## 一致的外观

在因特网上，您会发现很多网站都具有一致的外观和风格：

1. 每个页面有相同的头部
2. 每个页面有相同的底部
3. 每个页面有相同的样式和布局

（1）您还可以使用布局模板（布局文件）为站点的所有网页定义一致的布局。通过 Web Pages，您可以使用 **@RenderPage()** 方法从不同的文件导入内容。

内容块（来自另一个文件）能被导入网页中的任何地方。内容块可以包含文本，标记和代码，就像任何普通的网页一样。将共同的头部和底部写成单独的文件，这样会帮您节省大量的工作。您不必在每个页面中书写相同的内容，当内容有变动时，您只要修改头部或者底部文件，就可以看到站点中的每个页面的相应内容都已更新。

<html>  
<body>  
@RenderPage("header.cshtml")  
<h1>Hello Web Pages</h1>   
<p>This is a paragraph</p>  
@RenderPage("footer.cshtml")  
</body>  
</html>

1. 另一种创建一致外观的方法是使用布局页。一个布局页包含了网页的结构，而不是内容。当一个网页（内容页）链接到布局页，它会根据布局页（模板）的结构进行显示。布局页中使用 **@RenderBody()** 方法嵌入内容页，除此之外，它与一个正常的网页没有什么差别。每个内容页都必须以**布局指令**开始。

<html>  
<body>  
<p>This is header text</p>  
@RenderBody()  
<p>&copy; 2012 Runoob. All rights reserved.</p>  
</body>  
</html>

## @{Layout="Layout.cshtml";} <h1>Welcome to Runoob.com</h1>

## 在 Web 启动之前：\_AppStart

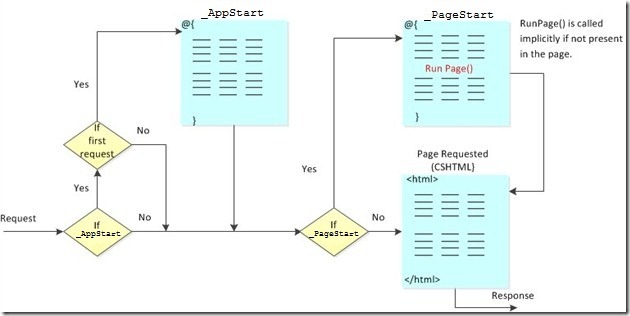
## \_AppStart 的典型用途是启动代码和初始化全局数值（比如计数器和全局名称）。

## 在每一个页面之前：\_PageStart

就像 \_AppStart 在您的站点启动之前就运行一样，您可以编写在每个文件夹中的任何页面之前运行的代码。

对于您网站中的每个文件夹，您可以添加一个名为 \_PageStart 的文件。

\_PageStart 的典型用途是为一个文件夹中的所有页面设置布局页面，或者在运行某个页面之前检查用户是否已经登录。



当接收到一个请求时，ASP.NET 会首先检查 \_AppStart 是否存在。 如果 \_AppStart 存在且这是站点接收到的第一个请求，则运行 \_AppStart。

然后 ASP.NET 检查 \_PageStart 是否存在。如果 \_PageStart 存在，则在其它被请求的页面运行之前先运行 \_PageStart。

您可以在 \_PageStart 中调用 RunPage() 来指定被请求页面的运行位置。否则，默认情况下，被请求页面是在 \_PageStart 运行之后才被运行。

### 使用ASP.NET Identity 身份验证

有了对身份验证和授权机制基本了解后，那么现在就该使用ASP.NET Identity 进行身份验证了。

**1. 实现身份验证所需的准备工作**

当我们匿名访问授权资源时，会被Redirect 到 /Account/Login 时，此时的URL结构如下：

**http://localhost:60533/Account/Login?ReturnUrl=%2Fhome%2Findex**

因为需要登陆，所以可以将Login 设置为允许匿名登陆，只需要在Action的上面添加 **[AllowAnonymous]** 特性标签，如下所示：

注意，在这儿我将ReturnUrl 存储了起来，ReturnUrl 顾名思义，当登录成功后，重定向到最初的地址，这样提高了用户体验。

[AllowAnonymous]

public ActionResult Login(string returnUrl)

{

//如果登录用户已经Authenticated，提示请勿重复登录

if (HttpContext.User.Identity.IsAuthenticated)

 {

   return View("Error", new string[] {"您已经登录！"});

 }

   ViewBag.returnUrl = returnUrl;

   return View();

}

由于篇幅的限制，Login View 我不将代码贴出来了，事实上它也非常简单，包含如下内容:

