你了解 HTTP 么？你知道如何减少 HTTP 请求确保你的性能最优

1.HTTP 简介

超文本传输协议（HTTP，HyperText Transfer Protocol)是互联网上应用最为广泛的一种网络协议。所有的 WWW 文件都必须遵守这个标准。设计 HTTP 最初的目的是为了提供一种发布和接收 HTML 页面的方法。1960 年美国人 Ted Nelson 构思了一种通过计算机处理文本信息的方法，并称之为超文本（hypertext）,这成为了 HTTP 超文本传输协议标准架构的发展根基。Ted Nelson 组织协调万维网协会（World Wide Web Consortium）和互联网工程工作小组（Internet Engineering Task Force ）共同合作研究，最终发布了一系列的 RFC，其中著名的 RFC 2616 定义了 HTTP 1.1。

1.0 版本

1996 年 5 月，HTTP/1.0 版本发布，内容大大增加。

首先，任何格式的内容都可以发送。这使得互联网不仅可以传输文字，还能传输图像、视频、二进制文件。这为互联网的大发展奠定了基础。

其次，除了 GET 命令，还引入了 POST 命令和 HEAD 命令，丰富了浏览器与服务器的互动手段。

再次，HTTP 请求和回应的格式也变了。除了数据部分，每次通信都必须包括头信息（HTTP header），用来描述一些元数据。

其他的新增功能还包括状态码（status code）、多字符集支持、多部分发送（multi-part type）、权限（authorization）、缓存（cache）、内容编码（content encoding）等。

请求格式

下面是一个 1.0 版的 HTTP 请求的例子。

```
GET / HTTP/1.0
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5)
Accept: */*
```

可以看到，这个格式与 0.9 版有很大变化。

第一行是请求命令，必须在尾部添加协议版本（HTTP/1.0）。后面就是多行头信息，描述客户端的情况。

回应格式

服务器的回应如下。

```
HTTP/1.0 200 OK
Content-Type: text/plain
Content-Length: 137582
Expires: Thu, 05 Dec 1997 16:00:00 GMT
Last-Modified: Wed, 5 August 1996 15:55:28 GMT
Server: Apache 0.84
```

回应的格式是"头信息 + 一个空行（\r\n） + 数据"。其中，第一行是"协议版本 + 状态码（status code） + 状态描述"。

Content-Type 字段

关于字符的编码，1.0 版规定，头信息必须是 ASCII 码，后面的数据可以是任何格式。因此，服务器回应的时候，必须告诉客户端，数据是什么格式，这就是 Content-Type 字段的作用。

下面是一些常见的 Content-Type 字段的值。

```
text/plain
text/html
text/css
image/jpeg
image/png
image/svg+xml
audio/mp4
video/mp4
application/javascript
application/pdf
application/zip
application/atom+xml
```

这些数据类型总称为 MIME type，每个值包括一级类型和二级类型，之间用斜杠分隔。

除了预定义的类型，厂商也可以自定义类型。

```
application/vnd.debian.binary-package
```

上面的类型表明，发送的是 Debian 系统的二进制数据包。

MIME type 还可以在尾部使用分号，添加参数。

```
Content-Type: text/html; charset=utf-8
```

上面的类型表明，发送的是网页，而且编码是 UTF-8。

客户端请求的时候，可以使用 Accept 字段声明自己可以接受哪些数据格式。

```
Accept: */*
```

上面代码中，客户端声明自己可以接受任何格式的数据。

MIME type 不仅用在 HTTP 协议，还可以用在其他地方，比如 HTML 网页。

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<!-- 等同于 -->
<meta charset="utf-8" />
```

Content-Encoding 字段

由于发送的数据可以是任何格式，因此可以把数据压缩后再发送。Content-Encoding 字段说明数据的压缩方法。

```
Content-Encoding: gzip
Content-Encoding: compress
Content-Encoding: deflate
```

客户端在请求时，用 Accept-Encoding 字段说明自己可以接受哪些压缩方法。

```
Accept-Encoding: gzip, deflate
```

缺点

HTTP/1.0 版的主要缺点是，每个 TCP 连接只能发送一个请求。发送数据完毕，连接就关闭，如果还要请求其他资源，就必须再新建一个连接。

TCP 连接的新建成本很高，因为需要客户端和服务器三次握手，并且开始时发送速率较慢（slow start）。所以，HTTP 1.0 版本的性能比较差。随着网页加载的外部资源越来越多，这个问题就愈发突出了。

为了解决这个问题，有些浏览器在请求时，用了一个非标准的 Connection 字段。

