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

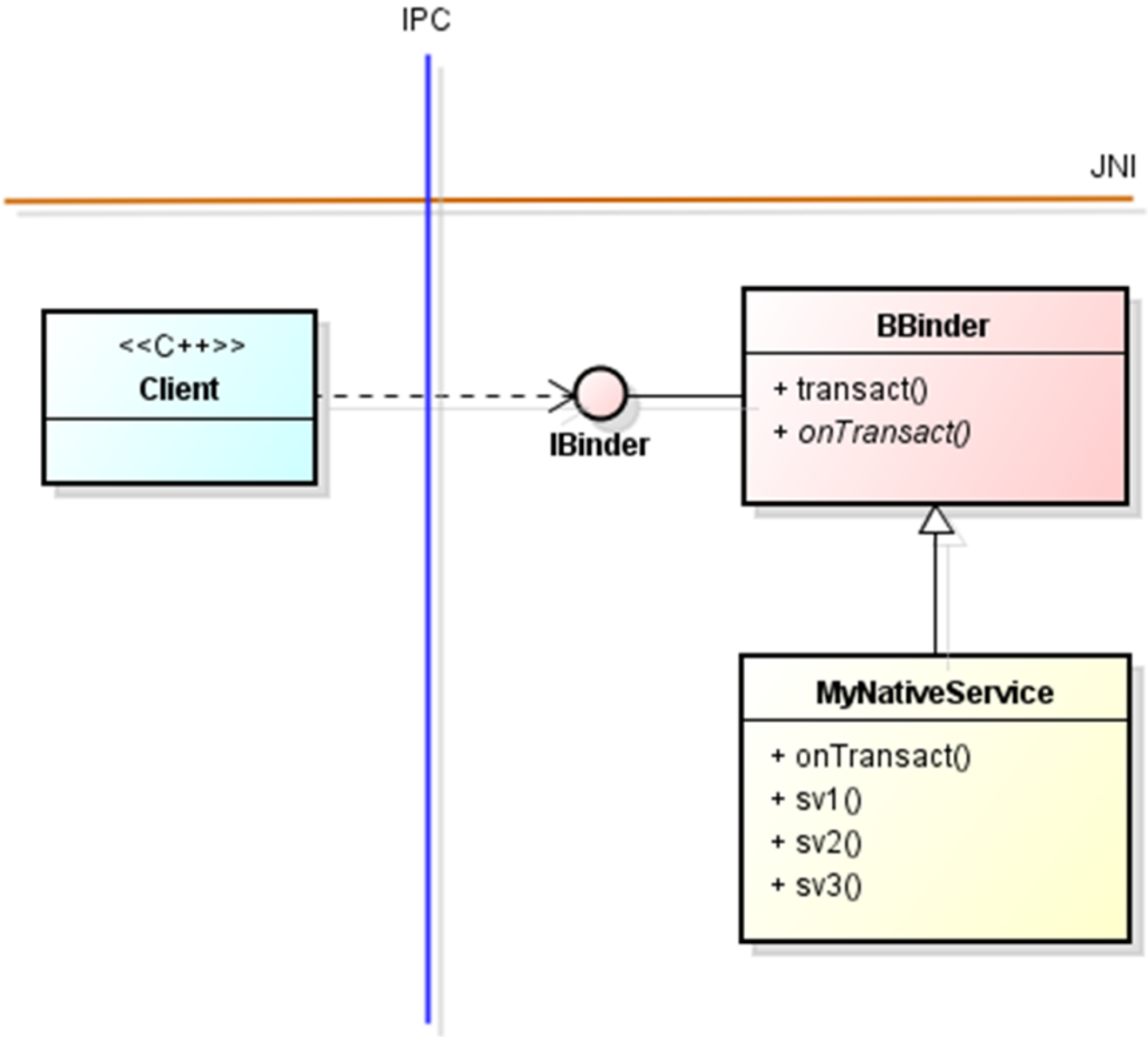
F01_c

观摩：Session模式与Proxy-Stub模式的搭配(c)

By 高煥堂

4、复习：Proxy-Stub模式

- Proxy-Stub模式的主要用途在于封装接口，以便提供更好的新接口。
- 也因而，它成为<挟天子以令诸侯>的主要架构设计模块。例如，有个Native系统服务，如下：


IPC
JNI
<<C++>>
Client
IBinder
BBinder
+ transact()
+ onTransact()
MyNativeService
+ onTransact()
+ sv1()
+ sv2()
+ sv3()

- Android提供了BpInterface<T>和BnInterface<T>两个模板，来协助创建Proxy和Stub两个类。
- 例如，BnInterface<T>模板定义如下：

```
template<typename INTERFACE>
class BnInterface :public INTERFACE, public BBinder {
   public:
     virtual sp<IInterface>  queryLocalInterface(const
                    String16& _descriptor);
     virtual String16        getInterfaceDescriptor() const;
   protected:
     virtual IBinder*        onAsBinder();
  };
```

- 基于这个模板，并定义接口如下：

```
class IMyService :public IInterface {
    public:
        DECLARE_META_INTERFACE(MyService);
        virtual void sv1(…) = 0;
        virtual void sv1(…) = 0;
        virtual void sv1(…) = 0;
};
```

- 此时可使用BnInterafce<T>模板来产生BnInterafce<IMyService>类别。如下：

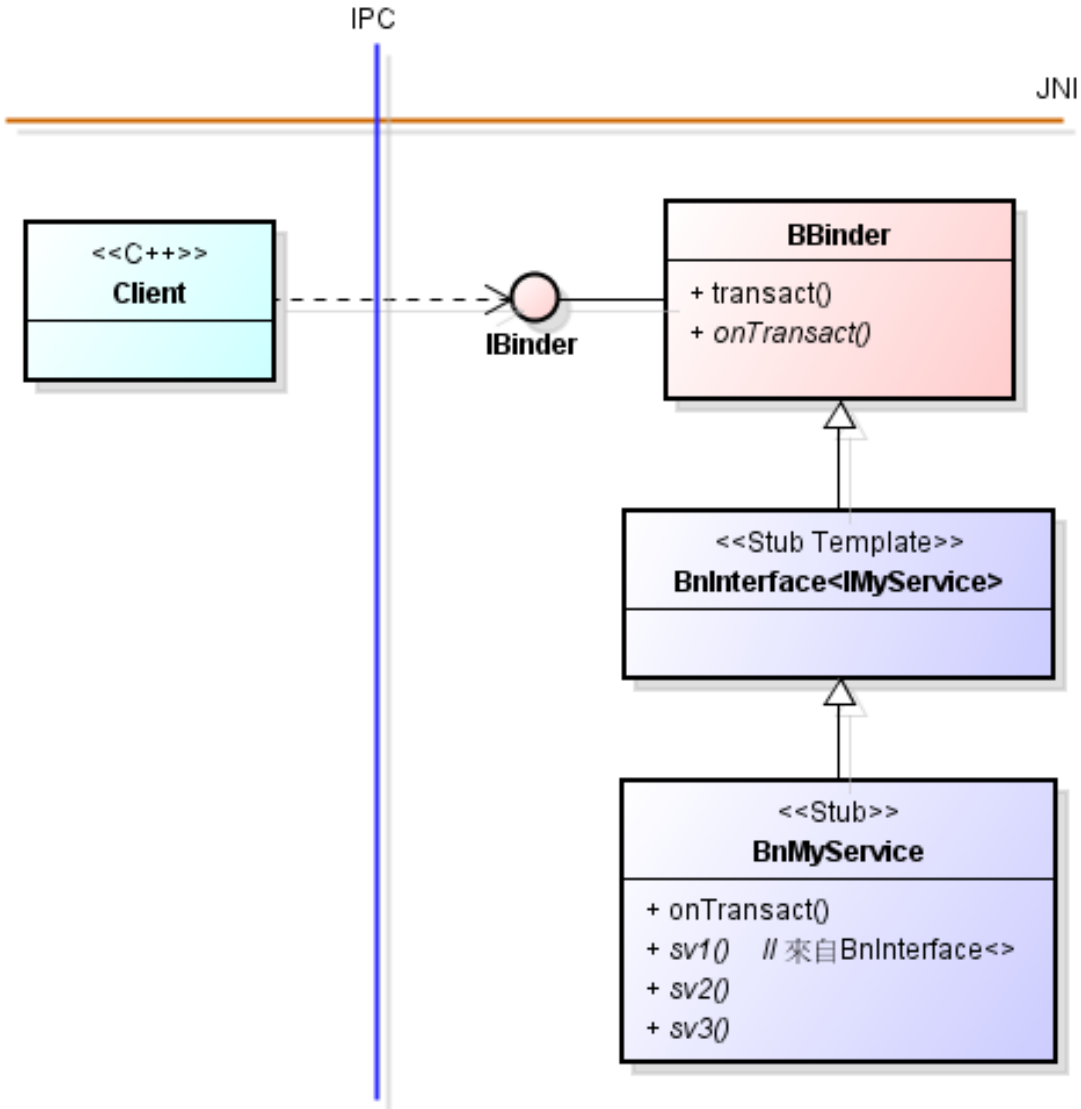
  BnInterface<IMyService>

- 它一方面继承了Binder框架基类来得到IBinder接口。同时。也继承了IMyService接口所定义的sv1(), sv2()和sv3()函数。

- 基于这个模板产生的类别，就可衍生出Stub类别，如下：

```
class BnMyService : public BnInterface<IMyService>
    {
        //.....
    }
```

