MICROOH 麦可网

Android-从程序员到架构师之路

出品人：Sundy
讲师：高焕堂（台湾）

http://www.microoh.com

B01_c

认识进程与IPC架构(c)

By 高煥堂

4、IPC的IBinder接口
-- 定义与实现

IBinder接口的定義

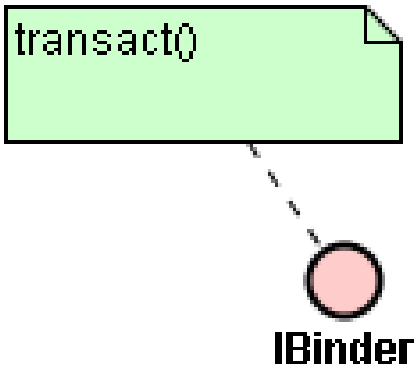
- 大家都知道，当两个类都在同一个进程里执行时，两者之间的沟通，只要采取一般的函数调用(Function Call)就行了，既快速又方便。一旦两个类分别在不同的进程里执行时，两者之间的沟通，就不能采取一般的函数调用途径了。只好采取IPC沟通途径。

- Android框架的IPC沟通仰赖单一的IBinder接口。此时Client端调用IBinder接口的transact()函数，透过IPC机制而调用到远方(Remote)的onTransact()函数。

- 在Android的源代码里，Java层的IBinder接口是定义于IBinder.java代码文档里。此程序文件如下：

```
// IBinder.java
// .......
public interface IBinder {
    // ..........
    public boolean transact(int code, Parcel data, Parcel reply, int flags)
        throws RemoteException;
   // ...........
}
```

- IBinder接口定义了一些函数，可以让Client程序可以进行跨进程的調用(当然也能支持同进程的短程調用)。其中，最主要的一个函数就是：transact()函数。于此，以图形来表达IBinder接口与transact()函数之间的关系。如下述的UML图：

