关于KunDB的gRPC的补充

首先阅读KUNDB Security中的第一节和第三节

涉及jira：WARP-42080，WARP-42372，WARP-42232

jira中问题包括针对客户端到vtgate这一段给用户加ip限制和空闲时间限制，分别需要改在kunDB的mysql客户端连接方式和JDBC的gRPC路线，这篇文章的组织结构为第一节简述ip限制中的设计思路，第二节简述空闲时间连接断开的设计思路，最后补充gRPC的框架特性来解释第二节的结论。

一，kunDB中传输端认证与加密

针对client到vtgate这一段的远程调用，mysql客户端利用插件形式实现用户与vtgate之间的用户名密码认证,其中server会接收到mysql协议报文，现阶段kunDB的mysql只支持其中的一部分协议功能。可以参考：http://hutaow.com/blog/2013/11/06/mysql-protocol-analysis/；

JDBC使用gRPC在传输端进行认证与加密，gRPC默认基于HTTP/2的TLS
对客户端和服务端交换的所有数据进行加密传输，kunDB现阶段使用ldap管理用户名密码认证，即为了tls证书认证配置的复杂，用ldap对加密和认证做了很好的切分，实现在transprot_auth，该功能只有在开安全的时候才会走配置，否则这一段的认证和加密都不生效。

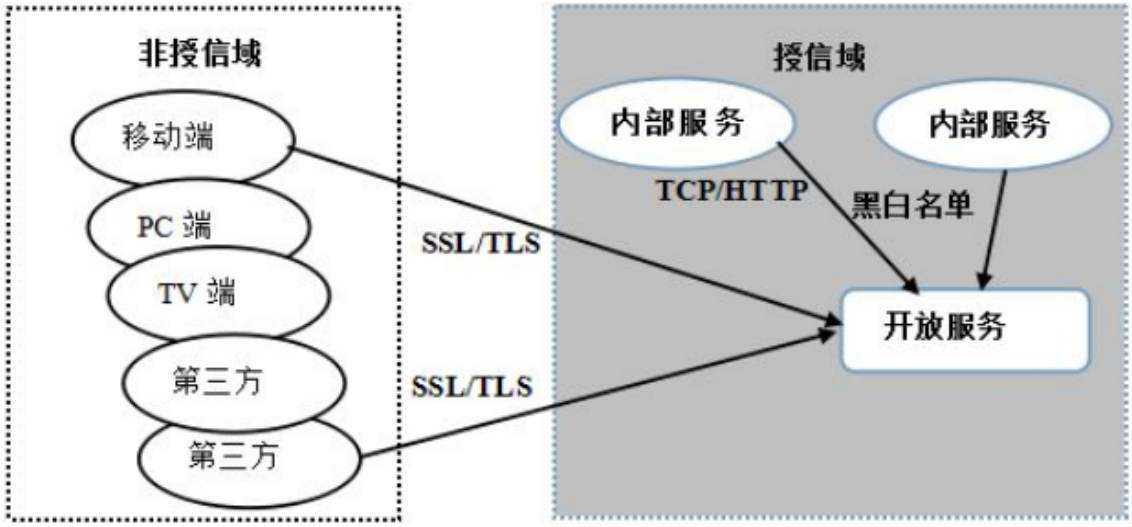

fig.1 gRPC一般安全性设计

另外关于在vtgate里通过context拿连接，“在Kundb内部，认证的结果（即客户端的身份，注意是应用客户端，实际RPC调用者的身份可以与此不同）被称作Caller ID。Caller ID是由认证时生成的，因此客户端的Caller ID是由VTGate来认证和生成的。在grpc TLS认证中，这个Caller ID等于客户端所提供证书中的Common Name。而在mysql协议的认证中，用户名则是Caller ID。”也就是说callerinfo是认证时产生的可能和调用者身份不同，在用gRPC调用时，认证ip被写成remote，username被写成gRPC。

