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

D06_b

核心服务Callback的IBinder接口设计(b)

By 高煥堂

4、Client模块
范例代码实现

- 撰写Client模块：即SQR类别

```
// SQR.h
#ifndef ANDROID_MISOO_SQR_H
#define ANDROID_MISOO_SQR_H
namespace android {
class SQR : public BBinder
  {
      sp<IBinder> m_ib;
      int value;
const void getAddService();
  public:
    SQR();
    void execute(int n);
    int getValue();
    virtual status_t onTransact(uint32_t, const Parcel&, Parcel*,
uint32_t);
  };
}; //namespace
#endif
```

```
//  SQR.cpp
#include <utils/IServiceManager.h>
#include <utils/IPCThreadState.h>
#include "SQR.h"
namespace android {
SQR::SQR()
{  getSQRService();  }

const void SQR::getSQRService(){
    sp<IServiceManager> sm = defaultServiceManager();
            m_ib = sm->getService(String16("misoo.sqr"));
                LOGE("SQR.getSQRService %p\n", sm.get());
            if (m_ib == 0)
                LOGW("SQRService not published, waiting...");
          return;
}
```

```
void SQR::execute(int n) {
	Parcel data, reply;
	data.writeInt32(n);

	data.writeStrongBinder( this );

	 LOGE("SQR::execute\n");
	m_ib->transact(0, data, &reply);
	return;
}
int SQR::getValue() {
		return value;
}
```

```
status_t SQR::onTransact(uint32_t code, const Parcel& data,
                    Parcel* reply, uint32_t flags){
        switch(code) {
           case 0: {
              value = data.readInt32();
             // LOGE("SQRService::onTransact. %d\n", x);
              return NO_ERROR;
            }
break;
        default:
                return BBinder::onTransact(code, data, reply, flags);
    }}
}; //namespace
```

- 其中，大家比较好奇的是，核心服务如何去绑定Client模块的IBinder接口呢?
- 事实上，也很简单，答案是：由于正向调用发生在先，而回调时间在后；所以在正向调用时，将Client模块的IBinder接口，当作参数传递给核心服务就行了。
- 于是，上述代码的执行情境，兹说明如下：

- 首先，要求SM(Service Manager)协助绑定SQRService核心服务，如下代码：

```
const void SQR::getSQRService()
 {
      sp<IServiceManager> sm = defaultServiceManager();
      m_ib = sm->getService(String16("misoo.sqr"));
      …….
 }
```

- 然后，进行正向调用，其程序代码为：

```
void SQR::execute(int n) {
        Parcel data, reply;
        data.writeInt32(n);
        data.writeStrongBinder(this);

        m_ib->transact(0, data, &reply);
        return;
}
```

- 這先执行指令：

```
Parcel data, reply;
data.writeInt32(n);
data.writeStrongBinder(this);
```

- 这准备好参数，并将 SQR自己的IBinder接口存入参数变量里。

- 接着执行正向调用指令：

  m_ib->transact()

  就将SQR的 IBinder接口传递给了SQRService了。

- 此时，就开始执行了SQRService的onTransact()函数了。
- 也就是指令：

```
status_t SQRService::onTransact(uint32_t code, const
    Parcel& data, Parcel* reply, uint32_t flags)
      {
        switch(code) {
           case 0: {
              int x = data.readInt32();
              callback_ib = data.readStrongBinder();
              // ……..
              callback_ib->transact(0, data1, &reply1);
              // ……..
      }
}
```