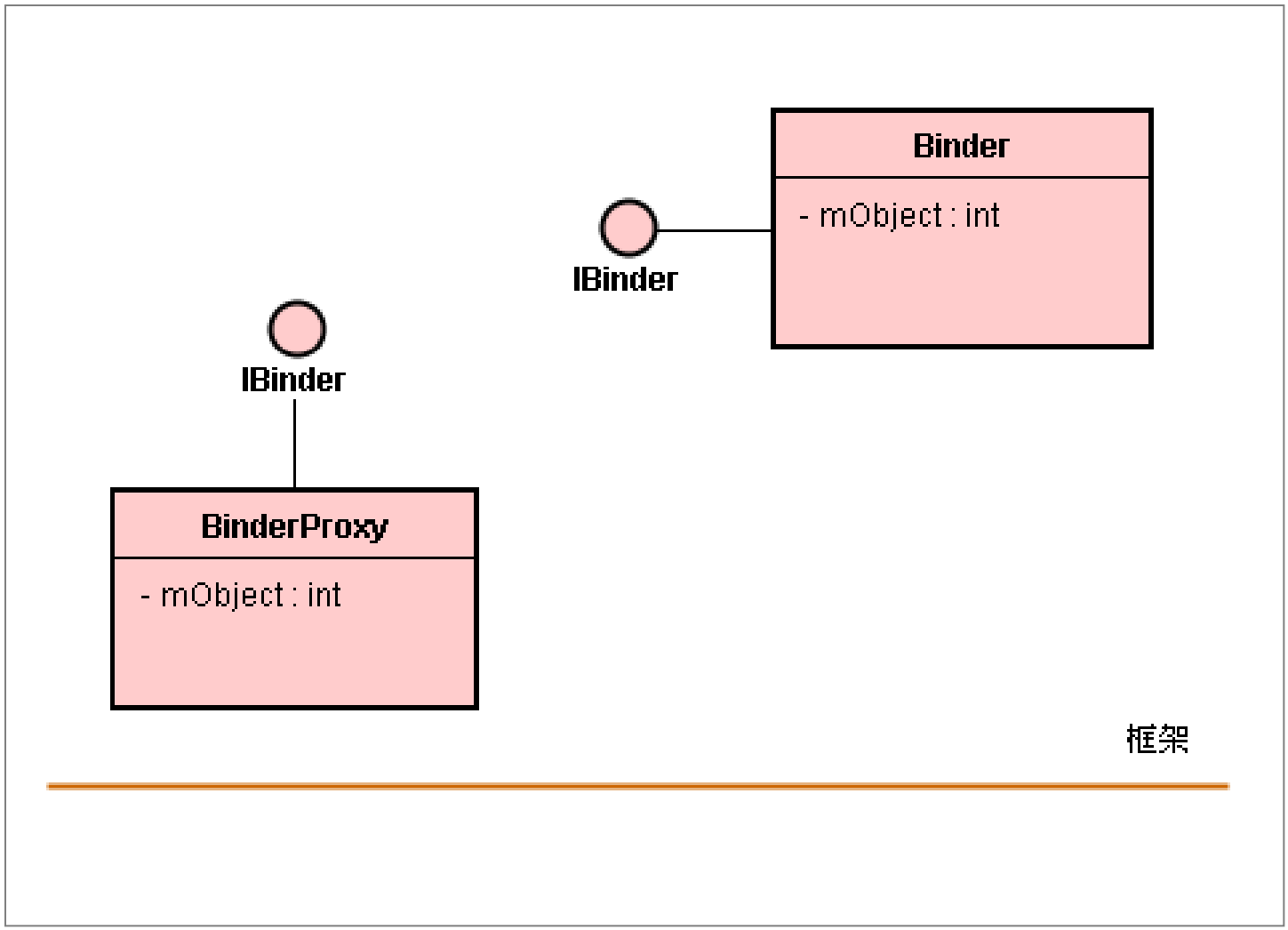


框架

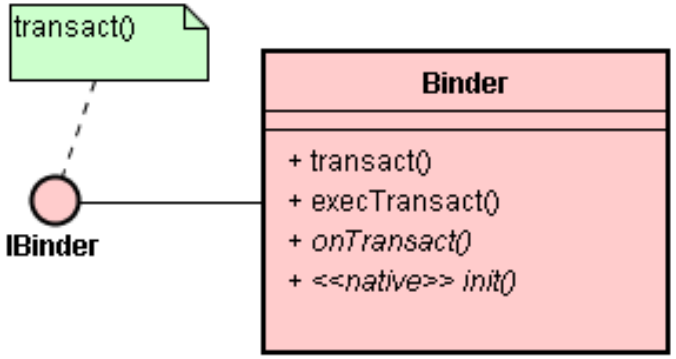
IBinder接口的實現類

- 基于这个IBinder.java定义档，我们就可以开发类别来实作(Implement)它，然后提供给其它App来調用了。在Android的框架里，也撰写了Binder基类和BinderProxy类别来实作 IBinder接口。

IBinder实作类之例：Binder和BinderProxy类



Java层的Binder基类定义



- Binder基类的很重要目的是支持跨进程調用Service，也就是让远程的Client可以跨进程調用某个Service。
- Binder基类定义于Binder.java档案里：

```
// Binder.java
// …….
public class Binder implements IBinder {
    // .....
    private int mObject;

    public Binder() {
        init();
        // .....
    }

    public final boolean transact(int code, Parcel data, Parcel reply,  int
     flags)
                throws RemoteException {
                // ..............
                boolean r = onTransact(code, data, reply, flags);
                return r;
    }
```

```
 private boolean execTransact(int code, int dataObj, int replyObj,
            int flags) {
        Parcel data = Parcel.obtain(dataObj);
        Parcel reply = Parcel.obtain(replyObj);

        boolean res;
        res = onTransact(code, data, reply, flags);
        // .....
        return res;
    }
   protected boolean onTransact(int code, Parcel data, Parcel reply,  int
     flags)
                    throws RemoteException {
        }
   private native final void init();
}
```

Binder基类的主要函数是：

- **transact()**函数

  --- 用来实作IBinder的transact()函数接口。

- **execTransact()**函数

  --- 其角色与transact()函数是相同的，只是这是用来让C/C++本地程序来調用的。

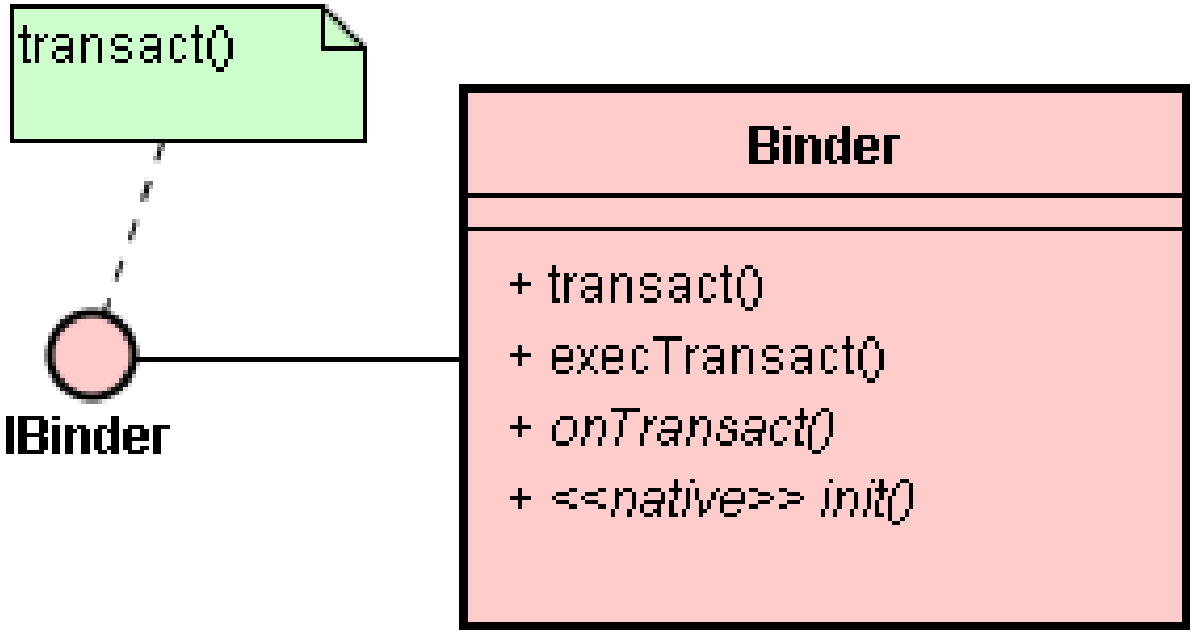
- **onTransact()**函数

  --- 这是一个抽象函数，让应用子类来覆写(Override)的。上述的transact()和execTransact()两者都是調用onTransact()函数来实现反向調用(IoC, Inversion of Control)的。