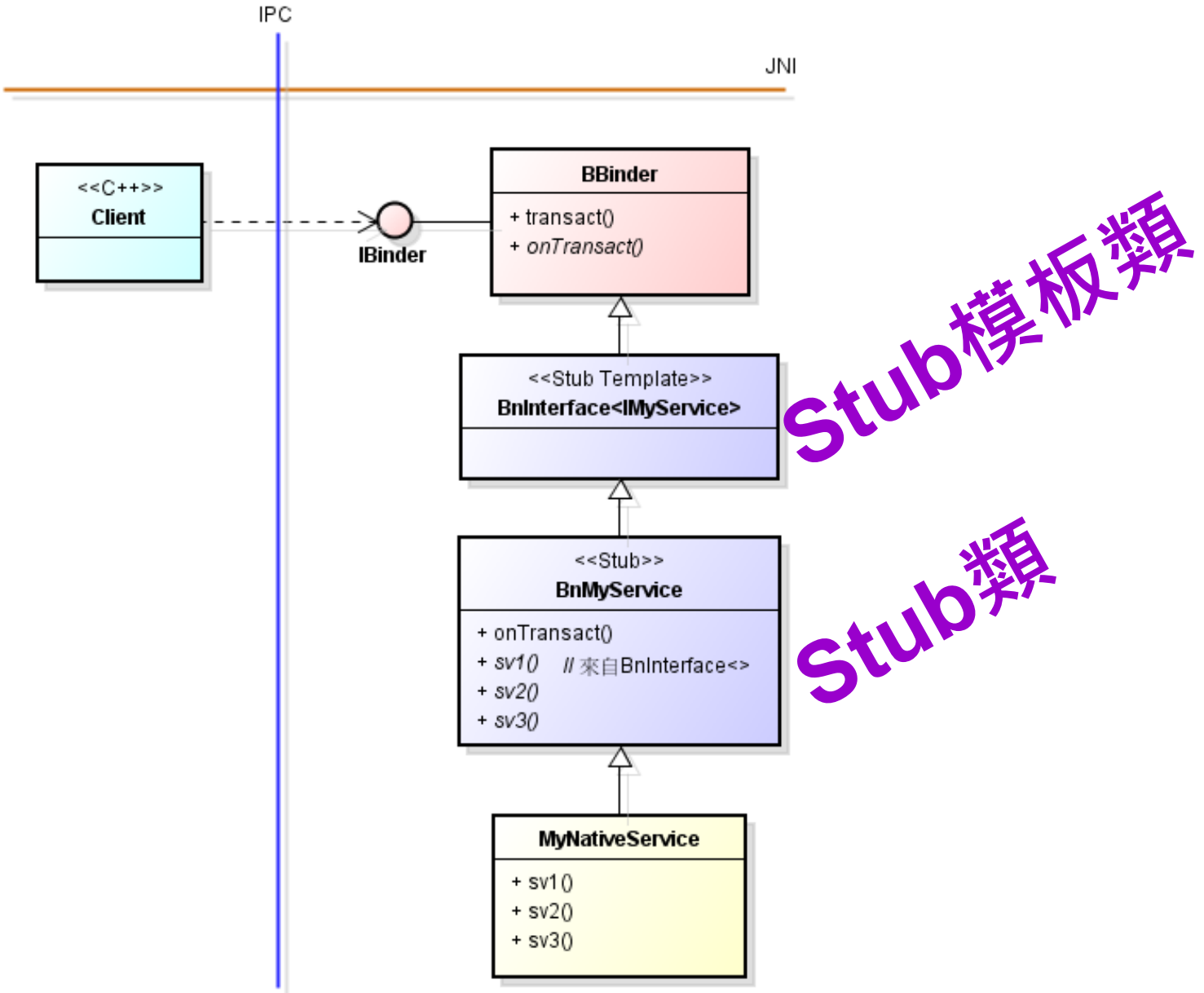
- 如下图所示：


IPC
JNI
<<C++>>
Client
IBinder
BBinder
+ transact()
+ onTransact()
<<Stub Template>>
BnInterface<IMyService>
<<Stub>>
BnMyService
+ onTransact()
+ sv1() // 來自BnInterface<>
+ sv2()
+ sv3()

- 基于这个Stub类别(即BnMyService)，我们只要撰写MyNativeService类别，它来继承上述的BnMyService类别即可，如下定义：

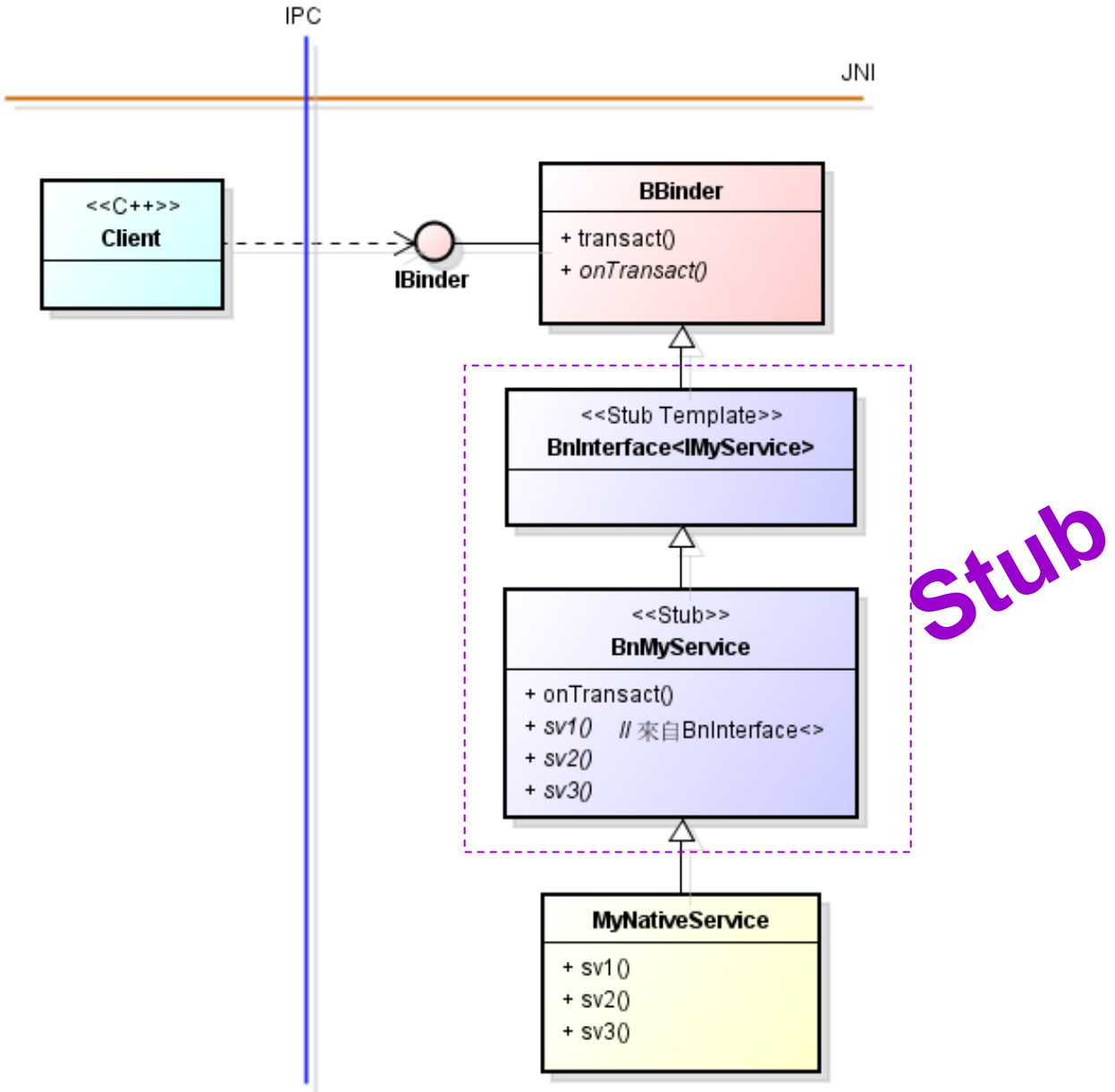
```
class MyNativeService : public BnMyService
    {
      //.....
    }
```

