

Asp.Net 构架

Part.3 - Http Module 介绍

张子阳

www.tracefact.net

jimmy_dev@163.com

引言

[Http 请求处理流程](#) 和 [Http Handler 介绍](#) 这两篇文章里，我们首先了解了Http请求在服务器端的处理流程，随后我们知道Http请求最终会由实现了IHttpHandler接口的类进行处理(应该记得Page类实现了IHttpHandler)。从 [Http 请求处理流程](#) 一文的最后的一幅图中可以看到，在Http请求由IHttpHandler处理之前，它需要通过一系列的Http Module；在请求处理之后，它需要再次通过一系列的Http Module，那么这些Http Module是如何组成的？用来做什么呢？本文将对Http Module作以介绍。

Http Module 概述

暂时先不考虑我们自己实现 Http Module 的情况。在.Net 中，Http Module 是实现了 IHttpModule 接口的程序集。IHttpModule 接口本身并没有什么好大写特写的，由它的名字可以看出，它不过是一个普普通通的接口而已。实际上，我们关心的是实现了这些接口的类，如果我们也编写代码实现了这个接口，那么有什么用途。一般来说，我们可以将 Asp.Net 中的事件分成三个级别，最顶层是 应用程序级事件、其次是页面级事件、最下面是控件级事件，事件的触发分别与 应用程序周期、页面周期、控件周期 紧密相关。而 **Http Module 的作用是与 应用程序事件 密切相关的。**

我们通过 Http Module 在 Http 请求管道(Pipeline)中注册期望对应用程序事件做出反应的方法，在相应的事件触发的时候(比如说 BeginRequest 事件，它在应用程序收到一个 Http 请求并即将对其进行处理时触发)，便会调用 Http Module 注册了的方法，实际的工作在这些方法中执行。 .Net 本身已经有很多的 Http Module ， 其中包括 表单验证 Module(FormsAuthenticationModule)， Session 状态 Module(SessionStateModule)，输出缓存 Module (OutputCacheModule)等。

注册 Http Module

在注册我们自己编写的 Http Module 之前，先来看看 Asp.Net 中已经有的HttpModule。与 Http Handler 类似，我们需要打开机器上 C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\CONFIG 目录下的 web.config 文件。找到 <httpModules/> 结点，应该可以看到下面的内容：

```
<httpModules>
  <add name="OutputCache" type="System.Web.Caching.OutputCacheModule" />
  <add name="Session" type="System.Web.SessionState.SessionStateModule" />
  <add name="WindowsAuthentication"
type="System.Web.Security.WindowsAuthenticationModule" />
  <add name="FormsAuthentication"
type="System.Web.Security.FormsAuthenticationModule" />
  <add name="PassportAuthentication"
type="System.Web.Security.PassportAuthenticationModule" />
```

```

<add name="RoleManager" type="System.Web.Security.RoleManagerModule" />
<add name="UrlAuthorization" type="System.Web.Security.UrlAuthorizationModule"
/>
... 略
</httpModules>

```

我们先从结点上看，type 属性与上一节所说的 http handler 结点的 type 属性类似，都代表了相应的程序集。但是，与 http handler 不同，module 只提供了一个 name 属性，没有诸如 path 这样指定某一特定(或者用通配符 * 代表某一类)文件的处理程序。这是与 Module 的特点相关的，我们知道 module 是响应应用程序周期中触发的事件，对于所有提交到 aspnet_isapi.dll 的请求都一样，即便请求只是像类似 http://www.tracefact.net/images/logo.gif 这样获取一张图片而已(对 ISAPI 进行过设置以后，默认 aspnet_isapi.dll 不接手图片文件)。

与 Http handler 类似，在这册我们自己的 http module 时，假设类名为 ModuleDemo，位于 myNameSpace 命名空间下，程序集名称为 myDll，我们只需将 myDll.dll 拷贝到 Bin 目录下，并在站点的 web.config 文件 system.web 结点下创建 httpModules 结点：

```

<system.web>
  <httpModules>
    <add name="CustomModuleName" type="myNameSpace.ModuleDemo, myDll" />
  </httpModules>
</system.web>

