**移动端缓存性能优化**

三个主要的改善性能的策略是：

（1）、减少每个页面需要获取额外资源的HTTP请求数。

（2）、减少每个请求加载的大小。

（3）、优化客户端执行的优先级和脚本执行的效率。

1. 减少请求

最大的性能漏洞就是一个页面需要发起几十个网络请求来获取诸如样式表、脚本或者图片这样的资源，这个在相对低带宽和高延迟的移动设备连接上来说影响更严重。

CDNs（内容分发网络）把资源放在离用户地理位置更近的地方对解决这个问题能起到很大作用，但是比起获取请求，大量的请求对页面加载时间的影响更为严重，而且最近的发现表明，CDNs对移动端用户的性能影响越来越低。

2. 整合资源

对开发者来说，将Javascript代码和CSS样式放到公共的文件中供多个页面共享是一种标准的优化方法，这个方法能很简单的维护代码，并且提高客户端缓存的使用效率。

在Javascript文件中，要确保在一个页面中相同的脚本不会被加载多次，当大团队或者多个团队合作开发的时候，这种冗余的脚本就很容易出现，你可能会对它的发生频率并不低感到非常吃惊。

Sprites是css中处理图片的一项技术，Sprites就是将多张图片整合到一个线性的网状的大图片中，页面就可以将这个大图片一次性获取回来并且做为css的背景图，然后使用css的背景定位属性展示页面需要的图片部分，这种技术将多个请求整合成一个，能显著地改善性能。

平稳地改进但是需要对资源有控制权限，根据开发者的网站不同权限，一些资源并不需要被整合起来（例如，一些由CMS生成的资源），还有，对于一些外部域引用的资源，强行整合可能会导致问题，整合资源对手机浏览器来说是一把双刃剑，整合资源确实会在首次访问减少请求，但是大的资源文件可能会导致缓存失效，所以，需要小心地使用各种技术整合资源，以达到优化本地存储的目的。

3. 使用浏览器缓存和本地缓存

现在所有的浏览器都会使用本地资源去缓存住那些被Cache-Control或者Expires头标记的资源，这些头能标记资源需要缓存的时间，另外，ETag（实体标签）和Last-Modified头来标识当资源过期后是否需要重新请求，浏览器为了减少不必要的服务器请求，尽可能地从本地缓存中获取资源，并且将那些已经过期的、或者当缓存空间减小的时候将那些很久不用的资源进行清理，浏览器缓存通常包括图片，CSS，Javascript代码，这些缓存能合理地提高网站的性能（比如为了支持后退和前进的按钮，使用一个单独的缓存来保存整个渲染的页面）。

移动浏览器缓存，通常是比桌面PC小的多，这就导致了缓存的数据会很经常被清理，HTML5的缓存基于浏览器缓存提供了一个很好的替换方案，Javascript的localStorage已经在所有主流的桌面和移动端浏览器上都实现了，使用脚本代码能简便地支持HTML5的localStorage操作，可以读写键值数据，每个域名大概有5MB的容量，虽然不同的移动浏览器上读写速度相差很大，但是localStorage大容量的缓存使得它很适合作为客户端的缓存，从localStorage获取资源明显快于从服务器上获取资源，而且在大多数移动设备上也比依靠缓存头或者浏览器的本地缓存更灵活可靠，这是移动浏览器比桌面PC更有优势的一个地方，在桌面PC上，本地缓存仍然优先使用标准的浏览器缓存，导致桌面PC本地缓存的性能落后于移动浏览器。

4. 首次使用的时候在HTML中嵌入资源

HTML的标准是使用链接来加载外部资源，这使得更容易在服务器上（或者在CDN上）操作更新这些资源，而不是在每个页面上修改更新这些资源，根据上文讨论的，这种模式也使得浏览器能从本地缓存而不是服务器上获取资源。

但是对还没有缓存到浏览器localStorage的资源来说，这种模式对网站的性能有负面的影响，一般来说，一个页面需要几十个单独的请求来获取资源从而渲染页面。

所以说，从性能的角度来说，如果一个资源没有很高的被缓存的几率的话，最好把它嵌入到页面的HTML中（叫inlining），而不是使用链接外部，脚本和样式是支持内嵌到HTML中的，但是图片和其他的二进制资源其实也是可以通过内嵌包含base64编码的文本来嵌入到HTML中的。

内嵌的缺点是页面的大小会变得非常大，所以对于Web应用来说，关键的是能够跟踪分析这个资源什么时候需要从服务端获取，什么时候已经缓存到客户端了。

另外，在第一次请求资源后必须能够使用代码在客户端缓存资源，因此，在移动设备上，使用HTML5 localStorage能很好地做到内嵌。

由于不知道用户是否已经访问过这个页面了，所以需要网站有机制能生成不同版本的页面。

5.使用HTML5服务端发送事件

Web应用已经使用了各种从服务器上轮询资源的方法来持续地更新页面，HTML5的EventSource对象和Server-Sent事件能通过浏览器端的JavaScript代码打开一个服务端连接客户端的单向通道，服务端可以使用这个写通道来发送数据，这样能节省了HTTP创建多个轮询请求的消耗。

