页面性能优化

1.减少http请求
2.添加Expires头来设置缓存
3.使用gzip编码压缩组件
4.CSS放在页面顶部，javascript放在页面底部
5.减少页面重绘与回流：减少dom操作
6.减少DNS查找
7.避免重定向
8.精简和混淆JS代码
9.CSS 选择器优化
10.javascript语法优化
11.使用内容分发网络
12.使用require.js

1.减少http请求

1.将CSS、JavaScript合并成一个文件，这样浏览器就只需要一次请求。
2.将多张图片合并成一张。
(1) 图片地图
允许你在一个图片上关联多个URL。目标URL的选择取决于用户单击了图片上的哪个位置。
比如导航栏，点击不同图标的时候将分别打开一个新的窗口。
(2) CSS精灵
合并图片，然后通过CSS中的background-image和background-position属性，按照需要显示这幅图像中的不同部分。

减少http请求数的优点:
(1) 减少DNS请求所耗费的时间。
(2) 减少服务器压力。
(3) 减少http请求头。

2.添加Expires头来设置缓存

Expires存储的是一个用来控制缓存失效的日期。当浏览器看到响应中有一个Expires头时，它会和相应的组件一起保存到其缓存中，只要组件没有过期，浏览器就会使用缓存版本而不会进行任何的HTTP请求。Expires设置的日期格式必须为GMT（格林尼治标准时间）。

3.使用gzip编码压缩组件

gzip编码：gzip是GUNzip的缩写，是使用无损压缩算法的一种，最早是用于Unix系统的文件压缩，凭借着良好的压缩效率，现在已经成为Web上使用最为普遍的数据压缩格式。

基于文本的资源如html，js，css，xml都适用于压缩。然而对于图片而言，却不应该对图片进行压缩，因为图片本身是已经被压缩过了，如果再进行gzip压缩，有可能得到的结果是和图片本身大小相差不大或更大，这样就浪费了服务器的CPU资源来做无用功了。

优点：压缩组件可以减少Http响应时间，提升传输效率。
缺点：服务器要通过花费额外的CPU周期来完成压缩，客户端要对压缩文件进行解压缩。

4.CSS放在页面顶部，javascript放在页面底部

1.CSS样式表放在页面顶部

浏览器会在下载完成全部CSS之后才对整个页面进行渲染，因此最好的做法是将CSS放在页面最上面，让浏览器尽快下载CSS。如果将 CSS放在其他地方比如 BODY中，则浏览器有可能还未下载和解析到 CSS就已经开始渲染页面了，这就导致页面由无 CSS状态跳转到 CSS状态，用户体验比较糟糕，所以可以考虑将CSS放在HEAD中。

2.javascript脚本放在页面底部

Javascript则相反，浏览器在加载javascript后立即执行，有可能会阻塞整个页面，造成页面显示缓慢，因此javascript最好放在页面最下面。

3.使用外置的js和css

使用外部文件会比内联产生较快的访问速度，这是由于外部js和css有机会被浏览器缓存起来，当再次请求相同的js或css的时候，浏览器将不会发出http请求，而是使用缓存的组件，减少了总体需要下载文件的大小。

5.减少页面重绘与回流：减少dom操作

dom操作是非常耗时的，所以页面中要尽可能的减少dom的操作。

```
for (var i=0; i < items.length; i++){
    var item = document.createElement("li");
    item.appendChild(document.createTextNode("Option " + i);
    list.appendChild(item);
}
```

优化：先将list对象缓存，然后一次性添加。

```
var fragment = document.createDocumentFragment();
for (var i=0; i < items.length; i++){
    var item = document.createElement("li");
    item.appendChild(document.createTextNode("Option " + i);
    fragment.appendChild(item);
}
list.appendChild(fragment);
```

document.createDocumentFragment()

创建了一个文档碎片，之后可以把所有的新节点附加其上，然后把文档碎片的内容一次性添加到document中。对于循环批量操作页面的DOM有很大用处，节约使用DOM。

6.减少DNS查找

DNS(Domain Name System)： 负责将域名URL转化为服务器主机IP。

DNS查找流程：首先查看浏览器缓存是否存在，不存在则访问本机DNS缓存，再不存在则访问本地DNS服务器。所以DNS也是开销，通常浏览器查找一个给定URL的IP地址要花费20-120ms，在DNS查找完成前，浏览器不能从host那里下载任何东西。

