把玩Android多进程

任玉刚 singwhatiwanna@gmail.com

关于我

任玉刚（singwhatiwanna），滴滴出行Android技术专家，《Android开发艺术探索》作者，热爱开源和分享，活跃在Github和CSDN，目前在滴滴出行从事APP架构相关的开发工作。

微博：@任玉刚Coder

微信公众号：

提纲

- 进程间通信的方式
- 多进程特性
- AIDL和Binder
- 一种可扩展的进程间通信模型

进程间通信的方式

- 指定四大组件的android:process属性来开启新进程

Intent	文件共享	Binder	Socket
• 基本数据类型 • 序列化数据 • IBinder • RemoteViews	• SharedPreferences • Xml等文件	• AIDL • Messenger • ContentProvider	• 字节流

进程间通信的方式-对比

名称	优点	缺点	适用场景
Intent	简单易用	只能传输Bundle所支持的数据类型	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	简单的数据共享，无高并发场景
AIDL	功能强大，支持一对多并发实时通信	使用稍微复杂，需要注意线程同步	复杂的进程间调用，Android中最常用
Messenger	比AIDL稍微简单易用些	比AIDL功能弱，只支持一对多串行实时通信	简单的进程间通信
ContentProvider	强大的数据共享能力，可通过call方法扩展	受约束的AIDL，主要对外提供数据线的CRUD操作	进程间的大量数据共享
RemoteViews	在跨进程访问UI方面有奇效	比较小众的通信方式	某些特殊的场景
Socket	跨主机，通信范围广	只能传输原始的字节流	常用于网络通信中

多进程特性

- 不同的内存空间，数据无法共享
- 每个进程都有自己单独的Application，并且会独立地创建和销毁
- 谨慎处理代码中的线程同步
- 多进程并发导致的文件锁和数据库锁失效的问题

AIDL和Binder

- Binder是Android平台中有特色的进程间通信方式；
- AIDL是最常用的进程间通信方式，是对Binder的封装；
- AIDL是app和系统进行通信的基石，消息机制和AIDL构成了Android系统最核心的两个部分。

AIDL和Binder- 概念

- Binder：概念意义的Binder，和一个BinderProxy对象相关联；
- IBinder：一个接口，Binder类实现了它，是Binder的抽象；
- IInterface：一个接口，AIDL接口必须继承它；
- asInterface()：从BinderProxy中获取IInterface对象(Stub$Proxy)；
- asBinder()：获取IInterface接口所关联的BinderProxy。

AIDL和Binder- Stub和Proxy

```
1   /*
2    * This file is auto-generated.  DO NOT MODIFY.
3    * Original file: /Users/didi/AndroidStudioProjects/Chapter_2/app/src/main/aidl/com/ryg/chapter_2/aidl/IOnNewBookAr
4    */
5   package com.ryg.chapter_2.aidl;
6   public interface IOnNewBookArrivedListener extends android.os.IInterface
7   {
8       /** Local-side IPC implementation stub class. */
9       public static abstract class Stub extends android.os.Binder implements
10                  com.ryg.chapter_2.aidl.IOnNewBookArrivedListener
11      {
12          private static final java.lang.String DESCRIPTOR = "com.ryg.chapter_2.aidl.IOnNewBookArrivedListener";
13          /** Construct the stub at attach it to the interface. */
14          public Stub() { this.attachInterface(this, DESCRIPTOR); }
18          /**
19           * Cast an IBinder object into an com.ryg.chapter_2.aidl.IOnNewBookArrivedListener interface,
20           * generating a proxy if needed.
21           */
22  @       public static com.ryg.chapter_2.aidl.IOnNewBookArrivedListener asInterface(android.os.IBinder obj)
23          {...}
33          @Override public android.os.IBinder asBinder() { return this; }
37          @Override public boolean onTransact(int code, android.os.Parcel data,
38                                              android.os.Parcel reply, int flags) throws android.os.RemoteException
39          {...}
64          private static class Proxy implements com.ryg.chapter_2.aidl.IOnNewBookArrivedListener
65          {...}
101         static final int TRANSACTION_onNewBookArrived = (android.os.IBinder.FIRST_CALL_TRANSACTION + 0);
102     }
103     public void onNewBookArrived(com.ryg.chapter_2.aidl.Book newBook) throws android.os.RemoteException;
104 }
105
```

