给MySQL写一个代理网关

由于一个突发的需求，年前年后大部分时间都在mysql-proxy这么个项目上折腾；现在一期功能算是上线了，趁这个机会简单总结一下

由于MySQL通信是通过MySQL协议交互的，所以主要其实就是折腾MySQL协议

项目背景

先简单说下为什么需要做mysql-proxy，开发平时查看数据库，是通过 jumpserver 连接到现网的灾备库查看，灾备库和现网业务库的数据内容是一样的，里面有很多敏感数据，比如手机号，身份证等，所以存在着数据安全的问题

常规的处理思路是，对所有敏感数据的查询都进行申请和审批，这样固然能解决数据安全的问题，但是让开发查数据变得麻烦了许多，降低了排查问题的效率，同时要分析出所有包含敏感数据的库表也是一件麻烦事

综合考虑，我们决定增加一个代理，直接对敏感数据进行脱敏，这样开发的使用习惯和之前一样，如果查询到的数据集中包含敏感数据，就会被其他的符号遮掩

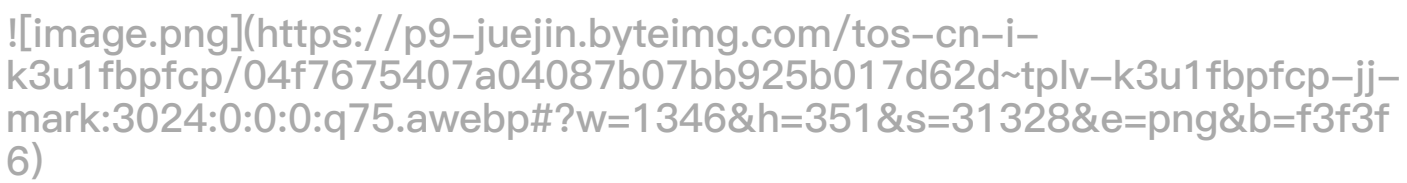
当然某些场景下，开发还是需要知道数据的真实值，这时候就必须得进行申请了，为了数据的安全，一些必要的效率折损，还是可以接受的

实现思路

要实现MySQL代理并进行数据脱敏，分为以下两步

1. 模拟mysql client 与mysql server 的双端交互，成为中间人

2. pick up出查询请求，篡改返回数据包



MySQL建连过程分析

平时连接MySQL，我们会执行类似下面这行命令

```
mysql -u root -h127.0.0.1 -p123qwe -P3306 --ssl-mode=DISABLED
```

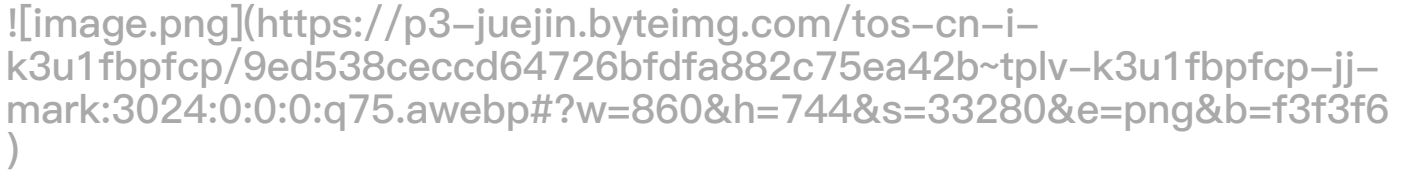
执行这行命令会发生什么?

1. 客户端向服务端建立TCP连接，也就是三次握手

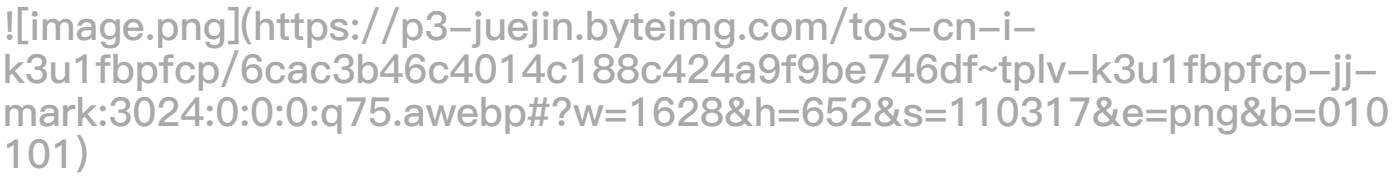
2. 服务端返回InitialHandshake消息包，其中包括服务端版本号和一些功能性标记（比如是否丢弃EOF包等）

