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_e

# Native核心服务的Proxy-Stub设计模式( e)

By 高煥堂

# 5、使用模板，产生Proxy类

- 在上节里，介绍了BnMyService类就是服务端(进程)的Stub类；那么，在Client端(进程)也能设计一个Proxy类；两者构成Proxy-Stub设计模式，来包装IBinder接口，以提供新的接口，来简化核心服务及其Client开发的复杂度。

- Android SDK提供了BpInterface<T>类别模板：

```
template<typename INTERFACE>
class BpInterface : public INTERFACE, public BpRefBase
        {
        public:
            BpInterface(const sp<IBinder>& remote);

        protected:
            virtual IBinder* onAsBinder();
        };
```

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

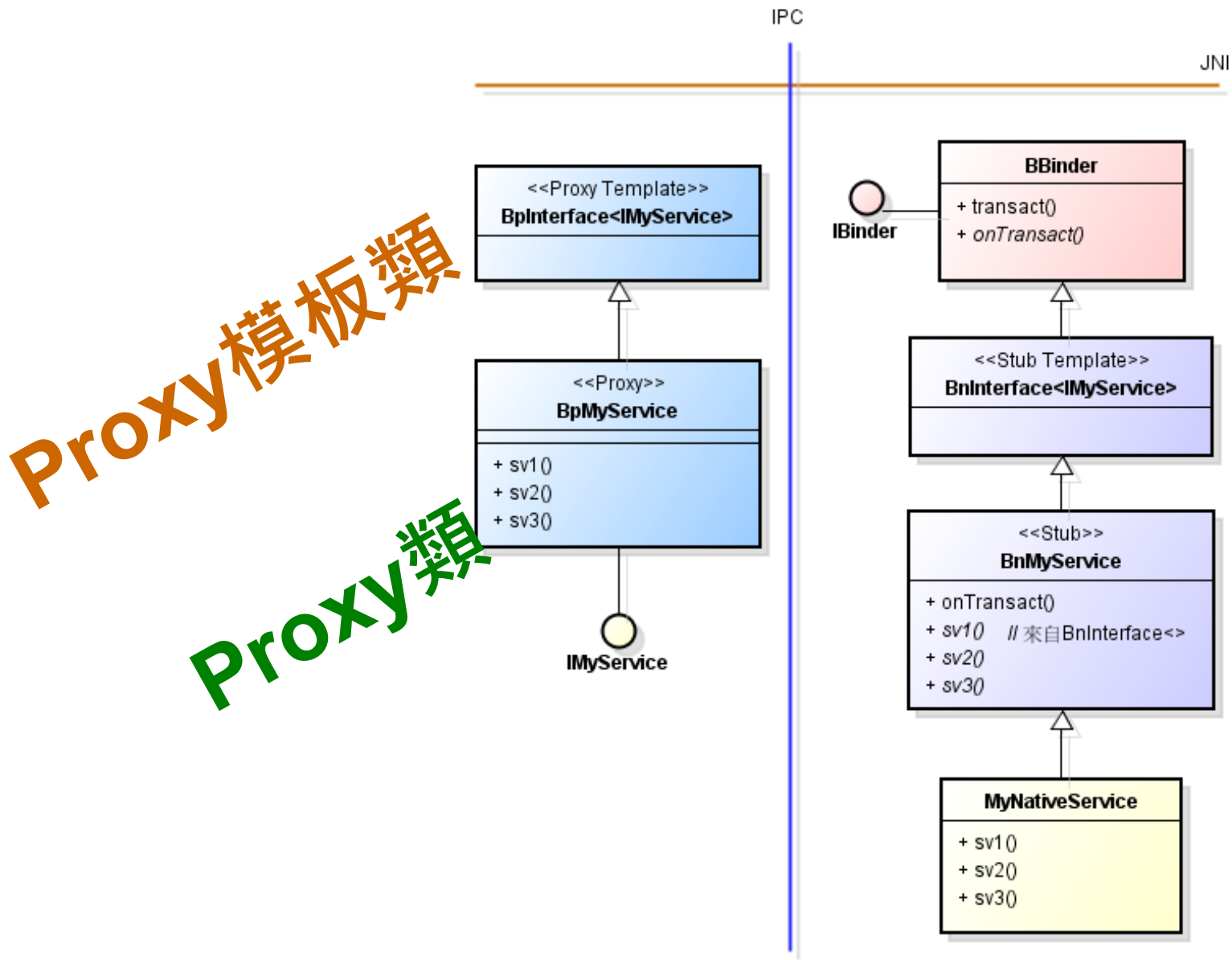
  BpInterface<IMyService>

- 它继承了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Proxy类别，如下：

```
class BpMyService : public BpInterface<IMyService>
        {
          //……
         }
```