- 如下图所示：

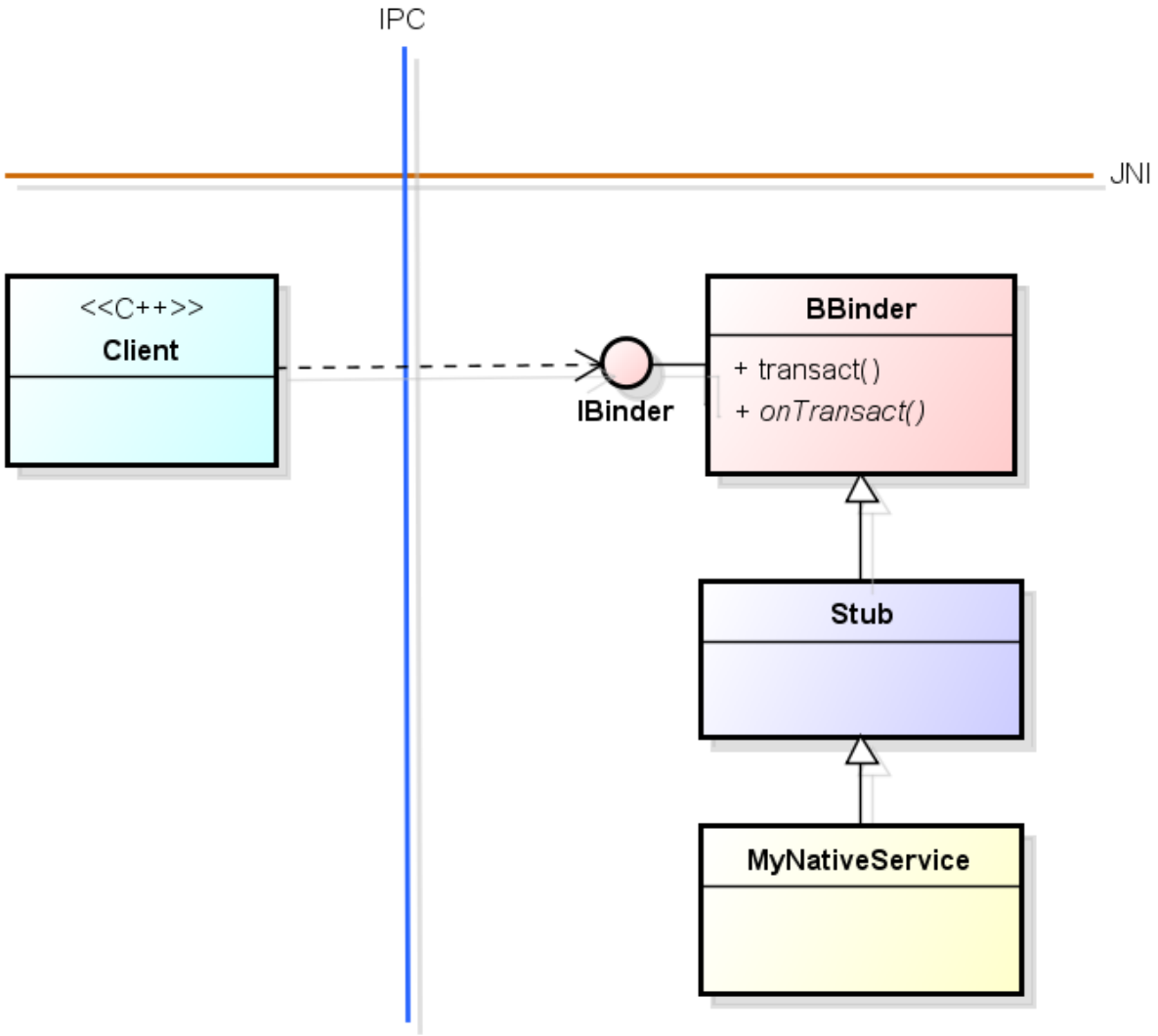

IPC
JNI
<<C++>>
Client
IBinder
BBinder
+ transact()
+ onTransact()
<<Stub Template>>
BnInterface<IMyService>
<<Stub>>
BnMyService
+ onTransact()
+ sv1() // 來自BnInterface<>
+ sv2()
+ sv3()
MyNativeService
+ sv1()
+ sv2()
+ sv3()
Stub模板類
Stub類

这两个类合起来，
扮演Stub的角色

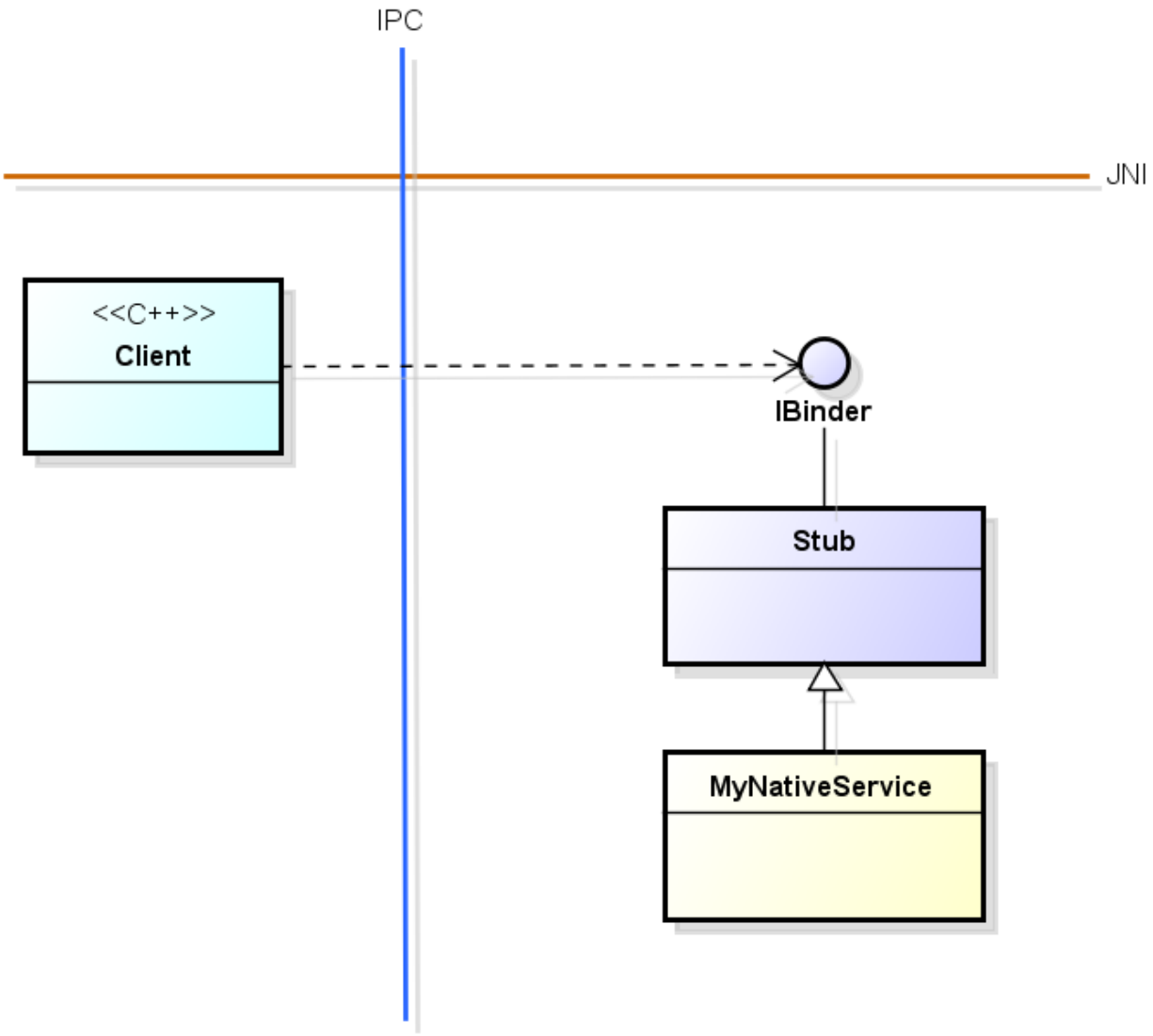
相当于

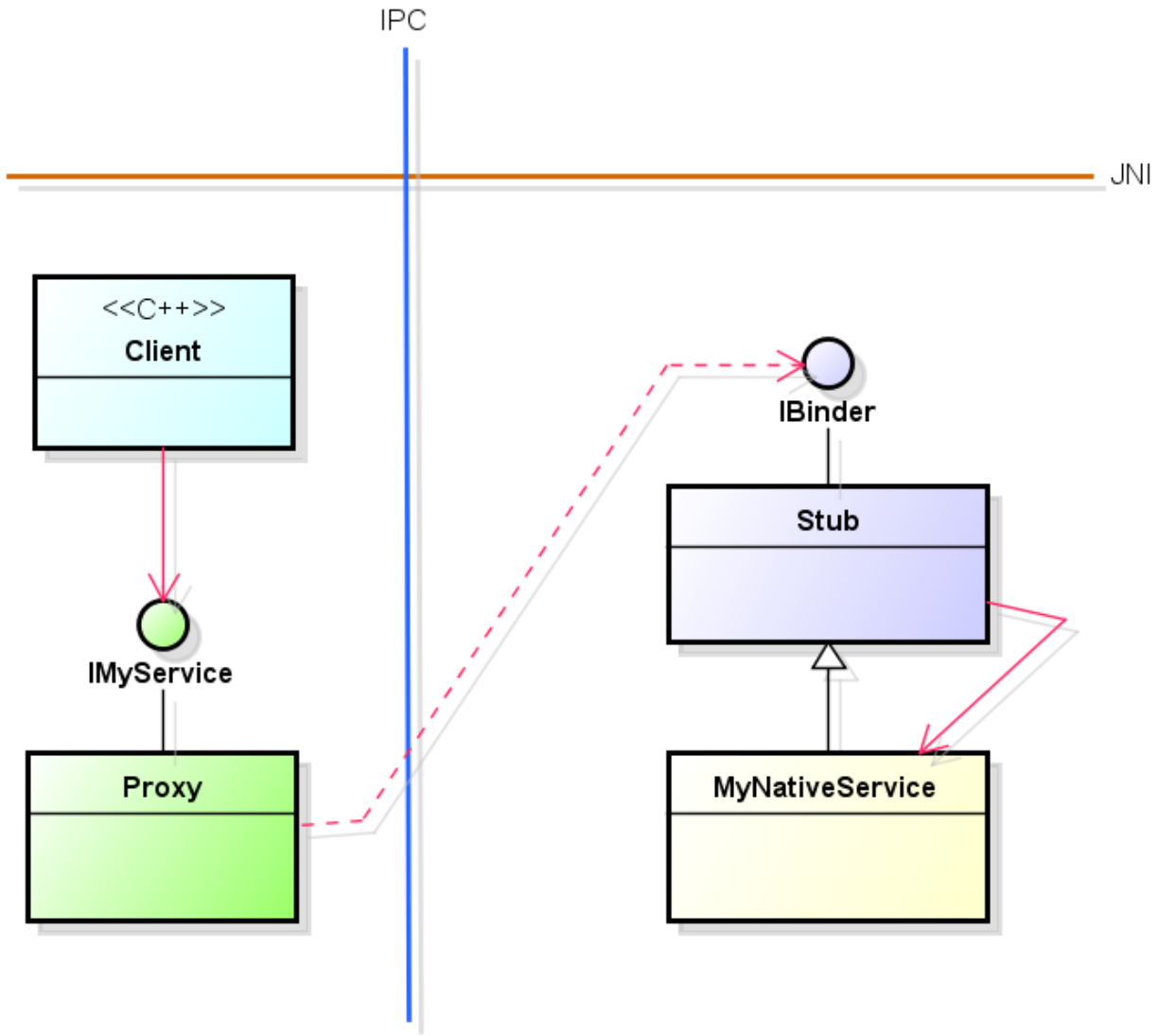

IPC
JNI
<<C++>>
Client
IBinder
BBinder
+ transact()
+ onTransact()
<<Stub Template>>
BnInterface<IMyService>
<<Stub>>
BnMyService
+ onTransact()
+ sv1() // 来自BnInterface<>
+ sv2()
+ sv3()
MyNativeService
+ sv1()
+ sv2()
+ sv3()
Stub

