Web前端性能测试

安大叔

2013-03-26

web前端性能测试目录

目录:

web前端性能测试的目的

- **前端性能测试对象：**
  HTML、CSS、JS、AJAX等前端技术开发的Web页面
- **影响用户浏览网页速度的因素：**
  服务端数据返回、网络传输、页面渲染等
- **前端性能测试目的：**
  计算出包含页面渲染、网络传输以及服务器端解析等综合因素在内的加载时间等指标，对该页面性能进行评估分析，找出影响性能的主要因素和瓶颈，并在此结果的基础上，给出一定的优化建议和解决方案，从而提升用户体验
- **YSlow/PageSpeed：**
  通过网页JS/CSS/Image数及请求数量、请求类型、缓存等方面的静态分析
- **DynaTrace：**
  支持IE，不仅可以显示所有请求和文件在网络中传输的时间，还会记录浏览器Render、CPU消耗、JavaScript解析和运行情况等详细的动态分析
- **WebPageTest：**
  在线的站点性能评测网站，地址http://www.webpagetest.org/

web前端性能测试的目的

- **ShowSlow：**

  showslow是yslow的数据收集与展示平台http://www.showslow.com/，它是一个开源的php项目，可以用来与firefox的yslow插件、page speed插件或者dynatrace通信，收集插件或程序所发送过来的信息并集中展示。只需要在dynatrace安装目录下进行一些设置，即可自动实现上传结果到showslow平台作为存档、分析及监控。

- **Speed Tracer：**

  基于chrome的插件，同样是有google产的，这个是web前端页的性能记录和分析工具，同时还提供一个规则和建议的评测，这个工具收集的东西主要是资源或事件消耗的时间，它会实时的记录某个页面的加载过程，并且一直跟踪所有的事件；在易用性方面数据可以到出来，还有可以根据时间轴来分析具体某端的性能规则和建议

- **Fiddler / HttpAnalyzer/ HttpWatch / DynaTrace Ajax Edition：**

  一些单独工具类的web前端性能分析工具

web前端性能测试

- 网页速度除了后台需要在性能上做优化外，其实前端的页面更需要在性能优化上下功夫，只有这样才能给我们的用户带来更好的用户体验。不仅仅如此，如果前端优化得好，他不仅可以为企业节约成本，他还能给用户带来更多的用户，因为增强的用户体验
- 前端的页面主要包括xhtml,css,js。其实xhtml就是现实中所谈到的内容，页面的内容：文字，图片，flash,视频等。
- 而前端开发工作者可以控制的是什么呢？那就是xhtml，css，js的代码及相应的修饰（背景）图片

web前端性能测试简介

- 浏览器大致工作流程

  下载：可以细分为：DNS解析、建立连接、发送请求、等待响应、接收数据。这部分的时间主要消耗在服务器端生成动态html上。

  解析：分为：解析、执行、绘制、重绘等过程。且对不同的对象又各有不同，如html/CSS/JS的解析）

web前端性能测试简介

一个完整的页面请求及响应过程：

1、浏览器的url请求
2、递归寻找DNS服务器
3、连接目标IP并建立TCP连接
4、向目标服务器发送http请求
5、web服务器接收请求后处理
6、web服务器返回相应的结果【无效、重定向、正确页面等】
7、浏览器接收返回的http内容
8、开始解析html文件，当然是自上而下，先是头部，后是body
9、当解析到头部css外部链接时，同步去下载，如果遇到外部js链接也是下载【不过js链接不建议放在头部，因为耽误页面第一展现时间】
10、接着解析body部分，边解析边开始生成对应的DOM树，同时等待css文件下载
11、一旦css文件下载完毕，那么就同步去用已经生成的DOM节点+CSS去生成渲染树
12、渲染树一旦有结构模型了，接着就会同步去计算渲染树节点的布局位置
13、一旦计算出来渲染的坐标后，又同步去开始渲染

