现在补上URL路由的学习，至于蒋老师自建的MVC小引擎和相关案例就放在论文提交后再实践咯。通过ASP,NET的路由系统，可以完成请求URL与物理文件的分离，其优点是：灵活性、可读性、SEO优化。接下来通过一个最简单的路由例子进入这部分的学习，这是一个蒋老师提供的WebForm路由的例子，回想起刚做ASP.NET时，每次看到.aspx页面的前台代码时的茫然和无措，茫茫多的标签，属性，数据源的绑定吓死小兄弟俺了，也花过不少时间去理解记忆，效果不也不大。现在回头看看感觉好了很多，看到IsPostback老亲切了，觉得在理解的基础上拖拉控件也是很幸福的事情，嘿嘿。

|  |
| --- |
| void Application\_Start(object sender, EventArgs e)  {  //路由配置  var defaults = new RouteValueDictionary { { "name", "\*" }, { "id", "\*" } };  RouteTable.Routes.MapPageRoute("default", "employees/{name}/{id}", "~/Default.aspx", true, defaults);  } |
| //前台代码  <form id="form1" runat="server">  <div>  <asp:GridView ID="GridViewEmployees" runat="server" AutoGenerateColumns="False" Width="100%">  <Columns>  <asp:HyperLinkField DataNavigateUrlFields="Name,Id" DataNavigateUrlFormatString="~/employees/{0}/{1}" DataTextField="Name" HeaderText="姓名" />  <asp:BoundField DataField="Gender" HeaderText="性别" />  <asp:BoundField DataField="Birthday" DataFormatString="{0:dd/MM/yyyy}" HeaderText="出生日期" />  <asp:BoundField DataField="Department" HeaderText="部门" />  </Columns>  </asp:GridView>  <asp:DetailsView ID="DetailsViewEmployee" runat="server" Height="50px" Width="100%" AutoGenerateRows="False">  <Fields>  <asp:BoundField DataField="Id" HeaderText="ID" />  <asp:BoundField DataField="Name" HeaderText="姓名" />  <asp:BoundField DataField="Gender" HeaderText="性别" />  <asp:BoundField DataField="Birthday" DataFormatString="{0:dd/MM/yyyy}" HeaderText="出生日期" />  <asp:BoundField DataField="Department" HeaderText="部门" />  </Fields>  </asp:DetailsView>  </div>  </form> |
| //后台代码  public partial class Default : System.Web.UI.Page  {  private EmployeeRepository \_repository;  public EmployeeRepository Repository  {  get  {  return null == \_repository ? \_repository = new EmployeeRepository() : \_repository;  }  }  protected void Page\_Load(object sender, EventArgs e)  {  if (this.IsPostBack)  {  return;  }  string employeeId = this.RouteData.Values["id"] as string;  if (employeeId == "\*" || string.IsNullOrEmpty(employeeId))  {  this.GridViewEmployees.DataSource = this.Repository.GetEmployees();  this.GridViewEmployees.DataBind();  this.DetailsViewEmployee.Visible = false;  }  else  {  this.DetailsViewEmployee.DataSource = this.Repository.GetEmployees(employeeId);  this.DetailsViewEmployee.DataBind();  this.GridViewEmployees.Visible = false;  }  }  } |

接下来通过一个表格简要介绍下路由系统的相关类型：

|  |  |
| --- | --- |
| 类型 | 简介 |
| RouteBase | 提供GetRouteData方法获得RouteData，该对象中属性RouteHandler用于提供HttpHandler对象，Values提供解析Url后的数据，DataTokens提供自己在路由类型中添加的数据；GetVirtualPath方法根据提供的变量和URL模板生成虚拟路径，是GetRouteData方法的逆过程，用于响应阶段。 |
| Route | Url属性表示Url模板，如world/{country}/{city}匹配world/China/Shanghai，通配符为{\*pathInfo}。Defaults属性提供模板变量默认值，Constraints提供约束条件 |
| RouteTable | 静态属性Routes维护全局路由表，属性RouteExistingFiles用于控制是否需要对存在的物理文件实施路由，默认为False；属性AppendTrailingSlash和LowercaseUrls用于GetVirtualPath方法是否转变url为小写或在末尾添加”/” |

