**不改代码就能优化ASP.NET网站性能的方法**

2012-12-24 09:23 Fish Li 博客园 [我要评论(0)](http://developer.51cto.com/art/201212/373937.htm#commment) 字号：**[T](javascript:setfont(12);" \t "_self)** | [**T**](javascript:setfont(16);)

[一键收藏，随时查看，分享好友！](javascript:favorBox('open');)

不改代码就能优化ASP.NET网站性能的方法，它们主要分为二个方面：1. 利用ASP.NET自身的扩展性进行优化。2. 优化IIS设置。

AD：[51CTO 网+ 第十二期沙龙：大话数据之美\_如何用数据驱动用户体验](http://mobile.51cto.com/mobile/mdsa12/" \t "_blank)

本文将介绍一些方法用于优化ASP.NET网站性能，这些方法都是不需要修改程序代码的。

它们主要分为二个方面：

1. 利用ASP.NET自身的扩展性进行优化。

2. 优化IIS设置。

**配置OutputCache**

用缓存来优化网站性能的方法，估计是无人不知的。 ASP.NET提供了HttpRuntime.Cache对象来缓存数据，也提供了OutputCache指令来缓存整个页面输出。虽然OutputCache指令使用起来更方便，也有非常好的效果，不过，它需要我们在那些页面中添加这样一个指令。

对于设置过OutputCache的页面来说，浏览器在收到这类页面的响应后，会将页面响应内容缓存起来。只要在指定的缓存时间之内，且用户没有强制刷新的操作，那么就根本不会再次请求服务端，而对于来自其它的浏览器发起的请求，如果缓存页已生成，那么就可以直接从缓存中响应请求，加快响应速度。因此，OutputCache指令对于性能优化来说，是很有意义的（除非所有页面页面都在频繁更新）。

在网站的优化阶段，我们可以用Fiddler之类的工具找出一些内容几乎不会改变的页面，给它们设置OutputCache，但是，按照传统的开发流程，我们需要针对每个页面文件执行以下操作：

1. 签出页面文件。

2. 添加OutputCache指令。

3. 重新发布页面。

4. 签入文件（如果遇到多分支并行，还可能需要合并操作）。

以上这些源代码管理制度会让一个简单的事情复杂化，那么，有没一种更简单的方法能解决这个问题呢？

接下来，本文将介绍一种方法，它利用ASP.NET自身的扩展性，以配置文件的方式为页面设置OutputCache参数。配置文件其它就是一个XML文件，内容如下：

1. **<?xml** version="1.0" encoding="utf-8"**?>**
2. **<OutputCache** xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3. xmlns:xsd="http://www.w3.org/2001/XMLSchema"**>**
4. **<Settings>**
5. **<Setting** Duration="3" FilePath="/Pages/a3.aspx"  **/>**
6. **<Setting** Duration="10" FilePath="/Pages/a5.aspx"  **/>**
7. **</Settings>**
8. **</OutputCache>**

看了这段配置，我想您应该也能猜到它能有什么作用。

每一行配置参数为一个页面指定OutputCache所需要的参数，示例文件为了简单只使用二个参数，其它可以支持的参数请参考[OutputCache指令](http://msdn.microsoft.com/zh-cn/library/hdxfb6cy%28v=VS.80%29.aspx?ppud=4" \t "_blank)。

为了能让这个配置文件有效，需要在web.config中配置以下内容（适用于IIS7）：

1. **<system.webServer>**
2. **<modules>**
3. **<add** name="SetOutputCacheModule" type="WebSiteOptimize.SetOutputCacheModule, WebSiteOptimize" **/>**
4. **</modules>**
5. **</system.webServer>**

