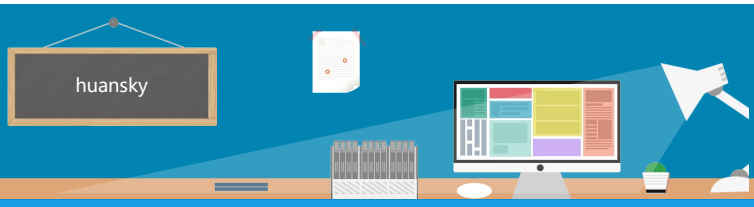


深入浅出：HTTP/2

概述

HTTP/2 的目的就是通过支持请求与响应的多路复用来减少延迟，通过压缩HTTP首部字段将协议开销降至最低，同时增加对请求优先级和服务器端推送的支持。为达成这些目标，HTTP/2 还会给我们带来大量其他协议层面的辅助实现，比如新的流量控制、错误处理和更新机制。上述几种机制虽然不是全部，但却是最重要的，所有Web开发者都应该理解并在自己的应用中利用它们。

HTTP/2 不会改动HTTP的语义。HTTP方法、状态码、URI及首部字段，等等这些核心概念一如往常。但是，HTTP/2 修改了格式化数据（分帧）的方式，以及客户端与服务器间传输这些数据的方式。这两点统帅全局，通过新的组帧机制向我们的应用隐藏了所有复杂性。换句话说，所有原来的应用都可以不必修改而在新协议运行。这当然是好事。

下面我们就来详细介绍一下这些新的机制。

历史及其与SPDY的渊源

SPDY是谷歌开发的一个实验性协议，于2009年年中发布，其主要目标是通过解决HTTP 1.1中广为人知的一些性能限制，来减少网页的加载延迟。大致上，这个项目设定的目标如下：

- 页面加载时间（PLT，Page Load Time）降低50%；
- 无需网站作者修改任何内容；
- 把部署复杂性降至最低，无需变更网络基础设施；
- 与开源社区合作开发这个新协议；
- 收集真实性能数据，验证这个实验性协议是否有效。

2012年，这个新的实验性协议得到了Chrome、Firefox和Opera的支持，很多大型网站（如谷歌、Twitter、Facebook）都对兼容客户端提供SPDY会话。换句话说，SPDY在被行业采用并证明能够大幅提升性能之后，已经具备了成为一个标准的条件。最终，HTTP-WG（HTTP Working Group）在2012年初把HTTP/2 提到了议事日程，吸取SPDY的经验教训，并在此基础上制定官方标准。

走向HTTP/2

从那时起，SPDY 已经经过了很多变化和改进，而且在 HTTP/2 官方标准公布之前，还将有很多变化和改进。在此，有必要回顾一下HTTP/2 宣言草稿。因为这份宣言明确了该协议的范围和关键设计要求：

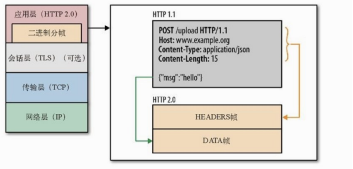
HTTP/2 应该满足如下条件：

- 相对于使用TCP的HTTP 1.1，用户在大多数情况下的感知延迟要有实质上、可度量的改进；
- 解决HTTP中的“队首阻塞”问题；
- 并行操作无需与服务器建立多个连接，从而改进TCP的利用率，特别是拥塞控制方面；
- 保持HTTP 1.1的语义，利用现有文档，包括（但不限于）HTTP方法、状态码、URI，以及首部字段；
- 明确规定HTTP/2 如何与HTTP 1.x互操作，特别是在中间介质上；
- 明确指出所有新的可扩展机制以及适当的扩展策略；

HTTP/2 特征

二进制分帧层

HTTP/2 性能增强的核心，全在于新增的二进制分帧层（如下图所示），它定义了如何封装HTTP消息并在客户端与服务器之间传输。



这里所谓的“层”，指的是位于套接字接口与应用可见的高层HTTP API之间的一个新机制：HTTP的语义，包括各种动词、方法、首部，都不受影响，不同的是传输期间对它们的编码方式变了。HTTP 1.x以换行符作为纯文本的分隔符，而HTTP/2 将所有传输的信息分割为更小的消息和帧，并对它们采用二进制格式的编码。

