Android Binder

孟芳芳 已于 2022-08-11 07:49:22 修改

1. Binder

Binder译为“粘合剂”，是Android系统中进程间通信的一种方式。

Android的四大组件Activity、Service、BroadCast、ContentProvider以及不同的APP等都运行在不同的进程中，Binder是这些进程间通信的桥梁。它好比“粘合剂”一样，把系统中各个组件粘合到一起。

Binder就是Android中的血管，在Android中使用Activity、Service等组件时都需要和AMS进行通信，这种跨进程的通信都是通过Binder完成。Activity、Service等组件和AMS不是同一个进程，也是多进程通信。

①为什么要用多进程?

虚拟机给每一个进程分配的内存是有限的，LMK会优先回收对系统资源占用多的进程。因此，使用多进程的好处是：

1)为了突破内存限制，防止占用内存过多被杀；

2)功能稳定性，一个进程崩溃对另外进程不造成影响：将不稳定功能放入独立进程；

3)规避内存泄漏，独立的WebView进程阻隔内存泄漏导致问题；

操作系统中，进程与进程间内存是不共享的，A进程没法直接访问B进程的数据，这称为进程隔离。A进程和B进程之间要进行数据交互就得采用特殊的通信机制，即进程间通信IPC（Interprocess Communication）。

Android系统中涉及到多进程间的通信，其底层都是依赖于Binder IPC机制。例如当进程A中的Activity要向进程B中的Service通信，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方案，当然也存在部分其他的IPC方式，如管道、Socket等。

②为什么使用Binder

1)性能方面：

Binder相对于传统的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。

优势	描述
性能	只需要一次数据拷贝，性能上仅此于共享内存
稳定性	基于 C/S 架构，职责明确、架构清晰
安全性	为每个 APP 分配 UID，进程的UID是鉴别进程身份的重要标志

IPC 方式	数据拷贝次数
共享内存	0
Binder	1
Socket/管道/消息队列	2

2)安全方面：

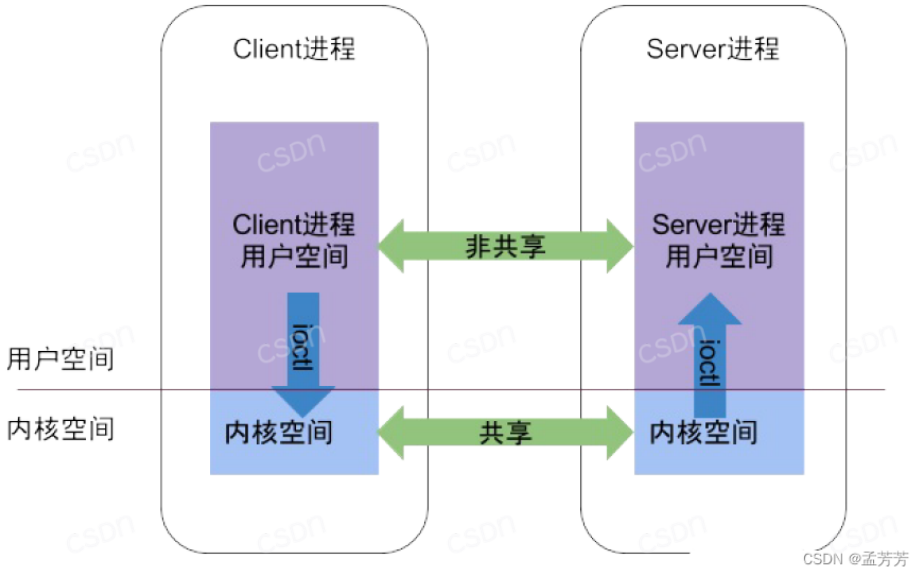
传统的进程通信方式对于通信双方的身份并没有做出严格的验证，比如Socket通信的IP地址是客户端手动填入，很容易进行伪造；

Binder机制从协议本身就支持对通信双方做身份校检，从而大大提升了安全性。

2.Binder原理

①IPC原理

从进程角度来看进程间通信IPC机制：



进程空间划分：用户空间(User Space) + 内核空间(Kernel Space)

1)每个Android进程，只能运行在自己进程所拥有的虚拟地址空间。例如，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，内核空间的大小是可以通过参数配置调整的；