TTL(Time To Live)：表示查找返回的DNS记录包含的一个存活时间，过期则这个DNS记录将被抛弃。

优化方法

当客户端的DNS缓存为空时，DNS查找的数量与Web页面中唯一主机名的数量相等。所以减少唯一主机名的数量就可以减少DNS查找的数量。

7.避免重定向

重定向用于将用户从一个URL重新路由到另一个URL。

当页面发生了重定向，就会延迟整个HTML文档的传输。在HTML文档到达之前，页面中不会呈现任何东西，也没有任何组件会被下载。

常用重定向的类型

301：永久重定向，主要用于当网站的域名发生变更之后，告诉搜索引擎域名已经变更了，应该把旧域名的的数据和链接数转移到新域名下，从而不会让网站的排名因域名变更而受到影响。

302：临时重定向，主要实现post请求后告知浏览器转移到新的URL。

304：Not Modified，主要用于当浏览器在其缓存中保留了组件的一个副本，同时组件已经过期了，这是浏览器就会生成一个条件GET请求，如果服务器的组件并没有修改过，则会返回304状态码，同时不携带主体，告知浏览器可以重用这个副本，减少响应大小。

优化方法

1. 使用Referer日志来跟踪内部流量

当拥有一个门户主页的时候，同时想对用户离开主页后的流量进行跟踪，这时可以使用重定向。以yahoo.com为例，主页新闻的链接主机名是http://hsrd.yahoo.com/，后面跟着识别的参数，点击后再产生一个301重定向，这样就记录了离开门户主页后的流量去向。

我们知道重定向是如何损伤性能的，为了实现更好的效率，可以使用Referer日志来跟踪内部流量去向。每个HTTP请求都有一个Referer表示原始请求页(除了从书签打开或直接键入URL等操作)，记录下每个请求的Referer，就避免了向用户发送重定向，从而改善了响应时间。

2. 使用信标来跟踪出站流量

有时链接可能将用户带离你的网站，在这种情况下，使用Referer就不太现实了。

同样也可以使用重定向来解决跟踪出站流量问题。以百度搜索为例，百度通过将每个链接包装到一个302重定向来解决跟踪的问题，例如搜索关键字“跟踪出站流量”，搜索结果的第一个URL为http://www.baidu.com/link?url=后面跟着一连串字符，即使搜索结果并没有变，但这个字符串是动态改变的，我认为这里的搜索连接URL好像没有改变的需要，不知道这里起到怎样的作用？

除了重定向外，我们还可以选择使用信标(beacon)——一个HTTP请求，其URL中包含有跟踪信息。跟踪信息可以从信标Web服务器的访问日记中提取出来，信标通常是一个1px*1px的透明图片，不过204响应更优秀，因为它更小，从来不被缓存，而且绝不会改变浏览器的状态。

8.精简和混淆JS代码

精简：

从javascript代码中移除所有的注释以及不必要的空白字符(空格，换行和制表符)，减少javascript文件的大小。

混淆：

和精简一样，会从javascript代码中移除注释和空白，另外也会改写代码。作为改写的一部分，函数和变量的名字将被转换为更短的字符串，所以进一步减少了javascript文件的大小。

混淆的缺点

1. 缺陷：混淆过程本身很有可能引入错误。
2. 维护：由于混淆会改变javascript符号，因此需要对任何不能改变的符号进行标记，防止混淆器修改它们。
3. 调试：经过混淆的代码很难阅读，这使得在产品环境中更加难以调试。

对精简和混淆的抉择

我们知道启用gzip压缩能减少组件的传送大小，压缩后精简和混淆的差别会进一步减少，综合考虑混淆可能带来的额外的风险，所以优先考虑使用精简。不过，如果对于性能的极致追求，可以使用混淆，但要做足测试，确保混淆不会带来其他的问题。