web前端性能测试简介

一个完整的页面请求及响应过程：

15、同14步，如果渲染过程中出现js代码调整DOM树机构的情况，也会再次重新来过，从修改DOM那步开始

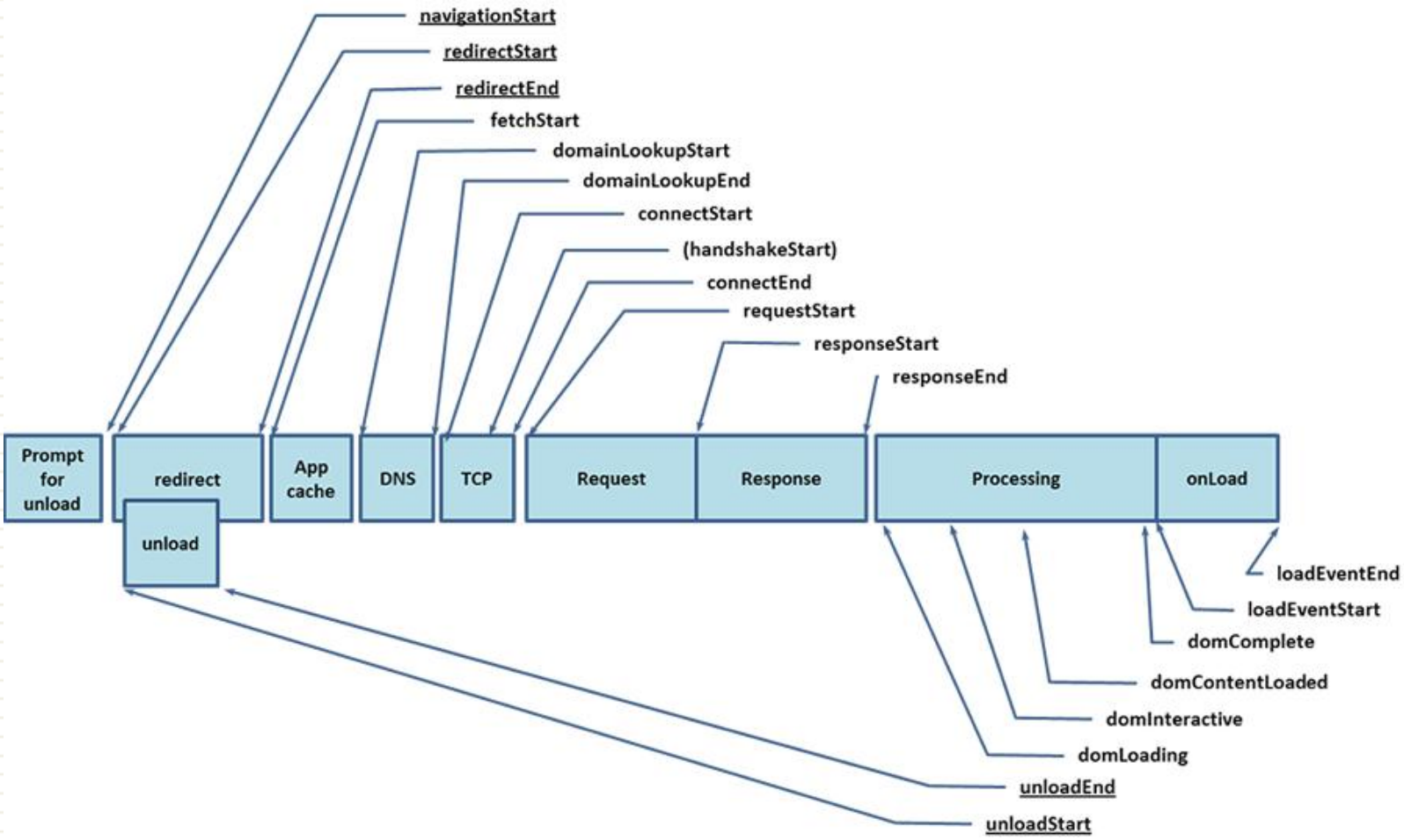
16、最终所有节点和资源都会渲染完成

17、渲染完成后开始page的onload事件

18、整个页面load完成

整个过程中会有很多的分别请求，所以TCP连接会很多，并且每一个用完都会自己关了，除非是keep-live类型的可以请求多次才关闭。对web应用前端性能的研究并不是为了准确的得到一个响应时间数据，实际上，web性能一部分取决于web服务器和应用服务器（建立连接、下载资源文件），另一部分取决于浏览器的实现机制、web页面上的js文件的执行等（分割线以内的步骤过程），我们并不仅仅关注页面资源的解析和展示响应时间，而是要关注总时间；我们进行web前端性能测试的目的是计算出包含页面渲染、网络传输以及服务器端解析等综合因素在内的加载时间等指标，对该页面性能进行评估分析，找出影响性能的主要因素和瓶颈，并在此结果的基础上，给出一定的优化建议和解决方案，从而提升用户体验。

web前端性能测试简介



web前端性能测试简介

- 前端性能指标：

  1. Dynatrace时间：

     1、首次显示时间（Time to First Impression）：在浏览器地址栏输入URL按回车到用户看到网页的第一个视觉标志为止。

     2、onLoad事件时间（Time to onLoad Event）：这个时间是直到浏览器触发onLoad事件的时间，当原始文档和所有引用的内容完全下载后才会触发这个事件

     3、完全载入的时间（Time to Fully Loaded）：等于直到所有onLoad JavaScript 处理程序执行完毕，所有动态的或延迟加载的内容都通过这些处理程序触发的时间

- 2. 页面大小： 组成页面的所有资源总大小，图像/css/js的大小也可以单独成为一个指标。
- 3. 请求数量： 从网站下载资源时所有网络请求的总数，尽量少。

web前端性能测试简介

- 4. 网络方面的指标（yslow中的一些指标的意义）：

  1、DNS时间：托管网站资源的每个域名都会发生一次DNS查找，如果你在多个网页之间移动，当前一个页面已经请求过一次DNS查找后，浏览器不会再对同一个域名请求另一个DNS查找，但通过查看总体DNS时间，可以确定是否存在DNS查找时间问题，有可能牵出DNS配置不当的问题。

  2、连接时间：根据浏览器和资源的大小不同，浏览器可能会在域名上建立一到多个连接，连接时间就是浏览器与Web服务器建立TCP/IP连接的时间，连接通常会保持打开状态，除非Web服务器命令浏览器关闭连接。当使用SSL建立安全通信时，连接时间也包括SSL握手的时间，连接时间过长有以下原因：到Web服务器的网络连接速度较慢，使用了SSL，不允许浏览器保持连接打开。

  3、服务器时间：高服务器时间意味着Web/应用程序服务器需要很长的时间处理请求，监视服务器时间对于找出性能瓶颈和应用程序的扩展问题是至关重要的，通过增加Web服务器实现负载均衡，使静态内容的扩展是很容易的，当然也可以购买CDN加速服务来达到同样的目的，但以这种方法实现动态应用程序扩展就行不通了。

  4、传输时间：这个时间与传输内容的大小，浏览器与服务器之间的连接速度紧密相关，保持低传输时间是确保页面快速载入的关键，可以通过减小总的页面大小，或者通过CDN将内容放在离最终用户较近的地方改善传输时间。

  5、等待时间：等待时间与相同域名下资源的数量直接相关，受浏览器同一域名物理网络连接数的限制，访问某个资源时可能必须等待另一个连接的释放，减少资源的数量，或将资源分布在多个域名上，可以有效减少等待时间。

  6、域名的数量: 托管网站资源域名的数量也很重要，因为它会影响DNS，连接和等待时间，要下载的资源使用额外的域名将会直接减少等待时间.

