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

D03_a

Native核心服务的Proxy-Stub设计模式(a)

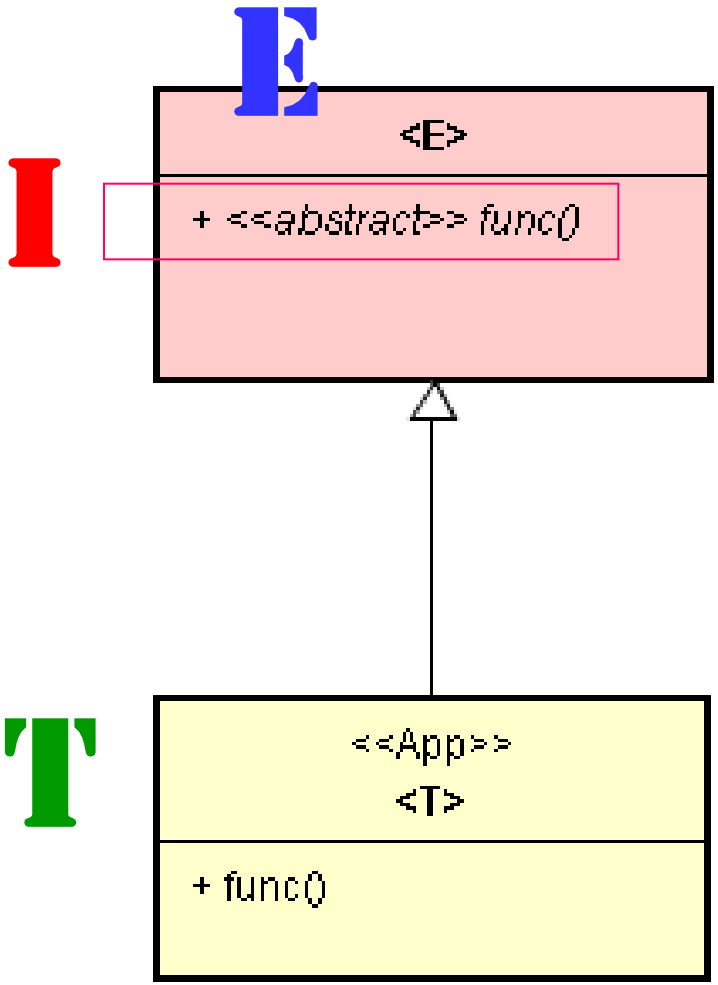
By 高煥堂

内容

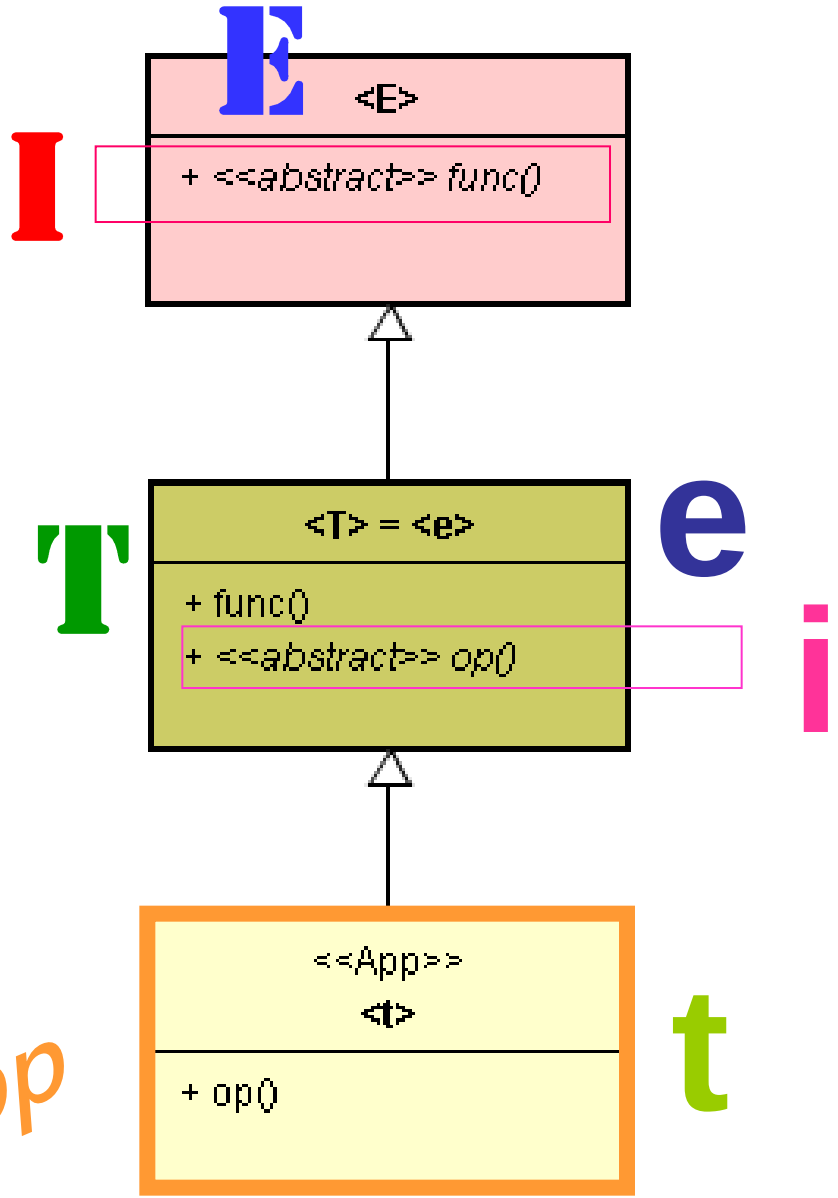
1. 复习：Java层的Proxy-Stub设计模式
2. 从IBinder接口说起
3. 使用模板，产生Stub类
4. 举例：以既有的Native Service为例
5. 使用模板，产生Proxy类
6. 让Client使用Proxy类的新接口

