Html 辅助方法

1、HtmlHelper 简介

asp.net mvc 中还提供了一些 HtmlHelper（Html 辅助方法），可以简化表单的编写。现在已经是 Ajax 做页面的时代了，连 cshtml 都用的越来越少了，特别是用 HtmlHelper 对于美工和程序员分工很不好，所以我不推荐使用。但是总是有公司特别是一些做企业内部系统的公司还在用 HtmlHelper 的，所以还是要了解一下。

@Html 是 HtmlHelper 类型的成员，可以在 cshtml 中使用。

表单：

```
@using (Html.BeginForm("Login","Default"))
{

}
```

生成：

```
<form action="/Default/Login" method="post"></form>
```

HtmlHelper 的方法有 N 多重载方法，根本记不住，随用随查定义就行。

2、TextBox

```
@Html.TextBox("name")
```

会生成：

```
<input id="name" name="name" type="text" value="" />
```

还可以赋值：

```
@Html.TextBox("name","hello")
```

生成：

```
<input id="name" name="name" type="text" value="hello" />
```

3、Label

```
@Html.Label("UserName", "用户名：")@Html.TextBox("UserName")
```

会生成

```
<label for="UserName">用户名：</label><input id="UserName" name="UserName" type="text" value="" />
```

还有复选框、单选框、TextArea 等玩意，自己研究一下就行了。

4、自定义属性

使用 HtmlHelper 之后，怎么搞标签的属性呢？HtmlHelper 中所有方法都有一个 object htmlAttributes 参数，这个参数可以传任意对象进去会把每个属性的名字和值输出到标签中，为了方便一般传递一个匿名类对象进去（asp.net mvc 中讲过）：

```
@Html.TextBox("name","hello",new { abc=123,haha=false,yes="rupeng"})
```

生成：

```
<input abc="123" haha="False" id="name" name="name" type="text" value="hello"
```

yes="rupeng" />

class 是 C#关键字，不能用作属性名字，怎么办？用@class。

@Html.TextBox("name","hello",new {@class="warn"})

生成：

<input class="warn" id="name" name="name" type="text" value="hello" />

生成 v-id=””这样的标签怎么办？-是 C#中的特殊字符。用下划线代替所有下划线的属性都会转换为连字符

@Html.TextBox("name","hello",new {@class="warn",v_id=5})

转换为：

<input class="warn" id="name" name="name" type="text" v-id="5" value="hello" />

5、表单验证

@Html.ValidationSummary()

会把提交后表单的验证错误消息汇总显示

Person.cs：

```
public class Person
{
    [DisplayName("用户名")]
    [Required]
    [StringLength(8)]
    public string UserName { get; set; }

    [DisplayName("密码")]
    [Required]
    [StringLength(8)]
    public string Password { get; set; }
}
```

Index.cshtml：

```
@using (Html.BeginForm("Login","Default"))
{
    <span>用户名：</span>@Html.TextBox("UserName")
    <span>密码：</span>@Html.Password("Password")
    <input type="submit" value="提交"/>
    @Html.ValidationSummary()
}
```

DefaultController.cs：

```
public ActionResult Index()
{
    return View();
}

public ActionResult Login(Person model)
```

```
{
    if(!ModelState.IsValid)
    {
        return View("Index");//一定要返回表单原始页面
    }
    return Content(model.UserName);
}
```

如果字段比较多，汇总消息显示不合适，那么就用@Html.ValidationMessage("UserName")值只显示 UserName 属性相关的错误信息。

还可以在 Action 中写 ModelState.AddModelError("UserName", "不知道为啥，感觉有点错");添加自定义错误消息，也可以写 ModelState.AddModelError("", "不知道为啥，感觉有点错");搞和属性无关的错误消息。

验证消息的标签定义了样式名字，需要项目自定义样式文件。

6、隐式从 ViewBag 取数据

如果在 ViewBag/ViewData 中放数据，那么 HtmlHelper 可以隐式从 ViewBag 中取数据：

```
ViewBag.UserName = "admin";
@Html.TextBox("UserName")
```

DropDownList 的数据也可以来自于 ViewBag：

```
Book[] books = new Book[] {
    new Book{ Id=1,Name="如鹏网"},
     new Book{ Id=2,Name="腾讯"},
      new Book{ Id=3,Name="天猫"}
};
SelectList slBooks = new SelectList(books,"Id","Name",2);
ViewBag.books = slBooks;

@Html.DropDownList("books")
```

不建议用这个特性，只要知道有这东西，避免进坑就行了。

7、强类型视图绑定

对于强类型视图，可以使用 Html.***For 方法，传递 lambda 表达式，避免编译错误

```
@model WebApplication6.Models.Book
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index1</title>
</head>
<body>
```

```
    <div>
        @Html.Label("Id")：@Html.TextBoxFor(m=>m.Id)<br/>
        @Html.Label("Name")：@Html.TextBoxFor(m => m.Name)<br />
    </div>
</body>
</html>
```

8、数据客户端验证

使用 asp.net mvc 的完整模板。 只要页面中引入 jquery.validate.js 和 jquery.validate.unobtrusive.js（在 asp.net mvc 模板中有），那么把这两个文件引入页面，然后会自动启用客户端校验，服务器端校验当然还会继续有