```
Connection: keep-alive
```

这个字段要求服务器不要关闭 TCP 连接，以便其他请求复用。服务器同样回应这个字段。

```
Connection: keep-alive
```

一个可以复用的 TCP 连接就建立了，直到客户端或服务器主动关闭连接。但是，这不是标准字段，不同实现的行为可能不一致，因此不是根本的解决办法。

1.1 版本

1997 年 1 月，HTTP/1.1 版本发布，只比 1.0 版本晚了半年。它进一步完善了 HTTP 协议，一直用到了 20 年后的今天，直到现在还是最流行的版本。

持久连接

1.1 版的最大变化，就是引入了持久连接（persistent connection），即 TCP 连接默认不关闭，可以被多个请求复用，不用声明 Connection: keep-alive。

客户端和服务器发现对方一段时间没有活动，就可以主动关闭连接。不过，规范的做法是，客户端在最后一个请求时，发送 Connection: close，明确要求服务器关闭 TCP 连接。

```
Connection: close
```

目前，对于同一个域名，大多数浏览器允许同时建立 6 个持久连接。

管道机制

1.1 版还引入了管道机制（pipelining），即在同一个 TCP 连接里面，客户端可以同时发送多个请求。这样就进一步改进了 HTTP 协议的效率。

举例来说，客户端需要请求两个资源。以前的做法是，在同一个 TCP 连接里面，先发送 A 请求，然后等待服务器做出回应，收到后再发出 B 请求。管道机制则是允许浏览器同时发出 A 请求和 B 请求，但是服务器还是按照顺序，先回应 A 请求，完成后再回应 B 请求。

Content-Length 字段

一个 TCP 连接现在可以传送多个回应，势必就要有一种机制，区分数据包是属于哪一个回应的。这就是 Content-length 字段的作用，声明本次回应的数据长度。

```
Content-Length: 3495
```

上面代码告诉浏览器，本次回应的长度是 3495 个字节，后面的字节就属于下一个回应了。

在 1.0 版中，Content-Length 字段不是必需的，因为浏览器发现服务器关闭了 TCP 连接，就表明收到的数据包已经全了。

分块传输编码

使用 Content-Length 字段的前提条件是，服务器发送回应之前，必须知道回应的数据长度。

对于一些很耗时的动态操作来说，这意味着，服务器要等到所有操作完成，才能发送数据，显然这样的效率不高。更好的处理方法是，产生一块数据，就发送一块，采用"流模式"（stream）取代"缓存模式"（buffer）。

因此，1.1 版规定可以不使用 Content-Length 字段，而使用"分块传输编码"（chunked transfer encoding）。只要请求或回应的头信息有 Transfer-Encoding 字段，就表明回应将由数量未定的数据块组成。

```
Transfer-Encoding: chunked
```

每个非空的数据块之前，会有一个 16 进制的数值，表示这个块的长度。最后是一个大小为 0 的块，就表示本次回应的数据发送完了。下面是一个例子。

