[Android进程间通信（IPC）机制Binder简要介绍](http://blog.csdn.net/luoshengyang/article/details/6618363)

在Android系统中，每一个应用程序都是由一些Activity和Service组成的，一般Service运行在独立的进程中，而Activity有可能运行在同一个进程中，也有可能运行在不同的进程中。那么，不在同一个进程的Activity或者Service是如何通信的呢？这就是本文中要介绍的Binder进程间通信机制了。

        我们知道，Android系统是基于Linux内核的，而Linux内核继承和兼容了丰富的Unix系统进程间通信（IPC）机制。有传统的管道（Pipe）、信号（Signal）和跟踪（Trace），这三项通信手段只能用于父进程与子进程之间，或者兄弟进程之间；后来又增加了命令管道（Named Pipe），使得进程间通信不再局限于父子进程或者兄弟进程之间；为了更好地支持商业应用中的事务处理，在AT&T的Unix系统V中，又增加了三种称为“System V IPC”的进程间通信机制，分别是报文队列（Message）、共享内存（Share Memory）和信号量（Semaphore）；后来BSD Unix对“System V IPC”机制进行了重要的扩充，提供了一种称为插口（Socket）的进程间通信机制。若想进一步详细了解这些进程间通信机制，建议参考[Android学习启动篇](http://blog.csdn.net/luoshengyang/article/details/6557518)一文中提到《Linux内核源代码情景分析》一书。

        但是，Android系统没有采用上述提到的各种进程间通信机制，而是采用Binder机制，难道是因为考虑到了移动设备硬件性能较差、内存较低的特点？不得而知。Binder其实也不是Android提出来的一套新的进程间通信机制，它是基于[OpenBinder](http://www.angryredplanet.com/~hackbod/openbinder/docs/html/BinderIPCMechanism.html)来实现的。OpenBinder最先是由[Be Inc.](http://en.wikipedia.org/wiki/Be_Inc.)开发的，接着[Palm Inc.](http://en.wikipedia.org/wiki/Palm,_Inc.)也跟着使用。现在OpenBinder的作者[Dianne Hackborn](http://www.angryredplanet.com/~hackbod/)就是在Google工作，负责Android平台的开发工作。

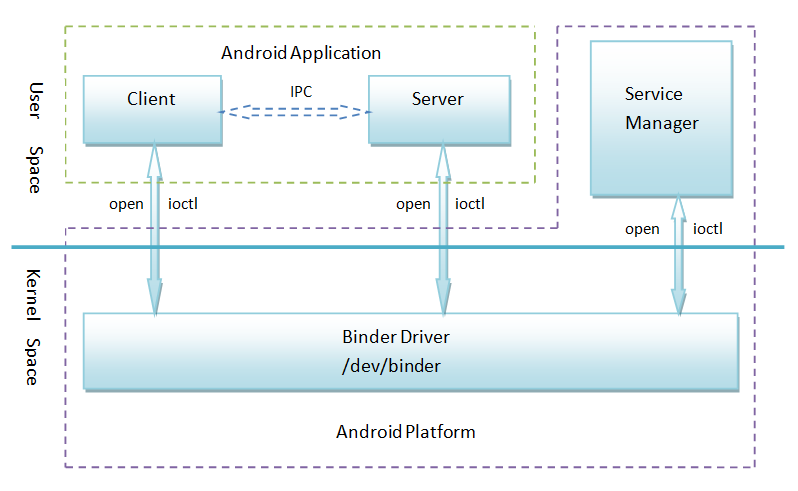
        前面一再提到，Binder是一种进程间通信机制，它是一种类似于COM和CORBA分布式组件架构，通俗一点，其实是提供远程过程调用（RPC）功能。从英文字面上意思看，Binder具有粘结剂的意思，那么它把什么东西粘结在一起呢？在Android系统的Binder机制中，由一系统组件组成，分别是Client、Server、Service Manager和Binder驱动程序，其中Client、Server和Service Manager运行在用户空间，Binder驱动程序运行内核空间。Binder就是一种把这四个组件粘合在一起的粘结剂了，其中，核心组件便是Binder驱动程序了，Service Manager提供了辅助管理的功能，Client和Server正是在Binder驱动和Service Manager提供的基础设施上，进行Client-Server之间的通信。Service Manager和Binder驱动已经在Android平台中实现好，开发者只要按照规范实现自己的Client和Server组件就可以了。说起来简单，做起难，对初学者来说，Android系统的Binder机制是最难理解的了，而Binder机制无论从系统开发还是应用开发的角度来看，都是Android系统中最重要的组成，因此，很有必要深入了解Binder的工作方式。要深入了解Binder的工作方式，最好的方式莫过于是阅读Binder相关的源代码了，Linux的鼻祖Linus Torvalds曾经曰过一句名言RTFSC：Read The Fucking Source Code。