3. 客户端发送HandshakeResponse消息包，其中包括功能性标记、连接使用的用户名密码，字符集和数据库名等

4. 服务端校验通过后，返回ok消息，即完成连接的建立

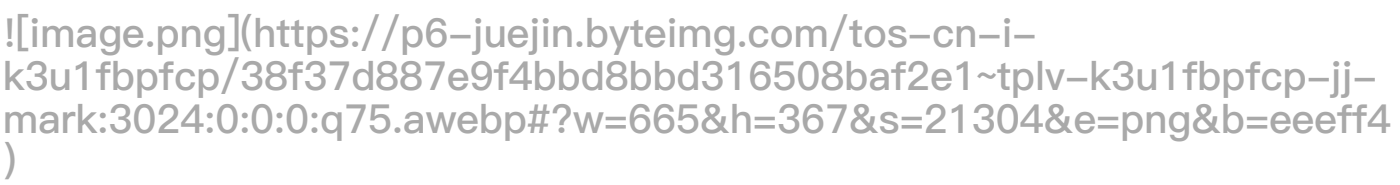


完成握手之后就进入了下图中的交互模式，红框的版本信息就是此次交换得到的数据



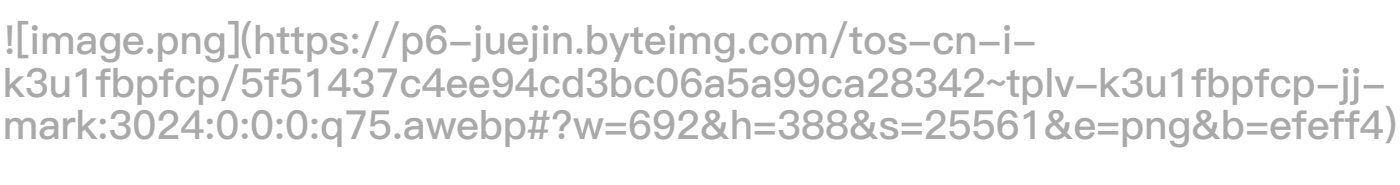
MySQL代理建连过程分析

代理处于client和server中间，最简单的做法就是直接转发



如果只有一个client和一个server，这么做当然是没问题的

但实际场景中，我们有多个数据库实例，proxy应该代理到哪个数据库实例，是由client端决定的



看来proxy不能偷懒了，它必须做到以下几件事

1．识别到client端想连接哪个server端

2．连接到对应的server端

要做到这两件事，首先要分析，不同的数据库实例是靠什么区分的?

显然是ip:port，实例的ip是不同的，端口都是3306

但是client端要走proxy，ip自然要填proxy的ip，proxy的ip肯定只有一个，此

时就无法通过ip来进行区分，必须另寻他径

回头看看MySQL的连接过程，TCP三次握手肯定不能携带任何信息，只能想办法在HandshakeResponse包里携带标识

经过对TCP协议包的分析，TCP包确实留置了option段来放一些自定义信息，但是这样需要修改client端来做适配，这显然不合适，不同的开发客户端各有千秋，navicat，datagrip等等

所以只能对open的信息做手脚，观察MySQL的连接命令，发现可以在用户名上动手脚，所以将数据库实例id的拼接在用户名上

```
mysql -u {instance_id}.root -h127.0.0.1 -p123qwe -P3306 --ssl-mode=DISABLED
```

proxy在拿到{instance_id}.root后，进行拆分，由instance_id得到需要连接的数据库实例，再进行数据库实例的连接，所以代理的建连过程就变成下图所示

注：proxy的配置中，存有instance_id对应的ip:port，这样拿到instance_id就等于拿到的实例的地址

TODO: wireshark抓报示例整个过程

MySQL鉴权过程分析

先看下一个常规的鉴权过程

1．客户端发送账号密码

2．服务端校验，并返回结果

这里往往会消耗一次来回的消息交换

为了减少建立连接过程种消息交换的次数，MySQL协议有一个鉴权的快速通道 FastPath

在服务端发送InitialHandshake消息时，会先默认猜一个AuthMethod，并随机生成8字节或者更长的challenge payload，放在InitialHandshake消息中一起发给客户端

