

Asp.Net 构架

Part. 2 – HttpHandler 介绍

张子阳

www.tracefact.net

jimmy_dev@163.com

引言

在 Part.1 [Http请求处理流程](#) 一文中，我们了解了Http请求的处理过程以及其它一些运作原理。我们知道Http管道中有两个可用接口，一个是IHttpHandler，一个是IHttpModule，但在Part.1中，我并没有详细讲述如何对它们进行编程，只是轻描淡写地一笔带过。所谓学以致用，前面已经介绍了不少概念和原理。在本文中，我们通过几个范例来了解 IHttpHandler，看看掌握这些原理的实际用途。

IHttpHandler 概述

可能和我一样，很多 Asp.Net 开发人员都有过 Asp 的背景，以至于我们在开发程序的时候，通常都是在“页面级”上思考，也就是说我们现在正在做的这个页面应该有什么样的功能，是进行一个问卷调查还是一个数据库查询等等。而很少在“请求级”思考，考虑有没有办法来通过编码的方式来操控一个 Http 请求。

实际上，Framework 提供了一系列的接口和类，允许你对于 Http 请求进行编程，而实现这一操作的一个主要的接口，就是 IHttpHandler(另一个是 IHttpModule)。

应该还记得第一节中我们提到过 ISAPI，它根据文件名后缀把不同的请求转交给不同的处理程序。但是仔细看看就会发现：几乎一大半的文件都交给 aspnet_isapi.dll 去处理了。很明显，aspnet_isapi.dll 不可能对每种文件采用同一种方式处理，那么 aspnet_isapi.dll 是如何更进一步处理不同的文件，交由谁去处理呢？为了搞清楚这个问题，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。

NOTE: 我查阅了很多资料，都说是在 machine.config 中，但实际上 v2.0.50727 下的 machine.config 中 httpHandlers 结点是这样的：<httpHandlers />，并没有给出详细的处理程序，在 Web.config 中才能看到。而 v1.1.4322 下的 machine.config 中却有。

找到 httpHandlers 结点，应该可以看到如下这样的代码(做了省略)：

```
<httpHandlers>
... ..
<add path="*.axd" verb="*" type="System.Web.HttpNotFoundHandler" validate="True" />
  <add path="*.aspx" verb="*" type="System.Web.UI.PageHandlerFactory"
validate="True" />
  <add path="*.ashx" verb="*" type="System.Web.UI.SimpleHandlerFactory"
validate="True" />
  <add path="*.asax" verb="*" type="System.Web.HttpForbiddenHandler"
validate="True" />
<add path="*.ascx" verb="*" type="System.Web.HttpForbiddenHandler" validate="True"
/>
  <add path="*.config" verb="*" type="System.Web.HttpForbiddenHandler"
validate="True" />
  <add path="*.cs" verb="*" type="System.Web.HttpForbiddenHandler" validate="True"
```

```
/>
  <add path="*" verb="GET,HEAD,POST" type="System.Web.DefaultHttpHandler"
validate="True" />
  ... ..
</httpHandlers>
```

可以看到，在<httpHandlers>结点中将不同的文件类型映射给不同的 Handler 去处理，对于.aspx 来说，是由 System.Web.UI.PageHandlerFactory 来处理。而对于.cs 来说，是由 System.Web.HttpForbiddenHandler 处理，从 ForbiddenHandler 名字中出现的 Forbidden（翻译过来是“禁止”）可以看出，这个 Handler 可以避免我们的源码被看到。

NOTE: System.Web.UI.PageHandlerFactory 是一个 IHttpHandlerFactory，而不是一个单一的 HttpHandler，IHttpHandlerFactory 用来做什么后面会说明。

上面列出的是 .Net Framework 在处理 Http 请求时所采用的默认 Handler。而如果我们要用编程的方式来操控一个 Http 请求，我们就需要实现 IHttpHandler 接口，来定制我们自己的需求。

IHttpHandler 的定义是这样的：

```
public interface IHttpHandler{
    void ProcessRequest(HttpContext context);
    bool IsReusable { get; }
}
```

由上面可以看出 IHttpHandler 要求实现一个方法和一个属性。其中 ProcessRequest，从名字(处理请求)看就知道这里应该放置我们处理请求的主要代码。

IsReusable 属性，MSDN 上是这样解释的：获取一个值，该值指示其他请求是否可以使用 IHttpHandler 实例。也就是说后继的 Http 请求是不是可以继续使用实现了该接口的类的实例，一般来说，我把它设置成 true。

