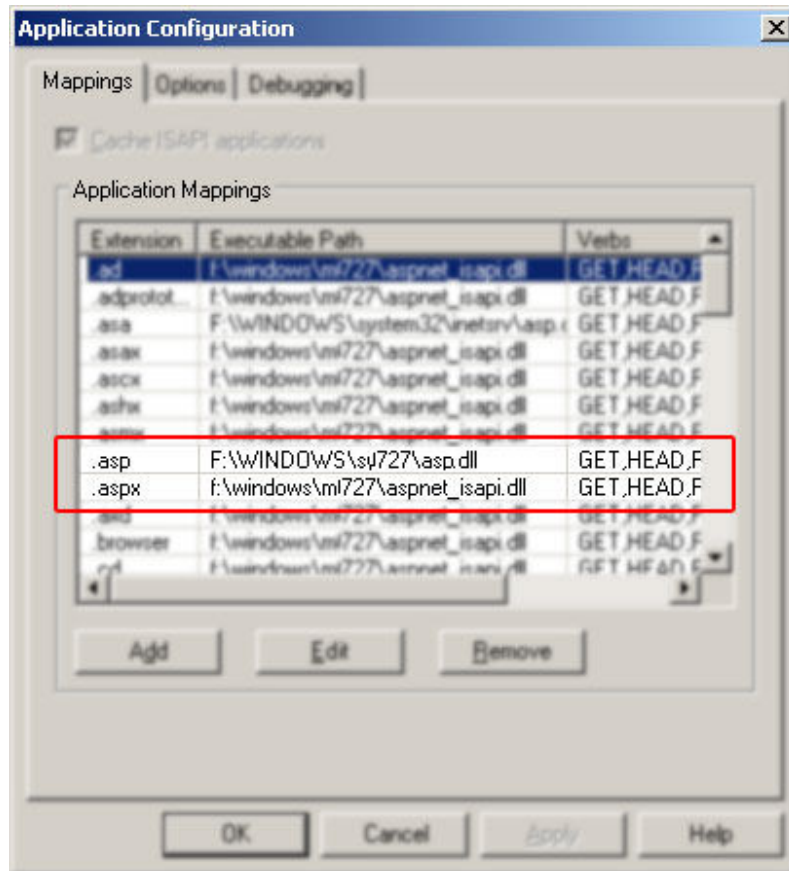


## HttpModules و HttpHandlers

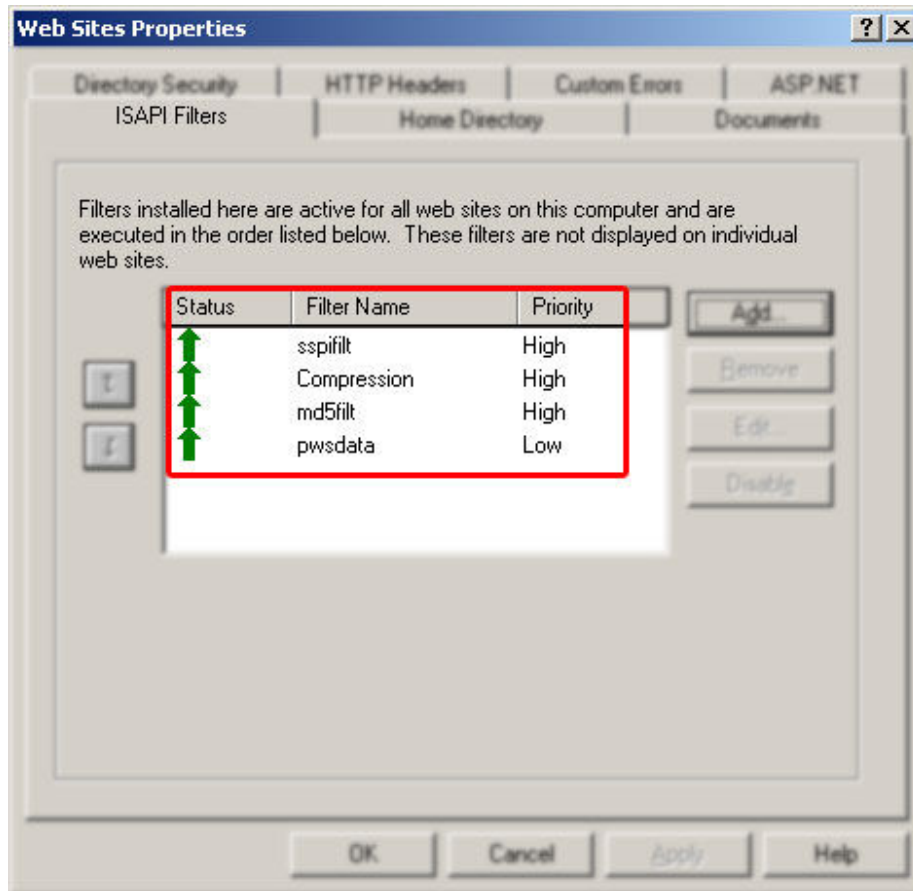
با استفاده از HttpModule و HttpHandler شما می توانید عملیات سطح پائین تری را با درخواست ها و پاسخ های Client انجام دهید. در حقیقت HttpModule و HttpHandler دو ابزار برای Componentizing کردن Web App هستند. وظیفه ی یک HttpHandler رندر کردن منبع خاص و یا نوع خاصی از یک منبع خاص می باشد. در حقیقت هر وقت که شما یک صفحه ی aspx را به پروژه خود اضافه می کنید در واقع یک HttpHandler ایجاد کرده اید. به این خاطر که هنگامی که بخشی از HTML صفحه ی aspx شما در هنگام زمان اجرا کامپایل می شود و به صورت مستقیم یا غیر مستقیم از System.Web.UI.Page به ارث می برد که این باعث پیاده سازی یک HttpHandler می شود. در حقیقت کلاس System.Web.UI.Page یک HttpHandler هستش که باعث رندر شدن صفحه ی aspx می شود. البته احتمالاً شما از این موضوع تا به حال کاملاً بی اطلاع بودید و نیازی هم نیست که بدانید در پشت صحنه چه می گذرد.

