

Web前端性能优化——如何提高页面加载速度

前言：

在同样的网络环境下，两个同样能满足你的需求的网站，一个“Duang”的一下就加载出来了，一个纠结了半天才出来，你会选择哪个？研究表明：用户最满意的打开网页时间是2-5秒，如果等待超过10秒，99%的用户会关闭这个网页。也许这样讲，各位还不会有太多感触，接下来我列举一组数据：Google网站访问速度每慢400ms就导致用户搜索请求下降0.59%；Amazon每增加100ms网站延迟将导致收入下降1%；雅虎如果有400ms延迟会导致流量下降5-9%。网站的加载速度严重影响了用户体验，也决定了这个网站的生死存亡。可能有人会说：网站的性能是后端工程师的事情，与前端并无多大关系。我只能说，too young too simple。事实上，只有10%~20%的最终用户响应时间是用在从Web服务器获取HTML文档并传送到浏览器的，那剩余的时间去哪儿了？来瞄一下**性能黄金法则**：只有10%~20%的最终用户响应时间花在了下载HTML文档上。其余的80%~90%时间花在了下载页面中的所有组件上。接下来我们将研究一下前端攻城狮如何提高页面的加载速度。

一、减少HTTP请求

上面说到80%~90%时间花在了下载页面中的所有组件进行的HTTP请求上。因此，改善响应时间最简单的途径就是减少HTTP请求的数量。

图片地图：

假设导航栏上有五幅图片，点击每张图片都会进入一个链接，这样五张导航的图片在加载时会产生5个HTTP请求。然而，使用一个图片地图可以提高效率，这样就只需要一个HTTP请求。



服务器端图片地图：将所有点击提交到同一个url，同时提交用户点击的x、y坐标，服务器端根据坐标映射响应

客户端图片地图：直接将点击映射到操作

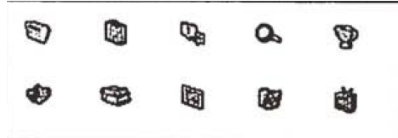
```


<map name="planetmap" id="planetmap">
  <area shape="rect" coords="180,139,14" href="venus.html" alt="Venus" />
  <area shape="rect" coords="129,161,10" href="mercur.html" alt="Mercury" />
  <area shape="rect" coords="0,0,110,260" href="sun.html" alt="Sun" />
  <area shape="rect" coords="140,0,110,260" href="star.html" alt="Sun" />
</map>
```

使用图片地图的缺点：指定坐标区域时，矩形或圆形比较容易指定，而其它形状手工指定比较难

CSS Sprites

CSS Sprites直译过来就是CSS精灵，但是这种翻译显然是不够的，其实就是通过将多个图片融合到一副图里面，然后通过CSS的一些技术布局到网页上。特别是图片特别多的网站，如果能用css sprites降低图片数量，带来的将是速度的提升。



```
<div>
  <span id="image1" class="nav"></span>
  <span id="image2" class="nav"></span>
  <span id="image3" class="nav"></span>
  <span id="image4" class="nav"></span>
  <span id="image5" class="nav"></span>
</div>
```

```
.nav {
  width: 50px;
  height: 50px;
  display: inline-block;
  border: 1px solid #000;
  background-image: url('E:/1.png');
}

#image1 {
  background-position: 0 0;
}

#image2 {
  background-position: -95px 0;
}

#image3 {
  background-position: -185px 0;
}

#image4 {
```

```
background-position: -275px 0;
}
#image5 {
background-position: -366px -3px;
}
```



运行结果：



PS：使用CSS Sprites还有可能降低下载量，可能大家会认为合并后的图片会比分离图片的总和要大，因为还有可能会附加空白区域。实际上，合并后的图片会比分离的图片总和要小，因为它降低了图片自身的开销，譬如颜色表、格式信息等。

字体图标

在可以大量使用字体图标的地方我们可以尽可能使用字体图标，字体图标可以减少很多图片的使用，从而减少http请求，字体图标还可以通过CSS来设置颜色、大小等样式，何乐而不为。

合并脚本 和样式表

将多个样式表或者脚本文件合并到一个文件中，可以减少HTTP请求的数量从而缩短效应时间。

然而合并所有文件对许多人尤其是编写模块化代码的人来说是不能忍的，而且合并所有的样式文件或者脚本文件可能会导致在一个页面加载时加载了多于自己所需要的样式或者脚本，对于只访问该网站一个（或几个）页面的人来说反而增加了下载量，所以大家应该自己权衡利弊。