那么实现此接口的类形式应该是这样的：

```
public class CustomHandler : IHttpHandler{
    public void ProcessRequest(HttpContext context) {
        // 处理请求的代码
    }
    public bool IsReusable {
        get { return true; }
    }
}
```

而为了使用这个自定义的 HttpHandler，我们需要在应用程序目录下的 Web.config 中注册它。

```
<system.web>
  <httpHandlers>
    <add path="*.jpg" verb="*" type="MyNameSpace.MyClass, MyDllName" />
  </httpHandlers>
</system.web>
```

应该发现这与之前在 C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\CONFIG\目录下 web.config 中看到的几乎完全一样。这里，path 指的是请求的文件名称，可以使用通配符扩大范围，也可以明确指定这个 handler 仅用于处理某个特定的文件(比如说：filename.aspx)的请求。verb 指的是请求此文件的方式，可以是 post 或 get，用*代表所有访问方式。type 属性由“,”分隔成两部分，第一部分是实现了接口的类名，第二部分是位于 Bin 目录下的编译过的程序集名称。

NOTE: 如果你新建一个项目，并且在项目下创建 HandlerTest.cs，然后让站点引用该项目，那么在生成解决方案的时候会自动将编译好的.dll 文件添到 Bin 目录中。

NOTE: MyDll 只写程序集名，不要加后面的.dll。

使用 HttpHandler 实现图片防盗链

有了之前这么多的准备知识，实现现在的目标就容易得多了：

NOTE: 这个例子，以及下面的一个例子均来自于《Maximizing ASP.NET Real World, Object-Oriented Development》一书：

Step. 1: 创建文件 CustomHandler.cs，代码如下：

```
using System;
using System.Web;

namespace CustomHandler{
    public class JpgHandler : IHttpHandler{
        public void ProcessRequest(HttpContext context){
            // 获取文件服务器端物理路径
            string FileName = context.Server.MapPath(context.Request.FilePath);
            // 如果 UrlReferrer 为空，则显示一张默认的禁止盗链的图片
            if (context.Request.UrlReferrer.Host == null){
                context.Response.ContentType = "image/JPEG";
                context.Response.WriteFile("/error.jpg");
            }else{
                // 如果 UrlReferrer 中不包含自己站点主机域名，则显示一张默认的禁止盗链的图片
                if (context.Request.UrlReferrer.Host.IndexOf("yourdomain.com") >
0){
                    context.Response.ContentType = "image/JPEG";
                    context.Response.WriteFile(FileName);
                }else{
```

```

        context.Response.ContentType = "image/JPEG";
        context.Response.WriteFile("/error.jpg");
    }
}

public bool IsReusable{
    get{ return true; }
}
}
}

```

Step.2 编译这个文件

```
csc /t:library /r:System.Web.dll CustomHandler.cs
```

Step.3 将编译好的 CustomHandler.dll 拷贝到站点的 Bin 目录下。

Step.4 在 Web.Config 中注册这个 Handler。

```

<system.web>
  <httpHandlers>
    <add path="*.jpg" verb="*" type="CustomHandler.JpgHandler, CustomHandler" />
  </httpHandlers>
</system.web>

```

OK, 诸位可以按步骤自行测试一下, 这里就不赘述了。

通过 IHttpHandler 实现图片验证码

也可以在一个 .ashx 文件中实现 IHttpHandler, 而不是采用这种提前编译的方式。

Step.1 打开 Vs2005, “添加新项”, “一般处理程序”。新建文件后, VS 会自动在文件中添加如下的代码:

```

<%@ WebHandler Language="C#" Class="Handler" %>

using System;
using System.Web;

public class Handler : IHttpHandler {
    public void ProcessRequest (HttpContext context) {
        context.Response.ContentType = "text/plain";
        context.Response.Write("Hello World");
    }
}

