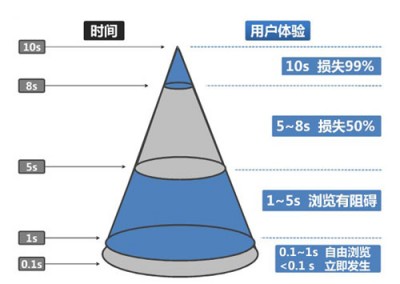
Web前端优化最佳实践及工具集锦

[Web优化](http://www.csdn.net/tag/web%E4%BC%98%E5%8C%96/news)[Google](http://www.csdn.net/tag/google/news)[雅虎](http://www.csdn.net/tag/%E9%9B%85%E8%99%8E/news)[PageSpeed](http://www.csdn.net/tag/pagespeed/news)[YSlow](http://www.csdn.net/tag/yslow/news)

**摘要：**前端的性能对于Web应用的用户体验来说非常重要。不要以为你的Web应用的性能已经足够好了，其实还会有很多可以提升的地方。本文将介绍Google和雅虎关于前端优化的最佳实践以及工具，你可以逐一检验你的Web应用。

前端的性能对于一个Web应用来说非常重要，如果一个Web应用的页面加载速度非常快、对于用户的操作可以及时响应，那么产品的用户体验将会极大地提升。下图显示了页面加载速度对于用户体验的影响。

[](http://cms.csdnimg.cn/article/201309/23/52402c4950288.jpg)

你的Web页面的速度是否已经足够快了？其实可能还有很多可以提升的地方。Google和雅虎也提出了一些Web应用的前端优化建议，并发布了一些工具，你可以逐一检验你的Web应用，以便达到更高的性能。

这些优化不仅仅可以给用户提供更好的体验，从开发者角度来说，进行优化还可以减少页面的请求数、降低请求所占的带宽、减少资源的浪费。

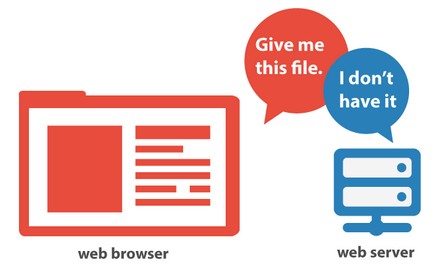
下面来看看Google和雅虎提供的Web页面优化最佳实践。

**一、Google的Web优化最佳实践**

**1.**[**避免坏请求**](http://www.feedthebot.com/pagespeed/avoid-bad-requests.html)

有时页面中的HTML或CSS会向服务器请求一个不存在的资源，比如图片或HTML文件，这会造成浏览器与服务器之间过多的往返请求，类似于：

* 浏览器：“我需要这个图像。”
* 服务器：“我没有这个图像。”
* 浏览器：“你确定吗？这个文档说你有。”
* 服务器：“真的没有。”

[](http://cms.csdnimg.cn/article/201309/23/5240283e1adbb.jpg)

如此一来，会降低页面的加载速度。因此，检查页面中的坏链接非常有必要，你可以通过 [Google的PageSpeed工具](https://developers.google.com/speed/pagespeed/insights) 来检测，找到问题后，补充相应的资源文件或者修改资源的链接地址即可。

**2.**[**避免CSS @import**](http://www.feedthebot.com/pagespeed/avoid-css-import.html)

使用 @import方法引用CSS文件可以能会带来一些影响页面加载速度的问题，比如导致文件按顺序加载（一个加载完后才会加载另一个），而无法并行加载。

你可以使用 [CSS delivery工具](http://www.feedthebot.com/tools/css-delivery/)来检测页面代码中是否存在@import方法。比如，如果检测结果中存在

**[css]** [view plaincopy](http://www.csdn.net/article/2013-09-23/2817020-web-performance-optimization)

1. @import url("style.css")

则建议你使用下面的代码来替代。

**[html]** [view plaincopy](http://www.csdn.net/article/2013-09-23/2817020-web-performance-optimization)

1. **<link** rel="style.css" href="style.css" type="text/css"**>**

**3.**[**避免使用document.write**](http://www.feedthebot.com/pagespeed/avoid-document-write.html)

在JavaScript中，可以使用 document.write在网页上显示内容或调用外部资源，而通过此方法，浏览器必须采取一些多余的步骤——下载资源、读取资源、运行JavaScript来了解需要做什么，调用其他资源时需要重新再执行一次这个过程。由于浏览器之前不知道要显示什么，所以会降低页面加载的速度。

