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

B07_c

Messenger框架与IMesseger接口(c)

By 高煥堂

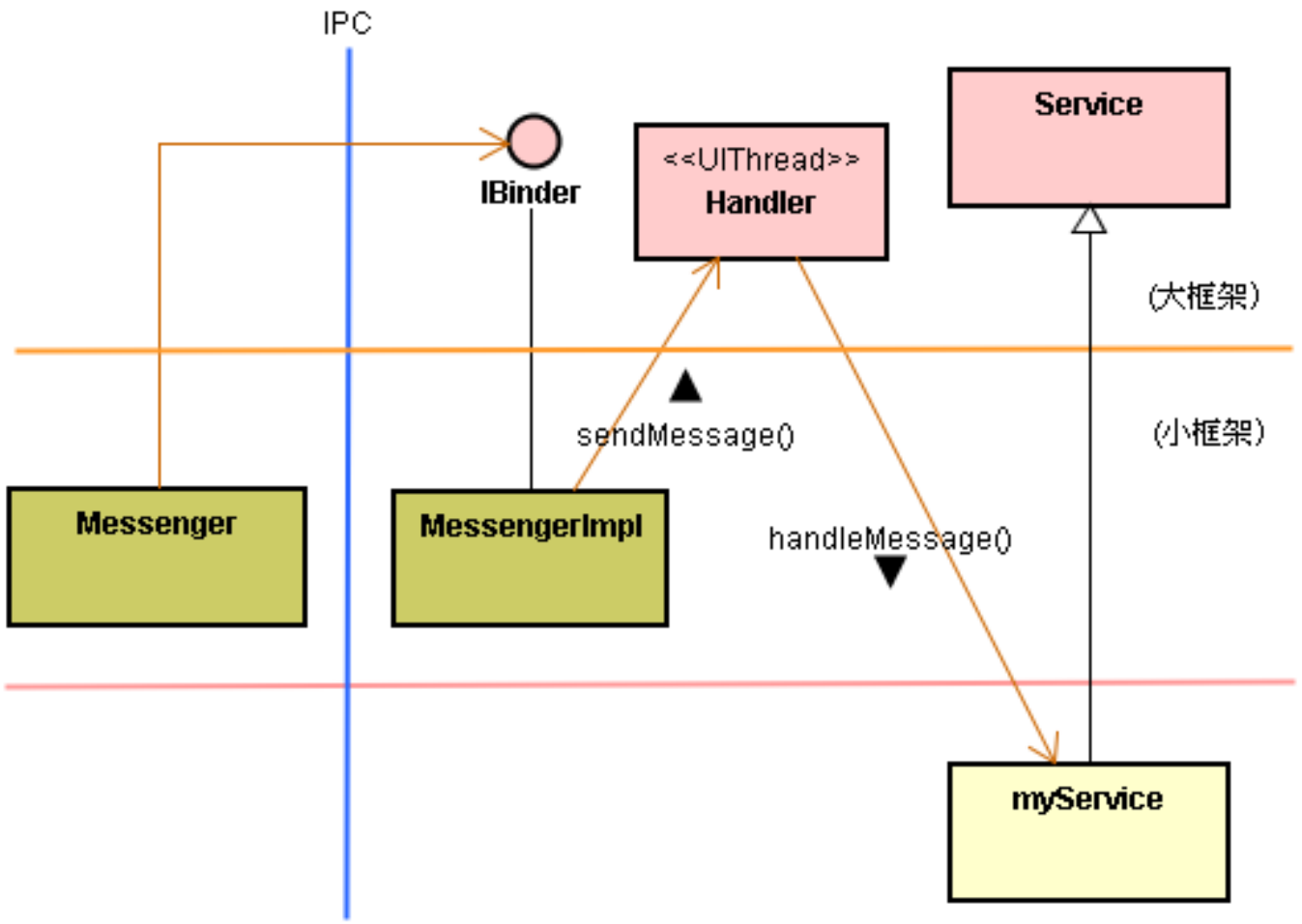
3、双向沟通的Messenger框架

- 这个Messenger框架是对Binder框架加以扩充而来的。在双向沟通上，也继承了Binder框架机制。