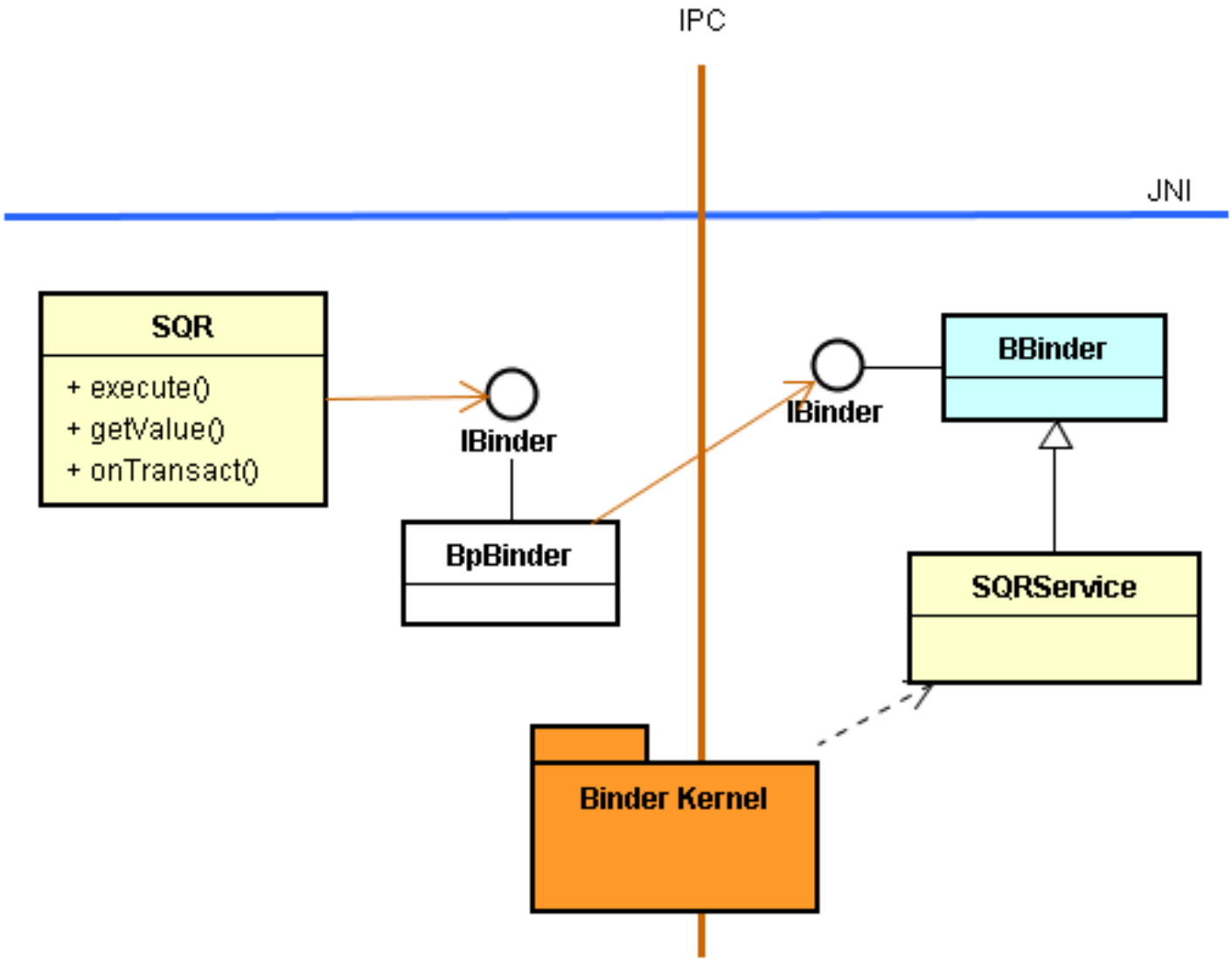
- 就從參數裡取出IBinder接口，绑定了SQR的IBinder接口。
- 然后，就执行callback_ib->transact()指令，顺利回调到SQR类别了。
- 此时，就开始执行了SQR类别的onTransact()函数了。

