Shizuku的作用是给普通APP提供一个adb或root权限的service，原理是作为普通app和系统service的中间人，转发请求和结果，例如可以用于：

1. 系统API，例如PMS/AMS等
2. Shell命令

系统API容易理解，shizuku只需代执行系统API即可，但是Shizuku支持通过binder在client端编写一定逻辑，但是可以将这段逻辑执行的权限提升至adb或root，虽说binder可以达到跨进程通信，但是执行的主体无法变更，shizuku作为服务端是如何以server端的权限执行client端的代码？

Shizuku接入流程

Shizuku中的角色分为：

- Shizuku server 通过shell权限运行的server，ShizukuService真正实现所在
- Shizuku Manager 管理端app
- Client 接入Shizuku的普通APP

client声明ShizukuProvider

接入Shizuku的app需要申明rikka.shizuku.ShizukuProvider。当app启动时，shizuku会通过该provider向client app发送shizuku的binder即moe/shizuku/server/IShizukuService.aidl

client自定义aidl

```
 private final Shizuku.UserServiceArgs
userServiceStandaloneProcessArgs =
            new Shizuku.UserServiceArgs(new
ComponentName(BuildConfig.APPLICATION_ID,
UserService.class.getName()))
                    .processNameSuffix("service")
                    .debuggable(BuildConfig.DEBUG)
                    .version(BuildConfig.VERSION_CODE);

    private void bindUserServiceStandaloneProcess() {
        StringBuilder res = new StringBuilder();
        try {
            if (Shizuku.getVersion() < 10) {
                res.append("requires Shizuku API 10");
            } else {

Shizuku.bindUserService(userServiceStandaloneProcessArgs,
userServiceConnection);
            }
        } catch (Throwable tr) {
            tr.printStackTrace();
        }
    }
```

核心： Shizuku.bindUserService(userServiceStandaloneProcessArgs, userServiceConnection);

其中userServiceStandaloneProcessArgs会携带客户端声明的aidl的stub实现类、client app包名等参数。

通过第一步获取的binder调用ShizuKuService#addUserService方法，该方法会创建一个新的moe.shizuku.starter.ServiceStarter进程。

在SerivceStarter的main方法中，可以发现，会创建app包名对应的context并获取classloader，然后加载对应的stub类并实例化，也就是说，shizuku作为服务端是如何以server端的权限执行client端的代码实际是通过反射来完成。

实例化完client service stub类之后，通过在manager中声明的ContentProvider将实例化的stub instance通过ShizukuService.attachUserService保存在UserServiceRecord中（UserServiceRecord#setBinder）。

> manager中的moe.shizuku.manager.ShizukuManagerProvider本身也是一个rikka.shizuku.ShizukuProvider，所以也会通过ShizukuService与Shizuku server通信。

那么bindUserService中的userServiceConnection是什么时候被回调的呢，userServiceConnection是一个android.content.ServiceConnection的子类实现，在正常binder通信时onServiceConnected方法会在binder建立之后回调，在shizuku中，则是在UserServiceRecord#setBinder中执行该方法，userServiceConnection会从client一路传递到shizuku server到新创建的一个独立的进程到manager最后又到shizuku server中，在第一次传递到shizuku server中时就会在server中创建一个保存UserService相关信息的UserServiceRecord，例如使用RemoteCallbackList保存ServiceConnection，然后在最后获取到客户端的binder instance之后回到shizuku server时，会从UserServiceRecord中取出serviceConnection，并通过RemoteCallbackList机制调用onServiceConnected方法，并将之前在独立进程中创建的binder实例传入，如此一来，客户端就能收到回调时机并且拿到客户端自定义的service的binder。

因此客户端自定义的aidl，stub端实际上是在Shizuku新建的一个独立进程，而proxy端在客户端app自身。