相当于


IPC
JNI
<<C++>>
Client
IBinder
BBinder
+ transact()
+ onTransact()
Stub
MyNativeService

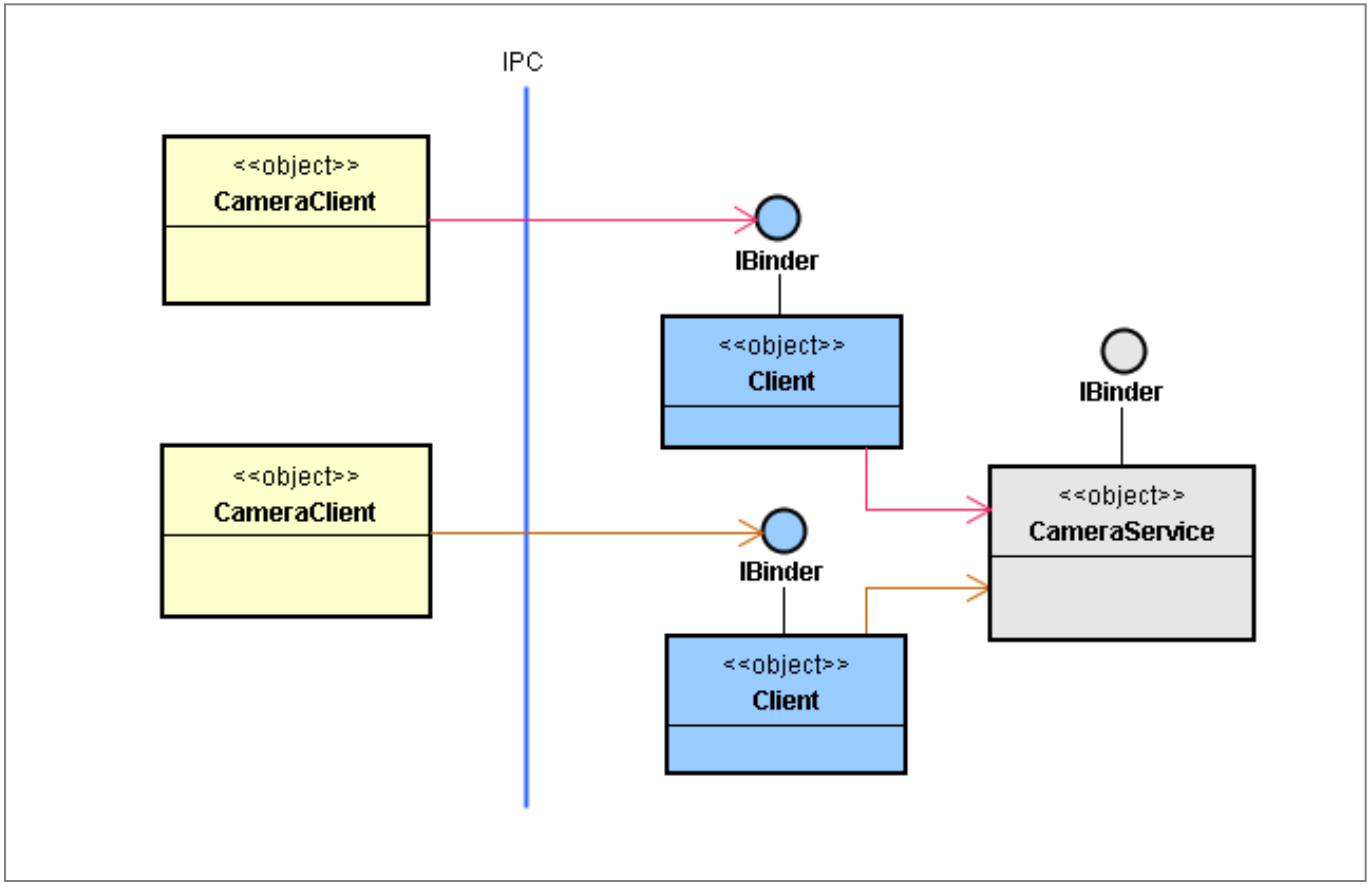
相当于



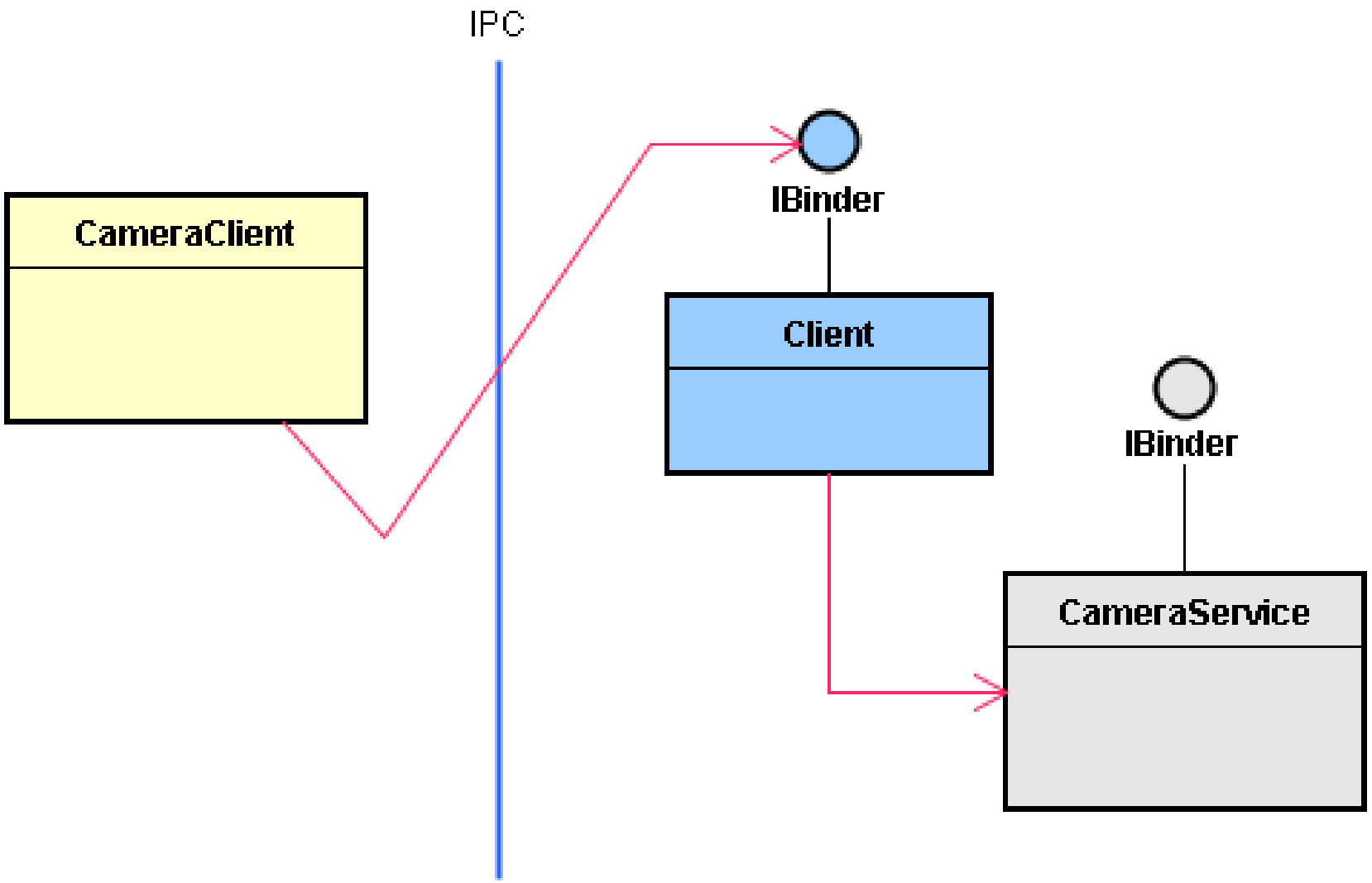

IPC
JNI
<<C++>>
Client
