简单理解Binder机制的原理 - 简书

jianshu.com/p/4920c7781afe

一、概述

Android系统中，涉及到多进程间的通信底层都是依赖于Binder IPC机制。例如当进程A中的Activity要向进程B中的Service通信，这便需要依赖于Binder IPC。不仅于此，整个Android系统架构中，大量采用了Binder机制作为IPC（进程间通信）方案。

当然也存在部分其他的IPC方式，如管道、SystemV、Socket等。那么Android为什么不使用这些原有的技术，而是要使开发一种新的叫Binder的进程间通信机制呢？

为什么要使用Binder？

性能方面

在移动设备上（性能受限制的设备，比如要省电），广泛地使用跨进程通信对通信机制的性能有严格的要求，Binder相对出传统的Socket方式，更加高效。Binder数据拷贝只需要一次，而管道、消息队列、Socket都需要2次，共享内存方式一次内存拷贝都不需要，但实现方式又比较复杂。

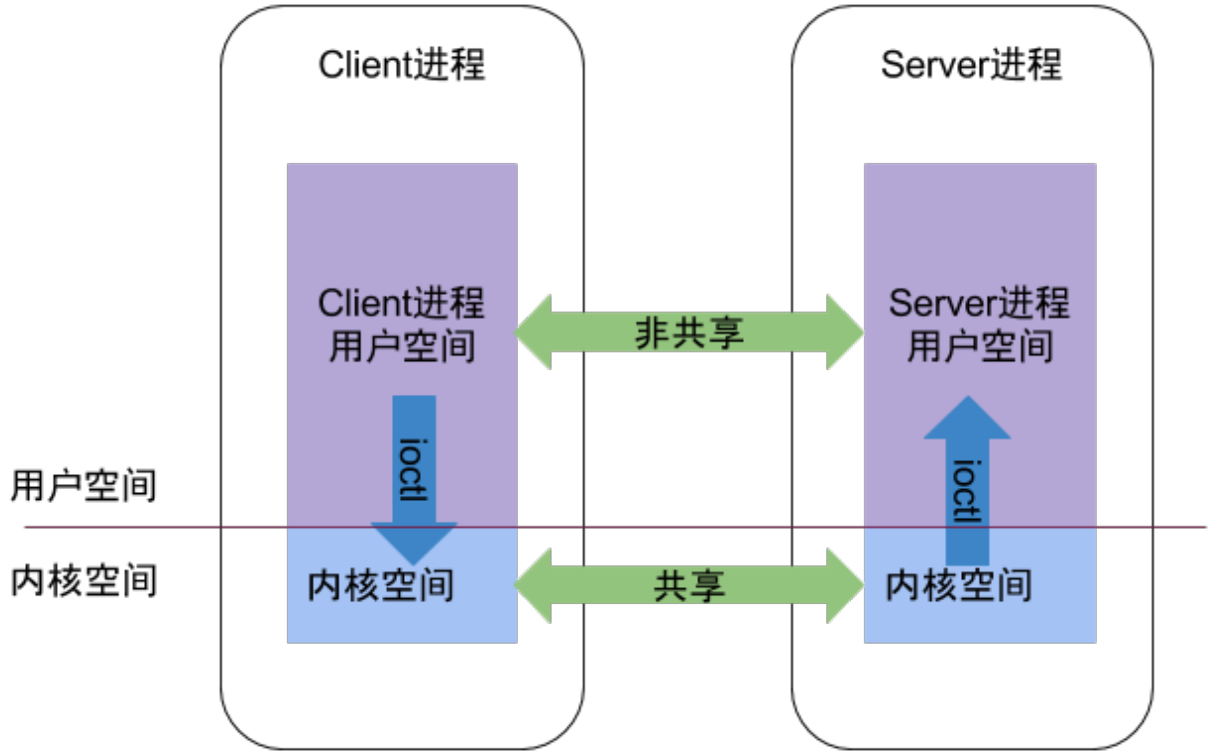
安全方面

传统的进程通信方式对于通信双方的身份并没有做出严格的验证，比如Socket通信ip地址是客户端手动填入，很容易进行伪造，而Binder机制从协议本身就支持对通信双方做身份校检，因而大大提升了安全性。