要知道，任何能够被document.write调用的资源，都可以通过HTML来调用，这样速度会更快。检查你的页面代码，如果存在类似于下面的代码：

**[js]** [view plaincopy](http://www.csdn.net/article/2013-09-23/2817020-web-performance-optimization)

1. document.write('<script src="another.js"></script>');

建议修改为：

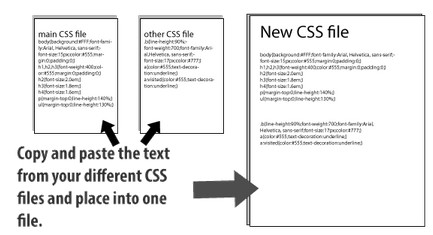
**[html]** [view plaincopy](http://www.csdn.net/article/2013-09-23/2817020-web-performance-optimization)

1. **<script** src="another.js"**></script>**

**4.**[**合并多个外部CSS文件**](http://www.feedthebot.com/pagespeed/combine-external-css.html)

在网站中每使用一个CSS文件，都会让你的页面加载速度慢一点点。如果你有一个以上的CSS文件，你应该将它们合并为一个文件。

你可以通过  [CSS delivery工具](http://www.feedthebot.com/tools/css-delivery/)来检测页面代码中的CSS文件，然后通过复制粘贴的方式将它们合并为一个。合并后记得修改页面中的引用代码，并删除旧的引用代码。

[](http://cms.csdnimg.cn/article/201309/23/5240285d0242f.jpg)

**5.**[**合并多个外部JavaScript文件**](http://www.feedthebot.com/pagespeed/combine-external-javascript.html)

大部分情况下，网站往往会包含若干个 JavaScript文件，但并不需要将这些文件都独立出来，其中有些是可以合并为一个文件的。

你可以通过 [resource check工具](http://www.feedthebot.com/tools/requests/)来检测页面中所引用的JavaScript文件数，然后可以通过复制粘贴的方式将多个文件合并为一个。

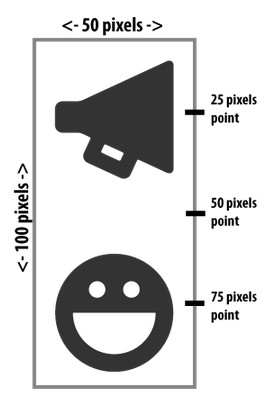
**6.**[**通过CSS sprites来整合图像**](http://www.feedthebot.com/pagespeed/combine-images-css-sprites.html)

如果页面中有6个小图像，那么浏览器在显示时会分别下载。你可以通过CSS sprites将这些图像合并成1个，可以减少页面加载所需的时间。

CSS sprites需要有两个步骤：整合图像、定位图像。比如你可以通过下面的代码来分别定位下面图像中的上下两部分。

**[css]** [view plaincopy](http://www.csdn.net/article/2013-09-23/2817020-web-performance-optimization)

1. .megaphone {**width**:50px; **height**:50px; **background**:url(images/sprite.png) 0 0px;}
2. .smile {**width**:50px; **height**:50px; **background**:url(images/sprite.png) 0 -50px;}

**[](http://cms.csdnimg.cn/article/201309/23/5240287e14d30.jpg)**

**7.**[**延迟JavaScript的加载**](http://www.feedthebot.com/pagespeed/defer-loading-javascript.html)

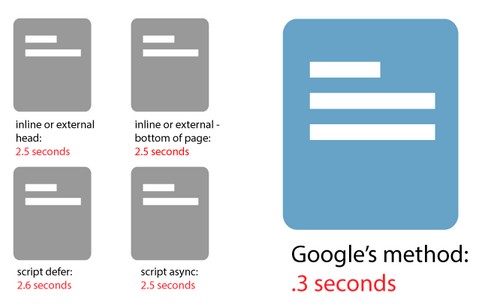
浏览器在执行JavaScript代码时会停止处理页面，当页面中有很多JavaScript文件或代码要加载时，将导致严重的延迟。尽管可以使用defer、异步或将JavaScript代码放到页面底部来延迟JavaScript的加载，但这些都不是一个好的解决方案。