```

```

public bool IsReusable {
    get {
        return false;
    }
}
}
}

```

Step. 2 将代码改写成如下所示:

```

<%@ WebHandler Language="C#" Class="Handler" %>

using System;
using System.Drawing;
using System.Drawing.Imaging;
using System.Text;
using System.Web;
using System.Web.SessionState;

public class Handler : IHttpHandler, IRequiresSessionState {

    public void ProcessRequest(HttpContext context) {
        context.Response.ContentType = "image/gif";
        //建立 Bitmap 对象, 绘图
        Bitmap basemap = new Bitmap(200, 60);
        Graphics graph = Graphics.FromImage(basemap);
        graph.FillRectangle(new SolidBrush(Color.White), 0, 0, 200, 60);
        Font font = new Font(FontFamily.GenericSerif, 48, FontStyle.Bold,
GraphicsUnit.Pixel);

        Random r = new Random();
        string letters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
        string letter;
        StringBuilder s = new StringBuilder();

        //添加随机的五个字母
        for (int x = 0; x < 5; x++) {
            letter = letters.Substring(r.Next(0, letters.Length - 1), 1);
            s.Append(letter);
            graph.DrawString(letter, font, new SolidBrush(Color.Black), x * 38,
r.Next(0, 15));
        }

        //混淆背景
        Pen linePen = new Pen(new SolidBrush(Color.Black), 2);
        for (int x = 0; x < 6; x++)
            graph.DrawLine(linePen, new Point(r.Next(0, 199), r.Next(0, 59)), new
Point(r.Next(0, 199), r.Next(0, 59)));
    }
}

```

```

        //将图片保存到输出流中
        basemap.Save(context.Response.OutputStream, ImageFormat.Gif);
        context.Session["CheckCode"] = s.ToString(); //如果没有实现
IRequiresSessionState, 则这里会出错, 也无法生成图片
        context.Response.End();
    }

    public bool IsReusable {
        get { return true; }
    }
}

```

需要特别注意的是, Handler 类不仅需要实现 IHttpHandler 接口(这个显然), 为了在这个 Handler 类中使用 SessionState, 还需要实现 IRequiresSessionState 接口, 对于这个接口, MSDN 的解释是这样的: Specifies that the target HTTP handler requires read and write access to session-state values. This is a marker interface and has no methods. (翻译过来是: 指定当前 Http Handler 需要对 SessionState 值的读写访问权。这是一个标记接口, 没有任何方法)。

而实际上, IRequiresSessionState 的接口定义是这样的:

```
public interface IRequiresSessionState{}
```

可见, 这个接口没有任何需要实现的方法或属性, 大家只要记得: 如果想在 HttpHandler 中使用 SessionState, 必须实现这个接口, 实际上也就是在类的标头将这个接口加进去。

Step. 3 新建一个 ImageCode.aspx 页面, 在 HTML 代码中写下:

```

```

OK, 在浏览器中打开 ImageCode.aspx, 应该可以看到如下所示:



利用 HttpHandler 创建自定义后缀 Rss 源

RSS 如今已经可以说是随处可见, 而 RSS 的实现方式, 通常是在一个.aspx 的 CodeBehind 文件中写一个 XML 文件, 然后加载到 Response 的 OutputStream 中, Rss 源通常是 Rss.aspx 这种形式的。通过第一章学到的 ISAPI 的知识, 再结合本章学到的关于 HttpHandler 的知识, 很容易想到: 我们可以自定一个以 .rss 作为后缀名的文件来实现 Rss 源, 比如说 Article.rss。现在我们就一步步来实现它:

NOTE: 关于RSS的更多内容, 可以参阅我编译的 [在Web站点中创建和使用RSS源](#)。本文不再解

释Rss是什么，如何创建Rss源，为了文章的独立性，仅给出创建过程。

Step.1 创建范例数据库

```
Create Table RssSample
(
    SampleId          Int Identity(1,1)          Not Null,
    Title             Varchar(100)              Not Null Constraint uq_Title Unique,
    Author            Varchar(50)                Not Null,
    PubDate           DateTime                    Not Null Default GetDate(),
    [Description]     Varchar(500)                Not Null,
    Link              Varchar(150)                Not Null

    Constraint pk_RssSample Primary Key(SampleId)
)
-- 插入范例数据
Insert Into RssSample(Title, Author, [Description], Link)
Values('标题 1', '作者 1', '文章摘要 1', 'http://127.0.0.1/#' )

-- 省略 ....
```

Step.2 建立站点，在 App_Code 目录下建立 RssFeedsLib.cs 文件。

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.IO;
using System.Web;
using System.Xml;
using System.Text;

namespace RssFeedsLib {
    public class RssGenerator {
        public static string GetRSS() {
            MemoryStream ms = new MemoryStream();
            XmlTextWriter writer = new XmlTextWriter(ms, null);
            SqlConnection conn = new SqlConnection("Data Source=.;Initial
Catalog=Sample;User ID=sa;Password=sa"); //修改这里成你的数据库连接
            SqlCommand cmd = new SqlCommand("select * from RssSample order by pubdate
desc", conn);

            conn.Open();
            SqlDataReader reader = cmd.ExecuteReader();
            writer.WriteStartElement("rss");
            writer.WriteAttributeString("version", "2.0");
            writer.WriteStartElement("channel");
```

