IPC机制

# 概念

## IPC

IPC是**Inter Process Communication**的缩写，即进程间通信或者跨进程通信，是指两个进程之间进行数据交换的过程。

## AIDL

AIDL是**Android Interface Definition Language**的缩写，即Android接口定义语言，在项目就是中以.aidl结尾的文件。

## Parcelable序列化

在Android开发中最常用到的就是对象的序列化，一般对象序列化实现Serializable即可，但是Serializable不具备进程间通信的能力，所以在IPC机制中是通过实现Parcelable完成序列化。

## Binder

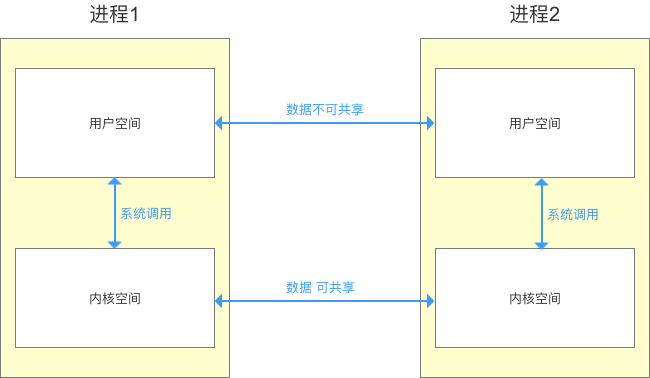
Binder的中文即粘合剂，意思为粘合了两个不同的进程。Binder是跨进程通信方式，它实现了IBinder接口，是连接 ServiceManager的桥梁。

## 进程

进程一般指一个执行单元，在PC或者移动程序指一个程序或者一个应用；一个进程可以包含多个线程，如一个应用程序中有一个UI线程和多个子线程。在Android开发中，一个应用程序可以有多个进程；一个应用使用多进程只有一个方法，就是给Android中的四大组件在清单文件制定android:process属性。

## 进程空间分配

一个进程空间分为用户空间和内核空间（Kernel），即把进程内用户与内核隔离开来。二者的区别是：进程间用户空间数据不可共享，所以用户空间是不可共享空间；进程间内核空间数据可以共享，所以内核空间就是可共享空间。进程中的用户空间和内核空间的交互成为系统调用。



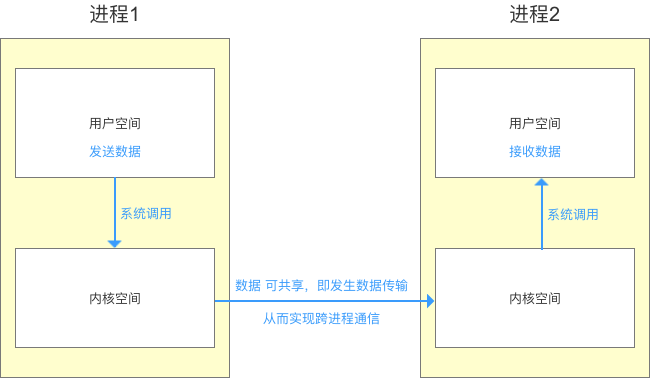
## 进程隔离

为了保证进程的安全性和独立性，一个进程不能直接操作或者访问另一个进程，即Android的进程是相互独立、隔离的。

## 进程间数据交互

进程隔离后，由于某些需求需要进程间合作交互完成，进程间数据交互通过以下两步实现：

1. 先通过进程间的内核空间进行数据交互；
2. 再通过进程内的用户空间与内核空间进行数据交互，从而实现进程间的用户空间的数据交互。



从上面的图可以看出，最重要的部分在于如何实现两个进程的内核空间的数据交互，事实上Android系统已经封装好连接两个进程的通道。

# IPC实现方式

常用实现进程间通信的方式有Bundle、文件共享、Messenger 、AIDL、ContentProvider等。

## Bundle

Android中的四大组件都支持在Intent中传递Bundle数据，而且Bundle实现了Parcelable接口，所以可在不同的进程间传输；Bundle中传输的数据必须能够被序列化，此种方式比较简单，就不详细介绍了。