如下图：

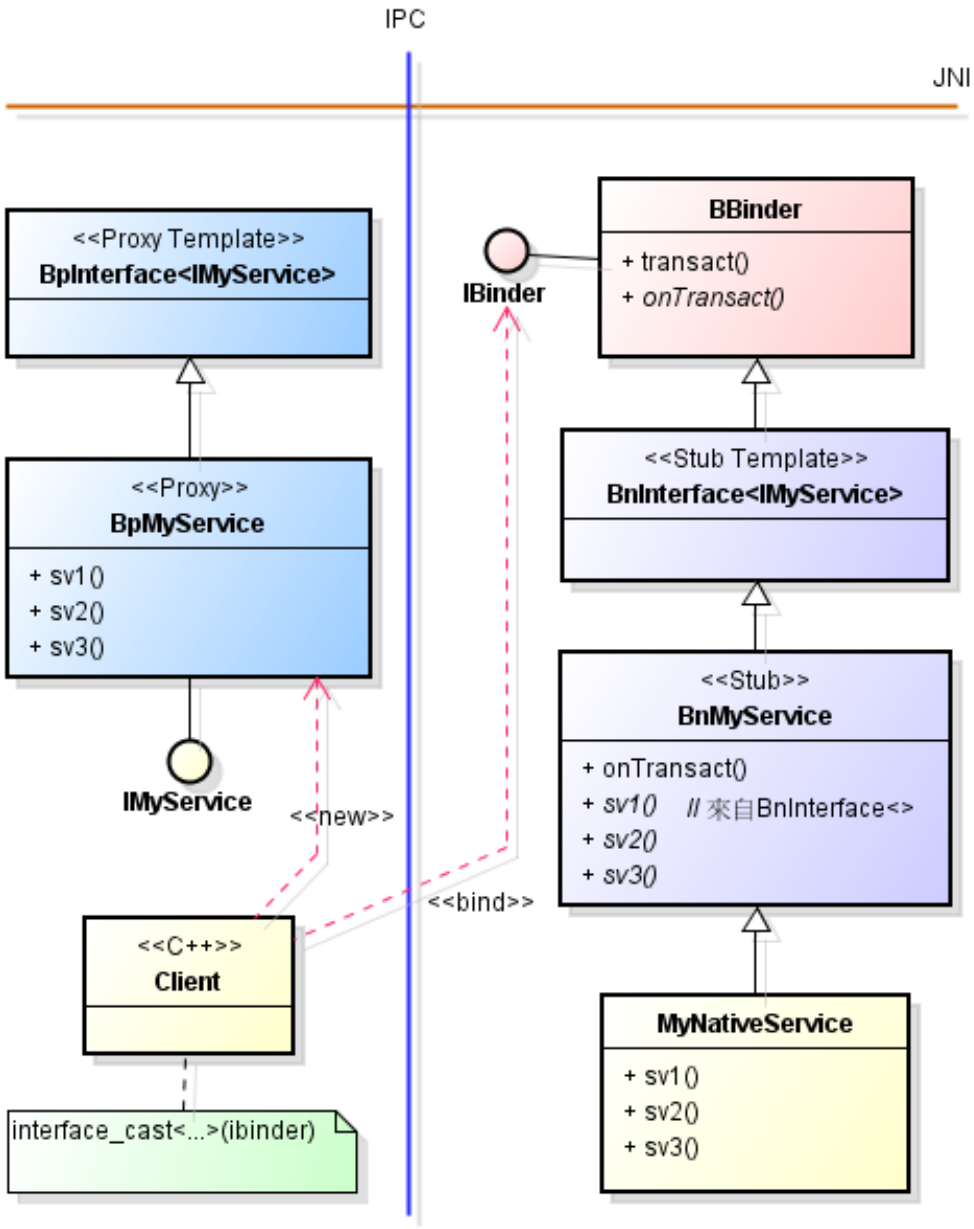

Proxy模板類
Proxy類
IPC
JNI
<<Proxy Template>>
BpInterface<IMyService>
<<Proxy>>
BpMyService
+ sv1()
+ sv2()
+ sv3()
IMyService
IBinder
BBinder
+ transact()
+ onTransact()
<<Stub Template>>
BnInterface<IMyService>
<<Stub>>
BnMyService
+ onTransact()
+ sv1() // 來自BnInterface<>
+ sv2()
+ sv3()
MyNativeService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

# 6、让Client使用Proxy类的新接口

- Client先透过SM(ServiceManager)核心服务去绑定MyNativeService的IBinder接口，取得接口的指针(即ibinder)。

```
sp<IServiceManager> sm = defaultServiceManager();
sp<IBinder> ibinder =
        sm->getService(String16("misoo.myNS"));
```

- 这要求SM协助绑定MyNativeService核心服务。
- 绑定了，SM就会在Client进程里诞生一个分身：BpBinder对象。
- 接着，SM就将BpBinder的IBinder接口(如ibinder指针)，回传给Client模块。


IPC
JNI
<<Proxy Template>>
BpInterface<IMyService>
<<Proxy>>
BpMyService
+ sv1()
+ sv2()
+ sv3()
IMyService
<<new>>
<<C++>>
Client
interface_cast<...>(ibinder)
IBinder
BBinder
+ transact()
+ onTransact()
<<Stub Template>>
BnInterface<IMyService>
<<Stub>>
BnMyService
+ onTransact()
+ sv1() // 來自BnInterface<>
+ sv2()
+ sv3()
<<bind>>
MyNativeService
+ sv1()
+ sv2()
+ sv3()