AIDL和Binder- 同步or异步

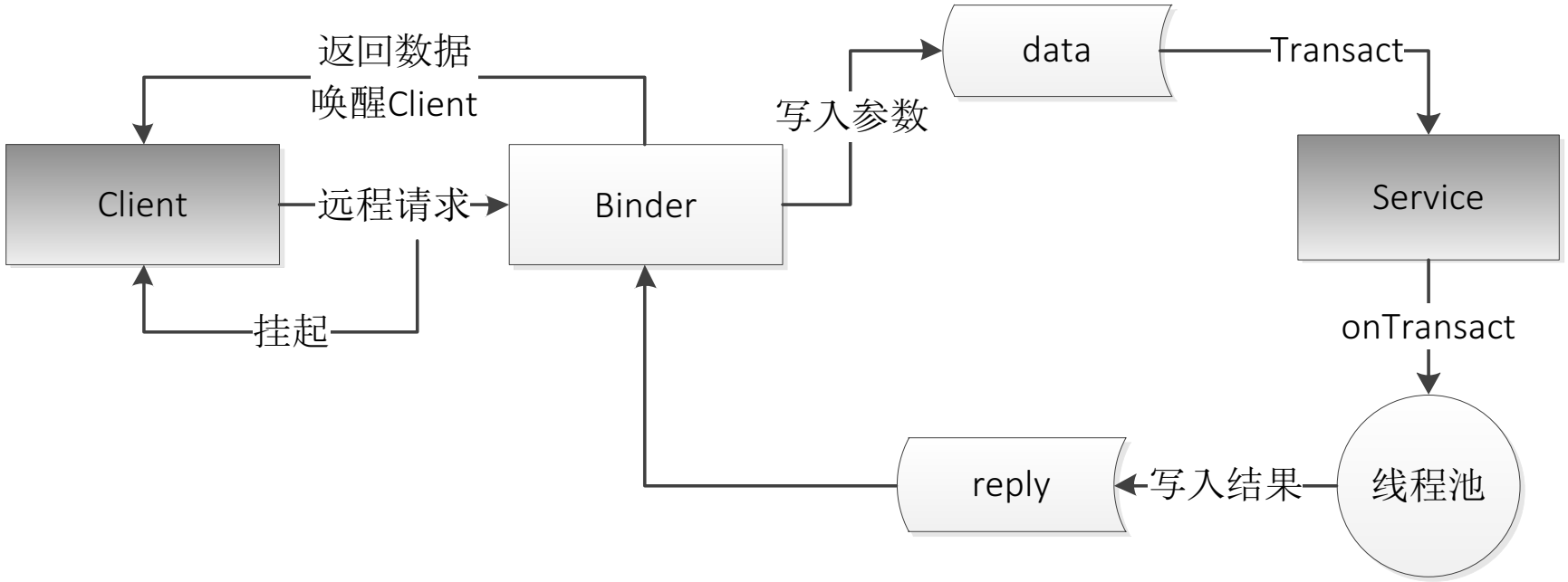
- 在多进程环境中，Binder调用默认是同步的，运行在Binder线程池中；
- 在同一个进程中，Binder调用只能是同步的，运行在调用者所在的线程中；
- 通过oneway关键字或者FLAG_ONEWAY标记位可以将Binder调用改为异步，异步Binder接口不能有返回值。

AIDL和Binder- 同步or异步

```
1  package com.ryg.chapter_2.aidl;
2
3  import com.ryg.chapter_2.aidl.Book;
4
5  oneway interface IOnNewBookArrivedListener {
6      void onNewBookArrived(in Book newBook);
7  }
8
```

```
@Override public void onNewBookArrived(com.ryg.chapter_2.aidl.Book newBook) throws android.os.RemoteException
{
    android.os.Parcel _data = android.os.Parcel.obtain();
    try {
        _data.writeInterfaceToken(DESCRIPTOR);
        if ((newBook!=null)) {
            _data.writeInt(1);
            newBook.writeToParcel(_data, 0);
        }
        else {
            _data.writeInt(0);
        }
        mRemote.transact(Stub.TRANSACTION_onNewBookArrived, _data, null, android.os.IBinder.FLAG_ONEWAY);
    }
    finally {
        _data.recycle();
    }
}
```

