Web 应用性能优化黄金法则：先优化前端程序 （front-end） 的性能，因为这是 80% 或以上的最终用户响应时间的花费所在。

法则 1. 减少 HTTP 请求次数

80%的最终用户响应时间花在前端程序上，而其大部分时间则花在各种页面元素，如图像、样式表、脚本和 Flash 等，的下载上。减少页面元素将会减少 HTTP 请求次数。这是快速显示页面的关键所在。

一种减少页面元素个数的方法是简化页面设计。但是否存在其他方式，能做到既有丰富内容，又能获得快速响应时间呢？以下是这样一些技术：

Image maps 组合多个图片到一张图片中。总文件大小变化不大，但减少了 HTTP 请求次数从而加快了页面显示速度。该方式只适合图片连续的情况；同时坐标的定义是烦人又容易出错的工作。

CSS Sprites 是更好的方法。它可以组合页面中的图片到单个文件中，并使用 CSS 的 background-image 和 background-position 属性来现实所需的部分图片。

Inline images 使用 data: URL scheme 来在页面中内嵌图片。这将增大 HTML 文件的大小。组合 inline images 到你的（缓存）样式表是既能较少 HTTP 请求，又能避免加大 HTML 文件大小的方法。

Combined files 通过组合多个脚本文件到单一文件来减少 HTTP 请求次数。样式表也可采用类似方法处理。这个方法虽然简单，但没有得到大规模的使用。10 大美国网站每页平均有 7 个脚本文件和 2 个样式表。当页面之间脚本和样式表变化很大时，该方式将遇到很大的挑战，但如果做到的话，将能加快响应时间。

减少 HTTP 请求次数是性能优化的起点。这最提高首次访问的效率起到很重要的作用。据 Tenni Theurer 的文章 Browser Cache Usage - Exposed! 描述，40-60% 的日常访问是首次访问，因此为首次访问者加快页面访问速度是用户体验的关键。

法则 2. 使用 CDN(Content Delivery Network, 内容分发网络 ）

用户离 web server 的远近对响应时间也有很大影响。从用户角度看，把内容部署到多个地理位置分散的服务器上将有效提高页面装载速度。但是该从哪里开始呢？

作为实现内容地理分布的第一步，不要试图重构 web 应用以适应分布架构。改变架构将导致多个周期性任务，如同步 session 状态，在多个 server 之间复制数据库交易。这样缩短用户与内容距离的尝试可能被应用架构改版所延迟，或阻止。

我们还记得 80-90%的最终用户响应时间花在下载页面中的各种元素上，如图像文件、样式表、脚本和 Flash 等。与其花在重构系统这个困难的任务上，还不如先分布静态内容。这不仅能大大减少响应时间，而且由于 CDN 的存在，分布静态内容非常容易实现。

CDN 是地理上分布的 web server 的集合，用于更高效地发布内容。通常基于网络远近来选择给具体用户服务的 web server。

一些大型网站拥有自己的 CDN，但是使用如 Akamai Technologies, Mirror Image Internet, 或 Limelight Networks 等 CDN 服务提供商的服务将是划算的。在 Yahoo!把静态内容分布到 CDN 减少了用户影响时间 20%或更多。切换到 CDN 的代码修改工作是很容易的，但能达到提高网站的速度。

法则 3. 增加 Expires Header

网页内容正变得越来越丰富，这意味着更多的脚本文件、样式表、图像文件和 Flash。首次访问者将不得不面临多次 HTTP 请求，但通过使用 Expires header，您可以在客户端缓存这些元素。这在后续访问中避免了不必要的 HTTP 请求。Expires header 最常用于图像文件，但是它也应该用于脚本文件、样式表和 Flash。

浏览器（和代理）使用缓存来减少 HTTP 请求的次数和大小，使得网页加速装载。Web server 通过 Expires header 告诉客户端一个元素可以缓存的时间长度。

如果服务器是 Apache 的话，您可以使用 ExpiresDefault 基于当期日期来设置过期日期，如：

ExpiresDefault “access plus 10 years” 设置过期时间为从请求时间开始计算的 10 年。

请记住，如果使用超长的过期时间，则当内容改变时，您必须修改文件名称。在 Yahoo!我们经常把改名作为 release 的一个步骤：版本号内嵌在文件名中，如 yahoo_2.0.6.js。

法则 4. 压缩页面元素

通过压缩 HTTP 响应内容可减少页面响应时间。从 HTTP/1.1 开始，web 客户端在 HTTP 请求中通过 Accept-Encoding 头来表明支持的压缩类型，如：

Accept-Encoding: gzip, deflate.

