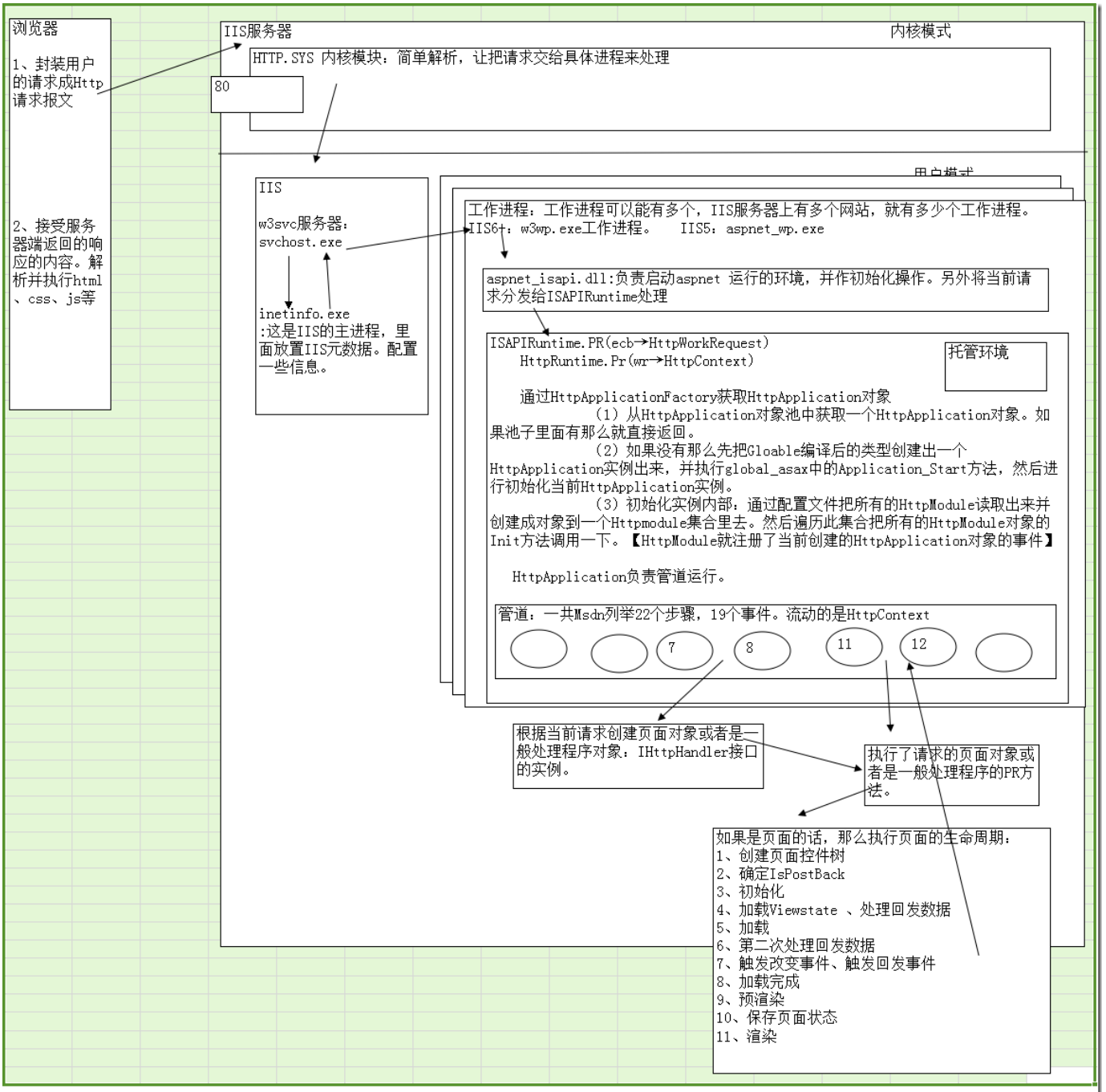
ASP.NET路由模型解析 - 文章 - 伯乐在线

大话ASP.NET模型首先我们先来了解下一个请求的悲欢离合的命运，看看它的一生中所走过的蜿蜒曲折的道路。如下图所示：（在这里感谢马伦老师唾沫横飞的讲解，终于让我理解它的一生。）



在如上所示的风光旖旎的画卷中，我们可以看到一个“请求”从客户端浏览器出发，经历千山万水到达

服务器，服务器的内核模块的HTTP.SYS热情款待了它，对它进行简单的修饰之后，就和它依依惜别了，因为HTTP.SYS知道它是一个有梦想的“请求”，它应该去它该去的地方，于是就把它送到了IIS。

IIS是片神奇的土地，这里有一位伟大的神灵叫做inetinfo.exe，于是它便去神灵的居所W3SVC服务（windows服务）祈祷，希望能给他一些指示，神灵通过查阅天书（IIS的配置文件），知道了它不是一般的静态文件，不能把它直接送回去，应该让它去它的族人开办的加工厂（即对应网站的工作进程中）好好修习一番。

现任加工厂老大叫w3wp.exe，在IIS6以前是aspnet_wp.exe,其因为没有管理好各个加工厂之间的地盘问题被罢免了（asp.net_wp.exe用一个进程寄宿所有的网站，用应用程序域进行分割的，结果导致网站之间相互影响），现任老大w3wp.exe通过一个网站一个进程的方式把问题解决了，因此顺利上位。

初入加工厂的“请求”拜访了门卫asp.net_isapi.dll,门卫发现它是第一个过来的“请求”，于是为它打开了工厂的生产车间（即第一个请求到达时，启动了asp.net运行的环境，后来的请求就可以直接进入这个环境里。），并请车间主任ISAPIRuntime来负责它，主任兴高采烈的来欢迎它（即ISAPIRuntime调用ProcessRequest（简称PR）方法，访问当前请求所在的ecb句柄），并让土里土气的它换上了统一服装HttpWorkRequest（即把请求进行简单的封装），然后叫来班长HttpRuntime,让班长安排它的工作。