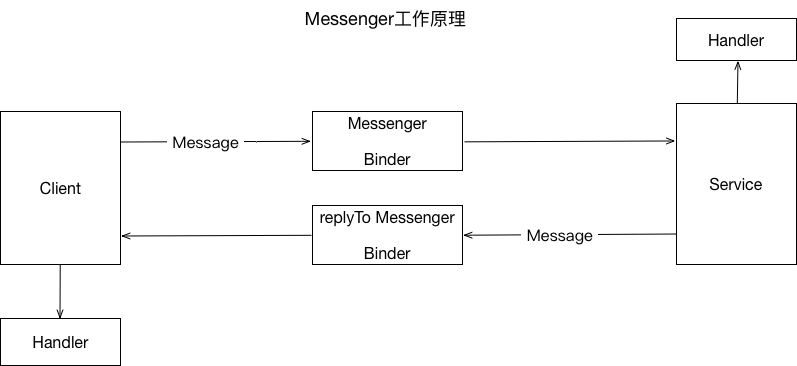
## 文件共享

通过文件共享的方式两个进程通过读写同一个文件的数据，从而达到进程间数据共享的效果。文件共享对文件的格式没有具体严格的要求，但需要两个进程约定好文件的格式和数据的格式；当两个进程同时对文件进行读写的操作时，就可能会产生数据不同步甚至脏数据。

从上述的描述中我们可以知道，文件共享的方式适合对数据同步要求不高的进程间的通信，并且要妥善处理数据的并发读写问题。

## Messenger

Messenger的中文翻译时信使，顾名思义，通过它可以在不同的进程中传递Message对象，在Message中放入我们需要传递的数据，就可以轻松地实现数据的进程间传递了。Messenger是一种轻量级的IPC方式，它的底层实现是AIDL。

