创建一个WheelHandler类并实现IHttpHandler接口

public class WheelHandler : IHttpHandler

{

public WheelHandler(RequestContext requestContext)

{

this.RequestContext = requestContext;

}

#region IHttpHandler Members

public bool IsReusable

{

get { return true; }

}

public void ProcessRequest(HttpContext context)

{

context.Response.Write(String.Format("this is a Wheel for {0}Controller and {1}action "

, this.RequestContext.RouteData.Values["Controller"]

, this.RequestContext.RouteData.Values["Action"]));

context.Response.End();

}

#endregion

public RequestContext RequestContext { get; private set; }

}

定义WheelRouteHandler

public class WheelRouteHandler : IRouteHandler

{

public IHttpHandler GetHttpHandler(RequestContext requestContext)

{

return new WheelHandler(requestContext);

}

IHttpHandler IRouteHandler.GetHttpHandler(RequestContext requestContext)

{

return this.GetHttpHandler(requestContext);

}

}

创建HttpModule 对象

自定义的HttpModule的责任是，构建上下文，创建HttpHandler对象并将它映射到Http处理程序里去。

public class WheelRoutingModule : IHttpModule

{

public void Dispose()

{

}

public void Init(HttpApplication context)

{

context.PostResolveRequestCache += new EventHandler(context\_PostResolveRequestCache);

}

void context\_PostResolveRequestCache(object sender, EventArgs e)

{

HttpContextBase context = new HttpContextWrapper(((HttpApplication)sender).Context);

this.PostResolveRequestCache(context);

}

private void PostResolveRequestCache(HttpContextBase context)

{

//GetRouteData返回第一个匹配的RouteData ，所以添加顺序很重要

RouteData routeData = RouteTable.Routes.GetRouteData(context);

if (routeData == null)

{

return;

//throw new InvalidOperationException();

}

IRouteHandler routeHandler = routeData.RouteHandler;

if (routeHandler == null)

{

return;

//throw new InvalidOperationException();

}

RequestContext requestContext = new RequestContext(context, routeData);

IHttpHandler httpHandler = routeHandler.GetHttpHandler(requestContext);

if (httpHandler == null)

{

return;

//throw new InvalidOperationException("无法创建对应的HttpHandler对象");

}

context.RemapHandler(httpHandler);

}

}

注册路由 和Module

void Application\_Start(object sender, EventArgs e)

{

RouteCollection RouteCollection = RouteTable.Routes;

Route route = new Route("{controller}/abc", new WheelRouteHandler()) {

Defaults = new RouteValueDictionary(new { controller = "Home" }),

Constraints = new RouteValueDictionary(),

DataTokens = new RouteValueDictionary()

};

RouteCollection.Add("",route); //将路由添加到RouteCollection的RouteData集合中

}

<system.webServer>

<modules runAllManagedModulesForAllRequests="true">

<add name="WheelRoutingModule" type="WebApplication2.WheelHttp.WheelRoutingModule"/>

</modules>

</system.webServer>

运行原理

1在配置文件中注册WheelRoutingModule，Iss会载入WheelRoutingModule，载入时会运行Init函数并传入HttpApplication，此时在Init中添加HttpApplication的PostResolveRequestCache 事件（HttpApplication中的事件在每次请求时都会调用，HttpApplication中的事件的参数(object sender, EventArgs e)，sender为HttpApplication）

2在Application\_Start添加路由

RouteCollection RouteCollection = RouteTable.Routes;

Route route = new Route("{controller}/abc", new WheelRouteHandler()) {

Defaults = new RouteValueDictionary(new { controller = "Home" }),

Constraints = new RouteValueDictionary(),

DataTokens = new RouteValueDictionary()

};

RouteCollection.Add("",route);

3 PostResolveRequestCache 中实现

3.1根据上下文文本从路由控制器中查找出匹配的RouteData

RouteData routeData = RouteTable.Routes.GetRouteData(context);

3.2从RouteData 中取出路由routeHandler (RouteData的RouteHandler指向的是IRouteHandler接口 ，IRouteHandler接口自己实现)

IRouteHandler routeHandler = routeData.RouteHandler;

3.3生成请求文本

RequestContext requestContext = new RequestContext(context, routeData);

3.4调用routeHandler 的GetHttpHandler取出httpHandler （从RouteHandler取出IHttpHandler ，IHttpHandler 接口自己实习）

IHttpHandler httpHandler = routeHandler.GetHttpHandler(requestContext);

3.5执行IHttpHandler （只有实习了IHttpHandler 接口的类才可被执行）

context.RemapHandler(httpHandler);