```
HTTP/1.1 200 OK
Content-Type: text/plain
Transfer-Encoding: chunked

25
This is the data in the first chunk

1C
and this is the second one

3
con

8
sequence
```

0

其他功能

1.1 版还新增了许多动词方法：PUT、PATCH、HEAD、 OPTIONS、DELETE。

另外，客户端请求的头信息新增了 Host 字段，用来指定服务器的域名。

Host: www.example.com

有了 Host 字段，就可以将请求发往同一台服务器上的不同网站，为虚拟主机的兴起打下了基础。

缺点

虽然 1.1 版允许复用 TCP 连接，但是同一个 TCP 连接里面，所有的数据通信是按次序进行的。服务器只有处理完一个回应，才会进行下一个回应。要是前面的回应特别慢，后面就会有许多请求排队等着。这称为"队头堵塞"（Head-of-line blocking）。

为了避免这个问题，只有两种方法：一是减少请求数，二是同时多开持久连接。这导致了很多的网页优化技巧，比如合并脚本和样式表、将图片嵌入 CSS 代码、域名分片（domain sharding）等等。如果 HTTP 协议设计得更好一些，这些额外的工作是可以避免的。

对于前端来说我们也要了解 hTTP 相关的内容。那么对于前端优化来说，HTTP 方面该如何进行优化呢。今天想说一个方向——减少 HTTP 请求次数减少性能开销。该怎么做，能减少哪些方向的性能开销，以及性能如何提升。

2. 关于减少 http 请求数

关于减少 http 请求数,是前端开发性能优化的一个非常重要方面,所以在基本所有的优化原则里,都有这一条原则:减少 http 请求数.

先不考虑其他的,我们先考虑为什么减少 http 请求可以优化性能.

减少 http 请求有这样几个优点:

(1) 减少 DNS 请求所耗费的时间.

且不说对错,因为从基本来说,减少 http 请求数的确可以减少 DNS 请求和解析耗费的时间.

(2) 减少服务器压力.

这个通常是被考虑最多的,也是我用来讲解给别人听的最大理由,因为每个 http 请求都会耗费服务器资源,特别是一些需要计算合并等操作的服务器,耗费服务器的 cpu 资源可不是开玩笑的事情,硬盘可以用钱买来,cpu 资源可就没那么廉价了.

(3)　　减少 http 请求头.

当我们对服务器发起一个请求的时候,我们会携带着这个域名下的 cookie 和一些其他的信息在 http 头部里,然后服务器响应请求的时候也会带回一些 cookie 之类的头部信息.这些信息有的时候会很大,在这种请求和响应的时候会影响带宽性能.

3. 解释原因

(1) 什么是 DNS 请求和解析呢?

简单来说,例如:www.taobao.com 这样一个 url,其中 www 部分被称为主机名(hostname),taobao 这部分则是二级域,com 则是一级域,如果是这样一个网址:www.ali.tao.com 那么 ali 就是三级域.

当我们去请求一个 url 的时候,首先会到本地服务器里去寻找缓存中是否有解析结果,如果没有解析结果,就去根域名服务器请求,根域名服务器返回给本 地域名服务器一个所查询的域的主域名服务器的 ip 地址,然后我们再去请求刚才返回的 ip 地址的域名服务器,然后返回下一级域名的 ip 地址,直到我们找到域 名中所指的服务器 ip,然后将结果缓存起来供下次使用,并返回此结果.

一个第一次请求的 url 的 DNS 解析过程可能耗费是很高的.但是解析一次之后,结果就会被缓存起来,之后再请求的时候就不用走上面这一套复杂的解析过程了.

关于一个正常的 DNS 请求到底会耗费多少时间,这个没有定论,要看网速状况和地域,但是考虑一个 dns 解析解析过后会被缓存起来,像淘宝这样的大网站,来的都是回头客,我们是否可以忽略 DNS 解析花费的时间呢?

在前端优化里还有一个优化方法,那就是增加 hostname,例如,淘宝图片服务器,分为 img01, img02, img03 等主机名,我们在一个页面里的图片放在不同的 hostname 下,这样就可以同时下载多个图片了,浏览器 http 连接数的限制可以被缓解一下。

为什么对于淘宝的图片来说,使用不同的 hostname 是个更优的方案呢?

首先,因为淘宝网的特殊性,淘宝网大多数访问者都是回头客,他们电脑里大多缓存着 dns 记录.这种情况,如果是小网站或者新兴网站可能要考虑,因为新用户比较多,可能 dns 请求的消耗更大一些,而且第一印象对于这些网站来说更为重要.

