html.ValidationMessageFor(m=>m.email)

<br />

@Html.Label("user\_name","My User") : @Html.TextBox("user\_name")<br />

DisplayFor: @Html.DisplayFor(m=>m.USER\_NAME) Display:@Html.Display("user\_name")<br />

<br />

@Html.DisplayName("email"): @Html.EditorFor(m=>m.email)<br />

DisplayFor: @Html.DisplayFor(m=>m.email) Display: @Html.Display("email") <br />

<br />

@Html.Label("birthdate"): @Html.EditorFor(m => m.birthdate)<br />

DisplyFor: @Html.DisplayFor(m => m.birthdate) Display: @Html.Display("birthdate")<br />

<br />

@Html.DisplayNameFor(m => m.hiredate): @Html.EditorFor(m => m.hiredate)<br />

DisplyFor: @Html.DisplayFor(m => m.hiredate) Display: @Html.Display("hiredate")<br />

<br />

@Html.DisplayNameFor(m => m.salary): @Html.EditorFor(m => m.salary)<br />

DisplyFor: @Html.DisplayFor(m => m.salary) Display: @Html.Display("salary")<br />

<br />

<input type="submit" name="btnSubmit" class="btn btn-info" value="Submit Me" />

@{ Html.EndForm(); }

<form method="post" action="/">

<input id="user\_name" type="text" value="Vivian View" name="user\_name">

DisplayFor: Hello world Display:Vivian View

Email Addres:

<input id="email" class="text-box single-line" **type="email"** value="" name="email">

DisplayFor:

<a href="mailto:"></a>

Display:

<a href="mailto:"></a>

<label for="birthdate">生日快乐</label>

<input id="birthdate" class="text-box single-line" **type="date"** value="1/1/0001" name="birthdate" data-val-required="The 生日快乐 field is required." data-val-date="The field 生日快乐 must be a date." data-val="true">

DisplyFor: 1/1/0001 Display: 1/1/0001

hiredate:

<input id="hiredate" class="text-box single-line" **type="datetime"** value="1/1/0001 12:00:00 AM" name="hiredate" data-val-required="The hiredate field is required." data-val-date="The field hiredate must be a date." data-val="true">

DisplyFor: 1/1/0001 12:00:00 AM Display: 1/1/0001 12:00:00 AM

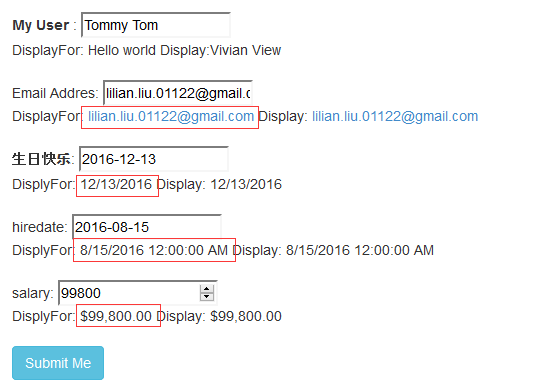
salary:

<input id="salary" class="text-box single-line" type="number" value="0" name="salary" data-val-required="The salary field is required." data-val-number="The field salary must be a number." data-val="true">

DisplyFor: $0.00 Display: $0.00

<input class="btn btn-info" type="submit" value="Submit Me" name="btnSubmit">

</form>



总结：

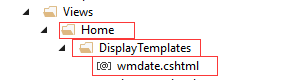
Html.DisplayFor() & Html.EditorFor() 会自动根据数据模型的Metadata来寻找系统预定义的模板来输出

* 1. Html.Display() & Html.DisplayFor() 自定义显式模板

根据命名约定:

1. 显示模板的名称就是文件的名称，templateName.cshtml
2. 按下面的顺序寻找模板：

* /Views/ControllerName/DisplayTemplates/templateName.cshtml
* /Views/Shared/DisplayTemplates/templateName.cshtml
* If not found, using system default matched template



\_detailLayout.cshtml :

<!DOCTYPE html>

<html>

<head>

<title>@Page.Title</title>

@RenderSection("head", required: false)

</head>

<body>

<h1>This is Sales Detail WMView Layout Page</h1>

@RenderBody()

<h1>End of sales Detail WMView layout page</h1>

<br />

</body>

</html>

**wmdate**.cshtml:

@model DateTime

@{

Layout = "~/Views/sales/\_detailLayout.cshtml";

}

<span style="color:blue;font-weight:700;">

Controller Home Display Template wmdate :

@string.Format("{0}-{1}-{2} {3}:{4} Asian China",

Model.Year, Model.Month, Model.Day, Model.Hour, Model.Minute)

Other: @ViewBag.ctrlGood ViewBag: @ViewBag.addGood

</span><br />

主板页面也被输出, 在display Template里可以共享使用当前页面的 ViewData的数据， 但是 Model 已经被

DisplayFor() 字段所使用@model DateTime

可以额外传递其他数据，并合并到新的ViewData里供应给模板使用

public ActionResult Index(User userObj)

{

ViewData["ctrlGood"] = "Controller ViewData";

return View(userObj);

}

/home/index

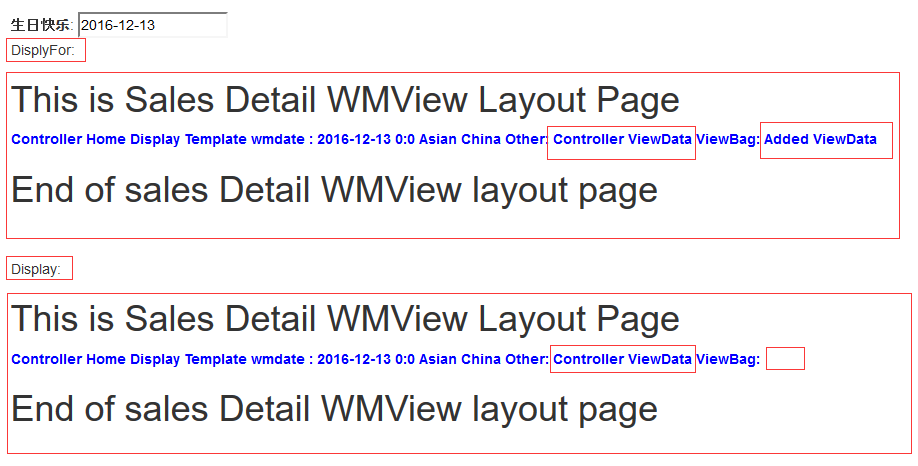
@Html.Label("birthdate"): @Html.EditorFor(m => m.birthdate)<br />

DisplyFor: @Html.DisplayFor(m => m.birthdate, "**wmdate**", new { addGood="Added ViewData"})

- 可以直接指定display template, 并且额外添加新的ViewData, 合并到当前ViewData传递到display template

<br />

Display: @Html.Display("birthdate", "wmdate")<br />



还可以通过数据模型来指定模板

[Display(Name = "生日快乐")]

[DataType(DataType.Date)]

[UIHint("wmdate")] - 将显示模板指定为 “wmdate”

public DateTime birthdate { get; set; }

@Html.Label("birthdate"): @Html.EditorFor(m => m.birthdate)<br />

DisplyFor: @Html.DisplayFor(m => m.birthdate, new { addGood="Added ViewData"})

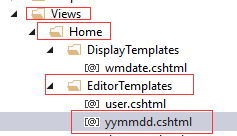
<br />

Display: @Html.Display("birthdate")<br />

同样的原理， 可以创建整个Model 模型的显示模板

Html.DisplayForModel("user") – 传入的是整个 ViewData.Model

* 1. Html.Editor() & Html.EditorFor() 自定义编辑模板



yymmdd.cshtml:

@model DateTime

<input type="date"

Id ="@Html.IdForModel()"

Name ="@Html.NameForModel()"

Value ="@Model.ToString("yy-MM-dd HH:mm")"

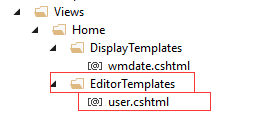
style="color:red; width:200px; font-weight:700" />

@Html.DisplayNameFor(m => m.hiredate): @Html.EditorFor(m => m.hiredate, "yymmdd")<br />

DisplyFor: @Html.DisplayFor(m => m.hiredate) Display: @Html.Display("hiredate")<br />



同样的原理， 可以创建整个Model 模型的编辑模板



user.cshtml:

@model User

user: @Html.EditorFor(m=>m.USER\_NAME)<br />

Salary: @Html.EditorFor(m=>m.salary )<br />

@Html.EditorForModel("user")



Display, DisplayFor, Editor, EditorFor的CSS 设置：

<span>

@Html.DisplayFor(p=>p.navi.keywords,

new { htmlAttributes = new { style="color:red;font-size:2em;" }})

</span>

@Html.EditorFor(p => p.navi.keywords,

new { htmlAttributes = new { style = "width:120px;color:blue;font-weight:bold;" } })



<span>Hotel : </span>

<input id="navi\_keywords" class="text-box single-line" type="text" value="Hotel"

style="width:120px;color:blue;font-weight:bold;" name="navi.keywords">

Display, DisplayFor 只是输出纯文本， 所以没有CSS

Editor, EditorFor 虽然没有这个重载方法， 使用

new { htmlAttributes = new { style = "width:120px;color:blue;font-weight:bold;" } }

* 1. 自定义Html 辅助方法
     1. 使用的是C#的扩展方法来自定义Html辅助方法。在此介绍一下C#的扩展方法

MSDN对扩展方法的描述：

扩展方法使您能够向现有类型“添加”方法，而无需创建新的派生类型、重新编译或以其他方式修

改原始类型。

扩展方法被定义为静态方法，但它们是通过实例方法语法进行调用的。 它们的第一个参数指定该

方法作用于哪个类型，并且该参数以 this 修饰符为前缀。

给出实例：

public static class stringExtents

{

public static int[] **StringToArray**(this string \_self, string[] s)

{

int[] ret = new int[0];

if (!string.IsNullOrEmpty(\_self))

{

int temp = 0;

string[] sarr = \_self.Split(s, StringSplitOptions.None);

ret = sarr.Select(p => int.TryParse(p, out temp)?int.Parse(p):0).ToArray();

}

return ret;

}

public static int **toInt**(this string \_self)

{

int ret;

int.TryParse(\_self, out ret);

return ret;

}

public static string **IntToString**(this int \_self)

{

return string.Format("IntToString: {0}", \_self);

}

}

string a = "10,20,30,,abb,,50";

int[] b = a.**StringToArray**(new string[] { "," });

string int\_string = string.Join(";", b.Select(p => "[" + p + "]").ToArray());

Console.WriteLine("Int[] :" + int\_string);

string c = "3558";

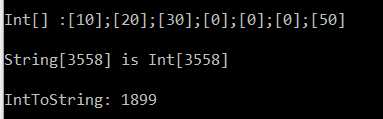
int d = c.**toInt**();

Console.WriteLine("String[{0}] is Int[{1}]", c, d);

int e = 1899;