雅虎评估网站性能的23条军规

- 雅虎曾经针对网站速度提出了非常著名34条准则：《Best Practices for Speeding Up Your Web Site》。而现在将34条精简为更加直观的23条，并针对每一条给出从F~A的评分以及最终的总分。
- 目前体现为**Yslow-23条规则**

雅虎评估网站性能的23条军规

- 雅虎曾经针对网站速度提出了非常著名34条准则：《Best Practices for Speeding Up Your Web Site》。而现在将34条精简为更加直观的23条，并针对每一条给出从F~A的评分以及最终的总分。
- 目前体现为**Yslow-23条规则**

Yslow-23条规则

- 1. 减少HTTP请求次数
  合并图片、CSS、JS，改进首次访问用户等待时间。
- 2. 使用CDN
  就近缓存==>智能路由==>负载均衡==>WSA全站动态加速
- 3. 避免空的src和href
  当link标签的href属性为空、script标签的src属性为空的时候，浏览器渲染的时候会把当前页面的URL作为它们的属性值，从而把页面的内容加载进来作为它们的值。测试
- 4. 为文件头指定Expires
  使内容具有缓存性。避免了接下来的页面访问中不必要的HTTP请求。
- 5. 使用gzip压缩内容
  压缩任何一个文本类型的响应，包括XML和JSON，都是值得的。旧文章
- 6. 把CSS放到顶部
- 7. 把JS放到底部
  防止js加载对之后资源造成阻塞。

Yslow-23条规则

- 8. 避免使用CSS表达式
- 9. 将CSS和JS放到外部文件中

  目的是缓存，但有时候为了减少请求，也会直接写到页面里，需根据PV和IP的比例权衡。
- 10. 权衡DNS查找次数

  减少主机名可以节省响应时间。但同时，需要注意，减少主机会减少页面中并行下载的数量。

  IE浏览器在同一时刻只能从同一域名下载两个文件。当在一个页面显示多张图片时，IE 用户的图片下载速度就会受到影响。所以新浪会搞N个二级域名来放图片。
- 11. 精简CSS和JS
- 12. 避免跳转

  同域：注意避免反斜杠“/”的跳转；

  跨域：使用Alias或者mod_rewirte建立CNAME（保存域名与域名之间关系的DNS记录）

Yslow-23条规则

- 13. 删除重复的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

  它用来判断浏览器缓存里的元素是否和原来服务器上的一致。比last-modified date更具有弹性，例如某个文件在1秒内修改了10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. 可缓存的AJAX

  “异步”并不意味着“即时”： Ajax并不能保证用户不会在等待异步的JavaScript和XML响应上花费时间。
- 16. 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。因此使用GET获取数据时更加有意义。

Yslow-23条规则

- 17. 减少DOM元素数量

  是否存在一个是更贴切的标签可以使用？人生不仅仅是DIV+CSS
- 18. 避免404

  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。
- 19. 减少Cookie的大小
- 20. 使用无cookie的域

  比如图片 CSS 等，Yahoo! 的静态文件都在主域名以外，客户端请求静态文件的时候，减少了 Cookie 的反复传输对主域名的影响。
- 21. 不要使用滤镜

  png24的在IE6半透明那种东西，别乱使，淡定的切成PNG8+jpg
- 22. 不要在HTML中缩放图片
- 23. 缩小favicon.ico并缓存

Yslow-23条规则

- 17. 减少DOM元素数量

  是否存在一个是更贴切的标签可以使用？人生不仅仅是DIV+CSS
- 18. 避免404

  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。
- 19. 减少Cookie的大小
- 20. 使用无cookie的域

  比如图片 CSS 等，Yahoo! 的静态文件都在主域名以外，客户端请求静态文件的时候，减少了 Cookie 的反复传输对主域名的影响。
- 21. 不要使用滤镜

  png24的在IE6半透明那种东西，别乱使，淡定的切成PNG8+jpg
- 22. 不要在HTML中缩放图片
- 23. 缩小favicon.ico并缓存

尽可能的减少HTTP的Request请求数

- **第一条：Make Fewer HTTP Requests 尽可能的减少HTTP的Request请求数。**

  80%的用户响应时间都是浪费在前端。而这些时间主要又是因为下载图片、样式表、JavaScript脚本、flash等文件造成的。减少这些资源文件的Request请求数将是提高网页显示效率的重点。

  这里好像有个矛盾，就是如果我减少了很多的图片，样式，脚本或者flash，那么网页岂不是光秃秃的，那多难看呢？其实这是一个误解。我们只是说尽量的减少，并没有说完全不能使用。减少这些文件的Request请求数，当然也有一些技巧和建议的：