2)不同进程之间用户空间是不能共享的，而内核空间是可共享的；

3)Client进程向Server进程通信，是利用进程间可共享的内核空间来完成底层通信工作的；

4)Client端与Server端进程往往采用ioctl等方法与内核空间的驱动进行交互；

(ioctl是设备驱动程序中设备控制接口函数，一个字符设备驱动通常会实现设备打开、关闭、读、写等功能，在一些需要细分的情境下，如果需要扩展新的功能，通常以增设ioctl()命令的方式实现。)

注意：只有系统调用才能操作内核空间。

系统调用：用户态与内核态

1)虽然从逻辑上进行了用户空间和内核空间的划分，但不可避免的用户空间需要访问内核资源，比如文件操作、访问网络等。

2)为了突破隔离限制，就需要借助系统调用来实现。系统调用是用户空间访问内核空间的唯一方式，保证了所有的资源访问都是在内核的控制下进行的，避免了用户程序对系统资源的越权访问，提升了系统安全性和稳定性。

3)Linux使用两级保护机制：0 级供系统内核使用，3 级供用户程序使用。

copy_from_user() //将数据从用户空间拷贝到内核空间

copy_to_user() //将数据从内核空间拷贝到用户空间

②Binder IPC实现原理

(1)动态内核可加载模块&&内存映射：

跨进程通信是需要内核空间做支持的。传统的IPC 机制如管道、Socket都是内核的一部分。但Binder并不是Linux系统内核的一部分，那怎么办呢？于是有了Linux的动态内核可加载模块（Loadable Kernel Module，LKM）机制：

1)模块是具有独立功能的程序，它可以被单独编译，但是不能独立运行。它在运行时被链接到内核作为内核的一部分运行；

2)Android系统可以通过动态添加一个内核模块运行在内核空间，用户进程之间通过这个内核模块作为桥梁来实现通信。在Android系统中，这个运行在内核空间、负责各个用户进程通过Binder实现通信的内核模块就叫Binder驱动。

Android系统中用户进程之间是如何通过这个内核模块（Binder驱动）来实现通信的呢？当然不是之前的：将数据从发送方进程拷贝到内核缓存区，然后再将数据从内核缓存区拷贝到接收方进程，

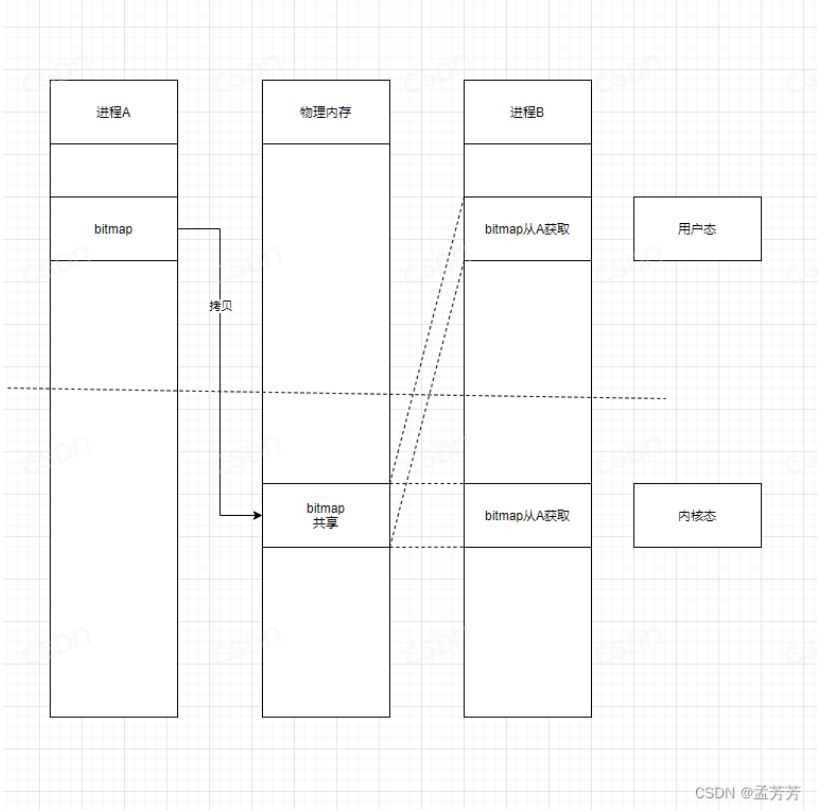
而是通过内存映射（mmap）：

内存映射简单的讲就是将用户空间的一块内存区域映射到内核空间，映射关系建立后，用户对这块内存区域的修改可以直接反应到内核空间；反之内核空间对这段区域的修改也能直接反应到用户空间。内存映射能减少数据拷贝次数，实现用户空间和内核空间的高效互动；

