

18 | 如何通过gRPC实现高效远程过程调用？

2020-06-15 陶辉

系统性能调优必知必会

[进入课程 >](#)



讲述：陶辉

时长 12:35 大小 11.53M



你好，我是陶辉。

这一讲我们将以一个实战案例，基于前两讲提到的 HTTP/2 和 ProtoBuf 协议，看看 gRPC 如何将结构化消息编码为网络报文。

直接操作网络协议编程，容易让业务开发过程陷入复杂的网络处理细节。RPC 框架以编程语言中的本地函数调用形式，向应用开发者提供网络访问能力，这既封装了消息的编解码，也通过线程模型封装了多路复用，对业务开发很友好。



其中，Google 推出的 gRPC 是性能最好的 RPC 框架之一，它支持 Java、Javascript、Python、GoLang、C++、Object-C、Android、Ruby 等多种编程语言，还支持安全验

证等特性，得到了广泛的应用，比如微服务中的 Envoy、分布式机器学习中的 Tensorflow，甚至华为去年推出重构互联网的 New IP 技术，都使用了 gRPC 框架。

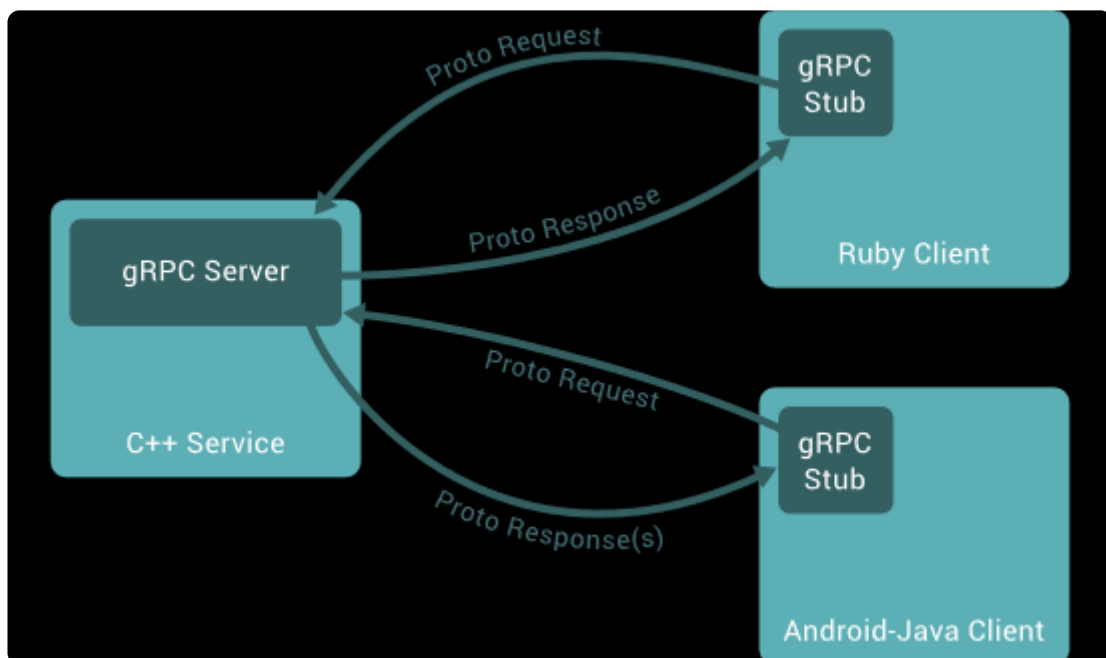
然而，网络上教你使用 gRPC 框架的教程很多，却很少去谈 gRPC 是如何编码消息的。这样，一旦在大型分布式系统中出现疑难杂症，需要通过网络报文去定位问题发生在哪个系统、主机、进程中时，你就会毫无头绪。即使我们掌握了 HTTP/2 和 Protobuf 协议，但若不清楚 gRPC 的编码规则，还是无法分析抓取到的 gRPC 报文。而且，gRPC 支持单向、双向的流式 RPC 调用，编程相对复杂一些，定位流式 RPC 调用引发的 bug 时，更需要我们掌握 gRPC 的编码原理。

