**[优化网站设计（十一）：避免重定向](http://www.cnblogs.com/chenxizhang/archive/2013/05/05/3060804.html)**

**前言**

网站设计的优化是一个很大的话题,有一些通用的原则,也有针对不同开发平台的一些建议。这方面的研究一直没有停止过，我在不同的场合也分享过这样的话题。

作为通用的原则，雅虎的工程师团队曾经给出过35个最佳实践。这个列表请参考**Best Practices for Speeding Up Your Web Site** <http://developer.yahoo.com/performance/rules.html>，同时，他们还发布了一个相应的测试工具Yslow <http://developer.yahoo.com/yslow/>

我强烈推荐所有的网站开发人员都应该学习这些最佳实践，并结合自己的实际项目情况进行应用。 接下来的一段时间，我将结合ASP.NET这个开发平台，针对这些原则，通过一个系列文章的形式，做些讲解和演绎，以帮助大家更好地理解这些原则，并且更好地使用他们。

**准备工作**

为了跟随我进行后续的学习，你需要准备如下的开发环境和工具

1. Google Chrome 或者firefox ，并且安装 Yslow这个扩展组件.请注意，这个组件是雅虎提供的，但目前没有针对IE的版本。
   1. <https://chrome.google.com/webstore/detail/yslow/ninejjcohidippngpapiilnmkgllmakh>
   2. <https://addons.mozilla.org/en-US/firefox/addon/yslow/>
   3. 你应该对这些浏览器的开发人员工具有所了解，你可以通过按下F12键调出这个工具。
2. Visaul Studio 2010 SP1 或更高版本，推荐使用Visual Studio 2012
   1. <http://www.microsoft.com/visualstudio/eng/downloads>
3. 你需要对ASP.NET的开发基本流程和核心技术有相当的了解，本系列文章很难对基础知识做普及。

**本文要讨论的话题**

这一篇我和大家讨论第十一条原则：[Avoid Redirects](http://developer.yahoo.com/performance/rules.html#redirects) （避免重定向）。

重定向的意思是，用户的原始请求（例如请求A）被重定向到其他的请求（例如请求B）。这是HTTP世界中本来就存在的技术和现象，它本身没有所谓的好和坏，它的存在也确实有其理由，为此[HTTP协议](http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html)中，规定了两个状态码来标识这种场景。它们分别是：

1. **301 Moved Permanently , 这个状态码标识用户所请求的资源被移动到了另外的位置，客户端收到此响应后，需要发起另外一个请求去下载所需的资源。这个状态码，日后可能会和另外一个状态码 308 （308 Permanent Redirect (approved as experimental** [**RFC**](http://en.wikipedia.org/wiki/Request_for_Comments)**)**[**[12]**](http://en.wikipedia.org/wiki/List_of_HTTP_status_codes#cite_note-PermanentRedirectDraft-12)**）有些关联（这个还没有最终确定为标准）**
2. **302 Found ，这个状态码标识用户所请求的资源被找到了，但不在原始位置，服务器会回复其他的一个地址，客户端收到此响应后，也需要发起另外一个请求去下载所需的资源。这个状态码，日后可能会和另外两个状态码有关联。**
   * [**303 See Other**](http://en.wikipedia.org/wiki/HTTP_303) **(since HTTP/1.1)**
   * **307 Temporary Redirect (since HTTP/1.1)**

目前，我们一直只要区分301和302即可。它们本质上的区别到底是什么呢？其实也不难：301表示永久重定向，302表示临时重定向。对于一般的用户而言，可能你还无法体会出来他们的区别，因为横竖都是要重定向的。但对于搜索引擎而言意义就非凡。我们都知道，搜索引擎是需要不定期对网站资源进行爬网，以便完善对应的索引结构的。当某个资源被永久重定向（301），搜索引擎会聪明地知道，在索引中应该记录就是永久重定向之后的新地址，而不是老地址，这样就可以避免用户通过搜索引擎来查询的时候，每次还需要先到老地址，再重定向到新地址。而对于临时重定向（302），则不会这么做。

**重定向会有什么影响**

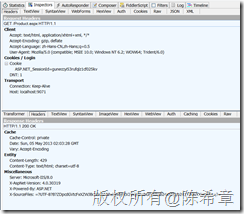
说了这么多理论知识，我们还是通过一个实例来看看具体重定向是如何发生的吧

[image](http://images.cnitblog.com/blog/9072/201305/05120916-eac7be6b0cbc434ab78d31eddf831a30.png)

我们首先请求的是default.aspx页面，出于某种原因，这个请求需要被重定向到另外一个页面，Product.aspx。所以，服务器首先为default.aspx这个请求返回302的状态码，表示说这里需要临时重定向。然后，在响应的头部（Header）中，还包含了新的地址：Product.aspx。（如下图所示）。在响应的正文（Body）中，则是空白的。

[](http://images.cnitblog.com/blog/9072/201305/05120921-efbb959553314d41b9558d44b81f22c1.png)