IMyService
Proxy
IBinder
Stub
MyNativeService

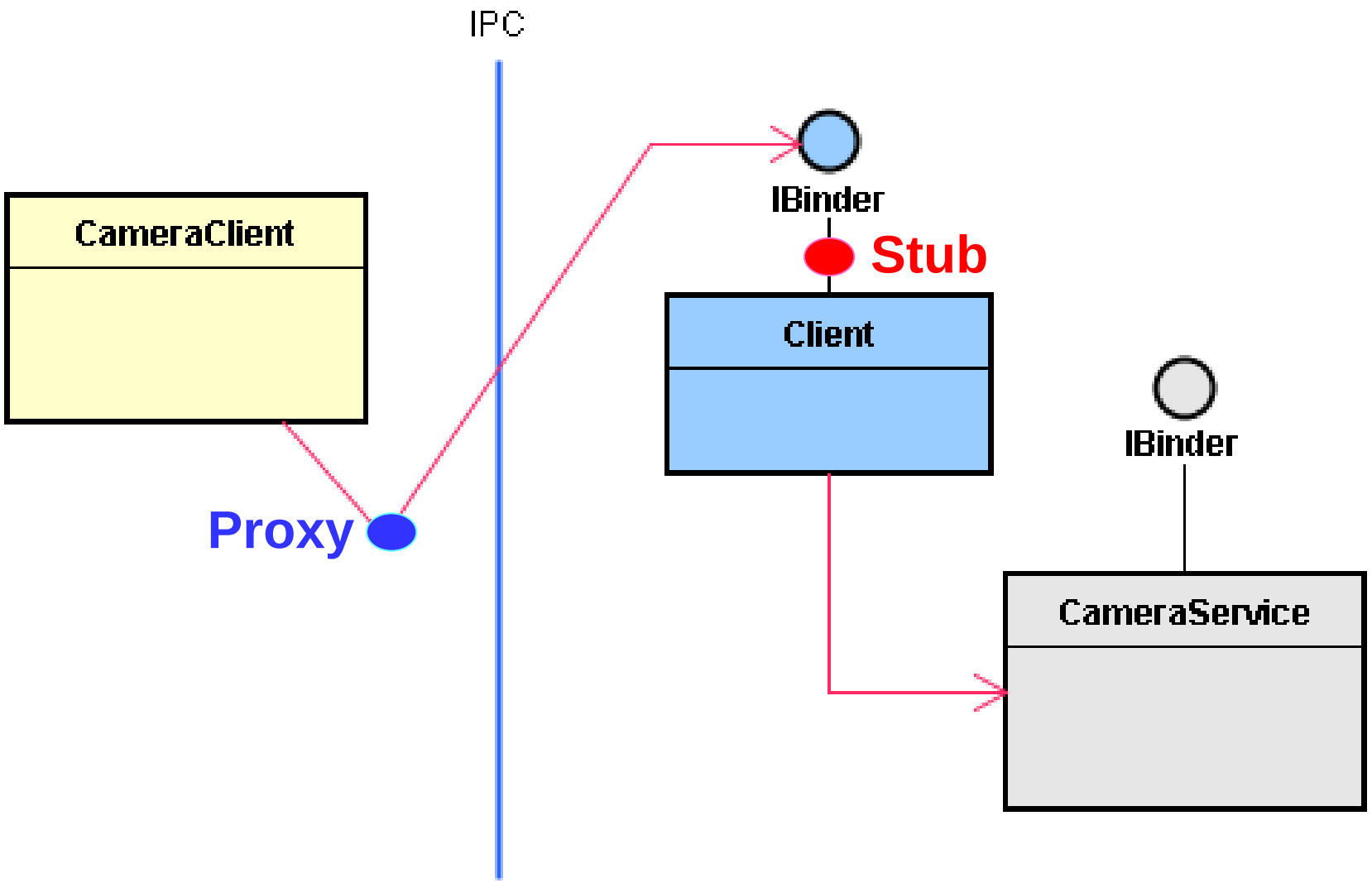
5、Proxy-Stub设计模式：以CameraService为例

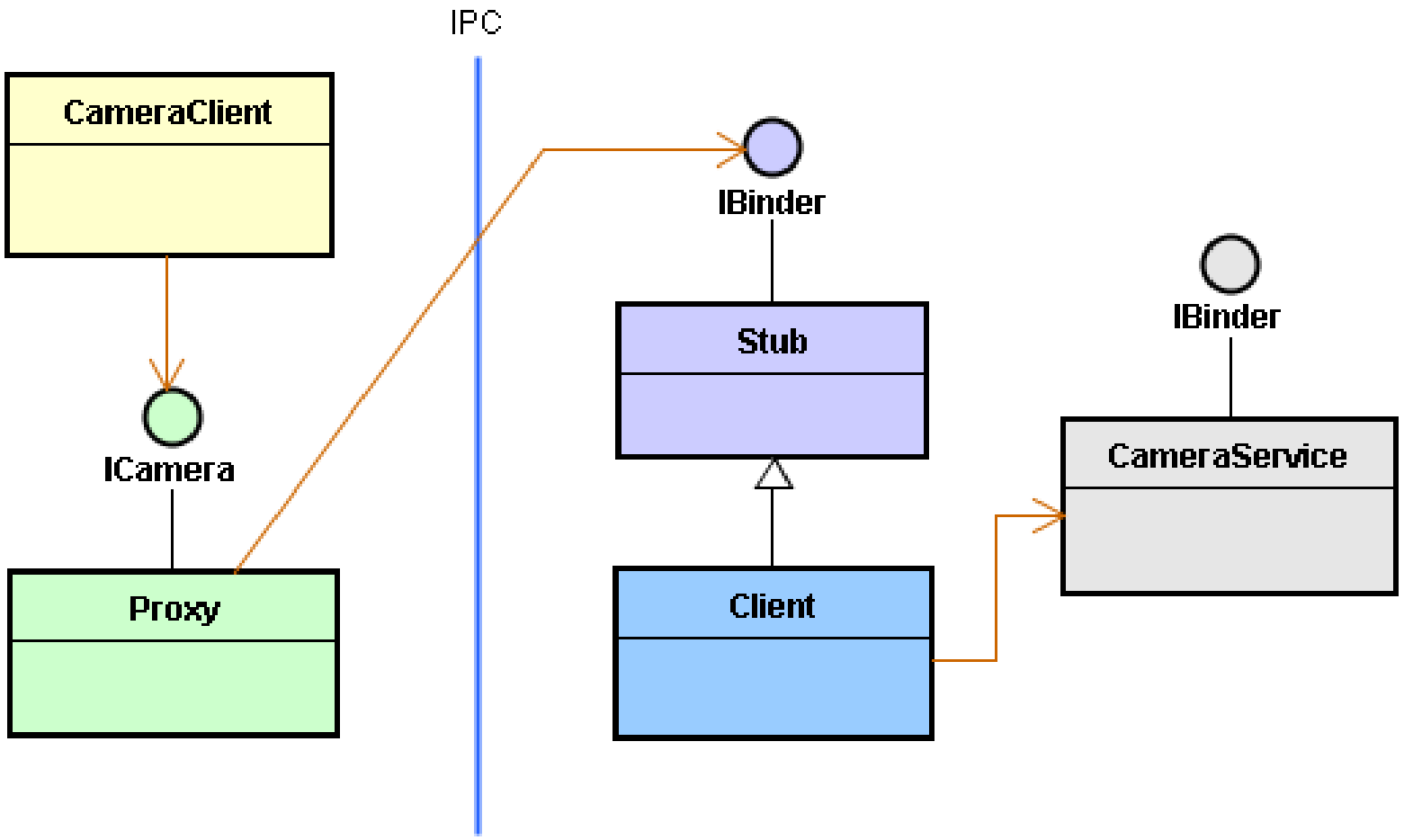
- 在Android里，CameraService采用了Session模式，如下图：