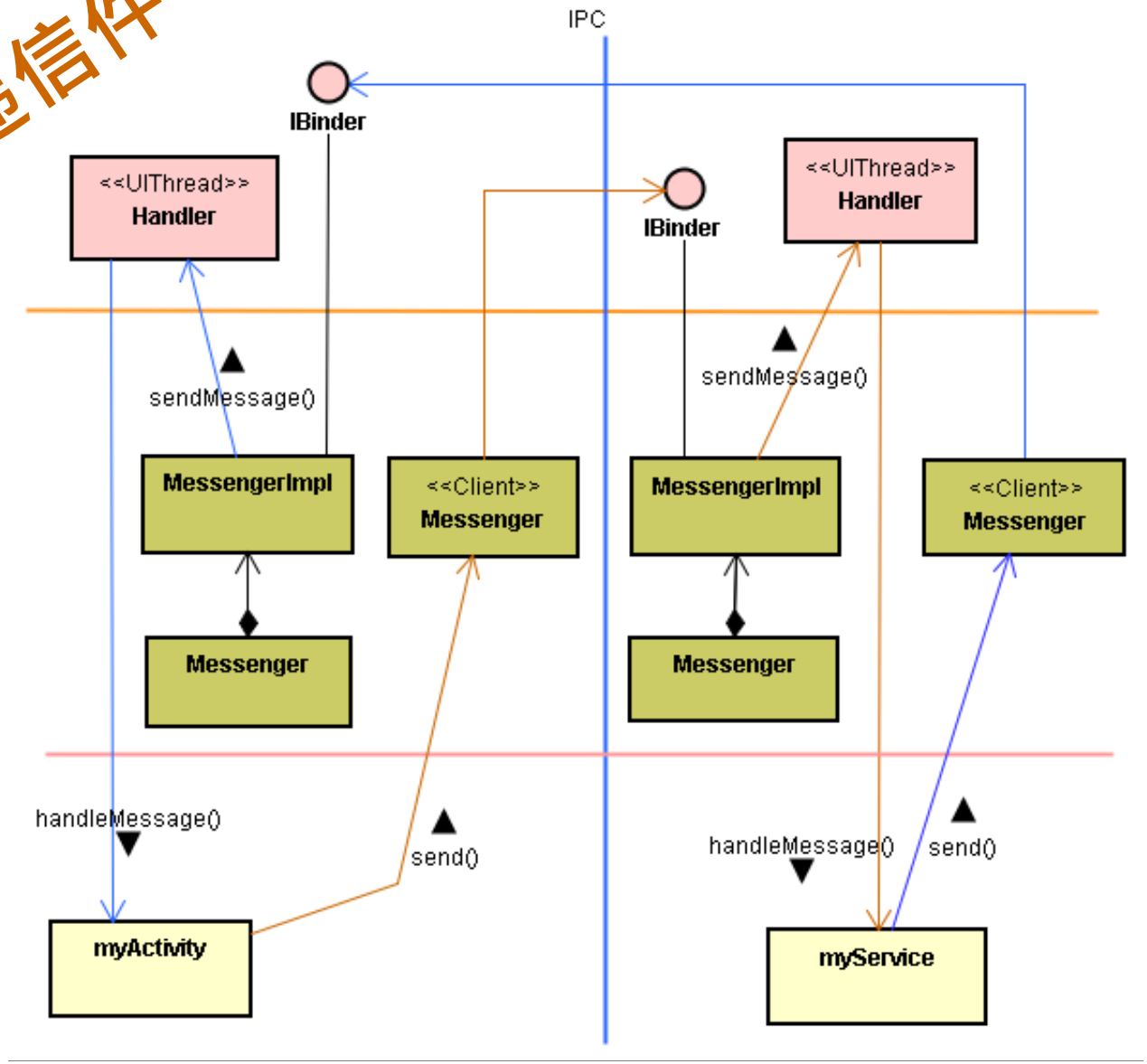
- Binder框架双向沟通的应用情境是：当myActivity透过IBinder接口调用myService的函数去执行任务时(例如使用子线程去播放mp3音乐)，万一发现底层播放系统故障了，则myService必须透过IBinder接口来通知myActivity。