```

type 属性由分号“,”分为两部分，前面是命名空间及类名，也就是类型名；后面是程序集名。如果我们将代码创建在 App_Code 目录中，则不需要再指定程序集名。

name 属性由我们自己命名，不一定与类名相同，此处我将它命名为“CustomModuleName”。我们可以通过应用程序(HttpApplication)的 Modules 属性获取 HttpModuleCollection 集合，然后通过 name 属性，进一步获取 HttpModule 对象。

通过 name 属性，我们还可以在 global.asax 中文件中编写自定义 HttpModule 暴露出的事件的程序，它采用的格式是：void ModuleName_EventName(object sender, EventArgs e)。我们将在后面做更详细介绍。

Asp.Net 内置的 Http Modules

下面这张表格列出了 C:\WINDOWS\Microsoft.NET\Framework\ v2.0.50727\CONFIG 下的 Web.Config 中的 Asp.Net 内置的 Http Modules 及其主要作用。

名称	类型	功能
OutputCache	System.Web.Caching.OutputCacheModule	页面级输出缓存
Session	System.Web.SessionState.SessionStateModule	Session 状态管理
WindowsAuthentication	System.Web.Security.WindowsAuthenticationModule	用集成 Windows

		身份验证进行客户端验证
FormsAuthentication	System.Web.Security.FormsAuthenticationModule	用基于 Cookie 的窗体身份验证进行客户端身份验证
PassportAuthentication	System.Web.Security.PassportAuthenticationModule	用 MS 护照进行客户端身份验证
RoleManager	System.Web.Security.RoleManagerModule	管理当前用户角色
UrlAuthorization	System.Web.Security.UrlAuthorizationModule	判断用户是否被授权访问某一 URL
FileAuthorization	System.Web.Security.FileAuthorizationModule	判断用户是否被授权访问某一资源
AnonymousIdentification	System.Web.Security.AnonymousIdentificationModule	管理 Asp.Net 应用程序中的匿名访问
Profile	System.Web.Profile.ProfileModule	管理用户档案文件的创立及相关事件
ErrorHandlerModule	System.Web.Mobile.ErrorHandlerModule	捕捉异常,格式化错误提示字符,传递给客户端程序

我们将在后面用编程的方式来查看它。

IHttpModule 接口

看了这么多理论知识，本节将开始动手写点程序，实现自己的 Http Module。我们首先需要看下 IHttpModule 接口，它包括下面两个方法：

```
public void Init(HttpApplication context);
public void Dispose();
```

Init()：这个方法接受一个 HttpApplication 对象，HttpApplication 代表了当前的应用程序，我们需要在这个方法内注册 HttpApplication 对象暴露给客户端的事件。可见，这个方法仅仅是用来对事件进行注册，而实际的事件处理程序，需要我们另外写方法。

整个过程很好理解：

1. 当站点第一个资源被访问的时候，Asp.Net 会创建 HttpApplication 类的实例，它代表着站点应用程序，同时会创建所有在 Web.Config 中注册过的 Module 实例。

2. 在创建 Module 实例的时候会调用 Module 的 Init() 方法。
3. 在 Init() 方法内，对想要作出响应的 HttpApplication 暴露出的事件进行注册。（仅仅进行方法的简单注册，实际的方法需要另写）。
4. HttpApplication 在其应用程序周期中触发各类事件。
5. 触发事件的时候调用 Module 在其 Init() 方法中注册过的方法。