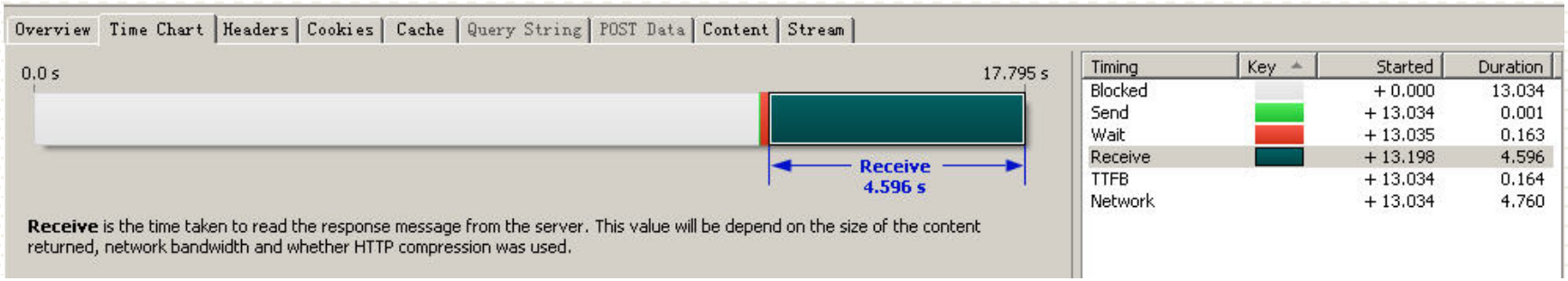
- **1：用一个大图片代替多个小图片。**

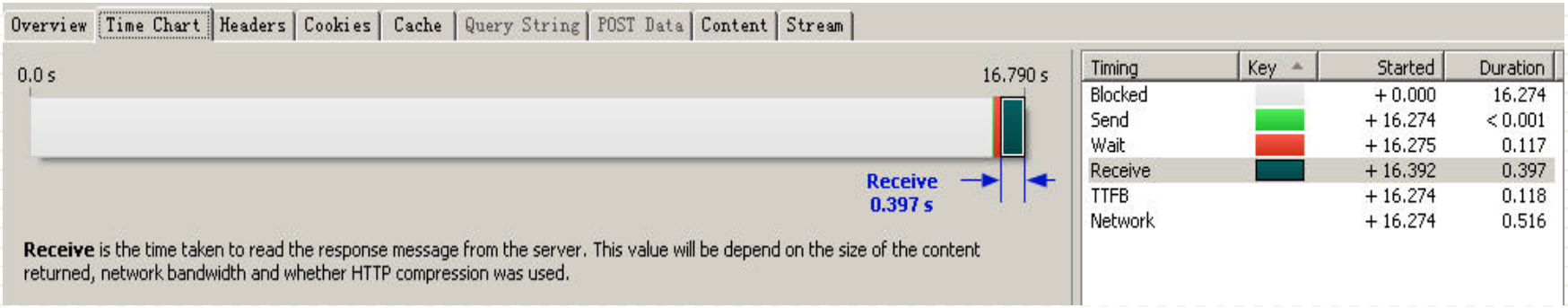
  这的确有点颠覆传统的思维了。以前我们一直以为多个小图片的下载速度之和会小于一个大图片的下载速度。但是现在利用httpwatch工具的对多个页面进行分析后的结果表明事实并不是这样。