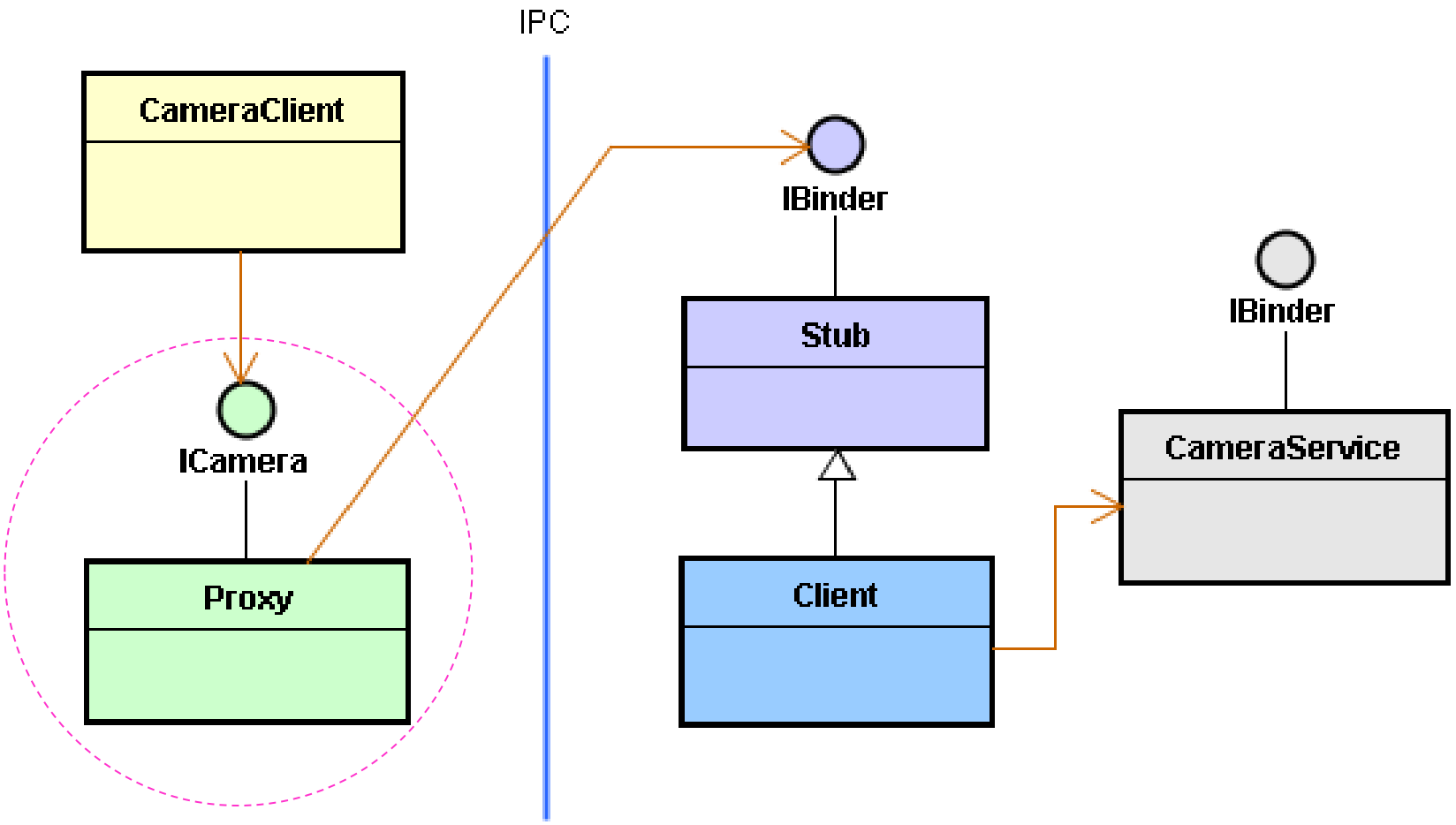


Session模式
+
Proxy-Stub模式


IPC
CameraClient
IBinder
Client
IBinder
CameraService


IPC
IBinder
Stub
CameraClient
Client
IBinder
Proxy
CameraService

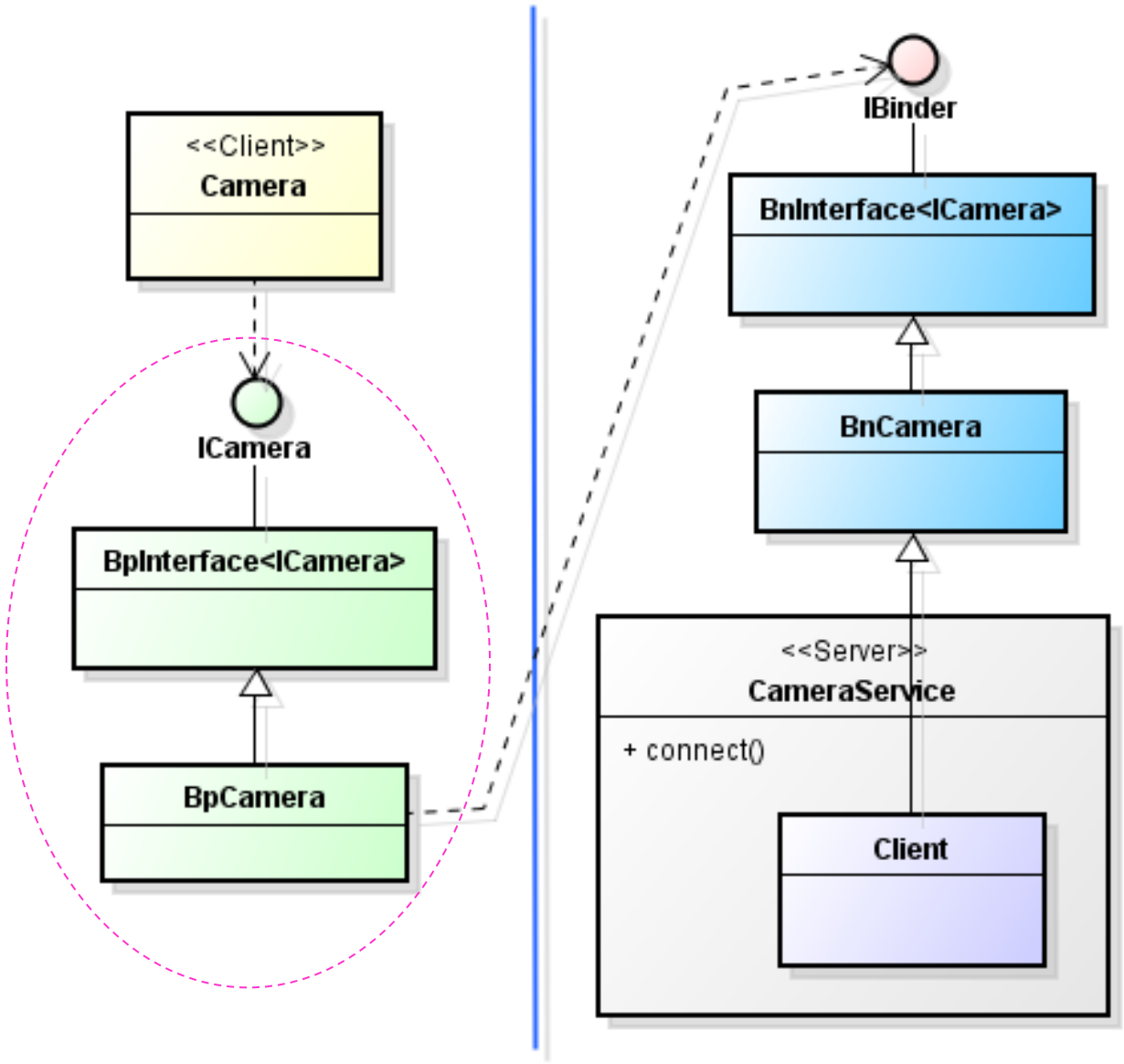

IPC
CameraClient
ICamera
Proxy
IBinder
Stub
Client
IBinder
CameraService


IPC
CameraClient
ICamera
Proxy
IBinder
Stub
Client
IBinder
CameraService