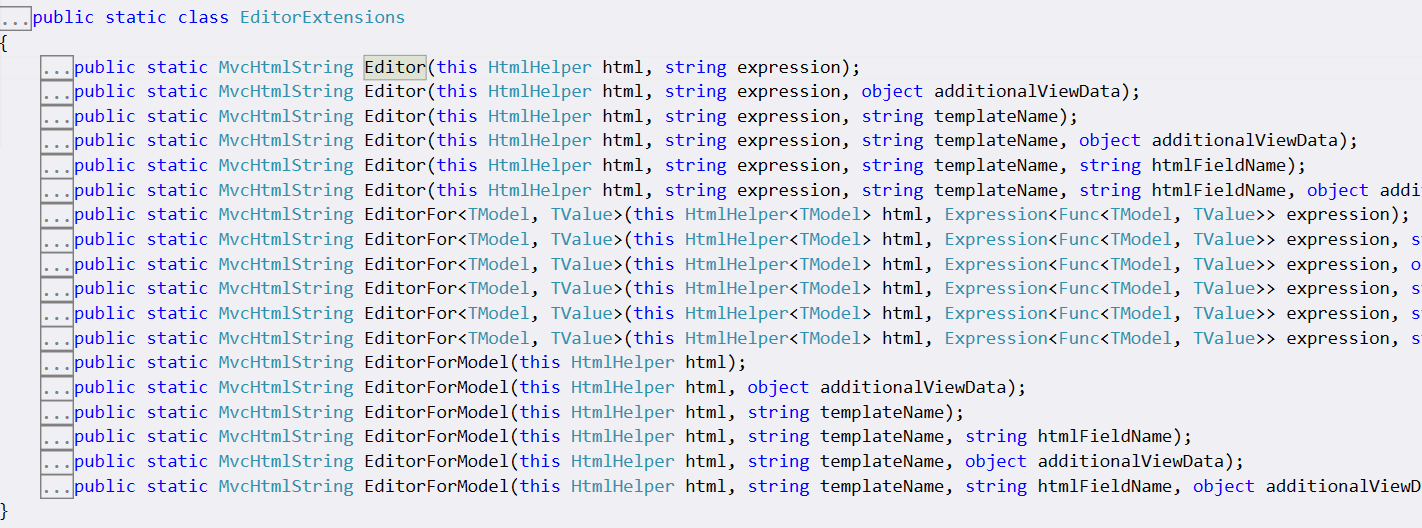
string f = e.**IntToString**();

Console.WriteLine(f);



可以看到C#扩展方法非常简单却很实用, 无需创建新的派生类或其他类。对于代码逻辑和维护带来极大方便。

* + 1. Html 系统自带的辅助方法：



* + 1. Html自定义辅助方法： 在静态类里定义静态方法：静态方法通常返回MvcHtmlString

public static class anyNameForExtensions - 容器静态类，名称其实无所谓

{

public static **MvcHtmlString** CheckBoxList(this HtmlHelper helper, string name, SelectList selectList)

{

StringBuilder strBuilder = new StringBuilder();

int[] selected = helper.ViewData[name] as int[];

if (selected == null) - prevent throw null value error

{

int[] selectedValues = selectList.SelectedValue as int[];

if (selectedValues == null) selectedValues = new int[0];

selected = selectedValues; - Set to Default Selected

}

foreach(SelectListItem el in selectList)

{

int el\_value = 0;

int.TryParse(el.Value, out el\_value); - string to int

string ck\_str = string.Empty;

if(selected.Contains(el\_value)) ck\_str = "checked='checked'"; - Selected

string temp = string.Format(@"<label style='margin-right:25px;'>

<input type='checkbox' id='{0}\_{1}' name='{2}' value='{3}' {4} /> {5}

</label>", name, el.Value, name, el.Value, ck\_str, el.Text

);

strBuilder.Append(temp);

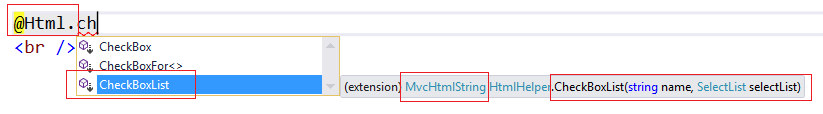
}

return new MvcHtmlString(strBuilder.ToString());

}

}

自定义完扩展方法后，Html.CheckBoxList() 就附加上了



public ActionResult Index(User userObj, int[] myAddress)

{

HashSet<Address> addrList = new HashSet<Address>

{

new Address { id=100, Street="AAAA AAA" },

new Address { id=200, Street="BBBB BBB" },

new Address { id=255, Street="CCC CCC" },

new Address { id=300, Street="DDD DDDD" },

new Address { id=400, Street="EEE EEEE" }

};

SelectList addrSelectList = new SelectList(addrList, "id", "Street",

new int[] { 200, 300 });

addrSelectList.Where(p => p.Value == "200").Single().Selected = true;

addrSelectList.Where(p => p.Value == "400").Single().Selected = true;

ViewData["shipAddress"] = addrSelectList;

ViewData["myAddress"] = myAddress;

return View(userObj);

}

@Html.CheckBoxList("myAddress", ViewData["shipAddress"] as SelectList)

<label style="margin-right:25px;">

<input id="myAddress\_100" type="checkbox" value="100" name="myAddress">

AAAA AAA

</label>

<label style="margin-right:25px;">

<input id="myAddress\_200" type="checkbox" checked="checked" value="200" name="myAddress">

BBBB BBB

</label>

<label style="margin-right:25px;">

<input id="myAddress\_255" type="checkbox" value="255" name="myAddress">

CCC CCC

</label>

<label style="margin-right:25px;">

<input id="myAddress\_300" type="checkbox" checked="checked" value="300" name="myAddress">

DDD DDDD

</label>

<label style="margin-right:25px;">

<input id="myAddress\_400" type="checkbox" value="400" name="myAddress"> - 400并没有被选中

EEE EEEE

</label>



全选以后，提交， 仍然显示正确，证明赋值成功



using System.Linq.Expressions;

public static MvcHtmlString CheckBoxListFor<TModel,TValue>(this HtmlHelper<TModel> helper, Expression<Func<TModel, TValue>> expression, SelectList selectList)

{

StringBuilder strBuilder = new StringBuilder();

return new MvcHtmlString(strBuilder.ToString());

}

需要熟悉 C# Expression, 请参考TextBoxFor source code 会有很大帮助

主要技术要点：

1. ModelMetadata.FromLambdaExpression(expression, helper.ViewData);
2. string fullName = ExpressionHelper.GetExpressionText(expression); - 获取全名称如：user.home
3. int[] selected = metadata.Model as int[]; - 获取模型数据 p=>p.user.home int[]类型

// ViewModel Definition

public class GUser

{

public User user { get; set; }

public SelectList addressList;

public GUser()

{

this.user = new User();

HashSet<Address> addrList = new HashSet<Address>

{

new Address { id=100, Street="AAAA AAA" },

new Address { id=200, Street="BBBB BBB" },

new Address { id=255, Street="CCC CCC" },

new Address { id=300, Street="DDD DDDD" },

new Address { id=400, Street="EEE EEEE" }

};

SelectList addrSelectList = new SelectList(addrList, "id", "Street",

new int[] { 200, 300 });

this.addressList = addrSelectList;

}

}

// Entity Definition

public class User

{

[Display(Name ="用户名")]

public string USER\_NAME { get; set; }

public int[] home { get; set; }

}

// Entity Definition

public class Address

{

public int id { get; set; }

public string Street { get; set; }

}

// Controlloer - Action

public ActionResult Index(GUser userObj)

{

return View(userObj);

}

// Html自定义辅助方法 CheckBoxList() & CheckBoxListFor()

public static class anyNameForExtensions

{

public static MvcHtmlString CheckBoxList(this HtmlHelper helper, string name, SelectList selectList)

{

StringBuilder strBuilder = new StringBuilder();

int[] selected = helper.ViewData[name] as int[];

if (selected == null)

{

int[] selectedValues = selectList.SelectedValue as int[];

if (selectedValues == null) selectedValues = new int[0];

selected = selectedValues;

}

foreach(SelectListItem el in selectList)

{

int el\_value = 0;

int.TryParse(el.Value, out el\_value);

string ck\_str = string.Empty;

if(selected.Contains(el\_value)) ck\_str = "checked='checked'";

string temp = string.Format(@"<label style='margin-right:25px;'>

<input type='checkbox' id='{0}\_{1}' name='{2}' value='{3}' {4} /> {5}</label>",

name, el.Value, name, el.Value, ck\_str, el.Text

);

strBuilder.Append(temp);

}

return new MvcHtmlString(strBuilder.ToString());

}

public static MvcHtmlString CheckBoxListFor<TModel,TValue>(this HtmlHelper<TModel> helper, Expression<Func<TModel, TValue>> expression, SelectList selectList)

{

StringBuilder strBuilder = new StringBuilder();

ModelMetadata metadata = ModelMetadata.FromLambdaExpression(expression, helper.ViewData);

string fullName = ExpressionHelper.GetExpressionText(expression);

int[] selected = metadata.Model as int[];

if (selected == null)

{

int[] selectedValues = selectList.SelectedValue as int[];

if (selectedValues == null) selectedValues = new int[0];

selected = selectedValues;

}

foreach (SelectListItem el in selectList)

{

int el\_value = 0;

int.TryParse(el.Value, out el\_value);

string ck\_str = string.Empty;

if (selected.Contains(el\_value)) ck\_str = "checked='checked'";

string temp = string.Format(@"<label style='margin-right:25px;'>

<input type='checkbox' id='{0}\_{1}' name='{2}' value='{3}' {4} /> {5}</label>",

fullName, el.Value, fullName, el.Value, ck\_str, el.Text

);

strBuilder.Append(temp);

}

return new MvcHtmlString(strBuilder.ToString());

}

public static object GetModelStateValue(this HtmlHelper htmlHelper, string key, Type destinationType)

{

ModelState modelState;

if (htmlHelper.ViewData.ModelState.TryGetValue(key, out modelState))

{

if (modelState.Value != null)

{

return modelState.Value.ConvertTo(destinationType, null /\* culture \*/);

}

}

return null;

}

}

// Index.cshtml:

@model Guser

@{Html.BeginForm();}

<br />

@Html.CheckBoxListFor(m => m.user.home, Model.addressList )

<br />

@Html.TextBoxFor(m=>m.user.USER\_NAME)

<br />

<input type="submit" name="btnSubmit" class="btn btn-info" value="Submit Me" />

@{ Html.EndForm(); }

1. Bundles文件包：ScriptBundle 和 StyleBundle
   1. Bundles 工作原理

Bundles用于打包CSS和javascript脚本文件，优化对它们的组织管理.

protected void Application\_Start()

{

AreaRegistration.RegisterAllAreas();

FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);

RouteConfig.RegisterRoutes(RouteTable.Routes);