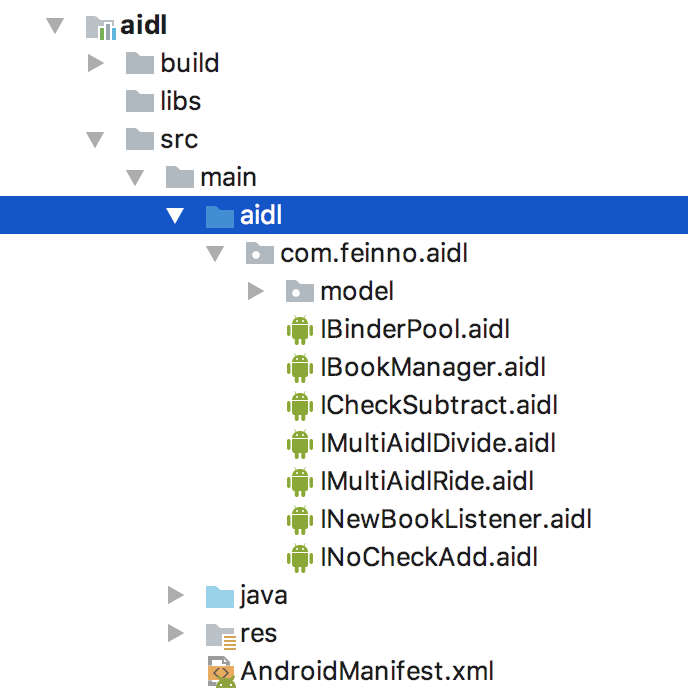




## AIDL

### 创建AIDL文件

在项目中创建aidl文件夹，并在文件夹中创建.aidl文件，如下图：

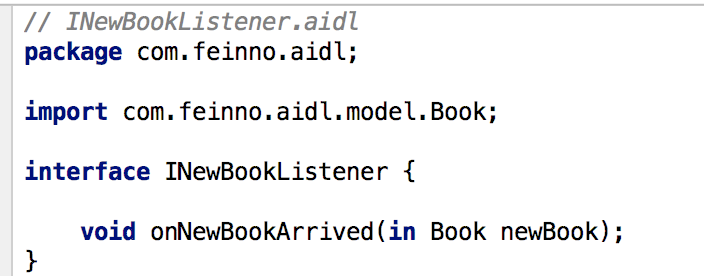


在创建.aidl文件时有以下几点需要注意：

#### AIDL文件的种类

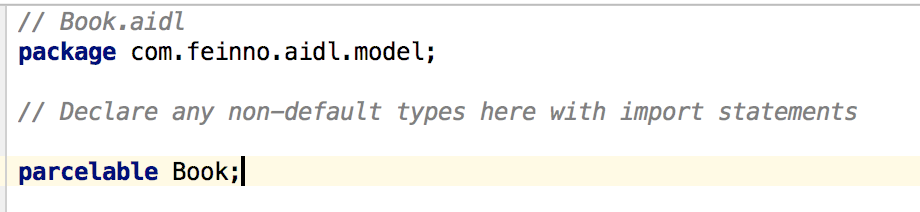
.aidl文件有两种，一种是接口，一种是实体类。

##### 接口AIDL文件



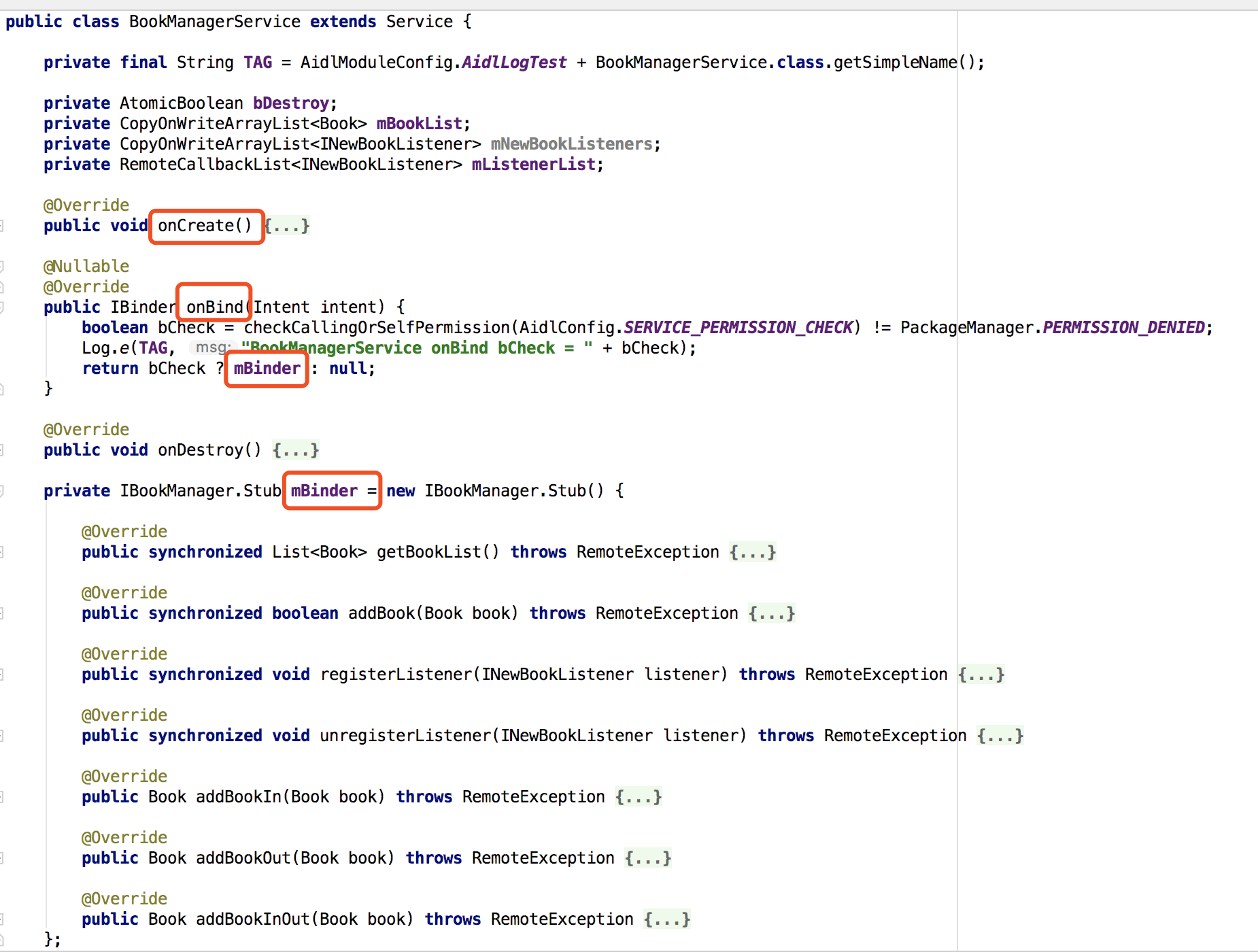
在接口aidl文件中，如果是基本数据类型时，方法中的参数定向tag有且只能是in，若不是基本数据类型参数定向tag默认是in，也可以是out和inout；若方法参数中有非基本数据类型时，不管该参数定义的aidl和接口aidl文件是否在同一个包名下，都需要手动引入包名，如Book必须import com.feinno.aidl.model.Book。