        虽说阅读Binder的源代码是学习Binder机制的最好的方式，但是也绝不能打无准备之仗，因为Binder的相关源代码是比较枯燥无味而且比较难以理解的，如果能够辅予一些理论知识，那就更好了。闲话少说，网上关于Binder机制的资料还是不少的，这里就不想再详细写一遍了，强烈推荐下面两篇文章：

[Android深入浅出之Binder机制](http://www.cnblogs.com/innost/archive/2011/01/09/1931456.html)

[Android Binder设计与实现 – 设计篇](http://disanji.net/2011/02/28/android-bnder-design/)

        Android深入浅出之Binder机制一文从情景出发，深入地介绍了Binder在用户空间的三个组件Client、Server和Service Manager的相互关系，Android Binder设计与实现一文则是详细地介绍了内核空间的Binder驱动程序的数据结构和设计原理。非常感谢这两位作者给我们带来这么好的Binder学习资料。总结一下，Android系统Binder机制中的四个组件Client、Server、Service Manager和Binder驱动程序的关系如下图所示：



        1. Client、Server和Service Manager实现在用户空间中，Binder驱动程序实现在内核空间中

        2. Binder驱动程序和Service Manager在Android平台中已经实现，开发者只需要在用户空间实现自己的Client和Server

        3. Binder驱动程序提供设备文件/dev/binder与用户空间交互，Client、Server和Service Manager通过open和ioctl文件操作函数与Binder驱动程序进行通信

        4. Client和Server之间的进程间通信通过Binder驱动程序间接实现

        5. Service Manager是一个守护进程，用来管理Server，并向Client提供查询Server接口的能力

        至此，对Binder机制总算是有了一个感性的认识，但仍然感到不能很好地从上到下贯穿整个IPC通信过程，于是，打算通过下面四个情景来分析Binder源代码，以进一步理解Binder机制：

        1. [Service Manager是如何成为一个守护进程的？即Service Manager是如何告知Binder驱动程序它是Binder机制的上下文管理者。](http://blog.csdn.net/luoshengyang/article/details/6621566)

        2. [Server和Client是如何获得Service Manager接口的？即defaultServiceManager接口是如何实现的。](http://blog.csdn.net/luoshengyang/article/details/6627260)

        3. [Server是如何把自己的服务启动起来的？Service Manager在Server启动的过程中是如何为Server提供服务的？即IServiceManager::addService接口是如何实现的。](http://blog.csdn.net/luoshengyang/article/details/6629298)

        4  [Service Manager是如何为Client提供服务的？即IServiceManager::getService接口是如何实现的。](http://blog.csdn.net/luoshengyang/article/details/6633311)

        在接下来的四篇文章中，将按照这四个情景来分析Binder源代码，都将会涉及到用户空间到内核空间的Binder相关源代码。这里为什么没有Client和Server是如何进行进程间通信的情景呢？ 这是因为Service Manager在作为守护进程的同时，它也充当Server角色。因此，只要我们能够理解第三和第四个情景，也就理解了Binder机制中Client和Server是如何通过Binder驱动程序进行进程间通信的了。

        为了方便描述Android系统进程间通信Binder机制的原理和实现，在接下来的四篇文章中，我们都是基于C/C++语言来介绍Binder机制的实现的，但是，我们在Android系统开发应用程序时，都是基于Java语言的，因此，我们会在最后一篇文章中，详细介绍Android系统进程间通信Binder机制在应用程序框架层的Java接口实现：

        5. [Android系统进程间通信Binder机制在应用程序框架层的Java接口源代码分析。](http://blog.csdn.net/luoshengyang/article/details/6642463)