二、 Binder

IPC原理

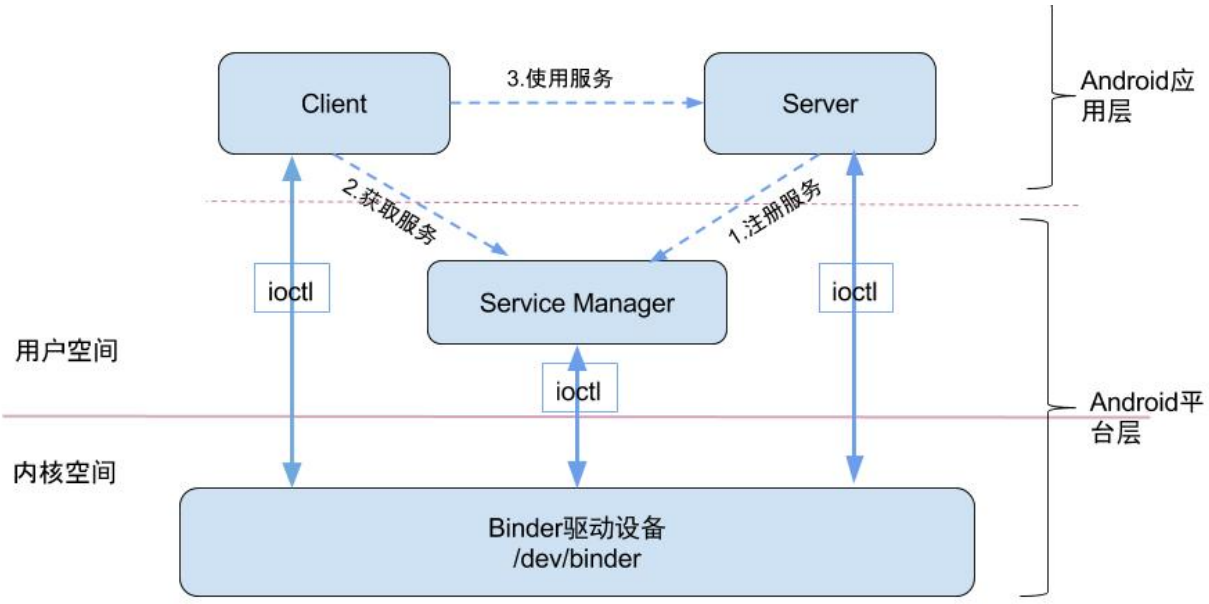
从进程角度来看IPC机制



每个Android的进程，只能运行在自己进程所拥有的虚拟地址空间。对应一个4GB的虚拟地址空间，其中3GB是用户空间，1GB是内核空间，当然内核空间的大小是可以通过参数配置调整的。对于用户空间，不同进程之间彼此是不能共享的，而内核空间却是可共享的。Client进程向Server进程通信，恰恰是利用进程间可共享的内核内存空间来完成底层通信工作的，Client端与Server端进程往往采用ioctl等方法跟内核空间的驱动进行交互。

Binder原理

Binder通信采用C/S架构，从组件视角来说，包含Client、Server、ServiceManager以及binder驱动，其中ServiceManager用于管理系统中的各种服务。架构图如下所示：



Binder通信的四个角色
Client进程：使用服务的进程。

Server进程：提供服务的进程。

ServiceManager进程：ServiceManager的作用是将字符形式的Binder名字转化成Client中对该Binder的引用，使得Client能够通过Binder名字获得对Server中Binder实体的引用。

Binder驱动：驱动负责进程之间Binder通信的建立，Binder在进程之间的传递，Binder引用计数管理，数据包在进程之间的传递和交互等一系列底层支持。

Binder运行机制

图中Client/Server/ServiceManage之间的相互通信都是基于Binder机制。既然基于Binder机制通信，那么同样也是C/S架构，则图中的3大步骤都有相应的Client端与Server端。

注册服务(addService)：Server进程要先注册Service到ServiceManager。该过程：Server是客户端，ServiceManager是服务端。

获取服务(getService)：Client进程使用某个Service前，须先向ServiceManager中获取相应的Service。该过程：Client是客户端，ServiceManager是服务端。

使用服务：Client根据得到的Service信息建立与Service所在的Server进程通信的通路，然后就可以直接与Service交互。该过程：client是客户端，server是服务端。

图中的Client,Server,Service Manager之间交互都是虚线表示，是由于它们彼此之间不是直接交互的，而是都通过与Binder驱动进行交互的，从而实现IPC通信方式。其中Binder驱动位于内核空间，Client,Server,Service Manager位于用户空间。Binder驱动和Service Manager可以看做是Android平台的基础架构，而Client和Server是Android的应用层，开发人员只需自定义实现client、Server端，借助Android的基本平台架构便可以直接进行IPC通信。