如果 Web server 检查到 Accept-Encoding 头，它会使用客户端支持的方法来压缩 HTTP 响应，会设置 Content-Encoding 头，如：Content-Encoding: gzip。

Gzip 是目前最流行及有效的压缩方法。其他的方式如 deflate，但它效果较差，也不够流行。通过 Gzip，内容一般可减少 70%。如果是 Apache，在 1.3 版本下需使用 mod_gzip 模块，而在 2.x 版本下，则需使用 mod_deflate。

Web server 根据文件类型来决定是否压缩。大部分网站对 HTML 文件进行压缩。但对脚本文件和样式表进行压缩也是值得的。实际上，对包括 XML 和 JSON 在内的任务文本信息进行压缩都是值得的。图像文件和 PDF 文件不应该被压缩，因为它们本来就是压缩格式保存的。对它们进行压缩，不但浪费 CPU，而且还可能增加文件的大小。

因此，对尽量多的文件类型进行压缩是一种减少页面大小和提高用户体验的简便方法。

法则 5. 把样式表放在头上

我们发现把样式表移到 HEAD 部分可以提高界面加载速度，因此这使得页面元素可以顺序显示。

在很多浏览器下，如 IE，把样式表放在 document 的底部的问题在于它禁止了网页内容的顺序显示。浏览器阻止显示以免重画页面元素，那用户只能看到空白页了。Firefox 不会阻止显示，但这意味着当样式表下载后，有些页面元素可能需要重画，这导致闪烁问题。

HTML 规范明确要求样式表被定义在 HEAD 中，因此，为避免空白屏幕或闪烁问题，最好的办法是遵循 HTML 规范，把样式表放在 HEAD 中。

法则 6. 把脚本文件放在底部

与样式文件一样，我们需要注意脚本文件的位置。我们需尽量把它们放在页面的底部，这样一方面能顺序显示，另方面可达到最大的并行下载。

浏览器会阻塞显示直到样式表下载完毕，因此我们需要把样式表放在 HEAD 部分。而对于脚本来说，脚本后面内容的顺序显示将被阻塞，因此把脚本尽量放在底部意味着更多内容能被快速显示。

脚本引起的第二个问题是它阻塞并行下载数量。HTTP/1.1 规范建议浏览器每个主机的并行下载数不超过 2 个。因此如果您把图像文件分布到多台机器的话，您

可以达到超过 2 个的并行下载。但是当脚本文件下载时，浏览器不会启动其他的并行下载，甚至其他主机的下载也不启动。

在某些情况下，不是很容易就能把脚本移到底部的。如，脚本使用 document.write 方法来插入页面内容。同时可能还存在域的问题。不过在很多情况下，还是有一些方法的。

一个备选方法是使用延迟脚本（deferred script）。DEFER 属性表明脚本未包含 document.write，指示浏览器刻继续显示。不幸的是，Firefox 不支持 DEFER 属性。在 IE 中，脚本可能被延迟执行，但不一定得到需要的长时间延迟。不过从另外角度来说，如果脚本能被延迟执行，那它就可以被放在底部了。

法则 7.　避免 CSS 表达式

CSS 表达式是功能强大的(同时也是危险的)用于动态设置 CSS 属性的方式。IE，从版本 5 开始支持 CSS 表达式，如 backgourd-color: expression((new Date()).getHours()%2?”#B8D4FF”:”#F08A00”)，即背景色每个小时切换一次。

CSS 表达式的问题是其执行次数超过大部分人的期望。不仅页面显示和 resize 时计算表达式，而且当页面滚屏，甚至当鼠标在页面上移动时都会重新计算表达式。

一种减少 CSS 表达式执行次数的方法是一次性表达式，即当第一次执行时就以明确的数值代替表达式。如果必须动态设置的话，可使用事件处理函数代替。如果您必须使用 CSS 表达式的话，请记住它们可能被执行上千次，从而影响页面性能。

法则 8.　把 JavaScript　和 CSS 放到外部文件中

上述很多性能优化法则都基于外部文件进行优化。现在，我们必须问一个问题：JavaScript 和 CSS 应该包括在外部文件，还是在页面文件中？

在现实世界中，使用外部文件会加快页面显示速度，因为外部文件会被浏览器缓存。如果内置 JavaScript 和 CSS 在页面中虽然会减少 HTTP 请求次数，但增大了页面的大小。另外一方面，使用外部文件，会被浏览器缓存，则页面大小会减小，同时又不增加 HTTP 请求次数。

因此，一般来说，外部文件是更可行的方式。唯一的例外是内嵌方式对主页更有效，如 Yahoo!和 My Yahoo!都使用内嵌方式。一般来说，在一个 session 中，主页访问此时较少，因此内嵌方式可以取得更快的用户响应时间。