再者,淘宝里的图片很多,一个页面里通常会用到几十张甚至上百张图片,在这种情况下,我们更需要突破浏览器的 http 连接数的限制,以便加快加载速 度,这时候加载速度的考虑优先级远远高于 DNS 的影响,而 yslow 中对于 DNS 的着重考虑可能更偏向中小网站,图片比较少的网站.

对于 DNS 请求或者 tcp(tcp 握手之类的也会消耗请求时间)请求之类的分配和解析的消耗,还有一个办法是 keeplive,让你的链接保持 keeplive,这样可以只建立一次链接,然后传输多个文件,可以有效减少建立连接的时间.

(2) 减少服务器压力

过多的 http 请求对于服务器来说是很危险的,如果你的服务器不是很强,请把这一条考虑放在第一位,其他的优化策略都只是优化,而这里涉及到的是服务器,你要保证你的服务器能正常运转.

当然如果你是在淘宝的话,你就可以安心坐下来跟一群牛人谈论为什么要忽略 http 对服务器的影响,因为我们要记住:我们是前端开发工程师,我们是在 做前端优化,后台和我们无关,因为我们有足够强大的技术支持和硬件支持,当网站的技术发展到一定程度的时候,我们的关注点应该向用户那里偏重,因为用户看 到的才是我们最终要展示的,用户感受到的体验和速度才是我们要达到的速度,后台我们做的再快,前台呈现慢了,我们的服务器消耗少了,省钱了,但是用户却因 此抛弃了我们,一切都是白费.所以,当后台足够支撑你不用你去考虑后台压力的时候,那就安心考虑如何做好前台的工作吧.

Yslow 真的是一个误导人的工具,只要我们按照它的原则对网站进行优化,肯定最后可以拿到高分来欺骗老板,但是对于有些场景,这些优化往往是一种 对性能的破坏,例如淘宝网的商城首页,为了提高 Yslow 评分,所有的图片都采用了一个 hostname,分数提高了,但是并行加载少了,不过商城首页都 是异步渲染和异步加载的,所以这种影响看起来并不明显.

商城首页有很多针对 yslow 的优化点,当然大多数优化是正确的,例如:导航那里,本来是全部写在页面里的,不要小看那个导航,里面有 N 个链接节 点,以至于从浏览器里复制源代码的时候浏览器会卡死,因为字节数太多了,这里 yslow 肯定不会饶过的,后来我们把导航做成了异步加载的,评分理所当然上 去了.但是这是淘宝网,我们有足够的速度来提供足够的用户体验.如果你的服务器提供不了这种速度,也承受不了这种频繁的异步请求的话,这种优化就要慎重 了,延迟可能造成导航不可用.这也是针对场景来协调的.

淘宝现在在广泛部署 CDN,CDN 可以给我们提供足够的后台资源保障,在 CDN 和后台环境不断万善的情况下,考虑重点应该更加专注于前台传输速度和展现解析速度的提升.

(3) 合并脚本和样式?

其实在前一篇文章里的那段讨论也是对于不同应用场景的不同考虑,

减少 http 请求数的一个方法,对于前端来说,那就是合并脚本和样式文件,称为 combo,通过将多个文件合并成一个文件,然后一次性传输到客户 端,这样可以减少 http 请求,的确是个有效的方法,甚至对于一些特殊的页面,例如首页,我们把样式和脚本都写在了页面里,根本没有分离出来,他们不会产 生 http 请求,当然,也不会被缓存,这是被牺牲的代价.

为什么我们要这么做,因为首页的访问量很大?这样可以有效减少 http 请求数?恩,这只是一部分原因,的确这样做有这样的好处,而且对于 assets 服务器不够强大的网站来说,在并发量大的首页上实行这一套是很有效的.

但是,淘宝访问量最大的页面并不是首页,而是 detail 页面,也就是商品详情页!

这才是我们讨论的重点,为什么首页采用 combo 甚至写在页面里,而 detail 则按照正常的样式和脚本来引用.首页是类似静态的页面,detail 则是应用型的.首页没有脚本,依然可以起到导向的作用,但是 detail 页脚本没有运行起来的话,甚至无法购买商品.

其实在这里这样讨论并不能明显看出问题所在,因为淘宝在这些方面也不是很成熟,detail 页引用了大量脚本和样式,很多内容是多余的过期的.

这从本质上来说代表两种网页类型,一种是内容型,一种则是应用型.对于内容型的网站,脚本并不是很重要(甚至样式),因为没有脚本,用户仍然可以浏 览页面,只是可能有些效果看不到而已,所以我们可以把脚本合并起来,一起放在 body 底部,在页面内容都加载完后,再一次性加载进来.而对于应用型的网 页,让应用跑起来才是最重要的,因为没有应用这个网页就变得没有意义了,这时候,按需加载脚本是一种趋势,我们需要先把应用的基本框架和功能按需加载进 来,让它们分别运行起来,而不是一起等脚本加载完再一起初始化,我们需要应用能够快速响应用户,

