前端优化：雅虎35条

时间 2015-04-03 18:18:51  [黯羽轻扬](http://www.tuicool.com/sites/EzIbEz)

原文  [http://ayqy.net/blog/前端优化：雅虎35条/](http://ayqy.net/blog/%E5%89%8D%E7%AB%AF%E4%BC%98%E5%8C%96%EF%BC%9A%E9%9B%85%E8%99%8E35%E6%9D%A1/?utm_source=tuicool&utm_medium=referral)

主题 [HTML](http://www.tuicool.com/topics/11060000)[JavaScript](http://www.tuicool.com/topics/11060004)

**写在前面**

[***英文原版链接***](https://developer.yahoo.com/performance/rules.html) ，若是觉得本文哪里不好还请指出，以便及时修改

目录（分7类，共35条）：

1. [内容]尽量减少HTTP请求数
2. [服务器]使用CDN（Content Delivery Network）
3. [服务器]添上Expires或者Cache-Control HTTP头
4. [服务器]Gzip组件
5. [css]把样式表放在顶部
6. [js]把脚本放在底部
7. [css]避免使用CSS表达式
8. [js, css]把JavaScript和CSS放到外面
9. [内容]减少DNS查找
10. [js, css]压缩JavaScript和CSS
11. [内容]避免重定向
12. [js]去除重复脚本
13. [服务器]配置ETags
14. [内容]让Ajax可缓存
15. [服务器]尽早清空缓冲区
16. [服务器]对Ajax用GET请求
17. [内容]延迟加载组件
18. [内容]预加载组件
19. [内容]减少DOM元素的数量
20. [内容]跨域分离组件
21. [内容]尽量少用iframe
22. [内容]杜绝404
23. [cookie]给Cookie减肥
24. [cookie]把组件放在不含cookie的域下
25. [js]尽量减少DOM访问
26. [js]用智能的事件处理器
27. [***[css]选择 舍弃@import***](http://www.tuicool.com/articles/J3uyaa)
28. [css]避免使用滤镜
29. [图片]优化图片
30. [图片]优化CSS Sprite
31. [图片]不要用HTML缩放图片
32. [图片]用小的可缓存的favicon.ico（P.S. 收藏夹图标）
33. [移动端]保证所有组件都小于25K
34. [移动端]把组件打包到一个复合文档里
35. [服务器]避免图片src属性为空

**译文：给网站提速的最佳实践**

我们已经发现了不少给站点提速的最佳实践，分7类共35条。

**1.尽量减少HTTP请求数**

分类: 内容

80%的终端用户响应时间都花在了前端上，其中大部分时间都在下载页面上的各种组件：图片，样式表，脚本，Flash等等。减少组件数必然能够减少页面提交的HTTP请求数。这是让页面更快的关键。

减少页面组件数的一种方式是简化页面设计。但有没有一种方法可以在构建复杂的页面同时加快响应时间呢？嗯，确实有鱼和熊掌兼得的办法。

合并文件是通过把所有脚本放在一个文件中的方式来减少请求数的，当然，也可以合并所有的CSS。如果各个页面的脚本和样式不一样的话，合并文件就是一项比较麻烦的工作了，但把这个作为站点发布过程的一部分确实可以提高响应时间。

[***CSS Sprites***](http://alistapart.com/articles/sprites)是减少图片请求数量的首选方式。把背景图片都整合到一张图片中，然后用CSS的 background-image 和 background-position 属性来定位要显示的部分。

[***图像映射***](http://www.w3.org/TR/html401/struct/objects.html#h-13.6)可以把多张图片合并成单张图片，总大小是一样的，但减少了请求数并加速了页面加载。图片地图只有在图像在页面中连续的时候才有用，比如导航条。给image map设置坐标的过程既无聊又容易出错，用image map来做导航也不容易，所以不推荐用这种方式。

行内图片（Base64编码）用 [***data: URL模式***](http://tools.ietf.org/html/rfc2397)来把图片嵌入页面。这样会增加HTML文件的大小，把行内图片放在（缓存的）样式表中是个好办法，而且成功避免了页面变“重”。但目前主流浏览器并不能很好地支持行内图片。

减少页面的HTTP请求数是个起点，这是提升站点首次访问速度的重要指导原则。就像Tenni Theurer的博客 [***Browser Cache Usage – Exposed!***](http://yuiblog.com/blog/2007/01/04/performance-research-part-2/) 里写到的,40%到60%的访客在访问你的站点时，缓存都是空的。所以，加快页面首次访问速度对提高用户体验是极其重要的。

**2.使用CDN（Content Delivery Network）**

分类: 服务器

用户与服务器的物理距离对响应时间也有影响。把内容部署在多个地理位置分散的服务器上能让用户更快地载入页面。但具体要怎么做呢？

实现内容在地理位置上分散的第一步是：不要尝试去重新设计你的web应用程序来适应分布式结构。这取决于应用程序，改变结构可能包括一些让人望而生畏的任务，比如同步会话状态和跨服务器复制数据库事务（翻译可能不准确）。缩短用户和内容之间距离的提议可能被推迟，或者根本不可能通过，就是因为这个难题。

记住终端用户80%到90%的响应时间都花在下载页面组件上了：图片，样式，脚本，Flash等等，这是 *业绩黄金法则* 。最好先分散静态内容，而不是一开始就重新设计应用程序结构。这不仅能够大大减少响应时间，还更容易表现出CDN的功劳。

内容分发网络（CDN）是一组分散在不同地理位置的web服务器，用来给用户更高效地发送内容。典型地，选择用来发送内容的服务器是基于网络距离的衡量标准的。例如：选跳数（hop）最少的或者响应时间最快的服务器。