AIDL和Binder- 工作过程



AIDL和Binder- 异常处理

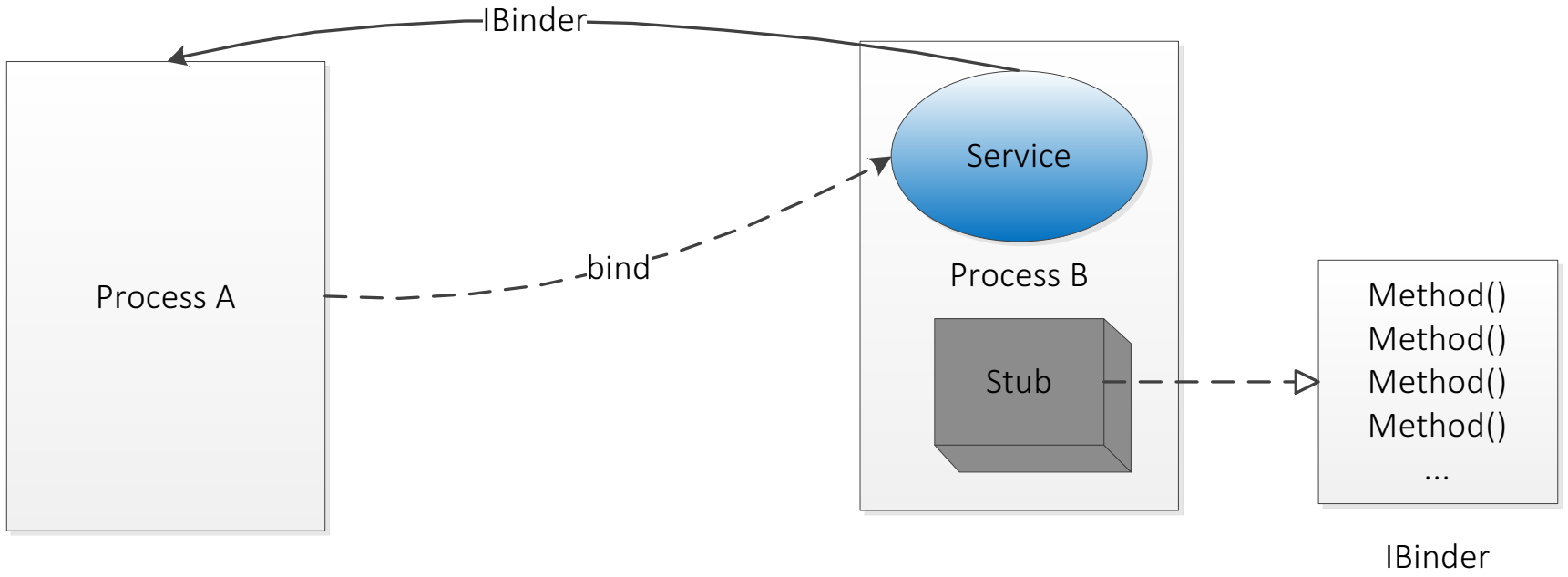
- 同步调用中，右侧图中的7种RuntimeException的子类可以抛出，这个时候本地和远程端都会crash，RuntimeException本身及其他子类、RemoteException均不会导致crash；
- 异步调用中，RuntimeException和RemoteException均不会导致本地和远程crash；
- 除此之外，理论上其他类型的异常均不会导致Binder crash，但是未经过测试证明，待验证。

```
public final void writeException(Exception e) {
    int code = 0;
    if (e instanceof SecurityException) {
        code = EX_SECURITY;
    } else if (e instanceof BadParcelableException) {
        code = EX_BAD_PARCELABLE;
    } else if (e instanceof IllegalArgumentException) {
        code = EX_ILLEGAL_ARGUMENT;
    } else if (e instanceof NullPointerException) {
        code = EX_NULL_POINTER;
    } else if (e instanceof IllegalStateException) {
        code = EX_ILLEGAL_STATE;
    } else if (e instanceof NetworkOnMainThreadException) {
        code = EX_NETWORK_MAIN_THREAD;
    } else if (e instanceof UnsupportedOperationException) {
        code = EX_UNSUPPORTED_OPERATION;
    }
    writeInt(code);
    StrictMode.clearGatheredViolations();
    if (code == 0) {
        if (e instanceof RuntimeException) {
            throw (RuntimeException) e;
        }
        throw new RuntimeException(e);
    }
    writeString(e.getMessage());
}
```