- 基于上述的IBinder双向通信机制，就能用Messenger来加以包装，而为跨进程双向的Messenger框架，如下图：

单向传递信件


IPC
IBinder
<<UIThread>>
Handler
Service
(大框架)
(小框架)
sendMessage()
handleMessage()
Messenger
MessengerImpl
myService

双向传递信件


IPC
IBinder
<<UIThread>>
Handler
IBinder
<<UIThread>>
Handler
sendMessage()
sendMessage()
MessengerImpl
<<Client>>
Messenger
MessengerImpl
<<Client>>
Messenger
Messenger
Messenger
handleMessage()
send()
handleMessage()
send()
myActivity
myService

基本設計原則

- **已知：**myActivity透过Android框架去配对才取得myService对象，然后才取得myService所在进程里的IBinder接口。
- **议题：**那么，myService又如何取得myActivity进程里的IBinder接口呢?
- **答案：**myActivity先将IBinder接口打包到信件(Message对象)里，随着信送到对方，对方(myActivity)就接到IBinder接口了。

```
// myActivity.java
public class myActivity extends Activity {
        @Override
         protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                 //……
                bindService(intent, connection, BIND_AUTO_CREATE);
            }
        class myHandler extends Handler {
            @Override public void handleMessage(Message msg) {
                    // ........
            }
         };
        final Messenger aMessenger
                    = new Messenger(new myHandler());
        private Messenger ibMessenger;
```

```
private ServiceConnection connection
    = new ServiceConnection() {
        public void onServiceConnected(ComponentName name,
                                IBinder ibinder) {
         ibMessenger = new Messenger(ibinder);
       }};
public void onClick(View v) {
          Message message = Message.obtain(null,
                    MessengerService.MSG_SET_VALUE);
           message.replyTo = aMessenger;
           ibMessenger.send(message);
       }
 }}
```