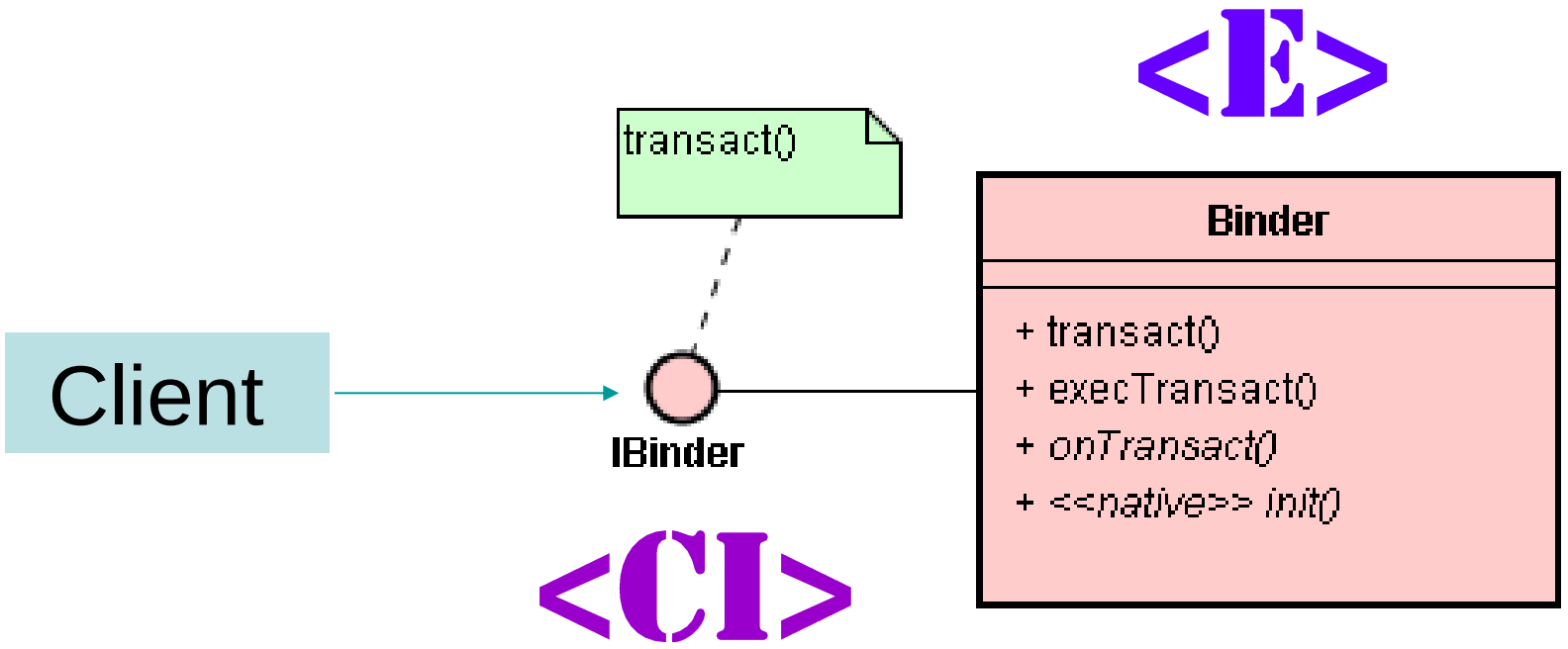
- **init()**函数

  --- 这是一个本地(Native)函数，让JNI模块来实现这个函数。Binder()构造函数(Constructor)会調用这个init()本地函数。

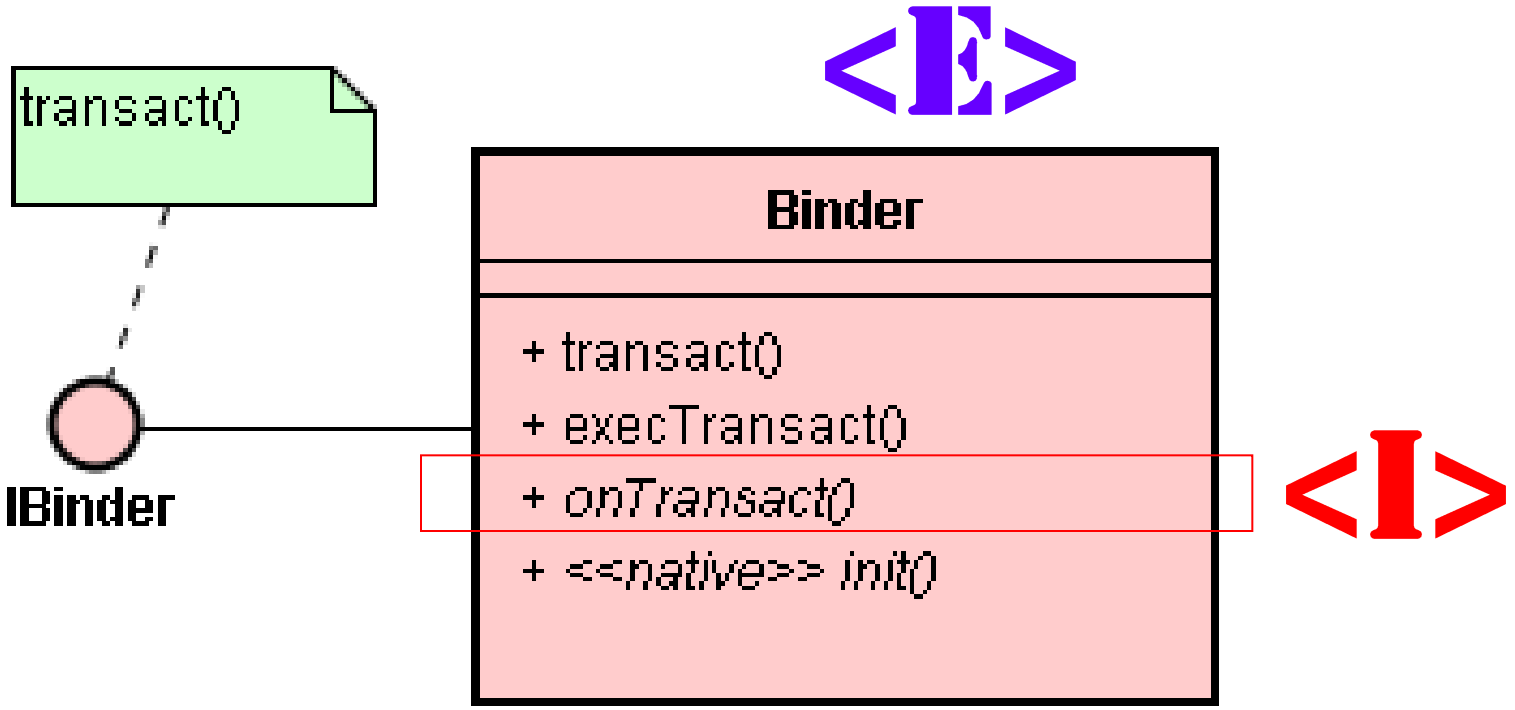
UML图形表示：


transact()