```

// Channel 下的结点静态写入
writer.WriteElementString("title", "TraceFact.Net 技术文章");
writer.WriteElementString("link", "http://www.tracefact.net");
writer.WriteElementString("description", "Dedicated to asp.net...");
writer.WriteElementString("copyright", "Copyright (C) 2007");
writer.WriteElementString("generator", "My RSS Generator");
// Item 结点从数据库读取
while (reader.Read()) {
    writer.WriteStartElement("item");
    writer.WriteElementString("author",
reader.GetString(reader.GetOrdinal("Author")));
    writer.WriteElementString("title",
    reader.GetString(reader.GetOrdinal("title")));
    writer.WriteElementString("link",
reader.GetString(reader.GetOrdinal("Link")));
    writer.WriteElementString("description",
reader.GetString(reader.GetOrdinal("Description")));
    writer.WriteElementString("pubDate",
reader.GetDateTime(reader.GetOrdinal("PubDate")).ToString(@"ddd, dd MMM yyyy
12:00:00 tt "));
    writer.WriteEndElement();
}

writer.WriteEndElement();
writer.WriteEndElement();
reader.Close();
conn.Close();

writer.BaseStream.Flush();
writer.Flush();
ms.Flush();

// 将流转换成 String 并返回
byte[] data = new byte[ms.Length];
ms.Seek(0, SeekOrigin.Begin);
ms.Read(data, 0, data.Length);
ms.Close();
return UTF8Encoding.UTF8.GetString(data);
}
}
}

```

Step. 3 创建可以处理 .rss 后缀名的 RssHandler

我们在这个 RssFeedsLib 命名空间下，再添加一个类，这个类用于处理对 .rss 后缀名文

件的 Http 请求。

```
public class RSSHandler:IHttpHandler{
    public bool IsReusable
    {
        get {return false;}
    }

    public void ProcessRequest(HttpContext context){
        context.Response.ContentType = "text/xml";
        string str = RssGenerator.GetRSS();
        context.Response.Write(str);
    }
}
```

Step. 4 在 Web.config 中进行配置

```
<httpHandlers>
    <add path="*.rss" type="RssFeedsLib.RSSHandler" verb="GET" />
</httpHandlers>
```

NOTE: 因为这个类和命名空间位于 App_Code 中, 这里就不需要再手动编译 RssFeedsLib.cs 然后将编译好的.dll 应用程序集放到 Bin 目录中了。至于为什么可以这样, 将会在《Asp.Net 构架与安全机制 Part. 5 - 页面生存周期与编译模型》中解释。

Step. 5 在 IIS 对 ISAPI 进行设置。

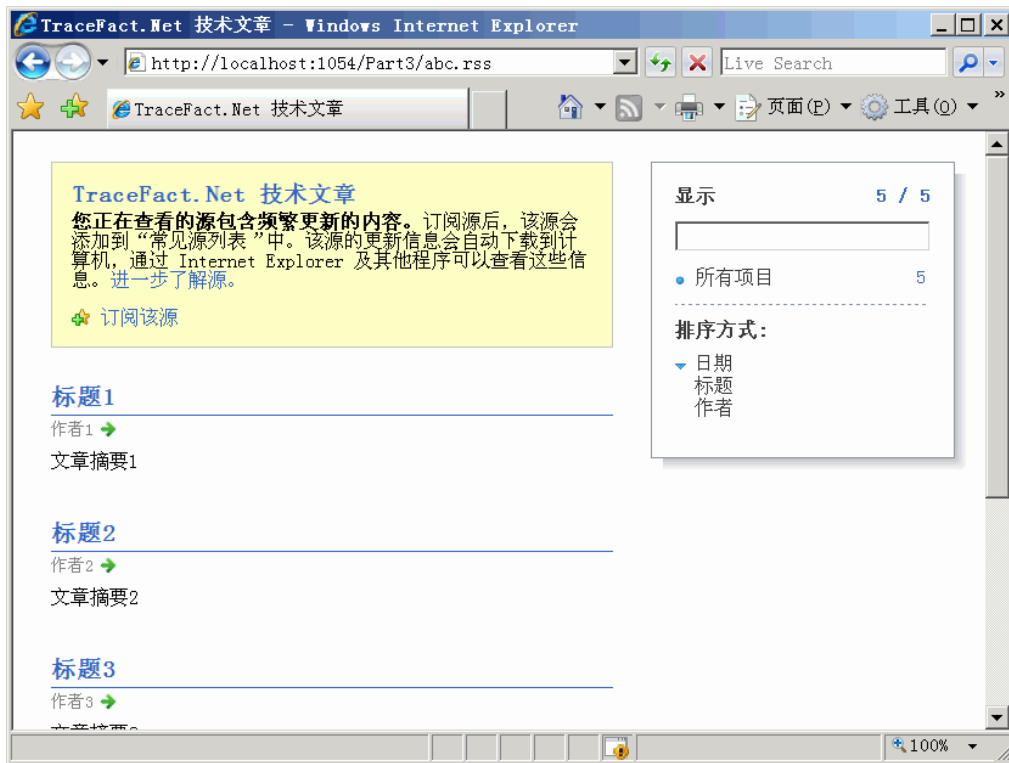
应该还记得在 Part. 1 中如何在 IIS 中设置 ISAPI 来进行文件与处理程序映射:

1. 打开 IIS, 选择本范例所用的站点, 右键, 选择“属性”。
2. 选择“主目录”选项卡, 点击“配置...”按钮。
3. 点击“添加”, 设置“可执行文件”为
“C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\aspnet_isapi.dll”, 设置“扩展名”为“.rss”, 点“确定”。
4. 注意, 不要勾选“检查文件是否存在”复选框, 这样不用创建文件, 只要在地址栏输入任意以.rss 后缀结尾的文件名, 均会交由上面创建的 Handler 去处理, 而不管这个文件是否存在, 也不管请求的是 Article.rss 还是 Sample.rss。

进行了这些设置以后, 现在 IIS 就知道如何去处理对.rss 后缀名文件的请求了。

Step. 6 测试范例

这个时候, 随便打开一个页面, 比如空白的 Default.aspx, 然后我们在地址栏将文件改为: Article.rss(改成 abc.rss 也是一样), 敲回车, 应该可以看到如下的画面。



IHttpHandlerFactory 概述

现在假设我们有这样的需求，我们不仅想要处理 .rss 后缀名，还想要能够处理 .atom 后缀名，假设处理 atom 的类命名为 AtomHandler，那么我们的 Web.config 该如何设置呢？我想应该是这样的：