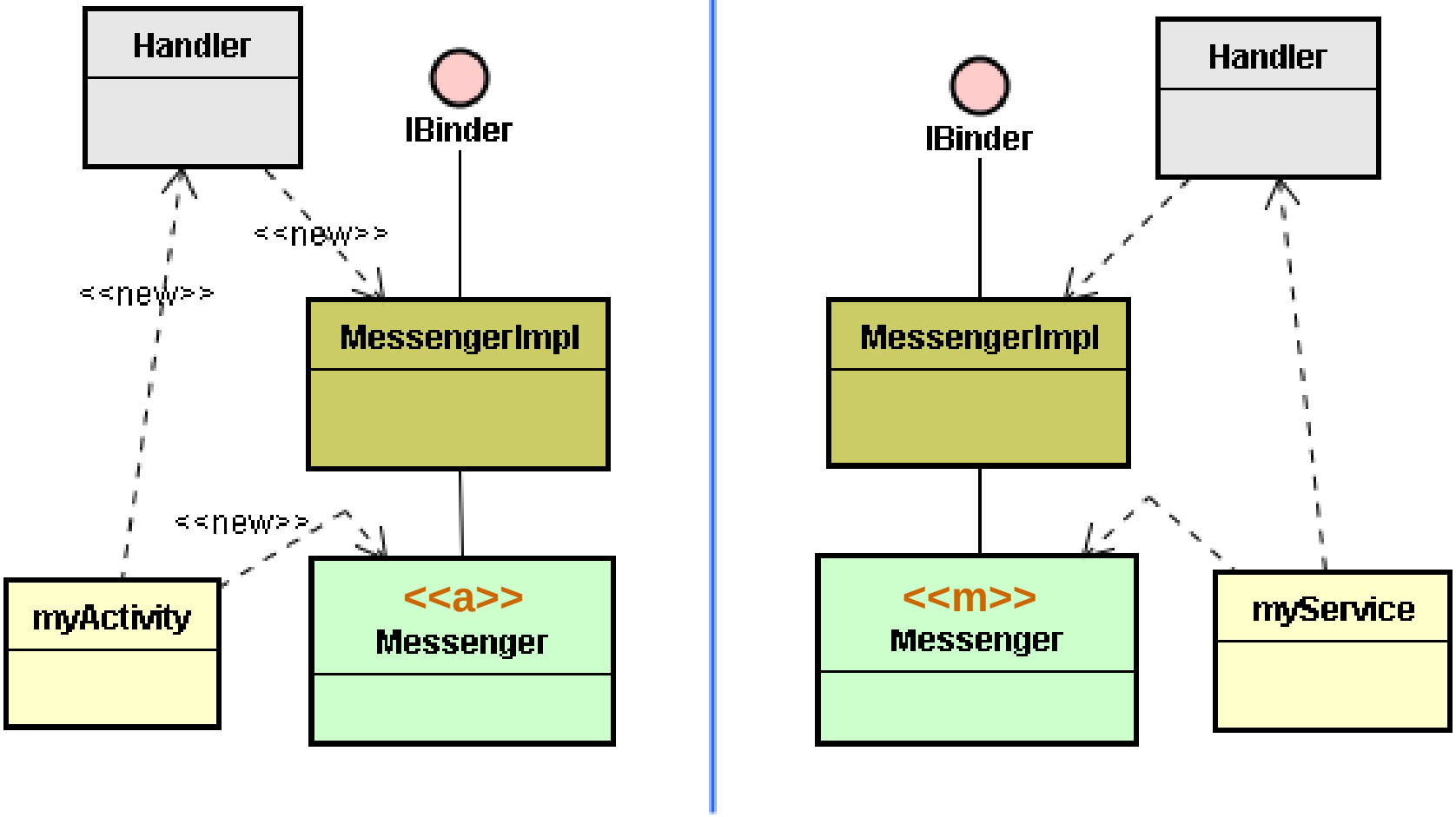
```
// myService.java
// ……
public class myService extends Service {
    private Messenger cbMessenger;

    class myHandler extends Handler {
        @Override public void handleMessage(Message msg) {
            Message message = Message.obtain(null, 0, “How are you”);
            cbMessenger = msg.replyTo;
            cbMessenger.send(message);
        }};
   final Messenger mMessenger = new Messenger(new myHandler());

   @Override public IBinder onBind(Intent intent) {
            return mMessenger.getBinder();
        }
}
```

myActivity的代码：

```
final Messenger aMessenger
                = new Messenger(new myHandler());
```

myService的代码：

```
final Messenger mMessenger
                = new Messenger(new myHandler());
```


Handler
IBinder
<<new>>
<<new>>
MessengerImpl
<<new>>
myActivity
<<a>>
Messenger
IBinder
Handler
MessengerImpl
<<m>>
Messenger
myService