AIDL和Binder- Binder的类型

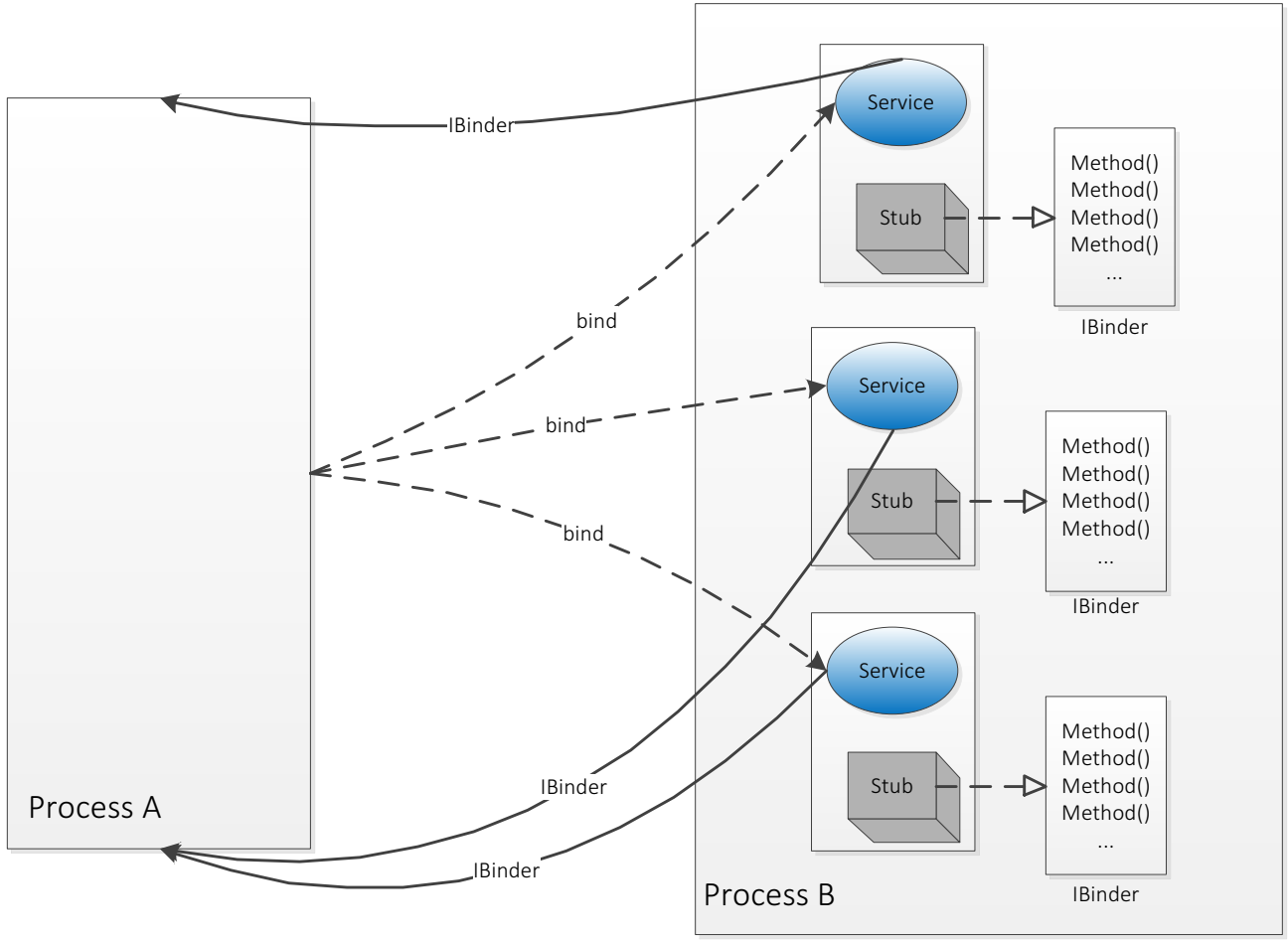
场景	Binder类型	方法示例
Service的onBind	Binder类型，一般为Stub	public IBinder onBind(Intent intent)
onServiceConnected	返回的service参数为BinderProxy	public void onServiceConnected(ComponentName name, IBinder service)
AIDL中AIDL类型的参数	① 调用端传递的是Stub ② 远程接收的是Stub$Proxy	void registerListener(IOnNewBookArrivedListener listener)
AIDL中IBinder类型的参数	① 调用端传递的是Stub ② 远程接收的是BinderProxy	void test(IBinder service)
通过Intent跨进程传递IBinder对象	① 调用端传递的是Stub ② 远程接收的是BinderProxy	public void putBinder(String key, IBinder value)