**BundleConfig.RegisterBundles(BundleTable.Bundles); -定义了静态方法来注册 BundleTable.Bundles**

}

public class BundleConfig

{

public static void RegisterBundles(BundleCollection bundles)

{

bundles.Add(new ScriptBundle("~/bundles/jquery").Include(

"~/Scripts/jquery-{version}.js"));

bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(

"~/Scripts/jquery.validate\*"));

bundles.Add(new ScriptBundle("~/bundles/modernizr").Include(

"~/Scripts/modernizr-\*"));

bundles.Add(new ScriptBundle("~/bundles/bootstrap").Include(

"~/Scripts/bootstrap.js",

"~/Scripts/respond.js"));

bundles.Add(new StyleBundle("~/Content/css").Include(

"~/Content/bootstrap.css",

"~/Content/site.css"));

bundles.Add(new ScriptBundle("~/lwh/mylwh").Include(

"~/Scripts/lwh.test.{version}.js",

"~/Scripts/lwh/lwh.gm.js"

));

bundles.Add(new StyleBundle("~/lwh/mycss").Include(

"~/Content/blue/lwh.good.css",

"~/Content/blue/lwh.test.{version}.css"

));

}

}

创建和注册 Bundles

1. 分成两类：ScriptBundle 和 StyleBundle
2. Include ：为文件包Bundle 包括实际需要的 js 文件，或者是 CSS 文件
3. Bundle 的名称，和实际的文件全路径名称， 都必须是以 “~”开头， 否则出错

Bundle名称： 是文件包的逻辑名称，当需要引用时使用的是逻辑名称， 和实际物理位置无关

@Scripts.Render("~/lwh/mylwh")

@Styles.Render("~/lwh/mycss")

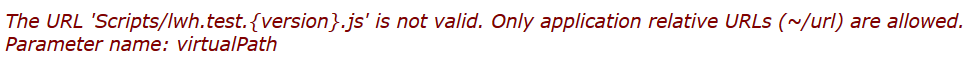
Include 文件物理位置名称： 文件的物理位置名称， 也必须使用“～”来映射正确的文件位置

bundles.Add(new ScriptBundle("~/lwh/mylwh").Include(

"Scripts/lwh.test.{version}.js",

"Scripts/lwh/lwh.gm.js"

));



bundles.Add(new ScriptBundle("lwh/mylwh").Include(

"~/Scripts/lwh.test.{version}.js",

"~/Scripts/lwh/lwh.gm.js"

));



1. 包含的文件目录必须存在，如果目录不存在则出错， 而包含的文件可以不存在， 不存在则当引用时不载入

bundles.Add(new ScriptBundle("~/lwh/mylwh").Include(

"~/**Script**/lwh.test.{version}.js",

"~/Scripts/lwh/lwh.gm.js"

));



1. Bundle 文件版本的约定： 可以使用 {version} 来自动包含不同的版本, 如果同一个文件的多个版本同时存在于目录，则系统会将所有的版本加载到页面， 这样存在潜在的冲突危险， 所以应该保留最新版本，就版本应该删除。

关于版本的规定， 必须是2位以上，如： x.x x.x.x 不能是 x-x x-x-x

// lwh.test.1.1.js

function show1() {

alert("js show1 Version - 1.1");

}

show1();

// lwh.test.2.0.js

function show1() {

alert("js show1 Version-2.0");

}

show1();

bundles.Add(new ScriptBundle("~/lwh/mylwh").Include(

"~/Scripts/lwh.test.{version}.js",

"~/Scripts/lwh/lwh.gm.js"

));

bundles.Add(new StyleBundle("~/lwh/mycss").Include(

"~/Content/blue/lwh.good.css",

"~/Content/blue/lwh.test.{version}.css"

));

@Scripts.Render("~/lwh/mylwh")

@Styles.Render("~/lwh/mycss")

1. 在需要的页面上引用文件包 Bundle

如： 主板页面： ~/Views/Shared/\_Layout.cshtml

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>@ViewBag.Title - My ASP.NET Application</title>

@Scripts.Render("~/bundles/jquery")

@Scripts.Render("~/bundles/bootstrap")

@Scripts.Render("~/bundles/modernizr")

@Styles.Render("~/Content/css")

@RenderSection("scripts", required: false)

</head>

如： About.cshtml

<div class="lwh">

Hello World

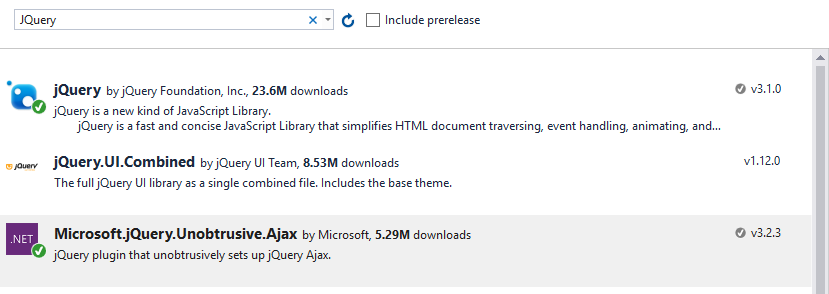
</div>

@Scripts.Render("~/lwh/mylwh")

@Styles.Render("~/lwh/mycss")

1. @Ajax 辅助方法
   1. @Ajax 辅助方法需要载入 jquery.unobtrusive-ajax.js 文件的支持

通过NuGet Package 获取上述文件



* 1. 使用@Ajax 功能的前提：

1. @Ajax 辅助方法是使用JQuery 的Ajax 方法， 所以必须载入Jquery

bundles.Add(new ScriptBundle("~/bundles/jquery").Include(

"~/Scripts/jquery-{version}.js"

));

@Scripts.Render("~/bundles/jquery")

1. @Ajax辅助方法必须要载入jquery.unobtrusive-ajax.js

@section scripts {

<script src="@Url.Content("~/Scripts/jquery.unobtrusive-ajax.js")"></script>

}

1. @Ajax 辅助方法必须被激活,

web.config

<appSettings>

<add key="webpages:Version" value="3.0.0.0" />

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

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

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

</appSettings>

也可以在指定页面，单独控制开关

@{ Html.EnableUnobtrusiveJavaScript(); }

* 1. @Ajax 超链接功能
     1. @Ajax.ActionLink() 比 @Html.ActionLink()多出一个AjaxOptions参数

AjaxOptions:

|  |  |  |
| --- | --- | --- |
|  | **Name** | **Description** |
| System_CAPS_pubproperty | [AllowCache](https://msdn.microsoft.com/en-us/library/system.web.mvc.ajax.ajaxoptions.allowcache.aspx#P:System.Web.Mvc.Ajax.AjaxOptions.AllowCache) |  |
| System_CAPS_pubproperty | [Confirm](https://msdn.microsoft.com/en-us/library/system.web.mvc.ajax.ajaxoptions.confirm.aspx#P:System.Web.Mvc.Ajax.AjaxOptions.Confirm) | Gets or sets the message to display in a confirmation window before a request is submitted. |
| System_CAPS_pubproperty | [HttpMethod](https://msdn.microsoft.com/en-us/library/system.web.mvc.ajax.ajaxoptions.httpmethod.aspx#P:System.Web.Mvc.Ajax.AjaxOptions.HttpMethod) | Gets or sets the HTTP request method ("Get" or "Post"). Default “Get” |
| System_CAPS_pubproperty | [InsertionMode](https://msdn.microsoft.com/en-us/library/system.web.mvc.ajax.ajaxoptions.insertionmode.aspx#P:System.Web.Mvc.Ajax.AjaxOptions.InsertionMode) | Gets or sets the mode that specifies how to insert the response into the target DOM element. |
| System_CAPS_pubproperty | [LoadingElementDuration](https://msdn.microsoft.com/en-us/library/system.web.mvc.ajax.ajaxoptions.loadingelementduration.aspx#P:System.Web.Mvc.Ajax.AjaxOptions.LoadingElementDuration) | Gets or sets a value, in milliseconds, that controls the duration of the animation when showing or hiding the loading element. |
| System_CAPS_pubproperty | [LoadingElementId](https://msdn.microsoft.com/en-us/library/system.web.mvc.ajax.ajaxoptions.loadingelementid.aspx#P:System.Web.Mvc.Ajax.AjaxOptions.LoadingElementId) | Gets or sets the **id** attribute of an HTML element that is displayed while the Ajax function is loading. |
| System_CAPS_pubproperty | [OnBegin](https://msdn.microsoft.com/en-us/library/system.web.mvc.ajax.ajaxoptions.onbegin.aspx#P:System.Web.Mvc.Ajax.AjaxOptions.OnBegin) | Gets or sets the name of the JavaScript function to call immediately before the page is updated. |
| System_CAPS_pubproperty | [OnComplete](https://msdn.microsoft.com/en-us/library/system.web.mvc.ajax.ajaxoptions.oncomplete.aspx#P:System.Web.Mvc.Ajax.AjaxOptions.OnComplete) | Gets or sets the JavaScript function to call when response data has been instantiated but before the page is updated. |
| System_CAPS_pubproperty | [OnFailure](https://msdn.microsoft.com/en-us/library/system.web.mvc.ajax.ajaxoptions.onfailure.aspx#P:System.Web.Mvc.Ajax.AjaxOptions.OnFailure) | Gets or sets the JavaScript function to call if the page update fails. |
| System_CAPS_pubproperty | [OnSuccess](https://msdn.microsoft.com/en-us/library/system.web.mvc.ajax.ajaxoptions.onsuccess.aspx#P:System.Web.Mvc.Ajax.AjaxOptions.OnSuccess) | Gets or sets the JavaScript function to call after the page is successfully updated. |
| System_CAPS_pubproperty | [UpdateTargetId](https://msdn.microsoft.com/en-us/library/system.web.mvc.ajax.ajaxoptions.updatetargetid.aspx#P:System.Web.Mvc.Ajax.AjaxOptions.UpdateTargetId) | Gets or sets the ID of the DOM element to update by using the response from the server. |
| System_CAPS_pubproperty | [Url](https://msdn.microsoft.com/en-us/library/system.web.mvc.ajax.ajaxoptions.url.aspx#P:System.Web.Mvc.Ajax.AjaxOptions.Url) | Gets or sets the URL to make the request to. |

public ActionResult getTime()

{

return Content(DateTime.Now.ToString());

}

@Ajax.ActionLink("Get Current Time", "getTime",

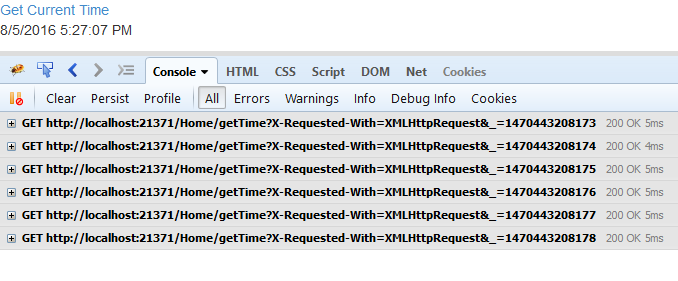
new AjaxOptions {UpdateTargetId= "current\_time", InsertionMode = InsertionMode.Replace })