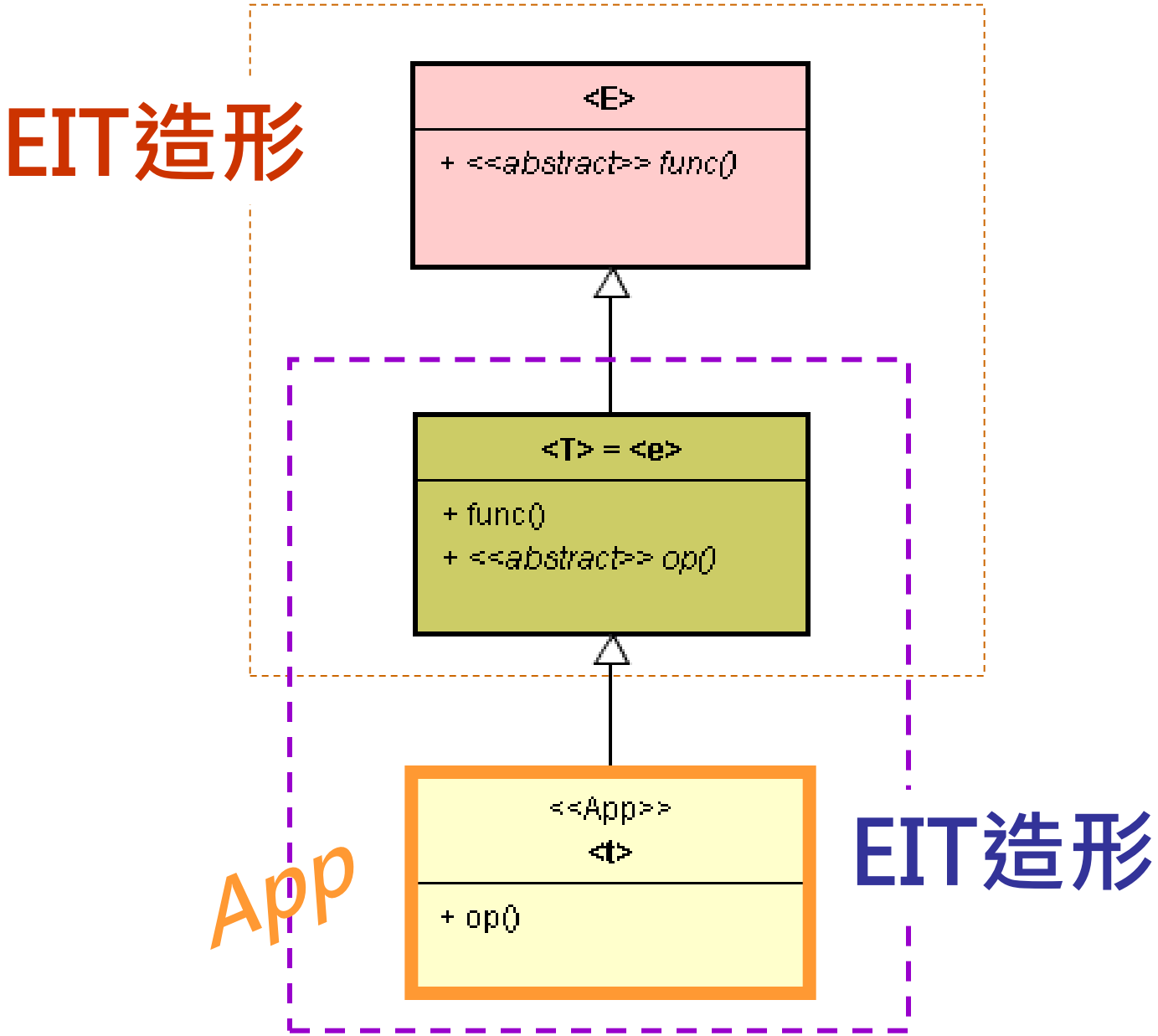
1、复习：Java层的Proxy-Stub设计模式

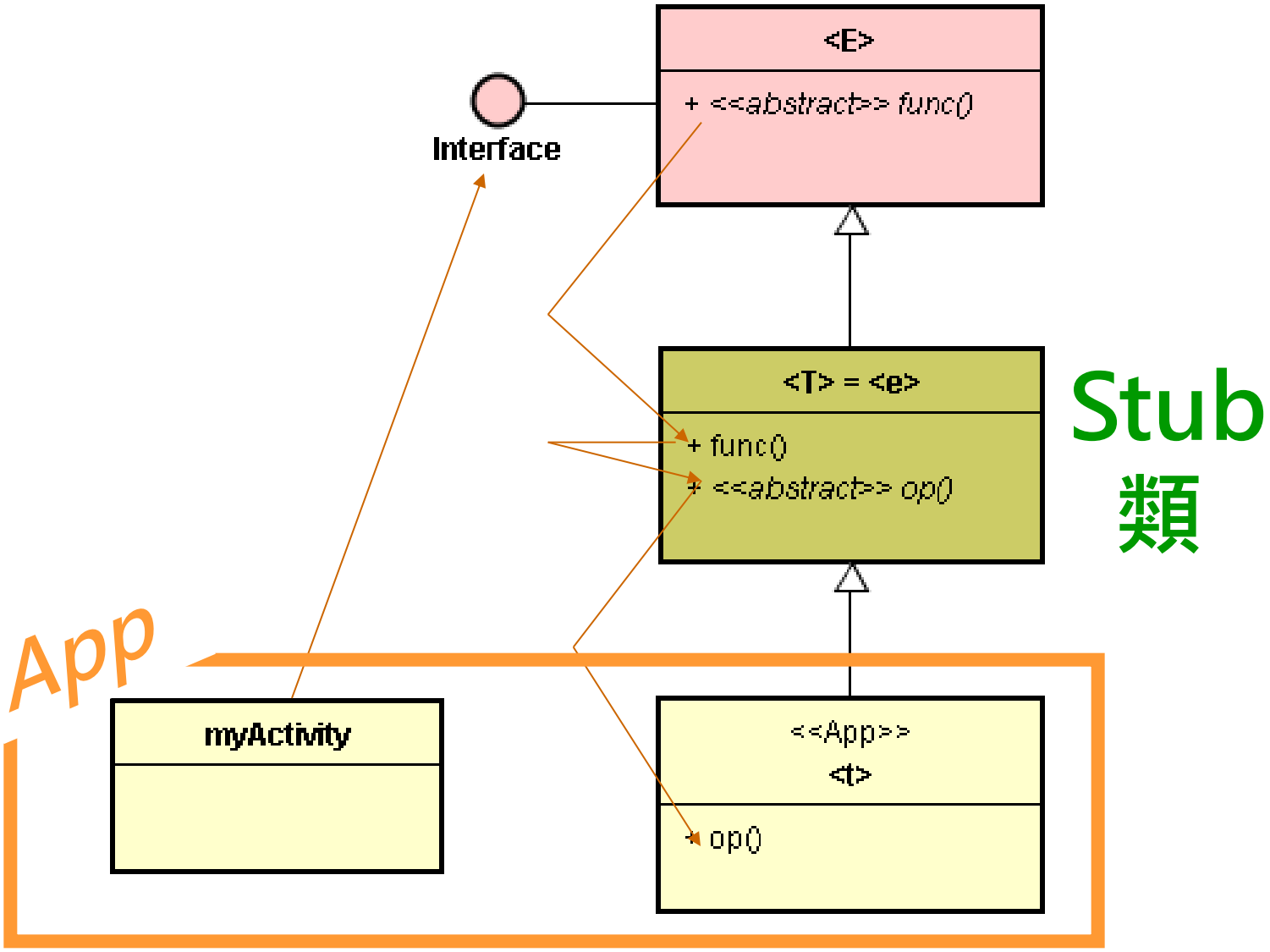
为什么要Stub类呢?


