**探究HTTP1.x 到 HTTP2.0的改变**

在http1.x的时代, 主要影响我们网页加载速度的不是带宽, 而是连接的时延. 特别是对于我们加载网站页面这种文件数量多,文件体积小的情况

**Http事务的时延：**
(1)TCP建立连接握手
(2)TCP慢启动拥塞控制
(3)数据聚集的Nagle算法
(4)用于捎带确认的TCP延迟确认算法
(5)TIME\_WAIT时延和端口耗尽
**TCP握手时延：**三次握手+四次挥手带来时延；
**TCP慢启动时延：**一开始会限制发送的分组的速度，避免拥塞控制；新建立的连接的传输速度会小于交换过一定量数据的连接，这个会带来时延；
**Nagle算法：**TCP发送大量包含少量数据的分组，网络的性能就会下降；Nagle算法试图在发送分组之前，会将大量的TCP分组捆绑在一起，以提高网络的效率；这样就会导致小的HTTP分组无法填满一个分组，可能会一直等待其它数据的到来，导致时延；
**延迟确认：**确认之后才能继续发送报文；由于确认报文非常的小，所以TCP允许把这个确认捎带在数据分组之中，而不是直接的发送这个确认；延迟确认算法会在一个时间之内，将确认放在缓冲区之中，以寻找能够捎带它的分组，如果在那段时间内没有输出分组，就将确认以单独的分组发送；
**TIME\_WAIT时延**：2MSL

关于http1.x时代的的连接延迟和解决方案的可以参考这个文章:

https://newcandy.gitbooks.io/http\_basic\_notes/content/chapter3.html

为了更好的理解http1.x上面的连接 我们需要理解下面的几个概念:



**并行连接:** 在 HTTP 1.x的时代,在同一个tcp的通道中,请求只能一个一个一个地发送。为了提高我们加载的速度 , 浏览器同时并行的打开多个tcp会话

我们以每个主机打开最多 6 个独立tcp连接为例

• 客户可以并行分派最多 6 个请求；

• 服务器可以同时处理最多 6 个请求；

• 第一次往返可以发送的累计分组数量（TCP cwnd）增长为原来的 6 倍



如果对于一个网站有几十个独立的资源, 那么简单的每个主机的6个连接肯定是远远不够的, 那么排队的时间就会很长, 这里可以将 静态资源的分散到多个不同的子域名, 这样子就可以突破浏览器的连接限制 这这种方法我们成为: 域名分区

**持久连接：**允许HTTP设备在事务处理结束之后将TCP连接保持在打开状态，以便未来的HTTP请求重用现在的连接



**在 HTTP 1.0 中**, 没有官方的 keepalive 的操作。通常是在现有协议上添加一个指数。如果浏览器支持 keep-alive，它会在请求的包头中添加：

Connection: Keep-Alive

然后当服务器收到请求，作出回应的时候，它也添加一个头在响应中：

Connection: Keep-Alive

**在 HTTP 1.1 中** 所有的连接默认都是持续连接，除非特殊声明不支持。可以通过设置 connection:close 进行关闭

**管道化连接：**
HTTP/1.1可以在持久连接上选择使用请求管道；在响应到达之前，可以将多条请求放入队列；当第一条请求到达时u，可以发送第二条连接；



**在http1.x时代:** 我们注重优化web性能一般会从以下方面来进行





**HTTP/2**

HTTP/2 源自 SPDY/2

SPDY 系列协议由谷歌开发，于 2009 年公开。它的设计目标是降低 50% 的页面加载时间。当下部分互联网公司，例如百度、淘宝 都在自己的网站或 APP 中采用了 SPDY 系列协议，因为它对性能的提升是显而易见的。主流的浏览器（谷歌、火狐、Opera）也都早已经支持 SPDY，它已经成为了工业标准，HTTP Working-Group 最终决定以 SPDY/2 为基础，开发 HTTP/2。