一种可扩展的进程间通信模型



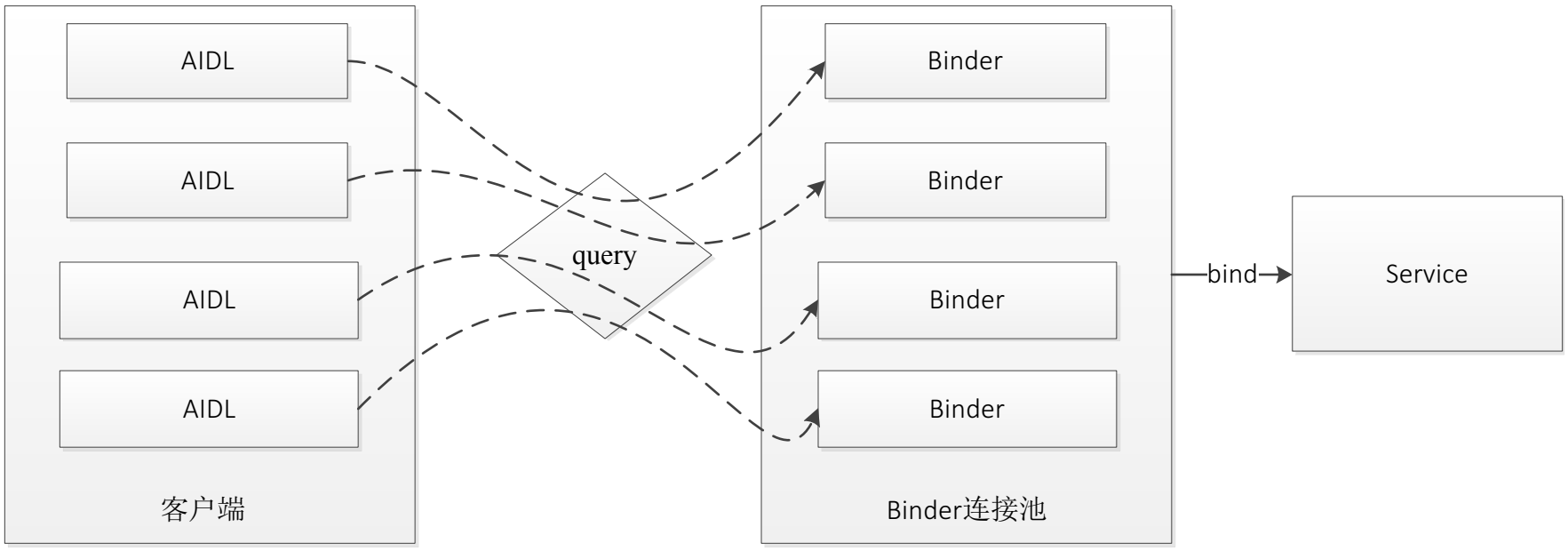
常规的进程间通信模型

一种可扩展的进程间通信模型