##### 实体类AIDL文件



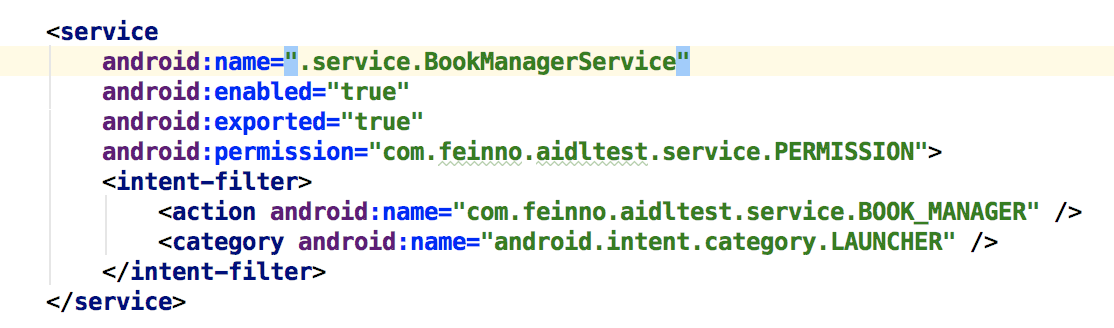
在实体类aidl文件只需要是：parcelable 包名 即可，但是要注意Book.aidl的包名必须和真正的Book.java的包名一致。

### 服务端创建Service



创建一个BookManagerService继承Service，并重写onCreate()和onBind()两个方法，在onBind()方法中返回一个IBinder对象；这个IBinder对象就是IBookManager.aidl编译生成的IBookManager类的内部类Stub，并重写IBookManager.aidl中定义的接口方法，做具体的实现。

BookManagerService需要在清单文件中申明，如下：



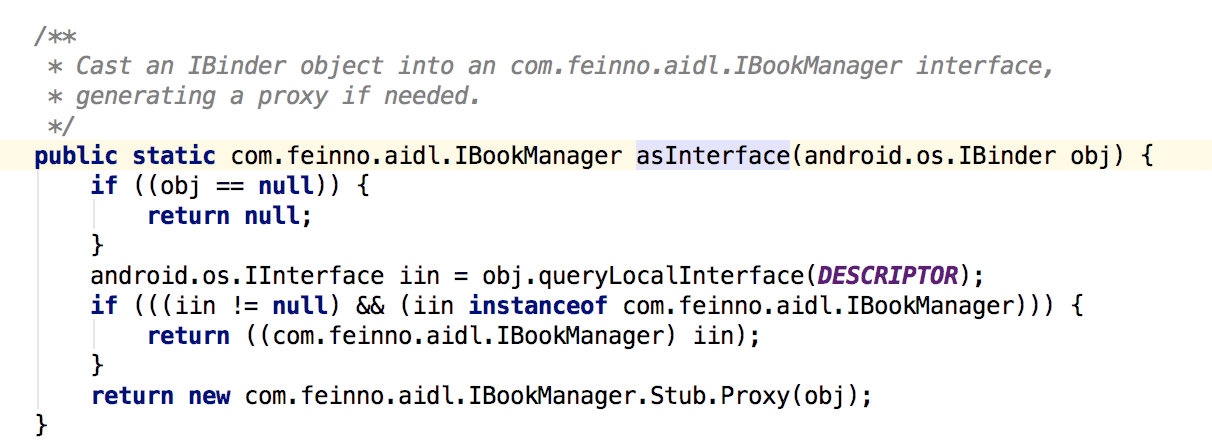
在清单文件中申明BookManagerService的action和权限，众所周知在隐式绑定服务的时候需要指定action，为了AIDL通信方式安全性，可以采用自定权限的方式，关于AIDL通信安全的问题，后来详细叙述。