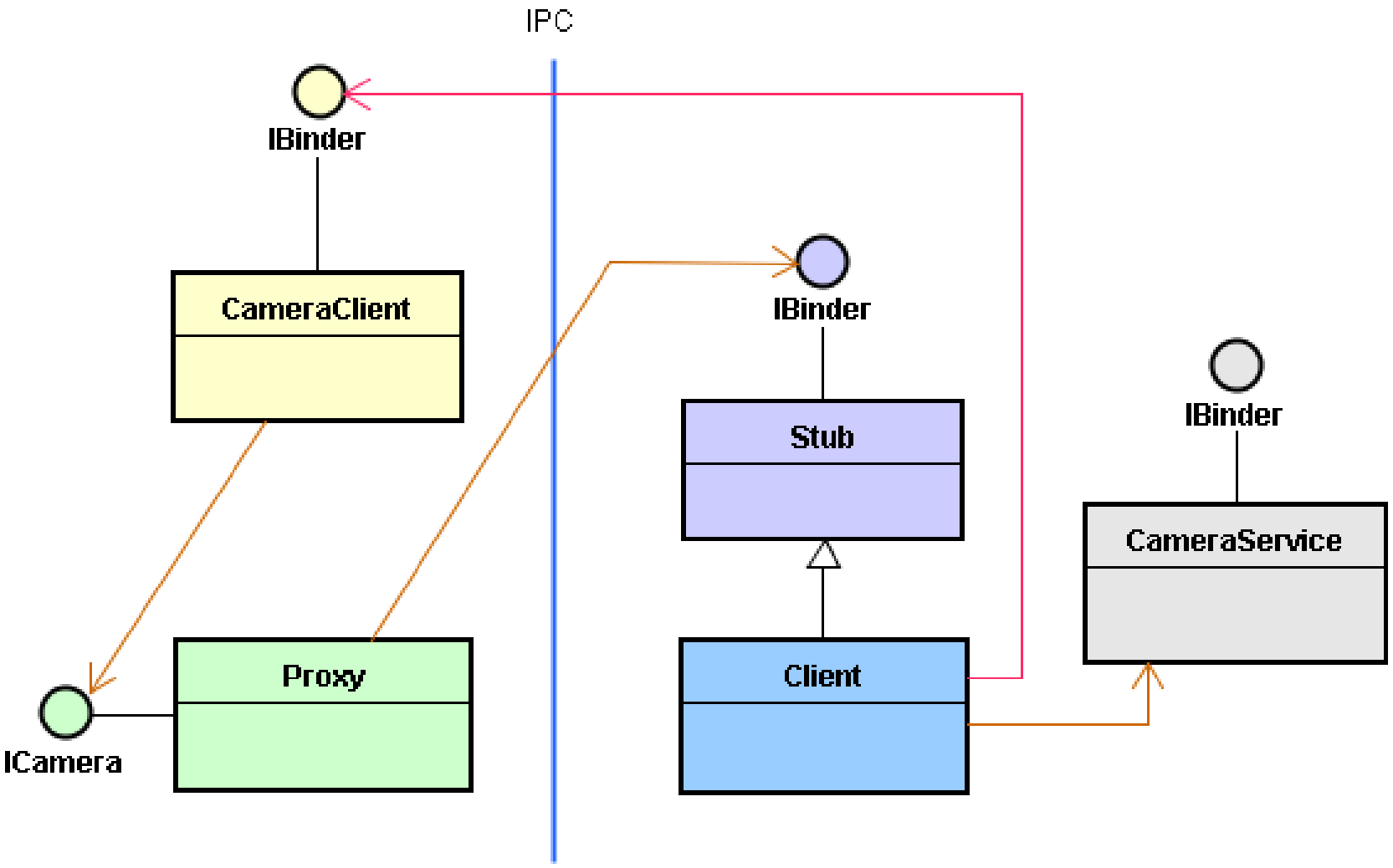
- 如果使用BpInterface<T>模板，这Proxy角色包含了两个类：

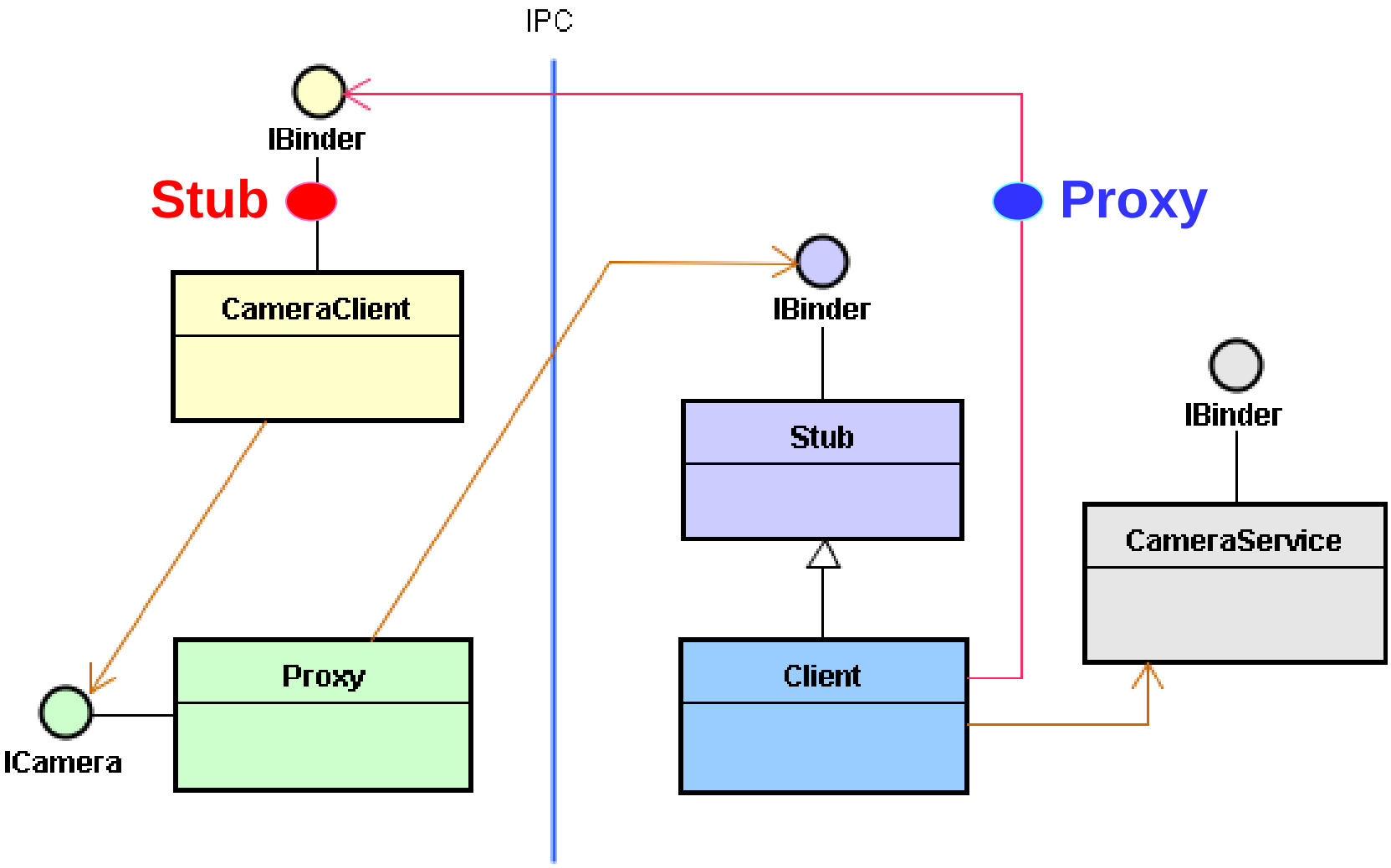
Proxy模板类：BpInterface<ICamera>

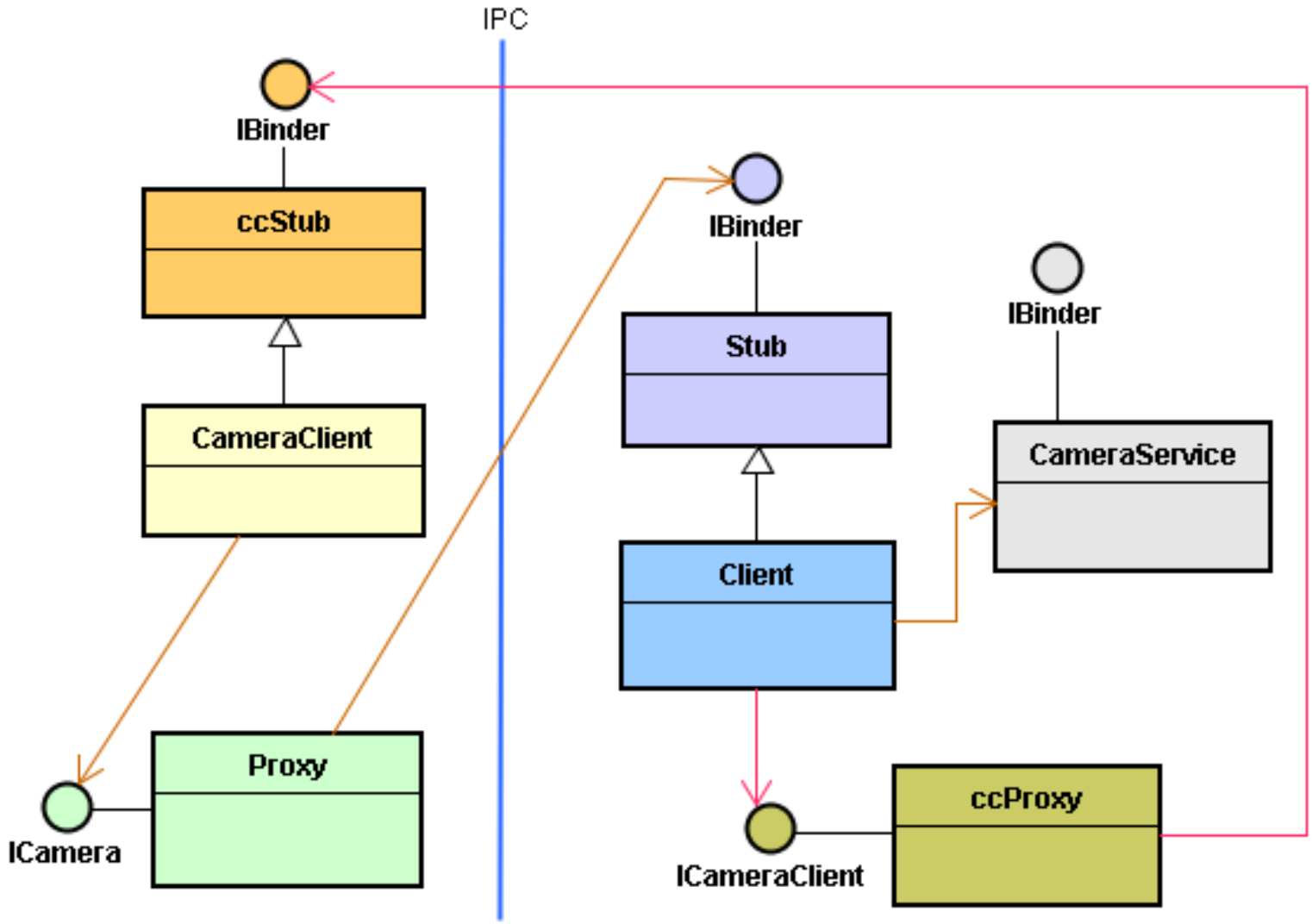
Proxy类：BpCamera

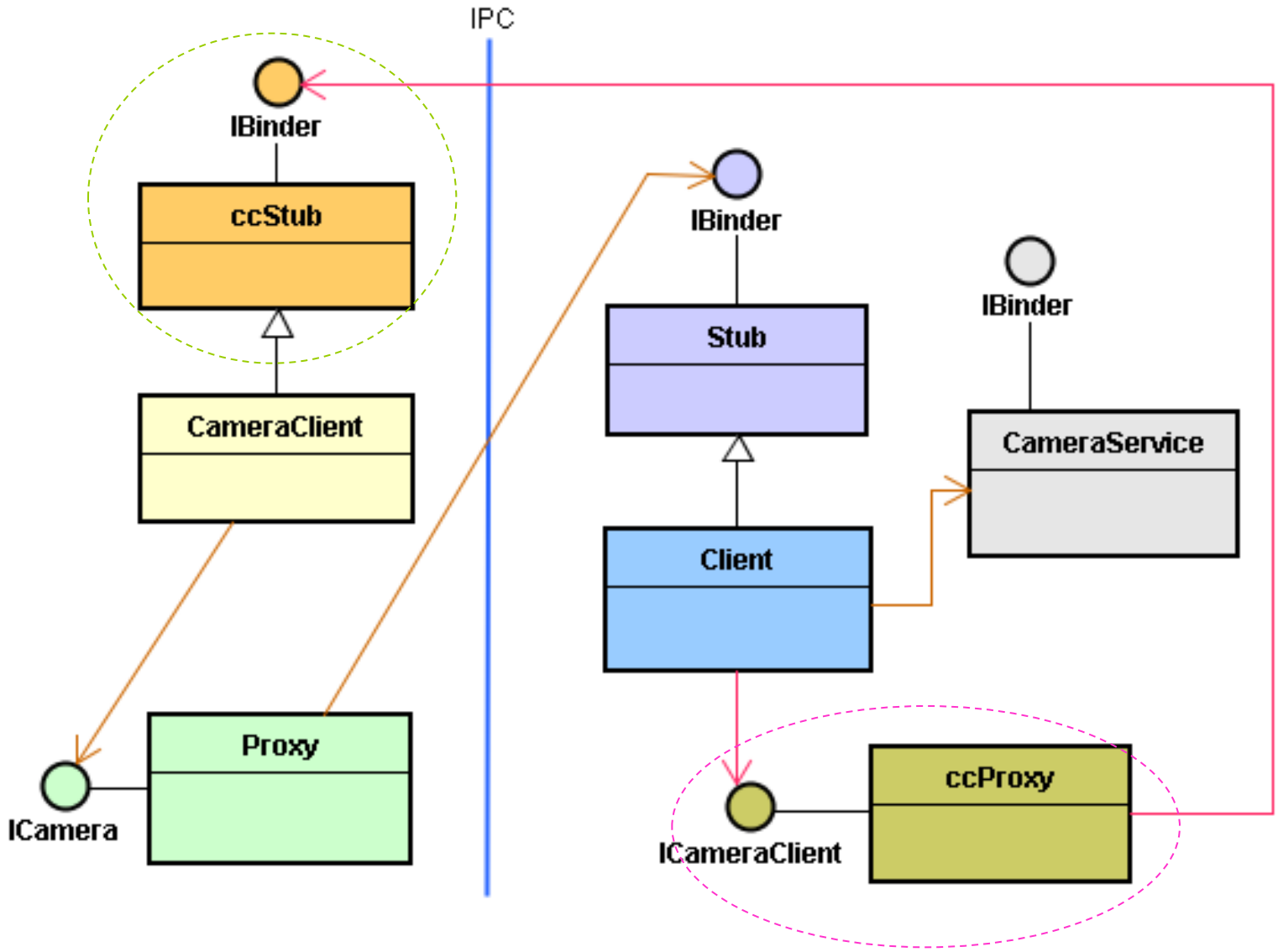

<<Client>>
Camera
ICamera
BpInterface<ICamera>
BpCamera
IBinder
BnInterface<ICamera>
BnCamera
<<Server>>
CameraService
+ connect()
Client