二、使用CDN

如果应用程序web服务器离用户更近，那么一个HTTP请求的响应时间将缩短。另一方面，如果组件web服务器离用户更近，则多个HTTP请求的响应时间将缩短。

CDN（内容发布网络）是一组分布在多个不同地理位置的Web服务器，用于更加有效地向用户发布内容。在优化性能时，向特定用户发布内容的服务器的选择基于对网络拥堵的测量。例如，CDN可能选择网络阶跃数最小的服务器，或者具有最短响应时间的服务器。

CDN还可以进行数据备份、扩展存储能力，进行缓存，同时有助于缓和Web流量峰值压力。

CDN的缺点：

- 1、响应时间可能会受到其他网站流量的影响。CDN服务提供商在其所有客户之间共享Web服务器组。
- 2、如果CDN服务质量下降了，那么你的工作质量也将下降
- 3、无法直接控制组件服务器

三、添加Expires头

页面的初次访问者会进行很多HTTP请求，但是通过使用一个长久的Expires头，可以使这些组件被缓存，下次访问的时候，就可以减少不必要的HTTP请求，从而提高加载速度。

Web服务器通过Expires头告诉客户端可以使用一个组件的当前副本，直到指定的时间为止。例如：

Expires: Fri, 18 Mar 2016 07:41:53 GMT

Expires缺点：它要求服务器和客户端时钟严格同步；过期日期需要经常检查

HTTP1.1中引入Cache-Control来克服Expires头的限制，使用max-age指定组件被缓存多久。

Cache-Control：max-age=12345600

若同时制定Cache-Control和Expires，则max-age将覆盖Expires头

四、压缩组件

从HTTP1.1开始，Web客户端可以通过HTTP请求中的Accept-Encoding头来表示对压缩的支持

Accept-Encoding: gzip,deflate

如果Web服务器看到请求中有这个头，就会使用客户端列出来的方法中的一种来进行压缩。Web服务器通过响应中的Content-Encoding来通知Web客户端。

Content-Encoding: gzip

代理缓存

当浏览器通过代理来发送请求时，情况会不一样。假设针对某个URL发送到代理的第一个请求来自于一个不支持gzip的浏览器。这是代理的第一个请求，缓存为空。代理将请求转发给服务器。此时响应是未压缩的，代理缓存同时发送给浏览器。现在，假设到达代理的请求是同一个url，来自于一个支持gzip的浏览器。代理会使用缓存中未压缩的内容进行响应，从而失去了压缩的机会。相反，如果第一个浏览器支持gzip，第二个不支持，你们代理缓存中的压缩版本将会提供给后续的浏览器，而不管它们是否支持gzip。

解决办法：在web服务器的响应中添加vary头Web服务器可以告诉代理根据一个或多个请求头来改变缓存的响应。因为压缩的决定是基于Accept-Encoding请求头的，因此需要在vary响应头中包含Accept-Encoding。

vary: Accept-Encoding

五、将样式表放在头部

首先说明一下，将样式表放在头部对于实际页面加载的时间并不能造成太大影响，但是这会减少页面首屏出现的时间，使页面内容逐步呈现，改善用户体验，防止“白屏”。

我们总是希望页面能够尽快显示内容，为用户提供可视化的回馈，这对网速慢的用户来说是很重要的。

将样式表放在文档底部会阻止浏览器中的内容逐步出现。为了避免当样式变化时重绘页面元素，浏览器会阻塞内容逐步呈现，造成“白屏”。这源自浏览器的行为：如果样式表仍在加载，构建呈现树就是一种浪费，因为所有样式表加载解析完毕之前务虚会之任何东西

六、将脚本放在底部

更样式表相同，脚本放在底部对于实际页面加载的时间并不能造成太大影响，但是这会减少页面首屏出现的时间，使页面内容逐步呈现。js的下载和执行会阻塞Dom树的构建（严谨地说是中断了Dom树的更新），所以script标签放在首屏范围内的HTML代码段里会截断首屏的内容。

下载脚本时并行下载是被禁用的——即使使用了不同的主机名，也不会启用其他的下载。因为脚本可能修改页面内容，因此浏览器会等待；另外，也是为了保证脚本能够按照正确的顺序执行，因为后面的脚本可能与前面的脚本存在依赖关系，不按照顺序执行可能会产生错误。