浏览器收到了这个回复之后，再重新发起一个请求，Product.aspx，这个请求被正常地处理了，返回200的状态码。

[](http://images.cnitblog.com/blog/9072/201305/05120923-6cca9aa172794ecfa3a036c4c8673b35.png)

事情就是这样。这样的问题在于：这样做显然增加了浏览器到服务器的往返次数，这违背了另外一个原则：[Make Fewer HTTP Requests](http://developer.yahoo.com/performance/rules.html#num_http) 。

**重定向是如何发生的**

事实上，重定向是经常发生的。有两种主要的情况下会发生重定向

1. 服务器本身的一些行为（针对某类请求）
2. 程序中明确地做了重定向

第一种情况很有意思，不同的服务器可能在处理的时候表现也是不一样的。例如我们来看下面这个请求

[http://cnblogs.com](http://cnblogs.com/)

通过监控我们发现，首先会有一个301的重定向，然后才是真正的请求（返回200），此时地址是 [http://www.cnblogs.com](http://www.cnblogs.com/)

[image](http://images.cnitblog.com/blog/9072/201305/05120923-1d5fa228e89147ccacac1caa0801d474.png)

这是为什么呢？如果有网站开发经验的朋友一定知道，域名都是需要解析的，实际上向用户提供的服务，是互联网上面的某个网络主机。做得比较好的网站，同时会考虑用户希望访问网站的不同的方式，上例中，用户既可以访问cnblogs.com 也可以访问[www.cnblogs.com](http://www.cnblogs.com/)，区别在于前者多一次重定向请求。

如果你就此认为是理所应当的，那么你就错了。我们再来看一下“著名”的12306.cn 吧

[](http://images.cnitblog.com/blog/9072/201305/05120927-77c59497e2bc4faa96b7b6c4bdb23e6c.png)

你如果这样访问的时候，直接就失败了。你必须完整地输入[www.12306.cn](http://www.12306.cn/) 才能访问。试问，这是多么简单、低级的问题。

其实要做到这个很简单，牺牲了一点点性能，对于用户来说，提高了用户体验。这里留一个疑问：**有人可以解释一下，到底如何实现这样的效果（cnblogs.com =>** [**www.cnblogs.com**](http://www.cnblogs.com/) **）吗？**

这是一种典型的重定向，这个无需任何代码就能实现。实际上是属于服务器的功能。当然，我们完全应该尽可能地使用[www.cnblogs.com](http://www.cnblogs.com/) 这种方式（例如给人们邮件中的链接，尽量是使用这个路径）。但提供了另外一个方式，是很好的设计。

还有一种典型的重定向，我们来看下面这个例子，请在浏览器中输入google.com

我们最终看到的页面是 <http://www.google.com.hk/>

[](http://images.cnitblog.com/blog/9072/201305/05120931-8e30282d0d794c808cab18719c40874f.png)

你会发现，会有几次重定向

[](http://images.cnitblog.com/blog/9072/201305/05120932-9211dcaaa337458eb99846a75baed872.png)

第一次重定向是301，从google.com 重定向到[www.google.com](http://www.google.com/)

第二次重定向是302，从[www.google.com](http://www.google.com/) 重定向到 <http://www.google.com.hk/url?sa=p&hl=zh-CN&pref=hkredirect&pval=yes&q=http://www.google.com.hk/&ust=1367722843791916&usg=AFQjCNEzkTX2uE5Jlo3NkA1vSHdwoCnnZQ>

第三次重定向是302，从<http://www.google.com.hk/url?sa=p&hl=zh-CN&pref=hkredirect&pval=yes&q=http://www.google.com.hk/&ust=1367722843791916&usg=AFQjCNEzkTX2uE5Jlo3NkA1vSHdwoCnnZQ> 重定向到 [http://www.google.com.hk](http://www.google.com.hk/)

同样的做法，我们从bing.com中也能看到。我们输入bing.com ，但实际看到的页面是cn.bing.com

[](http://images.cnitblog.com/blog/9072/201305/05121007-6e71b608e9fa422fa53facefbe19f0f5.png)

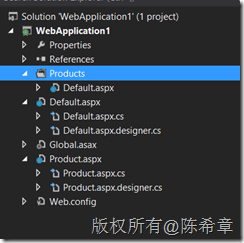
实际上，这里也发生了一次重定向

[image](http://images.cnitblog.com/blog/9072/201305/05121014-2706bf02d10f46ee8411e02f8a4bc730.png)

为什么Google和Microsoft都会这样设计搜索引擎的主界面呢？原因在于他们想给用户提供更加有个性化的本地服务，所以针对不同国家和地区的用户，实际上有独立的主机来进行处理。

Microsoft的做法有点不一样，他们使用了统一的域名bing.com，只是为不同的国家和地区准备了不同的主机，例如cn.bing.com, hk.bing.com 。