myActivity的代码：

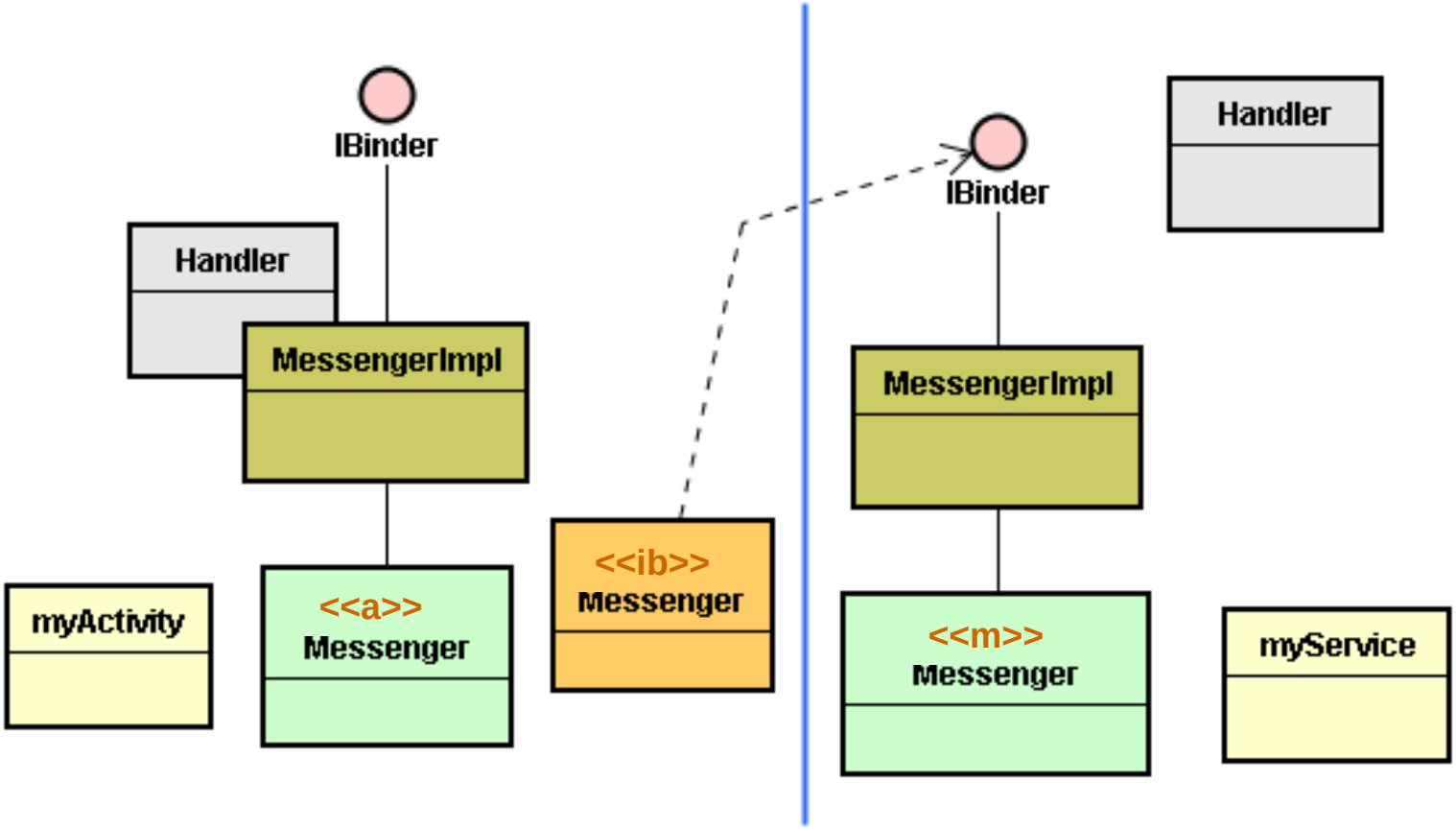
```
bindService(intent, connection,
                BIND_AUTO_CREATE);
```

