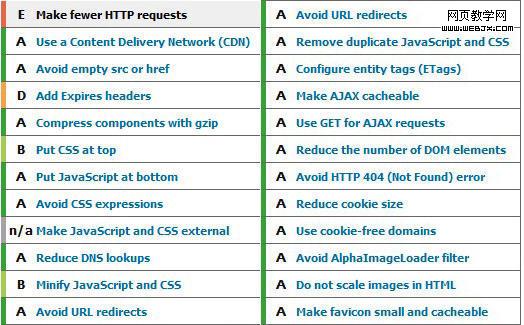
本来这是个老生常谈的问题，上周自成又分享了一些性能优化的建议，我这里再做一个全面的Tips整理，谨作为查阅型的文档，不妥之处，还请指正；  
如果你已经对yahoo这些优化建议烂熟于心，果断点这里

**一、 Yahoo的军规条例：**

谨记：80%-90%的终端响应时间是花费在下载页面中的图片，样式表，脚本，flash等；  
详细的解释来这里查：http://developer.yahoo.com/performance/rules.html  
也可以直接firebug上一项项比对，如下图：



简单翻译解释下：

**1、尽量减少HTTP请求个数——须权衡**

合并图片（如css sprites，内置图片使用数据）、合并CSS、JS，这一点很重要，但是要考虑合并后的文件体积。

**2、使用CDN（内容分发网络）**

这里可以关注CDN的三类实现：镜像、高速缓存、专线，以及智能路由器和负载均衡；

**3、为文件头指定Expires或Cache-Control，使内容具有缓存性。**

区分静态内容和动态内容，避免以后页面访问中不必要的HTTP请求。

**4、避免空的src和href**

留意具有这两个属性的标签如link，script，img，iframe等；

**5、使用gzip压缩内容**

Gzip压缩所有可能的文件类型以来减少文件体积

**6、把CSS放到顶部**

实现页面有秩序地加载，这对于拥有较多内容的页面和网速较慢的用户来说更为重要，同时，HTML规范清楚指出样式表要放包含在页面的<head />区域内；

**7、把JS放到底部**

HTTP/1.1 规范建议，浏览器每个主机名的并行下载内容不超过两个，而问题在于脚本阻止了页面的平行下载，即便是主机名不相同

**8、避免使用CSS表达式**

页面显示和缩放，滚动、乃至移动鼠标时，CSS表达式的计算频率是我们要关注的。可以考虑一次性的表达式或者使用事件句柄来代替CSS表达式。

**9、将CSS和JS放到外部文件中**

我们需要权衡内置代码带来的HTTP请求减少与通过使用外部文件进行缓存带来的好处的折中点。

**10、减少DNS查找次数**

我们需要权衡减少 DNS查找次数和保持较高程度并行下载两者之间的关系。

**11、精简CSS和JS**

目的就是减少下载的文件体积，可考虑压缩工具JSMin和YUI Compressor。

**12、避免跳转**

为了确保“后退”按钮可以正确地使用，使用标准的 3XXHTTP状态代码；同域中注意避免反斜杠 “/” 的跳转；  
跨域使用 Alias或者 mod\_rewirte建立 CNAME（保存一个域名和另外一个域名之间关系的DNS记录）

**13、剔除重复的JS和CSS**

重复调用脚本，除了增加额外的HTTP请求外，多次运算也会浪费时间。在IE和Firefox中不管脚本是否可缓存，它们都存在重复运算JavaScript的问题。

**14、配置ETags**

Entity tags（ETags）（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（“实体”就是所说的“内 容”，包括图片、脚本、样式表等），是比last-modified date更更加灵活的机制，单位时间内文件被修过多次，Etag可以综合Inode(文件的索引节点(inode)数)，MTime(修改时间)和 Size来精准的进行判断，避开UNIX记录MTime只能精确到秒的问题。 服务器集群使用，可取后两个参数。使用ETags减少Web应用带宽和负载。

**15、使AJAX可缓存**

利用时间戳，更精巧的实现响应可缓存与服务器数据同步更新。

**16、尽早刷新输出缓冲**

尤其对于css，js文件的并行下载更有意义

**17、使用GET来完成AJAX请求**

当使用XMLHttpRequest时，浏览器中的POST方法是一个“两步走”的过程：首先发送文件头，然后才发送数据。在url小于2K时使用GET获取数据时更加有意义。