这样一来，客户端和服务器为了相互理解，必须都使用新的二进制编码机制：HTTP 1.x客户端无法理解只支持HTTP/2 的服务器，反之亦然。不过不要紧，现有的应用不必担心这些变化，因为客户端和服务器会替它们完成必要的分帧工作。

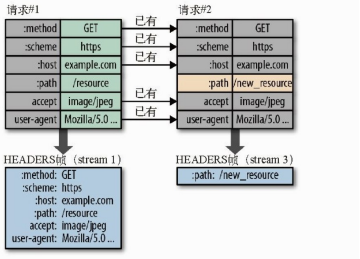
首部压缩

HTTP的每一次通信都会携带一组首部，用于描述传输的资源及其属性。在HTTP 1.x中，这些元数据都是以纯文本形式发送的，通常会给每个请求增加500～800字节的负荷。如果算上HTTP cookie，增加的负荷通常会达到上千字节。为减少这些开销并提升性能，HTTP/2会用 “HPACK” 算法来压缩头部数据。

“HPACK” 算法是专门为压缩 HTTP 头部定制的算法，与 gzip、zlib 等压缩算法不同，它是一个“有状态”的算法，需要客户端和服务器各自维护一份“索引表”，也可以说是“字典”（这有点类似 brotli），压缩和解压缩就是查表和更新表的操作。

- HTTP/2 在客户端和服务器端使用“首部表”来跟踪和存储之前发送的键 - 值对，对于相同的数据，不再通过每次请求和响应发送；
- 首部表在HTTP/2 的连接存续期内始终存在，由客户端和服务器共同渐进地更新；
- 每个新的首部键 - 值对要么被追加到当前表的末尾，要么替换表中之前的值。

于是，HTTP/2 连接的两端都知道已经发送了哪些首部，这些首部的值是什么，从而可以针对之前的数据只编码发送差异数据，具体如下图所示。



请求与响应首部的定义在HTTP/2 中基本没有改变，只是所有首部键必须全部小写，而且请求行要独立为:method 、:scheme 、:host 和:path 这些键 - 值对。

在前面的例子中，第二个请求只需要发送变化了的路径首部（:path ），其他首部没有变化，不用再发送了。这样就可以避免传输冗余的首部，从而显著减少每个请求的开销。

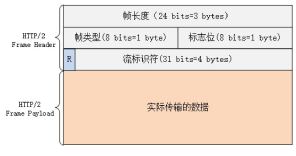
通信期间几乎不会改变的通用键 - 值对（用户代理、可接受的媒体类型，等等）只需发送一次。事实上，如果请求中不包含首部（例如对同一资源的轮询请求），那么首部开销就是零字节。此时所有首部都自动使用之前请求发送的首部！

二进制帧

头部数据压缩之后，HTTP/2 就要把报文拆成二进制的帧准备发送。

HTTP/2 的帧结构有点类似 TCP 的段或者 TLS 里的记录，但报头很小，只有 9 字节，非常地节省（可以对比一下 TCP 头，它最少是 20 个字节）。

二进制的格式也保证了不会有歧义，而且使用位运算能够非常简单高效地解析。



公告

Since 2016-03-17

昵称： huansky
园龄： 6年6个月
粉丝： 295
关注： 35
+加关注

最新随笔

我的标签

积分与排名

阅读排行榜

评论排行榜

帧开头是 3 个字节的长度（但不包括头的 9 个字节），默认上限是 2^14，最大是 2^24，也就是说 HTTP/2 的帧通常不超过 16K，最大是 16M。

长度后面的一个字节是帧类型，大致可以分成数据帧和控制帧两类，HEADERS 帧和 DATA 帧属于数据帧，存放的是 HTTP 报文，而 SETTINGS、PING、PRIORITY 等则是用来管理流的控制帧。