### 客户端绑定服务

客户端需要通过绑定服务的方式获取到服务端到IBinder对象，并转化为IBookManager类型，实现如下：



客户端主动调用绑定服务bindService方法，绑定成功后返回服务端的IBinder对象，通过asInterface将IBinder对象转化为IBookManager对象，如下：



客户端获取到的IBookManager对象，可以直接通过IBookManager对象调用服务端端提供的能力，如下：



通过以上的详细描述，可以得出AIDL的使用的大致流程：

1. 创建一个AIDL文件和Service
2. 接着创建一个类实现AIDL接口中的Stub类（也可以直接new一个AIDL接口中的Stub类对象），并实现Stub中的抽象方法；并在Service的onBinder返回A这个类的对象；
3. 在客户端绑定Service，绑定成功建立连接后就可以访问服务端的方法了。

# Binder连接池

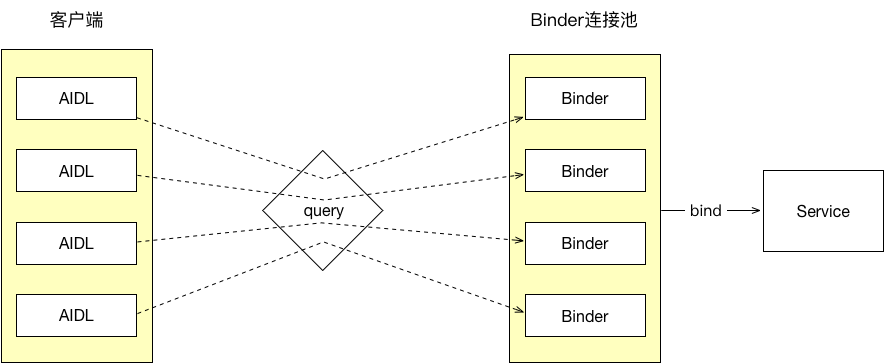
以上介绍了常用IPC实现方式，最常用的方式就是AIDL的方式，通常来说一个Service的onBind方法返回一个IBinder，而在实际的项目开发中，特别是大型的项目，可能会多个业务模块用到AIDL进行进程间通信，那我们应该怎么处理？难道需要在服务端创建多个Service来做吗？Service多了并定会造成资源的消耗，带来性能问题，能不能通过一个Service完成多个AIDL的任务呢？

在这种模式下，整个的工作机制是：

1. 每个业务模块创建自己的AIDL接口，
2. 在服务端的Service中，提供一个queryBinder接口，这个接口能够根据不同的标识，返回对应的Binder对象；
3. 不同的业务模块获取到不同的Binder对象即可进行远程方法调用。