بعضی وقت ها شما نیاز دارید که در ازای شروع هر درخواست یک کد خاص اجرا شود. مثلاً فرض کنید شما نیاز دارید که بدانید کاربر به چه صفحه هایی در وب سایتتان می رود. یک راه اینکه کد Logging که این عمل را برایتان انجام می دهد را در Page\_Load هر صفحه بگذارید که آشکارا دارای هزینه ی بالائی برای Maintenance و Reusability هست. مثلاً باید برای تمام صفحات این کار را انجام دهیم و یا تصور کنید سایتتان 300 صفحه دارد، حال می خواهید در کنار Logging یک عمل دیگر نیز انجام دهید، حتی تصورش هم عذاب آور است. اما راه خوب و آسان این کار که امتیاز خوبی نیز در Maintenance و Reusability دارد، استفاده از HttpModule هست. شما یک Dll ایجاد می کنید که از IHttpModule پیروی کند و بعد تمام. اگر خواستید به راحتی می تونید از این Module در سایر وب سایت ها نیز استفاده کنید با کمترین هزینه تغییرات.

اما بیاید نگاهی دقیق تر و عمیق تر به HttpHandler ها و HttpModule ها بیاندازیم. هنگامی که یک Request به IIS از طرف Client می رسد IIS پسوند فایل درخواست شده رو بررسی می کند تا در مورد اینکه چگونه Request رو پردازش کند تصمیم گیری کند. برای محتویات static و محتویاتی که نیاز به پردازش ندارند مثل فایل های HTML، js، images و غیره IIS خود Request ها را پردازش می کند. برای فایل هایی با محتویات داینامیک IIS با توجه به ISAPI Extension های ثبت شده تصمیم می گیرد که Request را به کدام ISAPI dll تحویل دهد. یک ISAPI یک قطعه کوچک unmanaged code است که می داند چگونه باید یک نوع خاص را Render کند. برای مثال asp.dll یک ISAPI است که فایل های asp را پردازش می کند و یا aspnet\_isapi.dll یک ISAPI است که هنگام درخواست برای فایل های aspx طلبیده (invoke) می شود.