在这里，我注册了一个HttpModule，它的全部代码如下：

1. **public** **class** SetOutputCacheModule : IHttpModule
2. {
3. **static** SetOutputCacheModule()
4. {
5. // 加载配置文件
6. **string** xmlFilePath = Path.Combine(HttpRuntime.AppDomainAppPath, "OutputCache.config");
7. ConfigManager.LoadConfig(xmlFilePath);
8. }
10. **public** **void** Init(HttpApplication app)
11. {
12. app.PreRequestHandlerExecute += **new** EventHandler(app\_PreRequestHandlerExecute);
13. }
15. **void** app\_PreRequestHandlerExecute(**object** sender, EventArgs e)
16. {
17. HttpApplication app = (HttpApplication)sender;
19. Dictionary<**string**, OutputCacheSetting> settings = ConfigManager.Settings;
20. **if**( settings == **null** )
21. **throw** **new** ConfigurationErrorsException("SetOutputCacheModule加载配置文件失败。");
23. // 实现方法：
24. // 查找配置参数，如果找到匹配的请求，就设置OutputCache
25. OutputCacheSetting setting = **null**;
26. **if**( settings.TryGetValue(app.Request.FilePath, **out** setting) ) {
27. setting.SetResponseCache(app.Context);
28. }
29. }

ConfigManager类用于读取配置文件，并启用了文件依赖技术，当配置文件更新后，程序会自动重新加载：

1. **internal** **static** **class** ConfigManager
2. {
3. **private** **static** **readonly** **string** CacheKey = Guid.NewGuid().ToString();
4. **private** **static** Exception s\_loadConfigException;
5. **private** **static** Dictionary<**string**, OutputCacheSetting> s\_settings;
7. **public** **static** Dictionary<**string**, OutputCacheSetting> Settings
8. {
9. **get**{
10. Exception exceptin = s\_loadConfigException;
11. **if**( exceptin != **null** )
12. **throw** exceptin;
14. **return** s\_settings;
15. }
16. }
18. **public** **static** **void** LoadConfig(**string** xmlFilePath)
19. {
20. Dictionary<**string**, OutputCacheSetting> dict = **null**;
22. **try** {
23. OutputCacheConfig config = XmlHelper.XmlDeserializeFromFile<OutputCacheConfig>(xmlFilePath, Encoding.UTF8);
24. dict = config.Settings.ToDictionary(x => x.FilePath, StringComparer.OrdinalIgnoreCase);
25. }
26. **catch**( Exception ex ) {
27. s\_loadConfigException = **new** System.Configuration.ConfigurationException(
28. "初始化SetOutputCacheModule时发生异常，请检查" + xmlFilePath + "文件是否配置正确。", ex);
29. }

32. **if**( dict != **null** ) {
33. // 注册缓存移除通知，以便在用户修改了配置文件后自动重新加载。
35. // 参考：细说 ASP.NET Cache 及其高级用法
36. //          http://www.cnblogs.com/fish-li/archive/2011/12/27/2304063.html
37. CacheDependency dep = **new** CacheDependency(xmlFilePath);
38. HttpRuntime.Cache.Insert(CacheKey, xmlFilePath, dep,
39. Cache.NoAbsoluteExpiration, Cache.NoSlidingExpiration, CacheItemPriority.NotRemovable, CacheRemovedCallback);
40. }
42. s\_settings = dict;
43. }
45. **private** **static** **void** CacheRemovedCallback(**string** key, **object** value, CacheItemRemovedReason reason)
46. {
47. **string** xmlFilePath = (**string**)value;
49. // 由于事件发生时，文件可能还没有完全关闭，所以只好让程序稍等。
50. System.Threading.Thread.Sleep(3000);
52. // 重新加载配置文件
53. LoadConfig(xmlFilePath);
54. }
55. }

有了AutoSetOutputCacheModule，我们就可以直接使用配置文件为页面设置OutputCache参数，而不需要修改任何页面，是不是很容易使用？

说明：[MyMVC框架](http://www.cnblogs.com/fish-li/archive/2012/02/21/2361982.html)已支持这种功能，所有相关的可以从MyMVC框架的源码中获取。

建议：对于一些很少改变的页面，缓存页是一种很有效的优化方法。

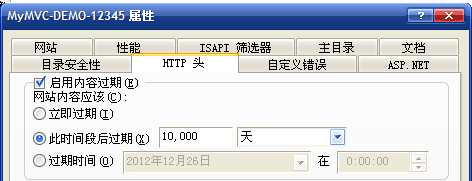
**启用内容过期**

每个网站都会有一些资源文件（图片，JS，CSS），这些文件相对于ASPX页面来说，它们的输出内容极有可能在一段长时间之内不会有变化，而IIS在响应这类资源文件时不会生成Cache-Control响应头。在这种情况下，浏览器或许会缓存它们，也许会再次发起请求（比如重启后），总之就是缓存行为不受控制且缓存时间不够长久。