七、避免CSS表达式

CSS表达式是动态设置CSS属性的一种强大并且危险的方式，它受到了IE5以及之后版本、IE8之前版本的支持。

```
p {
  width: expression(func(),document.body.clientWidth > 400 ? "400px" : "auto");
  height: 80px;
  border: 1px solid #f00;
}
```



```
<p><span></span></p>
<p><span></span></p>
<p><span></span></p>
<p><span></span></p>
<p><span></span></p>
<script>
  var n = 0;
  function func() {
    n++;
    // alert();
    console.log(n);
  }
</script>
```



3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045

鼠标移动了几次，函数的运行次数轻而易举的达到了几千次，危险性显而易见。

如何解决：

一次性表达式：

```
p {
  width: expression(func(this));
  height: 80px;
  border: 1px solid #f00;
}
```



```
<p><span></span></p>
<p><span></span></p>
<p><span></span></p>
<p><span></span></p>
<p><span></span></p>
<script>
  var n = 0;
  function func(elem) {
    n++;
    elem.style.width = document.body.clientWidth > 400 ? '400px' : "auto";
    console.log(n);
  }
</script>
```



事件处理机制

用js事件处理机制来动态改变元素的样式，使函数运行次数在可控范围之内。

八、使用外部的JavaScript和CSS

内联脚本或者样式可以减少HTTP请求，按理来说可以提高页面加载的速度。然而在实际情况中，当脚本或者样式是从外部引入的文件，浏览器就有可能缓存它们，从而在以后加载的时候能够直接使用缓存，而HTML文档的大小减小，从而提高加载速度。

影响因素：

- 1、每个用户产生的页面浏览量越少，内联脚本和样式的论据越强势。譬如一个用户每个月只访问你的网站一两次，那么这种情况下内联将会更好。而如果该用户能够产生很多页面浏览量，那么缓存的样式和脚本将会极大减少下载的时间，提交页面加载速度。
- 2、如果你的网站不同的页面之间使用的组件大致相同，那么使用外部文件可以提高这些组件的重用率。

加载后下载

有时候我们希望内联样式和脚本，但又可以为接下来的页面提供外部文件。那么我们可以在页面加载完成时动态加载外部组件，以使用户接下来的访问。



```
1  function doOnLoad() {
2      setTimeout("downloadFile()",1000);
3  }
4
5  window.onload = doOnLoad;
6
7  function downloadFile() {
8      downloadCss("http://abc.com/css/a.css");
9      downloadJS("http://abc.com/js/a.js");
10 }
11
12 function downloadCss(url) {
13     var ele = document.createElement('link');
14     ele.rel = "stylesheet";
15     ele.type = "text/css";
16     ele.href = url;
17
18     document.body.appendChild(ele);
19 }
20
21 function downloadJS(url) {
22     var ele = document.createElement('script');
23     ele.src = url;
24     document.body.appendChild(ele);
25 }
```



在该页面中，JavaScript和CSS被加载两次（内联和外部）。要使其正常工作，必须处理双重定义。将这些组件放到一个不可见的IFrame中是一个比较好的解决方式。

九、减少DNS查找

当我们在浏览器的地址栏输入网址（譬如：www.linux178.com），然后回车，回车这一瞬间到底看到了页面到底发生了什么呢？

域名解析 --> 发起TCP的3次握手 --> 建立TCP连接后发起http请求 --> 服务器响应http请求，浏览器得到html代码 --> 浏览器解析html代码，并请求html代码中的资源（如js、css、图片等） --> 浏览器对页面进行渲染呈现给用户

域名解析是页面加载的第一步，那么域名是如何解析的呢？以Chrome为例：

1. Chrome浏览器 会首先搜索浏览器自身的DNS缓存（缓存时间比较短，大概只有1分钟，且只能容纳1000条缓存），看自身的缓存中是否有www.linux178.com 对应的条目，而且没有过期，如果有且没有过期则解析到此结束。

注：我们怎么查看Chrome自身的缓存？可以使用 `chrome://net-internals/#dns` 来进行查看

2. 如果浏览器自身的缓存里面没有找到对应的条目，那么Chrome会搜索操作系统自身的DNS缓存,如果找到且没有过期则停止搜索解析到此结束。

注：怎么查看操作系统自身的DNS缓存，以Windows系统为例，可以在命令行下使用 `ipconfig /displaydns` 来进行查看

3. 如果在Windows系统的DNS缓存也没有找到，那么尝试读取hosts文件（位于C:\Windows\System32\drivers\etc），看看这里面有没有该域名对应的IP地址，如果有则解析成功。

