二

19 我老了，让我儿子来吧 - HTTP2

上一小节我们讲了HTTP1的缺点以及简单的介绍了一下HTTP2。 这一小节，让我们来认识HTTP2多一点。

多向请求和响应(解决了http1.x的队列阻塞)

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 2.0中新的二进制分帧层突破了这些限制。客户端和服务器可以把HTTP消息分解为互不依赖的帧，然后乱序发送，最后再在另一端把它们重新组合起来。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 2.0最重要的一项增强。事实上，这个机制在整个Web技术栈中引发了一系列连锁反应，从而带来巨大的性能提升。

- 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。

HTTP2.0多路复用有多好?

HTTP性能优化的关键并不在于高带宽, 而是低延迟。TCP连接会随着时间进行自我「调整」，起初会限制连接的最大速度，如果数据成功传输，会随着时间的推移提高传输的速度（还记得我们讲的TCP拥塞机制吗）。这种调整则被称为TCP慢启动。由于这种原因，让原本就具有突发性和短时性的HTTP连接变的十分低效。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，让高带宽也能真正的服务于 HTTP的性能提升。

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的31次方-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。

每个来源一个连接

有了新的分帧机制后，HTTP 2.0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 2.0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。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 2.0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。

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 2.0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流。

上面这个列表是不是让你想起了TCP流量控制? 如果是的话，恭喜你, 回答正确。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 2.0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 2.0流量控制机制出台的原因。

HTTP 2.0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。

服务器推送

这是HTTP 2.0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而无需客户端明确地请求。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。HTTPS协商过程中有一个环节会使用ALPN（Application Layer Protocol Negotiation）发现和协商HTTP 2.0的支持情况。减少网络延迟是HTTP 2.0的关键条件，因此在建立HTTPS连接时一定会用到ALPN协商。

Header 压缩

在HTTP/1中，我们使用文本的形式传输header，在header携带cookie的情况下，可能每次都需要重复传输几百到几千的字节。为了减少这块的资源消耗并提升性能，HTTP/2对这些首部采取了压缩策略。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。

升级

那怎么升级呢?

前文说了HTTP2.0其实可以支持非HTTPS的，但是现在主流的浏览器像chrome，firefox表示还是只支持基于 TLS 部署的HTTP2.0协议，所以要想升级成HTTP2.0还是先升级HTTPS为好。 以nginx为例

nginx官方提供了两种方法,第一种是升级操作系统,第二种是从源码编译新版本的nginx，我们用第二种方法.当前nginx最新的稳定版本是1.18,在服务器上执行以下命令:

```
wget http://nginx.org/download/nginx-1.18.0.tar.gz # 下载

tar -zxvfnginx-1.18.0.tar.gz # 解压

cd nginx-1.18.0

./configure # 确认系统环境，生成make文件

make # 编译

sudo make install #安装

复制
```

configure的时候后面可以带参数，参数可以用原先老版本nginx的参数，包括安装路径之类的，这个可以通过执行nginx -V得到，使得新nginx的配置和老nginx一样。如果configure提示缺一些库的话就相应地做些安装，基本上就是它提示的库后面带上devel，如以下提示:

```
./configure: error: the Google perftools module requires the Google perftools

library. You can either do not enable the module or install the library.

sudo yum install gperftools-devel

然后添加nginx配置，原本https的listen为:

listen 443 ssl;

现在在后面加上http2:
```

```
listen 443 ssl http2;
```

然后把nginx关了再开一下(因为新安装了一个nginx，要先关一下再开)。

复制

展望未来 - HTTP/3

虽然HTTP/2解决了很多之前旧版本的问题，但是它还是存在一个巨大的问题，主要是底层支撑的TCP协议造成的。前面提到 HTTP/2 使用了多路复用，一般来说同一域名下只需要使用一个 TCP 连接。但当这个连接中出现了丢包的情况，那就会导致 HTTP/2 的表现情况反倒不如 HTTP/1了。

因为在出现丢包的情况下，整个TCP都要开始等待重传，也就导致了后面的所有数据都被阻塞了。但是对于 HTTP/1.1 来说,可以开启多个 TCP 连接,出现这种情况反到只会影响其中一个连接,剩余的 TCP 连接还可以正常传输数据。

那么可能就会有人考虑到去修改TCP 协议，其实这已经是一件不可能完成的任务了。因为TCP存在的时间实在太长，已经充斥在各种设备中，并且这个协议是由操作系统实现的，更新起来不大现实。

基于这个原因，Google就另起炉灶搞了一个基于UDP协议的QUIC协议，并且使用在了HTTP/3上，HTTP/3之前名为HTTP-over-QUIC，从这个名字中我们也可以发现，HTTP/3最大的改造就是使用了QUIC。

因为HTTP/3离我们相对还远一点。我们就不多说了。

上一页 下一页