由此可见，Binder连接池的主要作用就是将每个业务模块的Binder请求统一转发到远程Service进程，避免创建多个Service。

## Binder连接池工作原理



# IPC方式的比较选择

IPC方式根据不同的业务场景选择不同的方式，每种方式都有自己的优缺点，但是只要能够完成业务开发需要和软件性能的要求，开发者可自行选择。

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 优点 | 缺点 | 适用的场景 |
| Bundle | 简单易用 | 只能传输Bundle支持的数据类型 | 四大组件间的进程间通信 |
| 文件共享 | 简单易用 | 不适合高并发的场景，并且无法做到进程间的即时通信 | 无并发的访问情形，交换简单的数据实时性不高的场景 |
| AIDL | 功能强大，支持一对多并发通信，支持实时通信 | 使用稍负责，需要处理好线程同步 | 一对多通信且有RPC需求 |
| Messenger | 功能一般，支持一对多的穿行通信和实时通信 | 不能很好的处理高并发情绪，不支持RPC，数据通过Mesasge传输，因此只能传输Bundle支持的数据类型 | 低并发的一对多的即时通信，无RPC需求 |
| ContentProvider | 在数据访问方面功能强大，支持一对多数据共享，可通过call方法扩展其它操作 | 可以理解为受约束的AIDL，主要是提供数据源的CRUD操作 | 一对多进程间数据共享 |
| Socket | 功能强大，可以通过网络传输字节流，支持一对多并发实时通信 | 实现细节复杂，不支持直接的RPC | 网络数据交换 |

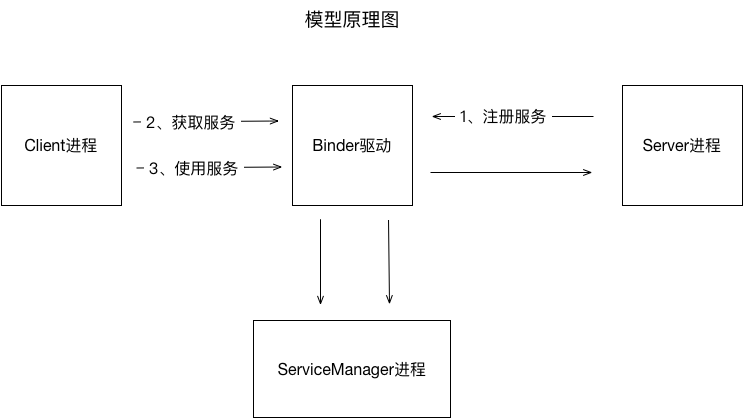
Android中的RPC机制是为了实现一个进程使用另一个进程中的远程对象，它使用了Android自己的AIDL（接口定义语言），使用户很方便地定义出一个接口作为规范，通过一个远程Service为代理 ，客户端在绑定该远程Service过程中获取远程对象，进而使用该对象。RPC的另一个目的是对客户端只声明接口及方法，隐藏掉具体实现类，供客户端直接获取此接口实例。

# Binder实现跨进程基本原理

## 模型原理

Binder 跨进程通信机制 模型 基于 Client - Server 模式，模型原理图如下：

IPC通信步骤



### Client进程

使用服务的进程

### Server进程

提供服务的进程

### ServiceManager进程

管理Service的注册与查询（将字符串中Binder的名字转化为Client对该Binder的引用），类似路由器。

### Binder驱动

Binder驱动是一种虚拟设备驱动，是连接Server进程、Client 进程和ServiceManager进程的桥梁，具体作用为：

传递进程间的数据，当Client向Service发起IPC请求时，Client会先将数据从用户空间拷贝到内核空间；数据被拷贝到内核空间后，Binder驱动将内核空间中的数据拷贝到Server位于到用户空间中。实现线程控制：采用Binder线程池，并由Binder驱动自身进行管理。

