文档模板

1.Android Binder IPC 机制

android 系统中大量使用了基于 C/S 模式的通信方式。诸如短信操作，电话操作，视频音频捕获，传感器等都以服务（Service）的形式提供，并由相应的 Server 负责管理，
应用程序作为 Client 只需要与这些 Server 建立连接并发送请求便能使用这些服务。因此，开发者完全不必关心 Service 的实现细节，直接与 Server 建立连接然后使用其提供的接
口即可。Client 和 Server 一般是运行在不同的进程中的，这就涉及到进程间通信（IPC,Inter-Process Communication）。为了保证系统安全性，提高通信效率以及提供对 C/S
模式的支持，Android 采用了一种基于共享内存的 IPC 机制——Binder 机制。

2.Binder 机制实现

为了实现 Binder IPC 机制，Android 在 **Linux** 内核挂载了一个虚拟的设备/dev/binder。Client 和 Server 进程运行在用户空间，使用 Binder 机制进行进程间通信时，双方看起来是“直接”通信的，实际上通过/dev/binder 的驱动程序即 Binder 驱动进行了数据的中转，如图 1 所示。Binder 机制的本质是共享内存，共享内存区的管理完全由 Binder 驱动来完成，对应用层的 Client 和 Server
来说是完全透明的。

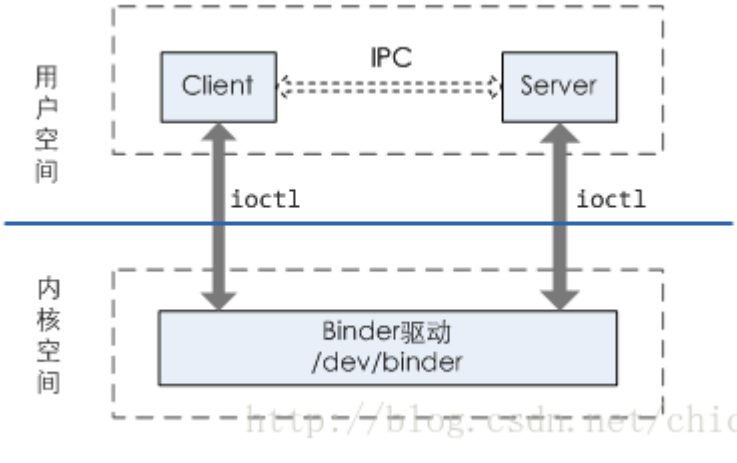

图 1 Binder 进程间通信模型

Client 和 Server 均通过函数 ioctl 与 Binder 驱动进行数据交互。ioctl 是 **linux** 中用于控制 I/O 设备的函数，提供了一种同时向设备发送控制参数和数据的手段。它是一个可变参数的函数，原型为：

[java] view plain copy

```
1. int ioctl(int fd, int cmd, ...);
```

fd 是打开/dev/binder 设备后得到的文件描述符，cmd 是对设备的控制命令。该函数执行成功返回 0，否则返回-1。

3.相关数据结构

当函数 ioctl 的第二个参数 cmd 为 BINDER_WRITE_READ 时，表示向 Binder 驱动发送一条读取或者写入/dev/binder 设备的命令，Binder 驱动会将对设备的读写“翻译”为对共享内存区的读写。这条命令是 Client 和 Server 进行进程间通信时最重要、使用最频繁的控制命令。

传入 BINDER_WRITE_READ 的同时，会传入一个 binder_write_read 结构体的指针作为 ioctl 的第三个参数，该结构中的 read_buffer 和 write_buffer 字段分别指向将要读取或者写入的缓冲区。这两个缓冲区中的数据都是以“数据类型+数据内容”的格式顺序存放的，而且多条不同类型的数据连续存放，如图 2 所示。write_buffer 中数据类型以“BC_”开头，而 read_buffer 中数据类型以“BR_”开头，图 2 中以 write_buffer 中的数据为例。在所有的数据类型中，又以 BC(R)_REPLY 和 BC(R)_TRANSACTION 最为重要：通过 BC_TRANSACTION/BC_REPLY 这对命令，发送方将数据发往接受方；通过 BR_TRANSACTION/BR_REPLY，接收方读取发送方发来的数据。

