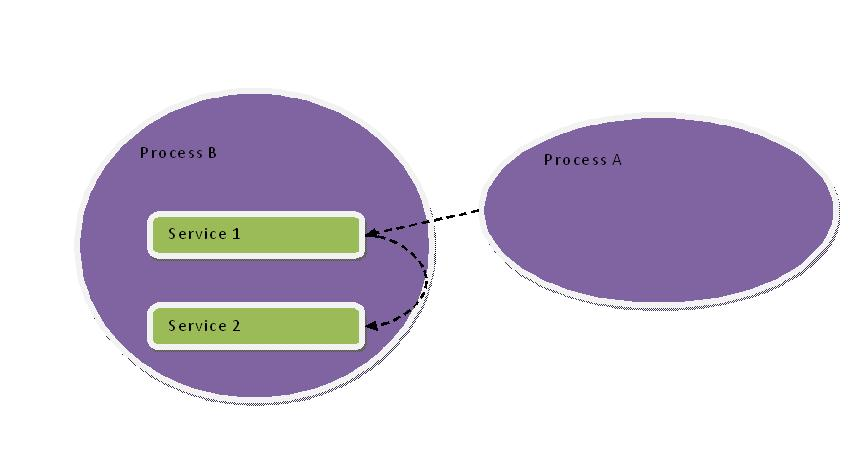
**clearCallingIdentity/restoreCallingIdentity**

客户端A调用服务端B的服务，然后B又调用自己的服务或者方法

1. process B在被process A IPC调用时, process B需知道process A的UID和PID，来检查process A的访问权限，此时mCallingUid和mCallingPid保存的是process A的UID和PID。

2. 在IPC远程调用process B的过程中，process B的方法调用了同进程中的service的接口，process B既是调用方也是被调用方，虽然这个过程比较无聊，但是鉴于IPC过程的不透明性，因此process B仍然需要进行权限检测。

在B进程自己调用自己资源的时候可通过如下两个方法

public static final native long clearCallingIdentity();

public static final native void restoreCallingIdentity(long token);

先将A的UID、PID进行备份并且替换B的UID、PID，开始操作B的资源

token = clearCallingIdentity();

调用完成后，再恢复成A的UID、PID

**binderDied**

**1. 原理**

当Binder非正常消亡的时候，会导致远程调用失败，这样客户端功能就会受到影响。

解决：给Binder设置一个死亡代理，当Binder死亡时，我们就会收到通知，这个时候可以重新发起连接。

**2. 制作**

**2.1 前期准备**

客户端：MainActivity.java

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

createService();

}

/\*连接Service端，获取mIBookManger\*/

private void createService(){

ServiceConnection connection = new ServiceConnection() {

@Override

public void onServiceConnected(ComponentName name, IBinder service) {

//初始化mIBookManger

mIBookManager = IBookManager.Stub.asInterface(service);

}

@Override

public void onServiceDisconnected(ComponentName name) {

}

};

Intent intent = new Intent(this,BookService.class);

bindService(intent,connection,BIND\_AUTO\_CREATE);

}

Service端：BookService.java

**2.2使用**

创建IBinder.DeathRecipient接口，重写其中的binderDied()，当Binder死亡时候，回调该方法。

private IBinder.DeathRecipient mDeathRecipient = new IBinder.DeathRecipient() {

@Override

public void binderDied() {

if (mIBookManager != null){

//解除绑定当前接口

mIBookManager.asBinder().unlinkToDeath(mDeathRecipient,0);

}

mIBookManager = null;

createService();

}

};

当然解除绑定之后，还需要绑定接口

/\*连接Service端，获取mIBookManger\*/

private void createService(){

...(连接之前的代码)

try {

//绑定接口,等待回调

mIBookManager.asBinder().linkToDeath(mDeathRecipient,0);

} catch (RemoteException e) {

e.printStackTrace();

}

}