一些互联网公司巨头拥有他们自己的CDN，但用一个CDN服务提供者是比较划算的，比如 [***Akamai Technologies***](http://www.akamai.com/) ， [***EdgeCast***](http://www.edgecast.com/) ，或者 [***level3***](http://www.level3.com/index.cfm?pageID=36) 。对刚刚起步的公司和个人网站来说，CDN服务的成本是很高的，但如果你的用户群却越来越大，越来越全球化，那么用CDN来换取更快的响应时间还是很有必要的。在Yahoo!，把静态内容从应用程序的web服务器搬到CDN(包括上面提到的3rd party和Yahoo自己的 [***CDN***](https://cwiki.apache.org/TS/traffic-server.html) )能够提高终端用户20%甚至更多的响应时间。换到CDN是一个相当简单的代码变更，但这将急剧提升站点的响应速度。

**3.添上Expires或者Cache-Control HTTP头**

分类: 服务器

这条规则有两个方面：

* 对于静态组件：通过设置一个遥远的将来时间作为 Expires 来实现永不失效
* 多余动态组件：用合适的 Cache-Control HTTP头来让浏览器进行条件性的请求

网页设计越来越丰富，这意味着页面里有更多的脚本，图片和Flash。站点的新访客可能还是不得不提交几个HTTP请求，但通过使用有效期能让组件变得可缓存，这避免了在接下来的浏览过程中不必要的HTTP请求。有效期HTTP头通常被用在图片上，但它们应该用在 *所有* 组件上，包括脚本、样式和Flash组件。

浏览器（和代理）用缓存来减少HTTP请求的数目和大小，让页面能够更快加载。web服务器通过有效期HTTP响应头来告诉客户端，页面的各个组件应该被缓存多久。用一个遥远的将来时间做有效期，告诉浏览器这个响应在2010年4月15日前不会改变。

Expires: Thu, 15 Apr 2010 20:00:00 GMT

如果你用的是Apache服务器，用ExpiresDefault指令来设置相对于当前日期的有效期。下面的例子设置了从请求时间起10年的有效期：

ExpiresDefault "access plus 10 years"

记住，如果你用一个遥远的未来时间做有效期，就不得不在组件发生变化后及时修改组件的文件名。在Yahoo!，我们经常把这一步作为构建过程的一部分：把版本号内嵌在组件的文件名里，例如：yahoo\_2.0.6.js

用一个遥远的未来时间做有效期HTTP头，只有在用户已经访问过站点之后才会影响页面视图。如果是新访客或者浏览器的缓存被清空时，对HTTP请求的数量并没有影响。因此这种性能提升取决于已缓存各个组件的用户访问站点的频率。我们 [***在Yahoo!测量了这个数据***](http://yuiblog.com/blog/2007/01/04/performance-research-part-2/) ，发现已缓存各个组件的页面访问量（PV）占75%到85%。通过把一个遥远的未来时间作为有效期HTTP头，增加了被浏览器缓存的组件数量，在后续页面访问量中不需要用Internet连接多发送哪怕一个字节。

**4.Gzip组件**

分类: 服务器

前端工程师可以想办法明显地缩短通过网络传输HTTP请求和响应的时间。毫无疑问，终端用户的带宽速度，网络服务商，对等交换点的距离等等，都是开发团队所无法控制的。但还有别的能够影响响应时间的因素，压缩可以通过减少HTTP响应的大小来缩短响应时间。

从HTTP/1.1开始，web客户端就有了支持压缩的Accept-Encoding HTTP请求头。

Accept-Encoding: gzip, deflate

如果web服务器看到这个请求头，它就会用客户端列出的一种方式来压缩响应。web服务器通过Content-Encoding相应头来通知客户端。

Content-Encoding: gzip

Gzip是目前最常见的高效压缩方法，由GNU项目开发并被 [***RFC 1952***](http://www.ietf.org/rfc/rfc1952.txt) 标准化。唯一一个你可能会看到的其它压缩格式是deflate，但它效率不高而且并不常见。

Gzipping一般能够把响应压缩到70%左右，目前大约90%的通过浏览器的网络传输都支持gzip。如果是Apache服务器，配置gzip的模块取决于版本：Apache 1.3用[***mod\_gzip***](http://sourceforge.net/projects/mod-gzip/) 而Apache 2.x是 [***mod\_deflate***](http://httpd.apache.org/docs/2.0/mod/mod_deflate.html) 模块。

浏览器和代理的某些因素可能会引起浏览器所期望的和它收到的压缩内容不匹配。幸运的是，随着老旧浏览器的淘汰，这些极少遇到的情况正在逐渐减少，而且Apache模块可以通过自动添加合适的Vary响应头来帮你搞定。

服务器会根据文件类型来决定要不要用gzip压缩，但这非常有限。大多数网站都用gzip压缩HTML文件，其实压缩脚本，样式表也是不错的选择，但很多网站却错失了这个机会。其实，可以压缩任何文本内容，包括XML和JSON，而图片和PDF不用压缩，因为它们已经被压缩过了，再用gzip压缩不仅浪费CPU还可能会越压越大。

尽可能多地用gzip压缩能够给页面减肥，这也是提升用户体验最简单的方法。

**5.把样式表放在顶部**

分类: css

在Yahoo!研究性能的时候，我们发现把样式表放到文档的HEAD部分能让页面 *看起来* 加载地更快。这是因为把样式表放在head里能让页面逐步渲染。

关注性能的前端工程师想让页面逐步渲染。也就是说，我们想让浏览器尽快显示已有内容，这在页面上有一大堆内容或者用户网速很慢时显得尤为重要。给用户显示反馈（比如进度指标）的重要性已经被广泛研究过，并且被 [***记录***](http://www.useit.com/papers/responsetime.html) 下来了。在我们的例子中，HTML页面就是进度指标！当浏览器逐渐加载页面头部，导航条，顶部logo等等内容的时候，这些都被正在等待页面加载的用户当作反馈，能够提高整体用户体验。

