# 单点登录系统设计文档

## 数据库模型（\*表示主键）：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 字段类型 | 是否非空 | 说明 |
| \*account | varchar32 | 是 | 登录账户的用户名 |
| password | varchar200 | 是 | 登录账户的密码 |

## 前后端交互接口介绍：

1. **登录接口：从数据库中获取用户数据，验证账号密码。密码验证一旦成功则清除内存中的RSA参数。**

入参：账号、密码、设备唯一标识

出参：无结果集，只标记是否登录成功

1. **注册（并登录）接口：向数据库中创建用户数据，注册成功后客户端也进入登录状态。密码验证一旦成功则清除内存中的RSA参数。**

入参：账号、密码、设备唯一标识；

出参：无结果集，只标记是否注册成功

1. **RSA参数生成接口：生成RSA公钥、私钥。将公钥返回给客户端。每次调用含有敏感信息的接口，都需要调用本接口申请RSA加密信息。**

入参：设备唯一标识

出参：此设备对应的RSA公钥

## 单点登录实现逻辑：

使用gRPC双向流式调用。服务器可以缓存用来给客户端发数据（调用客户端）的观察者对象。

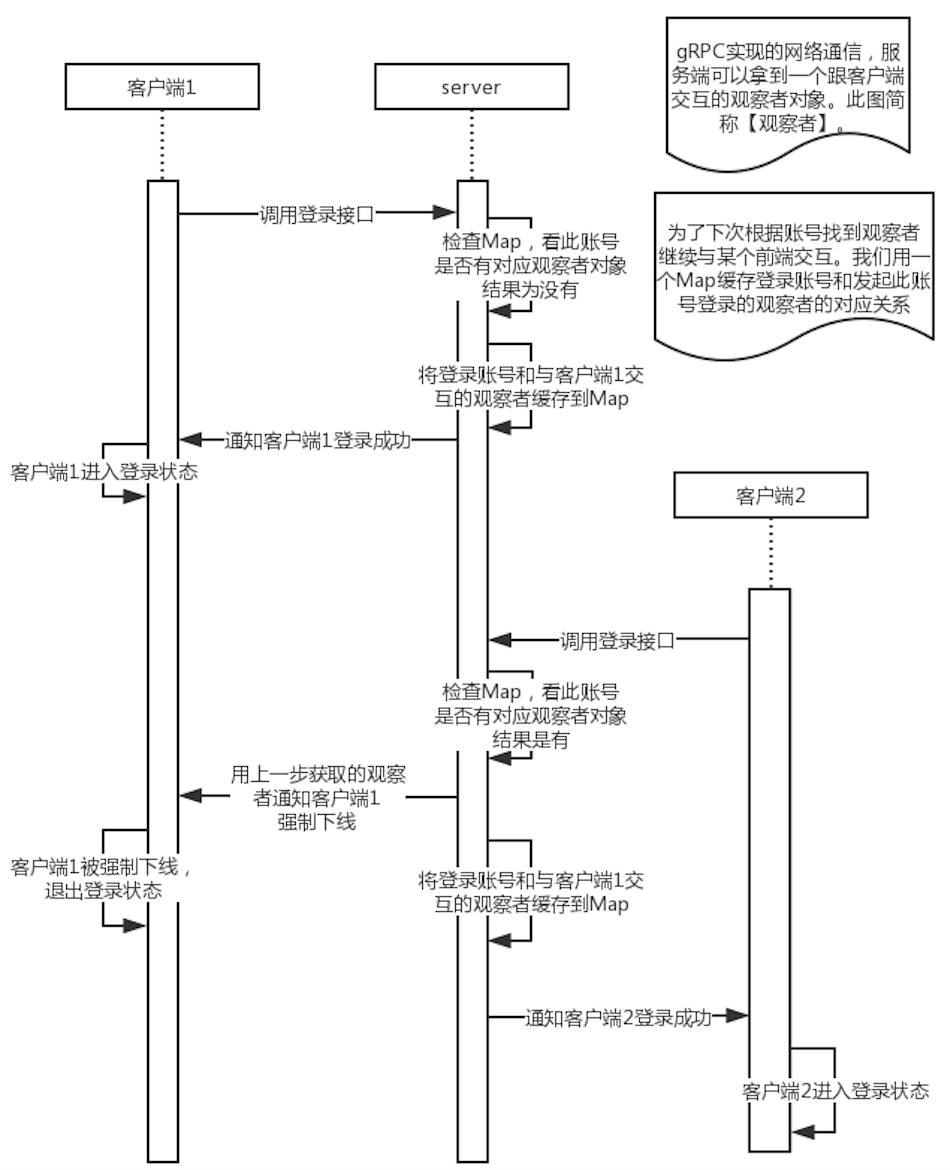
|  |
| --- |
| 单点登录逻辑流程步骤： |
| 客户端1调用登录接口到服务器，服务器验证成功； |
| 服务器用一个【观察者Map】缓存用户账号和调用客户端1的观察者对象的对应关系； |
| 此时客户端2登录相同账号； |
| 服务器从【观察者Map】中检查发现同一个账号已经有一个观察者对象在Map中； |
| 此时说明之前已经有别的客户端（也就是客户端1）登录了这个账号； |
| 拿出之前缓存的观察者对象，给客户端1发送数据，通知其强制下线； |
| 将调用客户端2的观察者存入缓存，代替客户端1的观察者对象； |