myService的代码：

```
return mMessenger.getBinder();
```

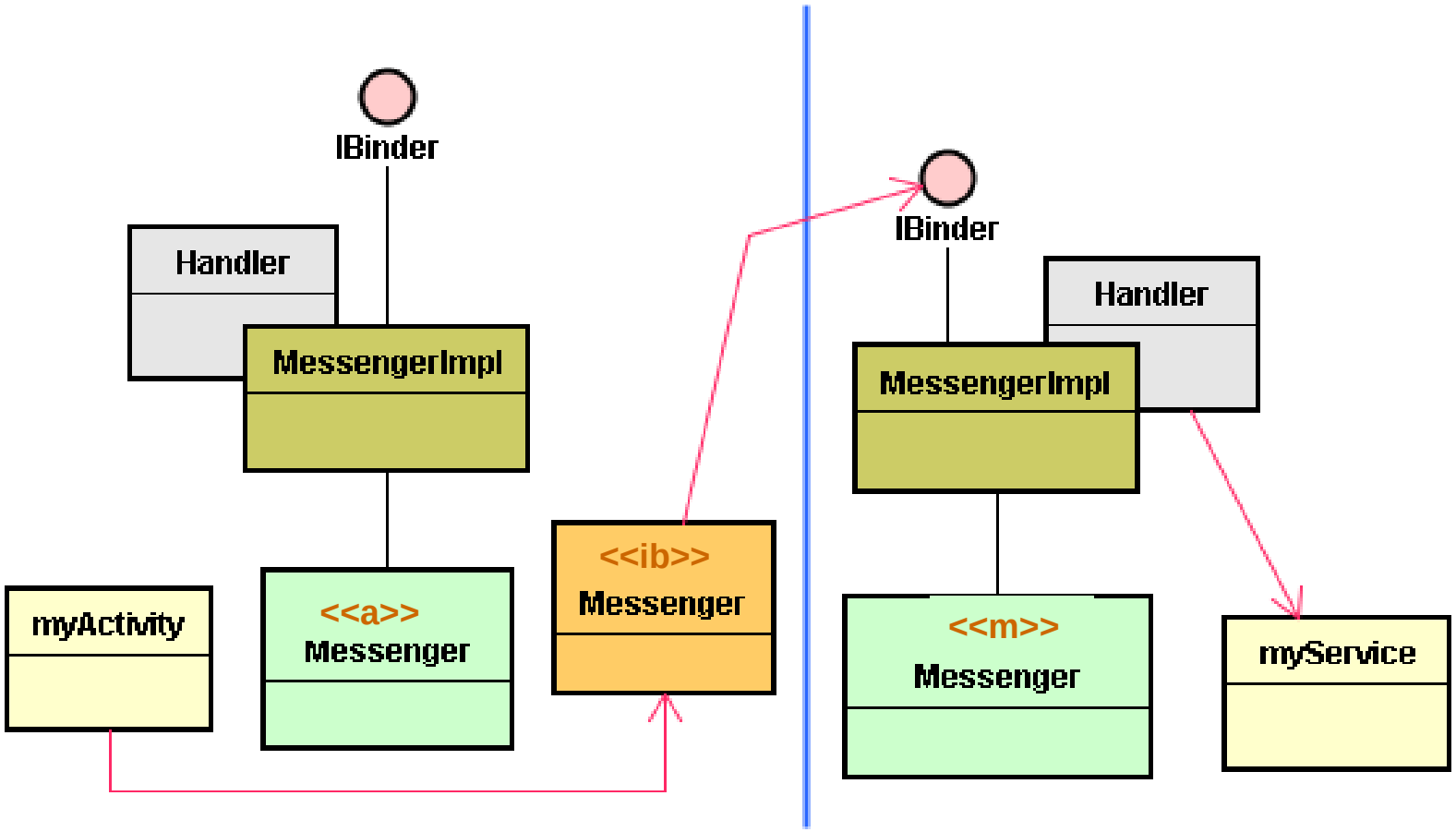
myActivity的代码：

```
public void onServiceConnected(
        ComponentName name, IBinder ibinder) {
            ibMessenger = new Messenger(ibinder);
```


IBinder
Handler
MessengerImpl
myActivity
<<a>>
Messenger
<<ib>>
Messenger
IBinder
Handler
MessengerImpl
<<m>>
Messenger
myService