为什么说是“猜”呢，因为不同用户可能设置不同的AuthMethod，然而在这一阶段，服务端还不知道要连接的用户是哪一个，自然不知道正确的AuthMethod应该是什么了

AuthMethod 比较常见的有 mysql_native_password，caching_sha2_password，主要区别在于hash算法不同

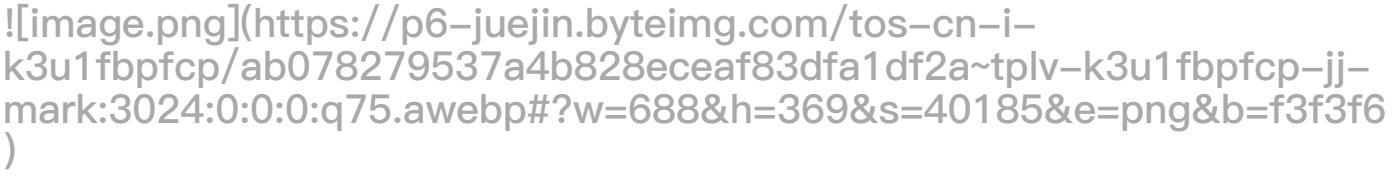
challenge payload是一串随机码，每一次连接都会不同，客户端会用这一串随机码给密码加密

客户端根据AuthMethod定义的方法对密码+payload加以计算，计算结果连同AuthMethod一起放在HandshakeResponse里一起发给服务端

如果服务端读取对应的用户表之后，发现AuthMethod跟猜测的一致，那么就可以直接验证客户端的计算结果了，成功后直接返回Ok，这样就完成连接建立了，否则，服务端需要发送AuthMethodSwitchRequest来重新进行鉴权

AuthMethod的特殊处理

由于MySQL协议中鉴权FastPath的存在，这个过程是有问题的：客户端收到的加密随机码是一开始由proxy生成的，它跟mysql-server发给proxy的显然不一致，这将导致mysql-server在收到HandshakeResponse后校验失败报错



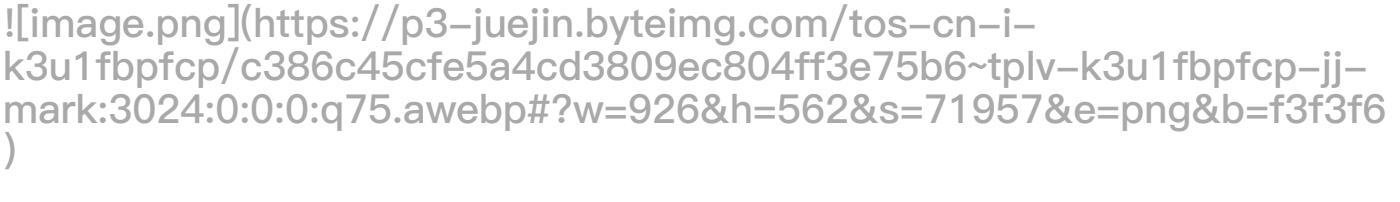
要通过server的校验，需要随机码2生成的密码

最简单的做法显然是，proxy配置实例的原始密码，然后用随机码2去加密

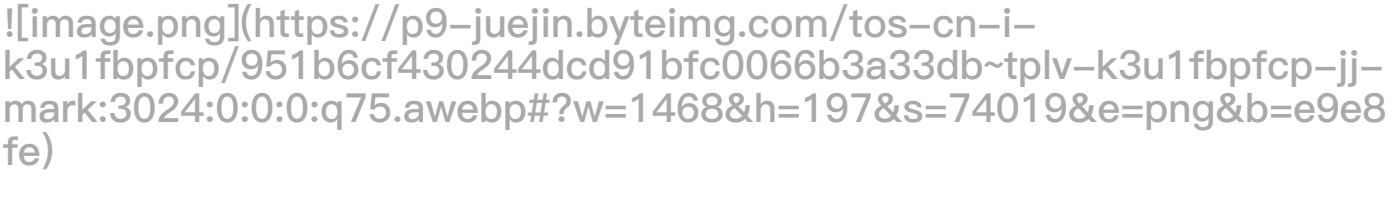
但是这样做显然很麻烦，更主要的是不安全，所有数据库的密码都放在proxy，要是泄密了不得背大锅，所以必须要做到proxy不存储密码也要能完成鉴权