علاوه بر ISAPI، IIS اجازه تعریف و مشخص کردن ISAPI Filterها نیز را می دهد. یک ISAPI Filter یک کد unmanaged هست که می تواند به event هایی که از سمت IIS بر می آید (Raising) پاسخ دهد. در دوره ی زندگی یک Request، IIS مراحل مختلفی رو طی می کند که هر مرحله Event مخصوص به خود را تولید می کند. برای مثال هنگام شروع دریافت درخواست، هنگامی که درخواست در مورد Authentication است، هنگامی بازگرداندن محتویات Render شده به سمت Client و همانند آن. ISAPI Filterها معمولا برای فشرده سازی، دوباره نوشتن URL، عملیات Authentication و .. مورد استفاده قرار می گیرند.



هنگامی که درخواست asp.net از سمت Client به IIS ارسال می شود، IIS آن را به موتور Asp.net راهنمایی می کند تا محتویات آن توسط Asp.net مورد پردازش قرار گیرد. موتور Asp.net بسیار شبیه به IIS عمل می کند و در حین پردازش درخواست Event های مختلفی را ایجاد می کند. علاوه بر این موتور Asp.net عملیات Rendering یک Resource درخواست شده را در غالب یک کلاس ویژه نمایندگی می کند.

### نقاط پنهان پردازش درخواست

یکی از اشیاء بسیار مهم که در حقیقت حاوی درخواست IIS می باشد در Asp.net، شیء `ISAPIWorkerRequest` هست بعد از آن سطح پائین ترین شیء که حامل درخواست IIS می باشد، شیء `HttpWorkerRequest` می باشد، این شیء یکی از اعضاء خصوصی شیء `HttpContext` می باشد و شما در حالت عادی به هیچ طریقی نمی توانید به آن دسترسی داشته باشید. برای دسترسی به آن می توانید از طریق کد زیر اقدام فرمائید :

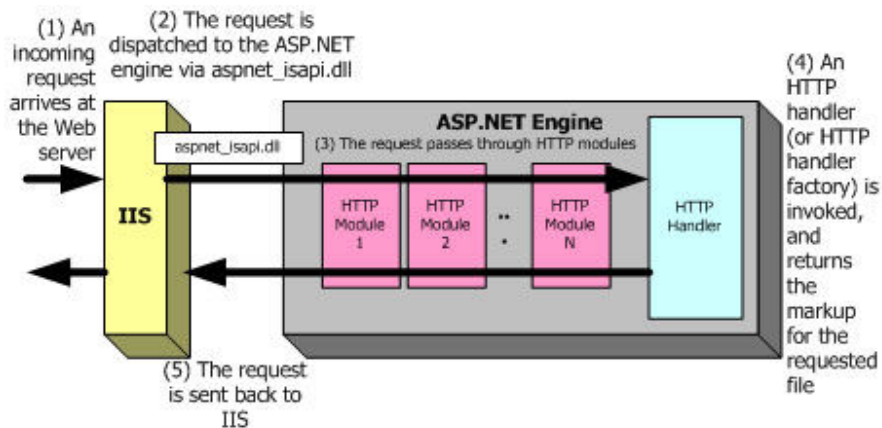
```
BindingFlags bindingFlags = BindingFlags.Instance | BindingFlags.NonPublic;
HttpWorkerRequest wr=
(HttpContext) pObjContext.GetType().GetProperty("WorkerRequest",
bindingFlags).GetValue(pObjContext, null);
```

```
HttpWorkerRequest wr=
(HttpWorkerRequest) ((IServiceProvider) context) .GetService (typeof (HttpWorkerRequest)) ;
```

خوب برگردیم به سراغ `HttpHandler` ها و `HttpModule` ها.

یک `HttpHandler` کلاسی ست که موظف به `Render` کردن نوع خاصی از منابع است. به عنوان مثال همان کلاس `Page` یک `HttpHandler` است که موظف به `Render` کردن محتویات صفحات `aspx` می باشد. یک `HttpModule` کلاسی ست که می تواند از وقایع (`Events`) مختلفی که در طی مراحل مختلف پردازش منبع در `Server` ایجاد می شود بهره برداری کند. به عنوان مثال یکی از `HttpModule` هایی رو که می توانید تصور کنید یک `ErrorLogger` می تواند باشد که هنگامی فعال می شود که یک `Exception` ناخواسته در `Web App` ایجاد شود.