  第一张图是一个大小为40528bytes的337*191px的大图片的分析结果。

  第二张图是一个大小为13883bytes的280*90px的小图片的分析结果。

尽可能的减少HTTP的Request请求数



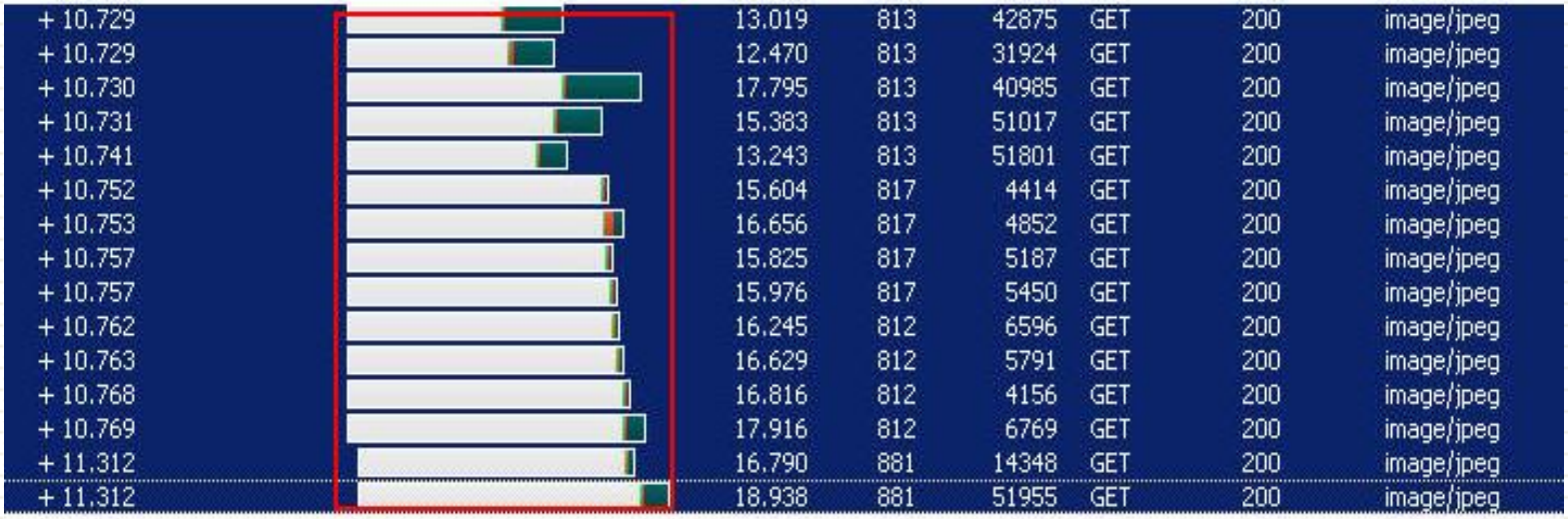


尽可能的减少HTTP的Request请求数

- 第一张大图片花费时间为：
  Blocked：13.034s
  Send：0.001s
  Wait：0.163s
  Receive：4.596s
  TTFB：0.164s
  NetWork：4.760s
  功耗时：17.795s
  真正用于传输大文件花费的时间为Reveive时间，即4.596s，多数的时间是用来检索缓存和确定链接是否有效的Blocked时间，供花费13.034s，占总时间的73.2%。
- 第二张小图片花费时间为：
  Blocked：16.274s
  Send：小于0.001s
  Wait：0.117s
  Receive：0.397s
  TTFB：0.118s
  NetWork：0.516s
  功耗时：16.790s
  真正用于传输文件的花费时间是Reveive时间，即0.397s，这的确要比刚才大文件的4.596s小很多。但是他的Blocked时间为16.274s，占总时间的97%。

尽可能的减少HTTP的Request请求数

- 如果这些数据还不够说服你的话，让我们看看下面这张图。这里列出了某个网页中所有图片中的花费时间示意图。当然，里面的图片有大有小，规格不一。



尽可能的减少HTTP的Request请求数

- 上图中大约80%以上的时间是用来检索缓存和确定链接是否有效的Blocked时间。
- 其中藏青色的为传输文件花费的Reveive时间，而前面白色的为检索缓存和确认链接是否有效的Blocked时间。铁一样的事实告诉我们：

  1.大文件和小文件下载所需时间的确是不同的，差异的绝对值不大。而且下载所需时间占总耗费时间比例很小。

  2.大约80%以上的时间是用来检索缓存和确定链接是否有效的Blocked时间。无论文件大小，这个时间的花费大致是相同的。而且所占总耗费时间的比例是极大的。

  3.一个100k的大图片总耗费时间绝对大于4个25k的小图片的总耗费时间。而且主要差别就是4个小图片的Blocked时间绝对大于1个大图片的Blocked时间。

  4.所以如果可能还是使用大图片来替代过多的琐碎的小图片吧。这也是为什么翻转门的效率要高于图片替换实现的滑动门的原因。
  但是，请注意：也不能用太大的单张图片，因为那样会影响到用户体验。例如个几兆的背景图片的使用绝对不是一个好主意。

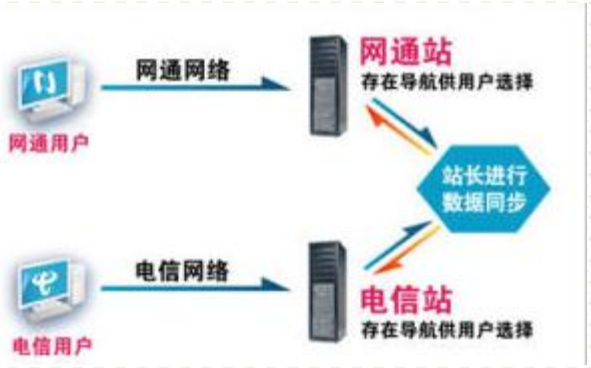
尽可能的减少HTTP的Request请求数

- 2：合并你的css文件
- 3：合并你的javascript文件。

使用CDN

- 这个看上去好像很深奥的样子，但是只要结合中国的网络特色，这个便不难理解了。“北方服务器”、“南方服务器”、“电信服务器”、“网通服务器”.....这些词听起来是那么熟悉和压抑。如果，一个北京的电信用户试图从广东的网通服务器上打开一个类似《壁纸合集》帖子的网页时，你就能很深刻的理解。

  鉴于这个不是我们开发人员力所能及的准则，所以这里也就不多言了。



启用Gzip压缩/添加周期头

- **Add an Expires Header** 在中间件/应用服务器中配置
- **Gzip Components** 在中间件/应用服务器中配置

把CSS样式放在页面的上方/将脚本放在底部

- **Put CSS at the Top：** 无论是HTML还是XHTML还是CSS都是解释型的语言，而非编译型的。所以CSS到上方的话，那么浏览器解析结构的时候，就已经可以对页面进行 渲染。这样就不会出现，页面结构光秃秃的先出来，然后CSS渲染，页面又突然华丽起来，这样太具有“戏剧性”的页面浏览体验了。
- **Move Scripts to the Bottom ：** 原因同上面一样。只是脚本一般是用来于用户交互的。所以如果页面还没有出来，用户连页面都不知道什么样子，那谈交互简直就是扯谈。所以，脚本和CSS正好相反，脚本应该放在页面的底部。

避免使用CSS中的Expressions

- 首先说明一下CSS Expressions是什么一个东西。其实它就像其它语言中的if……else……语句。这样在CSS中就可以进行简单的逻辑判断了。举个简单的例子——
- <style>
  input{background-color:expression((this.readOnly && this.readOnly==true)?”#0000ff”:”#ff0000”)}
  </style>
  <INPUT TYPE=”text” NAME=””>
  <INPUT TYPE=”text” NAME=”" readonly=”true”>
- 这样css就可以根结一些情况分别使用不同的样式了。但是CSS中Expressions 的代价却是极高的。当你的页面需要根据判断来渲染效果的元素很多的时候，那么你的浏览器将长期处于假死状态，从而给用户带来极差的用户体验。