还有一种容易比我们忽视的重定向。请参考下面的实例

[](http://images.cnitblog.com/blog/9072/201305/05121014-2610d6d28c194668843e9980786ce564.png)

在这个示例网站中，有一个文件夹，叫做Products，里面会有很多页面，例如至少会有一个Default.aspx。我们都知道，通常Default.aspx是所谓的默认页面，也就是说，要访问这个页面的话，并不需要输入Default.aspx这个部分，而是直接通过访问文件夹名称即可，例如下面这样

<http://localhost:9071/Products>

和你想象的一样，这样的简写路径会返回Products目录中的Default.aspx页面的内容（这很不错，对吧）但是，通过监控我们发现，这样一个请求都会发生了一次重定向

[image](http://images.cnitblog.com/blog/9072/201305/05121018-26373889b4b54745b9261412b58518f6.png)

有意思的是，它会将地址重定向到 <http://localhost:9071/Products/> (只是比原始地址多了一个路径斜线），很神奇吗？但这是真的，凡是访问地址中，没有带文件名后缀的（例如aspx,asp等等），服务器都会尝试解析为一个文件夹，自动加上一个路径斜线，然后再查找内部的默认页面。

【备注】这个行为，在ASP.NET MVC中是不会存在的，因为ASP.NET MVC的请求处理是被路由处理的。

**ASP.NET中的重定向**

如果上述网站是用ASP.NET开发的（cnblogs.com 显然是的，而Google.com则显然不是的，bing.com 会是的吗？），那么会怎么样来实现上述所提到的重定向呢？

ASP.NET 4.0 有如下几种方式来做重定向

1. Response.Redirect  <http://msdn.microsoft.com/EN-US/library/08za4s98(v=VS.110,d=hv.2).aspx>
   * 这个会进行临时重定向，也就是返回302
2. Response.RedirectPermanent <http://msdn.microsoft.com/EN-US/library/dd322058(v=VS.110,d=hv.2).aspx> （这是新增的方法）
   * 这个会进行永久性重定向，也就是返回301
3. Server.Transfer <http://msdn.microsoft.com/EN-US/library/y0w8173d(v=VS.110,d=hv.2).aspx>
   * 这个不会产生301，也不会产生302，实际上，它直接返回200，浏览器根本不会知道发生了重定向，不会有多出来的一个请求
     + 听起来不错，不是吗？那么，**请问，这样做有什么附加的问题吗**？

**ASP.NET中重定向附加的问题**

以上的三个方法，我相信大家至少对其中一两个很熟悉，但是很多人不清楚他们的代价。

实际上,不管是用Redirect还是Transfer方法，他们内部都会调用Response.End方法（这个很好理解，因为需要重定向了，所以当前的这个请求应该就不需要再提供响应了），但这个方法会导致与该请求有关的处理线程强制被中断掉，具体来说，这将引发一个异常（**ThreadAbortException**），通过Try…catch就能捕捉到。

我们都知道，异常处理在.NET中是由CLR来做的，异常处理的代价是较高的，所以如果过于频繁地抛出异常，会给性能带来显著的影响。

关于这个问题，以及如何改善，可以参考微软官方的文档

<http://support.microsoft.com/kb/312629/en-us>

我摘录解决方案的部分如下

To work around this problem, use one of the following methods:

* For **Response.End**, call the **HttpContext.Current.ApplicationInstance.CompleteRequest** method instead of **Response.End** to bypass the code execution to the **Application\_EndRequest** event.
* For **Response.Redirect**, use an overload, **Response.Redirect(String url, bool endResponse)** that passes **false** for the**endResponse** parameter to suppress the internal call to **Response.End**. For example:
* Response.Redirect ("nextpage.aspx", false);

If you use this workaround, the code that follows **Response.Redirect** is executed.

* For **Server.Transfer**, use the **Server.Execute** method instead.

**如何尽可能避免重定向**

从上面的实例和分析来看，重定向是无法完全避免的，适当地使用重定向能为网站提供更好的功能。（例如本地化，用户体验等方面）。

但是过多地进行重定向也肯定会给网站性能带来显著的影响。那么，有哪些方法可以作为我们改善这一点的参考呢

1. 在定义链接地址的href属性的时候，尽量使用最完整的、直接的地址。例如
   * 使用[www.cnblogs.com](http://www.cnblogs.com/) 而不是cnblogs.com
   * 使用cn.bing.com 而不是bing.com
   * 使用[www.google.com.hk](http://www.google.com.hk/) 而不是google.com
   * 使用[www.mysite.com/products/](http://www.mysite.com/products/) 而不是 [www.mysite.com/products](http://www.mysite.com/products)
2. 在使用Response.Redirect的时候，设置第二个参数为false
   * 考虑是否可用Server.Execute代替
   * 考虑Respone.RedirectPermanent
3. 如果涉及到从测试环境到生产环境的迁移，建议通过DNS中的[CNAME](http://en.wikipedia.org/wiki/CNAME_record)的机制来定义别名，而不是强制地重定向来实现