E
<E>
I
+ <<abstract>> func()
T
<<App>>
<T>
+ func()

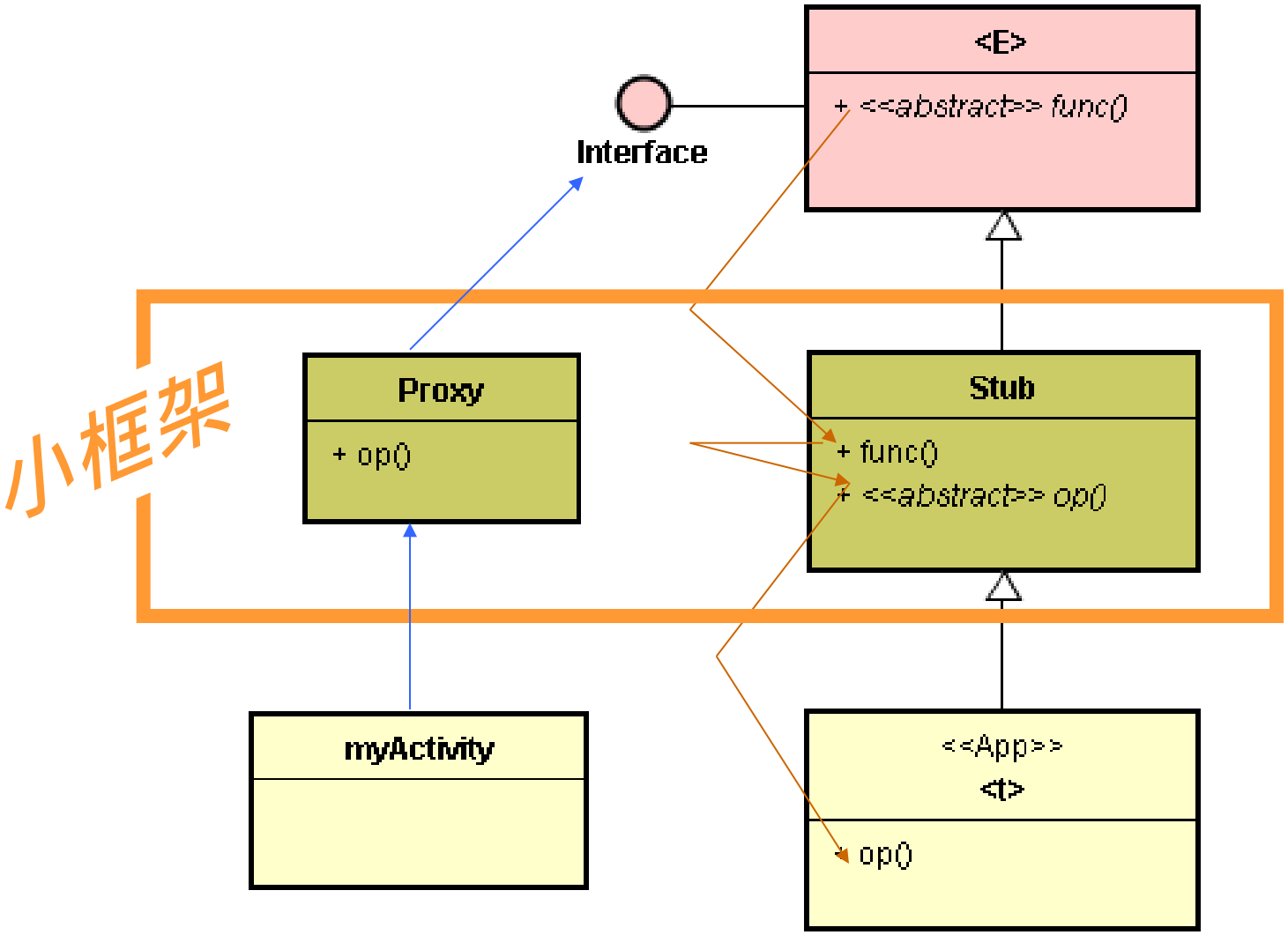
兩層EIT



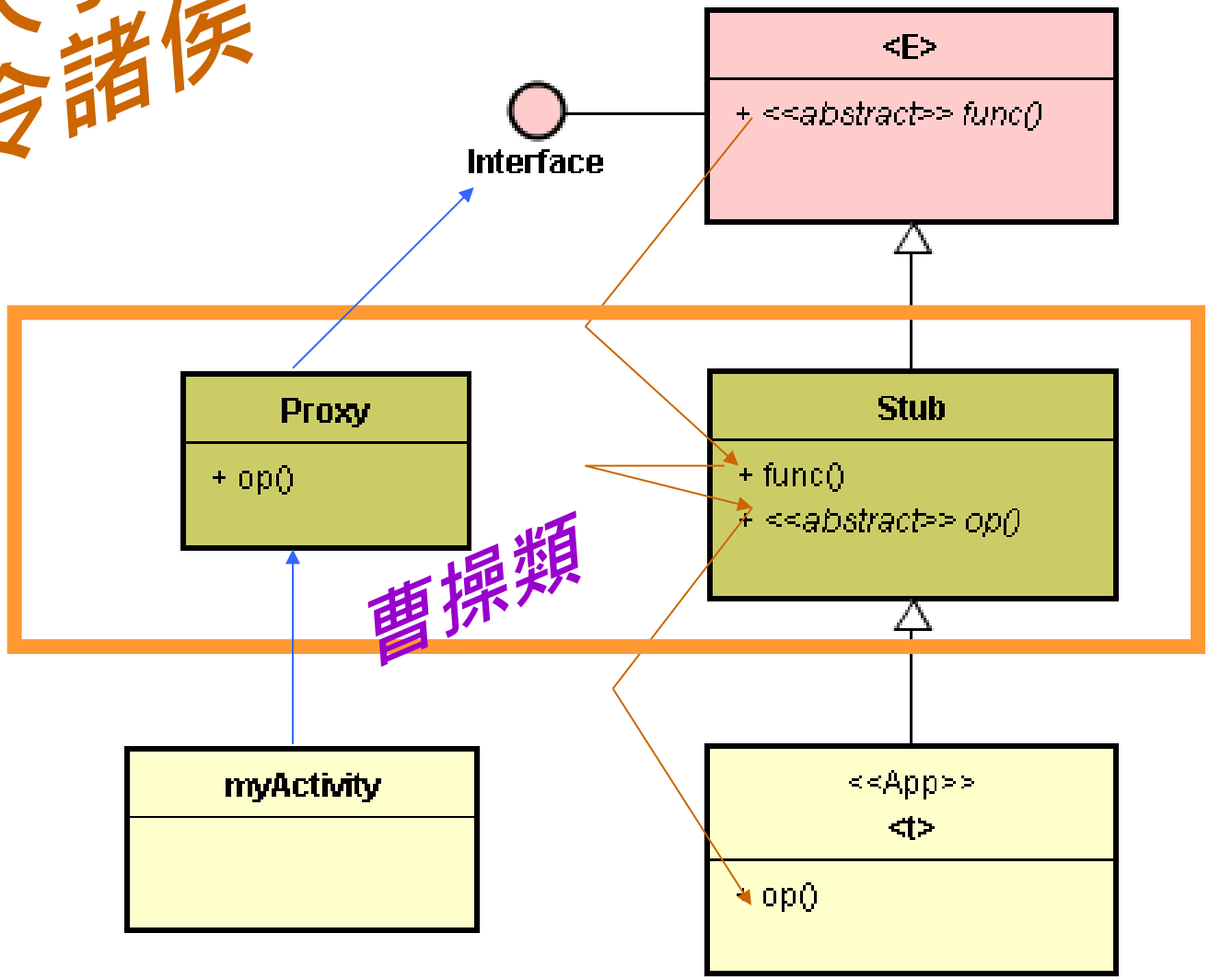

EIT造形
<E>
+ <<abstract>> func()
<T> = <e>
+ func()
+ <<abstract>> op()
<<App>>
<t>
+ op()
App
EIT造形


<E>
+ <<abstract>> func()
Interface
<T> = <e>
+ func()
+ <<abstract>> op()
Stub
類
App
myActivity
<<App>>
<t>
op()

为什么要proxy-stub模式呢?