**18、延迟加载**

确定页面运行正常后，再加载脚本来实现如拖放和动画，或者是隐藏部分的内容以及折叠内容等。

**19、预加载**

关注下无条件加载，有条件加载和有预期的加载。

**20、减少DOM元素个数**

使用更适合或者在语意是更贴切的标签，要考虑大量DOM元素中循环的性能开销。

**21、根据域名划分页面内容**

很显然， 是最大限度地实现平行下载

**22、尽量减少iframe的个数**

考虑即使内容为空，加载也需要时间，会阻止页面加载，没有语意，注意iframe相对于其他DOM元素高出1-2个数量级的开销，它会在典型方式下阻塞onload事件，IE和Firefox中主页面样式表会阻塞它的下载。

**23、避免404**

HTTP请求时间消耗是很大的，有些站点把404错误响应页面改为“你是不是要找\*\*\*”，这虽然改进了用户体验但是同样也会浪费服务器资源（如数 据库等）。最糟糕的情况是指向外部 JavaScript的链接出现问题并返回404代码。首先，这种加载会破坏并行加载；其次浏览器会把试图在返回的404响应内容中找到可能有用的部分当 作JavaScript代码来执行。

**24、减少Cookie的大小**

去除不必要的coockie  
使coockie体积尽量小以减少对用户响应的影响  
注意在适应级别的域名上设置coockie以便使子域名不受影响  
设置合理的过期时间。较早地Expire时间和不要过早去清除coockie，都会改善用户的响应时间。

**25、使用无cookie的域**

确定对于静态内容的请求是无coockie的请求。创建一个子域名并用他来存放所有静态内容。

**26、减少DOM访问**

缓存已经访问过的有关元素  
线下更新完节点之后再将它们添加到文档树中  
避免使用JavaScript来修改页面布局

**27、开发智能事件处理程序**

有时候我们会感觉到页面反应迟钝，这是因为DOM树元素中附加了过多的事件句柄并且些事件句病被频繁地触发。这就是为什么说使用event delegation（事件代理）是一种好方法了。如果你在一个div中有10个按钮，你只需要在div上附加一次事件句柄就可以了，而不用去为每一个按 钮增加一个句柄。事件冒泡时你可以捕捉到事件并判断出是哪个事件发出的。  
你同样也不用为了操作DOM树而等待onload事件的发生。你需要做的就是等待树结构中你要访问的元素出现。你也不用等待所有图像都加载完毕。  
你可能会希望用DOMContentLoaded事件来代替 事件应用程序中的onAvailable方法。

**28、用<link>代替@import**

在IE中，页面底部@import和使用<link>作用是一样的，因此最好不要使用它。

**29、避免使用滤镜**

完全避免使用AlphaImageLoader的最好方法就是使用PNG8格式来代替，这种格式能在IE中很好地工作。如果你确实需要使用 AlphaImageLoader，请使用下划线\_filter又使之对IE7以上版本的用户无效。

**30、优化图像**

尝试把GIF格式转换成PNG格式，看看是否节省空间。在所有的PNG图片上运行pngcrush（或者其它PNG优化工具）

**31、优化CSS Spirite**

在Spirite中水平排列你的图片，垂直排列会稍稍增加文件大小；  
Spirite中把颜色较近的组合在一起可以降低颜色数，理想状况是低于256色以便适用PNG8格式；  
便于移动，不要在Spirite的图像中间留有较大空隙。这虽然不大会增加文件大小但对于用户代理来说它需要更少的内存来把图片解压为像素地图。 100×100的图片为1万像素，而1000×1000就是100万像素。

**32、不要在HTML中缩放图像——须权衡**

不要为了在HTML中设置长宽而使用比实际需要大的图片。如果你需要：

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

那么你的图片（mycat.jpg）就应该是100×100像素而不是把一个500×500像素的图片缩小使用。这里在下文有更有趣的分析。

**33、favicon.ico要小而且可缓存**

favicon.ico是位于服务器根目录下的一个图片文件。它是必定存在的，因为即使你不关心它是否有用，浏览器也会对它发出请求，因此最好不要 返回一 个404 Not Found的响应。由于是在同一台服务器上，它每被请求一次coockie就会被发送一次。这个图片文件还会影响下载顺序，例如在IE中当你在 onload中请求额外的文件时，favicon会在这些额外内容被加载前下载。