9.CSS 选择器优化

CSS选择器匹配规则

CSS选择器是从右到左进行规则匹配。只要当前选择符的左边还有其他选择符，样式系统就会继续向左移动，直到找到和规则匹配的选择符，或者因为不匹配而退出。最右边的选择符称之为关键选择器。

CSS 选择器的执行效率从高到低排序：

1.id选择器（#myid）
2.类选择器（.myclassname）
3.标签选择器（div,h1,p）
4.相邻选择器（h1+p）
5.子选择器（ul < li）
6.后代选择器（li a）
7.通配符选择器（*）
8.属性选择器（a[rel="external"]）
9.伪类选择器（a:hover, li:nth-child）

优化方法：

1、避免使用通用选择器
2、用 id 选择器代替 class 选择器
3、用 class 选择器代替标签选择器
4、避免使用多层标签选择器。使用 class 选择器替换，减少css查找
5、用子选择器代替后代选择器

10.javascript语法优化

http://www.cnblogs.com/Walker-lyl/p/5676389.html

避免使用 eval 和 Function

每次 eval 或Function 构造函数作用于字符串表示的源代码时，脚本引擎都需要将源代码转换成可执行代码。这是很消耗资源的操作 —— 通常比简单的函数调用慢 100倍以上。

eval 函数效率特别低，由于事先无法知晓传给 eval 的字符串中的内容，eval在其上下文中解释要处理的代码，也就是说编译器无法优化上下文，因此只能有浏览器在运行时解释代码。这对性能影响很大。

Function 构造函数比 eval略好，因为使用此代码不会影响周围代码 ;但其速度仍很慢。

此外，使用 eval和 Function也不利于Javascript 压缩工具执行压缩。

减少作用域链查找

前文谈到了作用域链查找问题，这一点在循环中是尤其需要注意的问题。如果在循环中需要访问非本作用域下的变量时请在遍历之前用局部变量缓存该变量，并在遍历结束后再重写那个变量，这一点对全局变量尤其重要，因为全局变量处于作用域链的最顶端，访问时的查找次数是最多的。

慎用 with

with(obj){ p = 1}; 代码块的行为实际上是修改了代码块中的执行环境 ，将obj放在了其作用域链的最前端，在 with 代码块中访问非局部变量是都是先从 obj上开始查找，如果没有再依次按作用域链向上查找，因此使用 with相当于增加了作用域链长度。而每次查找作用域链都是要消耗时间的，过长的作用域链会导致查找性能下降。

因此，除非你能肯定在 with代码中只访问 obj 中的属性，否则慎用 with，替代的可以使用局部变量缓存需要访问的属性。

数据访问

Javascript中的数据访问包括直接量（字符串、正则表达式 ）、变量、对象属性以及数组，其中对直接量和局部变量的访问是最快的，对对象属性以及数组的访问需要更大的开销。当出现以下情况时，建议将数据放入局部变量：

a. 对任何对象属性的访问超过 1次
b. 对任何数组成员的访问次数超过 1次

另外，还应当尽可能的减少对对象以及数组深度查找。

字符串拼接

在 Javascript中使用”+”号来拼接字符串效率是比较低的，因为每次运行都会开辟新的内存并生成新的字符串变量，然后将拼接结果赋值给新变量。与之相比更为高效的做法是使用数组的 join方法，即将需要拼接的字符串放在数组中最后调用其 join方法得到结果。不过由于使用数组也有一定的开销，因此当需要拼接的字符串较多的时候可以考虑用此方法。

11.使用内容分发网络

服务器离用户越近，HTTP请求的响应时间将更短。

内容分发网络 CDN（Content Deliver Network）是一组分布在多个不同地理位置的Web服务器，通过将网站的资源发布到最接近用户的网络”边缘“，供用户就近取得所需内容。CDN可以看作一种缓存代理，主要用于对静态资源（如图片，css，js等）的缓存。

12.使用require.js

使用require.js。 require.js是一个JavaScript模块加载器。异步加载，在加载模块的时候不会影响后续代码的执行，避免网页失去响应。 按需加载，保证js文件只在被需要的时候加载。