但是，HTTP/2 跟 SPDY 仍有不同的地方，主要是以下两点：

1. HTTP/2 支持明文 HTTP 传输，而 SPDY 强制使用 HTTPS
2. HTTP/2 消息头的压缩算法采用 [HPACK](http://http2.github.io/http2-spec/compression.html)，而非 SPDY 采用的 [DEFLATE](http://zh.wikipedia.org/wiki/DEFLATE)

当前的HTTP/2的进度

* 三月，2012：征集提案HTTP / 2
* 十一月，2012：HTTP的初稿/ 2（基于SPDY）
* 八月，2014年：HTTP / 2草案-17和HPACK草案-12发布
* 八月，2014年：工作组最后一次通话的HTTP / 2
* 二月，2015：IESG批准HTTP / 2和HPACK草案
* 五月，2015：RFC 7540（HTTP / 2）和RFC 7541（HPACK）公布

相比 HTTP/1.x，HTTP/2 在底层传输做了很大的改动和优化：

1. HTTP/2 采用二进制格式传输数据，而非 HTTP/1.x 的文本格式。二进制格式在协议的解析和优化扩展上带来更多的优势和可能。
2. HTTP/2 对消息头采用 HPACK 进行压缩传输，能够节省消息头占用的网络的流量。而 HTTP/1.x 每次请求，都会携带大量冗余头信息，浪费了很多带宽资源。头压缩能够很好的解决该问题。
3. 多路复用，直白的说就是所有的请求都是通过一个 TCP 连接并发完成。HTTP/1.x 虽然通过 [pipeline](http://en.wikipedia.org/wiki/HTTP_pipelining) 也能并发请求，但是多个请求之间的响应会被[阻塞](http://en.wikipedia.org/wiki/Head-of-line_blocking)的，所以 [pipeline](http://en.wikipedia.org/wiki/HTTP_pipelining) 至今也没有被普及应用，而 HTTP/2 做到了真正的并发请求。同时，流还支持优先级和流量控制。
4. Server Push：服务端能够更快的把资源推送给客户端。例如服务端可以主动把 JS 和 CSS 文件推送给客户端，而不需要客户端解析 HTML 再发送这些请求。当客户端需要的时候，它已经在客户端了。

**下面来解释以下http/2的几个关键的技术点:**

**1二进制分帧:**

HTTP 1.x 以,作为文本的分隔符，而 HTTP/2.0 将所有传输的信息分为更小的帧,并对其使用二进制编码

为了说明http2中的客户端和服务器数据交互的方式, 我们需要理解以下的几个概念:

流: 已经建立连接的双向字节流

消息 : 与逻辑消息对应的一个完整的数据,可以理解为以前的一个http1.x数据包

帧: http2.0的最小传输单位, 每个帧都会包含帧首部，会标出帧所属的流



简单的总结一下:

1 所有的通信都是在一个tcp连接上面的

2 流是一个虚拟的通道, 可以承载双向的消息, 每个流都有个唯一的id

3 消息是 指逻辑上的http消息, 比如一个请求,或者是一个响应

4 帧 最小的通信单位, 可以理解为把一个http包分解成两个帧, 一个是headers帧,一个是data帧.

**2多向请求和响应**

在没有理解http2. 的相关概念前,我分不清http1中的keep-alive和http/2 中的共享连接有什么区别,不都是公用一个tcp连接来提高性能么, 答案并没有这么简单



简单的解释一下 :

1 使用同一个tcp连接,

2 在同一个tcp连接上建立多个stream流(每个stream流可以理解为以前http1.x上面的一个请求和返回的的一条通道 )

3 在tcp中, http消息分解为多个互不依赖的二进制帧, 然后乱序发送, 最后通过 帧上面的stream标识id再重新组装起来,

可以实现以下的功能:



主要解决的问题:

1 队首阻塞问题,

2 消除并行处理中存在的连接依赖问题

3 慢启动的时间减少,拥塞和对丢包的恢复速度更快

4 减少了大量的tcp连接, 延时更少, 相比于keep-alive的tcp复用和 浏览器多个tcp并行连接

5 http/1直接结合两者的优点, 在公用一个tcp连接的同事实现多个流stream的并行传输

**3请求优先级**:

在http2中每个流都有一个31bit的优先值,

**0 表示 最高的优先级**

**数字越到表示优先级越低**

服务器根据stream 不同的优先级 来响应 ,优先将优先级最高的帧发送给 客户端

为加快页面加载速度，所有现代浏览器都会基于资源的类型以及它在页面中的位置排定请求的优先次序，甚至通过之前的访问来学习优先级模式——比如，之前的渲染如果被某些资源阻塞了，那么同样的资源在下一次访问时可能就会被赋予更高的优先级。

**4服务器推送:**

简单的解释服务器推送: 服务器可以对客户端的一个请求发送多个响应 .利用这个功能我们可以做些什么来优化我们的网站呢?



通常的web页面都会有几十个资源, 客户端需要先请求到第一个文档然后解析逐个加载其他的资源, 如果使用服务器推送的话, 我们可以在接受一个请求之后提前将 页面所需要的资源一并返回给客户端, 这样带来的效果是非常卓越的, 这样做的效果等同于 在返回网页的同时 , 把网页的css和js都内嵌到页面里面

**实现:**

http2标准没有明确规定使用哪种算法, 所以要根据浏览器的具体实现来使用, 比如应用可以通过 在http首部添加一个它洗完服务器推送的资源列表,比如apache的mod\_spdy :

<https://developers.google.com/speed/spdy/mod_spdy/>

**5首部压缩:**



在服务器和客户端都知道发送那些首部之后,那么再次发送首部的数据时, 就可以只发送差异的数据

客户端和服务器同时维护一个一样的 “首部表” ,这个表在连接存在的周期一直存在,并且由浏览器和服务器同时渐进的更新

**具体的压缩方法:**

HTTP / 2通过使用两个简单但功能强大的技术HPACK压缩格式压缩请求和响应头的元数据：

1. 它允许所发送的报头字段经由静态霍夫曼码，这降低了它们各自的传输大小进行编码。
2. 它要求在客户端和服务器维护和更新先前看到头字段的索引列表（即建立一个共享压缩上下文），然后将其用作基准来高效率地编码先前发送的值。

哈夫曼编码允许传输时被压缩，而此前被传值的索引表使我们能够编码重复值 , 可用于有效地查找和重建的完整报头的键和值的索引值。

 HTTP 2.0 压缩算法，请看这里：http://tools.ietf.org/html/draft-ietf-httpbis-

header-compression。

**解释以下http2.0在共享一个 tcp连接的下的 数据流**



**最后比较一下各种**Multipexing 多路复用的差异,可以参考下图:



在http/2.0中 每次请求/响应使用不同的 Stream ID。就像同一个 TCP 链接上的数据包通过 IP:PORT来区分出数据包去往哪里一样。通过 Stream ID 标识，所有的请求和响应都可以欢快的同时跑在一条 TCP 链接上了。

**调试工具**

CloudFlare 已经有一篇 [总结得很好的 HTTP/2 调试工具总结](https://blog.cloudflare.com/tools-for-debugging-testing-and-using-http-2/)。另外，我们发现 Chrome 的 Net-internals 工具 （Chrome 中通过 chrome://net-internals/#http2 可以访问） 也很有用。

相关的文档

[拥抱HTTP2.0时代 - HTTP2.0实现服务器端推送Push功能](http://www.cnblogs.com/techgogogo/p/4284503.html)

[使用 Wireshark 调试 HTTP/2 流量](https://imququ.com/post/http2-traffic-in-wireshark.html)

[HTTP/2 十分钟速知](https://www.bokeyy.com/post/get-to-know-http-2-in-10-minutes.html#more-1612)

[http2升级实例](https://www.bokeyy.com/post/enabling-http2-for-dropbox-web-services-experiences-and-observations.html#more-2364)

[http2 RFC7540文档](http://httpwg.org/specs/rfc7540.html)