HTTP/2 总共定义了 10 种类型的帧，但一个字节可以表示最多 256 种，所以也允许在标准之外定义其他类型实现功能扩展。这就有点像 TLS 里扩展协议的意思了，比如 Google 的 gRPC 就利用了这个特点，定义了几种自用的新帧类型。

第 5 个字节是非常重要的帧标志信息，可以保存 8 个标志位，携带简单的控制信息。常用的标志位有END_HEADERS表示头数据结束，相当于 HTTP/1 里头后的空行（“\r\n”），END_STREAM表示单方向数据发送结束（即 EOS，End of Stream），相当于 HTTP/1 里 Chunked 分块结束标志（“0\r\n\r\n”）。

报文头里最后 4 个字节是流标识符，也就是帧所属的“流”，接收方使用它就可以从乱序的帧里识别出具有相同流 ID 的帧序列，按顺序组装起来就实现了虚拟的“流”。

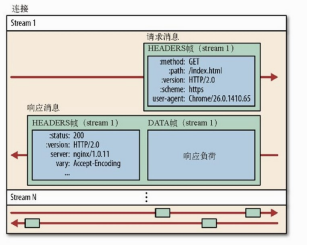
流标识符虽然有 4 个字节，但最高位被保留不用，所以只有 31 位可以使用，也就是说，流标识符的上限是 2^31，大约是 21 亿。

流、消息和帧

新的二进制分帧机制改变了客户端与服务器之间交互数据的方式（如下图所示）。为了说明这个过程，我们需要了解HTTP/2 的两个新概念。

- **流**：已建立的连接上的双向字节流。
- **消息**：与逻辑消息对应的完整的一系列数据帧。
- **帧**：HTTP/2 通信的最小单位，每个帧包含帧首部，至少也会标识出当前帧所属的流。

所有HTTP/2 通信都在一个连接上完成，这个连接可以承载任意数量的双向数据流。相应地，每个数据流以消息的形式发送，而消息由一或多个帧组成，这些帧可以乱序发送，然后再根据每个帧首部的流标识符重新组装。”



这简简单单的几句话里浓缩了大量的信息，我们再重申一次。要理解HTTP/2 ，就必须理解流、消息和帧这几个基本概念。

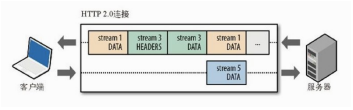
- 所有通信都在一个TCP连接上完成。
- 流是连接中的一个虚拟信道，可以承载双向的消息；每个流都有一个唯一的整数标识符（1、2...N）。
- 消息是指逻辑上的HTTP消息，比如请求、响应等，由一或多个帧组成。
- 帧是最小的通信单位，承载着特定类型的数据，如HTTP首部、负荷，等等。

简言之，HTTP/2 把HTTP协议通信的基本单位缩小为一个一个的帧，这些帧对应着逻辑流中的消息。相应地，很多流可以并行地在同一个TCP连接上交换消息。

多向请求与响应

在HTTP 1.x中，如果客户端想发送多个并行的请求以及改进性能，那么必须使用多个TCP连接。这是HTTP 1.x交付模型的直接结果，该模型会保证每个连接每次只交付一个响应（多个响应必须排队）。更糟糕的是，这种模型也会导致队首阻塞，从而造成底层TCP连接的效率低下。

HTTP/2 中新的二进制分帧层突破了这些限制，实现了多向请求和响应：客户端和服务器可以把HTTP消息分解为互不依赖的帧（如下图所示），然后乱序发送，最后再在另一端把它们重新组合起来。



上包含了同一个连接上多个传输中的数据流：客户端正在向服务器传输一个DATA帧（stream 5），与此同时，服务器正向客户端乱序发送stream 1和stream 3的一系列帧。此时，一个连接上有3个请求/响应并行交换！

把HTTP消息分解为独立的帧，交错发送，然后在另一端重新组装是HTTP/2 最重要的一项增强。事实上，这个机制会在整个Web技术栈中引发一系列连锁反应，从而带来巨大的性能提升，因为：