Binder驱动持有每个Server进程在内核空间中的Binder实体，并给Client进程提供Binder实体的引用。

### 注册服务

注册服务分为三步：

1. Server进程向Binder 驱动发去服务注册请求；
2. Binder 驱动将服务注册请求转发给ServiceManager进程；
3. ServiceManager添加该Server进程（即已注册服务）。

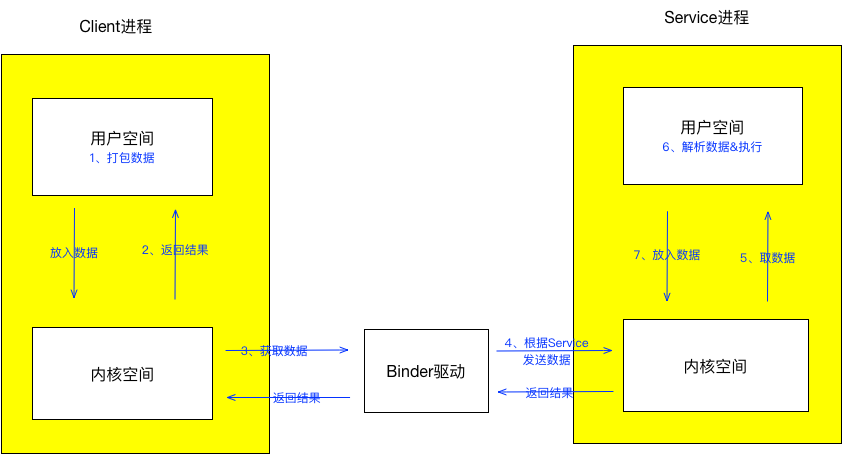
### 获取服务

获取服务分为四步：

1. Client进程向Binder驱动发起获取服务的请求，传递要获取的服务名称；
2. Binder驱动将该请求转发到ServiceManager进程；
3. ServiceManager进程查找Client进程需要的Server进程的服务信息；
4. 通过Binder驱动将服务信息返回给Client 进程。

### 使用服务

使用具体步骤见下图：



#### 步骤1

Client进程将参数数据发送到Server进程

1. Client进程将需要传递的数据放入到Client的共享内存中（当前现场被挂起）；
2. Binder 驱动从Client进程的共享内存中读取数据，并根据ServiceManager进程里的Server信息找到对应的Server 进程；
3. Binder 驱动将数据拷贝到Server进程的共享内存中，并通知Server 进程进行处理数据。