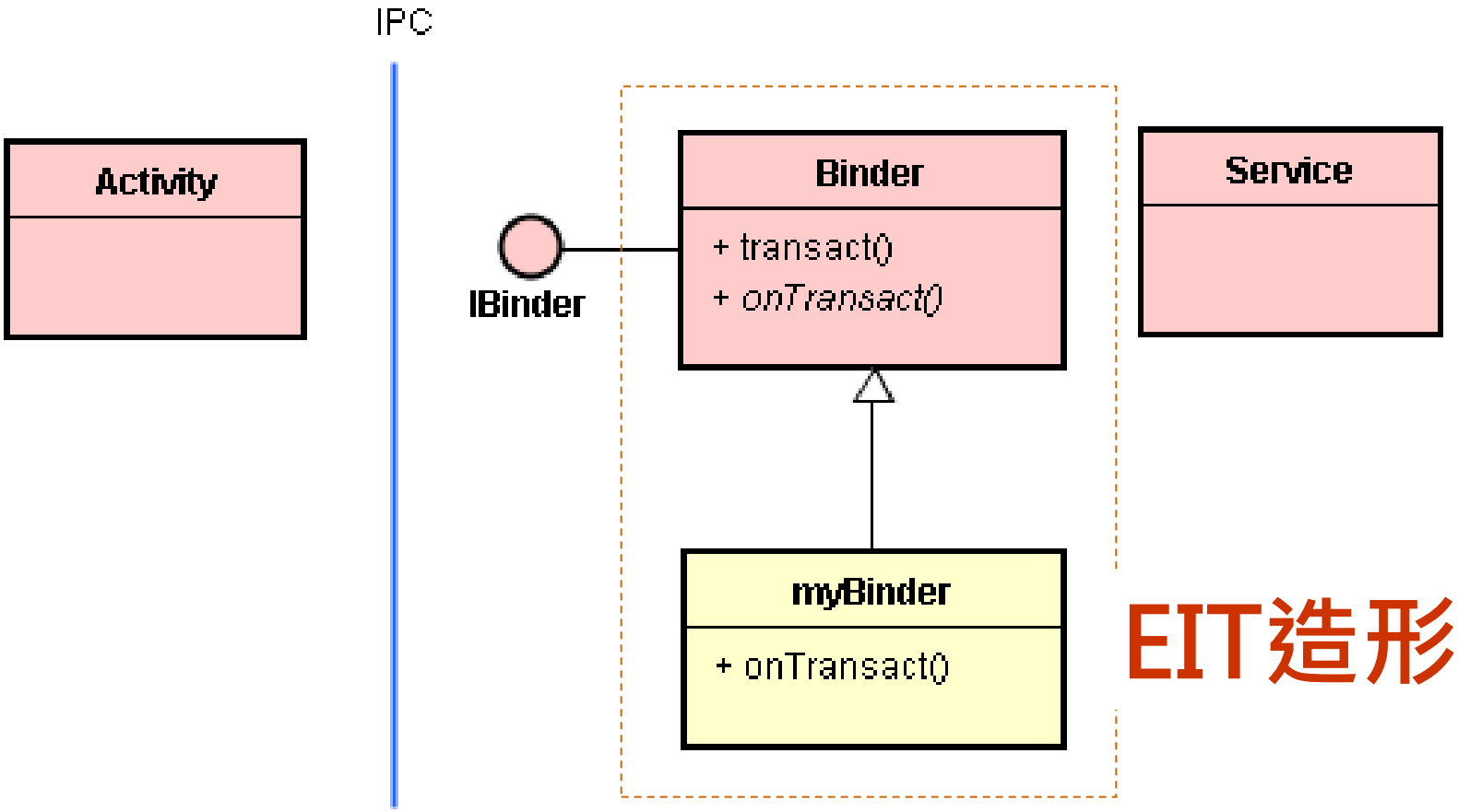

小框架
<E>
+ <<abstract>> func()
Interface
Proxy
+ op()
Stub
+ func()
+ <<abstract>> op()
myActivity
<<App>>
<t>
op()

挾天子
以令諸侯


<E>
+ <<abstract>> func()
Interface
Proxy
+ op()
Stub
+ func()
+ <<abstract>> op()
曹操類
myActivity
<<App>>
<t>
op()