تصویر زیر نمای گرافیکی از مراحل مختلف پردازش منابع `Asp.net` را نشان می دهد. توجه داشته باشید که پردازش هنگامی شروع می شود که درخواست به `IIS` برسد. فرض کنید درخواستی برای منابع `Asp.net` به `IIS` ارسال می شود. ابتدا `IIS` پسوند فایل را بررسی نموده و سپس درخواست را به `aspnet_isapi.dll` ارائه می کند. در حین چرخه زندگی درخواست ممکن است چندین `HttpModule` اجرا شوند که این بستگی به ثبت شدن و یا نشدن آنها دارد. در پایان درخواست به یک `HttpHandler` ویژه ارائه می شود. (این `HttpHandler` ویژه توسط یک `HttpHandlerFactory` شناسائی می شود.) سپس بعد از پردازش درخواست توسط `HttpHandler` درخواست چرخه برگشت خود به `IIS` را طی می کند.



`HttpHandler` ها امکاناتی دارند که می توانید با استفاده از آنها منابع و یا مسیرهای خاصی را به آنها نسبت دهید و هنگام درخواست آن منابع یا مسیرها از سمت `Client`، موتور `asp.net` این درخواست را به `HttpHandler` شما تسلیم می کند و حالا `HttpHandler` شما وظیفه `Render` کردن آن درخواست را بر عهده دارد. مثلا در مثال زیر شما تعیین می کنید که در صورتی که کاربر صفحاتی با پسوند `ashx` را درخواست کرده بود، آنگاه کنترل درخواست به `HttpHandler` ای که شما مشخص کرده اید هدایت شود. (این کد را باید در `web.config` در میان تگ های `<system.web>` و `</system.web>` قرار دهید.)

```
<httpHandlers>
  <add path="*.ashx" verb="*" type="Farahy.MyHttpHandler, MyHttpHandler"/>
</httpHandlers>
```

هر `HttpHandler` باید اینترفیس `IHttpHandler` را پیاده سازی کند. این اینترفیس دو عضو دارد. یکی خصوصیت `IsReusable` است که مشخص می کند آیا یک درخواست دیگر می تواند از نمونه ی آن استفاده کند یا خیر. هنگامی که یک درخواست از طریق `Client` به سرور ارسال می شود، موتور `Asp.net` به ازای پردازش هر درخواست یک `Thread` ایجاد می کند `IHttpHandler.IsReusable`. مشخص می کند که آیا یک نمونه از کلاس `HttpHandler` شما می تواند برای پردازش چندین درخواست همزمان مورد استفاده قرار گیرد یا خیر. در صورتی که `HttpHandler` شما مقدار `true` را برای این خصوصیت باز گرداند، کد موجود در متد `ProcessRequest` باید `ThreadSafe` بوده و همچنین مراقب باشید که هیچ گونه حالتی را نمی توانید در این مد در اختیار داشته باشید. همچنین در این حالت باید مراقب منابع مشترک که مورد استفاده ی درخواست های همزمان دیگر می باشد باشید. این حالت نسبت به حالتی دیگر (حالتی که `IHttpHandler.IsReusable` شما مقدار `false` را بر می گرداند) هزینه کمتری من باب استفاده از منابع سیستمی دارد. مثلا در مثال تولید تصاویر امنیتی با استفاده از `HttpHandler`، چون از منابع مشترک استفاده نمی کند و به حالت وابسته نیست می توان برای پائین آوردن هزینه اشغالی منابع در وضعیتی که چندین `Request` به صورت `Concurrent` پردازش می شود، از حالت `Reusable` استفاده نمود. البته کارائی این خصوصیت در سایت های بزرگ که یکباره با طیف وسیعی از درخواست ها مواجه می شوند، محسوس تر است.

پیشنهاد می کنم اگر تمایل به ایجاد یک `HttpHandler` داشتید مقدار بازگشتی `IsReusable` را با دقت انتخاب کنید. عضو دیگر `IHttpHandler` متد `ProcessRequest` است که وظیفه ی پردازش یک `Request` را بر عهده دارد.