```
<httpHandlers>
<add path="*.rss" type="RssFeedsLib.RSSHandler" verb="GET" />
<add path="*.atom" type="RssFeedsLib.AtomHandler" verb="GET" />
</httpHandlers>
```

如果我们有很多个 IHttpHandler 分别映射不同后缀名的请求，这样我们的 Web.config 会变得很冗长，或者，我们只有在程序运行时才能确切地知道使用哪个 Handler，这个时候，可以考虑实现 IHttpHandlerFactory 来完成这一过程。

IHttpHandlerFactory 的定义是这样的：

```
public interface IHttpHandlerFactory{
    IHttpHandler GetHandler(HttpContext context, string requestType, string url,
string pathTranslated);
    void ReleaseHandler(IHttpHandler handler);
}
```

可见，需要实现两个方法，分别是 GetHandler() 和 ReleaseHandler()。

- GetHandler(), 返回实现了 IHttpHandler 接口的类的实例。
- ReleaseHandler(), 使得 Factory 可以重复使用一个已经存在的 Handler 实例。

对于上面 .atom 和 .rss 的问题, 我们可以这样来实现 IHttpHandlerFactory 接口:

```
class HandlerFactory:IHttpHandlerFactory{
    public IHttpHandler GetHandler(HttpContext context, string requestType, string
url, string pathTranslated){
        string path = context.Request.PhysicalPath;
        if (Path.GetExtension(path) == ".rss"){
            return new RSSHandler();
        }

        if (Path.GetExtension(path) == ".atom"){
            return new ATOMHandler();
        }
        return null;
    }

    public void ReleaseHandler(IHttpHandler handler){
    }
}
```

这时, 在 Web.Config 中<system.web>节点下进行如下设置即可:

```
<httpHandlers>
<add path="*.rss,*.atom" type=" RssFeedsLib.HandlerFactory" verb="GET" />
</httpHandlers>
```

但是, 这不能简化 IIS 中 ISAPI 的设置, 还是需要手动去对 .rss 和 .atom 分别设置。

总结

在本文中, 我们首先讨论了 aspnet_isapi.dll 如何将对不同后缀名文件的请求分发给相应的处理程序, 如何查看 Framework 默认的处理程序 Handler。

然后, 我们通过三个实例, 图片防盗链、图片验证码、处理自定义后缀名请求, 详细讲解了 IHttpHandler 的实现方法和使用过程。

最后, 我向大家概要地介绍了 IHttpHandlerFactory 接口。

希望这篇文章能给你带来帮助

本文的源代码可以在 <http://www.tracefact.net/SourceCode/HttpHandler.rar> 下载。