这一讲，我就将以 gRPC 官方提供的 example: [data_transmission](#) 为例，介绍 gRPC 的编码流程。在这一过程中，会顺带回顾 HTTP/2 和 Protobuf 协议，加深你对它们的理解。虽然这个示例使用的是 Python 语言，但基于 gRPC 框架，你可以轻松地将它们转换为其他编程语言。

如何使用 gRPC 框架实现远程调用？

我们先来简单地看下 gRPC 框架到底是什么。RPC 的全称是 Remote Procedure Call，即远程过程调用，它通过本地函数调用，封装了跨网络、跨平台、跨语言的服务访问，大大简化了应用层编程。其中，函数的入参是请求，而函数的返回值则是响应。

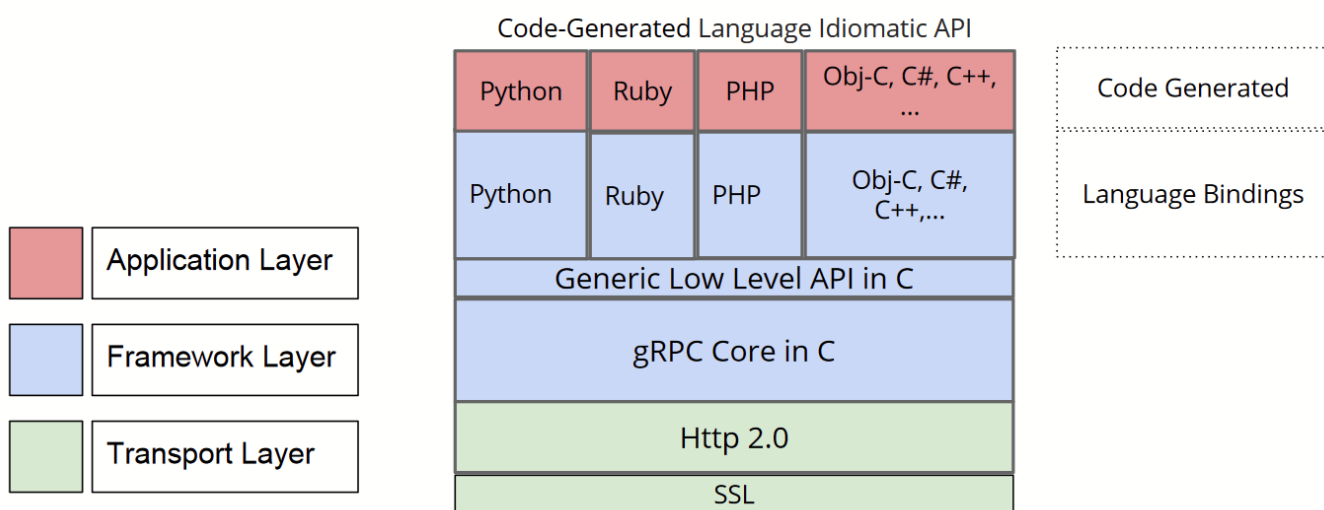
gRPC 就是一种 RPC 框架，在你定义好消息格式后，针对你选择的编程语言，gRPC 为客户端生成发起 RPC 请求的 Stub 类，以及为服务器生成处理 RPC 请求的 Service 类（服务器只需要继承、实现类中处理请求的函数即可）。如下图所示，很明显，gRPC 主要服务于面向对象的编程语言。



图片来源: <https://grpc.io/>

gRPC 支持 QUIC、HTTP/1 等多种协议，但鉴于 HTTP/2 协议性能好，应用场景又广泛，因此 HTTP/2 是 gRPC 的默认传输协议。gRPC 也支持 JSON 编码格式，但在忽略编码细节的 RPC 调用中，高效的 Protobuf 才是最佳选择！因此，这一讲仅基于 HTTP/2 和 Protobuf，介绍 gRPC 的用法。

gRPC 可以简单地分为三层，包括底层的数据传输层，中间的框架层（框架层又包括 C 语言实现的核心功能，以及上层的编程语言框架），以及最上层由框架层自动生成的 Stub 和 Service 类，如下图所示：