<br />

<div id="current\_time">

Hello World

</div>



默认是Http.Get, 提交时会自动添加一个时间戳，以便生成不同的URL,每次点击都是不同URL

关于Ajax 缓存问题： 默认当Ajax 发出的请求URL 没有改变，当第一次点击请求完成以后， 第二次点击同样的网址，是不会向服务端发出请求的。

如果希望Ajax回传即时更新

@Ajax.ActionLink("Get Current Time", "getTime",

new AjaxOptions { UpdateTargetId= "current\_time",

**AllowCache**=true,

InsertionMode = InsertionMode.Replace,

HttpMethod="Post"

})<br />

<div id="current\_time">

Hello World

</div>

[OutputCache(**NoStore** = true, **Duration** = 5)]

public ActionResult getTime()

{

return Content(DateTime.Now.ToString());

}

控制缓存的参数有：**AllowCache；**  **NoStore; Duration**

1. **Duration** 是设置几秒钟以后点击可以更新: 需要即时更新，设置如下：

[OutputCache(NoStore = true, Duration = 0)]

public ActionResult getTime()

{

return Content(DateTime.Now.ToString());

}

AjaxOptions的其他参数：

@Ajax.ActionLink("Get Current Time", "getTime1",

new AjaxOptions {

UpdateTargetId = "current\_time",

AllowCache =true,

InsertionMode = InsertionMode.Replace,

HttpMethod ="Post",

Confirm="Are you sure to update?",

OnBegin="alert('OnBegin')",

OnComplete="alert('OnComplete')",

OnSuccess="alert('OnSuccess')",

OnFailure="alert('OnFailure')"

}

)

Event:

1. Success Case: Confirm – OnBegin – OnSuccess - OnComplete
2. Failure Case: Confirm – OnBegin – OnFailure - OnComplete
3. 以上Event 是前端Javascript的语言

@Ajax.ActionLink("Get Current Time", "getTime",

new AjaxOptions {

UpdateTargetId = "current\_time",

AllowCache =true,

InsertionMode = InsertionMode.Replace,

HttpMethod ="Post",

Confirm="Are you sure to update?",

OnBegin="on\_begin()", - 可以是 Javascript function

OnComplete="on\_done()", - 可以是 Javascript function

OnSuccess="on\_ok()", - 可以是 Javascript function

OnFailure="on\_fail()" - 可以是 Javascript function

}

)

<br />

<div id="current\_time">

Hello World

</div>

<script>

function on\_begin() {

alert("is Begin");

}

function on\_ok() {

alert("is Success");

}

function on\_done() {

alert("is Complete");

}

function on\_fail() {

alert("is Failure");

}

</script>

* + 1. @Ajax.RouteLink() 基本同上相似，只是可以使用指定的路由表
    2. Ajax.BeginForm()

写法之一：

@{ Ajax.BeginForm("ajaxAbout", "home", new AjaxOptions

{

UpdateTargetId= "ajax\_div",

HttpMethod = "post",

Confirm = "Are you sure submit?",

OnBegin = "on\_begin()",

OnComplete = "on\_done()",

OnSuccess = "on\_ok()",

OnFailure = "on\_fail()"

});

}

@Html.DisplayNameFor(p=>p.Suite) : @Html.EditorFor(p=>p.Suite)<br />

@Html.DisplayName("Street") : @Html.Editor("Street")<br />

@Html.DisplayName("City"): @Html.EditorFor(p=>p.City)<br />

<input type="submit" name="mysubmit" value="Submit" />

@{Html.EndForm();}

写法之二：

@using (Ajax.BeginForm("ajaxAbout", "home",

new AjaxOptions

{

UpdateTargetId = "ajax\_div",

HttpMethod = "post",

Confirm = "Are you sure submit?",

OnBegin = "on\_begin()",

OnComplete = "on\_done()",

OnSuccess = "on\_ok()",

OnFailure = "on\_fail()"

})

)

{

<span>@Html.DisplayNameFor(p => p.Suite) :</span> @Html.EditorFor(p => p.Suite) <br />

<span>@Html.DisplayNameFor(p => p.Street) :</span> @Html.Editor("Street") <br />

<span>@Html.DisplayNameFor(p => p.City) :</span> @Html.EditorFor(p => p.City) <br />

<input type="submit" name="mysubmit" value="Submit" />

}

public ActionResult ajaxAbout(Address myaddress)

{

//注意我们在此只清理 Suite 和 City 的 ModelState

ModelState.RemoveStateFor(myaddress, p => p.Suite);

ModelState.Remove("city");

//注意我们故意在此修改提交过来的值

myaddress.Suite += 10000;

myaddress.Street += " About Street";

myaddress.City += " About City";

return PartialView("ajaxAbout", myaddress);

}

ajaxAbout.cshtml:

@using WebApp7.Models

@model Address

<br />

@Html.DisplayNameFor(p => p.Suite): @Html.EditorFor(p => p.Suite)<br />

@Html.DisplayName("Street"): @Html.Editor("Street")<br />

@Html.DisplayName("City"): @Html.EditorFor(p => p.City)<br />

<br />

@Html.DisplayFor(p => p.Suite)<br />

@Html.DisplayFor(p => p.Street )<br />

@Html.DisplayFor(p => p.City )<br />



如果我们提交的数据在Controller里由于某些原因需要修改，

但是因为 ModelState 的值没有更新，里面的值还是 提交过来的值。  
但我们是没办法修改它里面的值的，所以我们必须清除它里面旧有的值。   
要清除很简单，只要一行 ModelState.Clear(); 但这是全部清除，如果不想全部清除，就用 ModelState.Remove("字段名称"); 就行了。

1. Data Annotation & Data Validation 数据注解和数据验证
   1. Data Annotation
      1. StringLength : ErrorMessage(name, max, min)

[StringLength(5, MinimumLength = 2)]

public string USER\_NAME { get; set; }



[StringLength(5, MinimumLength = 2, ErrorMessage = "{0} length between {2} and {1}")]

public string USER\_NAME { get; set; }



* + 1. Required : ErrorMessage(name)

[Required]

public string USER\_NAME { get; set; }



[Required(AllowEmptyStrings =true, ErrorMessage = "字段 {0} 不许空")]

public string USER\_NAME { get; set; }



* + 1. Range : ErrorMessage(name, min, max)

[Range(10,20, ErrorMessage ="{0} 必须在 {1} 和 {2}之间")]

public int salary { get; set; }



* + 1. RegularExpression : ErrorMessage(name, pattern)

[RegularExpression(@"^A[1-9]\*$")]

public string email { get; set; }



[RegularExpression(@"^A[1-9]\*$", ErrorMessage = "字段{0}格式{1}")]

public string email { get; set; }



* + 1. [DataType(DataType.xxx)] @Html.EditorFor() 是Client端的验证

Salary: @Html.EditorFor(m=>m.user.salary)

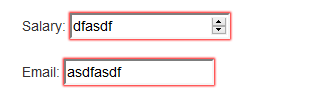
@Html.ValidationMessageFor(m=>m.user.salary )

<br />

<br />

Email: @Html.EditorFor(m => m.user.email )

@Html.ValidationMessageFor(m => m.user.email )<br />



* + 1. CustomValidation

public class stringValidate: ValidationAttribute

{

private string \_min;

private string \_max;

public stringValidate()

{

\_min = "a";

\_max = "z";

}

public stringValidate(string min, string max)

{

\_min = min;

\_max = max;

}

public override string FormatErrorMessage(string name)

{

//return "{0} good {1} bad {2}";

return string.Format("{0} not allowed {1} or {2}", name, \_min, \_max);

}

public override bool IsValid(object value)

{

string o = value as string;

if (string.IsNullOrEmpty(o)) return false;

if (o.Substring(0,1) == this.\_min || o.Substring(0,1) == this.\_max )

return false;

else

return true;

}

}

[stringValidate("h","z", ErrorMessage ="{0} good {1} bad {2}")]

public string USER\_NAME { get; set; }



* + 1. CustomValidation - Client

前提：必须激活客户端验证，加载JQuery 客户端验证

bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(

"~/Scripts/jquery.validate\*"));

@Scripts.Render("~/bundles/jqueryval")

注册客户端验证：IClientValidatable

public class NumberRangeAttribute: ValidationAttribute, IClientValidatable

{

private int Max;

private int Min;

public NumberRangeAttribute(int mm, int mx):base("{0} is out of range")

{

Min = mm;

Max = mx;

}

public override string FormatErrorMessage(string name)

{

return string.Format("Field {0} should have value between {1} and {2}", name, Min, Max);

}

public IEnumerable<ModelClientValidationRule>

GetClientValidationRules(ModelMetadata metadata, ControllerContext context)

{

var rule = new ModelClientValidationRule();

rule.ValidationType = "**numberofrange**"; - 注意必须小写，因为 HTML JQuery 属性规定小写

rule.ErrorMessage = this.FormatErrorMessage(metadata.GetDisplayName());

rule.ValidationParameters.Add("**minvalue**", this.Min); - 注意必须小写

rule.ValidationParameters.Add("**maxvalue**", this.Max); - 注意必须小写

yield return rule;

}

protected override ValidationResult IsValid(object value, ValidationContext validationContext)

{

if (value != null)

{

int cval = 0;

int.TryParse(value.ToString(), out cval);

if (cval >= Min && cval <= Max)

{

return ValidationResult.Success;

}

else

{

return new ValidationResult(this.FormatErrorMessage(validationContext.DisplayName));

}

}

return ValidationResult.Success;

}

}

需要客户端 Javascript编写验证逻辑代码：

<script>

$.validator.unobtrusive.adapters.add("numberofrange", ["minvalue", "maxvalue"],

function (options) {

options.rules["**numberofrange**"] = {

minVV: options.params.minvalue, - 可以影射到其他名字

maxVV: options.params.maxvalue - 可以影射到其他名字

};

options.messages["**numberofrange**"] = options.message; - 错误信息必须要有，重要

});

$.validator.addMethod("**numberofrange**", function (value, element, params) {

var min = parseInt(params.minVV);

var max = parseInt(params.maxVV);

console.log("value: " + value + " min:" + min + " max:" + max);

$(element).css("border", "1px solid red"); - 可以操作 HTML 元素

return value >= min && value <= max;

});

</script>



客户端验证： PasswordCompare, Password

public class User

{

[Password(6)]

public string name { get; set; }

[PasswordCompare("name")]

[Password(6)]

public string comname { get; set; }

[NumberRangeAttribute(10,20)]

[Display(Name ="Class Level")]

public int? level { get; set; }

}

public class PasswordCompareAttribute: ValidationAttribute, IClientValidatable

{

private string \_compareTo;

public PasswordCompareAttribute(string CompareTo)

{

this.\_compareTo = CompareTo;