* 用户名文本框
* 密码框
* 存储ReturnUrl的隐藏域
* @Html.AntiForgeryToken()，用来防止CSRF跨站请求伪造

**2.添加用户并实现身份验证**

当输入了凭据之后，POST Form 表单到/Account/Login 下，具体代码如下：

## 在 .NET Framework 应用程序中使用基于角色的安全性

每个应用程序线程都可以具有一个关联的主体对象（可通过 [CurrentPrincipal](https://msdn.microsoft.com/zh-cn/library/075dt2hk(v=vs.100)) 属性访问），该对象表示代码正在代表其运行的用户的安全上下文。 主体对象封装用户的用户安全上下文（或标识）对象。 使用两个对象可允许分离身份验证（在标识对象中）和授权（在主体中）。

标识对象必须实现 [IIdentity](https://msdn.microsoft.com/zh-cn/library/cfh703cy(v=vs.100)) 接口。 标识对象表示特定用户，并公开所要求的以下属性[IIdentity](https://msdn.microsoft.com/zh-cn/library/cfh703cy(v=vs.100))接口： [Name](https://msdn.microsoft.com/zh-cn/library/9cba0wt3(v=vs.100))、[IsAuthenticated](https://msdn.microsoft.com/zh-cn/library/z2x1475h(v=vs.100)) 和 [AuthenticationType](https://msdn.microsoft.com/zh-cn/library/w3yfx519(v=vs.100))。 标识对象通常具有执行用户身份验证的其他私有成员。

主体对象必须实现 [IPrincipal](https://msdn.microsoft.com/zh-cn/library/f8kt7fb8(v=vs.100)) 接口。 主体对象封装用户的安全上下文将公开下列成员所要求的[IPrincipal](https://msdn.microsoft.com/zh-cn/library/f8kt7fb8(v=vs.100))接口： [IsInRole](https://msdn.microsoft.com/zh-cn/library/3cdewxwy(v=vs.100))方法，它执行授权，和[Identity](https://msdn.microsoft.com/zh-cn/library/499z5w4e(v=vs.100))属性，它提供了对用户的标识对象的访问。

**过滤器**



用户通过URL访问Web系统不一定都能得到相应的内容，一方面不同的用户权限不同，另一方面是为了保护系统，防止被攻击，这就是**过滤器的核心**所在，我们总计一下过滤器都有哪些作用：