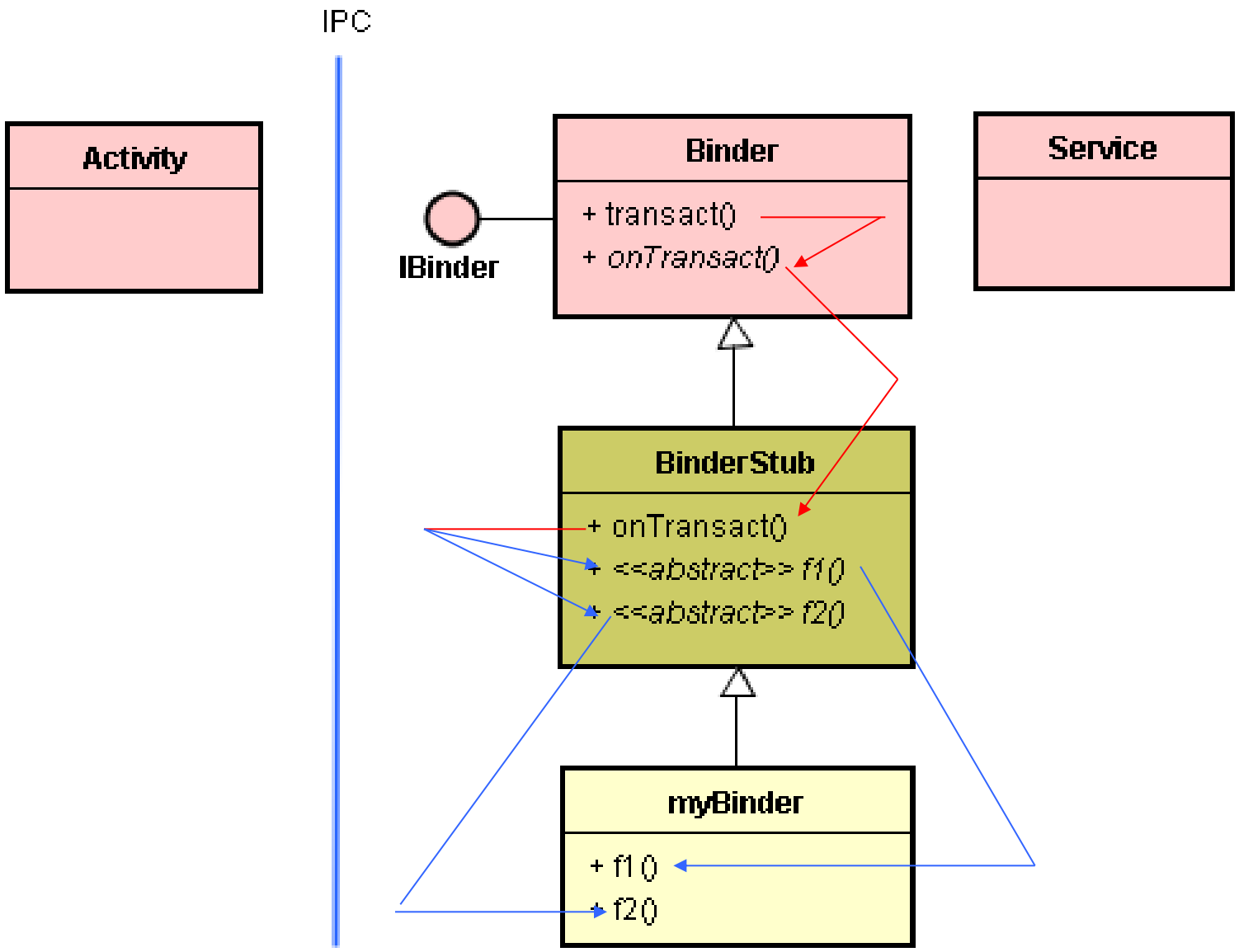
proxy-stub模式：Binder应用范例

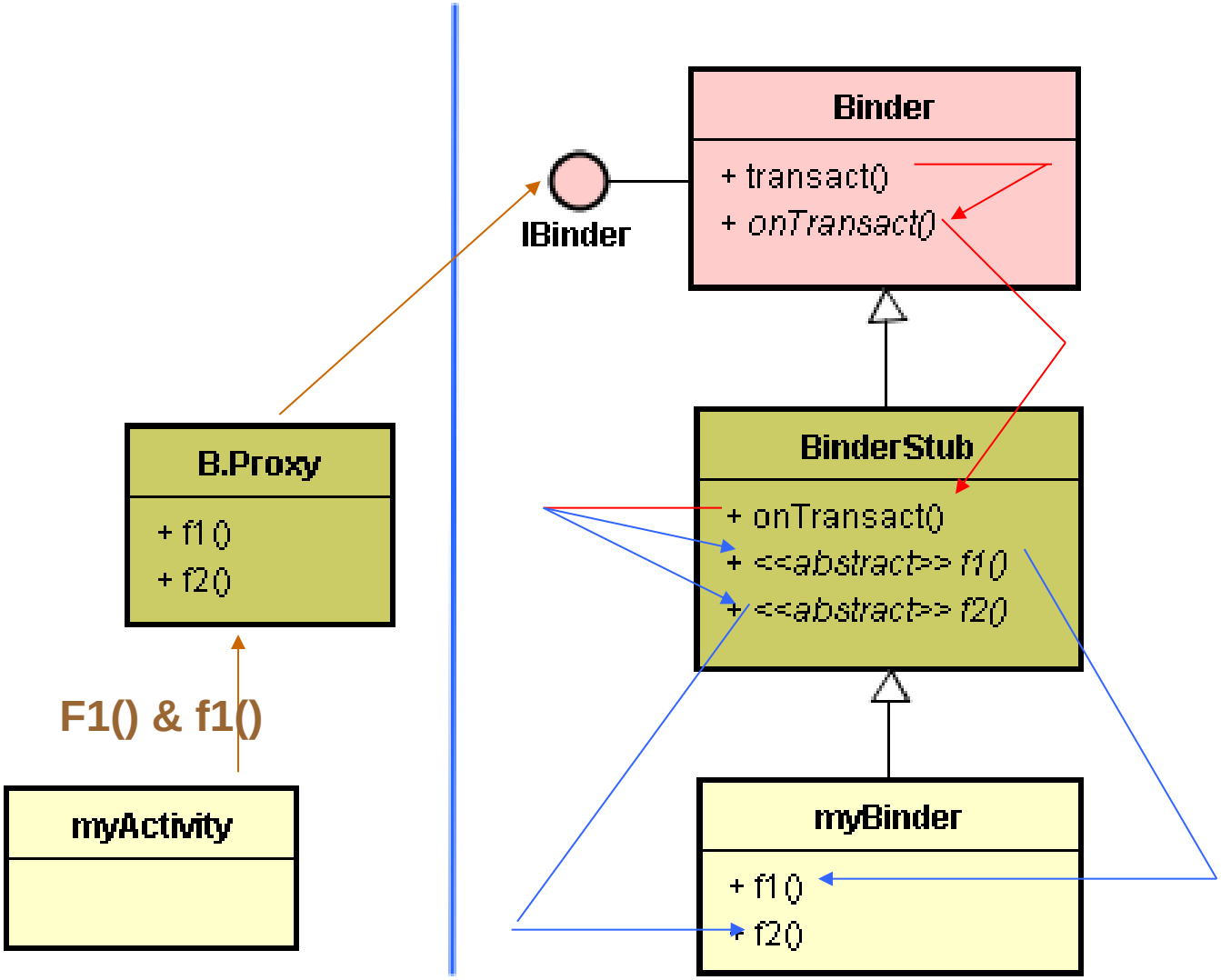
Android的典型架构