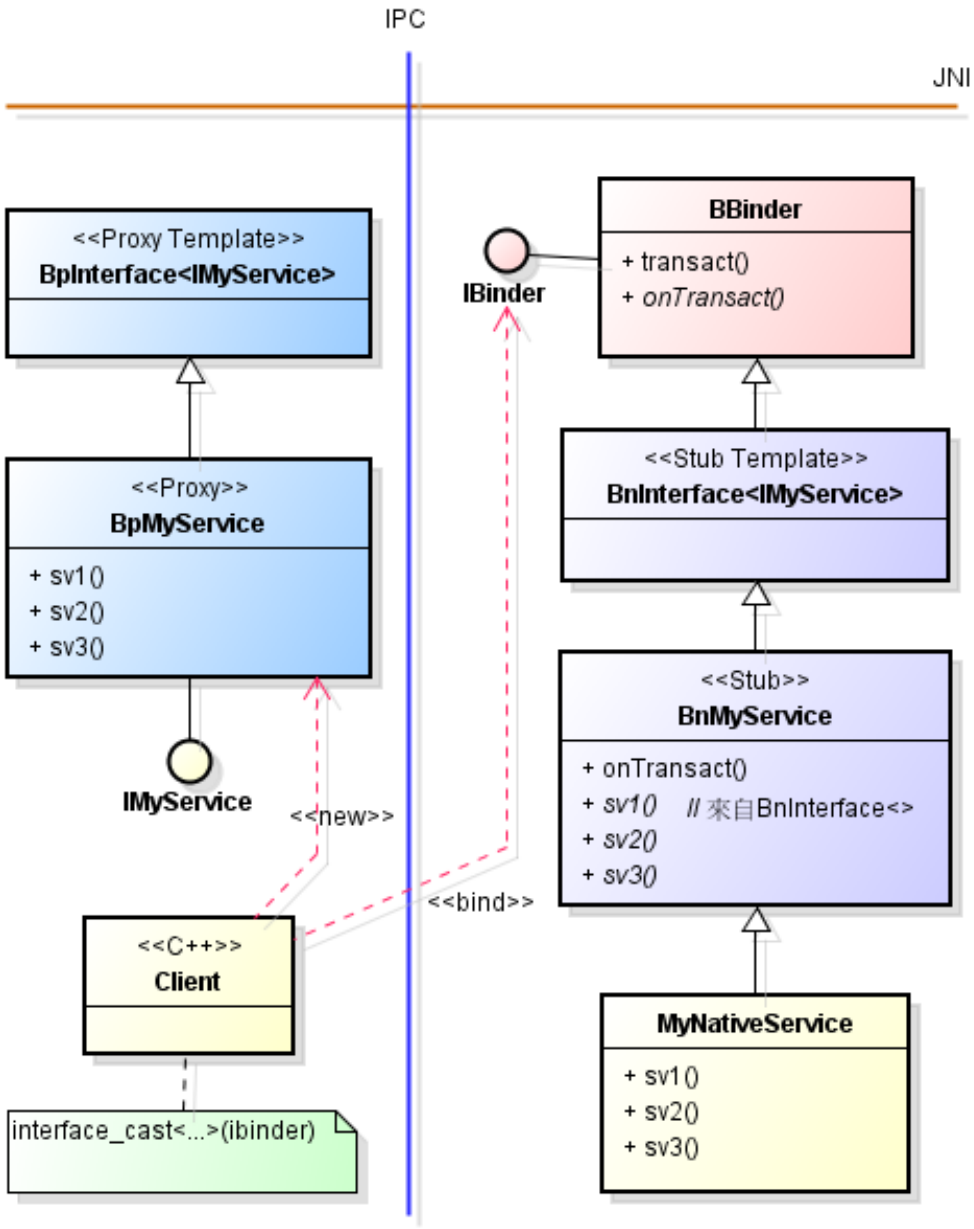
- 当Client取得ibinder之后，就使用interface_cast<T>()函数模板：

```
interface_cast<IMyService>(ibinder);
```

- 撰写如下的指令：

```
sp<IMyService> sService =
    interface_cast<IMyService>(ibinder);
```