IBinder
Binder
+ transact()
+ execTransact()
+ onTransact()
+ <<native>> init()

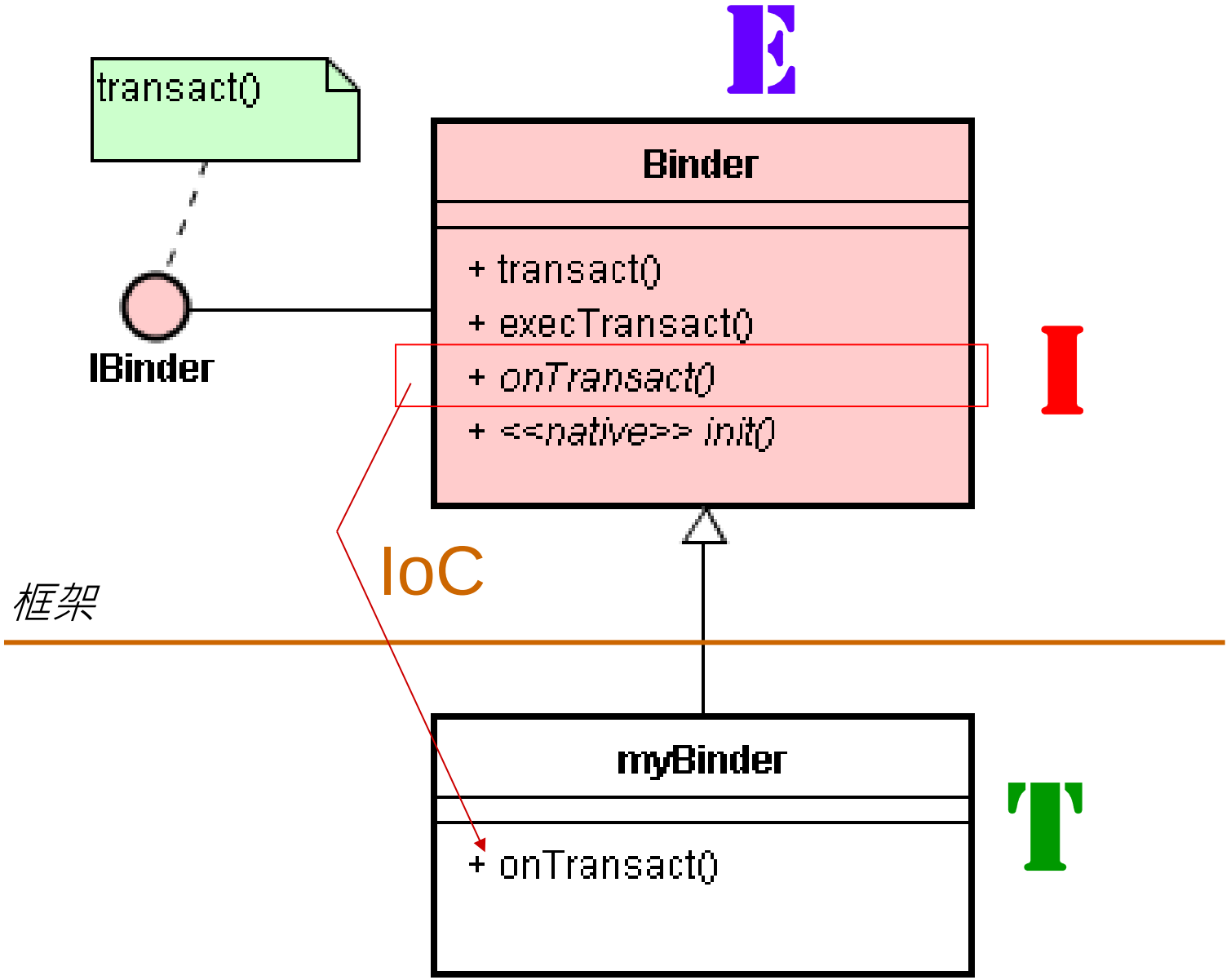
- Binder就是EIT造形里的<E>
- 这个IBinder接口是Binder(即<E>)提供给Client的接口，简称为<CI>。