(2)Binder IPC实现原理：

Binder IPC就是基于内存映射来实现的，但是mmap()通常是用在有物理介质的文件系统上，进程中的用户区域是不能直接和物理设备打交道的。比如想要把磁盘上的数据读取到进程的用户区域，需要两次拷贝（磁盘–>内核空间–>用户空间），通常在这种场景下mmap()就能发挥作用，通过在物理介质和用户空间之间建立映射，减少数据的拷贝次数，用内存读写取代I/O读写，提高文件读取效率。

Binder一次拷贝原生实现：



副本直接copy进内核态映射给B用，那么什么是内存映射?

内存映射：

1)底层原理其实就是虚拟内存。

简单来说：不同的虚拟内存指向相同的物理内存，从而实现共享内存和共享文件。

2)涉及外存和页交换，起因就是每个进程的地址空间都是0-L4的max，但是很多进程共用L4，这时候就需要拿L5来暂存L4暂时不用的数据和代码段，L4不用的换到L5，L5要用的换进L4就叫页交换。

3)现代多核CPU：

L0寄存器

L1是指令cache和数据cache

L2是单核总cache

L3是所有核的总cache

L4是内存

L5是硬盘，磁盘等

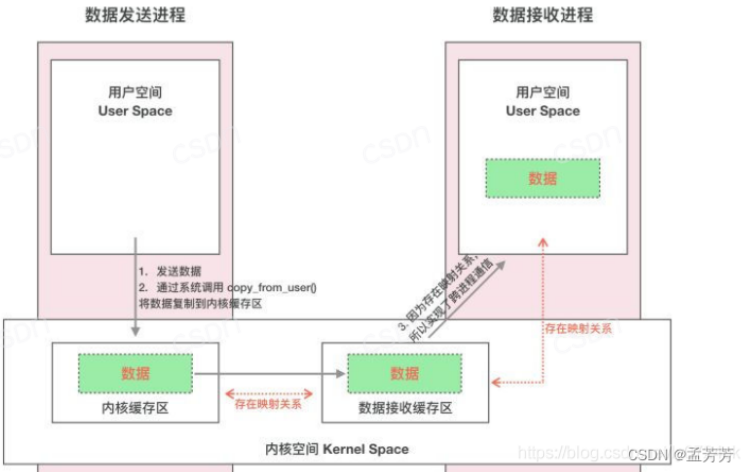
L6是通过网络挂载的远程文件系统

4)操作系统默认是指向不同的地址，L4不够了就指到L5去，L5就是外存，所以有C盘空间不够了电脑卡这个说法(变量malloc不出来，阻塞了)

很多进程一起跑慢是因为频繁页交换，那么一起跑为啥会导致频繁的页交换呢?

页交换：内存不足时，把进程传输到磁盘上，增加了磁盘I/O的负载。页交换就是把内存页往外换，把要运行的内存换进来。

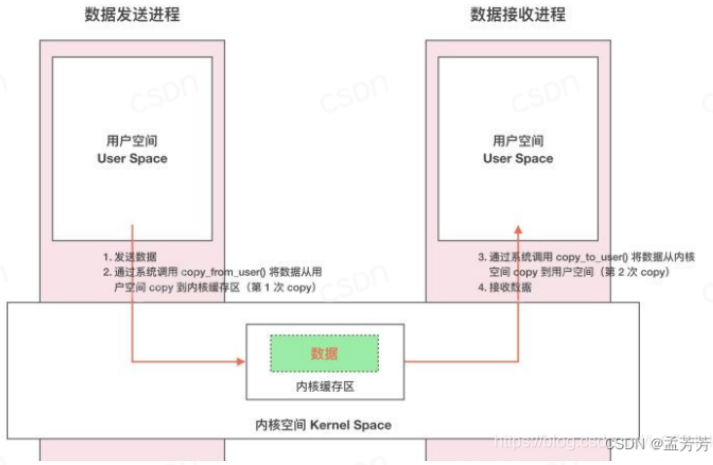
假设你4G的电脑，8个运行时稳定占用2G的进程在跑，假设之前在跑的是进程1和2，这两个就把4G用完了。接下来跑进程3，就需要页交换把进程3稳定运行时需要的2G换进来，1和2换出去2G。接下来跑进程4，4进来2G，1,2,3出去2G，内存和硬盘频繁IO，那必须慢啊。严重不足的时候，需要把赖着不走不运行的全赶走才有内存运行需要的代码，比如内存就剩500KB，接下来要跑英雄联盟，那可不得把其他人都换出去（IO操作是很耗时的）