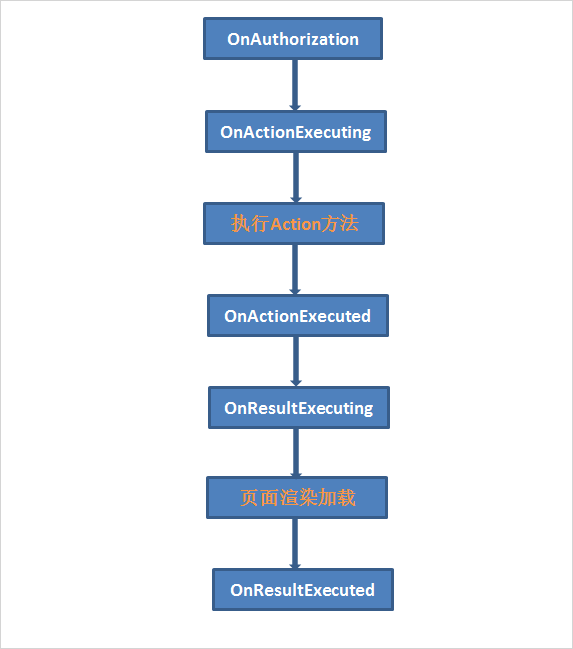
　　①：判断用户是否登录以及不同用户对应不同的权限问题。

　　②：防盗链、防爬虫。

　　③：系统中语言版本的切换(本地化和国际化)。

　　④：权限管理系统中动态Action。

　　⑤：决策输出缓存。



# [ASP.NET三剑客 HttpApplication HttpModule HttpHandler 解析](https://www.cnblogs.com/songwenjie/p/9064119.html)

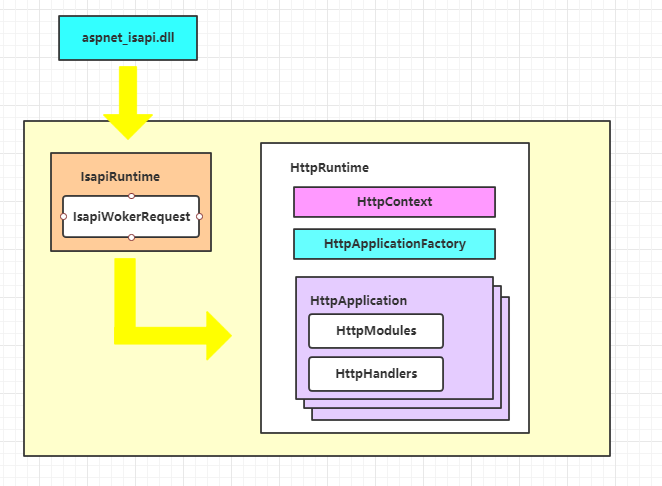
这里以IIS6.0为例，它在工作进程**w3wp.exe**中会利用**aspnet\_isapi.dll**加载.NET运行时。IIS6.0引入了应用程序池的概念，一个工作进程对应着一个应用程序池。一个应用程序池可以承载一个或多个Web应用。如果**HTTP.SYS**（HTTP监听器，是Windows TCP/IP网络子程序的一部分，用于持续监听HTTP请求）接收的请求是对该Web应用的第一次访问，在成功加载运行时后，IIS会通过AppDomainFactory为该Web应用创建一个应用程序域。也就是说一个应用程序池中会有多个应用程序域，它们共享一个工作进程资源，但是又不会互相牵连影响。

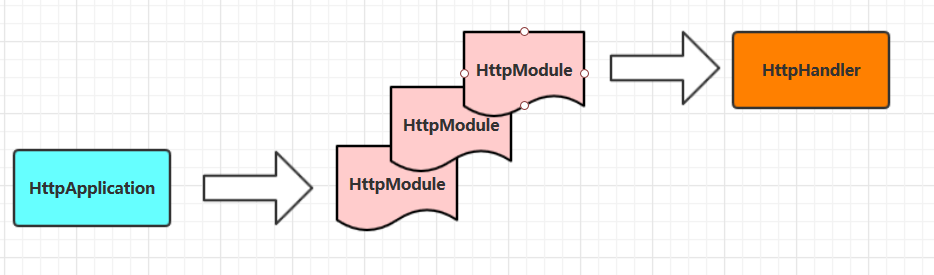
随后一个特殊的运行时**IsapiRuntime**被加载，会接管该HTTP请求。**IsapiRuntime**首先会创建一个**IsapiWorkerRequest**对象来封装当前的HTTP请求，随后将此对象传递给ASP.NET运行时**HttpRunTime**。从此时起，HTTP请求正式进入了ASP.NET管道。

**HttpRunTime**会根据**IsapiWorkerRequest**对象创建用于表示当前HTTP请求的上下文对象**HttpContext**。随着**HttpContext**对象的创建，**HttpRunTime**会利用**HttpApplicationFactory**创建或获取现有的**HttpApplication**对象。

**HttpApplication**负责处理当前的HTTP请求。在**HttpApplication**初始化过程中，ASP.NET会根据配置文件加载并初始化注册的**HttpModule**对象。对于**HttpApplication**来说，在它处理HTTP请求的不同阶段会触发不同的事件，而**HttpModule**的意义在于通过注册**HttpApplication**的相应事件，将所需的操作注入整个HTTP请求的处理流程。

最终完成对HTTP请求的处理在**HttpHandler**中，不同的资源类型对应着不同类型的**HttpHandler**。





每个Web应用程序都会有一个Global实例，作为应用程序的唯一入口。我们知道ASP.NET应用程序启动时，ASP.NET运行时只调用一次Application\_Start。这似乎意味着在我们的应用程序中只有一个Global对象实例，但是可不是只有一个HttpApplication对象实例。

ASP.NET运行时维护一个HttpApplication对象池。当第一个请求抵达时，ASP.NET会一次创建多个HttpApplication对象，并将其置于HttpApplication对象池中，然后选择其中一个对象来处理该请求。当后续请求到达时，运行时会从池中获取一个HttpApplication对象与请求进行配对。该对象与请求相关联，并且只有该请求，直到请求处理完成。当请求完成后，HttpApplication对象不会被回收，而是会返回到池中，以便稍后将其拉出为其他请求提供服务。通过使用HttpApplication对象来处理到的请求，HttpApplication对象每次只能处理一个请求，这样其成员才可以于储存针对每个请求的数据。下面我们来了解一下HttpApplication的成员。



我们可以通过HttpModule注册相应的事件，将处理逻辑注入到HttpApplication处理请求的某个阶段。这里需要注意的是，从BeginRequest开始的事件，并不是每个管道事件都会被触发。因为在整个处理过程中，随时可以调用Response.End()或者有未处理的异常发生而提前结束整个过程。所有事件中，只有EndRequest事件是肯定会触发的，(部分Module的)BeginRequest有可能也不会被触发。这个我们会在后面的HttpModule中提及。