图片来源: <https://platformlab.stanford.edu/Seminar%20Talks/gRPC.pdf>

接下来我们以官网上的 [data_transmisstion](#) 为例，先看看如何使用 gRPC。

构建 Python 语言的 gRPC 环境很简单，你可以参考官网上的 [QuickStart](#)。


使用 gRPC 前，先要根据 Protobuf 语法，编写定义消息格式的 proto 文件。在这个例子中只有 1 种请求和 1 种响应，且它们很相似，各含有 1 个整型数字和 1 个字符串，如下所示：

 复制代码

```
1 package demo;
2
3 message Request {
4     int64 client_id = 1;
5     string request_data = 2;
6 }
7
8 message Response {
9     int64 server_id = 1;
10    string response_data = 2;
11 }
```

请注意，这里的包名 demo 以及字段序号 1、2，都与后续的 gRPC 报文分析相关。

接着定义 service，所有的 RPC 方法都要放置在 service 中，这里将它取名为 GRPCDemo。GRPCDemo 中有 4 个方法，后面 3 个流式访问的例子我们呆会再谈，先来看简单的一元访问模式 SimpleMethod 方法，它定义了 1 个请求对应 1 个响应的访问形式。其中，SimpleMethod 的参数 Request 是请求，返回值 Response 是响应。注意，分析报文时会用到这里的类名 GRPCDemo 以及方法名 SimpleMethod。

 复制代码

```
1 service GRPCDemo {
2     rpc SimpleMethod (Request) returns (Response);
3 }
```

用 grpc_tools 中的 protoc 命令，就可以针对刚刚定义的 service，生成含有 GRPCDemoStub 类和 GRPCDemoServicer 类的 demo_pb2_grpc.py 文件（实际上还包括完成 Protobuf 编解码的 demo_pb2.py），应用层将使用这两个类完成 RPC 访问。我简化了官网上的 Python 客户端代码，如下所示：

```
1 with grpc.insecure_channel("localhost:23333") as channel:
2     stub = demo_pb2_grpc.GRPCDemoStub(channel)
3     request = demo_pb2.Request(client_id=1,
4                               request_data="called by Python client")
5     response = stub.SimpleMethod(request)
```

[复制代码](#)

示例中客户端与服务器都在同一台机器上，通过 23333 端口访问。客户端通过 stub 对象的 SimpleMethod 方法完成了 RPC 访问。而服务器端的实现也很简单，只需要实现 GRPCDemoServicer 父类的 SimpleMethod 方法，返回 response 响应即可：

```
1 class DemoServer(demo_pb2_grpc.GRPCDemoServicer):
2     def SimpleMethod(self, request, context):
3         response = demo_pb2.Response(
4             server_id=1,
5             response_data="Python server SimpleMethod Ok!!!!")
6         return response
```

[复制代码](#)

可见，gRPC 的开发效率非常高！接下来我们分析这次 RPC 调用中，消息是怎样编码的。

gRPC 消息是如何编码的？

定位复杂的网络问题，都需要抓取、分析网络报文。如果你在 Windows 上抓取网络报文，可以使用 Wireshark 工具（可参考 [《Web 协议详解与抓包实战》第 37 课](#)），如果在 Linux 上抓包可以使用 tcpdump 工具（可参考 [第 87 课](#)）。当然，你也可以从 [这里](#) 下载我抓取好的网络报文，用 Wireshark 打开它。需要注意，23333 不是 HTTP 常用的 80 或者 443 端口，所以 Wireshark 默认不会把它解析为 HTTP/2 协议。你需要鼠标右键点击报文，选择“解码为”（Decode as），将 23333 端口的报文设置为 HTTP/2 解码器，如下图所示：