}

public override string FormatErrorMessage(string name)

{

return string.Format("{0} doesn't math {1}.", name, this.\_compareTo);

}

protected override ValidationResult IsValid(object value, ValidationContext validationContext)

{

string otherVal = string.Empty;

if(string.IsNullOrEmpty(this.\_compareTo))

{

**//重点是如何获取其他 字段的值**

var basePropetyInfo = validationContext.ObjectType.GetProperty(this.\_compareTo);

var pVal = basePropetyInfo.GetValue(validationContext.ObjectInstance, null);

otherVal = pVal.ToString();

}

string thisVal = string.Empty;

if(value!=null)

{

thisVal = value.ToString();

}

if (thisVal == otherVal)

return ValidationResult.Success;

else

return new ValidationResult(this.FormatErrorMessage(validationContext.DisplayName));

}

#region ClientSide Custom Validation

public IEnumerable<ModelClientValidationRule> GetClientValidationRules(ModelMetadata metadata, ControllerContext context)

{

var rule = new ModelClientValidationRule();

rule.ValidationType = "passwordcompare";

rule.ErrorMessage = this.FormatErrorMessage(metadata.GetDisplayName());

rule.ValidationParameters.Add("compareto", this.\_compareTo);

yield return rule;

}

#endregion ClientSide Custom Validation

}

public class PasswordAttribute: ValidationAttribute, IClientValidatable

{

private int minlen;

public PasswordAttribute(int minLength)

{

this.minlen = minLength;

}

public override string FormatErrorMessage(string name)

{

return string.Format("{0} at least {1} characters.", name, minlen);

}

protected override ValidationResult IsValid(object value, ValidationContext validationContext)

{

if (value != null)

{

int cval = value.ToString().Length;

if (cval >= this.minlen)

{

return ValidationResult.Success;

}

else

{

return new ValidationResult(this.FormatErrorMessage(validationContext.DisplayName));

}

}

return ValidationResult.Success;

}

#region ClientSide Custom Validation

public IEnumerable<ModelClientValidationRule> GetClientValidationRules(ModelMetadata metadata, ControllerContext context)

{

var rule = new ModelClientValidationRule();

rule.ValidationType = "password";

rule.ErrorMessage = this.FormatErrorMessage(metadata.GetDisplayName());

rule.ValidationParameters.Add("minlength", this.minlen);

yield return rule;

}

#endregion ClientSide Custom Validation

}

Place this JS code to “xxx.cshtml” tempate

<script>

$.validator.unobtrusive.adapters.add("passwordcompare", ["compareto"], function (options) {

options.rules["passwordcompare"] = {

compareto: options.params.compareto

};

options.messages["passwordcompare"] = options.message;

});

$.validator.addMethod("passwordcompare", function (value, element, params) {

var otherval = $("#" + params.compareto).val();

console.log("value: " + value + " other: " + otherval);

/\*

if (value == otherval) {

element.message = "";

$("#" + params.compareto + "-error").empty();

return true;

} else {

return false;

}

\*/

return value == otherval;

});

$.validator.unobtrusive.adapters.add("password", ["minlength"], function (options) {

options.rules["password"] = {

minlength: options.params.minlength

};

options.messages["password"] = options.message;

});

$.validator.addMethod("password", function (value, element, params) {

var minlenth = parseInt(params.minlength);

console.log("MinLength: " + minlenth + " Value: " + value);

return value.toString().length >= minlenth;

});

</script>

Index.cshtml

@{ Html.EnableClientValidation(); Html.EnableUnobtrusiveJavaScript(); }

@using ( Html.BeginForm("index", "home", FormMethod.Get) ) {

@Html.DisplayNameFor(m => m.level)

<span> : </span>

@Html.EditorFor(m => m.level) @Html.ValidationMessageFor(m=>m.level)

<br /><br />

@Html.DisplayNameFor(m => m.name)

<span> : </span>

@Html.EditorFor(m => m.name) @Html.ValidationMessageFor(m => m.name)

<br /><br />

@Html.DisplayNameFor(m => m.comname)

<span> : </span>

@Html.EditorFor(m => m.comname) @Html.ValidationMessageFor(m => m.comname)

<input type="submit" name="submit" value="sumbit" />

}

* 1. Compare

using System.ComponentModel.DataAnnotations;

using System.Web.Mvc;

在这两个命名空间里都有，其实一样的，但是注意名字冲突

[Compare("email")]

public string USER\_NAME { get; set; }

public string email { get; set; }



[Compare("email", ErrorMessage ="{0}和{1}的输入值不相同")]

public string USER\_NAME { get; set; }



* 1. MinLength & MaxLength : MinLength(5) MaxLength(10)
  2. Remote

Remote 使用的前提是：

1. 因为是属于客户前端验证: 所以必须引用JQuery, Ajax, Jquery,validation

bundles.Add(new ScriptBundle("~/bundles/jquery").Include(

"~/Scripts/jquery-{version}.js"));

bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(

"~/Scripts/jquery.validate\*"));

bundles.Add(new ScriptBundle("~/bundles/ajax").Include(

"~/Scripts/jquery.unobtrusive-ajax.min.js"));

1. Web.config 允许Ajax, client validation

<appSettings>

<add key="webpages:Version" value="3.0.0.0" />

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

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

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

</appSettings>

1. 页面引用

@Scripts.Render("~/bundles/ajax")

@Scripts.Render("~/bundles/jqueryval")

1. 定义远程验证的 Action

public class HomeController : Controller

{

public JsonResult nameValidate(string USER\_NAME, string email)

{

if (USER\_NAME == "JACKSON" && email == "william@abc.com")

return Json(true, JsonRequestBehavior.AllowGet);

else

return Json(string.Format("user: {0} - email: {1} invalid !!", USER\_NAME, email),

JsonRequestBehavior.AllowGet);

// 也可以返回 false, 则错误信息使用 REMOTEAttribute 定义的， 也就是说这里优先级高

}

}

1. 为模型定义[Remote]

public int user\_id { get; set; }

[Display(Name ="用户名")]

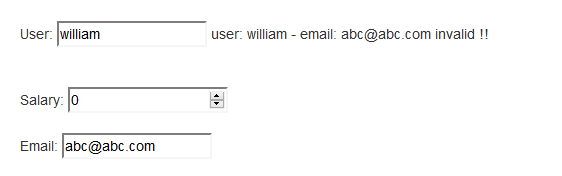
[Remote("nameValidate","home", AdditionalFields = "email",

ErrorMessage ="remote user {0} 无效值")]

public string USER\_NAME { get; set; }

[Display(Name ="Email Addres")]

public string email { get; set; }



注意模型的结构，

User: @Html.EditorFor(m => m.user.USER\_NAME)

@Html.ValidationMessageFor(m => m.**user**.USER\_NAME)<br />

<br />

<br />

Email: @Html.EditorFor(m => m.user.email )

@Html.ValidationMessageFor(m => m.**user**.email )<br />

public JsonResult nameValidate(User user)

- 注意传进来的是 user.USER\_NAME, user.email, 所以我们要使用 user object 来接收传入的数据

{

if (user.USER\_NAME == "JACKSON" && user.email == "william@abc.com")

return Json(true, JsonRequestBehavior.AllowGet);

else

return Json(string.Format("user: {0} - email: {1} invalid !!",

user.USER\_NAME, user.email), JsonRequestBehavior.AllowGet);

}

1. dfasdfasd
2. 发发送到发送到

Appendix

* **ASP.NET MVC Custom Model Binder**

## Introduction

In the example below I would like to break up the date into three fields: Day, Month, and Year, on my view, but I would like my model to have just one property that can store the combination of these fields.

## Background

Recently I was looking at an unusual problem where we had to add custom fields to a view on an ad hoc basis. Not really a problem if it was an MVC app talking directly to a data source. But mine was over several REST based services that were talking to a CRM dynamics backend, spread across multiple domains. Which meant we had to change the data models through out the system. That's when it occurred to  me that we can have one dedicated field  that can store all the ad hoc field data using JASON or xml and we append it to one of the existing fields. Probably not the ideal solution, but that's when I came across custom binding for MVC. I would like to share with you how simple and easy it is to use custom binding to solve similar issues.

## Using the Code

#### View

In my view I am going to provision three fields for the date within my form.

Hide   Copy Code

<form id="Home" action="" method="POST">

Day <input id="Day" name="Day" value="" type="text" />

Month <input id="Month" name="Month" value="" type="text" />

Year <input id="Year" name="Year" value="" type="text" />

<input id="Submit" type="Submit" value="Submit" />

</form>

#### Model

In my model I will only have one property for the three fields called a Date:

Hide   Copy Code

public class HomePageModels

{

public string Title { get; set; }

public string Date { get; set; }

}

#### Custom Binding

The custom binding class needs to inherit form IModelBinder. Here we capture the current request and extract the Form fields individually. Then we can manipulate these fields any way we like. In this example as you can see I am adding them to a single property called Date.

Hide   Copy Code

public class HomeCustomBinder : IModelBinder

{

public object BindModel(ControllerContext controllerContext,

ModelBindingContext bindingContext)

{

HttpRequestBase request = controllerContext.HttpContext.Request;

string title = request.Form.Get("Title");

string day = request.Form.Get("Day");

string month = request.Form.Get("Month");

string year = request.Form.Get("Year");

return new HomePageModels

{

Title = title,

Date = day +"/"+ month +"/"+ year

};

}

}

Alternatively if we do not want to implement  custom binding for each and every Model and Property in our application we can inherit from the DefaultModelBinder and override the BindModel method  as below

Hide   Shrink http://www.codeproject.com/images/arrow-up-16.png  Copy Code

public class HomeCustomDataBinder : DefaultModelBinder

{

public override object BindModel(ControllerContext controllerContext, ModelBindingContext bindingContext)

{

if (bindingContext.ModelType == typeof(HomePageModels))

{

HttpRequestBase request = controllerContext.HttpContext.Request;

string title = request.Form.Get("Title");

string day = request.Form.Get("Day");

string month = request.Form.Get("Month");

string year = request.Form.Get("Year");

return new HomePageModels

{

Title = title,

Date = day + "/" + month + "/" + year

};

//// call the default model binder this new binding context

//return base.BindModel(controllerContext, newBindingContext);

}

else

{

return base.BindModel(controllerContext, bindingContext);

}

}

}

Once we have completed coding our custom class we will need to register the class which I do in the Global.asax under Application\_Start().