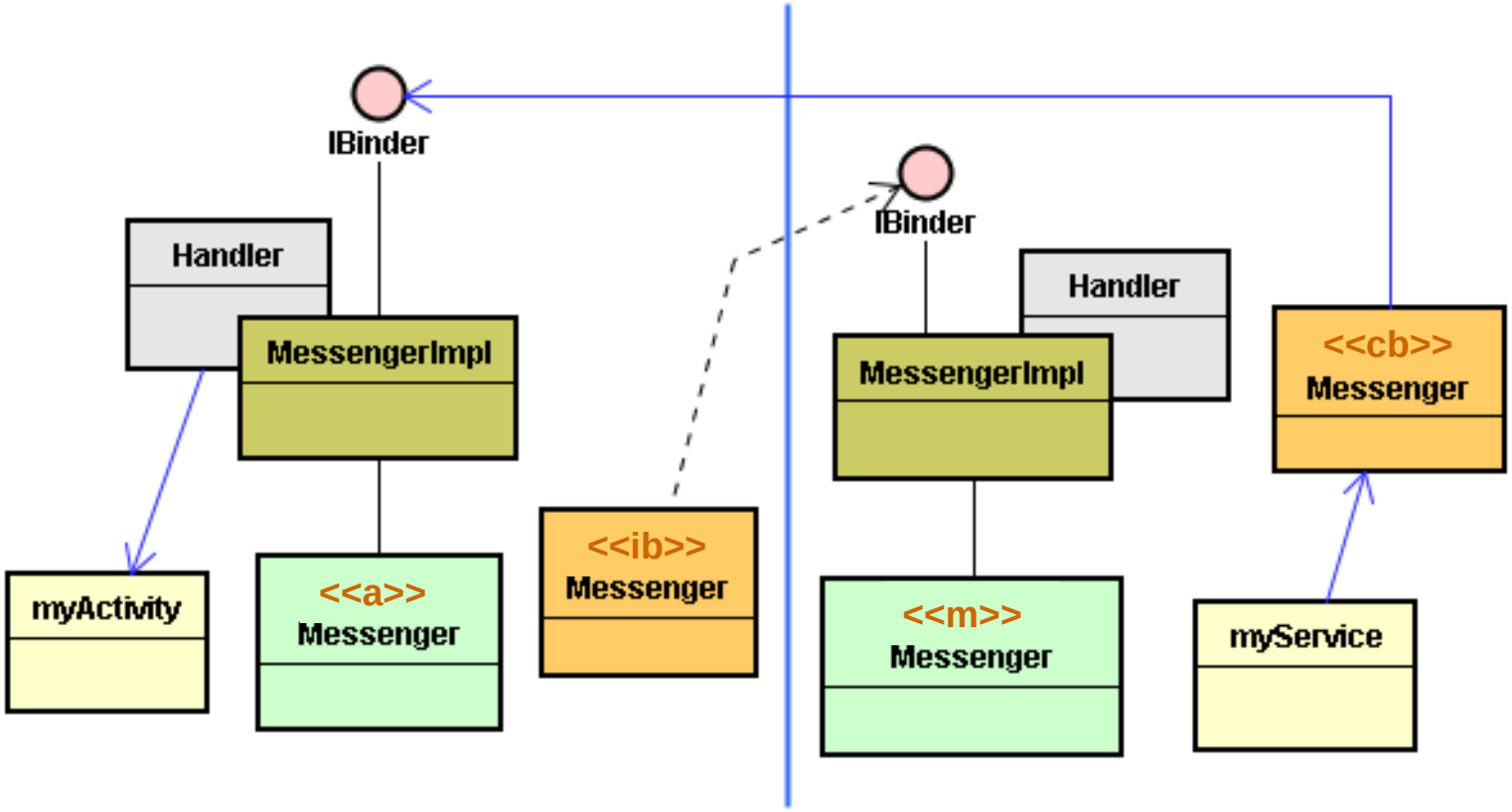
myActivity的代码：

```
message.replyTo = aMessenger;
        ibMessenger.send(message);
```


IBinder
Handler
MessengerImpl
myActivity
<<a>>
Messenger
<<ib>>
Messenger
IBinder
Handler
MessengerImpl
<<m>>
Messenger
myService

myService的代码：

```
@Override public void handleMessage(Message msg) {
        Message message
                = Message.obtain(null, 0, “How are you”);
        cbMessenger = msg.replyTo;
        cbMessenger.send(message);
```


IBinder
Handler
MessengerImpl
myActivity
<<a>>
Messenger
<<ib>>
Messenger
IBinder
Handler
MessengerImpl
<<m>>
Messenger
<<cb>>
Messenger
myService