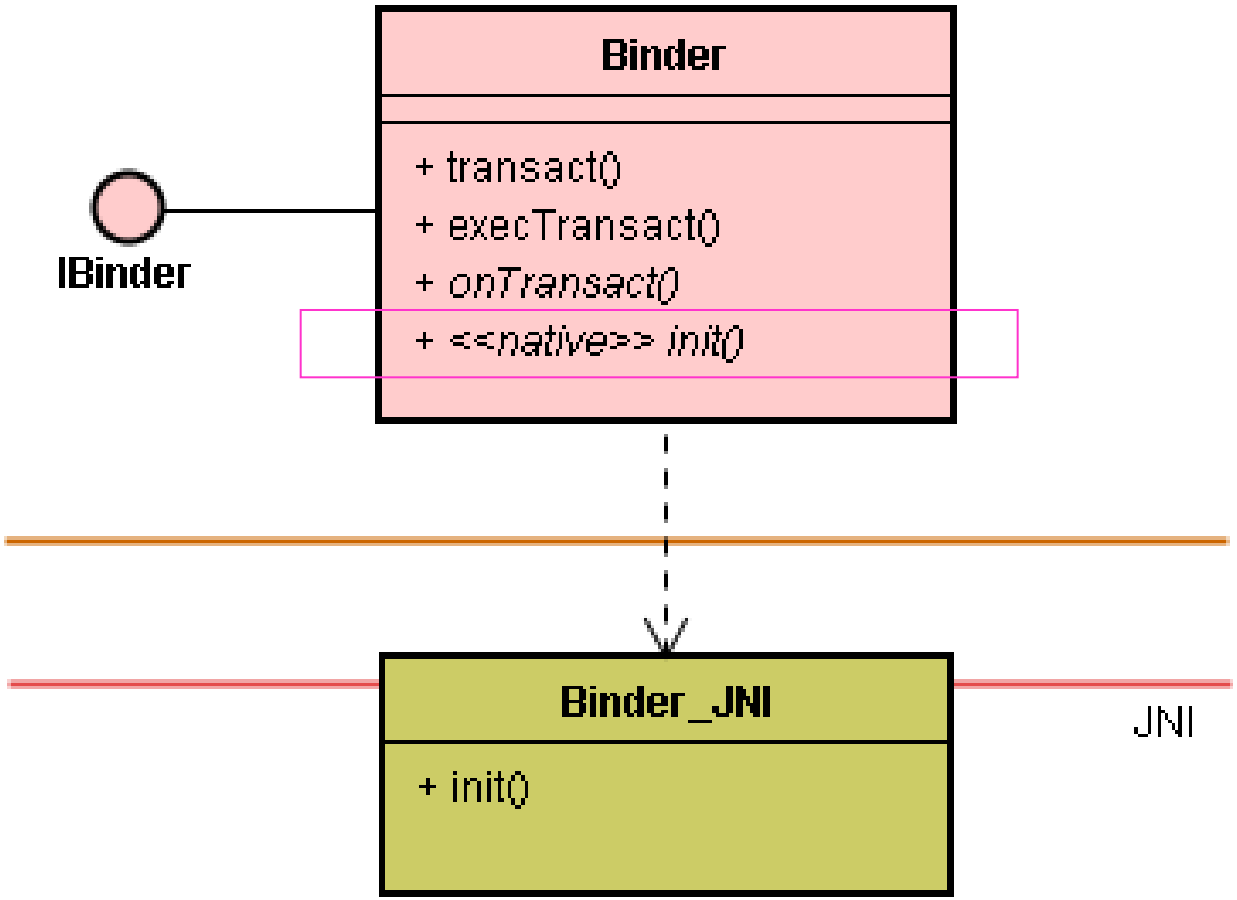
- onTransact()就是EIT造形里的<I>

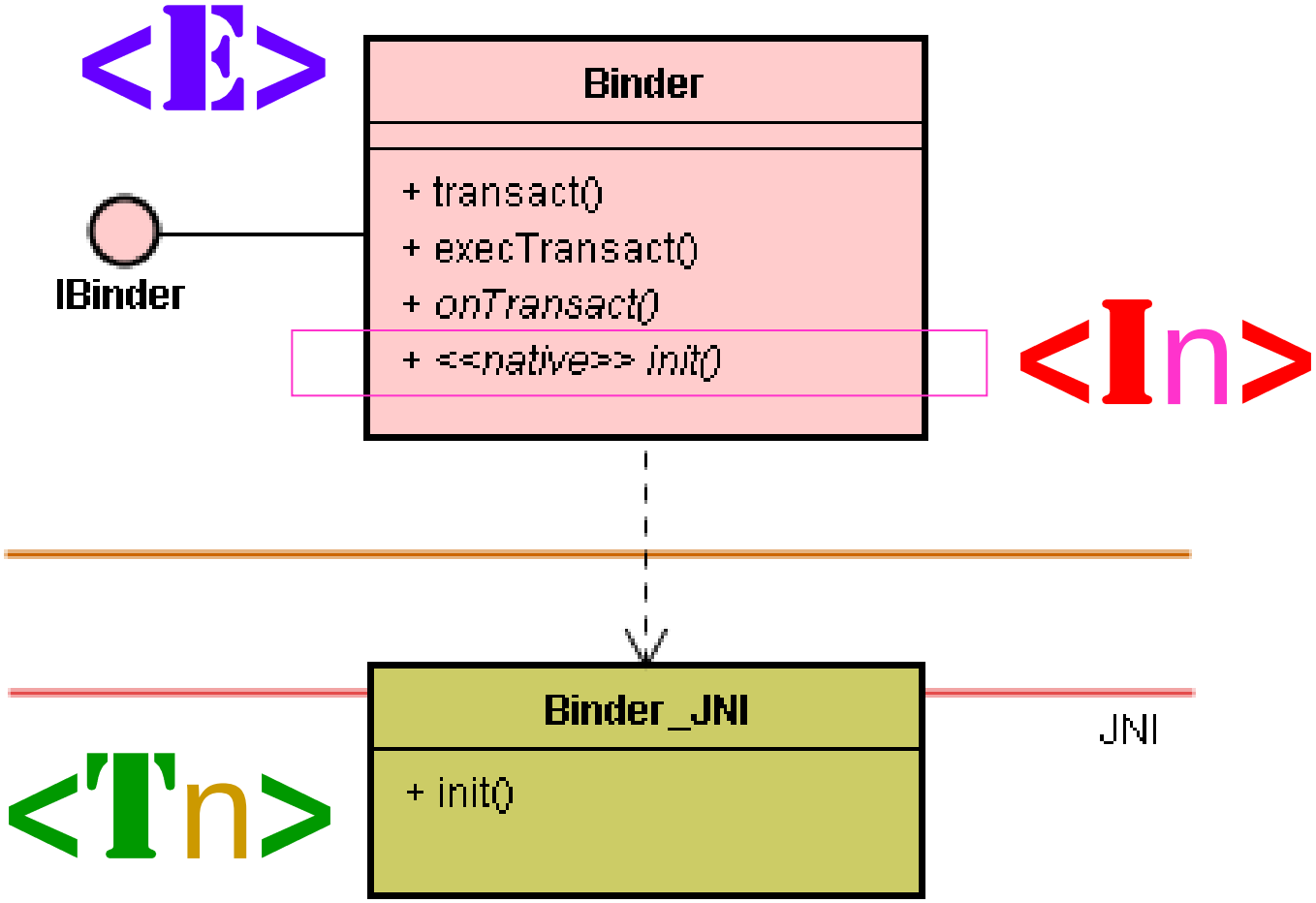


- 这是标准的EIT造形，其<I>是支持<基类/子类>之间IoC调用的接口。


E
transact()
Binder
+ transact()
+ execTransact()
+ onTransact()
+ <<native>> init()
IBinder