在很多浏览器（包括IE）中，把样式表放在HTML文档底部都会阻止页面逐渐渲染。这些浏览器阻塞渲染过程，以避免因为样式变动而重绘页面元素，用户这时就只能盯着空白页面。

[***HTML官方文档***](http://www.w3.org/TR/html4/struct/links.html#h-12.3) 清楚地描述了样式表应该放在页面的HEAD里面：”Unlike A, [LINK] may only appear in the HEAD section of a document, although it may appear any number of times.”（不像a标签，link标签可能只出现在HEAD部分，虽然它能可以出现任意多次）。空白屏幕或者没有样式的falsh内容都是不可取的。理想方案就是遵循HTML官方文档，把样式表放在HTML文档的HEAD部分。

**6.把脚本放在底部**

分类: javascript

脚本会阻塞并行下载， [***HTTP/1.1官方文档***](http://www.w3.org/Protocols/rfc2616/rfc2616-sec8.html#sec8.1.4) 建议浏览器每个主机名下并行下载的组件数不要超过两个，如果图片来自多个主机名，并行下载的数量就可以超过两个。如果脚本正在下载，浏览器就不开始任何其它下载任务，即使是在不同主机名下的。

有时候，并不容易把脚本移动到底部。举个例子，如果脚本是用 document.write插入到页面内容中的，就没办法再往下移了。还可能存在作用域问题，在多数情况下，这些问题都是可以解决的。

一个常见的建议是用推迟（deferred）脚本，有 DEFER 属性的脚本意味着不能含有document.write，并且提示浏览器告诉他们可以继续渲染。不幸的是，Firefox不支持DEFER 属性。在IE中，脚本可能被推迟，但不尽如人意。如果脚本可以推迟，我们就可以把它放到页面底部，页面就可以更快地载入。

**7.避免使用CSS表达式**

分类: css

用CSS表达式动态设置CSS属性，是一种强大又危险的方式。从IE5开始支持，但 [***从IE8起就不推荐使用了***](http://msdn.microsoft.com/en-us/library/ms537634%28VS.85%29.aspx) 。例如，可以用CSS表达式把背景颜色设置成按小时交替的：

background-color: expression( (new Date())**.getHours**()%2 ? "**#B8D4FF**" : "**#F08A00**" );

上面的代码中， expression 方法可以接受一个JavaScript表达式。CSS属性会被设置成表达式的计算结果。 expression 方法会被其它浏览器忽略，所以只有想办法实现跨浏览器的与IE一致的用户体验才有用。

表达式最大的问题是它们经常被重复计算，比我们想象的次数还要多。不仅仅是页面渲染和调整大小的时候，在页面被滚动，甚至用户在页面上移动鼠标时都会重新计算表达式。给CSS表达式添加一个计数器就可以追踪它重新计算的时间和频率，而在页面上动动鼠标就可以引发10000多次重新计算。

减少CSS表达式重新计算的一种方式就是用一次性表达式，即在表达式第一次计算后就把样式属性设置成一个明确的值，换掉表达式。如果必须要在页面的整个生命周期中动态设置样式属性，可以用事件处理器来代替CSS表达式。如果必须使用CSS表达式，要记得它们可能会被重复计算上千次，从而影响整个页面的性能。

**8.把JavaScript和CSS放到外面**

分类: javascript, css

很多性能原则都是关于如何管理外部组件的，然而，在这些顾虑出现之前你应该问一个更基础的问题：应该把JavaScript和CSS放到外部文件中还是直接写在页面里？

实际上，用外部文件可以让页面更快，因为JavaScript和CSS文件会被缓存在浏览器。HTML文档中的行内JavaScript和CSS在每次请求该HTML文档的时候都会重新下载。这样做减少了所需的HTTP请求数，但增加了HTML文档的大小。另一方面，如果JavaScript和CSS在外部文件中，并且已经被浏览器缓存起来了，那么我们就成功地把HTML文档变小了，而且还没有增加HTTP请求数。

关键因素是，外部文件被缓存的频率和页面被请求数量之间的关系。尽管这个因素很难量化，但我们还是可以用各种各样的指标来衡量。如果用户的每个会话中都有多次页面访问，那么相同的脚本和样式表就可以被多个页面复用，缓存的外部文件就会带来巨大的好处。

很多站点在度量中都处于中等水平，对这些站点来说，一般最好的解决方案就是把JavaScript和CSS部署为外部文件。唯一的例外是主页上行内方式优先，例如 [***Yahoo!的首页***](http://www.yahoo.com/) 和 [***My Yahoo!***](http://my.yahoo.com/) 。在每个会话中访问量比较少的主页会发现行内JavaScript和CSS能让终端用户的响应时间更快。

对典型的站点来说，首页是众多访问量的开始，有很多技术可以对减少HTTP请求起到杠杆作用，就像用外部文件缓存的好处一样。这样的一种技术就是在首页用行内JavaScript和CSS，但在页面载入完成之后动态加载外部文件，这样后续的页面所需的外部文件就已经被放到浏览器的缓存里了。

**9.减少DNS查找**

分类: 内容

域名系统建立了主机名和IP地址间的映射，就像电话簿上人名和号码的映射一样。当你在浏览器输入www.yahoo.com的时候，浏览器就会联系DNS解析器返回服务器的IP地址。DNS是有成本的，它需要20到120毫秒去查找给定主机名的IP地址。在DNS查找完成之前，浏览器无法从主机名下载任何东西。

DNS查找被缓存起来更高效，由用户的ISP（网络服务提供商）或者本地网络存在一个特殊的缓存服务器上，但还可以缓存在个人用户的计算机上。DNS信息被保存在操作系统的DNS cache(微软Windows上的”DNS客户端服务”)里。大多数浏览器有独立于操作系统的自己的cache。只要浏览器在自己的cache里还保留着这条记录，它就不会向操作系统查询DNS。