这种方式比HTML的WebSocket更高效，WebSocket的使用场景是，当有许多客户端和服务端的交互的时候（比如消息或者游戏），在全双工连接上建立一个双向通道。

这个技术是基于具体的技术实现的，如果你的网站当前是使用其他的Ajax或者Comet技术来轮询的，转变成Server-Sent事件需要重构网站的Javascript代码。

6、消除重定向

当用户在一个移动设备上访问桌面PC网站的时候，Web网站应用通常读取HTTP的user-agent头来判断这个用户是否是来自移动设备的，然后应用会发送带有空HTTP body和重定向HTTP地址头的HTTP 301（或者302）请求，把用户重定向到网站的移动版本上去，但是这个额外的客户端和服务端的交互通常在移动网络上会消耗几百毫秒，因此，在原先的请求上传递移动的web页会比传递一个重定向的信息并让客户端再请求移动页面更快。

对于那些想要在移动设备上看桌面PC网站的用户来说，你可以在移动web页面上提供一个链接入口，这样也能同时表示你的网站是并不提倡这种行为的。

虽然这个技术在理论上是简单的，但是实际上并不易于实施，由于有些m.sites是宿主在其他地方的，所以许多网站会选择重定向到一个不同的服务器上，有的网站则是会在重定向请求的时候种植上Cookie告诉Web应用这个用户是在使用移动设备，这种方法可能对web应用来说更容易控制。

7、减少资源负载

关于移动端页面的大小问题，渲染小页面更快，获取小资源也更快，减小每个请求的大小通常不如减少页面请求个数那么显著地提高性能。

但是有些技术在性能方面，特别是在需要对带宽和处理器性能精打细算的移动设备环境下，仍然是能带来很大利益的。

8、压缩文本和图像

诸如gzip这样的压缩技术，依靠增加服务端压缩和浏览器解压的步骤，来减少资源的负载，但是，一般来说，这些操作都是被高度优化过了，而且测试表明，压缩对网站还是起到优化性能的作用的，那些基于文本的响应，包括HTML，XML，JSON（Javascript Object Notation），Javascript，和CSS可以减少大约70%的大小。

浏览器在Accept-Encoding请求头中申明它的解压缩技术，并且当它们接收到服务端返回的Content-Encoding响应头标示的时候，就会按照这个响应头自动做解压操作。

8、代码简化

简化通常是使用在脚本和样式文件中，删除一些不必要的字符，比如空格，换行符，或者注释等，不需要暴露给外部的命名就可以被缩短为一个或者两个字符，比如变量名，合适的简化资源通常在客户端不需要做任何其他的处理，并且平均减少20%的资源大小，内嵌在HTML中的脚本和样式文件也是可以精简的，有很多很好的库来做精简化的操作，这些库一般也同时会提供合并多个文件这样减少请求数的服务。

简化带来的好处并不局限于减少带宽和延迟，对于那些移动设备上缓存无法保存的过大资源来说，也是很有改善的，Gzip在这个方面并没有任何帮助，因为资源是在被解压后才被缓存起来的。

Google的Closure Compiler已经难以置信地完成了理解和简化Javascript的工作，但是CSS的简化则没有那么容易，因为对不同浏览器来说有不同的CSS技术能迷惑CSS简化工具，然后让CSS简化后无法正常工作，即使只是删除了不必要的字符，简化工作也有可能破坏页面，所以当你应用简化技术之后，请做一下完整的功能测试工作。

9、调整图片大小

图片通常是占用了Web页面加载的大部分网络资源，也占用了页面缓存的主要空间，小屏幕的移动设备提供了通过调整图片大小来加速传输和渲染图片资源的机会，如果用户只是在小的移动浏览器窗口中看图片的话，高分辨率的图片就会浪费带宽、处理时间和缓存空间。

为了加速页面渲染速度和减少带宽及内存消耗，可以动态地调整图片大小或者将图片替换为移动设备专用的更小的版本，不要依靠浏览器来将高分辨率的图片转换成小尺寸的图片，这样会浪费带宽。

另外一个方法是先尽快加载一个低分辨率的图片来渲染页面，在onload或者用户已经开始和页面交互以后将这些低分辨率的图片替换成为高分辨率的图片。

特别应用在高度动态化的网站是有优势的。

10、使用HTML5和CSS 3.0来简化页面

HTML5包括了一些新的结构元素，例如header，nav，article和footer，使用这些语义化的元素比传统的使用div和span标签能使得页面更简单和更容易解析，一个简单的页面更小加载更快，并且简单的DOM（Document Object Model）代表着更快的JavaScript执行效率，新的标签能很快地应用在包括移动端的新浏览器版本上，并且HTML5设计让那些不支持它的浏览器能平稳过渡使用新标签。