	Protocol	Length	Info
建立TCP连接	TCP	94	38050 → 23333 [SYN] Seq=0 Win=43690 Len=0 MSS=65476 SACK_PERM=1 TSval=313819282 TSecr=313819282
	TCP	94	23333 → 38050 [SYN, ACK] Seq=0 Ack=1 Win=43690 Len=0 MSS=65476 SACK_PERM=1 TSval=313819282 TSecr=313819282
	TCP	86	38050 → 23333 [ACK] Seq=1 Ack=1 Win=43776 Len=0 TSval=313819282 TSecr=313819282
建立HTTP2会话	HTTP2	149	SETTINGS[0], WINDOW_UPDATE[0], PING[0]
	TCP	86	38050 → 23333 [ACK] Seq=1 Ack=64 Win=43776 Len=0 TSval=313819282 TSecr=313819282
	HTTP2	185	Magic, SETTINGS[0], WINDOW_UPDATE[0], PING[0]
	TCP	86	23333 → 38050 [ACK] Seq=64 Ack=100 Win=43776 Len=0 TSval=313819282 TSecr=313819282
请求	HTTP2	112	PING[0], SETTINGS[0]
	GRPC	420	HEADERS[1]: POST /demo.GRPCDemo/SimpleMethod, WINDOW_UPDATE[1], DATA[1] (GRPC)
响应	HTTP2	112	PING[0], SETTINGS[0]
	GRPC	305	HEADERS[1]: 200 OK, DATA[1], HEADERS[1] (GRPC) (PROTOBUF), WINDOW_UPDATE[0]

图中蓝色方框中，TCP 连接的建立过程请参见 [🔗 \[第 9 讲\]](#)，而 HTTP/2 会话的建立可参见 [🔗 《Web 协议详解与抓包实战》第 52 课](#)（还是比较简单的，如果你都清楚就可以直接略过）。我们重点看红色方框中的 gRPC 请求与响应，点开请求，可以看到下图中的信息：

HTTP头部	> Header: :scheme: http
	> Header: :method: POST
	> Header: :authority: localhost:23333
Length-Prefixed Message	> Header: :path: /demo.GRPCDemo/SimpleMethod
	> Header: te: trailers
	> Header: content-type: application/grpc
Protobuf消息	> Header: user-agent: grpc-python/1.28.1 grpc-c/9.0.0 (mar
	> Header: grpc-accept-encoding: identity,deflate,gzip
	> Header: accept-encoding: identity,gzip
	> Stream: WINDOW_UPDATE, Stream ID: 1, Length 4
	> Stream: DATA, Stream ID: 1, Length 32
	> GRPC Message: /demo.GRPCDemo/SimpleMethod, Request
	Compressed Flag: Not Compressed (0)
	Message Length: 27
	Message Data: 27 bytes
	> Protocol Buffers: application/grpc,/demo.GRPCDemo/SimpleMe
	> Field[1], 1 (uint32)
	.000 1... = Field Number: 1
000 = Wire Type: varint (0)
	> Value: 01
	Uint32: 1
	> Field[2]
	.001 0... = Field Number: 2
010 = Wire Type: Length-delimited (2)
	Value Length: 23
	Value: 63616c6c656420627920507974686f6e20636c69656e74


先来分析蓝色方框中的 HTTP/2 头部。请求中有 2 个关键的 HTTP 头部，path 和 content-type，它们决定了 RPC 方法和具体的消息编码格式。path 的值为 “/demo.GRPCDemo/SimpleMethod”，通过 “/ 包名. 服务名 / 方法名” 的形式确定了 RPC 方法。content-type 的值为 “application/grpc”，确定消息编码使用 Protobuf 格式。如果你对其他头部的含义感兴趣，可以看下这个 [文档](#)，注意这里使用了 ABNF 元数据定义语言（如果你还不了解 ABNF，可以看下 [《Web 协议详解与抓包实战》第 4 课](#)）。

HTTP/2 包体并不会直接存放 Protobuf 消息，而是先要添加 5 个字节的 Length-Prefixed Message 头部，其中用 4 个字节明确 Protobuf 消息的长度（1 个字节表示消息是否做过压缩），即上图中的桔色方框。为什么要多此一举呢？这是因为，gRPC 支持流式消息，即在 HTTP/2 的 1 条 Stream 中，通过 DATA 帧发送多个 gRPC 消息，而 Length-Prefixed Message 就可以将不同的消息分离开。关于流式消息，我们在介绍完一元模式后，再加以分析。

最后分析 Protobuf 消息，这里仅以 client_id 字段为例，对上一讲的内容做个回顾。在 proto 文件中 client_id 字段的序号为 1，因此首字节 00001000 中前 5 位表示序号为 1 的 client_id 字段，后 3 位表示字段的值类型是 varint 格式的数字，因此随后的字节 00000001 表示字段值为 1。序号为 2 的 request_data 字段请你结合上一讲的内容，试着做一下解析，看看字符串 “called by Python client” 是怎样编码的。