班长说：”车间里面有危险，你先穿上安全制服HttpContext。”（即通过PR方法把HttpWorkRequest封装成HttpContext），然后去组长宿舍（HttpApplicationFactory）准备叫一个组长（HttpApplication）来带领它，结果发现还没有组长，班长只好去招聘一个新组长。

每一个组长都是经过严格训练才能上岗的，先要熟读入厂准则Global.asax（即先编译Global.asax文件），再通过准则中Application_Start方法考验（即调用Application__Start方法），如此这般方成为一代组长。每位新任组长第一件事就是把所有的车间模块装配好，并创建好车间管道（通过读取配置文件，加载所有的IHttpModule，并调用他们的Init方法，一般init方法都是注册管道事件，之后通过BuidStepManager方法，根据经典模式或者集成模式生成对应的StepManager）。

新任组长见到“请求”，二话不说直接启动车间管道，将其丢进去。穿着安全制服HttpContext的“请求”要依次通过管道中所有的关卡（asp.net管道模型），其中在第7个关卡之后，生成了IHttpHandler类型的对象，并在第11个关卡之后执行该对象的ProcessRequest方法处理请求，在这里“请求”得到完美的加工塑造，生成了HttpResponse，再通过剩下的管道，实现了梦想的请求就沿着原路返回了。上图中第11、12个事件之间描述的是WebForm的Page对象处理请求的流程（即页面生命周期）。

至此，一个请求的跌宕起伏的人生就说完了，各位观众欲知路由模块具体怎么发挥作用的，还请先捧个人场，右下角点个赞。

路由模型解析

通过上文我们知道组长HttpApplication对象会负责组装所有的IHttpModule，它是如何加载的呢？我们观察反编译的代码：

C#

```
public virtual void PostResolveRequestCache(HttpContextBase context)
{
```

```
        RouteData routeData = this.RouteCollection.GetRouteData(context);//匹配路由，得到匹配结果RouteData。
        if (routeData != null)
        {
                IRouteHandler routeHandler = routeData.RouteHandler;
                if (routeHandler == null)
                {
                        throw new InvalidOperationException(string.Format(CultureInfo.CurrentCulture, SR.GetString("UrlRoutingModule_NoRouteHandler"), new object[0]));
                }
                if (!(routeHandler is StopRoutingHandler))
                {
                        RequestContext requestContext = new RequestContext(context, routeData);
                        context.Request.RequestContext = requestContext;
                        IHttpHandler httpHandler = routeHandler.GetHttpHandler(requestContext);//获取处理当前请求的IHttpHandler对象。
                        if (httpHandler == null)
                        {
                                object[] args = new object[] { routeHandler.GetType() };
                                throw new InvalidOperationException(string.Format(CultureInfo.CurrentUICulture, SR.GetString("UrlRoutingModule_NoHttpHandler"), args));
                        }
                        if (httpHandler is UrlAuthFailureHandler)
                        {
                                if (!FormsAuthenticationModule.FormsAuthRequired)
                                {
                                        throw new HttpException(0x191, SR.GetString("Assess_Denied_Description3"));
                                }
                                UrlAuthorizationModule.ReportUrlAuthorizationFailure(HttpContext.Current, this);
                        }
                        else
                        {
                                context.RemapHandler(httpHandler);//映射：用当前IHttpHandler对象处理请求。
                        }
                }
        }
```

```
}
```

代码已经加了注释，3步走：匹配路由→获取处理当前请求的IHttpHandler对象→映射：用当前IHttpHandler对象处理请求。之后会在第11、12个事件之间调用IHttpHandler对象的PR方法处理当前请求。

我们再整理下思路：ASP.NET先注册了UrlRoutingModule模块，他就是一个实现了IHttpModule接口的类，其Init方法就是在第7个事件上注册一个方法，该方法先匹配路由，如果匹配成功了，则用匹配结果RouteData中的IHttpHandler对象映射到当前上下文中，这样在之后第11、12个事件之间就会调用这个IHttpHandler对象处理请求。

那么问题来了，Route对象是什么时候注入进去的，IHttpHandler对象又是谁？

还记得路由规则是怎么添加的吗？如下面代码所示：

C#

```
 /// <summary>
/// 实现了IRouteHandler接口的类型
/// </summary>
internal class MyRouteHandler : IRouteHandler
{
        public IHttpHandler GetHttpHandler(RequestContext requestContext)
        {
                //返回一个Page对象，用于处理请求。
                return new WebForm1();
        }
}
```

其实这两种方式没有本质上的区别，因为方式二中路由规则参数都会实例化一个Route对象的。

我们分析方式二的源代码：

C#

```
1
2
3
4
5
6
7
8
9
10
public Route MapPageRoute(string routeName, string routeUrl, string physicalFile, bool
checkPhysicalUrlAccess, RouteValueDictionary defaults, RouteValueDictionary constraints,
RouteValueDictionary dataTokens)
```

```
{
        if (routeUrl == null)
        {
                throw new ArgumentNullException("routeUrl");
        }
        Route item = new Route(routeUrl, defaults, constraints, dataTokens, new
PageRouteHandler(physicalFile, checkPhysicalUrlAccess));
        this.Add(routeName, item);
        return item;
}
```

发现所有的路由规则参数都用来实例化一个Route对象了，其中参数physicalFile和checkPhysicalUrlAccess用来实例化PageRouteHandler对象了，其源码如下：

C#