- 可以并行交错地发送请求，请求之间互不影响；
- 可以并行交错地发送响应，响应之间互不干扰；
- 只使用一个连接即可并行发送多个请求和响应；
- 消除不必要的延迟，从而减少页面加载的时间；
- 不必再为绕过HTTP 1.x限制而多做很多工作。
- ……

总之，HTTP/2 的二进制分帧机制解决了HTTP 1.x中存在的队首阻塞问题，也消除了并行处理和发送请求及响应时对多个连接的依赖。结果，就是应用速度更快、开发更简单、部署成本更低。

支持多向请求与响应，可以省掉针对HTTP 1.x限制所费的那些脑筋和工作，比如拼接文件、图片精灵、域名分区。类似地，通过减少TCP连接的数量，HTTP/2 也会减少客户端和服务器的CPU及内存占用。

请求优先级

把HTTP消息分解为很多独立的帧之后，就可以通过优化这些帧的交错和传输顺序，进一步提升性能。为了做到这一点，每个流都可以带有一个31比特的优先值：

- 0 表示最高优先级。
- 2^{31} -1表示最低优先级。

有了这个优先值，客户端和服务器就可以在处理不同的流时采取不同的策略，以最优的方式发送流、消息和帧。具体来讲，服务器可以根据流的优先级，控制资源分配（CPU、内存、带宽），而在响应数据准备好之后，优先将最高优先级的帧发送给客户端。

> 浏览器在渲染页面时，并非所有资源都具有相同的优先级：HTML文档本身对构建DOM不可或缺，CSS对构建CSSOM不可或缺，而DOM和CSSOM的构建都可能受到JavaScript资源的阻塞（参见10.1节的附注栏“DOM、CSSOM和JavaScript”），其他资源（如图片）的优先级都可以降低。
> 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。
> 在HTTP 1.x中，浏览器极少能利用上述优先级信息，因为协议本身并不支持多路复用，也没有办法向服务器通告请求的优先级。此时，浏览器只能依赖并行连接，且最多只能同时向一个域名发送6个请求。于是，在等连接可用期间，请求只能在客户端排队，从而增加了不必要的网络延迟。理论上，HTTP管道可以解决这个问题，只是由于缺乏支持而无法付诸实践。
> HTTP/2 一举解决了所有这些低效的问题：浏览器可以在发现资源时立即分派请求，指定每个流的优先级，让服务器决定最优的响应次序。这样请求就不必排队了，既节省了时间，也最大限度地利用了每个连接。

HTTP/2 没有规定处理优先级的具体算法，只是提供了一种赋予数据优先级的机制，而且要求客户端与服务器必须能够交换这些数据。这样一来，优先值作为提示信息，对应的次序排定策略可能因客户端或服务器的实现而不同：客户端应该明确指定优先值，服务器应该根据该值处理和交付数据。

在这个规定之下，尽管你可能无法控制客户端发送的优先值，但或许你可以控制服务器。因此，在选择HTTP/2 服务器时，可以多留点心！为说明这一点，考虑下面几个问题。

- 如果服务器对所有优先值视而不见怎么办？
- 高优先值的流一定优先处理吗？
- 是否存在不同优先级的流应该交错的情况？

如果服务器不理解所有优先值，那么可能会导致应用响应变慢：浏览器明明在等关键的CSS和JavaScript，服务器却在发送图片，从而造成渲染阻塞。不过，规定严格的优先级次序也可能带来次优的结果，因为这可能又会引入队首阻塞问题，即某个高优先级的慢请求会不必要地阻塞其他资源的交付。

服务器可以而且应该交错发送不同优先级别的帧。只要可能，高优先级流都应该优先，包括分配处理资源和客户端与服务器间的带宽。不过，为了最高效地利用底层连接，不同优先级的混合也是必需的。

有了新的分帧机制后，HTTP/2 不再依赖多个TCP连接去实现多流并行了。现在，每个数据流都拆分成很多帧，而这些帧可以交错，还可以分别优先级。于是，所有HTTP/2 连接都是持久化的，而且客户端与服务器之间也只需要一个连接即可。