IE默认缓存DNS查找30分钟，写在 DnsCacheTimeout 注册表设置中。Firefox缓存1分钟，可以用 network.dnsCacheExpiration 配置项设置。(Fasterfox把缓存时间改成了1小时 P.S. Fasterfox是FF的一个提速插件)

如果客户端的DNS cache是空的（包括浏览器的和操作系统的），DNS查找数等于页面上不同的主机名数，包括页面URL，图片，脚本文件，样式表，Flash对象等等组件中的主机名，减少不同的主机名就可以减少DNS查找。

减少不同主机名的数量同时也减少了页面能够并行下载的组件数量，避免DNS查找削减了响应时间，而减少并行下载数量却增加了响应时间。我的原则是把组件分散在2到4个主机名下，这是同时减少DNS查找和允许高并发下载的折中方案。

**10.压缩JavaScript和CSS**

分类: javascript, css

压缩具体来说就是从代码中去除不必要的字符以减少大小，从而提升加载速度。代码最小化就是去掉所有注释和不必要的空白字符（空格，换行和tab）。在JavaScript中这样做能够提高响应性能，因为要下载的文件变小了。两个最常用的JavaScript代码压缩工具是 [***JSMin***](http://crockford.com/javascript/jsmin) 和 [***YUI Compressor***](https://developer.yahoo.com/yui/compressor/) ，YUI compressor还可以压缩CSS。

混淆是一种可选的源码优化措施，要比压缩更复杂，所以混淆过程也更容易产生bug。在对美国前十的网站调查中，压缩可以缩小21%，而混淆能缩小25%。虽然混淆的缩小程度更高，但比压缩风险更大。

除了压缩外部脚本和样式，行内的 <script> 和 <style> 块也可以压缩。即使启用了gzip模块，先进行压缩也能够缩小5%或者更多的大小。JavaScript和CSS的用处越来越多，所以压缩代码会有不错的效果。

**11.避免重定向**

分类: 内容

重定向用301和302状态码，下面是一个有301状态码的HTTP头：

**HTTP/1.1 301 Moved Permanently**

Location: http://example.com/newuri

Content-Type: text/html

浏览器会自动跳转到 Location 域指明的URL。重定向需要的所有信息都在HTTP头部，而响应体一般是空的。其实额外的HTTP头，比如 Expires 和 Cache-Control 也表示重定向。除此之外还有别的跳转方式：refresh元标签和JavaScript，但如果你必须得做重定向，最好用标准的3xxHTTP状态码，主要是为了让返回按钮能正常使用。

牢记重定向会拖慢用户体验，在用户和HTML文档之间插入重定向会延迟页面上的所有东西，页面无法渲染，组件也无法开始下载，直到HTML文档被送达浏览器。

有一种常见的极其浪费资源的重定向，而且web开发人员一般都意识不到这一点，就是URL尾部缺少一个斜线的时候。例如，跳转到[***http://astrology.yahoo.com/astrology***](http://astrology.yahoo.com/astrology) 会返回一个重定向到[***http://astrology.yahoo.com/astrology/***](http://astrology.yahoo.com/astrology/) 的301响应（注意添在尾部的斜线）。在Apache中可以用 Alias ， mod\_rewrite 或者 DirectorySlash 指令来取消不必要的重定向。

重定向最常见的用途是把旧站点连接到新的站点，还可以连接同一站点的不同部分，针对用户的不同情况（浏览器类型，用户帐号类型等等）做一些处理。用重定向来连接两个网站是最简单的，只需要少量的额外代码。虽然在这些时候使用重定向减少了开发人员的开发复杂度，但降低了用户体验。一种替代方案是用 Alias 和mod\_rewrite ，前提是两个代码路径都在相同的服务器上。如果是因为域名变化而使用了重定向，就可以创建一条CNAME（创建一个指向另一个域名的DNS记录作为别名）结合 Alias 或者 mod\_rewrite 指令。

**12.去除重复脚本**

分类: javascript

页面含有重复的脚本文件会影响性能，这可能和你想象的不一样。在对美国前10大web站点的评审中，发现只有2个站点含有重复脚本。两个主要原因增加了在单一页面中出现重复脚本的几率：团队大小和脚本数量。在这种情况下，重复脚本会创建不必要的HTTP请求，执行无用的JavaScript代码，而影响页面性能。

IE会产生不必要的HTTP请求，而Firefox不会。在IE中，如果一个不可缓存的外部脚本被页面引入了两次，它会在页面加载时产生两个HTTP请求。即使脚本是可缓存的，在用户重新加载页面时也会产生额外的HTTP请求。

除了产生没有意义的HTTP请求之外，多次对脚本求值也会浪费时间。因为无论脚本是否可缓存，在Firefox和IE中都会执行冗余的JavaScript代码。

避免不小心把相同脚本引入两次的一种方法就是在模版系统中实现脚本管理模块。典型的脚本引入方法就是在HTML页面中用SCRIPT标签：

<script type="text/javascript" src="menu\_1.0.17.js"></script>

PHP中一个可选方案是创建一个叫 insertScript 的函数：

**<?php** insertScript("menu.js") **?>**

除了防止相同脚本被多次引入，这个函数还可以解决脚本相关的其它问题，比如依赖性检查和给脚本文件名添加版本号来支持“永久”有效期HTTP头。

**13.配置ETags**

分类: 服务器

实体标签（ETags），是服务器和浏览器用来决定浏览器缓存中组件与源服务器中的组件是否匹配的一种机制（“实体”也就是组件：图片，脚本，样式表等等）。添加ETags可以提供一种实体验证机制，比最后修改日期更加灵活。一个ETag是一个字符串，作为一个组件某一具体版本的唯一标识符。唯一的格式约束是字符串必须用引号括起来，源服务器用相应头中的 ETag 来指定组件的ETag：