下面是Google的建议。

**[js]** [view plaincopy](http://www.csdn.net/article/2013-09-23/2817020-web-performance-optimization)

1. <script type="text/javascript">
2. **function** downloadJSAtOnload() {
3. **var** element = document.createElement("script");
4. element.src = "defer.js";
5. document.body.appendChild(element);
6. }
7. **if** (window.addEventListener)
8. window.addEventListener("load", downloadJSAtOnload, **false**);
9. **else** **if** (window.attachEvent)
10. window.attachEvent("onload", downloadJSAtOnload);
11. **else** window.onload = downloadJSAtOnload;
12. </script>

这段代码的意思是等待页面加载完成后，然后再加载外部的“defer.js”文件。下面是测试结果。

[](http://cms.csdnimg.cn/article/201309/23/524028cf68923.jpg)

**8.**[**启用压缩/ GZIP**](http://www.feedthebot.com/pagespeed/enable-compression.html)

使用gzip对HTML和CSS文件进行压缩，通常可以节省大约50％到70％的大小，这样加载页面只需要更少的带宽和更少的时间。

你可以通过这个 [Gzip压缩工具](http://www.feedthebot.com/tools/gzip/" \t "_blank)来检测页面是否已经经过Gzip压缩。

**9.**[**启用Keep-Alive**](http://www.feedthebot.com/pagespeed/keep-alive.html)

HTTP协议采用“请求-应答”模式，当使用普通模式（非KeepAlive模式）时，每个请求/应答客户和服务器都要新建一个连接，完成之后立即断开连接（HTTP协议为无连接的协议）；当使用 Keep-Alive模式（又称持久连接、连接重用）时，Keep-Alive功能使客户端到服务器端的连接持续有效，当出现对服务器的后继请求时，Keep-Alive功能避免了建立或者重新建立连接。

在HTTP 1.0中Keep-Alive默认是关闭的，需要在HTTP头中加入“Connection: Keep-Alive”，才能启用Keep-Alive；在 HTTP1.1中Keep-Alive默认启用，加入“Connection: close”可关闭。目前大部分浏览器都是用HTTP 1.1协议，也就是说默认都会发起Keep-Alive的连接请求了，所以是否能完成一个完整的Keep- Alive连接就看Web服务器的设置情况。

**10.**[**将小的CSS和JavaScript代码内嵌到HTML中**](http://www.feedthebot.com/pagespeed/inline-small-css.html)

如果你的CSS代码比较小，可以将这部分代码放到HTML文件中，而不是一个外部CSS文件，这样可以减少页面加载所需的文件数，从而加快页面的加载。同样，也可以将小的 JavaScript脚本代码内嵌到HTML文件中。

**[html]** [view plaincopy](http://www.csdn.net/article/2013-09-23/2817020-web-performance-optimization)

1. **<style** type="text/css"**>**
2. <!--CSS代码-->
3. **</style>**
5. **<script** type="text/javascript"**>**
6. <!--JavaScript代码-->
7. **</script>**

**11.**[**利用浏览器缓存**](http://www.feedthebot.com/pagespeed/leverage-browser-caching.html)

在显示页面时，浏览器需要加载logo、CSS文件和其他一些资源。浏览器缓存所做的工作就是“记住”已经加载的资源，让页面的加载速度更快。

**12.**[**压缩CSS代码**](http://www.feedthebot.com/pagespeed/minify-css.html)

不管你在页面中如何使用CSS，CSS文件都是越小越好，这会帮助你提升网页的加载速度。你可以通过 [Minify CSS工具](http://www.feedthebot.com/tools/css/)来压缩你的CSS代码。

压缩前：

**[css]** [view plaincopy](http://www.csdn.net/article/2013-09-23/2817020-web-performance-optimization)

1. body
2. {
3. **background-color**:#d0e4fe;
4. }
5. h1
6. {
7. **color**:orange;
8. **text-align**:center;
9. }

压缩后：

**[css]** [view plaincopy](http://www.csdn.net/article/2013-09-23/2817020-web-performance-optimization)

1. body {**background-color**:#d0e4fe;}
2. h1 {**color**:orange;**text-align**:center;}

**13.**[**尽量减少DNS查询次数**](http://www.feedthebot.com/pagespeed/dns-lookups.html)

当浏览器与Web服务器建立连接时，它需要进行DNS解析，将域名解析为IP地址。然而，一旦客户端需要执行DNS lookup时，等待时间将会取决于域名服务器的有效响应的速度。

虽然所有的ISP的DNS服务器都能缓存域名和IP地址映射表，但如果缓存的DNS记录过期了而需要更新，则可能需要通过遍历多个DNS节点，有时候需要通过全球范围内来找到可信任的域名服务器。一旦域名服务器工作繁忙，请求解析时就需要排队，则进一步延迟等待时间。