一次完整的Binder IPC通信过程通常是这样：

Binder驱动在内核空间创建一个数据接收缓存区；然后在内核空间开辟一块内核缓存区，建立内核缓存区和内核中数据接收缓存区之间的映射关系，以及内核中数据接收缓存区和接收进程用户空间地址的映射关系；发送方进程通过系统调用copy_from_user()将数据拷贝到内核中的内核缓存区，由于内核缓存区和接收进程的用户空间存在内存映射，因此也就相当于把数据发送到了接收进程的用户空间，这样便完成了一次进程间的通信。

③Linux 下的传统 IPC 通信原理



传统的 IPC 方式中，进程之间是如何实现通信的?

1)消息发送方将要发送的数据存放在内存缓存区中，通过系统调用进入内核态，然后内核程序在内核空间分配内存，开辟一块内核缓存区，调用copy_from_user() 函数将数据从用户空间的内存缓存区拷贝到内核空间的内核缓存区中

2)接收方进程在接收数据时在自己的用户空间开辟一块内存缓存区，然后内核程序调用 copy_to_user() 函数将数据从内核缓存区拷贝到接收进程的内存缓存区。这样数据发送方进程和数据接收方进程就完成了一次数据传输，我们称完成了一次进程间通信

传统的 IPC 方式存在的问题：

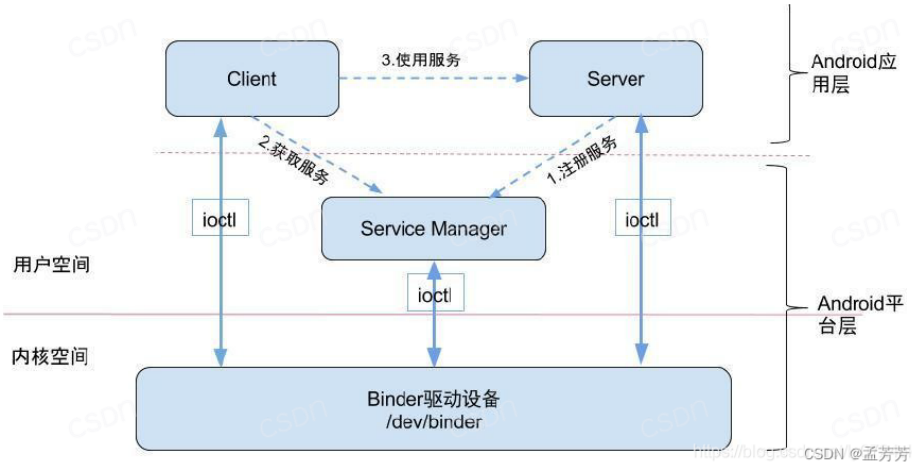
1)性能低下：内存缓存区 --> 内核缓存区 --> 内存缓存区，需要 2 次数据拷贝；

2)接收数据的缓存区由数据接收进程提供，但是不知道数据大小，只能开辟大空间或者提前调用API接受消息头获取，不是浪费空间就是浪费时间。

④Binder 通信模型

知道了Binder IPC的底层通信原理，继续看看实现层面是如何设计的。

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：



ioctl(input/output control)是一个专用于设备输入输出操作的系统调用。

Binder通信的四个角色:

1)Client进程：使用服务的进程。

2)Server进程：提供服务的进程。

3)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。

4)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:

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。

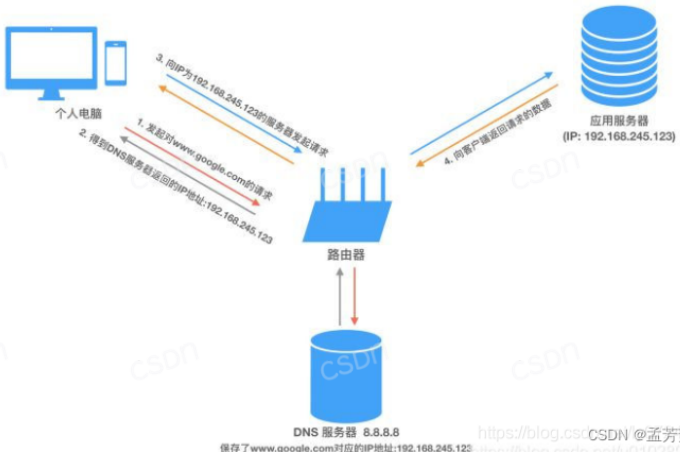
1)注册服务(addService)：Server进程要先注册Service到ServiceManager。该过程：Server是客户端，ServiceManager是服务端。

2)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。

3)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。



Client、Server、ServiceManager、Binder驱动就如同互联网中服务器（Server）、客户端（Client）、DNS域名服务器（ServiceManager）以及路由器（Binder 驱动）之前的关系。比如：

访问一个网页的步骤：在浏览器输入一个地址如 “http://www.google.com” 然后按下回车键。但是并没有办法通过域名地址直接找到要访问的服务器，因此需要首先访问 DNS 域名服务器，域名服务器中保存了 “http://www.google.com” 对应的 IP地址 “192.168.245.123”，然后通过这个 IP地址才能找到 “http://www.google.com” 对应的服务器。

Binder驱动就如同路由器一样，是整个通信的核心；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。

ServiceManager与实名Binder：

1)ServiceManager： ServiceManager和DNS类似，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 的名字获得对Binder实体的引用。

2)实名Binder：注册了名字的Binder叫实名Binder，就像网站一样除了除了有IP地址外还有自己的网址。

3)Server创建了Binder，并为它起一个字符形式将这个Binder实体连同名字一起以数据包的形式通过Binder驱动发送给ServiceManager ，通知ServiceManager注册一个名为“张三”的Binder。

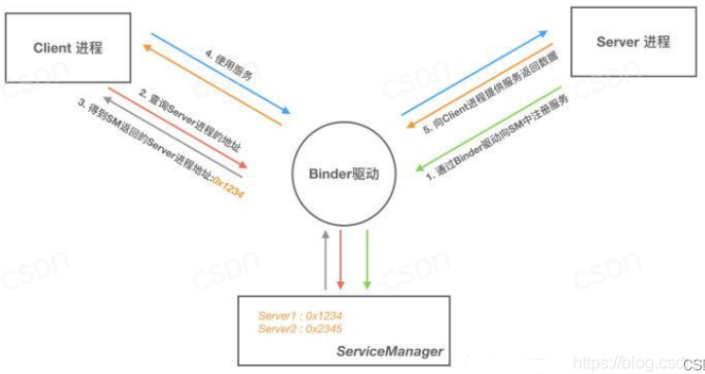
4)驱动为这个穿越进程边界的Binder创建位于内核中的实体节点以及ServiceManager对实体的引用 ，将名字以及新建的引用打包传给ServiceManager。ServiceManger收到数据后从中取出名字和引用填入查找表。

注意：有个很有意思的点，你可能会发现，ServierManager是一个进程，Server是另一个进程，Server向ServiceManager中注册Binder必然涉及到进程间通信。当前实现进程间通信又要用到进程间通信，这就好像蛋可以孵出鸡的前提却是要先找只鸡下蛋。Binder的实现比较巧妙，就是预先创造一只鸡来下蛋。ServiceManager和其他进程同样采用Bidner通信，ServiceManager是 Server端，有自己的Binder实体，其他进程都是Client，需要通过这个Binder的引用来实现Binder 的注册，查询和获取。ServiceManager提供的Binder比较特殊，它没有名字也不需要注册，当一个进程使用BINDERSETCONTEXT_MGR命令将自己注册成ServiceManager时Binder驱动会自动为它创建Binder实体（这就是那只预先造好的那只鸡）。其次这个Binder实体的引用在所有Client中都固定为0而无需通过其它手段获得。也就是说，一个Server想要向ServiceManager注册自己的Binder就必须通过这个0号引用和ServiceManager的Binder通信。类比互联网，0号引用就好比是域名服务器的地址，你必须预先动态或者手工配置好。要注意的是，这里说的Client 是相对于ServiceManager而言的，一个进程或者应用程序可能是提供服务的Server，但对于ServiceManager来说它仍然是个Client。