**HTTP/1.1 200 OK**

Last-Modified: Tue, 12 Dec 2006 03:03:59 GMT

ETag: "10c24bc-4ab-457e1c1f"

Content-Length: 12195

然后，如果浏览器必须验证一个组件，它用 If-None-Match 请求头来把ETag传回源服务器。如果ETags匹配成功，会返回一个304状态码，这样就减少了12195个字节的响应体。

GET /i/yahoo.gif HTTP/1.1

Host: us.yimg.com

**If**-Modified-Since: Tue, 12 Dec 2006 03:03:59 GMT

**If**-None-Match: "10c24bc-4ab-457e1c1f"

HTTP/1.1 304 Not Modified

ETags存在的问题是它们是由特定服务器构造的，所以如果浏览器从一个服务器获取最初的组件，然后想验证另一个服务器上的相同组件，ETags是无法匹配成功的，而用一群服务器处理请求在web站点中又非常普遍。默认情况下，Apache和IIS会在ETag中嵌入数据，以大大降低在多服务器站点上有效性测试成功的几率。

Apache 1.3和2.x中ETag的格式是 inode-size-timestamp 。就算给定的文件可能在多个服务器的相同目录下，而且文件大小、访问权限、时间戳等等全部相同，它的i节点（P.S. inode，UNIX中的索引文件）在不同服务器中也不一样。

IIS5.0和6.0也都存在类似的问题。IIS中ETags的格式是Filetimestamp:ChangeNumber ， ChangeNumber 是一个用来追踪IIS配置变更的计数器。 一个站点在不同的IIS服务器上的 ChangeNumber 是不可能相同的。

最终结果是Apache和IIS为完全相同的组件生成的ETags无法跨浏览器匹配，如果ETags不匹配，用户就无法收到为又小又快的304响应设计的ETags。反而，他们将收到一个携带着组件所有数据的200正常响应。如果站点部署在单一服务器上，就根本不存在这个问题。但如果站点部署在多个服务器上，而且打算用Apache或者IIS的默认ETags配置，用户将看到缓慢的页面，服务器负载更高，还会消耗更大的带宽，并且代理也无法有效缓存页面内容。即使组件有“永久” Expires HTTP头，用户点击重新加载或者刷新的时候，仍然会发出条件GET请求。

如果不想用ETags提供的灵活的验证模型，最好把所有的Etag全都去掉，可以用基于组件的时间戳的 Last-Modified HTTP头验证，而且去掉ETag可以减少HTTP响应头以及后续请求的大小。 [***Microsoft Support article***](http://support.microsoft.com/?id=922733) 里写了怎样移除ETags。在Apache中，可以简单地通过在Apache配置文件中添上如下代码来实现：

FileETag none

**14.让Ajax可缓存**

分类: 内容

Ajax的一个好处是可以给用户提供即时反馈，因为它能够从后台服务器异步请求信息。然而，用了Ajax就无法保证用户在等待异步JavaScript和XML响应返回期间不会非常无聊。在很多应用程序中，用户能够一直等待取决于如何使用Ajax。例如，在基于web的电子邮件客户端中，用户为了寻找符合他们搜索标准的邮件消息，将会保持对Ajax请求返回结果的关注。重要的是，要记得“异步”并不意味着“即时”。

要提高性能，优化这些Ajax响应至关重要。最重要的提高Ajax性能的方法就是让响应变得可缓存，就像在 [***添上Expires或者Cache-Control HTTP头***](https://developer.yahoo.com/performance/rules.html#expires) 中讨论的一样。下面适用于Ajax的其它规则：

* [***Gzip组件***](https://developer.yahoo.com/performance/rules.html#gzip)
* [***减少DNS查找***](https://developer.yahoo.com/performance/rules.html#dns_lookups)
* [***压缩JavaScript***](https://developer.yahoo.com/performance/rules.html#minify)
* [***避免重定向***](https://developer.yahoo.com/performance/rules.html#redirects)
* [***配置ETags***](https://developer.yahoo.com/performance/rules.html#etags)

我们一起看看例子，一个Web 2.0的电子邮件客户端用了Ajax来下载用户的通讯录，以便实现自动完成功能。如果用户从上一次使用之后再没有修改过她的通讯录，而且Ajax响应是可缓存的，有尚未过期的Expires或者Cache-Control HTTP头，那么之前的通讯录就可以从缓存中读出。必须通知浏览器，应该继续使用之前缓存的通讯录响应，还是去请求一个新的。可以通过给通讯录的Ajax URL里添加一个表明用户通讯录最后修改时间的时间戳来实现，例如 &t=1190241612 。如果通讯录从上一次下载之后再没有被修改过，时间戳不变，通讯录就将从浏览器缓存中直接读出，从而避免一次额外的HTTP往返消耗。如果用户已经修改了通讯录，时间戳也可以确保新的URL不会匹配缓存的响应，浏览器将请求新的通讯录条目。

即使Ajax响应是动态创建的，而且可能只适用于单用户，它们也可以被缓存，而这样会让你的Web 2.0应用更快。

**15.尽早清空缓冲区**

分类: 服务器

当用户请求一个页面时，服务器需要用大约200到500毫秒来组装HTML页面，在这期间，浏览器闲等着数据到达。PHP中有一个 [***flush()***](http://php.net/flush) 函数，允许给浏览器发送一部分已经准备完毕的HTML响应，以便浏览器可以在后台准备剩余部分的同时开始获取组件，好处主要体现在很忙的后台或者很“轻”的前端页面上（P.S. 也就是说，响应时耗主要在后台方面时最能体现优势）。