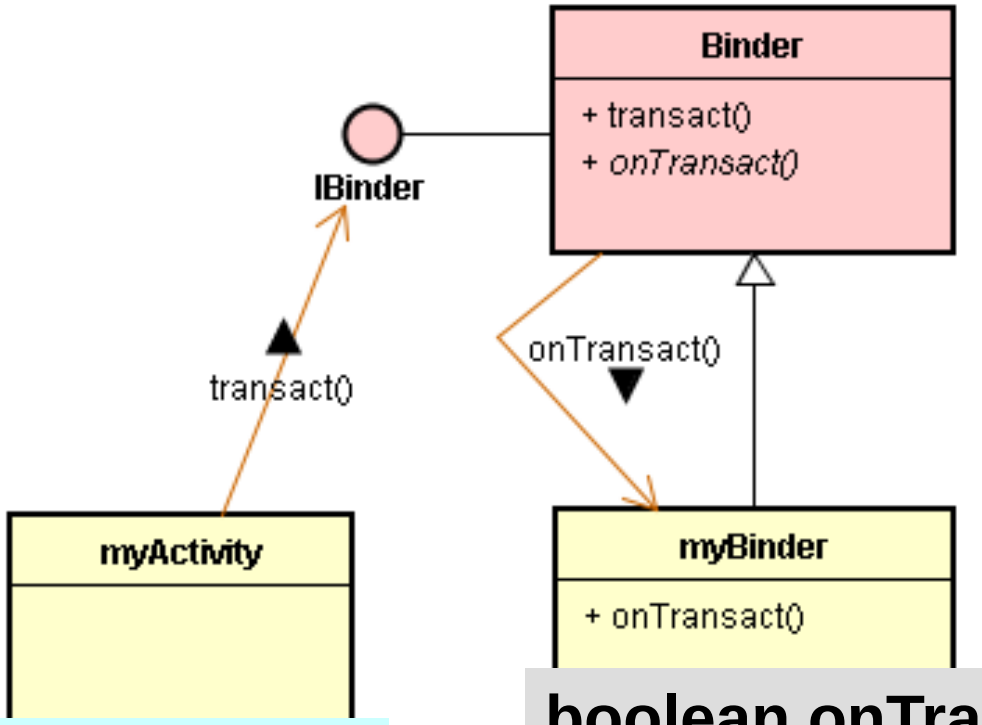
EIT造形

- 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。
- 运用曹操(Stub)类，形成两层EIT(两层框架)。