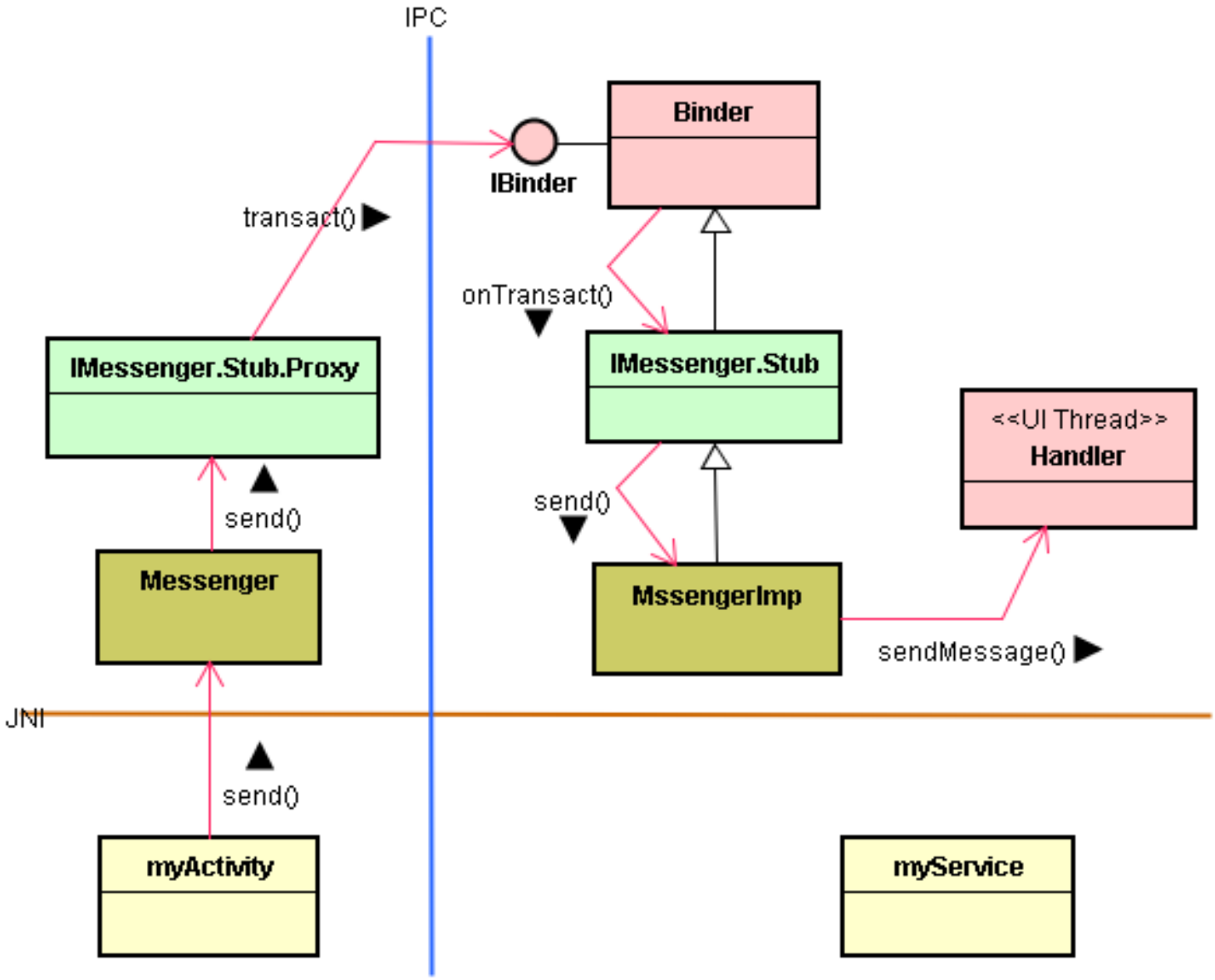
- 在myActivity调用Messenger的send()函数时，就顺便将己方的IBinder接口当作参数传递过去给myService。

- myService接到传递过来的IBinder接口时，就诞生一个新Messenger对象，并将该IBinder接口存进去。myService就能调用该新Messenger对象的send()函数，把Message对象传递到myActivity端了。

6、IMessenger接口

使用AIDL

- 在Messenger框架里还定义了IMessenger接口，让应用程序(App)可直接调用IMessenger接口的send()函数。如下图：


IPC
Binder
IBinder
transact()
onTransact()
IMessenger.Stub.Proxy
IMessenger.Stub
<<UI Thread>>
Handler
send()
send()
Messenger
MssengerImp
sendMessage()
JNI
send()
myActivity
myService

- 这是典型的Proxy-Stub模式来包装IBinder接口。
- 在myActivity进程里：Messenger类可以将IBinder接口转换成为IMessenger接口。
- 在myService进程里：也可以透过Messenger取得MessengerImpl类的IMessenger接口。

Thanks...

高煥堂