有没有想过可以把它们在浏览器中长久缓存起来呢？

为了告诉浏览器将这些文件长久缓存起来，减少一些无意义的请求（提高页面呈现速度），我们可以在IIS中启用内容过期，这样设置后，IIS就能生成Cache-Control响应头，明确告诉浏览器将文件缓存多久。

在IIS6中，这个参数很好找到：

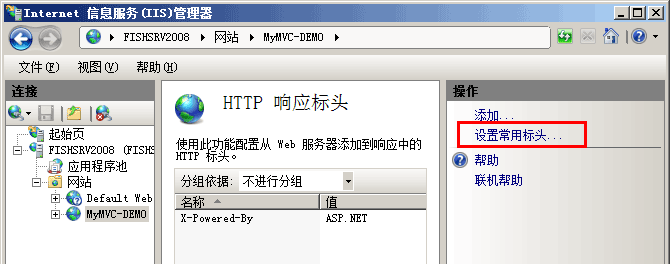


然而，在IIS7中，这个参数不容易被发现，需要以下操作才能找到：

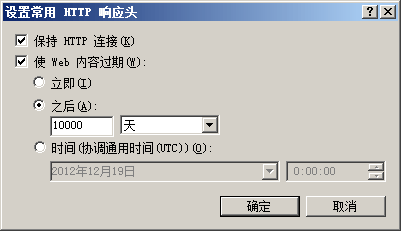
选择网站（或者网站子目录）节点，双击【HTTP响应标头】



再点击右边的【设置常用标头】链接



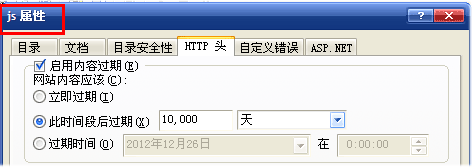
此时将会显示：



说明：【启用内容过期】这个设置可以基于整个网站，也可以针对子目录，或者一个具体的文件。

注意：如果您在IIS7中针对某个子目录或者文件设置【启用内容过期】，前面的对话框看起来是一模一样的，

然而，在IIS6中，我们可以清楚地从对话框的标题栏中知道我们在做什么：



有时真感觉IIS7的界面在退步！

最后我想说一句：可以直接为整个网站启用内容过期，ASPX页面是不会被缓存的！

说到这里可能有人会想：这个过期时间我该设置多久呢？

十分钟，2个小时，一天，还是一个月？

在我看来，这个时间越久越好。

可能有人又会说了：万一我要升级某个JS文件怎么办，时间设置久了，用户怎么更新呢？

如果你问我这个问题，我也只能说是你的代码不合理（毕竟你解决不了升级问题），想知道原因的话，请继续阅读。

**解决资源文件升级问题**

对于一些规模不大的网站来说，通常会将资源文件与程序文件一起部署到一个网站中。

这时可能会采用下面的方式来引用JS或者CSS文件：

1. <link type="text/css" href="aaaa.css" rel="Stylesheet" />
2. <script type="text/javascript" src="bbb.js"></script>

在这种情况下，如果使用了前面所说的【启用内容过期】方法，那么当有JS,CSS文件需要升级时，由于浏览器的缓存还没有过期，所以就不会请求服务器，此时会使用已缓存的版本，因此可能会出现各种奇怪的BUG

对于前面谈到的BUG，我认为根源在于引用JS,CSS文件的方式有缺陷，那种方法完全没有考虑到版本升级问题，正确的方法有二种：

1. 给文件名添加版本号，像jquery那样，每个版本一个文件(jquery-1.4.4.min.js)。

2. 在URL后面添加一个版本号，让原先的URL失效。

第一种方法由于每次升级都产生了一个新文件，所以不存在缓存问题，但是，维护一大堆文件的成本可能会比较大，因此我建议采用第二种方法来解决。

在[MyMVC](http://www.cnblogs.com/fish-li/archive/2012/02/21/2361982.html)的示例代码中，我使用了下面的方法来引用这些资源文件：