二，gRPC的超时重连控制连接

针对idletime功能，mysql协议这边采用项目自封装的timer，监听时启动一个goroutine做计时调整标志位，保证连接的有效退出；

gRPC连接时channel作为数据的载体，共分为5种状态：Idle,Connecting,Ready,TransientFailure,Shutdown，可见grpc/connectivity.go。重连机制通过启动一个Goroutine异步的去建立连接实现的，可以避免服务器因为连接空闲时间过长关闭连接、服务器重启等造成的客户端连接失效问题。也就是说通过gRPC的重连机制可以完美的解决连接池设计原则中的空闲连接的超时与保活问题。

```go
// connect starts creating a transport.
// It does nothing if the ac is not IDLE.
// TODO(bar) Move this to the addrConn section.
func (ac *addrConn) connect() error {
    ac.mu.Lock()
    if ac.state == connectivity.Shutdown {
        ac.mu.Unlock()
        return errConnClosing
    }
    if ac.state != connectivity.Idle {
        ac.mu.Unlock()
        return nil
    }
    // Update connectivity state within the lock to prevent subsequent or
    // concurrent calls from resetting the transport more than once.
    ac.updateConnectivityState(connectivity.Connecting, lastErr: nil)
    ac.mu.Unlock()

    // Start a goroutine connecting to the server asynchronously.
    go ac.resetTransport()
    return nil
}
```

fig.2 gRPC中相应状态对连接的控制

而gRPC原有设置channel状态来控制client连接的功能是可以coverJDBC中的空闲时间问题。

三，gRPC结构简单解析

gRPC有 关于命名解析grpc/naming和负载均衡grpc/balancer的包。
首先gRPC客户端会发出服务注册和域名管理请求，dnsresolver包装连接对象注册，并通过wrapper和client端回调更新。client端再调用balancer部分接口实现对连接的管理
负载均衡，用于管理连接状态和生命周期。并侦听每一个subconn的状态变化进行策略反馈，负责发起连接的创建和移除。负载均衡策略支持外部扩展以避免更改gRPC内部的东西，其中gRPCLB就是loadbalancer，用于包含外部的实现，内置的grpc/balancer/roundrobin应该是一个简单的可供扩展的循环平衡机制，总之为balancer管理状态而服务。这样通过对连接的管理优化client端和server之间的gRPC。

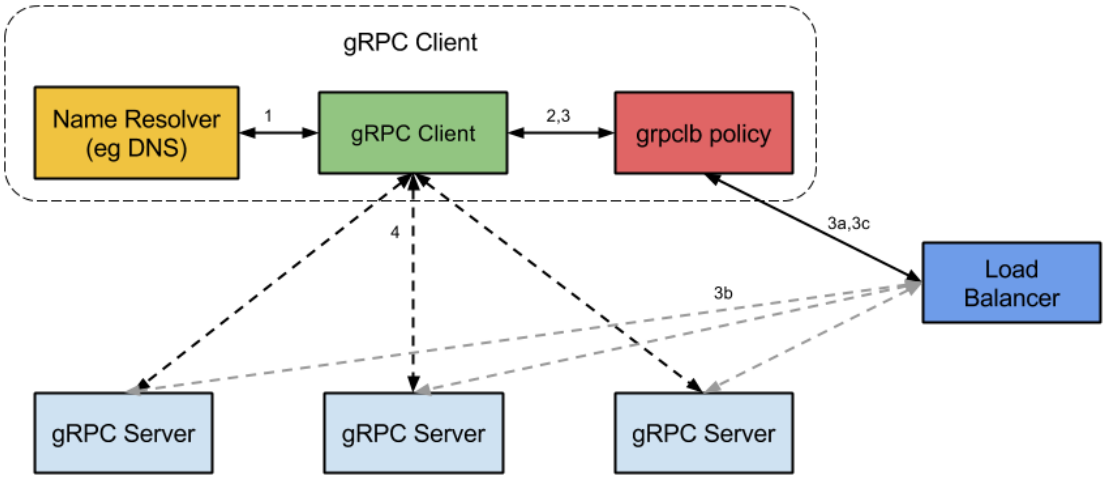

fig.3 gRPC简要框架

gRPC使用HTTP/2作为应用层的传输协议，很多机制与http/2相关，比如利用流的多路复用等。当由于缺少新的或待处理的RPC，channel没有尝试创建连接就会变成idle状态，或者当没有新的或挂起的（活动的）RPC，则READY或CONNECTING的channel也会切换到IDLE。此时，server端会给client 端发送一个http GOAWAY 的包，client 收到这个包之后就会主动关闭连接。下次需要发包的时候，就会重新建立连接，以避免试图断开连接的服务器上的连接过载。