因此，为了减少favicon.ico带来的弊端，要做到：  
文件尽量地小，最好小于1K  
在适当的时候（也就是你不要打算再换favicon.ico的时候，因为更换新文件时不能对它进行重命名）为它设置Expires文件头。你可以很安全地 把Expires文件头设置为未来的几个月。你可以通过核对当前favicon.ico的上次编辑时间来作出判断。  
Imagemagick可以帮你创建小巧的favicon。

**34、保持单个内容小于25K**

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。由于单纯gizp压缩可能达不要求，因此精简文件就显得十分重 要。

**35、打包组件成复合文本**

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请求中取得多个组件（切记：HTTP请求是很奢侈的）。当你使用这条规 则时，首先要确定用户代理是否支持（iPhone就不支持）。

**二、Yahoo军规之外的场景？**

**1、 使用json作为数据的交换格式**  
Json在浏览器解析的效率至少高于XML一个数量级，高级浏览器中内置的有生成和解析json的方法，IE6中要用额外的方法（http://json.org）,不要用eval，容易引发性能和安全问题。  
**2、 尽可能对images和table设定宽高值**

针对Yslow的不要在HTML中缩放图像——第33条，有人会误解为不要对图片加宽高值，其实这条建议本身的意思是不要为了获取一个特定大小的图片，而去强行通过设置宽高值拉伸或者压缩一个既有的图片。建议是另存一张符合尺寸的图片替代。  
对图片和table是设定宽高，是考虑到如果浏览器能立刻知道图片或者tables的宽高，它就能够直接呈现页面而不需要通过计算元素大小后重绘，而且即便是图片损毁而没有展现，也不会进而破坏了页面本来的布局。  
有一些应用场景需要注意：

* a、批量图片，图片源可控同时页面图片宽高值不可变，比如数据库有100张100\*100的图片要在页面中全部展示，那么建议是都写上

<img width=”100″ height=”120″ src=”" alt=”" />

* b、批量图片，图片源不可控同时页面图片宽高值不可变，比如数据库有100张图片，而已知图片有的尺寸是97\*100，有的 100\*105，而又不可能去一张张修改另存。这里视情况而定，根据图片尺寸与要求尺寸的偏离度，在保证图片不拉伸变形同时不影响页面布局的情况下，可以 对图片单独设定宽度100，同时对其包裹的容器设定100\*100的宽高来隐藏多出来的部分，注意不能同时设置宽高以防止变形。
* c、批量图片，图片源不可控，页面图片宽高值不定，比如数据库有100张各种尺寸偏差较大的，此时可不对图片设置宽高；  
  其他情况不一一罗列，原则是在最大程度保证图片不变形与图片最大面积展现的前提下，尽可能为图片设置宽高值，总之就是权衡。  
  Tables的宽高值同图片，尽可能设置。

**3、 拆离内容块**

尽量用div取代tables，或者将tables打破成嵌套层次深的结构；

避免用这样的嵌套

<table>

<table>

<table>

...