法则 9. 减少 DNS 查询次数

DNS 用于映射主机名和 IP 地址，一般一次解析需要 20～120 毫秒。为达到更高的性能，DNS 解析通常被多级别地缓存，如由 ISP 或局域网维护的 caching server，本地机器操作系统的缓存（如 windows 上的 DNS Client Service），浏览器。IE 的缺省 DNS 缓存时间为 30 分钟，Firefox 的缺省缓冲时间是 1 分钟。

减少主机名可减少 DNS 查询的次数，但可能造成并行下载数的减少。避免 DNS 查询可减少响应时间，而减少并行下载数可能增加响应时间。一个可行的折中是把内容分布到至少 2 个，最多 4 个不同的主机名上。

法则 10. 最小化 JavaScript 代码

最小化 JavaScript 代码指在 JS 代码中删除不必要的字符，从而降低下载时间。两个流行的工具是 JSMin 和 YUI Compressor。

混淆是最小化于源码的备选方式。象最小化一样，它通过删除注释和空格来减少源码大小，同时它还可以对代码进行混淆处理。作为混淆的一部分，函数名和变量名被替换成短的字符串，这使得代码更紧凑，同时也更难读，使得难于被反向工程。Dojo Compressor (ShrinkSafe)是最常见的混淆工具。

最小化是安全的、直白的过程，而混淆则更复杂，而且容易产生问题。从对美国 10 大网站的调查来看，通过最小化，文件可减少 21%，而混淆则可减少 25%。

除了最小化外部脚本文件外，内嵌的脚本代码也应该被最小化。即使脚本根据法则 4 被压缩后传输，最小化脚本刻减少文件大小 5%或更高。

法则 11. 避免重定向

重定向功能是通过 301 和 302 这两个 HTTP 状态码完成的，如：

```
HTTP/1.1 301 Moved Permanently
Location: http://example.com/newuri
Content-Type: text/html
```

浏览器自动重定向请求到 Location 指定的 URL 上，重定向的主要问题是降低了用户体验。

一种最耗费资源、经常发生而很容易被忽视的重定向是 URL 的最后缺少/，如访问 http://astrology.yahoo.com/astrology 将被重定向到

http://astrology.yahoo.com/astrology/。在 Apache 下，可以通过 Alias，mod_rewrite 或 DirectorySlash 等方式来解决该问题。

法则 12. 删除重复的脚本文件

在一个页面中包含重复的 JS 脚本文件会影响性能，即它会建立不必要的 HTTP 请求和额外的 JS 执行。

不必要的 HTTP 请求发生在 IE 下，而 Firefox 不会产生多余的 HTTP 请求。额外的 JS 执行，不管在 IE 下，还是在 Firefox 下，都会发生。

一个避免重复的脚本文件的方式是使用模板系统来建立脚本管理模块。除了防止重复的脚本文件外，该模块还可以实现依赖性检查和增加版本号到脚本文件名中，从而实现超长的过期时间。

法则 13. 配置 ETags

ETags 是用于确定浏览器缓存中元素是否与 Web server 中的元素相匹配的机制，它是比 last-modified date 更灵活的元素验证机制。ETag 是用于唯一表示元素版本的字符串，它需被包括在引号中。Web server 首先在 response 中指定 ETag：

```
      HTTP/1.1 200 OK
< 03:03:59 2006 Dec 12>
10c24bc-4ab-457e1c1f"
      Content-Length: 12195
```

后来，如果浏览器需要验证某元素，它使用 If-None-Match 头回传 ETag 给 Web server，如果 ETag 匹配，则服务器返回 304 代码，从而节省了下载时间：

```
      GET /i/yahoo.gif HTTP/1.1
      Host: us.yimg.com
< 03:03:59 2006 Dec 12>
10c24bc-4ab-457e1c1f"
      HTTP/1.1 304 Not Modified
```

ETags 的问题在于它们是基于服务器唯一性的某些属性构造的，如 Apache1.3 和 2.x，其格式是 inode-size-timestamp，而在 IIS5.0 和 6.0 下，其格式是 Filetimestamp:ChangeNumber。这样同一个元素在不同的 web server 上，其 ETag 是不一样的。这样在多 Web server 的环境下，浏览器先从 server1 请求某元素，后来向 server2 验证该元素，由于 ETag 不同，所以缓存失效，必须重新下载。

因此，如果您未用到ETags系统提供的灵活的验证机制，最好删除ETag。删除ETag会减少http response及后续请求的HTTP头的大小。微软支持文章描述了如何删除ETags，而在Apache下，只要在配置文件中设置FileETag none即可。

法则 14.　缓存 Ajax

性能优化法则同样适用于web 2.0应用。提高Ajax的性能最重要的方式是使得其response可缓存，就象“法则3增加Expires Header”讨论的那样。以下其他法则同样适用于Ajax，当然法则3是最有效的方式：