将javascript和css独立成外部文件

- **Make JavaScript and CSS External：** 这一条好像和第一条有点矛盾。的确，如果从HTTP的request请求数来讲的话，这样做的确是降低了效率。但是之所以这么做，是因为另外一个重要的考虑因素——缓存。因为外部的引用文件会被浏览器缓存，所以如果javascript和css体积较大的时候，我们将它们独立成外部文件。这样当用户 只要浏览一次以后，这些体积较大的js和css文件就能被缓存起来，从而极高地提高用户再次访问时的效率。

减少DNS查询

- **Reduce DNS Lookups：** DNS域名解析系统。大家都知道我们之所以能记住那么多的网址，是因为我们记住的都是单词，而非http://202.153.125.45这样的东西，而帮我们把那些单词和202.153.125.45这样的ip地址联系起来的就是DNS。那这一条对我们到底有什么真正意义上的指导意义呢？其实有 两条：
  1：如果不是必须，请不要把网站放到两台服务器上。
  2：网页中的图片、css文件、js文件、flash文件等等，不要太多的分散在不同的网络空间中。这就是为什么那种只发一个网站中的壁纸图片的帖子，要比壁纸图片来源于不同网站的帖子显示要快得多的原因。

减少JavaScript和CSS文件的体积

- **Minify JavaScript and CSS：** 这点很好理解。在你的最终发布版本中把没有必要的空行、空格和注释全部去掉。显然手工去处理效率太低，好在网上到处都是用于压缩这些东西的工具。压缩JavaScript代码体积的工具随处可见，我便不再列举了，这里我只提供一个用于压缩css代码体积的在线工具网站——
  http://www.cssdrive.com/index.php/main/csscompressor
  它提供了多种压缩方式，可以适应多种要求。

避免跳转

- **Avoid Redirects：**我只从网页开发人员的角度来解读此条。那么我们可以解读到什么东西呢？2点——
  1：“此域名已过期，5秒钟以后，页面将跳转到http://www.xxxxxx.com/index.html页面”，这句话看起来的确很熟悉。但是，我就奇怪了，为什么不直接链接到那个页面呢？
  2：一些链接地址请更明确的写出来。例如:将http://andashu.cnblogs.com/ 写成http://andashu.cnblogs.com（注意最后面一个“/”符号）。的确，这两个网址都能访问到我的博客，但是，事实上，它们是有区别的。http://andashucnblogs.com的结果是个301响应，它会被重新指向http://andashu.cnblogs.com/。但是显然，中间多浪费了一些时间。

移除重复的脚本

- **Remove Duplicate Scripts：** 这个准则的道理很浅显，但是真正在工作中，很多人却因为“项目时间紧”、“太累了”、“初期没有规划好”……这样的理由搪塞过去了。你，的确可以找很多的理由不去处理这些多余重复的脚本代码，如果你的网站不需要更高的效率和后期维护的话。
  也正是这点，我提醒大家一些，一些javascript框架、javascript包一定要慎用。至少要问一下：用了这个js kit 到底给我们多少方便，提高了多少工作效率。然后，再与它因为多余的、重复的代码带来的负面效果比较一下。

配置你的实体标签

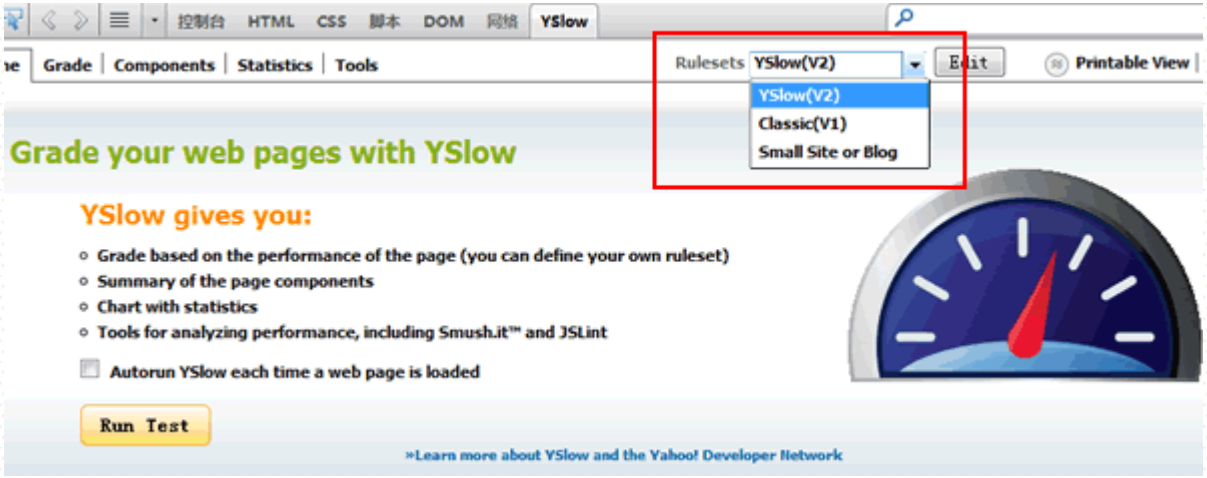
- **Configure ETags：** 首先来讲讲什么是Etag吧。Etag（Entity tags）实体标签。这个tag和你在网上经常看到的标签云那种tag有点区别。这个Etag不是给用户用的，而是给浏览器缓存用的。Etag是服务器告诉浏览器 缓存，缓存中的内容是否已经发生变化的一种机制。通过Etag，浏览器就可以知道现在的缓存中的内容是不是最新的，需不需要重新从服务器上重新下载。这和“Last-Modified”的概念有点类似。在服务器上添加。