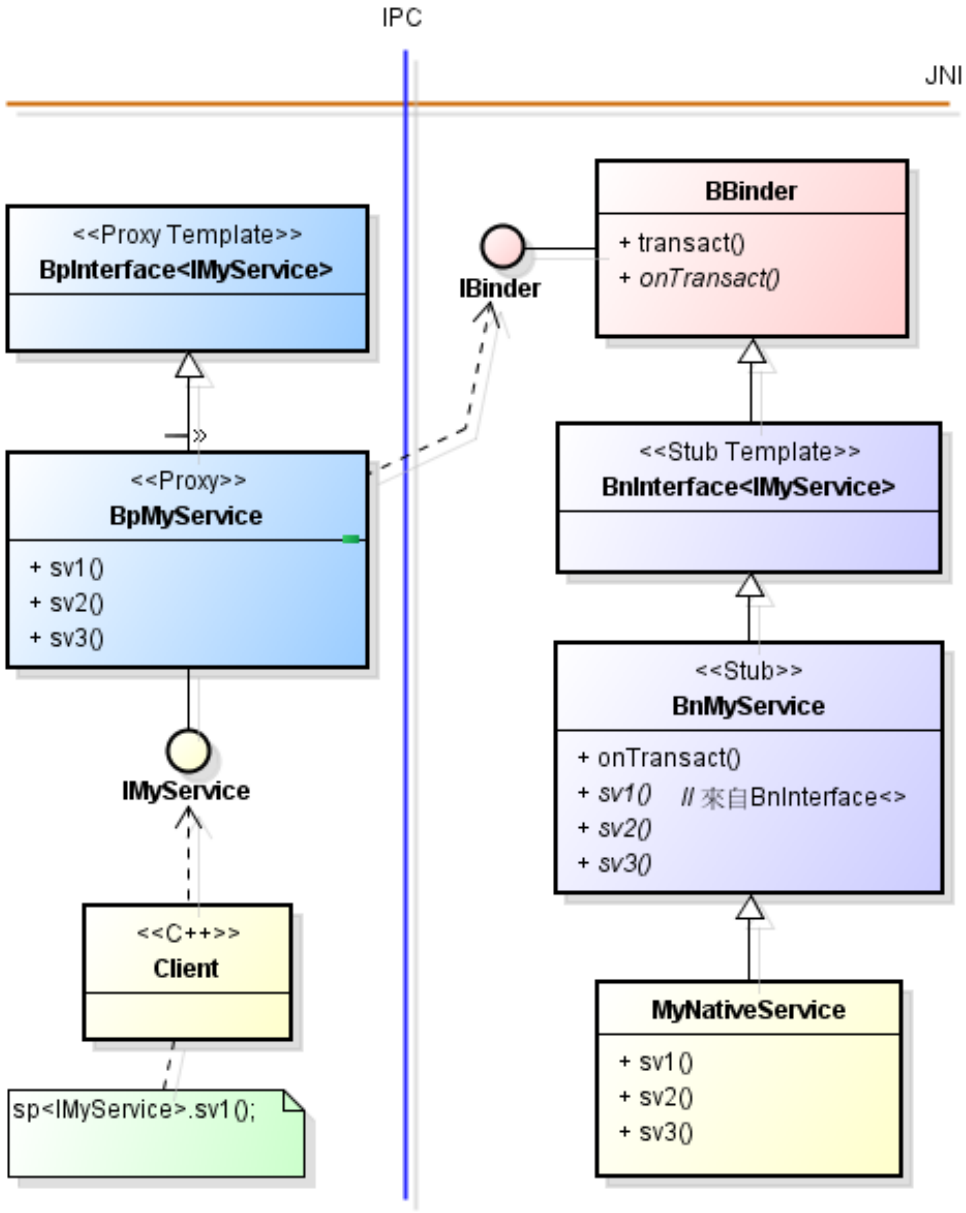
- 这使用樣板interface_cast<T>来转换出IMyService新接口。


IPC
JNI
<<Proxy Template>>
BpInterface<IMyService>
<<Proxy>>
BpMyService
+ sv1()
+ sv2()
+ sv3()
IMyService
<<new>>
<<C++>>
Client
interface_cast<...>(ibinder)
IBinder
BBinder
+ transact()
+ onTransact()
<<Stub Template>>
BnInterface<IMyService>
<<Stub>>
BnMyService
+ onTransact()
+ sv1() // 來自BnInterface<>
+ sv2()
+ sv3()
<<bind>>
MyNativeService
+ sv1()
+ sv2()
+ sv3()

- 继续执行到指令：

  sService->sv1(…);

- Client就使用了Proxy类提供的IMyService接口；并调用到 MyNativeServce了。


IPC
JNI
<<Proxy Template>>
BpInterface<IMyService>
IBinder
BBinder
+ transact()
+ onTransact()
<<Proxy>>
BpMyService
+ sv1()
+ sv2()
+ sv3()
<<Stub Template>>
BnInterface<IMyService>
IMyService
<<Stub>>
BnMyService
+ onTransact()
+ sv1() // 來自BnInterface<>
+ sv2()
+ sv3()
<<C++>>
Client
MyNativeService
+ sv1()
+ sv2()
+ sv3()
sp<IMyService>.sv1();

- 就Client模块的开发而言，此刻只需要使用新API(即IMyService接口)来呼叫所熟悉的sv1()、sv2()和sv3()函数即可，不必知道IBinder的存在及其写法，因而简化许多了。

# Thanks...

高煥堂