Hide   Copy Code

protected void Application\_Start()

{

AreaRegistration.RegisterAllAreas();

WebApiConfig.Register(GlobalConfiguration.Configuration);

FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);

RouteConfig.RegisterRoutes(RouteTable.Routes);

BundleConfig.RegisterBundles(BundleTable.Bundles);

AuthConfig.RegisterAuth();

ModelBinders.Binders.Add(typeof(HomePageModels), new HomeCustomBinder());

}

#### Controller

Finally we need to inform the controller as to the binding we want it to use. This we can specify using attributes [ModelBinder(typeof(HomeCustomBinder))] as below:

Hide   Copy Code

[HttpPost]

public ActionResult Index([ModelBinder(typeof(HomeCustomBinder))] HomePageModels home)

{

if (ModelState.IsValid)

{

ViewBag.Title = home.Title;

ViewBag.Date = home.Date;

}

return View();

}

* HttpUtility
* HtmlEncode

传给View 的时候， 是自动做 Encode 的动作。

如果想在 View 里显示原始的字符， 可以使用 @Html.Raw(XXX)

View 里面指定强类型：

@using WebApp1.Controllers;

@model IEnumerable<Album>; - 带分号出错

@using 可以带 分号 ；

@model 不可以带分号， 否则出错

@{

using WebApp1.Controllers;

model IEnumerable<Album>

}

只能 @using 和 @model 直接跟@， 不可以使用块 { }

* Razor 语法：

@{

string mystr = "Hello world string";

}

<span>@(mystr).Length + @mystr.Max() + [william@mystr.com</span](mailto:william@mystr.com%3c/span)>

Hello world string.Length + w + [william@mystr.com](mailto:william@mystr.com)

括弧是很重要的： @(variable) 避免二义

@@mystr.Length - 输出 @mystr.Length

@@ 则转义输出 @

* HTML 编码encode

@{

string mystr = "<script>alert('hello world')</script>";

}

Line 1: <span>@mystr</span><br />

Line 2: <span>@Html.Raw(mystr)</span>

默认的情况下：会自动对字符串 encode 处理

Line 1: <span>&lt;script&gt;alert(&#39;hello world&#39;)&lt;/script&gt;</span><br />

Line 2: <span><script>alert('hello world')</script></span>

Line 2: 会弹出对话框

@{

string mystr = "<script>alert('hello world')</script>";

}

Line 1: <span>@HttpUtility.HtmlDecode(mystr)</span><br />

Line 2: <span>@HttpUtility.HtmlEncode(Html.Raw(mystr))</span>

输出：

Line 1: <span>&lt;script&gt;alert(&#39;hello world&#39;)&lt;/script&gt;</span><br />

Line 2: <span>&lt;script&gt;alert(&#39;hello world&#39;)&lt;/script&gt;</span>

@HttpUtility.HtmlDecode() 不起作用，而 @Html.Raw() 起作用

@HttpUtility.HtmlEncode() 起作用

* 如何在VIEW 里设置多个提交按钮：

<% Html.BeginForm("MyAction", "MyController", FormMethod.Post); %>

<input type="submit" value="Send" />

<input type="submit" value="Cancel" />

<% Html.EndForm(); %>

方法一：

[AttributeUsage(AttributeTargets.Method, AllowMultiple = false, Inherited = true)]

public class MultipleButtonAttribute : ActionNameSelectorAttribute

{

public string Name { get; set; }

public string Argument { get; set; }

public override bool IsValidName(ControllerContext controllerContext, string actionName, MethodInfo methodInfo)

{

var isValidName = false;

var keyValue = string.Format("{0}:{1}", Name, Argument);

var value = controllerContext.Controller.ValueProvider.GetValue(keyValue);

if (value != null)

{

controllerContext.Controller.ControllerContext.RouteData.Values[Name] = Argument;

isValidName = true;

}

return isValidName;

}

}

[HttpPost]

[MultipleButton(Name = "action", Argument = "Save")]

public ActionResult Save(MessageModel mm) { ... }

[HttpPost]

[MultipleButton(Name = "action", Argument = "Cancel")]

public ActionResult Cancel(MessageModel mm) { ... }

方法二：

<% Html.BeginForm("MyAction", "MyController", FormMethod.Post); %>

<input type="submit" name="submitButton" value="Send" />

<input type="submit" name="submitButton" value="Cancel" />

<% Html.EndForm(); %>

public class MyController : Controller {

public ActionResult MyAction(string submitButton) {

switch(submitButton) {

case "Send":

// delegate sending to another controller action

return(Send());

case "Cancel":

// call another action to perform the cancellation

return(Cancel());

default:

// If they've submitted the form without a submitButton,

// just return the view again.

return(View());

}

}

private ActionResult Cancel() {

// process the cancellation request here.

return(View("Cancelled"));

}

private ActionResult Send() {

// perform the actual send operation here.

return(View("SendConfirmed"));

}

}

* 常用的Form 键值对操作方法：

生成键值对，有多种方法：

1. KeyValuePair<Tkey, TValue> - KeyValuePair<string, string>

var pairs = new List<KeyValuePair<string, string>>

{

new KeyValuePair<string, string>("app\_key", "dKqgHG6c7xhNs2Kp"),

new KeyValuePair<string, string>("keywords", evt.navi.keywords),

new KeyValuePair<string, string>("location", evt.navi.location),

new KeyValuePair<string, string>("date", "Future"),

new KeyValuePair<string, string>("page\_number", evt.navi.pageNumber.ToString()),

new KeyValuePair<string, string>("page\_size", evt.navi.pageSize.ToString())

};

KeyValuePair<string,DateTime> kdt = new KeyValuePair<string,DateTime>("Beijing", DateTime.Now);

1. Dictionary<string, string>

IDictionary<string, string> pairs = new Dictionary<string, string>

{

{"app\_key", "dKqgHG6c7xhNs2Kp"},

{"keywords", evt.navi.keywords},

{"location", evt.navi.location},

{"date", "Future"},

{"page\_number", evt.navi.pageNumber.ToString()},

{"page\_size", evt.navi.pageSize.ToString()}

};

如何遍历 Dictionary

string str = string.Empty;

foreach(string el in pairs.Keys)

{

str += string.Format("Diction Key:{0} - value:{1}\n", el, pairs[el]);

}

可以使用 KeyValuePair<Tkey, TValue> 来遍历 Dictionary

str = string.Empty;

foreach (KeyValuePair<string, string> el in pairs)

{

str += string.Format("Key:{0} value:{1} \n", el.Key, el.Value);

}

1. NameValueCollection

using System.Collections.Specialized;

NameValueCollection pairs = new NameValueCollection

{

{"app\_key", "dKqgHG6c7xhNs2Kp"},

{"keywords", evt.navi.keywords},

{"location", evt.navi.location},

{"date", "Future"},

{"page\_number", evt.navi.pageNumber.ToString()},

{"page\_size", evt.navi.pageSize.ToString()}

};

pairs.Add("key1", "value1");

pairs.Add(new NameValueCollection { { "key2", "value2" } });

NameValueCollection nvc = new NameValueCollection

{

{"t1", "Hellow" },

{"t2", "World" }

};

nvc.Add("t3", "godday");

foreach(string el in nvc)

{

Console.WriteLine("nvc key:{0} - {1}", el, nvc[el]);

}

1. FormUrlEncodedContent<

IEnumerable<KeyValuePair<string,string> nameValueCollection>

>

继承自：ByteArrayContent - HttpContent

using (HttpClient client = new HttpClient())

{

client.BaseAddress = new Uri("http://api.eventful.com/json/events/");

client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

client.DefaultRequestHeaders.AcceptCharset.Add(new StringWithQualityHeaderValue("utf-8"));

// IDictionary<string, string> 可以

IDictionary<string, string> pairs = new Dictionary<string, string>

{

{"app\_key", "dKqgHG6c7xhNs2Kp"},

{"keywords", evt.navi.keywords},

{"location", evt.navi.location},

{"date", "Future"},

{"page\_number", evt.navi.pageNumber.ToString()},

{"page\_size", evt.navi.pageSize.ToString()}

};

// Ienumerable< KeyValuePair<string, string> > 也可以

var pairs = new List<KeyValuePair<string, string>>

{

new KeyValuePair<string, string>("app\_key", "dKqgHG6c7xhNs2Kp"),

new KeyValuePair<string, string>("keywords", evt.navi.keywords),

new KeyValuePair<string, string>("location", evt.navi.location),

new KeyValuePair<string, string>("date", "Future"),

new KeyValuePair<string, string>("page\_number", evt.navi.pageNumber.ToString()),

new KeyValuePair<string, string>("page\_size", evt.navi.pageSize.ToString())

};

var content = new FormUrlEncodedContent(pairs);

var response = client.PostAsync("search", content).Result;

if (response.IsSuccessStatusCode)

{

result = response.Content.ReadAsStringAsync().Result;

evt.jsonParse(result);

}

}

注意FormUrlEncodedContent不能直接使用NameValueCollection

将NameValueCollection转化为IDictionary

pairs.AllKeys.ToDictionary(k => k, k => pairs[k]);

NameValueCollection pairs = new NameValueCollection();

pairs["app\_key"] = "dKqgHG6c7xhNs2Kp";

pairs["keywords"] = evt.navi.keywords;

pairs["location"] = "San Diego";

pairs["date"] = "Future";

pairs["page\_number"] = evt.navi.pageNumber.ToString();

pairs["page\_size"] = evt.navi.pageSize.ToString();

IDictionary<string, string> dpairs = pairs.AllKeys.ToDictionary(k => k, k => pairs[k]);

var content = new FormUrlEncodedContent(dpairs);

1. WebClient 可以直接使用NameValueCollection

using (WebClient client = new WebClient())

{

NameValueCollection postKeyValue = new NameValueCollection();

postKeyValue["app\_key"] = "dKqgHG6c7xhNs2Kp";

postKeyValue["keywords"] = evt.navi.keywords;

postKeyValue["location"] = "San Diego";

postKeyValue["date"] = "Future";

postKeyValue["page\_number"] = evt.navi.pageNumber.ToString();

postKeyValue["page\_size"] = evt.navi.pageSize.ToString();

byte[] respBytes = client.UploadValues(

"http://api.eventful.com/json/events/search",

"POST",

postKeyValue

);

result = ASCIIEncoding.UTF8.GetString(respBytes);

evt.jsonParse(result);

}

var client = new HttpClient(handler);

var request4 = new HttpRequestMessage();

client.DefaultRequestHeaders.Clear();

client.DefaultRequestHeaders.Add("Accept", "text/html, application/xhtml+xml, \*/\*");

client.DefaultRequestHeaders.Add("Accept-Encoding", "gzip, deflate");

client.DefaultRequestHeaders.Add("Accept-Language", "en-US,en;q=0.8,sv-SE;q=0.5,sv;q=0.3");

client.DefaultRequestHeaders.Add("User-Agent", "Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.2; WOW64; Trident/6.0)");

## **ASP.NET MVC 中＠Html.Partial，＠Html.Action，**

## **＠Html.RenderPartial，＠Html.RenderAction差别**

# ****对这四个的差别做一个总结，清理一下思路，便利今后应用：****

1、带有Render的办法返回值是void，在办法内部进行输出；不带的返回值类型为MvcHtmlString，所以只能如许应用：  
    ＠Html.Partial 对应 ＠{Html.RenderPartial（....）;}  
    ＠Html.Action 对应 ＠{Html.RenderAction（....）;}