1. <%= HtmlExtension.RefCssFileHtml("/css/StyleSheet.css")%>
2. <%= HtmlExtension.RefJsFileHtml("/js/MyPage/fish.js")%>

在页面运行时，会产生如下的输出结果：

1. <link type="text/css" rel="Stylesheet" href="/css/StyleSheet.css?\_t=634642185820000000" />
2. <script type="text/javascript" src="/js/MyPage/fish.js?\_t=634642154020000000"></script>

这二个工具方法的实现代码如下（在[MyMVC](http://www.cnblogs.com/fish-li/archive/2012/02/21/2361982.html" \t "_blank)的示例代码中）：

1. **private** **static** **readonly** **string** s\_root = HttpRuntime.AppDomainAppPath.TrimEnd('\\');
3. **public** **static** **string** RefJsFileHtml(**string** path)
4. {
5. **string** filePath = s\_root + path.Replace("/", "\\");
6. **string** version = File.GetLastWriteTimeUtc(filePath).Ticks.ToString();
7. **return** **string**.Format("<script type=\"text/javascript\" src=\"{0}?\_t={1}\"></script>\r\n", path, version);
8. }
10. **public** **static** **string** RefCssFileHtml(**string** path)
11. {
12. **string** filePath = s\_root + path.Replace("/", "\\");
13. **string** version = File.GetLastWriteTimeUtc(filePath).Ticks.ToString();
14. **return** **string**.Format("<link type=\"text/css\" rel=\"Stylesheet\" href=\"{0}?\_t={1}\" />\r\n", path, version);
15. }

上面这种获取文件版本号的方法，是一种比较简单的解决方案。每个引用的地方在生成HTML代码时，都会访问文件的最后修改时间，这会给磁盘带来一点读的开销，如果您担心这种实现方式可能会给性能带来影响，那么也可以增加一个配置文件的方式来解决（请自行实现），例如以下结构：

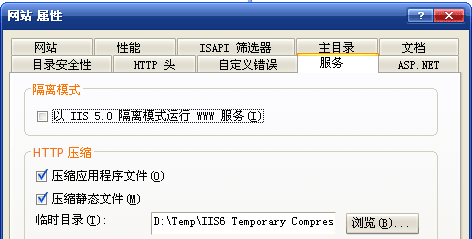
1. **<?xml** version="1.0" encoding="utf-8"**?>**
2. **<ArrayOfFileVersion** xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3. xmlns:xsd="http://www.w3.org/2001/XMLSchema"**>**
4. **<FileVersion** FilePath="/js/JScript.js" Version="255324" **/>**
5. **<FileVersion** FilePath="/css/StyleSheet.css" Version="2324235" **/>**
6. **</ArrayOfFileVersion>**

如果您认为这种配置文件需要手工维护，不够自动化，还可以采用程序的方式自动在运行时维护一个列表，总之，直接引用资源文件的方法是一种直接耦合，会给文件升级带来麻烦，我们可以通过一个外部方法来解开这个直接耦合（给FileVersion增加一个属性还还可以将内部地址改成一个CDN地址）。

**启用压缩**

压缩响应结果也是常用的网站优化方法，由于现在的浏览器都已支持压缩功能，因此，如果在服务端能压缩响应结果，对于网速较慢的用户来说，会减少很多网络传输时间，最终的体验就是网页显示速度变快了！

IIS6虽然提供压缩的设置界面，然而配置是基于服务器级别的：

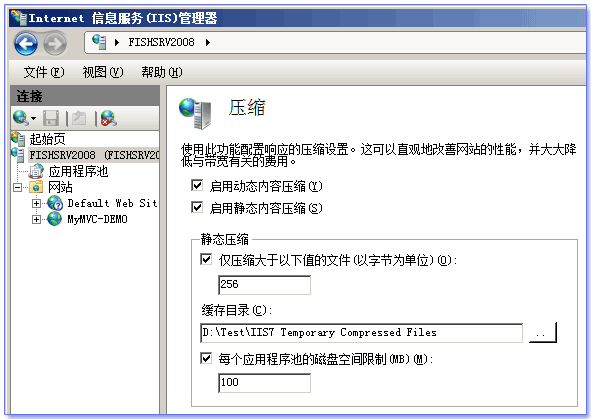


注意：这里的【应用程序文件】不包括aspx，如果需要压缩aspx的响应，需要手工修改x:\WINDOWS\system32\inetsrv\MetaBase.xml文件（参考加大字号部分）：