#### 步骤2

Server进程调用对应的方法

1. 收到Binder 驱动通知后，Server进程从线程池中取出线程，进行数据解包和调用目标方法；
2. 将最终执行的结果写入到共享内存中。

#### 步骤3

Server进程将目标方法的执行结果返回给Client进程

1. Binder 驱动将Server进程共享内存中的执行结果数据拷贝到Client进程的共享内存中；
2. 通知Client进程获取执行结果（此时之前被挂起的线程会被重新唤醒）。

## 详细说明

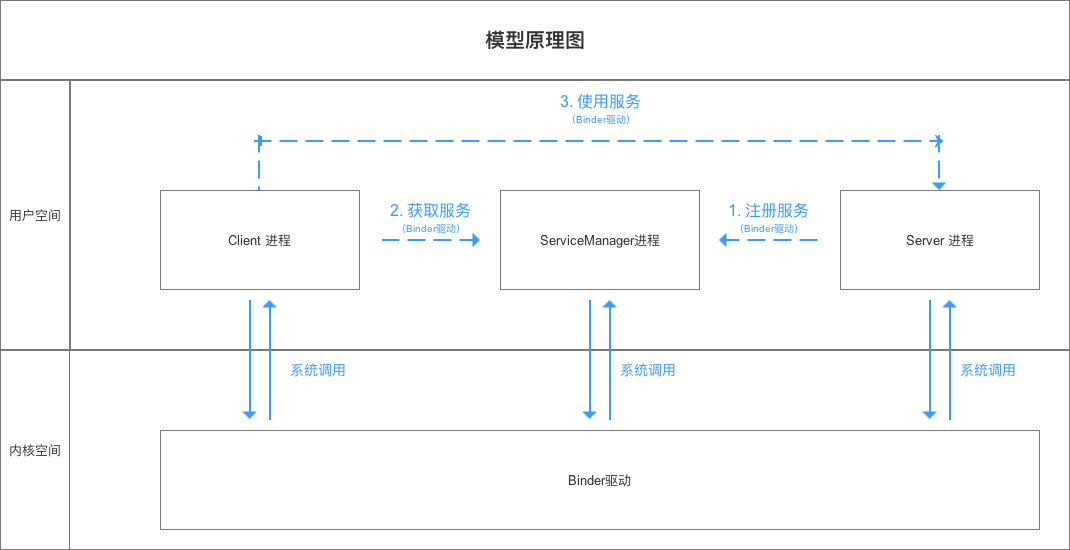
### 说明1

Client进程、Server进程和 ServiceManager 进程之间的交互都必须通过Binder驱动，而非直接交互，原因如下：

1、Client进程、Server进程 和Service Manager进程属于进程空间的用户空间，不可进行进程间交互

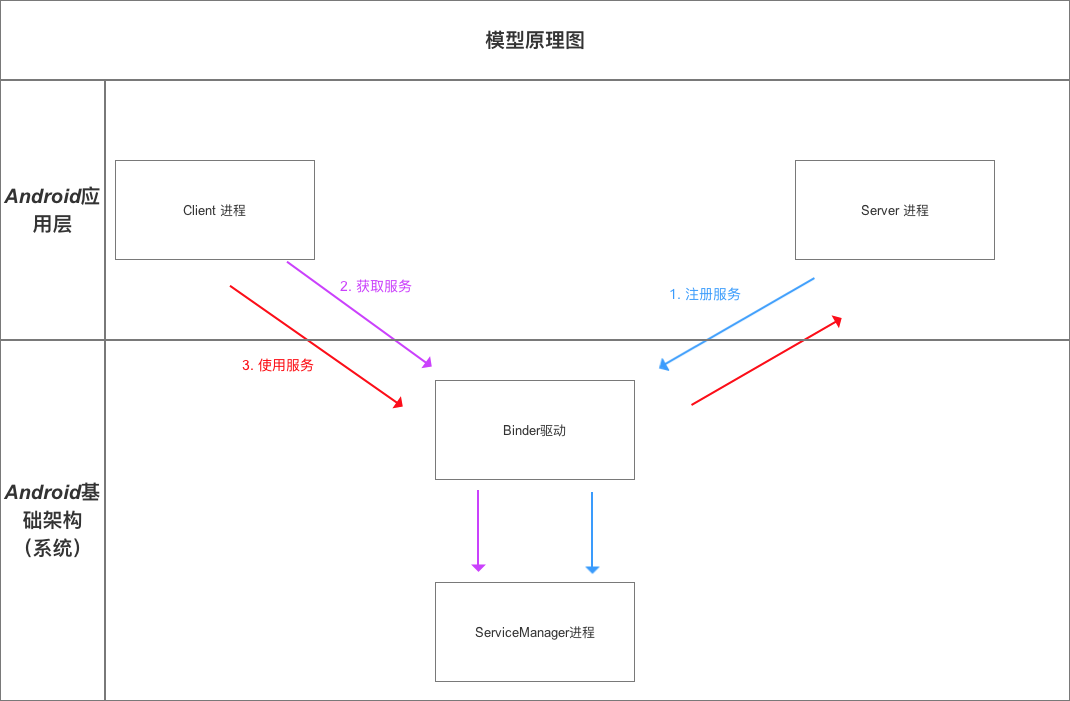
2、Binder驱动属于进程空间的内核空间，可进行进程间和进程内交互

所以，原理图可表示如下：



### 说明2

Binder驱动和Service Manager进程属于 Android基础架构（即系统已经实现好了）；而Client 进程 和 Server 进程 属于Android应用层（需要开发者自己实现）。所以，在进行跨进程通信时，开发者只需自定义Client 和Server 进程 并 显式使用上述3个步骤，最终借助 Android的基本架构功能就可完成进程间通信。



### 说明3

Server进程会创建很多线程来处理Binder请求。

管理Binder模型的线程是采用Binder驱动的线程池，并由Binder驱动自身进行管理 ，而不是由Server进程来管理；一个进程的Binder线程数默认最大是16，超过的请求会被阻塞等待空闲的Binder线程。所以，在进程间通信时处理并发问题时，如使用ContentProvider时，它的CRUD(创建、检索、更新和删除)方法只能同时有16个线程同时工作。