比较理想的清空缓冲区的位置是HEAD后面，因为HTML的HEAD部分通常更容易生成，并且允许引入任何CSS和JavaScript文件，这样就可以让浏览器在后台还在处理的时候就开始并行获取组件。

例如：

... *<!-- css, js -->*

</head>

**<?php flush(); ?>**

<body>

... *<!-- content -->*

[***Yahoo!搜索***](http://search.yahoo.com/) 开创了这项技术，而且真实用户测试研究也证明了使用这种技术的诸多好处。

**16.对Ajax用GET请求**

分类: 服务器

[***Yahoo!邮箱***](http://mail.yahoo.com/) 团队发现使用 XMLHttpRequest 时，浏览器的POST请求是通过一个两步的过程来实现的：先发送HTTP头，在发送数据。所以最好用GET请求，它只需要发送一个TCP报文（除非cookie特别多）。IE的URL长度最大值是2K，所以如果要发送的数据超过2K就无法使用GET了。

POST请求的一个有趣的副作用是实际上没有发送任何数据，就像GET请求一样。正如 [***HTTP说明文档***](http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html) 中描述的，GET请求是用来检索信息的。所以它的语义只是用GET请求来请求数据，而不是用来发送需要存储到服务器的数据。

**17.延迟加载组件**

分类: 内容

可以凑近看看页面并问自己：什么才是一开始渲染页面所必须的？其余内容都可以等会儿。

JavaScript是分隔onload事件之前和之后的一个理想选择。例如，如果有JavaScript代码和支持拖放以及动画的库，这些都可以先等会儿，因为拖放元素是在页面最初渲染之后的。其它可以延迟加载的部分包括隐藏内容（在某个交互动作之后才出现的内容）和折叠的图片。

工具可帮你减轻工作量： [***YUI Image Loader***](https://developer.yahoo.com/yui/imageloader/) 可以延迟加载折叠的图片，还有 [***YUI Get utility***](https://developer.yahoo.com/yui/get/) 是一种引入JS和CSS的简单方法。 [***Yahoo!主页***](http://www.yahoo.com/) 就是一个例子，可以打开Firebug的网络面板仔细看看。

最好让性能目标符合其它web开发最佳实践，比如“渐进增强”。如果客户端支持JavaScript，可以提高用户体验，但必须确保页面在不支持JavaScript时也能正常工作。所以，在确定页面运行正常之后，可以用一些延迟加载脚本增强它，以支持一些拖放和动画之类的华丽效果。

**18.预加载组件**

分类: 内容

预加载可能看起来和延迟加载是相反的，但它其实有不同的目标。通过预加载组件可以充分利用浏览器空闲的时间来请求将来会用到的组件（图片，样式和脚本）。用户访问下一页的时候，大部分组件都已经在缓存里了，所以在用户看来页面会加载得更快。

实际应用中有以下几种预加载的类型：

* *无条件* 预加载——尽快开始加载，获取一些额外的组件。google.com就是一个sprite图片预加载的好例子，这个sprite图片并不是google.com主页需要的，而是搜索结果页面上的内容。
* *条件性* 预加载——根据用户操作猜测用户将要跳转到哪里并据此预加载。在[***search.yahoo.com***](http://search.yahoo.com/) 的输入框里键入内容后，可以看到那些额外组件是怎样请求加载的。
* *提前* 预加载——在推出新设计之前预加载。经常在重新设计之后会听到：“这个新网站不错，但比以前更慢了”，一部分原因是用户访问先前的页面都是有旧缓存的，但新的却是一种空缓存状态下的体验。可以通过在将要推出新设计之前预加载一些组件来减轻这种负面影响，老站可以利用浏览器空闲的时间来请求那些新站需要的图片和脚本。

**19.减少DOM元素的数量**

分类: 内容

一个复杂的页面意味着要下载更多的字节，而且用JavaScript访问DOM也会更慢。举个例子，想要添加一个事件处理器的时候，循环遍历页面上的500个DOM元素和5000个DOM元素是有区别的。

大量的DOM元素是一种征兆——页面上有些内容无关的标记需要清理。正在用嵌套表格来布局吗？还是为了修复布局问题而添了一堆的 <div> s？或许应该用更好的语义化标记。

[***YUI CSS utilities***](https://developer.yahoo.com/yui/) 对布局有很大帮助：grids.css针对整体布局，fonts.css和reset.css可以用来去除浏览器的默认格式。这是个开始清理和思考标记的好机会，例如只在语义上有意义的时候使用 <div> ，而不是因为它能够渲染一个新行。

DOM元素的数量很容易测试，只需要在Firebug的控制台里输入：

document.getElementsByTagName('\*').length

那么多少DOM元素才算是太多呢？可以参考其它类似的标记良好的页面，例如[***Yahoo!主页***](http://www.yahoo.com/) 是一个相当繁忙的页面，但只有不到700个元素（HTML标签）。

**20.跨域分离组件**

分类: 内容

分离组件可以最大化并行下载，但要确保只用不超过2-4个域，因为存在DNS查找的代价。例如，可以把HTML和动态内容部署在 www.example.org ，而把静态组件分离到 static1.example.org 和 static2.example.org 。

更多信息请查看Tenni Theurer和Patty Chi的文章： [***Maximizing Parallel Downloads in the Carpool Lane***](http://yuiblog.com/blog/2007/04/11/performance-research-part-4/)