4. 如果在hosts文件中也没有找到对应的条目，浏览器就会发起一个DNS的系统调用，就会向本地配置的首选DNS服务器（一般是电信运营商提供的，也可以使用像Google提供的DNS服务器）发起域名解析请求（通过的是UDP协议向DNS的53端口发起请求，这个请求是递归的请求，也就是运营商的DNS服务器必须得提供给我们该域名的IP地址），运营商的DNS服务器首先查找自身的缓存，找到对应的条目，且没有过期，则解析成功。如果没有找到对应的条目，则有运营商的DNS代我们的浏览器发起迭代DNS解析请求，它首先是会找根域的DNS的IP地址（这个DNS服务器都内置13台根域的DNS的IP地址），找打根域的DNS地址，就会向其发起请求（请问www.linux178.com这个域名的IP地址是多少啊？），根域发现这是一个顶级域com域的一个域名，于是就告诉运营商的DNS

我不知道这个域名的IP地址，但是我知道com域的IP地址，你去找它去，于是运营商的DNS就得到了com域的IP地址，又向com域的IP地址发起了请求（请问www.linux178.com这个域名的IP地址是多少？），com域这台服务器告诉运营商的DNS我不知道www.linux178.com这个域名的IP地址，但是我知道linux178.com这个域的DNS地址，你去找它去，于是运营商的DNS又向linux178.com这个域名的DNS地址（这个一般就是由域名注册商提供的，像万网，新网等）发起请求（请问www.linux178.com这个域名的IP地址是多少？），这个时候linux178.com域的DNS服务器一查，诶，果真在我这里，于是就把找到的结果发送给运营商的DNS服务器，这个时候运营商的DNS服务器就拿到了www.linux178.com这个域名对应的IP地址，并返回给Windows系统内核，内核又把结果返回给浏览器，终于浏览器拿到了www.linux178.com对应的IP地址，该进行一步的动作了。

注：一般情况下是不会进行以下步骤的

如果经过以上的4个步骤，还没有解析成功，那么会进行如下步骤：

5. 操作系统就会查找NetBIOS name Cache（NetBIOS名称缓存，就存在客户端电脑中的），那这个缓存有什么东西呢？凡是最近一段时间内和我成功通讯的计算机的计算机名和Ip地址，就都会存在这个缓存里面。什么情况下该步能解析成功呢？就是该名称正好是几分钟前和我成功通过，那么这一步就可以成功解析。

6. 如果第5步也没有成功，那会查询WINS 服务器（是NETBIOS名称和IP地址对应的服务器）

7. 如果第6步也没有查询成功，那么客户端就要进行广播查找

8. 如果第7步也没有成功，那么客户端就读取LMHOSTS文件（和HOSTS文件同一个目录下，写法也一样）

如果第八步还没有解析成功，那么就宣告这次解析失败，那就无法跟目标计算机进行通信。只要这八步中有一步可以解析成功，那就可以成功和目标计算机进行通信。

DNS也是开销，通常浏览器查找一个给定域名的IP地址要花费20~120毫秒，在完成域名解析之前，浏览器不能从服务器加载到任何东西。那么如何减少域名解析时间，加快页面加载速度呢？

当客户端DNS缓存（浏览器和操作系统）缓存为空时，DNS查找的数量与要加载的Web页面中唯一主机名的数量相同，包括页面URL、脚本、样式表、图片、Flash对象等的主机名。减少主机名的数量就可以减少DNS查找的数量。

减少唯一主机名的数量会潜在减少页面中并行下载的数量（HTTP 1.1规范建议从每个主机名并行下载两个组件，但实际上可以多个），这样减少主机名和并行下载的方案会产生矛盾，需要大家自己权衡。建议将组件放到至少两个但不多于4个主机名下，减少DNS查找的同时也允许高度并行下载。