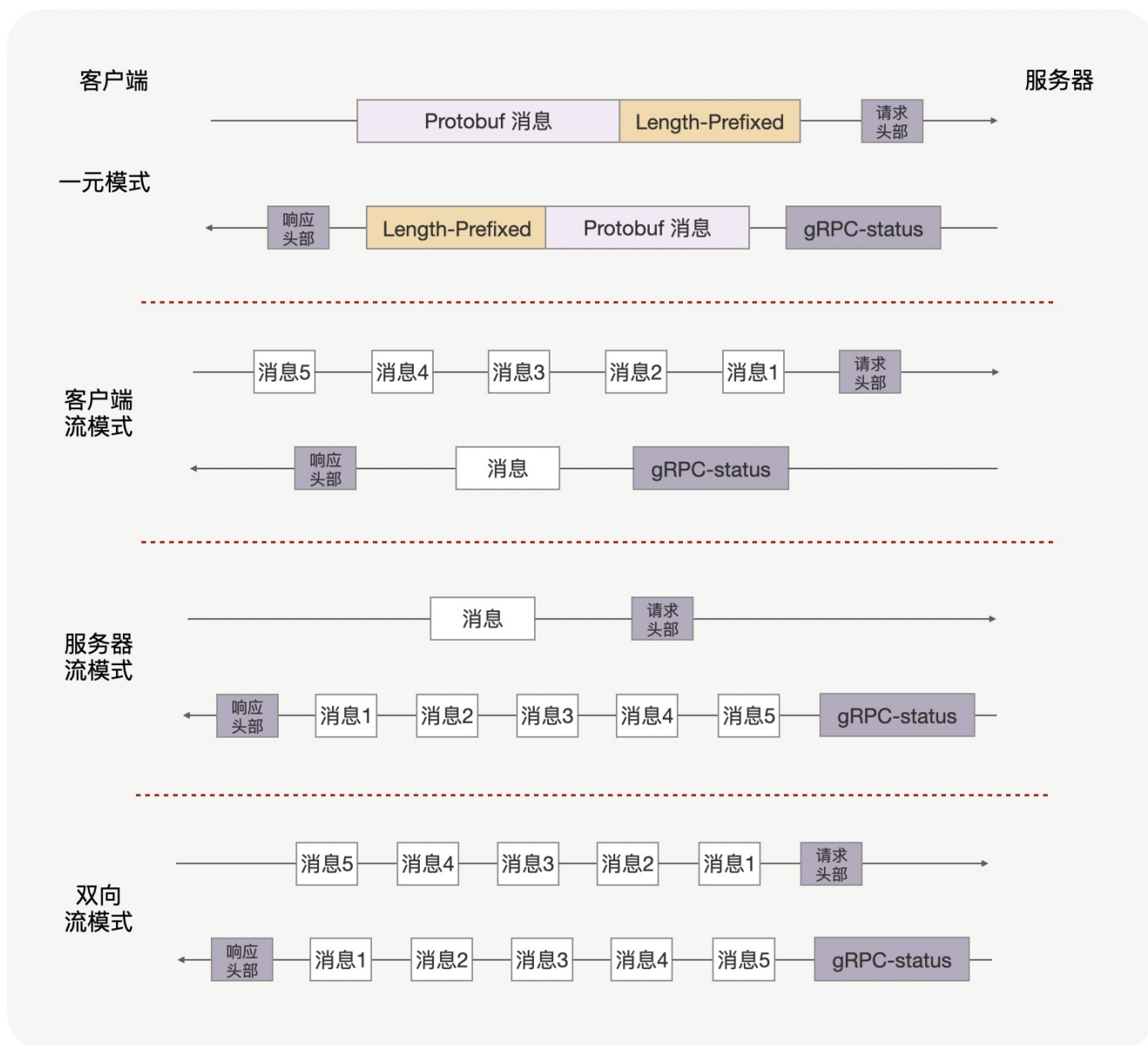
再来看服务器发回的响应，点开 Wireshark 中的响应报文后如下图所示：

所谓流模式，是指 RPC 通讯的一方可以在 1 次 RPC 调用中，持续不断地发送消息，这对订阅、推送等场景很有用。流模式共有 3 种类型，包括客户端流模式、服务器端流模式，以及两端双向流模式。在 [data_transmission](#) 官方示例中，对这 3 种流模式都定义了 RPC 方法，如下所示：