之后展示一个关于注册路由相对完整的例子，代码如下所示：

|  |
| --- |
| var defaults = new RouteValueDictionary { { "cityCode", "021" }, { "distinctCode", 1 } };  var constraints = new RouteValueDictionary { { "cityCode", @"0\d{2,3}" }, { "distinctCode", @"[1-7]{1}" } };  var dataTokens = new RouteValueDictionary { { "defaultCity", "Shanghai" }, { "defaultDistinct", "Pudong" } };  RouteTable.Routes.MapPageRoute("default", "{cityCode}/{distinctCode}", "~/default.aspx", false, defaults, constraints, dataTokens); |

再则是介绍一些路由模块在ASP.NET MVC中的扩展，包括是UrlParameter.Optional代表缺省的URL参数，通过Area来划分系统的模块，以及HtmlHelper与UrlHelper相关的内容，相对比较简单就不一一介绍了，只是补充一个Area注册的例子加强记忆，代码如下：

|  |
| --- |
| public class WeatherAreaRegistration : AreaRegistration  {  public override string AreaName  {  get { return "Weather"; }  }  public override void RegisterArea(AreaRegistrationContext context)  {  object defaults = new  {  areacode = "010",  days = 2,  defaultCity = "BeiJing",  defaultDays = 2  };  object constraints = new { areacode = @"0\d{2,3}", days = @"[1-3]{1}" };  context.MapRoute("weatherDefault", "weather/{areacode}/{days}", defaults, constraints);  }  } |

最后来说说整个路由系统是如何实现的，正如蒋老师所说，是通过HttpHandler的动态映射来实现的。UrlRoutingModule实现了IHttpModule，通过注册HttpApplication的PostResolveRequestCache事件对请求进行拦截，并利用路由表与请求URL进行模式匹配得到相应的路由数据，并获得关联的HttpHandler用于处理请求。注意PageRouteHandler和MvcRouteHanlder分别针对WebForm和MVC，部分相关代码如下所示：

|  |
| --- |
| public class UrlRoutingModule:IHttpModule  {  public RouteCollection RouteCollection { get; set; }  public void Init(HttpApplication context)  {  context.PostResolveRequestCache += new EventHandler(this.OnApplicationPostResolveRequestCache);  }  private void OnApplicationPostResolveRequestCache(object sender, EventArgs e) {  HttpContext context = ((HttpApplication)sender).Context;  HttpContextBase contextWrapper = new HttpContextWrapper(context);  RouteData routeData = this.RouteCollection.GetRouteData(contextWrapper);  RequestContext requestContext = new RequestContext(contextWrapper, routeData);  IHttpHandler handler = routeData.RouteHandler.GetHttpHandler(requestContext);  context.RemapHandler(handler);  }  //omit  } |

Tip: ASP.NET的处理过程始终是Request->HttpModule->HttpHandler->Response，

对了，现在关于Owin的相关内容很火，感觉其就是J2EE标准规范的.NET版本，提供规范与接口，大家一起来做好它的感觉。简单来说就是现在只有IIS支持ASP.NET管道，而JAVA却又Tomcat，Weblogic等很多优质的Web服务器支持Servlet，今后我们也可以用到很多开源高效的Web服务器了，因为它们一定会简化现有的ASP.NET管道，让XXXing，XXXed离我们远一点吧。真心的说，以前面试的时候感觉好难背，哈哈。

注：本文主要供自己学习，不妥之处望见谅。

参考资料：

[1]蒋金楠. ASP.NET MVC4框架揭秘[M]. 上海:电子工业出版社, 2012. 35-85