I
IoC
框架
myBinder
+ onTransact()
T

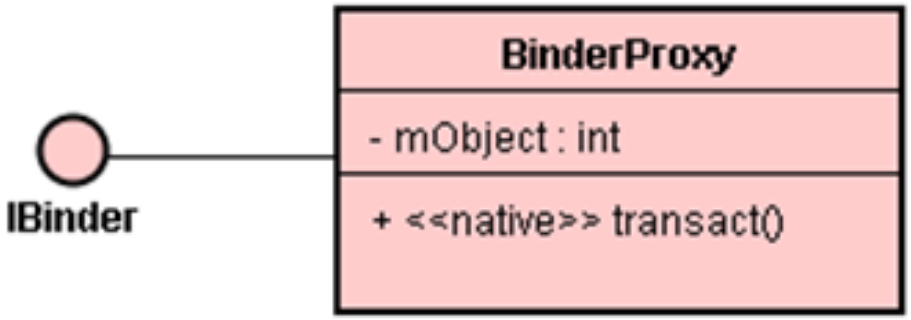
- Init()是EIT造形的另一种特殊接口，其支持<从Java到本地C>之间的调用接口。




<E>
Binder
+ transact()
+ execTransact()
+ onTransact()
+ <<native>> init()
IBinder
<In>
Binder_JNI
+ init()
JNI
<Tn>