Ajax的使用要恰当

- **Make Ajax Cacheable：** 现在的Ajax好像有点被神话了，好像网页只要Ajax了，那么就不存在效率问题了。其实这是一种误解。拙劣的使用Ajax不会让你的网页效率更高，反而会降低你的网页效率。Ajax的确是个好东西，但是请不要过分的神话它。使用Ajax的时候也要考虑上面的那些准则。

网站性能测试工具Yslow

- Yslow是雅虎开发的基于网页性能分析浏览器插件，一般与firebug一起使用



网站性能测试工具Yslow

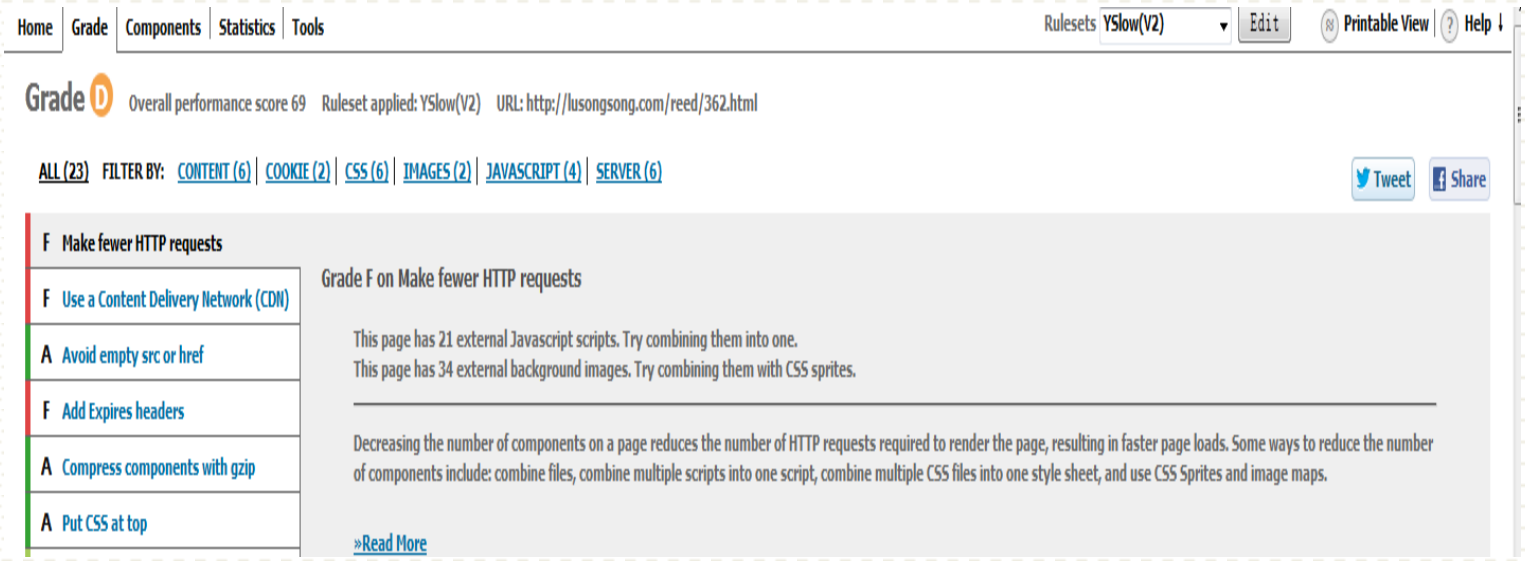
- 击 Run Test 运行Yslow，也可以点击 Grade, Components, 或Statistics选项开始对页面的分析，如果在 Autorun YSlow each time a web page is loaded 上打上对勾，它将自动对以后打开页面进行分。
- 注意图中的红框，这里是规则集，YSlow （V2）包含了所有22个测试的规则，YSlow （V1）包含原始13规则，小网站或博客-这个规则集包含14个规则，适用于小型网站或博客，建议对号入座。

网站性能测试工具Yslow

- 雅虎曾经针对网站速度提出了非常著名34条准则：《Best Practices for Speeding Up Your Web Site》。而现在将34条精简为更加直观的23条，并针对每一条给出从F~A的评分以及最终的总分。
- 而现在23条网站性能优化建议在YSlow的官网首页就能看到，当然也可以不看，在使用Yslow后，在控制面板里就会给你评分提示，和改进建议。