#### 可能出现的异常情况：

线程不安全，gRPC也没有自带的线程安全措施，两个客户端同时登录同一个账号，将都保持登录状态。

服务器如果重启，由于内存中变量丢失，客户端如保持登录状态的话，将永远不会被踢下线，此时只能让客户端和服务器断开连接时就下线。

时序图如下：



## PBKDF2加密存储密码

注册时，服务器获得明文密码后，使用PBKDF2做加密处理，得到密码密文存储到数据库中。后面需要校验密码后，将待校验密码也做PBKDF2处理，与数据库中存的密码密文做比较来判断是否匹配。这样服务器不存储真正的密码，只存储密码密文。

#### 可能出现的异常情况：

由于库中未保存密码原文，一旦数据库物理损坏等原因导致数据丢失，所有用户必须重新设置密码。

PBKDF2计算过程中耗费的内存等资源较少，通过高配置的硬件做暴力破解的成本比较低。不过可以把Hash计算次数提高，或者给登录接口加个验证码。

## RSA密码加密传输：

通过RSA非对称加密实现密码的安全传输。且RSA公私钥仅一次性有效。每次调用需校验密码的接口前，都要先调用申请RSA密钥的接口。这样攻击者不仅看不到密码，而且即便通道不安全，做中间人攻击也无法进入登录状态。

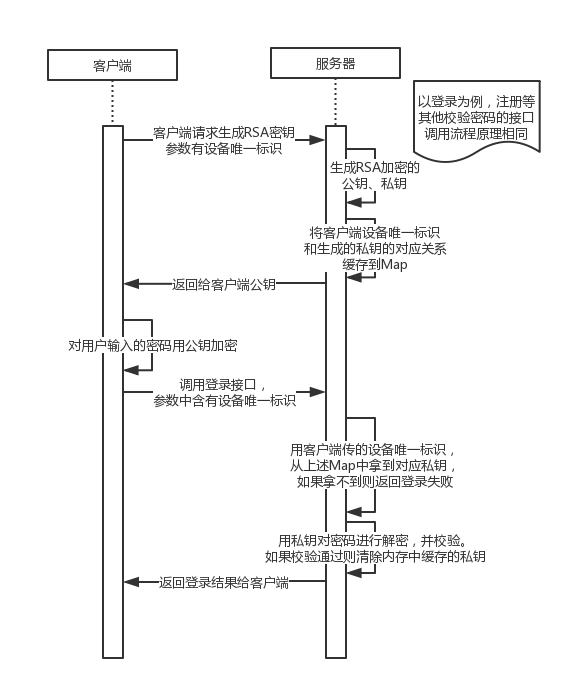
|  |
| --- |
| 调用登录接口逻辑（注册等其他需验证密码的逻辑一样）： |
| 1、客户端先调用服务器的生成RSA公钥、私钥接口，传设备唯一标识； |
| 2、服务端收到调用，生成RSA公钥、私钥； |
| 3、服务端将公钥、私钥及对应的设备唯一标识存储在缓存中； |
| 4、服务端返回公钥给客户端，客户端用公钥加密密码； |
| 5、客户端调用登录接口，将加密后的密码传给服务器，并带上设备唯一标识； |
| 6、服务器收到调用后，通过设备唯一标识找到之前缓存的RSA私钥； |
| 7、如果找不到，则说明客户端刚刚没有申请过RSA密钥，直接返回失败； |
| 8、如果能找到，用私钥解密客户端传来的密码，并校验； |
| 9、校验如果成功，则清空改设备唯一标识对应的RSA信息； |