而且还是说到 CDN,当 CDN 变得足够强的时候,连接数已经不是瓶颈,我们应该更多考虑怎么让网页更快的展现给用户,对于无需脚本也可以提供服务的 内容型的网页,将脚本放在页面底部,合并起来(减少连接数,我们仍然需要减少连接数,在不需要太快的使用脚本的情况下),而对于应用型的网页,我们需要尽 快让功能运转,甚至让他们一部分一部分按优先级初始化,这时候就要将脚本分开,按需加载.

(4) 减少 http 请求头

http 头是个庞大的家伙,你打开 taobao.com 的首页,alert 一下 document.cookie,会发现淘宝网的 cookie 是如此 庞大,甚至比小型网页都大,每次你请求淘宝的服务器都会往返一次这些数据,还有一些其他的头部信息,占用的空间也不小,可想而知这种消耗有多大.

然后其实自从用了 CDN,这一切都无需考虑太多,因为 CDN 和淘宝主站不在一个域名下,cookie 不会互相污染,而 CDN 的域名下基本是没有 cookie 和头部信息的,所以每次请求静态资源的时候,不会带着主站的 cookie 到处跑,而只是传输资源的主题内容,所以这对

于性能的影响在使用 cdn 之后会变得很小.但是如果你的静态资源服务器和主服务器在一个域名下,那就要控制好 cookie 和其他头部信息的大小了,因为每次传送都会传送他 们.

4. 如何减少 HTTP 请求

在终端用户响应的时间中，有 80%~90%时间用于下载 HTML 文档引用的所有组件。这部分时间包括下载页面中的图像、样式表、脚本、Flash 等。因此，改善响应时间的最简单、也是最有效的途径就是减少组件的数量，并由此减少 HTTP 请求的数量。

减少页面组件数量的方法其实就是简化页面设计。那么有没有一种方法既能保持页面内容的丰富又能减少页面组件的数量？可以很容易想到的方法就是合并多个组件。

CSS Sprites (雪碧图)

雪碧图可以将多张图片合并成为一张图片，然后使用 CSS 的 background-position 属性，将其设置到背景图片期望的位置上。

例如有个 id 为#nav 的导航栏，导航栏包含四个链接，每个链接被包围在一个 LI 中，他们使用同一背景图片。每个 LI 都有一个不同的类，通过 background-position 属性指定了期望的偏移量。

{% codeblock lang:css %} ul#nav li { float: left; width: 30px; height: 30px; background-image: url(./images/sprites.png); }

.home { background-position: 0 0;} .articles { background-position: -31px 0;} .tags { background-position: -62px 0;} .about { background-position: -93px 0;} {% endcodeblock %}

雪碧图已经被广泛使用，一般用在网站上的小图标这类，数量多、体积小、不常更新的图片上，例如:淘宝首页的 Sprites。它不仅降低了下载量，而且实际上，合并后的图片会比分离的图片的总和要小，这是因为它降低了图片自身的开销（颜色表、格式信息，等等）。

内联图片 (data:URL)

通过使用 data:URL 模式可以在 Web 页面中包含图片但无需额外的 HTTP 请求。

规范中对它的描述为：

允许将小块数据内联为‘立即（immediate）数’

数据就在 URL 自身之中，格式为

{% codeblock %} data:[][;base64], {% endcodeblock %}

其实就是所谓的 Base64 图片格式。由于 data:URL 是内联在页面中的，所以在跨越不同页面时不会被缓存（document 一般不设置缓存）。

所以，更聪明的做法是使用 CSS 并将内联图片作为背景，并将该 CSS 作为外部样式表引用，这样内联图片就能缓存在样式表中了。

合并脚本和样式表

一个页面会引入多个脚本或者样式表，如果可以将这些单独的文件合并到一个文件中，可以减少 HTTP 请求的数量并缩短最终用户的响应时间。

在理想情况下，一个页面应该使用不多于一个的脚本和样式表。

然而在实际的开发环境中是很难完成的。在大型的、复杂的 Web 应用中，我们需要使用 JavaScript 的模块化的思想，将所有东西合并到一个单独的文件中看起来就是一种倒退。因此，解决的方法是遵守编译型语言的模式，保持 JavaScript 的模块化，而在生成过程中从一组特定的模块生成一个目标文件。

最后

总之,优化原则不是绝对的,对于不同的场景应该考虑不同的侧重点,别人的解决方案对于你来说不一定是最优的,应该针对自己的网站规模和类型进行适度的优化,不能盲目追求标准和最佳实践.