十、精简JavaScript

精简

精简就是从代码中移除不必要的字符以减少文件大小，降低加载的时间。代码精简的时候会移除不必要的空白字符（空格，换行、制表符），这样整个文件的大小就变小了。

混淆

混淆是应用在源代码上的另外一种方式，它会移除注释和空白符，同时它还会改写代码。在混淆的时候，函数和变量名将会被转换成更短的字符串，这时代码会更加精炼同时难以阅读。通常这样做是为了增加对代码进行反向工程的难度，这也同时提高了性能。

缺点：

混淆本身比较复杂，可能会引入错误。

需要对不能改变的符号做标记，防止JavaScript符号（譬如关键字、保留字）被修改。

混淆会使代码难以阅读，这使得在产品环境中调试问题更加困难。

在以上提到了关于用gzip之类的压缩方式来压缩文件，这边说明一下，就算使用gzip等方式来压缩文件，精简代码依然是有必要的。一般来说，压缩产生的节省是高于精简的，在生产环境中，精简和压缩同时使用能够最大限度的获得更多的节省。

CSS的精简

CSS的精简带来的节省一般来说是小于JavaScript精简的，因为CSS中注释和空白相对较少。

除了移除空白、注释之外，CSS可以通过优化来获得更多的节省：

合并相同的类；

移除不使用的类；

使用缩写，譬如



```
.right {
    color: #fff;

    padding-top: 0;

    margin: 0 10px;

    border: 1px solid #111
}
.wrong {
    color: #ffffff;
```



```
padding-top: 0px;



margin-top: 0;
margin-bottom: 0;
margin-left: 10px;
margin-right: 10px;

border-color: #111;
border-width: 1px;
border-style: solid;
}
```



上面.right是正确的的写法，颜色使用缩写，使用0代替0px，合并可以合并的样式。另外，在精简的时候其实样式最后一行的';'也是可以省略的。

来看看精简的例子：

 jquery-2.0.3.js	2016/3/17 14:01	JS 文件	237 KB
 jquery-2.0.3.min.js	2016/3/6 11:44	JS 文件	82 KB

```

/*
(function( window, undefined ) {

// Can't do this because several apps including ASP.NET trace
// the stack via arguments.caller.callee and Firefox dies if
// you try to trace through "use strict" call chains. (#13335)
// Support: Firefox 18+
// "use strict";

var
    // A central reference to the root jQuery(document)
    rootjQuery,

    // The deferred used on DOM ready
    readyList,

    // Support: IE9
    // For `typeof xmlNode.method` instead of `xmlNode.method !== undefined`
    core_strundefined = typeof undefined,

    // Use the correct document accordingly with window argument (sandbox)
    location = window.location,
    document = window.document,
    docElem = document.documentElement,

    // Map over jQuery in case of overwrite
    _jQuery = window.jQuery,

    // Map over the $ in case of overwrite
    _$ = window.$,

    // [[Class]] -> type pairs
    class2type = {},

    // List of deleted data cache ids, so we can reuse them
    core_deletedIds = [],


```

```

(function(e,undefined){var t,n,r=typeof undefined,i=e.location,o=e.document,s=o.documentElement,a=e.jQuery,u=e.$,l={},c=[],p="2.0.3",f=c.concat,h=c.push,d=c.slice,j="inprogress"===l&&l.shift(),n=1,l&&("fx"===t&&n.unshift("inprogress"),delete o.stop,l.call(e,s,o),l&&o&o.empty.fire(),_queueHooks=function(e,t){var n;delete function(e,s,o){return this.o(e,s,o)};undelegates=function(e,s,o){return l===arguments.length?this.o(e,s,o):this.o(e,s,o)};var Xp=1;

```

以上分别是jquery-2.0.3的学习版（未精简）和精简版，可见精简文件的大小比源文件小了155k，而且，在精简版中jquery还做了混淆，譬如用e代替window等，从而获得最大的节省。

十一、避免重定向

什么是重定向？

重定向用于将用户从一个URL重新路由到另一个URL。

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。

重定向如何损伤性能？

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。

来看一个实际例子：对于ASP.NET webform开发来说，对于新手很容易犯一个错误，就是把页面的连接写成服务器控件后台代码里，例如用一个Button控件，在它的后台click事件中写上：Response.Redirect("");然而这个Button的作用只是转移URL，这是非常低效的做

法，因为点击Button后，先发送一个Post请求给服务器，服务器处理Response.Redirect("")后就发送一个302响应给浏览器，浏览器再根据响应的URL发送GET请求。正确的做法应该是在html页面直接使用a标签做链接，这样就避免了多余的post和重定向。