Client获得实名Binder的引用：

Server向ServiceManager中注册了Binderl以后， Client就能通过名字获得Binder的引用了，Clientt也利用保留的0号引用向ServiceManager请求访问某个Binder:：我申请访问名字叫张三的Binder引用，ServiceManager收到这个请求后从请求数据包中取出Binder名称，在查找表里找到对应的条目，取出对应的Binder引用作为回复发送给发起请求的Client。从面向对象的角度看，Server中的 Binder实体现在有两个引用：一个位于ServiceManager中，一个位于发起请求的Client中如果接下来有更多的Client 请求该Binder，系统中就会有更多的引用指向该Binder ，就像Java中一个对象有多个引用一样。

Binder 通信过程：



1)首先，一个进程使用BINDERSETCONTEXT_MGR命令通过Binder驱动将自己注册成为ServiceManager；

2)Server通过驱动向ServiceManager中注册Binder（Server中的Binder实体），表明可以对外提供服务。驱动为这个Binder创建位于内核中的实体节点以及ServiceManager对实体的引用，将名字以及新建的引用打包传给ServiceManager，ServiceManger将其填入查找表。

3)Client通过名字，在Binder驱动的帮助下从ServiceManager中获取到对Binder实体的引用，通过这个引用就能实现和Server进程的通信。

3.实例

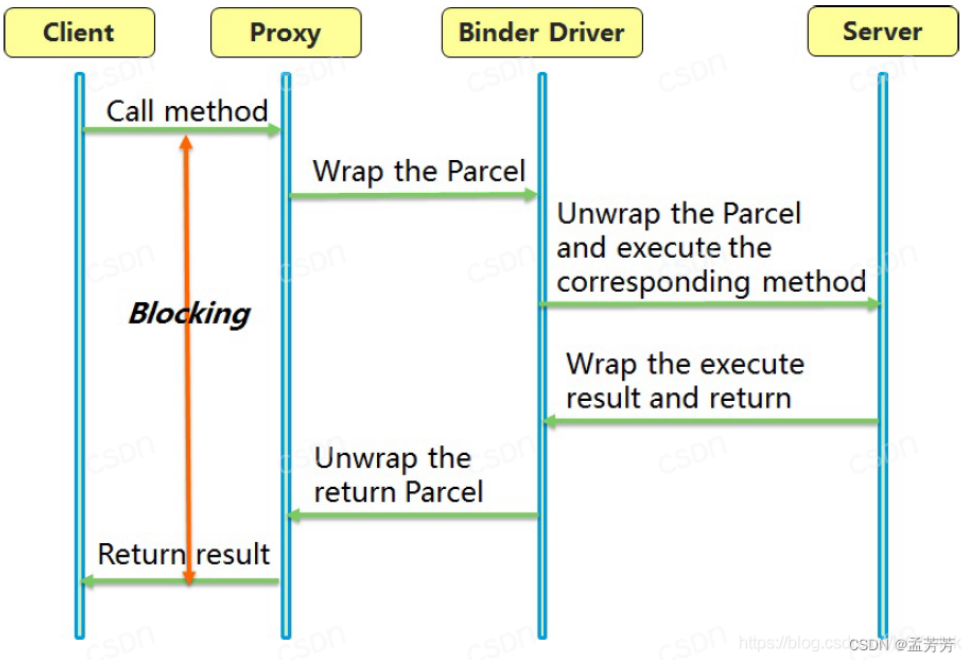
程序跨进程调用系统服务的简单示例，实现浮动窗口部分代码：

```
//获取WindowManager服务引用

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

//布局参数layoutParams相关设置略...

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

//添加view

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。



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中定义的方法是一一对应的。

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。