Index.cshtml 内容：

```
@model WebApplication7.Models.UserModel

@using (Html.BeginForm("Login","Home"))
{
    @Html.LabelFor(m=>m.UserName," 用 户 名 ") @Html.TextBoxFor(m=>m.UserName) @Html.ValidationMessageFor(m=>m.UserName)<br/>

    @Html.LabelFor(m => m.Password, " 密 码 ") @Html.TextBoxFor(m => m.Password) @Html.ValidationMessageFor(m => m.Password)<br />

    <input type="submit" value="提交"/>
}

public ActionResult Login(UserModel model)
{
    if(!ModelState.IsValid)
    {
        return View(nameof(Index));
    }
    return Content(model.UserName);
}
```

原理：服务器端根据参数的服务器端校验规则在客户端生成这样的验证标签：data-val="true" data-val-length=" 字 段 UserName 必 须 是 最 大 长 度 为 8 的 字 符 串 " data-val-length-max="8" data-val-required="UserName 字段是必需的"。jquery.validate.js 在客户端解析这些。

明白了这个原理，即使不适用 HtmlHelper，咱们也可以在自己的页面中用 jquery.validate.js。

路由

1、 路由的作用：

1)分析 url，拆解出 url 中的内容。路由不负责“找 Controller、找 Action”！

2)生成 Url 用；

2、 解析默认的路由

我们开发项目的时候一般默认的路由就足够了，如果自定义路由通常是不想用默认的 Url 访问。

默认路由是：

```
routes.MapRoute(
    name: "Default",
    url: "{controller}/{action}/{id}",
    defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
);
```

name 只是一个标志名字而已；url 相当于定义了一个类似于正则表达式的匹配模式：由/分割三部分，第一部分提取成一个名字叫 controller 的数据，第二部分提取成一个名字叫 action 的数据，第三部分提取成一个名字叫 id 的数据；defaults 部分可选，如果指定表示，如果部分内容缺失，那么每个部分的默认值是什么，defaults 一般用匿名类对象，属性的名字和 url 中定义的名字一致，UrlParameter.Optional 表示这部分可以不指定。

这样当我们请求/Home/Index/5 的时候，就会拆解成 controller=”Home”、action=”Index”、id=5，然后接下来 MVC 引擎会找到 HomeController 类的 Index 方法去执行，并且给 id 参数传递 5。controller、action 属于两个特殊的名字，用来对应 controller 类和 action 方法，属于必须的，否则就找不到对应的 controller 类和 action 方法，其他名字随意。

3、 改动一下默认路由

如果我们把 url 改成 url: "{id}/{action}/{controller}"，那么访问地址就要改成/3/Index/Home 才能正确的访问。

如果我们把 url 改成 url: "aa/rupeng/{id}/{action}/{controller}"，那么访问地址就要改成/aa/rupeng/3/Index/Home 才能正确的访问。

注意路由 URL 不能以“/”或“~”字符开头，并且不能包含“?”字符，所以路由规则中不能对于 querystring 做匹配。

为什么 url: "{controller}/{action}/{id}"来讲，访问/、/Home、/Home/Index 都能访问到 public ActionResult Index()。因为设定了 defaults，他们都有默认值，如果没有指定，则用默认值来代替。所以 defaults 设定的不像有些人想象的是你设定的匹配值，他只是默认值。

新增 Action 方法：

```
public ActionResult Test2(string name,int id)
{
    return Content("name="+name+",id="+id);
}
```

路由规则设置为：url: "{controller}/{action}/{id}/{name}"

然后可以通过/Home/Test2/33/abc 访问

如果我们访问/Home/Test2/33/abc/asdfa，就会匹配不到任何路由。

路由规则不一定用/做分割，比如：url: "{controller}-{action}" 就可以用/Home-Index 访问

但是路由规则不能让别人无法拆分解析，比如设置成如下就不行 url: "{controller}{action}"

4、 多路由规则

可以指定多个路由规则，只要 name 不一样就行，当一个请求过来的时候，会从上向下匹配，碰到一个能匹配的就结束了，不会继续向下走，如果所有规则不匹配的话则报 404 错误。

下面的讲解没什么实际意义，只是讲语法。

如下设置路由规则：

```
routes.MapRoute(
    name: "Default",
    url: "{controller}/{action}"
);
routes.MapRoute(
    name: "Default2",
    url: "rupeng/{action}/{controller}"
);
```

那么访问：/rupeng/Test1 /AA 则匹配 url: "rupeng/{action}/{controller}"；访问 /Home/Index 就会匹配：url: "{controller}/{action}"

5、 生成 Url

路由规则：

```
routes.MapRoute(
    name: "Default",
    url: "{controller}/{action}"
);
routes.MapRoute(
    name: "Default2",
    url: "{controller}-{action}/{*aa}"
);
routes.MapRoute(
    name: "Default3",
    url: "{controller}/{action}/{id}/{name}"
);
```

如下：@Url.RouteUrl("Default3", new { controller="Yes",action="Test",id=5,name="rupeng"})

生成：/Yes/Test/5/rupeng