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

public void ProcessRequest(HttpContext context)
{
    context.Response.Write("<BR/>HttpHandler :::: ProcessRequest<BR/>");
}
```

اما `element` های مهم تگ `add` میان `<httpHandlers>` و `</httpHandlers>` عبارت است از :

`path`: مشخص می کند این `HttpHandler` باید نسبت به چه مسیری و یا چه نوع منابعی گوش به زنگ باشد. به عنوان مثال `path="*.simg"`

`verb`: مشخص می کند این `HttpHandler` نسبت به چه نوع `Request` باید فعال شود. مقادیر مجاز برای این `element` عبارت است از `GET, POST, HEAD`.

با استفاده از تنظیمات زیر در فایل `web.config` شما می توانید نسبت به معرفی و ثبت یک `HttpModule` اقدام کنید.

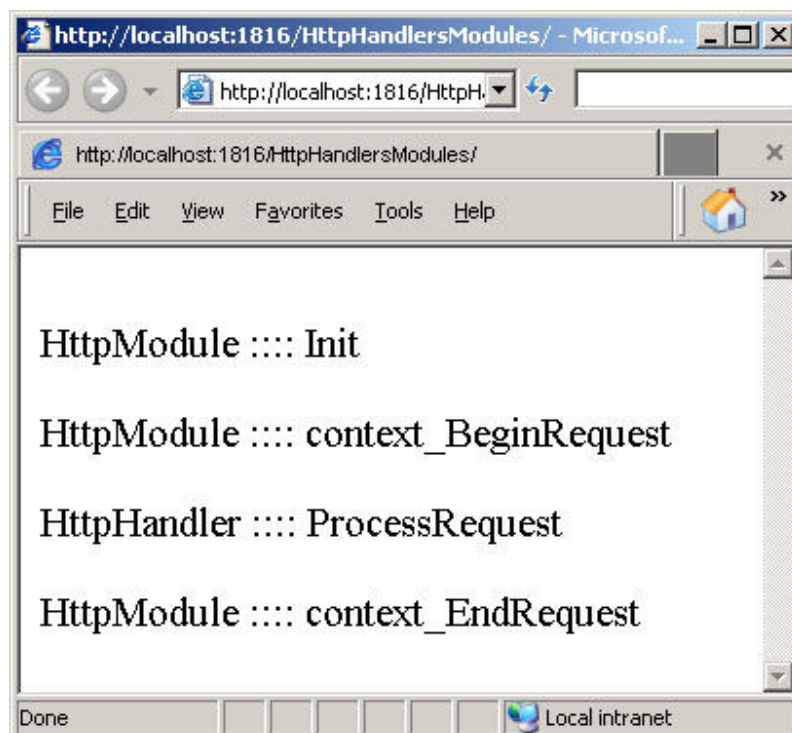
```
<add name="ModuleName"
      type=".NET Class, Assembly [,Version=version number]
          [,Culture=culture] [,PublicKeyToken=token]"/>
```

هر کلاس `HttpModule` باید اینترفیس `IHttpModule` را پیاده سازی کند. این اینترفیس دو عضو دارد. یکی متد مهم `Init` است که در آن شما به شیئی `HttpApplication` دسترسی دارید و با استفاده از آن می توانید وقایعی را که نیاز به `Handle` کردن دارند را مشخص کنید. مثلا در کد زیر من دو واقعه ی `BeginRequest` و `EndRequest` را `Handle` می کنم.

```
public void Init(HttpApplication context)
{
    HttpContext.Current.Response.Write("<BR/>HttpModule :::: Init<BR/>");
    context.BeginRequest += new EventHandler(context_BeginRequest);
    context.EndRequest += new EventHandler(context_EndRequest);
}
void context_EndRequest(object sender, EventArgs e)
{
    HttpApplication app = (HttpApplication)sender;
    app.Response.Write("<BR/>HttpModule :::: context_EndRequest<BR/>");
}
void context_BeginRequest(object sender, EventArgs e)
{
    HttpContext.Current.Response.Write("<BR/>HttpModule ::::
context_BeginRequest<BR/>");
}
```

شیئی `sender` در هر `Event` همان شیئی `HttpApplication` است.

با استفاده از مثال حاضر می توانید مراحل اجرای یک `HttpModule` و `HttpHandler` را مشاهده کنید.



---

نویسنده : سید محمد رضا فراحی

آدرس Email : [mfarahy@gmail.com](mailto:mfarahy@gmail.com)

تمامی حقوق این مقاله متعلق به سایت [www.DotNetSource.com](http://www.DotNetSource.com) می باشد .  
هر گونه اقتباس و یا برداشت از مطالب فوق با ذکر منبع آن بلامانع است .