Binder运行的实例解释

首先我们看看我们的程序跨进程调用系统服务的简单示例，实现浮动窗口部分代码：

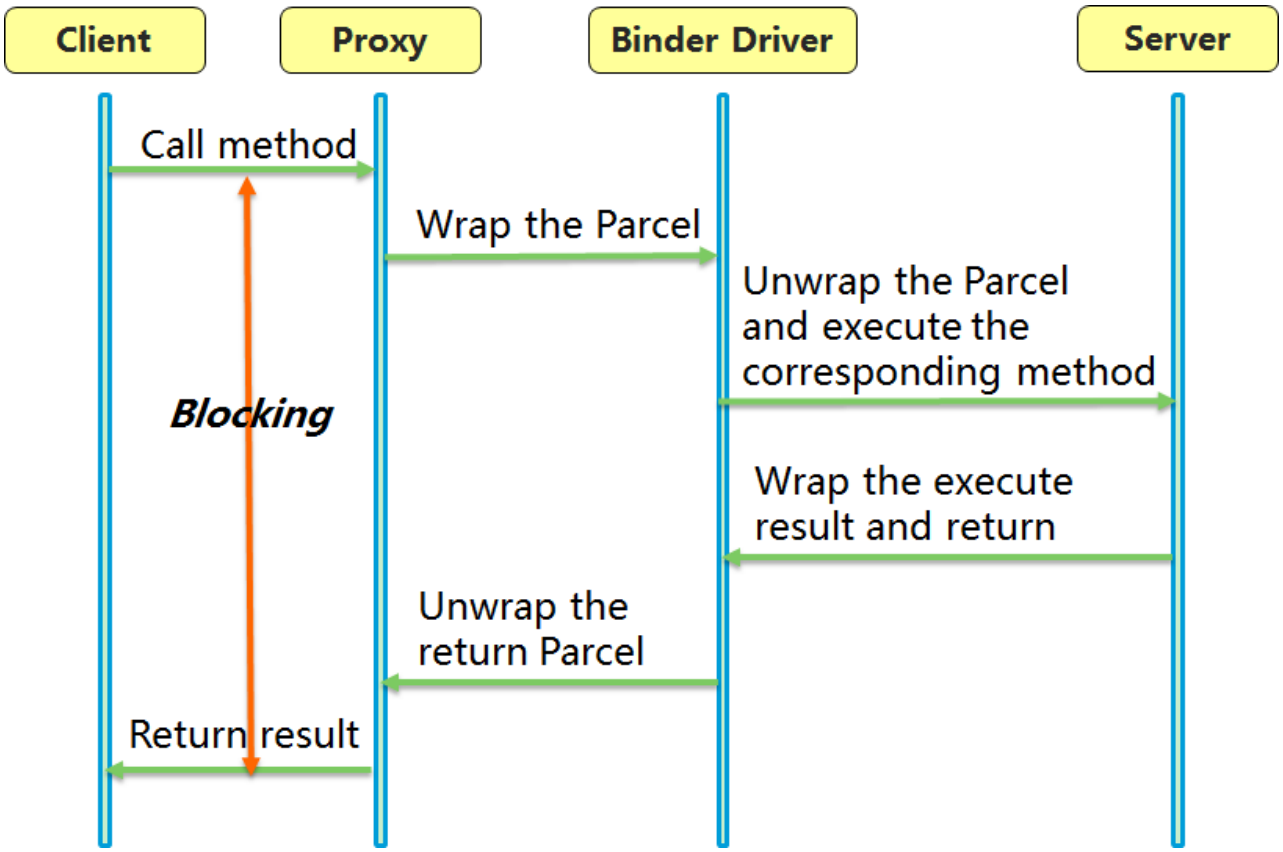
```
WindowManager wm =
(WindowManager)getSystemService(getApplication().WINDOW_SERVICE);

View view=LayoutInflater.from(getApplication()).inflate(R.layout.float_layout, null);

wm.addView(view, layoutParams);
```

注册服务(addService)：在Android开机启动过程中，Android会初始化系统的各种Service，并将这些Service向ServiceManager注册（即让ServiceManager管理）。这一步是系统自动完成的。

获取服务(getService)：客户端想要得到具体的Service直接向ServiceManager要即可。客户端首先向ServiceManager查询得到具体的Service引用，通常是Service引用的代理对象，对数据进行一些处理操作。即第2行代码中，得到的wm是WindowManager对象的引用。

使用服务：通过这个引用向具体的服务端发送请求，服务端执行完成后就返回。即第6行调用WindowManager的addView函数，将触发远程调用，调用的是运行在systemServer进程中的WindowManager的addView函数。

使用服务的具体执行过程



1. client通过获得一个server的代理接口，对server进行调用。
2. 代理接口中定义的方法与server中定义的方法时一一对应的。
3. client调用某个代理接口中的方法时，代理接口的方法会将client传递的参数打包成Parcel对象。
4. 代理接口将Parcel发送给内核中的binder driver。
5. server会读取binder driver中的请求数据，如果是发送给自己的，解包Parcel对象，处理并将结果返回。
6. 整个的调用过程是一个同步过程，在server处理的时候，client会block住。**因此client调用过程不应在主线程。**

以上就是Binder机制原理的简单介绍，后面会以AIDL来具体介绍Binder机制的使用，加深对其的了解。