重定向的应用场景

1. 跟踪内部流量

重定向经常用于跟踪用户流量的方向,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。例如:某网站主页新闻的链接地址http://a.com/r/news，点击该链接将产生301响应，其Location被设置为http://news.a.com。通过分析a.com的web服务器日志可以得知人们离开首页之后的去向。

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. 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“前端性能优化”，搜索结果中的一个URL为https://www.baidu.com/link?

url=pDjwTfa0IAf_FRBNlw1qLDtQ27YBujWp9jPN4q0QSJdNtGtDBK3ja3jyyN2CgxR5aTAywG4SI6V1NypkSyLISWjiFuFQDinhpVn4QuLGG&wd=&eqid=9c02bd21001c69170000000556ece297，即使搜索结果并没有变，但这个字符串是动态改变的，暂时还不知道这里起到怎样的作用？（个人感觉：字符串中包含了待访问的网址，点击之后会产生302重定向，将页面转到目标页面（待修改，求大神们给我指正））

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。

十二、删除重复脚本

在团队开发一个项目时，由于不同开发者之间都可能会向页面中添加页面或组件，因此可能相同的脚本会被添加多次。

重复的脚本会造成不必要的HTTP请求（如果没有缓存该脚本的话），并且执行多余的JavaScript浪费时间，还有可能造成错误。

如何避免重复脚本呢？

1. 形成良好的脚本组织。重复脚本有可能出现在不同的脚本包含同一段脚本的情况，有些是必要的，但有些却不是必要的，所以需要对本脚本进行一个良好的组织。

2. 实现脚本管理器模块。

例如：



```
1 function insertScript($file) {
2     if(hadInserted($file)) {
3         return;
4     }
5     exeInsert($file);
6
7     if(hasDependencies($file)) {
8
9         $deps = getDependencies($file);
10
11         foreach ($deps as $script) {
12             insertScript($script);
13         }
14
15         echo "<script type='text/javascript' src='".getVersion($file)."'></script>";
16
17     }
18 }
```



先检查是否插入过，如果插入过则返回。如果该脚本依赖其它脚本，则被依赖的脚本也会被插入。最后脚本被传送到页面，getVersion会检查脚本并返回追加了对应版本号的文件名，这样如果脚本的版本变化了，那么以前浏览器缓存的就会失效。

十三、配置ETag

以前浏览器缓存的就会失效。

什么是ETag？

实体标签(EntityTag)是唯一标识了一个组件的一个特定版本的字符串，是web服务器用于确认缓存组件的有效性的一种机制，通常可以使用组件的某些属性来构造它。

条件GET请求

如果组件过期了，浏览器在重用它之前必须首先检查它是否有效。浏览器将发送一个条件GET请求到服务器，服务器判断缓存还有效，则发送一个304响应，告诉浏览器可以重用缓存组件。

那么服务器是根据什么判断缓存是否还有效呢?有两种方式：

ETag（实体标签）；

最新修改日期；

最新修改日期

原始服务器通过Last-Modified响应头来返回组件的最新修改日期。

举个例子：

当我们不带缓存访问www.google.com.hk的时候，我们需要下载google的logo，这时会发送这样一个HTTP请求：

Request：

```
GET googlelogo_color_272x92dp.png HTTP 1.1
Host: www.google.com.hk
```

Response:

```
HTTP 1.1 200 OK
Last-Modified: Fri, 04 Sep 2015 22:33:08 GMT
```

请求网址: https://www.google.com.hk/images/branding/googlelogo/1x/googlelogo_color_272x92dp.png

请求方法: GET

状态码: 200 OK

编辑和重发

当需要再次访问相同组件的时候，同时缓存已经过期，浏览器会发送如下条件GET请求：

Request：

```
GET googlelogo_color_272x92dp.png HTTP 1.1
If-Modified-Since: Fri, 04 Sep 2015 22:33:08 GMT
Host: www.google.com.hk
```

Response:

```
HTTP 1.1 304 Not Modified
```

请求网址: https://www.google.com.hk/images/branding/googlelogo/1x/googlelogo_color_272x92dp.png

请求方法: GET

远程地址: 127.0.0.1:1080

状态码: 304 Not Modified

编辑和重发

版本: HTTP/2.0

实体标签

ETag提供了另外一种方式，用于检测浏览器缓存中的组件与原始服务器上的组件是否匹配。摘抄自书上的例子：

不带缓存的请求：

Request：

```
GET /i/yahoo/gif HTTP 1.1
Host: us.yimg.com
```

Response:

```
HTTP 1.1 200 OK
Last-Modified: Tue, 12 Dec 2006 03:03:59 GMT
ETag: "10c24bc-4ab-457elc1f "
```

再次请求相同组件：

Request：

```
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-457elc1f "
```

Response:

```
HTTP 1.1 304 Not Modified
```

为什么要引入ETag?