NOTE: 如果你不了解事件注册等相关内容，请参阅 [C#中的委托与事件](#) 一文。

Dispose(): 它可以在进行垃圾回收之前进行一些清理工作。

综上所述：实现一个 IHttpModule 的模板一般是这样的：

```
public class ModuleDemo:IHttpModule
{
    public void Init(HttpApplication context) {
        // 注册 HttpApplication 应用程序 BeginRequest 事件
        // 也可以是其他任何 HttpApplication 暴露出的事件
        context.BeginRequest += new EventHandler(context_BeginRequest);
    }

    void context_BeginRequest(object sender, EventArgs e) {
        HttpApplication application = (HttpApplication)sender;
        HttpContext context = application.Context;
        // 做些实际的工作，HttpContext 对象都获得了，剩下的基本可以自由发挥了
    }

    public void Dispose() {
    }
}
```

通过 Http Module 向 Http 请求输出流中写入文字

本例中，我们仅用 BeginRequest 事件和 EndRequest 事件对 Http Module 的使用作以说明。我们通过这个范例，了解 Http Module 基本的使用方法。

首先，请创建一个新的站点，在 App_Code 目录中添加类文件： ModuleDemo.cs:

```
public class ModuleDemo:IHttpModule
{
    // Init 方法仅用于给期望的事件注册方法
    public void Init(HttpApplication context) {
        context.BeginRequest += new EventHandler(context_BeginRequest);
        context.EndRequest += new EventHandler(context_EndRequest);
    }
}
```

```

}

// 处理 BeginRequest 事件的实际代码
void context_BeginRequest(object sender, EventArgs e) {
    HttpApplication application = (HttpApplication)sender;
    HttpContext context = application.Context;
    context.Response.Write("<h1 style='color:#00f'>来自 HttpModule 的处理，请求到达</h1><hr>");
}

// 处理 EndRequest 事件的实际代码
void context_EndRequest(object sender, EventArgs e) {
    HttpApplication application = (HttpApplication)sender;
    HttpContext context = application.Context;
    context.Response.Write("<hr><h1 style='color:#f00'>来自 HttpModule 的处理，请求结束</h1>");
}

public void Dispose() {
}
}

```

上面的代码很简单，它注册了 `HttpApplication` 实例的 `BeginRequest` 事件和 `EndRequest` 事件，事件处理方法的作用仅仅是在 http 请求开始和结束的时候，给 http 请求的输入流中分别写入不同的内容。

接下来在 `Web.config` 的 `System.web` 结点中写入以下内容：

```

<system.web>
  <httpModules>
    <add name="MyModule" type="ModuleDemo" />
  </httpModules>
</system.web>

```

然后，打开建立站点时自动创建的 `Default.aspx` 文件，在里面打几个字，为了做区分，我输入的是：位于 `.aspx` 页面上的文字。然后，我们在浏览器中打开它，应该会看到像这样：



然后我们再新建一个 Default2.aspx，在浏览器中浏览，可以看到，两个页面的效果相同。这说明对于不同的两个文件，http Module 都起了作用，可见它确实是位于应用程序级，而非页面级。

现在，我们再打开站点中的一张图片文件，发现显示出的是一个红叉叉，为什么呢？因为 Http Module 针对是 http 请求，而不是某个或某一类文件，所以当请求一张图片的时候，我们编写的 http Module 依然会起作用，将文字插入到二进制图片中，破坏了文件格式，自然只能显示红叉叉了。

Note: 如果你发现你的图片显示正常，请不要惊讶，事情是这样的：回想一下第一节我们讨论到的，对于图片文件，由 IIS 直接处理，并不会交由 aspnet_isapi.dll，所以，Module 无法捕获对于图片类型文件的请求。解决方法就是在 IIS 中进行设置一下。

这里需要提请注意的是：如果你使用 Vs2005 自带的 Local Server，那么你无需对 IIS 进行设置，所有的不论图片还是任何文件类型，都会交由 aspnet_isapi.dll 处理。

