ASP.NET由于采用了管道式设计，具有很好的扩展性，而整个ASP.NET MVC应用框架就是通过扩展ASP.NET实现的。通过上面对ASP.NET管道设计的介绍，我们知道ASP.NET的扩展点只要体现在HttpMoudle和HttpHandler这两个核心组建之上，实际上整个ASP.NET MVC框架就是通过自定义的HttpMoudle（[UrlRoutingModule](http://msdn2.microsoft.com/cc680118.aspx)）和HttpHandler（[MvcHandler](http://msdn2.microsoft.com/dd505105.aspx)）实现的。为了上读者从整体上把握ASP.NET MVC的工作机制，接下来我按照其原理通过一些自定义组件来模拟ASP.NET MVC的运行原理，我们也可以将此视为一个“迷你版”的ASP.NET MVC。值得一提的是，为了让读者根据该实例从真正的ASP.NET MVC中找到对应的组件，我完全采用了与ASP.NET MVC一致的类型命名方式。[源代码从[这里](http://files.cnblogs.com/artech/MiniMvc2.rar)下载]

在正式介绍我们自己创建 的“迷你版”ASP.NET MVC的实现原理之前，我们不妨来看看建立在该框架之上的Web应用如何实现。我们通过Visual Studio创建一个空的ASP.NET Web应用（注意不是ASP.NET MVC应用），我们不会引用System.Web.Mvc.dll这个程序集，所以你在接下来的程序中看到的定义在该程序集中的同名类型都是我们自行定义的。

我们首先定义了如下一个SimpleModel类型，它表示最终需要绑定到View上的数据。简单起见，同时也为了验证针对Controller和Action的解析机制，SimpleModel定义的两个属性分别表示当前请求的目标Controller和Action。

1: public class SimpleModel

2: {

3: public string Controller { get; set; }

4: public string Action { get; set; }

5: }

像真正的ASP .NET MVC应用开发一样，我们需要定义Controller类型。如下面的代码片断所示，按照我们熟悉的命名方式（以字符Controller作为后缀），我们定义了如下一个HomeController。HomeController实现的抽象类型ControllerBase是我们自行定义的。以自定义的ActionResult作为返回类型的Index方法表示Controller的Action，它接受一个SimpleModel类型的对象作为参数。该Action方法返回的ActionResult是一个RawContentResult对象，顾名思义，RawContentResult就是将指定的内容进行原样显示。在这里我们将作为参数的SimpleModel对象的Controller和Action属性显示出来。

1: public class HomeController: ControllerBase

2: {

3: public ActionResult Index(SimpleModel model)

4: {

5: string content = string.Format("Controller: {0}<br/>Action:{1}", model.Controller, model.Action);

6: return new RawContentResult(content);

7: }

8: }

ASP.NET MVC根据请求地址来解析出用于处理该请求的Controller的类型和Action方法名称。具体来说，我们预注册一些包含Controller和Action名称作为站位符的（相对）地址模板，如果请求地址符合相应地址模板的模式，Controller和Action名称就可以正确地解析出来。和ASP.NET MVC应用类似，我们在Global.asax中注册了如下一个地址模板（{controller}/{action}）。

1: public class Global : System.Web.HttpApplication

2: {

3: protected void Application\_Start(object sender, EventArgs e)

4: {

5: RouteTable.Routes.Add("default",new Route{Url = "{controller}/{action}"});

6: ControllerBuilder.Current.SetControllerFactory(new DefaultControllerFactory());

7: ControllerBuilder.Current.DefaultNamespaces.Add("WebApp");

8: }

9: }

在如上所示的用于进行地址模板注册的Application\_Start方法之中，我们还注册了一个用于创建Controller对象的工厂。前面定义的HomeController定义在命名空间WebApp下，由于请求地址中只能解析出Controller类型的名称，我们需要将该命名空间注册为当前ControllerBuilder的默认命名空间。RouteTable、ControllerBuilder和DefaultControllerFactory都是我们自定义的类型。

正如我上面所说，ASP.NET MVC是通过一个自定义的HttpModule实现的，在这个“迷你版”ASP.NET MVC框架中我们也将其起名为UrlRoutingModule。在运行Web应用之前，我们需要通过配置对该自定义HttpModule进行注册，下面是相关的配置。

1: <configuration>

2: <system.webServer>

3: <modules>

4: <add name="UrlRoutingModule"

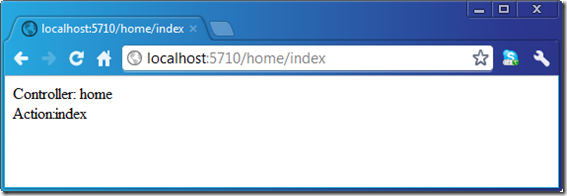
5: type="Artech.MiniMvc.UrlRoutingModule, Artech.MiniMvc"/>

6: </modules>

7: </system.webServer>

8: </configuration>

到目前为止，所有的编程和配置工作已经完成。为了 让定义在HomeController中的Action方法Index来处理针对该Web应用的访问请求，我们需要指定与之匹配的地址（符合定义在注册地址模板的URL模式）。如下图所示，由于在浏览器中输入地址（<http://.../Home/Index>）正好对应着HomeController的Index操作，所以对应的方法会被执行，而执行的结果就是将当前请求的目标Contrller和Action的名称显示出来。

[](http://images.cnblogs.com/cnblogs_com/artech/201203/201203201509331702.png)

上面我们我们演示了如何在我们自己创建的“迷你版”ASP.NET MVC框架中创建一个Web应用，从中我们可以看到和创建一个真正的ASP.NET MVC应用别无二致。接下来我们就来逐步地分析这个自定义的ASP.NET MVC框架是如何建立起来的，而它也代表了真正的ASP.NET MVC框架的工作原理。

[ASP.NET MVC是如何运行的[1]: 建立在“伪”MVC框架上的Web应用](http://www.cnblogs.com/artech/archive/2012/03/11/mvc-how-to-work-01.html)   
[ASP.NET MVC是如何运行的[2]: URL路由](http://www.cnblogs.com/artech/archive/2012/03/11/mvc-how-to-work-02.html)   
[ASP.NET MVC是如何运行的[3]: Controller的激活](http://www.cnblogs.com/artech/archive/2012/03/12/mvc-how-to-work-03.html)   
[ASP.NET MVC是如何运行的[4]: Action的执行](http://www.cnblogs.com/artech/archive/2012/03/12/mvc-how-to-work-04.html)