#### 可能出现的异常情况：

在通道不安全的前提下，攻击程序可以做中间人攻击时，可截取用户输入密码的RSA加密密文，然后篡改密文并提交到服务器。再将原密文提交到服务器以获取登录状态。

RSA加解密过程中，前端传来的密文必须通过Hex转码，后端也是先经过Hex解码然后解密的，否则传输过程中会出现乱码。

下图是RSA密码加密传输流程时序图：



## tsl或ssl通道加密

用rpc支持的tsl或ssl给通道进行加密。安装openssl工具，使用openssl命令生成服务器私钥证书key文件和客户端公钥证书pem文件。gRPC有默认的通道加密方案，只需我们传入证书文件。

Android端使用OkHttpChannelBuilder API连接服务器，并使用pem文件生成SslSocketFactory对象传入gRPC底层中；

服务端创建Server对象中将证书传入底层；

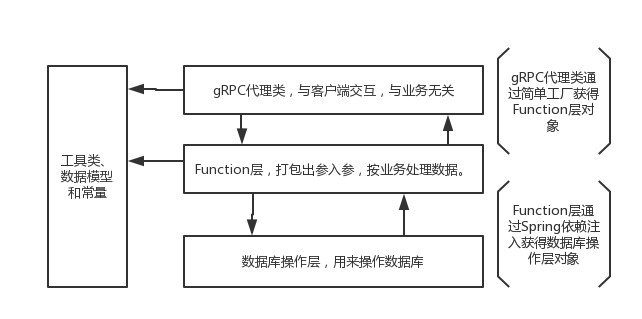
（由于时间问题，此部分未完成）

## 服务器程序结构说明：

1、rpc代理类，与业务无关，专门负责利用gRPC与客户端通信

2、功能号Function层，整理出参、入参。进行业务逻辑操作

3、数据库操作Dao层



## 客户端程序结构说明：

Android程序：

主页面，登录、注册按钮。显示登录状态和账号、密码输入框。子线程中调用NDK实现单点登录业务功能

NDK部分：

通过gRPC调用服务器接口实现登录、注册和监听登录状态，并返回数据给Android程序。登录后调用监听登录状态的服务端接口，当被踢下线后，调用Java静态方法通知Android程序改变登录状态。

由于本人没有NDK、C++和gRPC开发经验，且学习时间不足，此部分用Android程序实现。如按题目要求实现，操作步骤如下：

从官网提供的Github地址中下载gRPC底层库和Demo；

执行命令安装proto工具：$ make route\_guide.grpc.pb.cc route\_guide.pb.cc；

用Android Studio打开Demo路径下的项目：grpc\_demo/grpc/examples/android；

修改cmake文件或使用bazel工具，剔除会报错且客户端不需要的依赖库，使Demo能运行；

编写proto文件定义交互接口；

执行protoc命令，按照proto接口定义文件，生成C++类文件；

编写gPRC通信类，专门负责和服务器通信；

编写与Android Java程序交互的代理类

## 备用选型介绍：

#### 单点登录功能的其他实现：服务端定义登录状态检查接口，单点登录采用客户端定时调用此接口检查当前登录状态是否还有效。

优点：服务端不必将登录状态放到内存，也不用和客户端维持长连接。可以将登录状态存入数据库。

缺点：客户端需要用定时器定时调用接口，耗费了更多资源。

不选择原因：频繁建立连接更加耗费资源，且下线无法第一时间感知。将已登录的客户端的观察者对象存入内存Map中并不是很耗费内存空间。