遍历 Http Module 集合

现在，我们通过遍历 HttpModuleCollection 集合来查看注册给应用程序的所有 Http Module 的名称。

新建一个文件 RegisteredModules.aspx，在代码后置文件中添加如下方法：

```
private string ShowModules() {
    HttpApplication app = Context.ApplicationInstance; // 获取当前上下文的
    HttpApplication 环境
    HttpModuleCollection moduleCollection = app.Modules; //获取所有 Module 集合

    // 获取所有的 Module 名称
    string[] moduleNames = moduleCollection.AllKeys;

    System.Text.StringBuilder results = new System.Text.StringBuilder();

    //遍历结果集
    foreach (string name in moduleNames) {
        // 获得 Module 名称
        results.Append("<b style='color:#800800'>名称: " + name + "</b><br />");
        // 获得 Module 类型
        results.Append("类型: " + moduleCollection[name].ToString() + "<br />");
    }

    return results.ToString();
}
```

然后在 Page_Load 方法中输出一下：

```
protected void Page_Load(object sender, EventArgs e)
{
    Response.Write(ShowModules());
}
```

我们应该可以看到下面这样的画面：



与之前列出的那张表格比较一下，可以看出是几乎完全一致的（多了一个DefaultAuthentication）。另外注意上图的倒数第四行，那不是我们自己定义的Module么？name为MyModule，类型为ModuleDemo。

Global.asax 文件与 Http Module

早在 asp 时代，大家就知道这个文件了。它主要用于放置对于 应用程序事件 或者 Session 事件的响应程序。大家熟悉的有 Application_Start、Application_End、Session_Start、Session_End 等。

在 asp.net 中，Global 不仅可以注册应用程序和 Session 事件，还可以注册 Http Module 暴露出的事件；不仅可以注册系统 Module 的事件，也可以注册我们自己定义的 Module 暴露出的事件。在具体介绍之前，这里需要首先注意两点：

1. 在每处理一个 Http 请求时，应用程序事件都会触发一遍，但是 Application_Start 和 Application_End 例外，它仅在第一个资源文件被访问时被触发。
2. HttpModule 无法注册和响应 Session 事件，对于 Session_Start 和 Session_End，只能通过 Global.asax 来处理。

好了，我们现在修改之前 ModuleDemo 范例程序，给它像下面这样给它添加一个事件(为了使程序简洁一些，我做了简化)：

```
public class ModuleDemo : IHttpModule {

    // 声明一个事件
    public event EventHandler ExposedEvent;

    // Init 方法仅用于给期望的事件注册方法
    public void Init(HttpApplication context) {
        context.BeginRequest += new EventHandler(context_BeginRequest);
    }

    // 处理 BeginRequest 事件的实际代码
    void context_BeginRequest(object sender, EventArgs e) {
        HttpApplication application = (HttpApplication)sender;
        HttpContext context = application.Context;
        context.Response.Write("<h3 style='color:#00f'>来自 HttpModule 的处理，请求到达</h3><hr>");

        OnExposedEvent(new EventArgs()); // 调用方法
    }

    protected override void OnExposedEvent(EventArgs e) {
        if (ExposedEvent != null) // 如果 Global 中有注册
            ExposedEvent(this, e); // 调用注册了的方法
    }

    public void Dispose() {
    }
}
```

接下来，我们在站点中创建一个 Global.asax 文件，在里面添加如下代码，注意到格式是：void 模块名_事件名(object sender, EventArgs e)。

```
void MyModule_ExposedEvent(object sender, EventArgs e)
{
    Response.Write("<h3 style='color:#800800'>来自 Global.asax 的文字</h2>");
}
```

现在，我们打开之前的页面，应该可以见到这样，可见，我们成功的将 Global.asax 文件与我们自己定义的 Http Module 所暴露出的事件 ExposedEvent 联系到了一起：



总结

本文简单地介绍了什么是 Http Module。我们首先了解了 Http Module 的作用，然后查看了 Asp.Net 内置的 Module，接着我们介绍了 IHttpModule 接口，并通过了一个简单的范例实现了此接口，最后我们讨论了 Http Module 与 Global.asax 文件的联系。

本文仅仅是对 IHttpModule 作以简单介绍，对其更多的实际应用，会在后续文章中补充。

希望这篇文章能给你带来帮助！

代码下载：<http://www.tracefact.net/sourcecode/httpModule.rar>。