5、Summary

同步

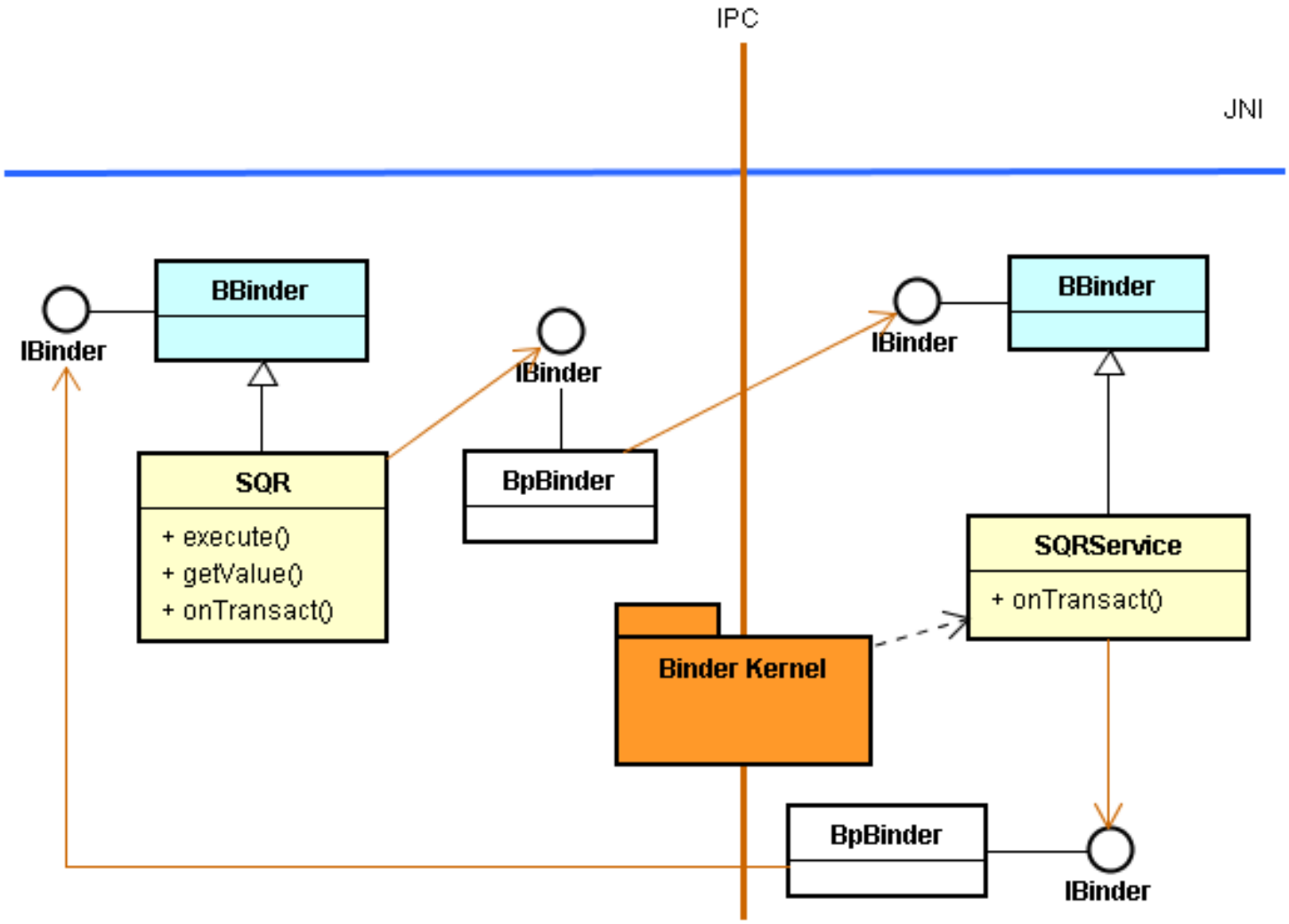
- 在前面的范例里，都采正向远距调用途径，其核心服务可以及时将参数值回传给Client模块，这是大家所熟悉的「同步」数据回传。

```
void SQR::execute(int n) {
	Parcel data, reply;
	data.writeInt32(n);
	data.writeStrongBinder( this );
	LOGE("SQR::execute\n");

	m_ib->transact(0, data, &reply);

	return;
}
```