数据的内容是一个 binder_transaction_data 结构。

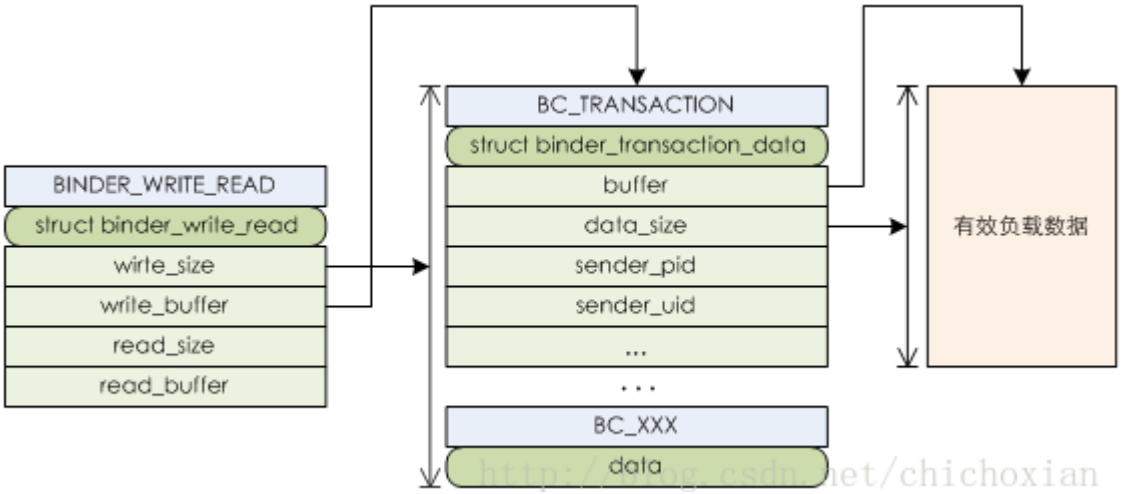

图 2 Binder IPC 中各**数据结构**的关系

binder_transaction_data 结构是对进程间通信数据的封装，可以看作网络通信中的一个数据包。其中的 sender_uid，sender_pid 成员变量指明了此数据发送方的用户 ID 和进程 ID，buffer 成员变量指向进程间通信最核心的有效负载数据，data_size 是有效负载数据的长度。在 Binder 机制中，sender_uid 和 sender_pid 是在内核中由 Binder 驱动填入的，无法被伪造，保证了身份标记的可靠性，由此可见 Binder 进程间通信机制进行是安全的。

4.Binder 之间的数据行为关系

Client 和 Server 使用 Binder 机制进行进程间通信时，通过分析 Client 发往 Server 的数据或者分析 Server 读取的 Client 的请求数据，便可以识别出 Client 的具体行为。

例如，当 Client 想要得到定位信息，请求 LocationServer 获取定位数据时，会访问 LocationServer 的 ILocationManager 接口，发往 LocationServer 中的有效负载数

据中包含“android.location.ILocationManager”字符串。所以我们分析 LocationServer 读取的 Client 发来的请求数据，判断其中是否包含“android.location.ILocationManager”，
我们就可以知道 Client 是否正在试图访问用户的地理位置信息。
由于 Android 系统中每个应用程序都有自己唯一的 UID，因此根据 binder_transaction_data 中的 sender_uid，我们就可以获取 Client 具体代表的应用程序。这样
就获得了具体软件的具体行为。
通过对 Android 系统的分析，我们发现：虽然系统提供的服务多达几十种，但是实际上只有三个 Server 进程负责管理

Android 系统 Server 进程与管理的服务

服务进程	管理的服务
com.android.phone	与通信功能相关的短信、电话服务
mediaserver	与媒体功能相关的视频、音频服务
system_server	其他服务，如地理位置、蓝牙、网络连接、程序安装卸载等