HTML5的一些表单元素提供了许多新属性来完成原本需要javascript来完成的功能，例如，新的placeholder属性用于显示在用户输入进入输入框之前显示的介绍性文字，autofocus属性用于标示哪个输入框应当被自动定位。

也有一些新的输入框元素能不用依靠Javascript就可以完成一些通用的需求，这些新的输入框类型包括像e-mail，URL，数字，范围，日期和时间这样需要复杂的用户交互和输入验证的元素，在移动浏览器上，当需要输入文本的时候，弹出的键盘通常是由特定的输入框类型来做选择的，不支持指定的输入类型的浏览器就会只显示一个文本框。

另外，只要浏览器支持内建的层次，圆角，阴影，动画，过渡和其他的图片效果，CSS 3.0就能帮助你创建轻便简易的页面了，而这些图片效果原先是需要加载图片才能完成的，这样，这些新特性就能加速页面渲染了。

人工地做这些改动是非常复杂和耗时的，如果你使用CMS，它可以帮你生成许多你不需要控制的HTML和CSS。

11、延迟渲染”BELOW-THE-FOLD”内容

可以确定的是如果我们将不可见区域的内容延迟加载，那么页面就会更快地展现在用户面前，这个区域叫做“below the fold”，为了减少页面加载后需要重新访问的内容，可以将图片替换为正确的高宽所标记的<img>标签。

一些好的Javascript库可以用来处理这些below-the-fold 延迟加载的图像。

12、延迟读取和执行的脚本

在一些移动设备上，解析Javascript代码的速度能达到100毫秒每千字节，许多脚本的库直到页面被渲染以后都是不需要的加载的，下载和解析这些脚本可以很安全地被推迟到onload事件之后来做。

例如，一些需要用户交互的行为，比如托和拽，都不大可能在用户看到页面之前被调用，相同的逻辑也可以应用在脚本执行上面，尽量将脚本的执行延迟到onload事件之后，而不是在初始化页面中重要的可被用户看到的内容的时候执行。

这些延迟的脚本可能是你自己写的，更重要的是，也有可能是第三方的，对广告、社交媒体部件、或者分析的差劲的脚本优化会导致阻塞页面的渲染，会增加珍贵的加载时间，当然，你需要小心地评估诸如jquery这样为移动网站设计的大型脚本框架，特别当你仅仅只是使用这些框架中的一些对象的时候更要小心评估。

许多第三方的框架现在提供延迟加载的异步版本的API，开发者只需要将原先的逻辑转化到这个异步版本，一些JavaScript要做延迟加载会有些复杂，因为在onload之后执行这些脚本需要注意很多注意事项（例如，你有个脚本需要绑定到onload事件上，你需要做什么？如果你将脚本延迟到onload事件之后，就一定就会失去很多执行的时机）。

13、使用Ajax来增强进程

Ajax（Asynchronous JavaScript and XML）是一项使用XHR（XMLHttpRequest）对象来从Web服务器上获取数据的技术，它并不需要更新正在运行的页面，Ajax能更新页面上的某个部分而不需要重新构建整个页面，它通常用来提交用户的交互相应，但是也可以用来先加载页面的框架部分，然后当用户准备好浏览网页的时候再填充详细的内容。

尽管是这个名字，但是XMLHttpRequest并不强制要求你只能使用XML，你可以通过调用overrideMineType方法来制定“application/json”类型来使用json替换XML，使用JSON.parse会比使用原生的eval()函数快了几乎两倍，并且更为安全。

同时，切记Ajax的返回响应也会得益于那些应用在普通的返回响应的优化技术上面，确保对你的Ajax返回响应使用了缓存头，简化，gzip压缩，资源合并等技术。

由于这个技术是根据具体应用不同而不同的，所以很难量化，或许由于跨域问题，你需要使用XHR2，这个技术能使用外部域的资源，从而能进行跨域的XHR请求。

14、根据网络状况进行适配处理

由于使用更多带宽会使用更多移动网络的费用，所以只有能检测网络的类型才能使用针对特定网络的优化技术。

例如，预加载未来使用到的请求是非常聪明的做法，但是如果用户的带宽很稀有，并且加载的有些资源是永远不会用到的话，这个技术就是不合理的了。

在Android 2.2+，navigator.connection.type属性的返回值能让你区分Wifi和2G/3G/4G网络，在Blackberry上，blackberry.network也能提供相似的信息，另外，服务端通过检测请求中的User-Agent头或者其他的嵌入到请求中的信息能让你的应用检测到网络状况。

检测网络信息的API最近已经有所变化了，接口现在不是直接定义Wi-Fi，3G等网络状况，而是给出了带宽信息和诸如“非常慢，慢，快和非常快”这样的建议，有个属性能给出估计的MB/s值和一个“meterd”的Boolean值来表示它的可信度，但是对浏览器来说，很难根据这个来判断环境，判断当前网络环境然后适配仍然是一种最好的方法，但是这种方法正在被考虑被替换。