> 实验表明，客户端使用更少的连接肯定可以降低延迟时间。HTTP/2 发送的总分组数量比HTTP差不多要少40%。而服务器处理大量并发连接的情况也变成了可伸缩性问题，因为HTTP/2 减轻了这个负担。——HTTP/2.0 Draft 2”

每个来源一个连接显著减少了相关的资源占用：连接路径上的套接字管理工作量少了，内存占用少了，连接吞吐量大了。此外，从上到下所有层面上也都获得了相应的好处：

- 所有数据流的优先次序始终如一；
- 压缩上下文单一使得压缩效果更好；
- 由于TCP连接减少而使网络拥塞状况得以改观；
- 慢启动时间减少，拥塞和丢包恢复速度更快。

大多数HTTP连接的时间都很短，而且是突发性的，但TCP只在长时间连接传输大块数据时效率才最高。HTTP/2 通过让所有数据流共用同一个连接，可以更有效地使用TCP连接。

HTTP/2 不仅能够减少网络延迟，还有助于提高吞吐量和降低运营成本！

等一等，我听你说了一大堆每个来源一个TCP连接的好处，难道它就一点坏处都没有吗？有，当然有。

- 虽然消除了HTTP队首阻塞现象，但TCP层次上仍然存在队首阻塞（参见2.4节“队首阻塞”）；
- 如果TCP窗口缩放被禁用，那带宽延迟积效应可能会限制连接的吞吐量；

2. 2017年秋季校招前端面经（百度，腾讯，网易，华为，乐视等）(8)
3. js实现A*寻路算法(8)
4. 滑动窗口算法基本原理与实践(6)
5. 手把手教你实现一个完整的 Promise(5)
6. 各个公司前端笔试题回顾(5)
7. JavaScript 模板引擎实现原理解析(4)
8. 微信公众号开发获取用户信息(4)
9. Android HandlerThread 详解(3)
10. 深入理解 Java 动态代理机制(3)
11. 通过bootstrap来学习less(3)
12. 深度优先搜索原理与实践(2)
13. ThreadPoolExcutor 原理探究(2)
14. Java 虚拟机结构(2)
15. Android Studio 运行java程序(2)

推荐排行榜

1. 总结一年来的前端学习心得(15)
2. synchronized(this) 与synchronized(class) 之间的区别(11)
3. 手把手教你实现一个完整的 Promise(10)
4. 滑动窗口算法基本原理与实践(9)
5. 各个公司前端笔试题回顾(7)
6. 从你输入网址，到看到网页——详解中间发生的过程(6)
7. 前端面试笔试知识汇总1（含答案）(6)
8. JavaScript 模板引擎实现原理解析(5)
9. 2017年秋季校招前端面经（百度，腾讯，网易，华为，乐视等）(5)
10. 漫谈 HTTP 连接(4)
11. 快速读懂 HTTP/3 协议(3)
12. 漫谈 HTTP 性能优化(3)
13. 深入浅出 HTTPS (详解版)(3)
14. 广度优先搜索原理与实践(3)
15. ThreadPoolExcutor 原理探究(3)

- 丢包时，TCP拥塞窗口会缩小。

上述每一点都可能对HTTP/2 连接的吞吐量和延迟性能造成不利影响。然而，除了这些局限性之外，实验表明一个TCP连接仍然是HTTP/2 基础上的最佳部署策略：

目前为止的测试表明，压缩和优先级排定带来的性能提升，已经超过了队首阻塞（特别是丢包情况下）造成的负面效果。

流量控制

在同一个TCP连接上传输多个数据流，就意味着要共享带宽。标定数据流的优先级有助于按序交付，但只有优先级还不足以确定多个数据流或多个连接间的资源分配。为解决这个问题，HTTP/2 为数据流和连接的流量控制提供了一个简单的机制：

- 流量控制基于每一跳进行，而非端到端的控制；
- 流量控制基于窗口更新帧进行，即接收方广播自己准备接收某个数据流的多少字节，以及对整个连接要接收多少字节；
- 流量控制窗口大小通过WINDOW_UPDATE 帧更新，这个字段指定了流ID和窗口大小递增值；
- 流量控制有方向性，即接收方可能根据自己的情况为每个流乃至整个连接设置任意窗口大小；
- 流量控制可以由接收方禁用，包括针对个别的流和针对整个连接。