</table>

</table>

</table>

采用下面的或者div重构：

<table></table>

<table></table>

<table></table>

**4、 高效的CSS书写规则**

众所周知，CSS选择符是从右向左进行匹配的。  
通常一个图片列表的的小模块

<div id="box">

<div class="hd">

<h3>我的旅途</h3>

</div>

<div class="bd">

<h4>旅途1</h4>

<ul id="pics">

<li>

<a href="#pic" title=""><img src="" alt="" /> </a>

<p>这是在<strong>图片1</strong></p>

</li>

</ul>

</div>

</div>

为了代码上缩进后内层的整洁性，我们html有可能这样写之外，更喜欢看这样的css写法：

.box{border:1px solid #ccc }

.box .hd{border-bottom:1px solid #ccc }

.box .hd h3{color:#515151}

.box .bd{color:#404040 }

.box .bd ul{margin-left:10px}

.box .bd ul li{border-bottom:1px dashed #f1f1f1}

.box .bd ul li a{text-decoration:none}

.box .bd ul li a:hover{text-decoration:underline}

.box .bd ul li a img{border:1px solid #ccc}

.box .bd ul li p{text-align:left;}

.box .bd ul li p strong{color:#ff6600}

其实写到这里，问题已经显而易见了。深达五层抑或六层的嵌套，同时右边的选择符都是采用标签，在满足我们视觉平整与代码结构系统化的时候，付出的是性能的代价。

不做进一步的代码书写方式的探讨，受个人习惯与应用场景影响。这里对css选择符按照开销从小到大的顺序梳理一下：

* ID选择符 #box
* 类选择符 .box
* 类型选择符 div
* 相邻兄弟选择符 h4 + #pics
* 子选择符 #pics li
* 后代选择符 .box a{}
* 通配选择符 \*
* 属性选择符 [href=”#pic”]
* 伪类和伪元素 a:hover

**参考《高性能网站建设-进阶指南》，有如下建议：**

* 避免使用统配规则；
* 不要限定ID选择符；
* 不要限定类选择符；
* 让规则越具体越好；
* 避免使用后代选择符；
* 避免使用标签-子选择符；
* 质疑子选择符的所有用途；
* 依靠继承；

还要注意到，即便是页面加载后，当页面被触发引起回流（reflow）的时候，低效的选择符依然会引发更高的开销，显然这对于用户是不佳的体验。

**4、Javascript 的性能优化点**

* *a、慎用Eval*

谨记：有“eval”的代码比没有“eval”的代码要慢上 100 倍以上。主要原因是：JavaScript 代码在执行前会进行类似“预编译”的操作：首先会创建一个当前执行环境下的活动对象，并将那些用 var 申明的变量设置为活动对象的属性，但是此时这些变量的赋值都是 undefined，并将那些以 function 定义的函数也添加为活动对象的属性，而且它们的值正是函数的定义。但是，如果你使用了“eval”，则“eval”中的代码（实际上为字符串）无法预先识 别其上下文，无法被提前解析和优化，即无法进行预编译的操作。所以，其性能也会大幅度降低。

* *b、推荐尽量使用局部变量*

JavaScript 代码解释执行，在进入函数内部时，它会预先分析当前的变量，并将这些变量归入不同的层级（level），一般情况下：  
局 部变量放入层级 1（浅），全局变量放入层级 2（深）。如果进入“with”或“try – catch”代码块，则会增加新的层级，即将“with”或“catch”里的变量放入最浅层（层 1），并将之前的层级依次加深。变量所在的层越浅，访问（读取或修改）速度越快，尤其是对于大量使用全局变量的函数里面。

* *c、字符串数组方式拼接避免在IE6下的开销*

var tips = 'tip1'+'tip2';

这是我们拼接字符串常用的方式，但是这种方式会有一些临时变量的创建和销毁，影响性能，尤其是在IE6下，所以推荐使用如下方式拼接：

var tip\_array = [],tips;

tip\_array.push('tip1');

tip\_array.push('tip2');

tips = tip\_array.join('');

当然，最新的浏览器（如火狐 Firefox3+，IE8+ 等等）对字符串的拼接做了优化,性能略快于数组的“join”方法。

* 以上仅列出三种常见的优化方法，仅抛砖以引玉石，更多的javascript优化点，比如避免隐式类型转换， 缩小对象访问层级，利用变量优化字符串匹配等大家可以继续深入挖掘；

**5、DOM 操作优化**

首先澄清两个概念——Repaint 和 Reflow：Repaint 也叫 Redraw，它指的是一种不会影响当前 DOM 的结构和布局的一种重绘动作。如下动作会产生 Repaint 动作：

* 不可见到可见（visibility 样式属性）;
* 颜色或图片变化（background, border-color, color 样式属性）;
* 不改变页面元素大小，形状和位置，但改变其外观的变化

Reflow 比起 Repaint 来讲就是一种更加显著的变化了。它主要发生在 DOM 树被操作的时候，任何改变 DOM 的结构和布局都会产生 Reflow。但一个元素的 Reflow 操作发生时，它的所有父元素和子元素都会放生 Reflow，最后 Reflow 必然会导致 Repaint 的产生。举例说明，如下动作会产生 Reflow 动作：