IPC
Activity
Binder
+ transact()
+ onTransact()
IBinder
Service
BinderStub
+ onTransact()
+ <<abstract>> f1()
+ <<abstract>> f2()
myBinder
+ f1()
+ f2()


Binder
+ transact()
+ onTransact()
IBinder
B.Proxy
+ f1()
+ f2()
BinderStub
+ onTransact()
+ <<abstract>> f1()
+ <<abstract>> f2()
F1() & f1()
myActivity
myBinder
+ f1()
+ f2()

Binder应用范例：<编码>与<译码>




```
ib.transact(1,
    data, reply, 0);
// .....
ib.transact(2,
    data, reply, 0);
```

```
boolean onTransact( int code,
    Parcel data, Parcel reply, int
    flags) {
        switch( code ){
     case 1:  // call Play()
     case 2:  // call Stop()
}}
```

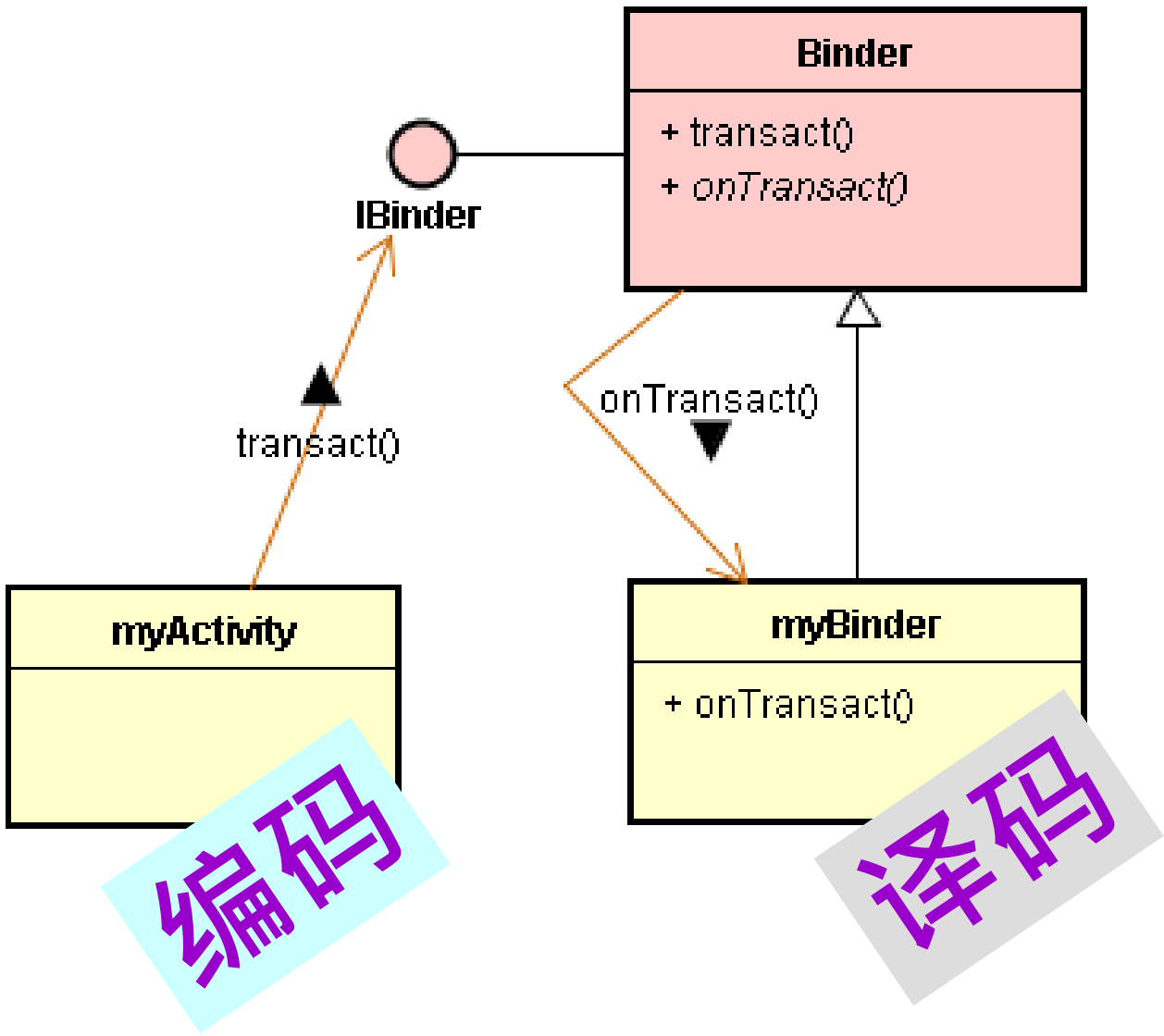
- 例如myActivity的代码：

编码

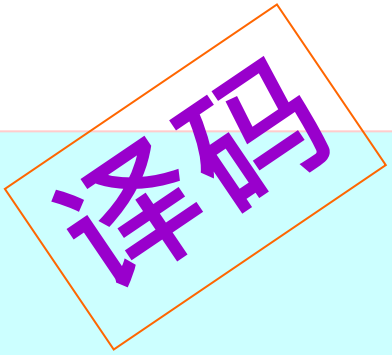
```
case 101: // Play Button
      //…..
   ib.transact(1, data, reply, 0);
case 102: // Stop Button
      // …..
   ib.transact(2, data, reply, 0);
```