2、Action、RenderAction加载办法的视图，履行Controller → Model → View的次序，然后把产生的页面带回到本来的View中再回传。而Partial、RenderPartial直接加载视图文件内容

3.  Html.Partial可以直接供给用户控件名作为参数，而Html.Action须要有对应的Action，在Action内部返回PartailResult（即retun PartialView（））。

4、对于简单的没有任何逻辑的用户控件，推荐应用Html.Partial；对于须要设置一些Model的用户控件，推荐应用Html.Action。当然，有Model数据也是可以应用Html.Partial办法的，可以看办法的重载。

5、应用Html.Action有个益处，就是可以按照不合的场景选择不合的用户控件。比如：  
＠Html.Action（"UserInfoControl"）  
在对应的UserInfoControl这个Action中，在用户未登录的时辰，可以retun PartialView（"LogOnUserControl"）;登录后，可以retun PartialView（"UserInfoControl"）;

## ASP.NET MVC Bundles 之学习笔记

分类：.NET  作者：nd   阅读：2019   评论：0   时间：2014-11-19

　 　在网页中，我们经常需要引用大量的javascript和css文件，在加上许多javascript库都包含debug版和经过压缩的release 版(比如jquery),不仅麻烦还很容易引起混乱，所以ASP.NET MVC4引入了Bundles特性，使得我们可以方便的管理javascript和css文件。

　　原来，我们引用css和javascript文件我们需要这样一个一个的引用:

1. <script src="~/Scripts/jquery-1.8.2.js"></script>
2. <script src="~/Scripts/jquery-ui-1.8.24.js"></script>
3. <script src="~/Scripts/jquery.validate.js"></script>
4. <link href="~/Content/Site.css" rel="stylesheet" />

　　当需要引用文件的数量较少时还好，但一旦每个页面都需要引用较多文件时，会造成极大的不便，当我们想更换某个引用文件时，将会浪费大量的时间。 发布时，还要将一些库替换成release版，比如上面的jquery-1.8.2.js所对应的jquery-1.8.2.min.js

　　还好，现在我们可以使用Bundles特性：

1. public class BundleConfig
2. {
3. public static void RegisterBundles(BundleCollection bundles)
4. {
5. bundles.Add(new ScriptBundle("~/bundles/jquery")
6. .Include("~/Scripts/jquery-{version}.js"));
7. bundles.Add(new ScriptBundle("~/bundles/jqueryui")
8. .Include("~/Scripts/jquery-ui-{version}.js"));
9. bundles.Add(new ScriptBundle("~/bundles/jqueryval")
10. .Include("~/Scripts/jquery.unobtrusive\*"
11. ,"~/Scripts/jquery.validate\*"));
12. bundles.Add(new StyleBundle("~/Content/css")
13. .Include("~/Content/site.css"));
14. }
15. }

　　接着在Global.asax文件的Application\_Start方法中调用BundleConfig.RegisterBundles方法:

1. protected void Application\_Start()
2. {
3. AreaRegistration.RegisterAllAreas();
4. WebApiConfig.Register(GlobalConfiguration.Configuration);
5. FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
6. RouteConfig.RegisterRoutes(RouteTable.Routes);
7. BundleConfig.RegisterBundles(BundleTable.Bundles);
8. }

　　在上面我们可以看到我们按照功能的不同，将不同的文件分到了相应的Bundle(Bundle就是包的意思)，其中构造函数中的string参 数是Bundle的名称，Include函数是将参数相应的文件包含成一个Bundle。可以发现，对于jquery库我们使用了这样的名 称~/Scripts/jquery-{version}.js，其中{version}部分代表版本号的意思，MVC将会替我们在Scripts文件中 寻找对应的"jquery-版本号.js"文件，并且在非debug模式下，MVC则会使用“jquery-版本号.min.js"文件。

　　我们还看到我们使用了这样的名称~/Scripts/jquery.validate\*的名称，\*是一个通配符，这就意味着Scripts文件夹下的所有前缀为jquery.validate的文件都将包含在同一个Bundle中。

　　最后，我们可以View上使用Bundle来代替原来引用的方式:

1. @Styles.Render("~/Content/css")
2. @Scripts.Render("~/bundles/jquery")

# [ASP.NET MVC 4 (十一) Bundles和显示模式](http://www.cnblogs.com/duanshuiliu/p/3709030.html)

Bundles用于打包CSS和javascript脚本文件，优化对它们的组织管理。显示模式则允许我们为不同的设备显示不同的视图。

### 默认脚本库

在VS创建一个MVC工程，VS会为我们在scripts目录下添加很多脚本库，下面来简单了解下这些脚本库的作用：

|  |  |
| --- | --- |
| **脚本文件** | **说明** |
| jquery-1.7.1.js | jquery的基本类库 |
| jquery-ui-1.8.20.js | jquery的UI类库，方便我们创建丰富的用户控件，基于jquery基本类库 |
| jquery.mobile-1.1.0.js | 用于移动设备UI控件的类库，在创建移动模板的工程时添加 |
| jquery-validate.js | 用于客户端验证的类库 |
| knockout-2.1.0.js | 客户端的模型-视图-视图模式类库，在客户端将显示数据和没模型分开，可以认为是浏览器上的MVC |
| modernizr-2.5.3.js | 用于检测浏览器对HTML5和CSS3的支持 |
| jquery-1.7.1.intellisense.js | 用于Visual studio在编写jQuery代码时提供函数名称的提示 |
| jquery.unobtrusive-ajax.js | 用于MVC框架的unobtrusive Ajax功能 |
| jquery.validate-vsdoc.js | 用于Visual studio在编写jQuery验证函数时提示函数名称 |
| jquery.validate.unobtrusive.js | 用于MVC的客户端验证，依赖jquery-validate.js |

一些脚本文件有常规和最小化两个版本，最小化版本删除注释剪短变量名以缩小文件尺寸，在功能上和正常版本一致。正常版本的jquery- 1.7.1.js文件大小252K，而缩小版的jquery-1.7.1.min.js只有92K，如果网站每天数以百万计的访问量，带来的流量节省还是 很巨大的。缩小版的脚本很难阅读，所以开发时我们使用正常版本的脚本库方便调试，发布时再切换为缩小版本。

### 打包脚本和风格

Bundles定义在/App\_Start/BundleConfig.cs文件，VS为我们创建的默认实现：

[复制代码](javascript:void(0);)

public class BundleConfig {

// For more information on Bundling, visit http://go.microsoft.com/fwlink/?LinkId=254725

public static void RegisterBundles(BundleCollection bundles) {

bundles.Add(new ScriptBundle("~/bundles/jquery").Include("~/Scripts/jquery-{version}.js"));

bundles.Add(new ScriptBundle("~/bundles/jqueryui").Include("~/Scripts/jquery-ui-{version}.js"));

bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include("~/Scripts/jquery.unobtrusive\*","~/Scripts/jquery.validate\*"));

// Use the development version of Modernizr to develop with and learn from. Then, when you're

// ready for production, use the build tool at http://modernizr.com to pick only the tests you need.

bundles.Add(new ScriptBundle("~/bundles/modernizr").Include("~/Scripts/modernizr-\*"));

bundles.Add(new StyleBundle("~/Content/css").Include("~/Content/site.css"));

bundles.Add(new StyleBundle("~/Content/themes/base/css").Include(

"~/Content/themes/base/jquery.ui.core.css",

"~/Content/themes/base/jquery.ui.resizable.css",

"~/Content/themes/base/jquery.ui.selectable.css",

"~/Content/themes/base/jquery.ui.accordion.css",

"~/Content/themes/base/jquery.ui.autocomplete.css",

"~/Content/themes/base/jquery.ui.button.css",

"~/Content/themes/base/jquery.ui.dialog.css",

"~/Content/themes/base/jquery.ui.slider.css",

"~/Content/themes/base/jquery.ui.tabs.css",

"~/Content/themes/base/jquery.ui.datepicker.css",

"~/Content/themes/base/jquery.ui.progressbar.css",

"~/Content/themes/base/jquery.ui.theme.css"));

}

}

[复制代码](javascript:void(0);)

ScriptBundle创建脚本包，StyleBundle创建CSS风格包，两者都使用Include包含一组文件。VS创建的默认包并不一定适合我们的需要，我们可以自行定义：

[复制代码](javascript:void(0);)

public class BundleConfig {

public static void RegisterBundles(BundleCollection bundles) {

bundles.Add(new StyleBundle("~/Content/css").Include("~/Content/\*.css"));

bundles.Add(new ScriptBundle("~/bundles/clientfeaturesscripts")

.Include("~/Scripts/jquery-{version}.js",

"~/Scripts/jquery.validate.js",

"~/Scripts/jquery.validate.unobtrusive.js",

"~/Scripts/jquery.unobtrusive-ajax.js"));

}

}

[复制代码](javascript:void(0);)

注意这里的“~/Scripts/jquery-{version}.js”，{version}匹配对应文件的任何版本并通过工程配置文件选择正 常版本还是缩小版，具体是web.config中的debug设置，如果为true选择正常版本，false则是缩小版。需要注意的是我们不能把相同文件 的不同版本号放在一起，比如“jquery-1.7.2.js”和“jquery-1.7.1.js”，两个版本号都会被发送给客户端反而造成混淆。

在布局文件中使用Scripts.Render()输出脚本包，Styles.Render()输出风格包：

[复制代码](javascript:void(0);)

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

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

<title>@ViewBag.Title</title>

**@Styles.Render**("~/Content/css")

</head>

<body>

@RenderBody()

**@Scripts.Render**("~/bundles/clientfeaturesscripts")

@RenderSection("scripts", required: false)

</body>

</html>

[复制代码](javascript:void(0);)

生成的HTML文件会通过<link href="XXX" rel="stylesheet"/>包含所有包里的CSS文件，所有的脚本文件则通过<script src="XXX"></script>引用。

上面的例子中还使用“@RenderSection("scripts", required: false)”输出一个节，requried=false表示不是必须的，只有在视图文件中定义了这个节才会渲染，我们可以利用它来包含视图需要的额外脚 本文件，比如我们在MakeBooking.cshtml中定义Scripts节来包含脚本文件：

[复制代码](javascript:void(0);)

@model ClientFeatures.Models.Appointment

@{

ViewBag.Title = "Make A Booking";

AjaxOptions ajaxOpts = new AjaxOptions {

OnSuccess = "processResponse"

};

}

<h4>Book an Appointment</h4>

@section scripts {

<script src="~/Scripts/Home/MakeBooking.js" type="text/javascript"></script>

}

...