所以这里需要绕过 FastPath，强制触发AuthMethodSwitchRequest进行重新鉴权，即用一个服务端一个不认识的AuthMethod时，触发服务端重新鉴权

最终整个流程如下



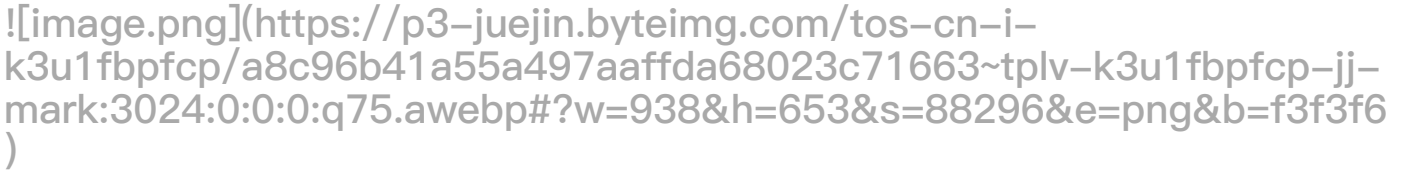
通过wireshark抓包可以看到整个包传输的流程



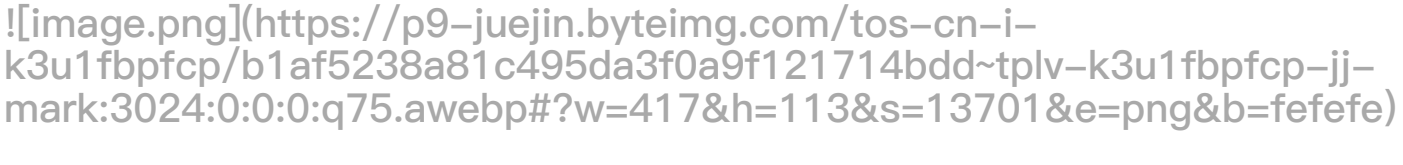
至此proxy完成了整个连接鉴权过程，现在可以进行查询操作了

MySQL查询过程分析

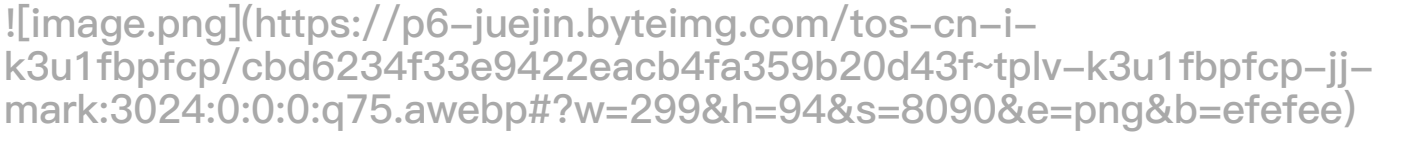
连接建立成功后，客户端就可以发送查询命令给服务端，再简单分析一下mysql的查询流程，常规的查询过程是串行的，发送一个查询包，返回一个查询结果包，然后重复上述过程



1．客户端向服务端发送request包，并指定command的为query，statement为具体查询语句，比如selelct @@version_comment limit1



2．服务端会返回一个叫做TABULAR的response组合包，该包由5个部分组成，column包指明这次查询有多少列，field包每个列的属性，比如类型，长度等，EOF 分隔包，指示, row packet 包含一行数据的值 , EOF 结束包



mysql-proxy的实现

上述分析只是理论上的核心流程，实际开发过程中会有很多问题要处理，其中最多的是对二进制字节流的处理，因为MySQL协议属于TCP协议，GO没有完整的SDK能够处理MySQL协议，所以都是依靠计算位和流来解析协议包

值得一提的是，MySQL经过多年的发展，具有丰富的功能，根据我们的业务场景，目前proxy只实现了简单查询的场景，像事务，prepare等功能，都是没有

实现的

原文链接: https://juejin.cn/post/7365348844991660070