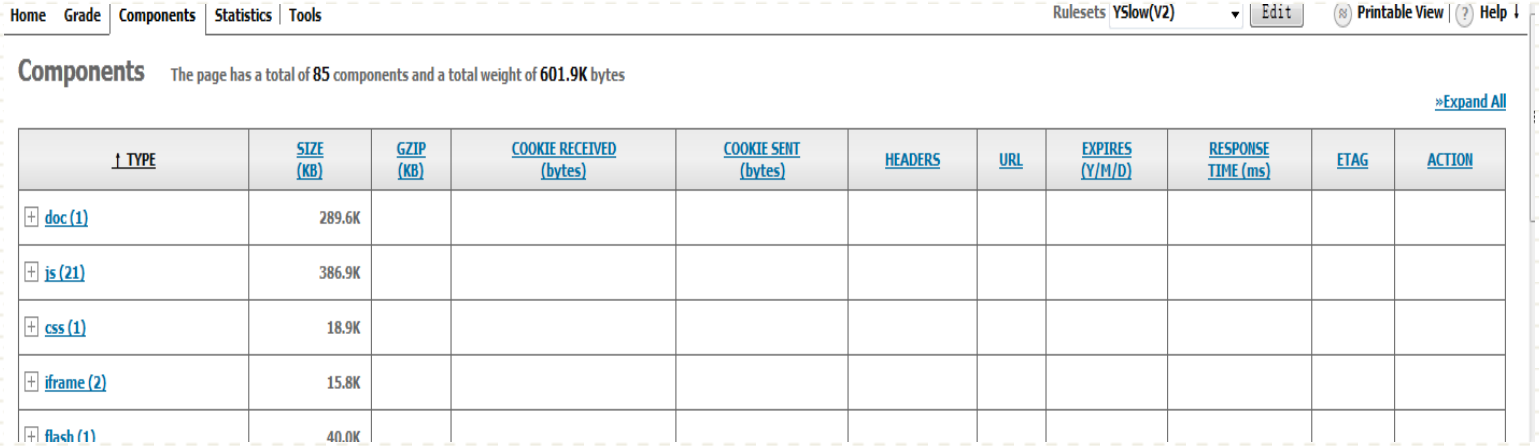
网站性能测试工具Yslow

- **Grade(等级视图)—**Yslow给出的网站性能评分，从F~A，A是最好的，并且有改进意见



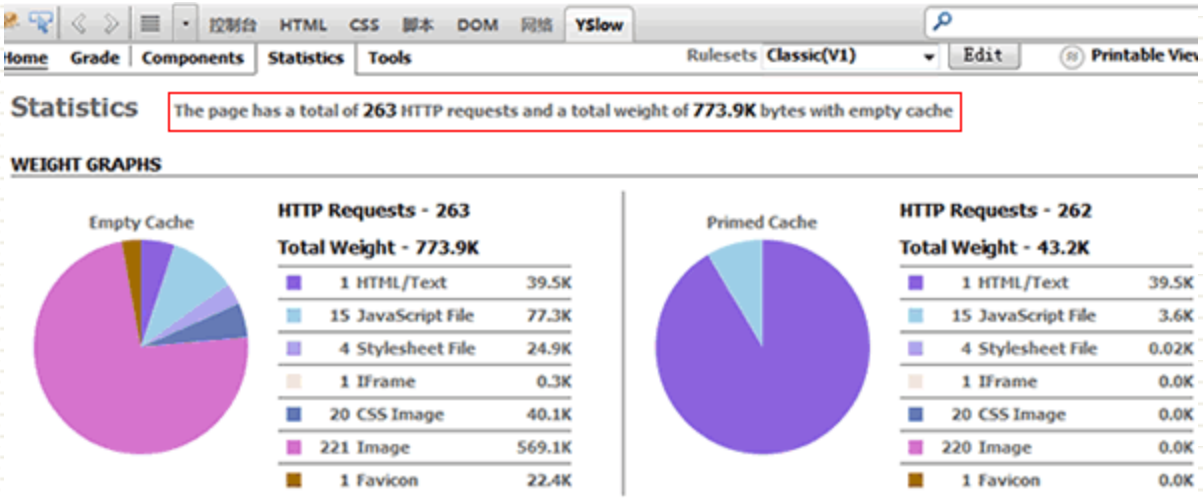
网站性能测试工具Yslow

- **Components：** 通过Components考验查看网页各个元素占用的空间大小，例如我博客某个页面，有236个images（图片），占用了489.2K，通过详 细查看，发现来自gravatar（评论头像）的引用图片非常大，在加上我博客本省评论量就打，每个头像就占用几K，几百个就占用了整个网页50%的大 小，而且图片还是引用的，加载就更慢。
- 所以，我得出的结论是：gravatar虽然增强了互动性和个性，但也结结实实影响了网站速度。



网站性能测试工具Yslow

- **Statistics：** 侧图表显示是页面元素在空缓存的加载情况，右侧为页面元素使用缓存后的页面加载情况。从图中可以直观的看出（尤其是我标的红框），这个网页263个 HTTP请求，网页的大小达到773.9K，意味着打开没打开一个页面几乎需要下载1M的东西，而通过使用缓存后我们可以看到效果图片基本靠缓存，而网页 的总大小压缩到43.2K。
- Statistics这个统计信息视图工具和Components（第三选项卡）一样，只是效果更直观，如果要获得性能优化建议还是要看Grade（第二选项卡）的详细建议。



网站性能测试工具Yslow

- **Tools（辅助工具）** JSLint是一个强大的工具，它可以检验HTML代码以及内联的Javascript代码，通过JSLint发现了google analytics上的一个js错误。
- ALL JS：查看你这个网页上一共引用了多少JS。
- All JS Beautified：把所有JS放在打开的页面中，利用站长统一检查（我感觉作用不大）。
- All JS Minified：同上，但它显示的是压缩过的js代码，如果你要JS优化，它已经给你优化好了，来过来直接用。
- All CSS：显示你网页所有CSS文件。
- YUI CSS Compressor：显示网页压缩后的CSS文件，也是拿过来可以直接用的。
- All Smush.it™：图片在线优化网站，点击它后会自动跳到smushit网站上给你自动优化CSS图片，该网站提供了优化前与优化后的对比，点击直接下载优化后的图片，在覆盖到自己网站上就可以了，强烈推荐。
- Printable View：这个是打印用的，部门开会、前端设计师讨论、向老板汇报时估计用的上。

结束语

- 谢谢！