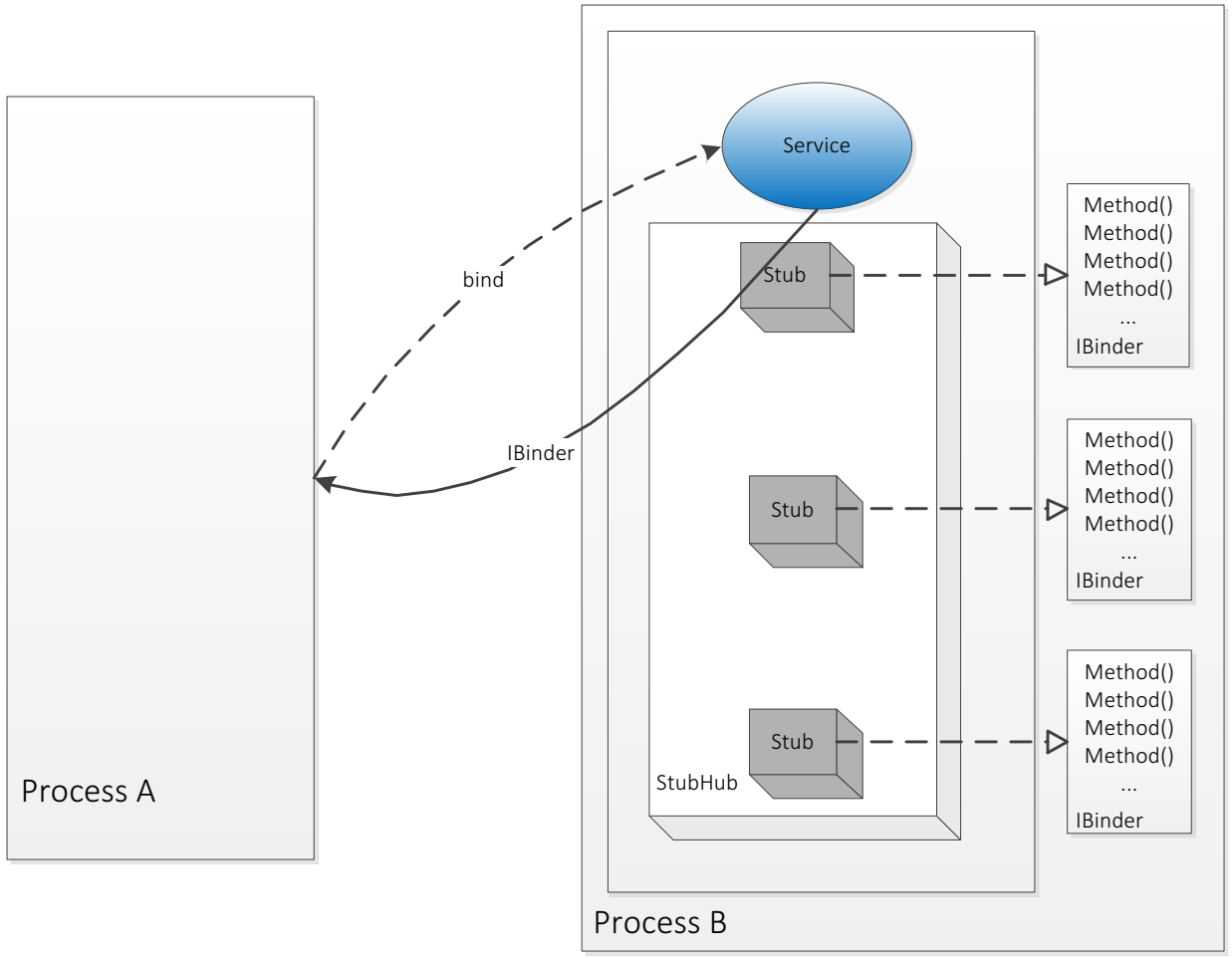
常规的进程间通信模型- 多Binder场景

一种可扩展的进程间通信模型