- 当Binder的子类别诞生对象时，会調用到Binder()构造函数。此时，Binder()会調用到init()本地函数。

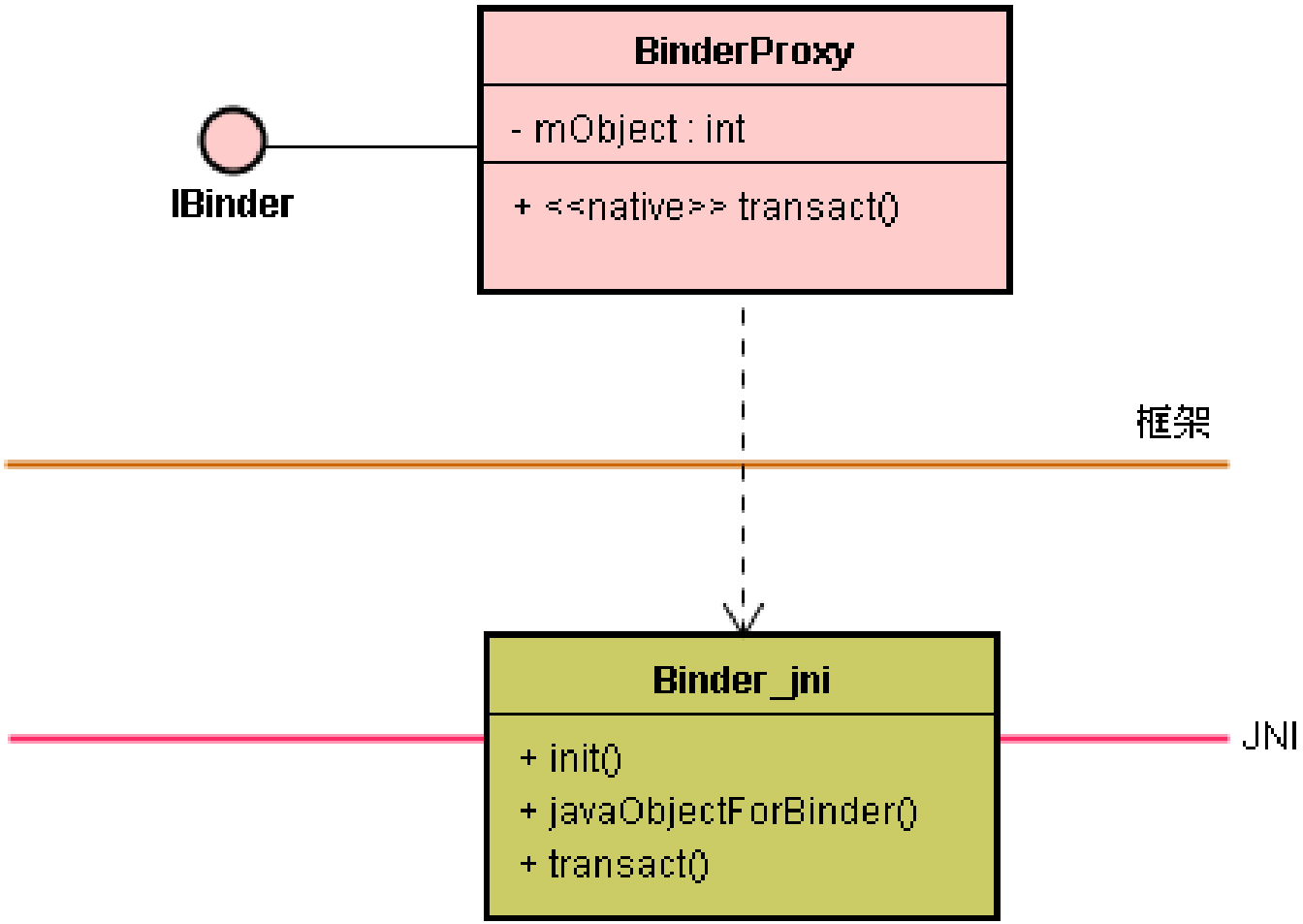
Java层的BinderProxy基类定义



- 这个BinderProxy类也定义在Binder.java档案里，如下：

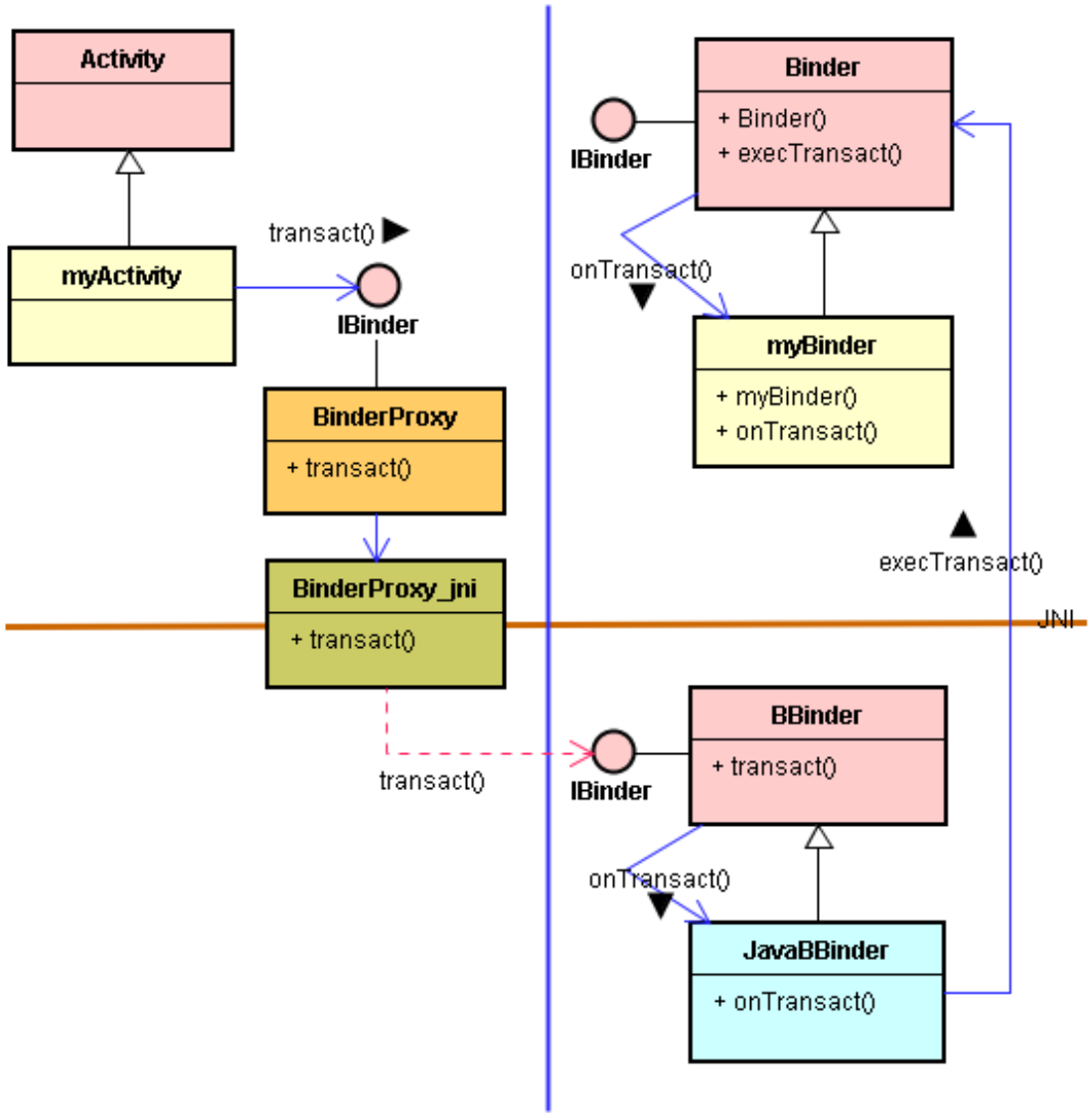
```
// Binder.java
// ……
final class BinderProxy implements IBinder {
    private int mObject;
    // .....
    BinderProxy() {
        // .....
    }
    public native boolean transact(int code, Parcel data, Parcel reply,
            int flags) throws RemoteException;
    private int mObject;
}
```