 复制代码

```
1 service GRPCDemo {
2     rpc ClientStreamingMethod (stream Request) returns Response;
3
4     rpc ServerStreamingMethod (Request) returns (stream Response);
5
6     rpc BidirectionalStreamingMethod (stream Request) returns (stream Response);
7 }
```

不同的编程语言处理流模式的代码很不一样，这里就不一一列举了，但通讯层的流模式消息编码是一样的，而且很简单。这是因为，HTTP/2 协议中每个 Stream 就是天然的 1 次 RPC 请求，每个 RPC 消息又已经通过 Length-Prefixed Message 头部确立了边界，这样，在 Stream 中连续地发送多个 DATA 帧，就可以实现流模式 RPC。我画了一张示意图，你可以对照它理解抓取到的流模式报文。



小结

这一讲介绍了 gRPC 怎样使用 HTTP/2 和 Protobuf 协议编码消息。

在定义好消息格式，以及 service 类中的 RPC 方法后，gRPC 框架可以为编程语言生成 Stub 和 Service 类，而类中的方法就封装了网络调用，其中方法的参数是请求，而方法的返回值则是响应。

发起 RPC 调用后，我们可以这么分析抓取到的网络报文。首先，分析应用层最外层的 HTTP/2 帧，根据 Stream ID 找出一次 RPC 调用。客户端 HTTP 头部的 path 字段指明了 service 和 RPC 方法名，而 content-type 则指明了消息的编码格式。服务器端的 HTTP 头部被分成 2 次发送，其中 DATA 帧发送完毕后，才会发送 grpc-status 头部，这样可以明确最终的错误码。

其次，分析包体时，可以通过 Stream 中 Length-Prefixed Message 头部，确认 DATA 帧中含有多少个消息，因此可以确定这是一元模式还是流式调用。在 Length-Prefixed Message 头部后，则是 Protobuf 消息，按照上一讲的内容进行分析即可。

思考题

最后，留给你一道练习题。gRPC 默认并不会压缩字符串，你可以通过在获取 channel 对象时加入 `grpc.default_compression_algorithm` 参数的形式，要求 gRPC 压缩消息，此时 Length-Prefixed Message 中 1 个字节的压缩位将会由 0 变为 1。你可以观察下执行压缩后的 gRPC 消息有何不同，欢迎你在留言区与大家一起探讨。

感谢阅读，如果你觉得这节课对你有一些启发，也欢迎把它分享给你的朋友。

更多课程推荐

MySQL 实战 45 讲

从原理到实战，丁奇带你搞懂 MySQL

林晓斌

网名丁奇
前阿里资深技术专家



涨价倒计时 🕒

今日秒杀 **¥79**，6月13日涨价至 **¥129**

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 17 | Protobuf是如何进一步提高编码效率的？

下一篇 期中考试 | 行至半程，你的收获如何呢？

精选留言 (4)

写留言



唐朝首都

2020-06-16

基本原理+工具使用，理论结合实践才能把这些协议搞清楚呀！

作者回复: 没错！



小美

2020-06-15

从http1.1 升级到 http2.0 要哪些地方改造呢，就改个versionCode 吗



远方的风

2020-06-15

老师的广度和深度是如何练成的呢，是看书，还是实际工作中都有涉及？一开始搞web开发的，对底层这些研究的就没这么深了

展开 ∨

作者回复: 你好远方的风，不用着急，能问出这问题，就表示你在朝这个方向进发，它需要时间。我建议这么加快速度：在实践中遇到问题时，多问自己几个为什么，就会顺着捋下来。在这个过程中，有些超过自己边界的问题，必须要增加广度，才能搞明白“为什么”。
比如做gRPC应用开发时，如果想搞明白gRPC为什么可以生成stub代码，就得先增加广度去学一学编译原理



安排

2020-06-15

client流模式中是不是每一个rpc请求消息的第一个data帧中才有Length-Prefixed Message，然后下一个data帧只有protobuf数据，直到这个rpc请求消息发完。然后同一个stream上的下一个rpc消息的第一个data帧再加入Length-Prefixed Message。以此类推。

展开 ∨

1