- 在上图的呼叫时，CameraClient可以将自己的IBinder接口传递给CameraService。
- 这让Client能调用CameraClient的IBinder接口，如下图：


IPC
IBinder
CameraClient
IBinder
Stub
IBinder
CameraService
Proxy
Client
ICamera


IPC
IBinder
Stub
Proxy
CameraClient
IBinder
IBinder
Stub
CameraService
Proxy
Client
ICamera


IPC
IBinder
ccStub
CameraClient
ICamera
Proxy
IBinder
Stub
Client
IBinder
CameraService
ICameraClient
ccProxy


IPC
IBinder
ccStub
CameraClient
ICamera
Proxy
IBinder
Stub
Client
IBinder
CameraService
ICameraClient
ccProxy

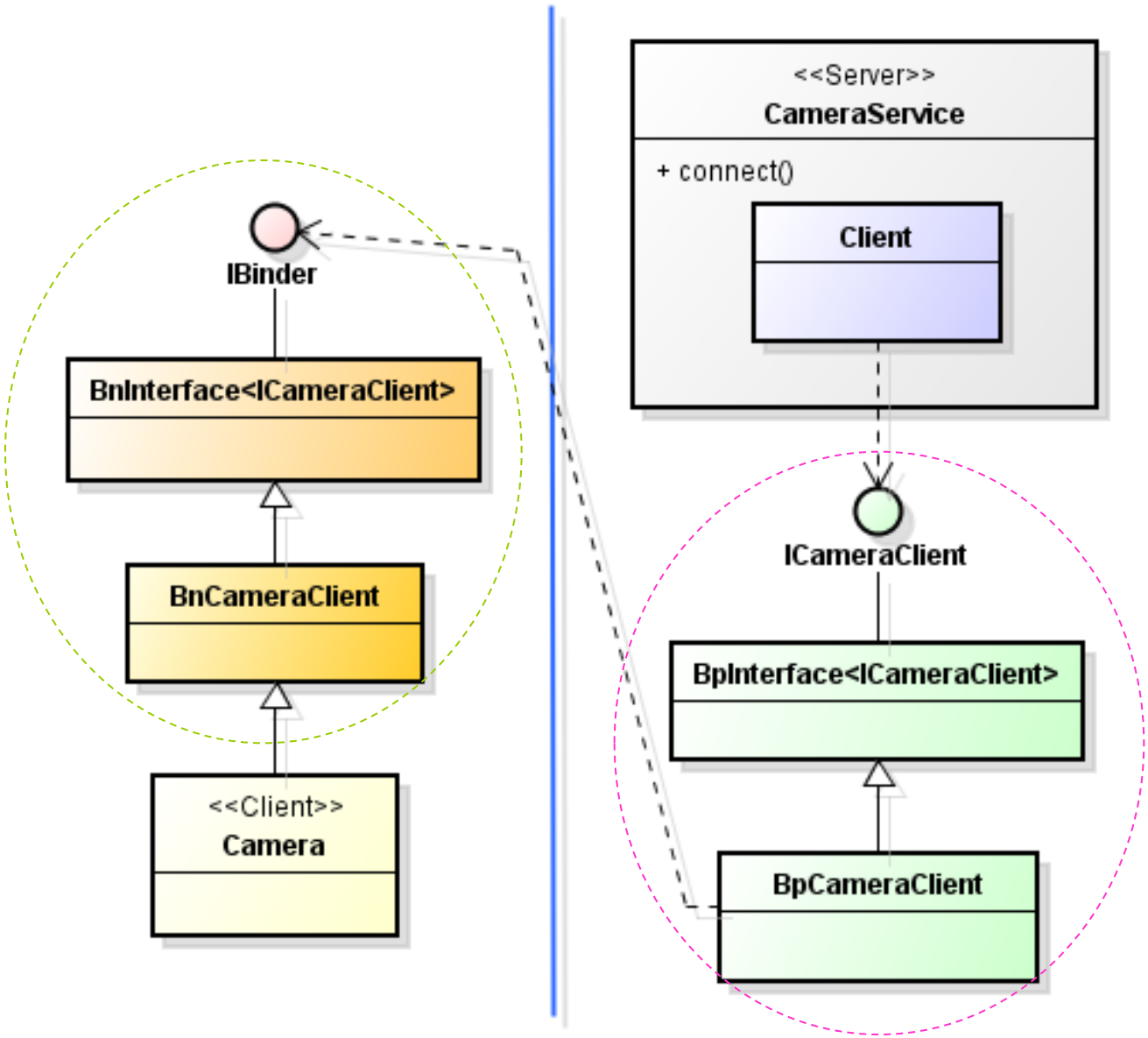
- 如果使用模板，这Proxy和Stub角色各包含了两个类：

Proxy模板类：BpInterface<ICameraClient>

Proxy类：BpCameraClient

Stub模板类：BnInterface<ICameraClient>

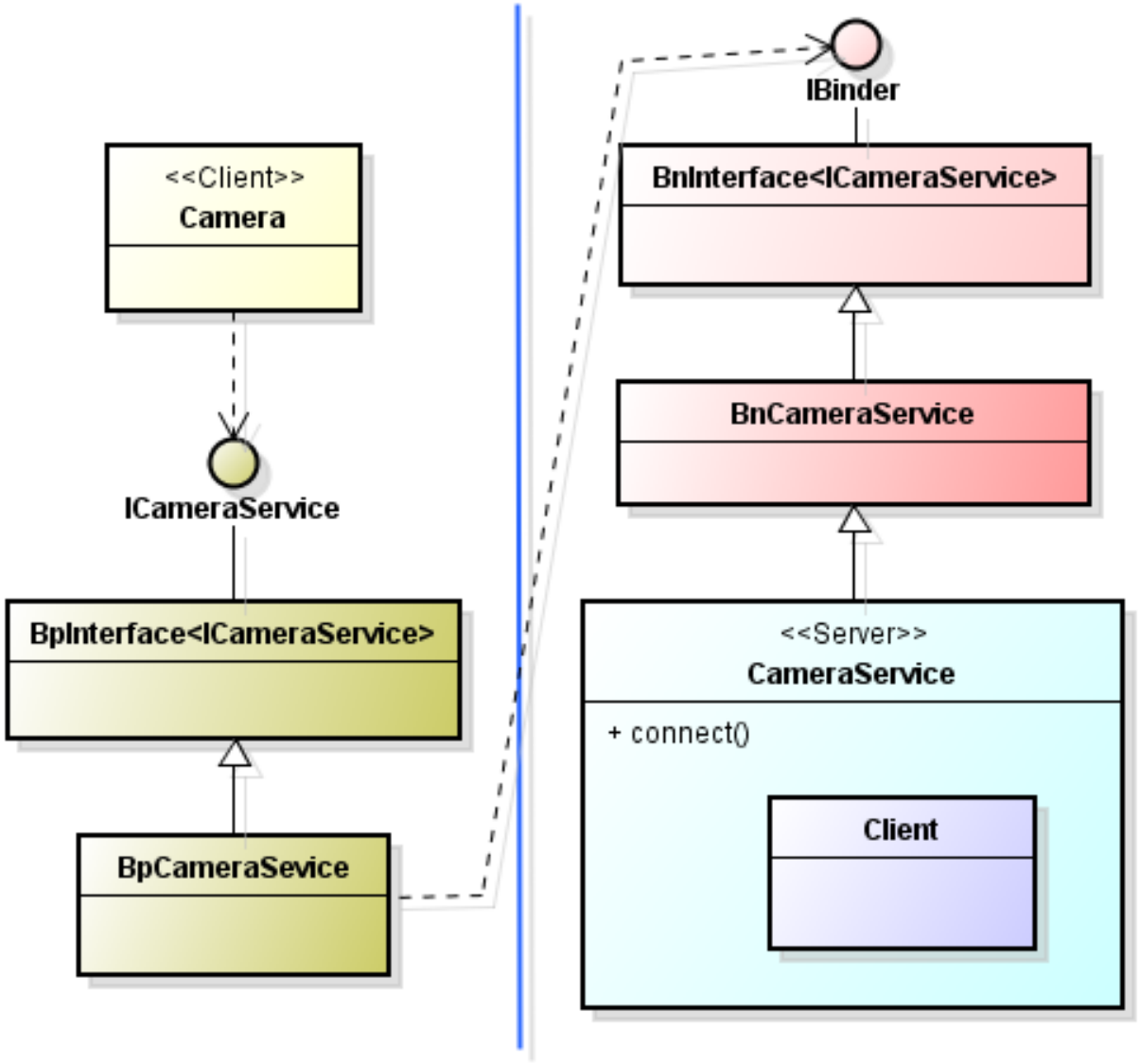
Stub类：BnCameraClient


<<Server>>
CameraService
+ connect()
Client
IBinder
BnInterface<ICameraClient>
BnCameraClient
<<Client>>
Camera
ICameraClient
BpInterface<ICameraClient>
BpCameraClient

- 以上说明了CameraService服务幕后，使用BpInterface<T>和BnInterface<T>模板来生成Proxy和Stub两个类

还有更多接口

例如，CameraService也提供了
<ICameraService>接口


IBinder
<<Client>>
Camera
BnInterface<ICameraService>
BnCameraService
ICameraService
BpInterface<ICameraService>
<<Server>>
CameraService
+ connect()
Client
BpCameraSevice

~ Continued ~