**21.尽量少用iframe**

分类: 内容

用iframe可以把一个HTML文档插入到父文档里，重要的是明白iframe是如何工作的并高效地使用它。

<iframe> 的优点：

* 引入缓慢的第三方内容，比如标志和广告
* 安全沙箱
* 并行下载脚本

<iframe> 的缺点：

* 代价高昂，即使是空白的iframe
* 阻塞页面加载
* 非语义

**22.杜绝404**

分类: 内容

HTTP请求代价高昂，完全没有必要用一个HTTP请求去获取一个无用的响应（比如404 Not Found），只会拖慢用户体验而没有任何好处。

有些站点用的是有帮助的404——“你的意思是xxx？”，这样做有利于用户体验，，但也浪费了服务器资源（比如数据库等等）。最糟糕的是链接到的外部JavaScript有错误而且结果是404。首先，这种下载将阻塞并行下载。其次，浏览器会试图解析404响应体，因为它是JavaScript代码，需要找出其中可用的部分。

**23.给Cookie减肥**

分类: cookie

使用cookie的原因有很多，比如授权和个性化。HTTP头中cookie信息在web服务器和浏览器之间交换。重要的是保证cookie尽可能的小，以最小化对用户响应时间的影响。

更多信息请查看Tenni Theurer和Patty Chi的文章： [***When the Cookie Crumbles***](http://yuiblog.com/blog/2007/03/01/performance-research-part-3/) 。相关经验原则可以总结如下：

* 清除不必要的cookie
* 保证cookie尽可能小，以最小化对用户响应时间的影响
* 注意给cookie设置合适的域级别，以免影响其它子域
* 设置合适的有效期，更早的有效期或者none可以更快的删除cookie，提高用户响应时间

**24.把组件放在不含cookie的域下**

分类: cookie

当浏览器发送对静态图像的请求时，cookie也会一起发送，而服务器根本不需要这些cookie。所以它们只会造成没有意义的网络通信量，应该确保对静态组件的请求不含cookie。可以创建一个子域，把所有的静态组件都部署在那儿。

如果域名是 www.example.org ，可以把静态组件部署到 static.example.org。然而，如果已经在顶级域 example.org 或者 www.example.org 设置了cookie，那么所有对 static.example.org 的请求都会含有这些cookie。这时候可以再买一个新域名，把所有的静态组件部署上去，并保持这个新域名不含cookie。Yahoo!用的是yimg.com ，YouTube是 ytimg.com ，Amazon是 images-amazon.com 等等。

把静态组件部署在不含cookie的域下还有一个好处是有些代理可能会拒绝缓存带cookie的组件。有一点需要注意：如果不知道应该用example.org还是www.example.org作为主页，可以考虑一下cookie的影响。省略www的话，就只能把cookie写到 \*.example.org ，所以因为性能原因最好用www子域，并且把cookie写到这个子域下。

**25.尽量减少DOM访问**

分类: javascript

用JavaScript访问DOM元素是很慢的，所以，为了让页面反应更迅速，应该：

* 缓存已访问过的元素的索引
* 先“离线”更新节点，再把它们添到DOM树上
* 避免用JavaScript修复布局问题

更多信息请查看YUI影院里Julien Lecomte的文章： [***High Performance Ajax Applications***](http://yuiblog.com/blog/2007/12/20/video-lecomte/)

**26.用智能的事件处理器**

分类: javascript

有时候感觉页面反映不够灵敏，是因为有太多频繁执行的事件处理器被添加到了DOM树的不同元素上，这就是推荐使用 *事件委托* 的原因。如果一个 div 里面有10个按钮，应该只给div容器添加一个事件处理器，而不是给每个按钮都添加一个。事件能够冒泡，所以可以捕获事件并得知哪个按钮是事件源。

不需要为了处理DOM树而等待onload事件，通常只要目标元素在DOM树中可访问即可，而不必等待所有的图片下载完成。可以考虑用 DOMContentLoaded 来代替onload事件，但为了让它在所有浏览器中都可用，可以用 [***YUI Event***](https://developer.yahoo.com/yui/event/) 工具，它有一个[***onAvailable***](https://developer.yahoo.com/yui/event/#onavailable) 方法。

更多信息请查看YUI影院里Julien Lecomte的文章： [***High Performance Ajax Applications***](http://yuiblog.com/blog/2007/12/20/video-lecomte/)

**27.选择<link>舍弃@import**

分类: css

前面提到了一个最佳实践：为了实现逐步渲染，CSS应该放在顶部。

在IE中用 @import 与在底部用 <link> 效果一样，所以最好不要用它。

**28.避免使用滤镜**

分类: css

IE专有的 AlphaImageLoader 滤镜可以用来修复IE7之前的版本中半透明PNG图片的问题。在图片加载过程中，这个滤镜会阻塞渲染，卡住浏览器，还会增加内存消耗而且是被应用到每个元素的，而不是每个图片，所以会存在一大堆问题。

最好的方法是干脆不要用 AlphaImageLoader ，而优雅地降级到用在IE中支持性很好的PNG8图片来代替。如果非要用 AlphaImageLoader ，应该用下划线hack：\_filter 来避免影响IE7及更高版本的用户。

**29.优化图片**

分类: 图片

设计师做好图片后，在把这些图片通过FTP上传到web服务器之前，我们还可以做一些事情。

* 检查GIF图片，看看图片中是不是用了调色板大小对应的颜色数，用 [***imagemagick***](http://www.imagemagick.org/)可以简单地检查：   
  identify -verbose image.gif   
  如果4色图片用了调色板中256色的“槽”，那就还有改进的余地
* 试着把GIF图片转换成PNG，看能不能缩减大小，往往可以。开发者通常不愿意用PNG图片，因为浏览器支持有限，但这都是过去的事情了。真正的问题是PNG图片完全支持alpha透明度，而GIF图片却不支持透明度渐变，所以GIF能做的任何事情，PNG都可以（除了动画）。下面这个简单的命令就能让PNG图片可以安全使用：   
  convert image.gif image.png   
  “我们强调的是：给PNG一个机会。”
* 用 [***pngcrush***](http://pmt.sourceforge.net/pngcrush/) （或者其它的PNG优化工具）处理所有的PNG图片，例如：   
  pngcrush image.png -rem alla -reduce -brute result.png
* 用jpegtran处理所有JPEG图片，这个工具支持对JPEG图片的无损操作比如旋转，还可以用来去除注释和其它无用信息（比如EXIF信息 P.S. 数码照片信息，焦距光圈之类的）：   
  jpegtran -copy none -optimize -perfect src.jpg dest.jpg