HTTP/2 连接建立之后，客户端与服务器交换SETTINGS 帧，目的是设置双向的流量控制窗口大小。除此之外，任何一端都可以选择禁用个别流或整个连接的流量控制。

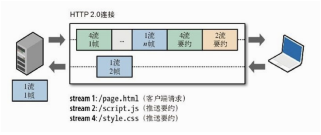
上面这个列表是不是让你想起了TCP流量控制？应该是，这两个机制实际上是一样的。然而，由于TCP流量控制不能对同一条HTTP/2 连接内的多个流实施差异化策略，因此光有它自己是不够的。这正是HTTP/2 流量控制机制出台的原因。

HTTP/2 标准没有规定任何特定的算法、值，或者什么时候发送WINDOW_UPDATE 帧。因此，实现可以选择自己的算法以匹配自己的应用场景，从而求得最佳性能。

> 优先级可以决定交付次序，而流量控制则可以控制HTTP/2 连接中每个流占用的资源：接收方可以针对特定的流广播较低的窗口大小，以限制它的传输速度。

服务器推送

HTTP/2 新增的一个强大的新功能，就是服务器可以对一个客户端请求发送多个响应。换句话说，除了对最初请求的响应外，服务器还可以额外向客户端推送资源（如下图所示），而无需客户端明确地请求。



建立HTTP/2 连接后，客户端与服务器交换SETTINGS 帧，借此可以限定双向并发的流的最大数量。因此，客户端可以限定推送流的数量，或者通过把这个值设置为0而完全禁用服务器推送。

> 为什么需要这样一个机制呢？
> 通常的Web应用都由几十个资源组成，客户端需要分析服务器提供的文档才能逐个找到它们。那为什么不让服务器提前就把这些资源推送给客户端，从而减少额外的时间延迟呢？服务器已经知道客户端下一步要请求什么资源了，这时候服务器推送即可派上用场。事实上，如果你在网页里嵌入过CSS、JavaScript，或者通过数据URI嵌入过其他资源，那你就已经亲身体验过服务器推送了。

把资源直接插入到文档中，就是把资源直接推送给客户端，而无需客户端请求。在HTTP/2 中，唯一的不同就是可以把这个过程从应用中拿出来，放到HTTP协议本身来实现。而且还带来了如下好处：

- 客户端可以缓存推送过来的资源；
- 客户端可以拒绝推送过来的资源；
- 推送资源可以由不同的页面共享；
- 服务器可以按照优先级推送资源。

所有推送的资源都遵守同源策略。换句话说，服务器不能随便将第三方资源推送给客户端，而必须是经过双方确认才行。

有了服务器推送后，HTTP 1.x时代的大多数插入或嵌入资源的做法基本上也就过时了。唯一有必要直接在网页中插入资源的情况，就是该资源只供那一个网页使用，而且编码代价不大；除此之外，所有应用都应该使用HTTP/2 服务器推送。

关于 HTTP 系列文章：

- **HTTP 概述**
- **TCP 三次握手和四次挥手图解（有限状态机）**
- **从你输入网址，到看到网页——详解中间发生的过程**
- **深入浅出 HTTPS (详解版)**
- **漫谈 HTTP 连接**
- **漫谈 HTTP 性能优化**
- **HTTP 报文格式简介**
- **深入浅出：HTTP/2**

参考文章

- **https://hpbn.co/http2/**
- Web性能权威指南
-

树林美丽、幽暗而深邃，但我有诺言尚待实现，还要奔行百里方可沉睡。 -- 罗伯特·弗罗斯特

标签： 网络

huansky
粉丝 - 295 关注 - 35
+加关注

« 上一篇： Okio源码分析
» 下一篇： 快速读懂 HTTP/3 协议

posted @ 2021-02-18 23:14 huansky 阅读(1026) 评论(0) 编辑 收藏 举报