- 当我们看到类别名称是 XXXProxy时，就自然会联想到它是摆在Client进程里，担任Service端的分身(Proxy)。
- 由于跨进程沟通时，并不是从Java层直接沟通的，而是透过底层的Binder Driver驱动来沟通的，所以Client端的Java类别(如Activity)必须透过BinderProxy分身的IBinder接口，转而調用JNI本地模块来衔接到底层Binder Driver驱动服务，进而調用到正在另一个进程里执行的Service。

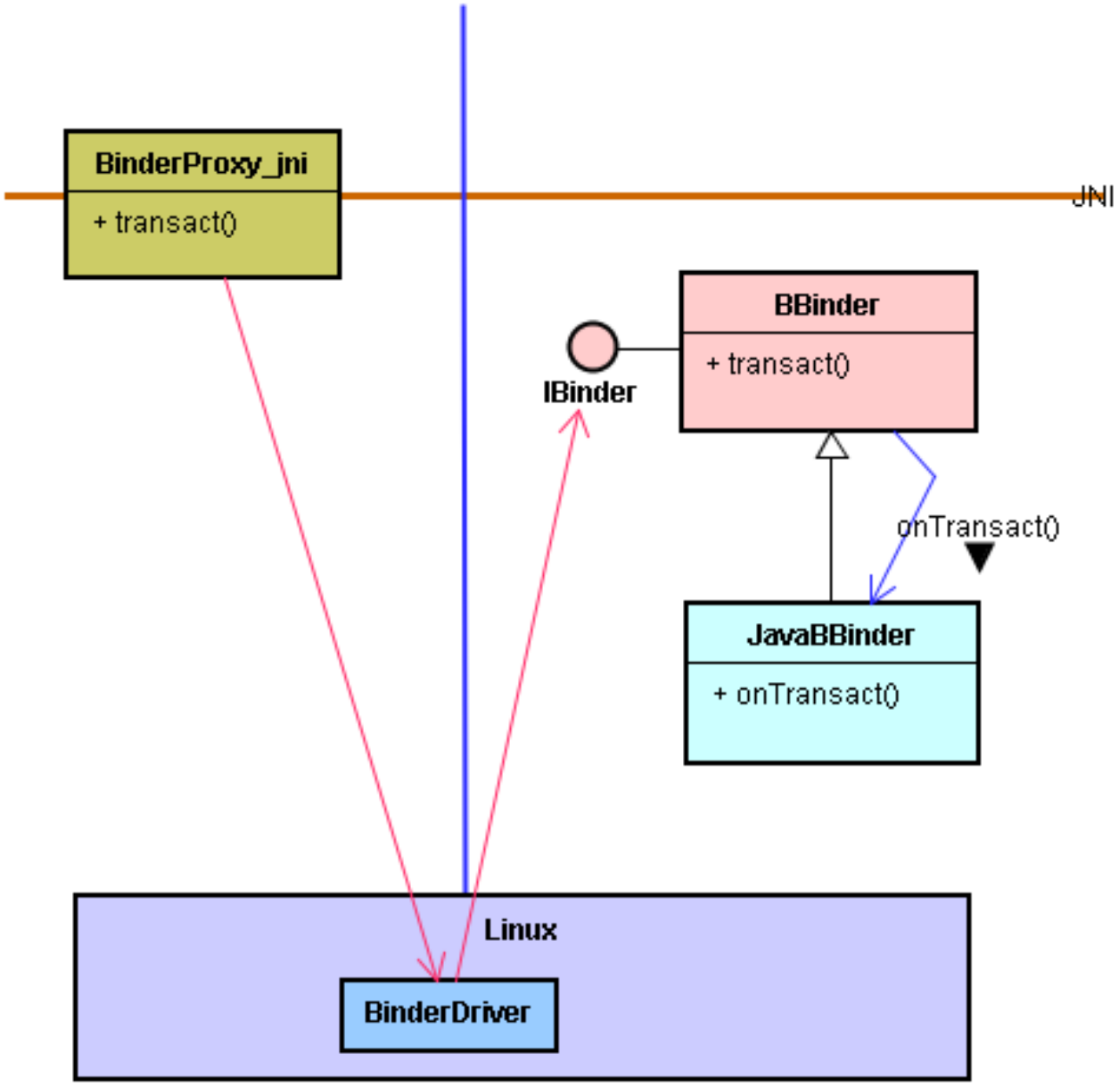

BinderProxy
- mObject : int
+ <<native>> transact()
IBinder
框架
Binder_jni
+ init()
+ javaObjectForBinder()
+ transact()
JNI

- 当Client透过IBinder接口而調用到BinderProxy的transact()函数，就調用到其 JNI本地模块的transact()函数，就能进而衔接到底层Binder Driver驱动服务了。

例如：


Activity
myActivity
transact()
IBinder
BinderProxy
+ transact()
BinderProxy_jni
+ transact()
transact()
Binder
+ Binder()
+ execTransact()
IBinder
onTransact()
myBinder
+ myBinder()
+ onTransact()
execTransact()
JNI
BBinder
+ transact()
IBinder
onTransact()
JavaBBinder
+ onTransact()

- 在上图里，从JNI本地模块拉了一条红色虚线，表示这并非直接的通信途径。也就是，实际上是透过底层Binder Driver驱动才調用到BBinder的IBinder接口。如下图：


BinderProxy_jni
+ transact()
JNI
BBinder
+ transact()
IBinder
onTransact()
JavaBBinder
+ onTransact()
Linux
BinderDriver

~ Continued ~