* 浏览器窗口的变化;
* DOM 节点的添加删除操作
* 一些改变页面元素大小，形状和位置的操作的触发

通过 Reflow 和 Repaint 的介绍可知，每次 Reflow 比其 Repaint 会带来更多的资源消耗，因此，我们应该尽量减少 Reflow 的发生，或者将其转化为只会触发 Repaint 操作的代码。

var tipBox = document.createElement('div');

document.body.appendChild('tipBox');//reflow

var tip1 = document.createElement('div');

var tip2 = document.createElement('div');

tipBox.appendChild(tip1);//reflow

tipBox.appendChild(tip2);//reflow

如上的代码，会产生三次reflow，优化后的代码如下：

var tipBox = document.createElement('div');

tip1 = document.createElement('div');

tip2 = document.createElement('div');

tipBox.appendChild(tip1);

tipBox.appendChild(tip2);

document.body.appendChild('tipBox');//reflow

当然还可以利用 display 来减少reflow次数

var tipBox = document.getElementById('tipBox');

tipBox.style.display = 'none';//reflow

tipBox.appendChild(tip1);

tipBox.appendChild(tip2);

tipBox.appendChild(tip3);

tipBox.appendChild(tip4);

tipBox.appendChild(tip5);

tipBox.style.width = 120;

tipBox.style.height = 60;

tipBox.style.display = 'block';//reflow

DOM元素测量属性和方法也会触发reflow,如下：

var tipWidth = tipBox.offsetWidth;//reflow

tipScrollLeft = tipBox.scrollLeft;//reflow

display = window.getComputedStyle(div,'').getPropertyValue('display');//reflow

触发reflow的属性和方法大概有这些：

* + offsetLeft
  + offsetTop
  + offsetHeight
  + offsetWidth
  + scrollTop/Left/Width/Height
  + clientTop/Left/Width/Height
  + getComputedStyle()
  + currentStyle(in IE))

我们可以用临时变量将“offsetWidth”的值缓存起来，这样就不用每次访问“offsetWidth”属性。这种方式在循环里面非常适用，可以极大地提高性能。

如果有批量的样式属性需要修改，建议通过替换className的方式来降低reflow的次数，曾经有这样一个场景：有三个 intput,分别对应下面三个图片和三个内容区域，第二input选中的时候，第二图片显示，其他图片隐藏，第二块内容显示，其他内容隐藏，直接操作 DOM节点的代码如下

var input = [];

pics = [];

contents = [];

......

for(var i = 0;i<3;i++){

input[i].onclick = function(e){

show(pics,i);//reflow两次

show(contents,i);//reflow两次

}

}

function show(target,j){

for(var i = 0,i<3;i++){

target[i].style.display = 'none';//reflow

}

target[j].style.display = 'block';//reflow

}

如果是通过css预先定义元素的隐藏和显示，通过对父级的className进行操纵，将会把reflow的次数减少到1次

.pbox .pic,.pbox content{display:none}

.J\_pbox\_0 .pic0,.J\_pbox\_0 .content0{diplay:block}

.J\_pbox\_1 .pic1,.J\_pbox\_1 .content1{diplay:block}

.J\_pbox\_2 .pic2,.J\_pbox\_2 .content2{diplay:block}

var input = [],

parentBox = document.getELementById('J\_Pbox');

......

for(var i = 0;i<3;i++){

input[i].onclick = function(e){

parentBox.className = 'pbox J\_pbox\_'+i;//reflow一次

}

}

**三、Yahoo军规再度挖掘会怎样？**

在网站性能优化的路上，是不会有终点的，这也是前端工程师永不会妥协的地方。  
想看到更牛P的优化建议么，请移步这里来关注李牧童鞋的分享：

* + 使用combo合并静态资源
  + Bigpipe技术合并动态数据
  + Comet:基于http的服务端推技术
  + 使用DataURI减少图片请求
  + 使用良好的JS,CSS版本管理方案
  + 尝试仅作必要的JS更新
  + 利用本地存储做缓存
  + 关于最小化HTML
  + 进一步讨论Gzip
  + 进一步讨论域名划分
  + 打开keep-alive,重用HTTP连接
  + 使用JSON进行数据交换
  + 保障页面可交互性
  + 缩短最快可交互时间
  + 异步无阻脚本下载
  + 优化内存使用,防止内存泄露
  + 高效的JavaScript
  + 第三方代码性能问题
  + Inline脚本不要与CSS穿插使用
  + 使用高效的CSS选择器
  + 进一步讨论及早Flush
  + 关于视觉和心理学