[复制代码](javascript:void(0);)

使用这种可选节我们可以有选择的视图中包含视图仅需的脚本文件。

### 面向移动设备

人们越来越多的使用移动设备浏览网站，MVC应用也要考虑如何兼容这些移动设备以提供的更好的阅读体验。我们可以使用安卓、苹果手机访问开发测试网 站，更方便的是从www.opera.com/developer/tools/mobile下载模仿移动版本的Opera浏览器，用它可以模仿不同设备 设置不同屏幕大小的显示分辨率来测试。

在MVC WEB应用中我们在普通的视图文件外可以添加面向移动设备的视图，视图文件名里在文件后缀名前加入“.Mobile”表示这是移动设备专用，比如“/Views/Home/MakeBooking.Mobile.cshtml”：

[复制代码](javascript:void(0);)

@model ClientFeatures.Models.Appointment

@{

ViewBag.Title = "Make A Booking";

AjaxOptions ajaxOpts = new AjaxOptions {

OnSuccess = "processResponse"

};

}

<h4>This is the MOBILE View</h4>

@section scripts {

<script src="~/Scripts/Home/MakeBooking.js" type="text/javascript"></script>

}

<div id="formDiv" class="visible">

@using (Ajax.BeginForm(ajaxOpts)) {

@Html.ValidationSummary(true)

<p>@Html.ValidationMessageFor(m => m.ClientName)</p>

<p>Name:</p><p>@Html.EditorFor(m => m.ClientName)</p>

<p>@Html.ValidationMessageFor(m => m.Date)</p>

<p>Date:</p><p>@Html.EditorFor(m => m.Date)</p>

<p>@Html.ValidationMessageFor(m => m.TermsAccepted)</p>

<p>@Html.EditorFor(m => m.TermsAccepted) Terms & Conditions</p>

<input type="submit" value="Make Booking" />

}

</div>

<div id="successDiv" class="hidden">

<h4>Your appointment is confirmed</h4>

<p>Your name is: <b id="successClientName"></b></p>

<p>The date of your appointment is: <b id="successDate"></b></p>

<button id="backButton">Back</button>

</div>

[复制代码](javascript:void(0);)

这里适当调整控件布局以更适合在移动设备上浏览，其他和桌面版基本一致。当我们从移动设备浏览时，MVC自动为我们应用移动版本的视图，MVC依赖 C:\Windows\Microsoft.NET\Framework\v4.0.30319\Config\Browsers目录下的各种浏览器的描 述文件检查浏览器版本，主要是匹配文件中定义的user agent特性，你会发现UC浏览器赫然在列。

### 自定义显示模式

上面的方法将所有的移动设备归为一类，如果我们还需要更细分具体是哪种移动设备，我们可以通过创建自定义显示模式来实现，这是在Application\_start中注册的：

[复制代码](javascript:void(0);)

public class MvcApplication : System.Web.HttpApplication {

protected void Application\_Start() {

**DisplayModeProvider.Instance.Modes.Insert(0,**

**new DefaultDisplayMode("OperaTablet") {**

**ContextCondition = (context => context.Request.UserAgent.IndexOf("Opera Tablet", StringComparison.OrdinalIgnoreCase) >= 0)**

**});**

AreaRegistration.RegisterAllAreas();

...

[复制代码](javascript:void(0);)

这里通过比较请求的User agent是否包含“Opera tablet”来标识OperaTablet显示模式，如果请求来自于这样的设备，MVC会查找包含OperaTablet的视图文件比如/Views /Home/MakeBooking.OperaTable.cshtml，这样我们就可以单为某种设备创建自定义的视图。

# MVC中modelstate的使用

更多 2013/12/17 来源：MVC学习浏览量：12262

学习标签： [MVC](http://www.studyofnet.com/news?wd=MVC) [modelstate](http://www.studyofnet.com/news?wd=modelstate)

**本文导读**：MVC中Controller类有一个ModelState属性集合，可以用来提示传递到视图的model对象是否有错误。ModelState中Error记录识别模型属性的名称和错误信息，并允许指定友好的错误信息。

**MVC中ModelState类需要引用 System.Web.Mvc命名空间，在 System.Web.Mvc.dll 中。**

**属性**

Errors

返回一个 ModelErrorCollection 对象，该对象包含在模型绑定期间发生的任何错误。

Value

返回一个 ValueProviderResult 对象，该对象封装在模型绑定期间绑定的值。

**Html 辅助方法和ModelState集成**

HTML 辅助方法，如Html.TextBox()，在输出内容时，会检查ModelState集合。如果发现该属性有异常或错误，将呈现用户输入的内容和CSS错误类。

例如，在Edit视图中，我们使用Html.TextBox() 辅助方法呈现Dinner对象的EventDate属性：

<%= Html.TextBox("EventDate", String.Format("{0:g}", Model.EventDate)) %>

当有错的时候呈现视图时，Html.TextBox() 方法检查ModelState集合，检查是否有错误关联到Dinner 对象的EventDate属性。当发现有错误时，将显示用户提交的”EntLib” 输入作为参数值，同时对<input type=”textbox” />元素添加CSS 错误类，如下所示：

<input class="input-validation-error" id="EventDate" name="EventDate"

type="text" value="BOGUS" />

你可以定制CSS错误类的样式。默认的CSS错误类 – input-validation-error定义在\content\site.css 文件中，样式定义如下：

.input-validation-error

{

border: 1px solid #ff0000;

background-color: #ffeeee;

}

**Html.ValidationMessage() 辅助方法**

Html.ValidationMessage() 辅助方法用来输出特定Model属性相关的ModelState错误信息：

<%= Html.ValidationMessage("EventDate") %>

上述代码输出：

<span class=”field-validation-error”> The value ‘EntLib’ is invalid</span>

Html.ValidationMessage() 辅助方法也支持第二个参数，允许开发人员覆盖错误消息：

<%= Html.ValidationMessage("EventDate", "\*") %>

上述代码输出：

<span class=”field-validation-error”> \*</span>，而不是默认的错误信息。

**Html.ValidationSummary() 辅助方法**

Html.ValidationSummary() 辅助方法将呈现总结的错误消息，通过<ul><li/></ul>元素列出在ModelState集合中所有详细的错误消息

Html.ValidationSummary() 辅助方法接收一个可选的字符串参数 – 定义一个概括性的错误消息，并显示在所有详细错误信息的前面：

<%= Html.ValidationSummary("Edit was unsuccessful. Please correct the errors and try again.") %>

你也可以定义CSS设置错误消息的样式。

**使用AddRuleViolations辅助方法**

初始的HTTP-POST Edit的实现方法使用了一个foreach循环语句，遍历Dinner对象的Rule Violations，并添加到controller的ModelState集合：

catch

{

foreach (var issue in dinner.GetRuleViolations())

{

ModelState.AddModelError(issue.PropertyName, issue.ErrorMessage);

}

return View(dinner);

}

**为了使代码更简洁一点，我们添加ControllerHelpers类到 NerdDinner项目中，并实现了AddRuleViolations扩展方法，添加了一个对ASP.NET MVC ModelStateDictionary 类的辅助方法。该扩展方法封装了使用RuleViolation 错误信息填充ModelStateDictionary 集合类的逻辑：**

public static class ControllerHelpers

{

public static void AddRuleViolations(this ModelStateDictionary modelState,

IEnumerable<RuleViolation> errors)

{

foreach (RuleViolation issue in errors)

{

modelState.AddModelError(issue.PropertyName, issue.ErrorMessage);

}

}

}

接下来，我们更新HTTP-POST Edit方法，使用上述扩展方法实现Dinner的Rule Violations填充ModelState集合。

**完成Edit Action方法的实现**

下面的代码实现了控制器中Edit的所有逻辑：

**C# 代码   复制**

http://www.studyofnet.com/Codefan-Controls/OutliningIndicators/None.gif

http://www.studyofnet.com/Codefan-Controls/OutliningIndicators/None.gif//

http://www.studyofnet.com/Codefan-Controls/OutliningIndicators/None.gif// GET: /Dinners/Edit/2

http://www.studyofnet.com/Codefan-Controls/OutliningIndicators/None.gifpublic ActionResult Edit(int id)

http://www.studyofnet.com/Codefan-Controls/OutliningIndicators/ExpandedBlockStart.gif{

http://www.studyofnet.com/Codefan-Controls/OutliningIndicators/InBlock.gifDinner dinner = dinnerRepository.GetDinner(id);

http://www.studyofnet.com/Codefan-Controls/OutliningIndicators/InBlock.gifreturn View(dinner);

http://www.studyofnet.com/Codefan-Controls/OutliningIndicators/ExpandedBlockEnd.gif}

http://www.studyofnet.com/Codefan-Controls/OutliningIndicators/None.gif

http://www.studyofnet.com/Codefan-Controls/OutliningIndicators/None.gif//

http://www.studyofnet.com/Codefan-Controls/OutliningIndicators/None.gif// POST: /Dinners/Edit/2

http://www.studyofnet.com/Codefan-Controls/OutliningIndicators/None.gif[AcceptVerbs(HttpVerbs.Post)]

http://www.studyofnet.com/Codefan-Controls/OutliningIndicators/None.gifpublic ActionResult Edit(int id, FormCollection formValues)

http://www.studyofnet.com/Codefan-Controls/OutliningIndicators/ExpandedBlockStart.gif{

http://www.studyofnet.com/Codefan-Controls/OutliningIndicators/InBlock.gifDinner dinner = dinnerRepository.GetDinner(id);

http://www.studyofnet.com/Codefan-Controls/OutliningIndicators/InBlock.gif

http://www.studyofnet.com/Codefan-Controls/OutliningIndicators/InBlock.giftry

http://www.studyofnet.com/Codefan-Controls/OutliningIndicators/ExpandedSubBlockStart.gif{

http://www.studyofnet.com/Codefan-Controls/OutliningIndicators/InBlock.gifUpdateModel(dinner);

http://www.studyofnet.com/Codefan-Controls/OutliningIndicators/InBlock.gifdinnerRepository.Save();

http://www.studyofnet.com/Codefan-Controls/OutliningIndicators/ExpandedSubBlockStart.gifreturn RedirectToAction("Details", new { id = dinner.DinnerID });

http://www.studyofnet.com/Codefan-Controls/OutliningIndicators/ExpandedSubBlockEnd.gif}

http://www.studyofnet.com/Codefan-Controls/OutliningIndicators/InBlock.gifcatch

http://www.studyofnet.com/Codefan-Controls/OutliningIndicators/ExpandedSubBlockStart.gif{

http://www.studyofnet.com/Codefan-Controls/OutliningIndicators/InBlock.gifModelState.AddRuleViolations(dinner.GetRuleViolations());

http://www.studyofnet.com/Codefan-Controls/OutliningIndicators/InBlock.gifreturn View(dinner);

http://www.studyofnet.com/Codefan-Controls/OutliningIndicators/ExpandedSubBlockEnd.gif}

http://www.studyofnet.com/Codefan-Controls/OutliningIndicators/ExpandedBlockEnd.gif}

http://www.studyofnet.com/Codefan-Controls/OutliningIndicators/None.gif

关于Edit方法的实现的优点，不仅Controller类，而且View视图模板都不必关心Dinner模型类的特定验证方法或者业务规则。以后，我们可以针对Model类增加额外的业务规则，而不必要求Controller和View更改代码。这样，我们可以根据需求，以最小的更改代码量，灵活改进应用程序。