因此，减少DNS的查询次数非常重要，页面加载时就尽量避免额外耗时。为了减少DNS查询次数，最好的解决方法就是在页面中减少不同的域名请求的机会。

你可以通过 [request checker工具](http://www.feedthebot.com/tools/requests/)来检测页面中存在多少请求，然后进行优化。

**14.**[**尽量减少重定向**](http://www.feedthebot.com/pagespeed/minimize-redirects.html)

有时为了特定需求，需要在网页中使用重定向。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）。

但是这会造成网站性能和速度下降，因为浏览器访问网址是一连串的过程，如果访问到一半而跳到新地址，就会重复发起一连串的过程，这将浪费很多的时间。所以我们要尽量避免重定向，Google建议：

* 不要链接到一个包含重定向的页面
* 不要请求包含重定向的资源

**15.**[**优化样式表和脚本的顺序**](http://www.feedthebot.com/pagespeed/style-script-order.html)

Style标签和样式表调用代码应该放置在JavaScript代码的前面，这样可以使页面的加载速度加快。

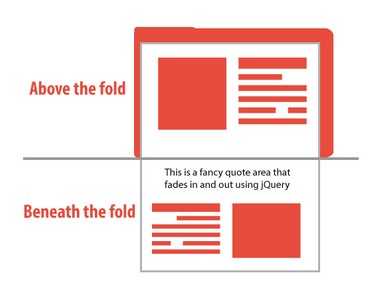
**[html]** [view plaincopy](http://www.csdn.net/article/2013-09-23/2817020-web-performance-optimization)

1. **<head>**
2. **<meta** name=description content="description"**/>**
3. **<title>**title**</title>**
4. **<style>**
5. page specific css code goes here
6. **</style>**
7. **<script** type="text/javascript"**>**
8. javascript code goes here
9. **</script>**
10. **</head>**

**16.**[**避免JavaScripts阻塞渲染**](http://www.feedthebot.com/pagespeed/render-blocking.html)

浏览器在遇到一个引入外部JS文件的<script>标签时，会停下所有工作来下载并解析执行它，在这个过程中，页面渲染和用户交互完全被阻塞了。这时页面加载就会停止。

谷歌 [建议](https://developers.google.com/speed/docs/insights/BlockingJS)删除干扰页面中第一屏内容加载的JavaScript，第一屏是指用户在屏幕中最初看到的页面，无论是桌面浏览器、手机，还是平板电脑。

[](http://cms.csdnimg.cn/article/201309/23/524028fe7e44d.jpg)

**17.  缩小原始图像**

如果无需在页面中显示较大的图像，那么就建议将图像的实际大小缩小为显示的大小，这样可以减少下载图像所需的时间。

**18.  指定图像尺寸**

当浏览器加载页面的HTML代码时，有时候需要在图片下载完成前就对页面布局进行定位。如果HTML里的图片没有指定尺寸（宽和高），或者代码描述的尺寸与实际图片的尺寸不符时，浏览器则要在图片下载完成后再“回溯”该图片并重新显示，这将消耗额外的时间）。