- 就是对<Play>和<Stop>两个功能进行”编码”的动作。

- 编好码之后，就将这编码值当做第1个参数传给IBinder接口的transact()函数。
- 于是编码值就跨进程地传递到myBinder类里的onTransact()函数了。


Binder
+ transact()
+ onTransact()
IBinder
transact()
onTransact()
myActivity
myBinder
+ onTransact()
编码
译码

例如myBinder的代码：

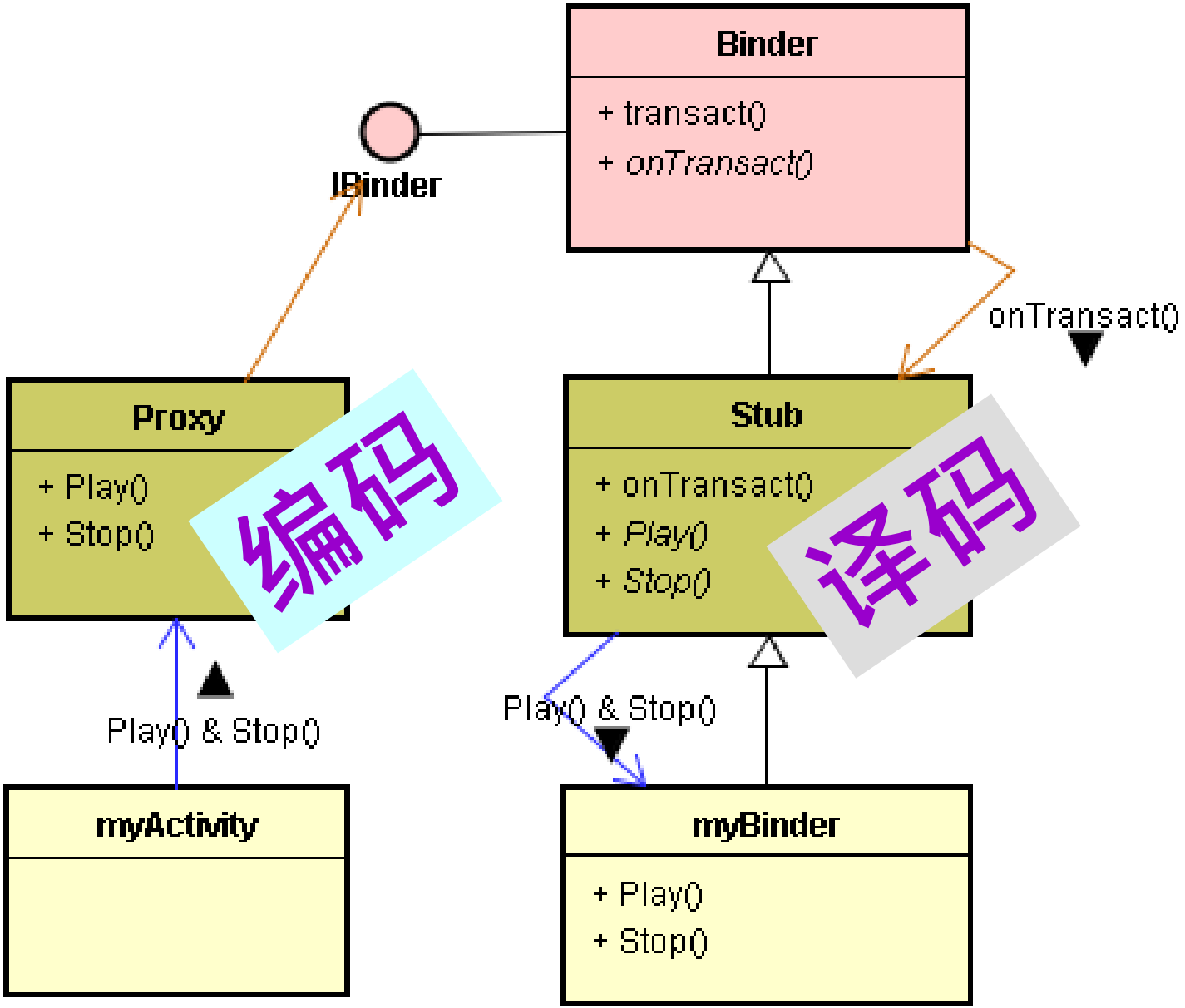



```
public class myBinder extends Binder{
  @Override  public boolean onTransact( int code, Parcel data,
            Parcel reply, int flags) throws android.os.RemoteException {
      switch( code ){
        case 1:
          // 将Message丢到子线程的MQ to play MP3
          Message msg = myService.h.obtainMessage(1,1,1, “Play”);
          myService.h.sendMessage(msg);    break;
        case 2:
          // 将Message丢到子线程的MQ to stop playing
          msg = myService.h.obtainMessage(1,1,1, “Stop”);
          myService.h.sendMessage(msg);    break;
        }
        return true;
}}
```

- 其代码就是对code进行“译码”动作。
- 如果code值為1就執行<Play>動作；如果code值為2就執行<Stop>動作。

Proxy类担任<编码>，
Stub类担任<译码>

- 回想，之前介绍过，在Java应用框架层里，使用了AIDL架构实现Proxy-Stub设计模式来封装IBinder接口，以便产生更亲切贴心的新接口。
- 并且担任<编码>和<译码>的工作。


Binder
+ transact()
+ onTransact()
IBinder
onTransact()
Proxy
+ Play()
+ Stop()
编码
Stub
+ onTransact()
+ Play()
+ Stop()
译码
Play() & Stop()
Play() & Stop()
myActivity
myBinder
+ Play()
+ Stop()

大大增加了

App开发者的负担!!

~ Continued ~