**30.优化CSS Sprite**

分类: 图片

* 在Sprite图片中横向排列一般都比纵向排列的最终文件小
* 组合Sprite图片中的相似颜色可以保持低色数，最理想的是256色以下PNG8格式
* “对移动端友好”，不要在Sprite图片中留下太大的空隙。虽然不会在很大程度上影响图片文件的大小，但这样做可以节省用户代理把图片解压成像素映射时消耗的内存。100×100的图片是1万个像素，而1000×1000的图片就是100万个像素了。

**31.不要用HTML缩放图片**

分类: 图片

不要因为在HTML中可以设置宽高而使用本不需要的大图。如果需要

<img width="100" height="100" src="mycat.jpg" alt="My Cat" />

那么图片本身（mycat.jpg）应该是100x100px的，而不是去缩小500x500px的图片。

**32.用小的可缓存的favicon.ico（P.S. 收藏夹图标）**

分类: 图片

favicon.ico是放在服务器根目录的图片，它会带来一堆麻烦，因为即便你不管它，浏览器也会自动请求它，所以最好不要给一个 404 Not Found 响应。而且只要在同一个服务器上，每次请求它时都会发送cookie，此外这个图片还会干扰下载顺序，例如在IE中，当你在onload中请求额外组件时，将会先下载favicon。

所以为了缓解favicon.ico的缺点，应该确保：

* 足够小，最好在1K以下
* 设置合适的有效期HTTP头（以后如果想换的话就不能重命名了），把有效期设置为几个月后一般比较安全，可以通过检查当前favicon.ico的最后修改日期来确保变更能让浏览器知道。

[***Imagemagick***](http://www.imagemagick.org/) 可以用来处理小收藏夹图标

**33.保证所有组件都小于25K**

分类: 移动

这个限制是因为iPhone不能缓存大于25K的组件，注意这里指的是 *未压缩的* 大小。这就是为什么缩减内容本身也很重要，因为单纯的gzip可能不够。

更多信息请查看Wayne Shea和Tenni Theurer的文章： [***Performance Research, Part 5: iPhone Cacheability – Making it Stick***](http://yuiblog.com/blog/2008/02/06/iphone-cacheability/)

**34.把组件打包到一个复合文档里**

分类: 移动

把各个组件打包成一个像有附件的电子邮件一样的复合文档里，可以用一个HTTP请求获取多个组件（记住一点：HTTP请求是代价高昂的）。用这种方式的时候，要先检查用户代理是否支持（iPhone就不支持）。

**35.避免图片src属性为空**

分类: 服务器

Image with empty string **src** 属性是空字符串的图片很常见，主要以两种形式出现：

1. straight HTML

<img src=””>

1. JavaScript

var img = new Image();img.src = “”;

这两种形式都会引起相同的问题：浏览器会向服务器发送另一个请求。

* **IE** 向页面所在目录发起一个请求
* **Safari和Chrome** 想当前页面本身发送一个请求
* **Firefox** 3及更早版本与Safari和Chrome处理方式一样，但3.5解决了这个问题 [***[bug 444931]***](https://bugzilla.mozilla.org/show_bug.cgi?id=444931) ，不会再发送请求了
* **Opera** 遇到有空src属性的图片不做任何处理

为什么图片src属性为空不好？

1. 意外发送大量的通信量对服务器来说是很伤的，尤其是在每天有几百万访问量页面的时候。
2. 浪费服务器资源去生成一个根本不可能被看到的页面
3. 可能会污染用户数据，如果追踪请求状态，要么通过cookie要么是其它方式，可能会破坏用户数据。即使图片请求并没有返回图片，整个HTTP头部也会被浏览器接受并读取，包括所有的cookie。虽然其余部分会被丢弃，但这可能已经造成破坏了。

问题的根本原因是各个浏览器在处理URI时的分歧，这在RFC 3986 – Uniform Resource Identifiers文档中有明确定义。如果URI是一个空串，会被看作一个相对URI，并按照5.2节定义的算法处理。实际情况是，Firefox、Safari和Chrome都是根据文档中5.4节列出的规范来处理空串，而IE并没有正确处理。据说在旧版本规范文档RFC 2396 – Uniform Resource Identifiers（被RFC 3986废弃了）中，所以从严格意义上来说浏览器处理相对URI的做法都是对的。问题是，在这种情形下，空串显然是无心的（P.S. 而不是什么相对URI）。

HTML5的4.8.2节有关于<img>标签src属性的描述，规定浏览器不再发送额外请求：

The src attribute must be present, and must contain a valid URL referencing a non-interactive, optionally animated, image resource that is neither paged nor scripted. If the base URI of the element is the same as the document’s address, then the src attribute’s value must not be the empty string.（P.S. “src属性必须存在，而且必须有一个合法的URL引用非交互式的动画或者图像资源，不能分页也不能含有脚本。如果元素的基URI和文档地址相同，那么src属性的值就不能是空串。”）

希望将来浏览器不会存在这个问题，不幸的是，没有针对<script src=””>和<link href=””>条款，也许还有时间调整，以确保浏览器不会意外地实现这一行为。

这条原则是受了Yahoo!的JavaScript大师Nicolas C. Zakas的启发，更多信息请查看他的文章： [***Empty image src can destroy your site***](http://www.nczonline.net/blog/2009/11/30/empty-image-src-can-destroy-your-site/)