所以，最好为页面中的每一张图片都指定尺寸，不管是在HTML里的<img>标签中，还是在CSS中。

**更多信息：** <https://developers.google.com/speed/docs/insights/rules>

**二、雅虎的Web优化最佳实践**

**1.  内容优化**

* [尽量减少HTTP请求](http://developer.yahoo.com/performance/rules.html#num_http)：常见方法包括合并多个CSS文件和JavaScript文件，利用CSS Sprites整合图像，Image map（图像中不同的区域设置不同的链接），内联图象（使用  [data: URL scheme](http://tools.ietf.org/html/rfc2397) 在实际的页面嵌入图像数据）等。
* [减少DNS查找](http://developer.yahoo.com/performance/rules.html#dns_lookups)
* [避免重定向](http://developer.yahoo.com/performance/rules.html#redirects)
* [使Ajax可缓存](http://developer.yahoo.com/performance/rules.html#cacheajax)
* [延迟加载组件](http://developer.yahoo.com/performance/rules.html#postload)：考虑哪些内容是页面呈现时所必需首先加载的、哪些内容和结构可以稍后再加载，根据这个优先级进行设定。
* [预加载组件](http://developer.yahoo.com/performance/rules.html#preload)：预加载是在浏览器空闲时请求将来可能会用到的页面内容（如图像、样式表和脚本）。当用户要访问下一个页面时，页面中的内容大部分已经加载到缓存中了，因此可以大大改善访问速度。
* [减少DOM元素数量](http://developer.yahoo.com/performance/rules.html#min_dom)：页面中存在大量DOM 元素，会导致JavaScript遍历DOM的效率变慢。
* [根据域名划分页面内容](http://developer.yahoo.com/performance/rules.html#split)：把页面内容划分成若干部分可以使你最大限度地实现平行下载。但要确保你使用的域名数量在2个到4个之间（否则与第2条冲突）。
* [最小化iframe的数量](http://developer.yahoo.com/performance/rules.html#iframes)：iframes 提供了一个简单的方式把一个网站的内容嵌入到另一个网站中。但其创建速度比其他包括JavaScript和CSS的DOM元素的创建慢了1-2个数量级。
* [避免404](http://developer.yahoo.com/performance/rules.html#no404)：HTTP请求时间消耗是很大的，因此使用HTTP请求来获得一个没有用处的响应（例如404没有找到页面）是完全没有必要的，它只会降低用户体验而不会有一点好处。

**2. 服务器优化**

* [使用内容分发网络（CDN）](http://developer.yahoo.com/performance/rules.html#cdn)：把你的网站内容分散到多个、处于不同地域位置的服务器上可以加快下载速度。
* [添加Expires或Cache-Control信息头](http://developer.yahoo.com/performance/rules.html#expires)：对于静态内容，可设置文件头过期时间Expires的值为“Never expire（永不过期）”；对于动态内容，可使用恰当的Cache-Control文件头来帮助浏览器进行有条件的请求。
* [Gzip压缩](http://developer.yahoo.com/performance/rules.html#gzip)
* [设置ETag](http://developer.yahoo.com/performance/rules.html#etags)：ETags（Entity tags，实体标签）是web服务器和浏览器用于判断浏览器缓存中的内容和服务器中的原始内容是否匹配的一种机制。
* [提前刷新缓冲区](http://developer.yahoo.com/performance/rules.html#flush)：当用户请求一个页面时，服务器会花费200到500毫秒用于后台组织HTML文件。在这期间，浏览器会一直空闲等待数据返回。在PHP中，可以使用flush()方法，它允许你把已经编译的好的部分HTML响应文件先发送给浏览器，这时浏览器就会可以下载文件中的内容（脚本等）而后台同时处理剩余的HTML页面。
* [对Ajax请求使用GET方法](http://developer.yahoo.com/performance/rules.html#ajax_get)：当使用XMLHttpRequest时，浏览器中的POST方法会首先发送文件头，然后才发送数据。因此使用GET最为恰当。
* [避免空的图像src](http://developer.yahoo.com/performance/rules.html#emptysrc)

**3. Cookie优化**

* [减小cookie大小](http://developer.yahoo.com/performance/rules.html#cookie_size)：去除不必要的coockie，并使coockie体积尽量小以减少对用户响应的影响
* [针对Web组件使用域名无关的Cookie](http://developer.yahoo.com/performance/rules.html#cookie_free)：对静态组件的Cookie读取是一种浪费，使用另一个无Cookie的域名来存放静态组件是一个好方法，或者也可以在Cookie中只存放带www的域名。

**4. CSS优化**

* [将CSS代码放在HTML页面的顶部](http://developer.yahoo.com/performance/rules.html#css_top)
* [避免使用CSS表达式](http://developer.yahoo.com/performance/rules.html#css_expressions)：CSS表达式在执行时候的运算量非常大，会对页面性能产生大的影响
* [使用<link>来代替@import](http://developer.yahoo.com/performance/rules.html#csslink)
* [避免使用Filters](http://developer.yahoo.com/performance/rules.html#no_filters)：IE独有属性AlphaImageLoader用于修正IE 7以下版本中PNG图片的半透明效果，但它的问题在于浏览器加载图片时它会终止内容的呈现并且冻结浏览器。

**5. JavaScript优化**

* [将JavaScript脚本放在页面的底部](http://developer.yahoo.com/performance/rules.html#js_bottom)
* [将JavaScript和CSS作为外部文件来引用](http://developer.yahoo.com/performance/rules.html#external)：在实际应用中使用外部文件可以提高页面速度，因为JavaScript和CSS文件都能在浏览器中产生缓存。
* [缩小JavaScript和CSS](http://developer.yahoo.com/performance/rules.html#minify)
* [删除重复的脚本](http://developer.yahoo.com/performance/rules.html#js_dupes)
* [最小化DOM的访问](http://developer.yahoo.com/performance/rules.html#dom_access)：使用JavaScript访问DOM元素比较慢
* [开发智能的事件处理程序](http://developer.yahoo.com/performance/rules.html#events)