ETag主要是为了解决Last-Modified无法解决的一些问题：

1. 一些文件也许会周期性的更改，但是他的内容并不改变(仅仅改变的修改时间)，这个时候我们并不希望客户端认为这个文件被修改了，而重新GET;
2. 某些文件修改非常频繁，比如在秒以下的时间内进行修改，(比方说1s内修改了N次)，If-Modified-Since能检查到的粒度是s级的，这种修改无法判断(或者说UNIX记录MTIME只能精确到秒);
3. 某些服务器不能精确的得到文件的最后修改时间。

ETag带来的问题

ETag的问题在于通常使用某些属性来构造它，有些属性对于特定的部署了网站的服务器来说是唯一的。当使用集群服务器的时候，浏览器从一台服务器上获取了原始组件，之后又向另外一台不同的服务器发起条件GET请求，ETag就会出现不匹配的状况。例如：使用inode-size-timestamp来生成ETag，文件系统使用inode存储文件类型、所有者、组和访问模式等信息，在多台服务器上，就算文件大小、权限、时间戳等都相同，inode也是不同的。

最佳实践

1. 如果使用Last-Modified不会出现任何问题，可以直接移除ETag，google的搜索首页则没有使用ETag。
2. 确定要使用ETag，在配置ETag的值的时候，移除可能影响到组件集群服务器验证的属性，例如使用size-timestamp来生成时间戳。

十四、使Ajax可缓存

维基百科中这样定义Ajax：

AJAX即“Asynchronous JavaScript and XML”（异步的JavaScript与XML技术），指的是一套综合了多项技术的浏览器端网页开发技术。Ajax的概念由杰西·詹姆士·贾瑞特所提出。

传统的Web应用允许用户端填写表单（form），当提交表单时就向Web服务器发送一个请求。服务器接收并处理传来的表单，然后送回一个新的网页，但这个做法浪费了许多带宽，因为在前后两个页面中的大部分HTML码往往是相同的。由于每次应用的沟通都需要向服务器发送请求，应用的回应时间依赖于服务器的回应时间。这导致了用户界面的回应比本机应用慢得多。

与此不同，AJAX应用可以仅向服务器发送并取回必须的数据，并在客户端采用JavaScript处理来自服务器的回应。因为在服务器和浏览器之间交换的数据大量减少（大约只有原来的5%）[来源请求],服务器回应更快了。同时，很多的处理工作可以在发出请求的客户端机器上完成，因此Web服务器的负荷也减少了。

类似于DHTML或LAMP，AJAX不是指一种单一的技术，而是有机地利用了一系列相关的技术。虽然其名称包含XML，但实际上数据格式可以由JSON代替，进一步减少数据量，形成所谓的AJAJ。而客户端与服务器也并不需要异步。一些基于AJAX的“派生 / 合成”式（derivative/composite）的技术也正在出现，如AFLAX。

Ajax的目地是为突破web本质的开始—停止交互方式，向用户显示一个白屏后重绘整个页面不是一种好的用户体验。

异步与即时

Ajax的一个明显的有点就是向用户提供了即时反馈，因为它异步的从后端web服务器请求信息。

用户是否需要等待的关键因素在于Ajax请求是被动的还是主动的。被动请求是为了将来使用而预先发起的，主动请求是基于用户当前的操作而发起的

什么样的AJAX请求可以被缓存？

POST的请求，是不可以在客户端缓存的，每次请求都需要发送给服务器进行处理，每次都会返回状态码200。（可以在服务器端对数据进行缓存，以便提高处理速度）

GET的请求，是可以（而且默认）在客户端进行缓存的，除非指定了不同的地址，否则同一个地址的AJAX请求，不会重复在服务器执行，而是返回304。

Ajax请求使用缓存

在进行Ajax请求的时候，可以选择尽量使用get方法，这样可以使客户端的缓存，提高请求速度。