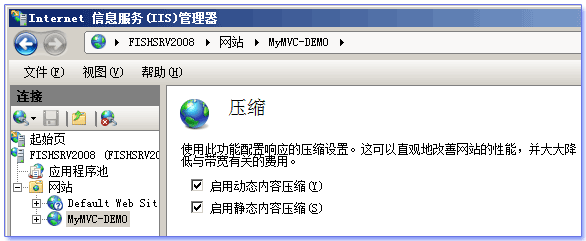
1. <IIsCompressionScheme    Location ="/LM/W3SVC/Filters/Compression/gzip"
2. HcCompressionDll="%windir%\system32\inetsrv\gzip.dll"
3. HcCreateFlags="1"
4. HcDoDynamicCompression="TRUE"
5. HcDoOnDemandCompression="TRUE"
6. HcDoStaticCompression="TRUE"
7. HcDynamicCompressionLevel="9"
8. HcFileExtensions="htm
9. html
10. txt
11. js
12. css
13. htc"
14. HcOnDemandCompLevel="10"
15. HcPriority="1"
16. HcScriptFileExtensions="asp
17. exe
18. aspx
19. axd"
20. >

说明：要修改MetaBase.xml，需要停止IIS Admin Service服务。

在IIS7中，我们可以在服务器级别配置压缩参数：



然后在每个网站中开启或者关闭压缩功能：



说明：IIS7中已经不再使用MetaBase.xml，所以我们找不到IIS6的那些设置了。 IIS7压缩的过滤条件不再针对扩展名，而是采用了mimeType规则（保存在applicationHost.config）。根据IIS7的压缩规则，当我们启用动态压缩后，会压缩aspx的响应结果。

二种压缩方法的差别：

1. 静态内容压缩：当服务器在第一次响应某个静态文件时，会生成一个压缩后的结果，并保存到磁盘中，以便重用。

2. 动态内容压缩：【每次】在响应客户端之前，压缩响应结果，在内存中完成，因此会给CPU带来一些负担。

注意：要不要【启用动态内容压缩】这个参数，需要评估服务器的CPU是否能以承受（观察任务管理器或者查看性能计数器）。

**删除无用的HttpModule**

对一个网站来说，ASP.NET提供的有些HttpMoudle可能并不是需要的，然而，**如果你不去手工禁用它们，它们其实会一直运行。  
比如 我 会禁用下面这些HttpMoudle：**

1. **<httpModules>**
2. **<remove** name="Session"**/>**
3. **<remove** name="RoleManager"**/>**
4. **<remove** name="PassportAuthentication"**/>**
5. **<remove** name="Profile"**/>**
6. **<remove** name="ServiceModel"**/>**
7. **</httpModules>**

对于使用[Forms身份认证](http://www.cnblogs.com/fish-li/archive/2012/04/15/2450571.html" \t "_blank)的网站的来说，下面这些HttpModule也是可以禁用的：

1. **<httpModules>**
2. **<remove** name="WindowsAuthentication"**/>**
3. **<remove** name="FileAuthorization"**/>**
4. **</httpModules>**

**其它优化选项**

优化ASP.NET网站是一个大的话题，除了博客中介绍的这些方法之外，还有以下方法也是可以参考的：

1. 升级服务器硬件配置。

2. 使用Windows Server 2008以上版本操作系统（网络性能比2003要好）。

3. 优化操作系统配置（例如禁用不需要的服务）。

4. 禁用调试模式。

5. 网站使用专用应用程序池。

原文链接：<http://www.cnblogs.com/fish-li/archive/2012/12/23/2830301.html>