**6. 图像优化**

* [优化图片大小](http://developer.yahoo.com/performance/rules.html#opt_images)
* [通过CSS Sprites优化图片](http://developer.yahoo.com/performance/rules.html#opt_sprites)
* [不要在HTML中使用缩放图片](http://developer.yahoo.com/performance/rules.html#no_scale)
* [favicon.ico要小而且可缓存](http://developer.yahoo.com/performance/rules.html#favicon)

**7. 针对移动优化**

* [保持组件大小在25KB以下](http://developer.yahoo.com/performance/rules.html#under25)：主要是因为iPhone不能缓存大于25K的文件（注意这里指的是解压缩后的大小）。
* [将组件打包成为一个复合文档](http://developer.yahoo.com/performance/rules.html#multipart)：把页面内容打包成复合文本就如同带有多附件的Email，它能够使你在一个HTTP请求中获取多个组件。

**更多信息：**[**http://developer.yahoo.com/performance/rules.html**](http://developer.yahoo.com/performance/rules.html)**（**[**中文翻译**](http://dudo.org/archives/2008051211216.html)**）**

**三、一些工具**

**1.**[**Google PageSpeed**](https://developers.google.com/speed/pagespeed/)

Google提供了 PageSpeed工具，这是一个浏览器插件，可以很好地应用上文中Google所提到的Web优化实践——帮助你轻松对网站的性能瓶颈进行分析，并为你提供优化建议。

* [在线分析你的网站](https://developers.google.com/speed/pagespeed/insights)
* [安装浏览器插件](https://developers.google.com/speed/pagespeed/insights_extensions)（ [Chrome](https://chrome.google.com/webstore/detail/gplegfbjlmmehdoakndmohflojccocli)、 [Firefox](https://dl-ssl.google.com/page-speed/current/page-speed.xpi)）
* 通过 [Insights API](https://developers.google.com/speed/docs/insights/v1/getting_started)在应用中嵌入PageSpeed功能

**2.**[**雅虎 YSlow**](http://developer.yahoo.com/yslow/)

YSlow是雅虎推出的一款浏览器插件，可以帮助你对网站的页面进行分析，并为你提供一些优化建议，以提高网站的性能。

* [Firefox插件](https://addons.mozilla.org/en-US/firefox/addon/5369)
* [Chrome插件](https://chrome.google.com/webstore/detail/ninejjcohidippngpapiilnmkgllmakh)
* [YSlow for Mobile/Bookmarklet](http://yslow.org/mobile/)
* [源码](https://github.com/marcelduran/yslow)

**3. 其他分析优化工具**

* [蜘蛛模拟器](http://www.feedthebot.com/tools/spider/)：这个工具可以分析你的页面，并提供一些优化建议。
* [图像SEO工具](http://www.feedthebot.com/tools/alt/)：这个工具可以检查图片的alt标签，并提供一些优化建议。
* [请求检查器](http://www.feedthebot.com/tools/requests/)：找出页面中需要加载哪些资源和服务。
* [链接检查器](http://www.feedthebot.com/tools/linkcount/)：检查页面中内部、外部和无效链接。
* [HTTP头检查](http://www.feedthebot.com/tools/headers/)：显示网页或资源的HTTP响应头。
* [社交检查器](http://www.feedthebot.com/tools/social/)：检查页面中的社交组件，比如Google+、Facebook、Twitter、Linkedin和Pinterest。
* [If modified检查器](http://www.feedthebot.com/tools/if-modified/)：检查页面是否接受 If-Modified-Since HTTP头。
* [Gzip检查器](http://www.feedthebot.com/tools/gzip/)：检查页面是否经过了Gzip压缩。
* [CSS delivery工具](http://www.feedthebot.com/tools/css-delivery/)：检查页面中所使用的CSS文件。
* [面包屑工具](http://www.feedthebot.com/tools/richsnippets/breadcrumb/)：可根据你输入的信息提供面包屑导航的代码。
* [CSS压缩工具](http://www.feedthebot.com/tools/css/)：用于压缩CSS代码。