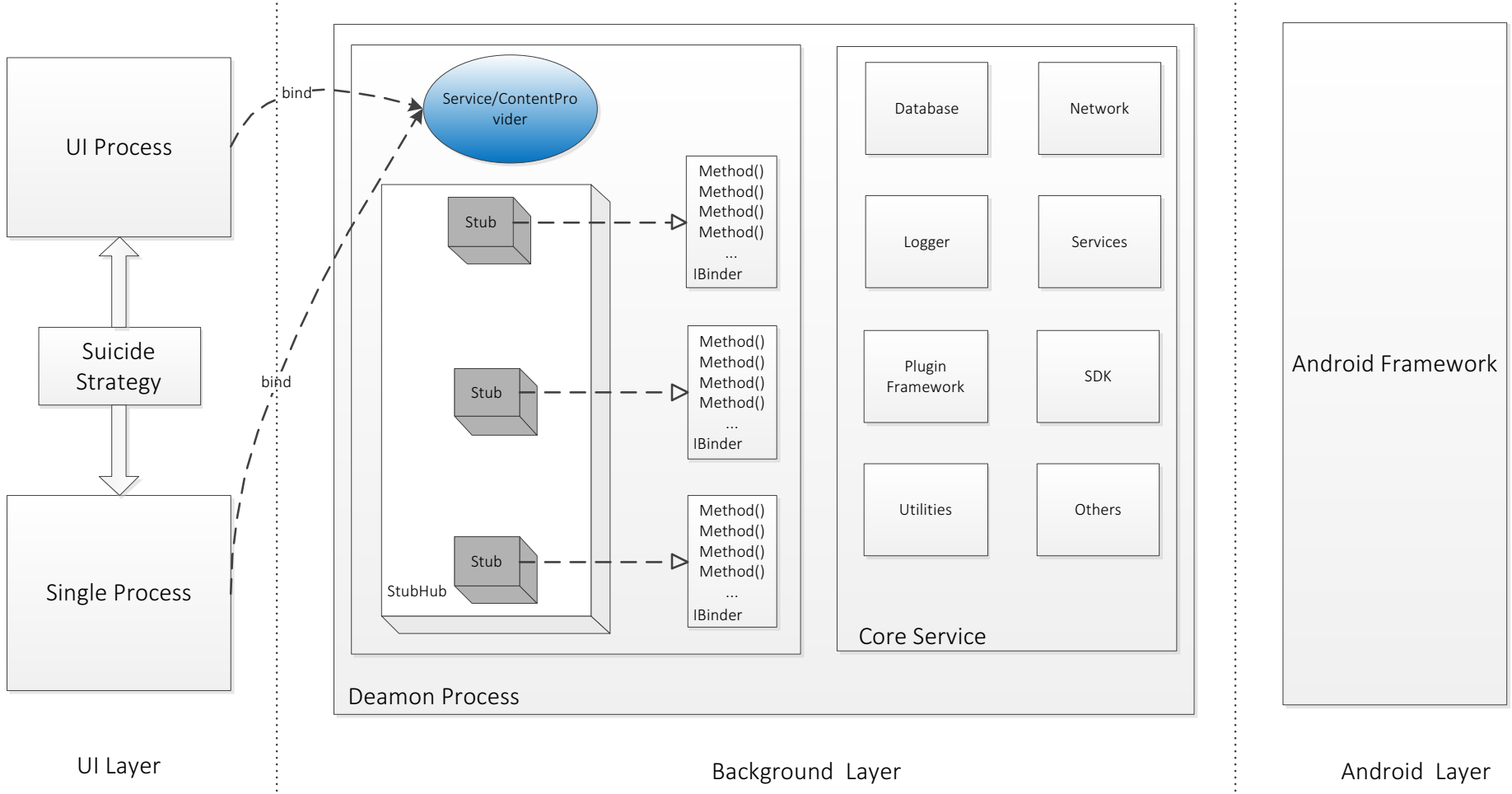


Binder连接池的概念

一种可扩展的进程间通信模型



一种可扩展的进程间通信模型



Q&A