IPC
JNI
SQR
+ execute()
+ getValue()
+ onTransact()
IBinder
BpBinder
IBinder
BBinder
SQRService
Binder Kernel

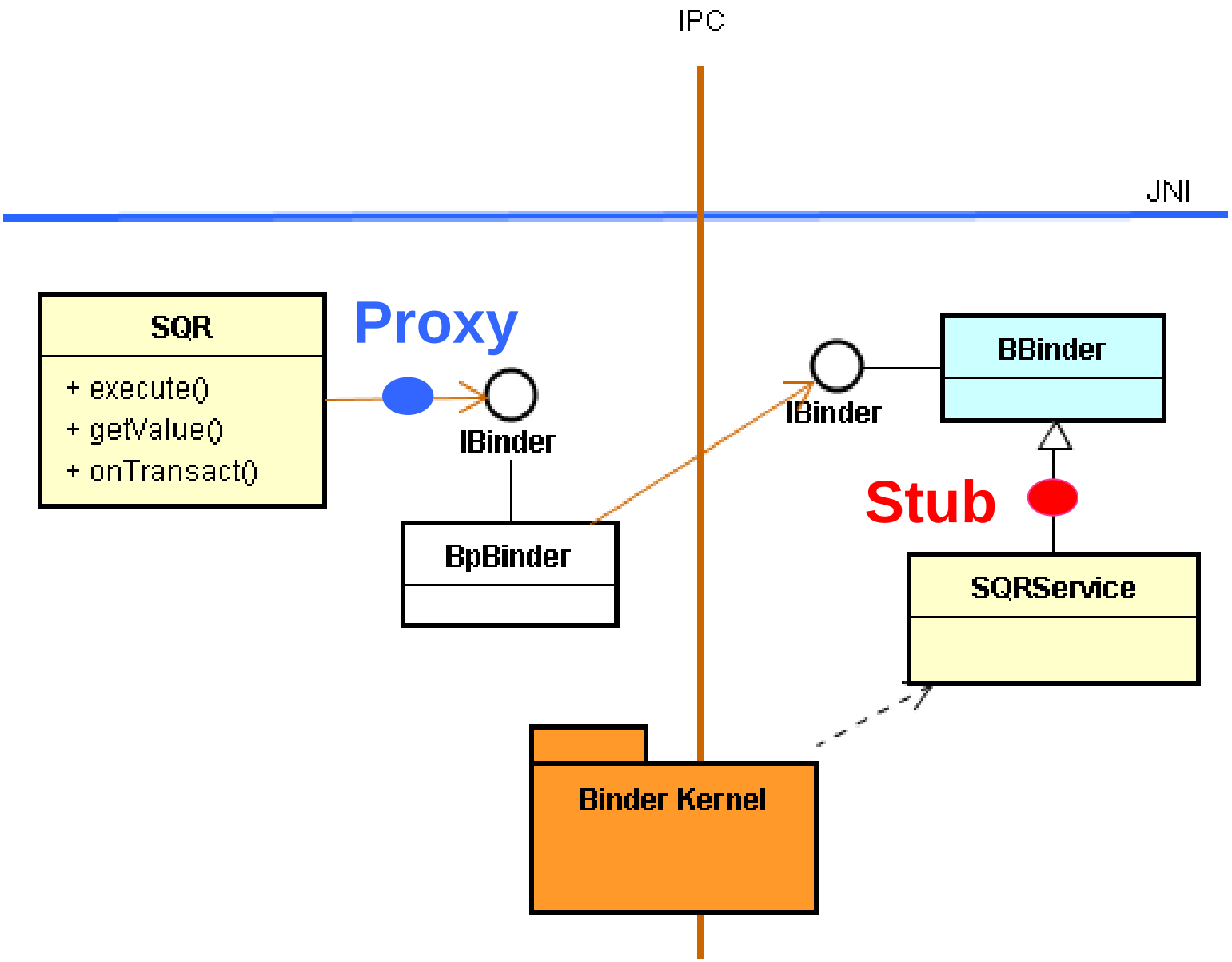
異步

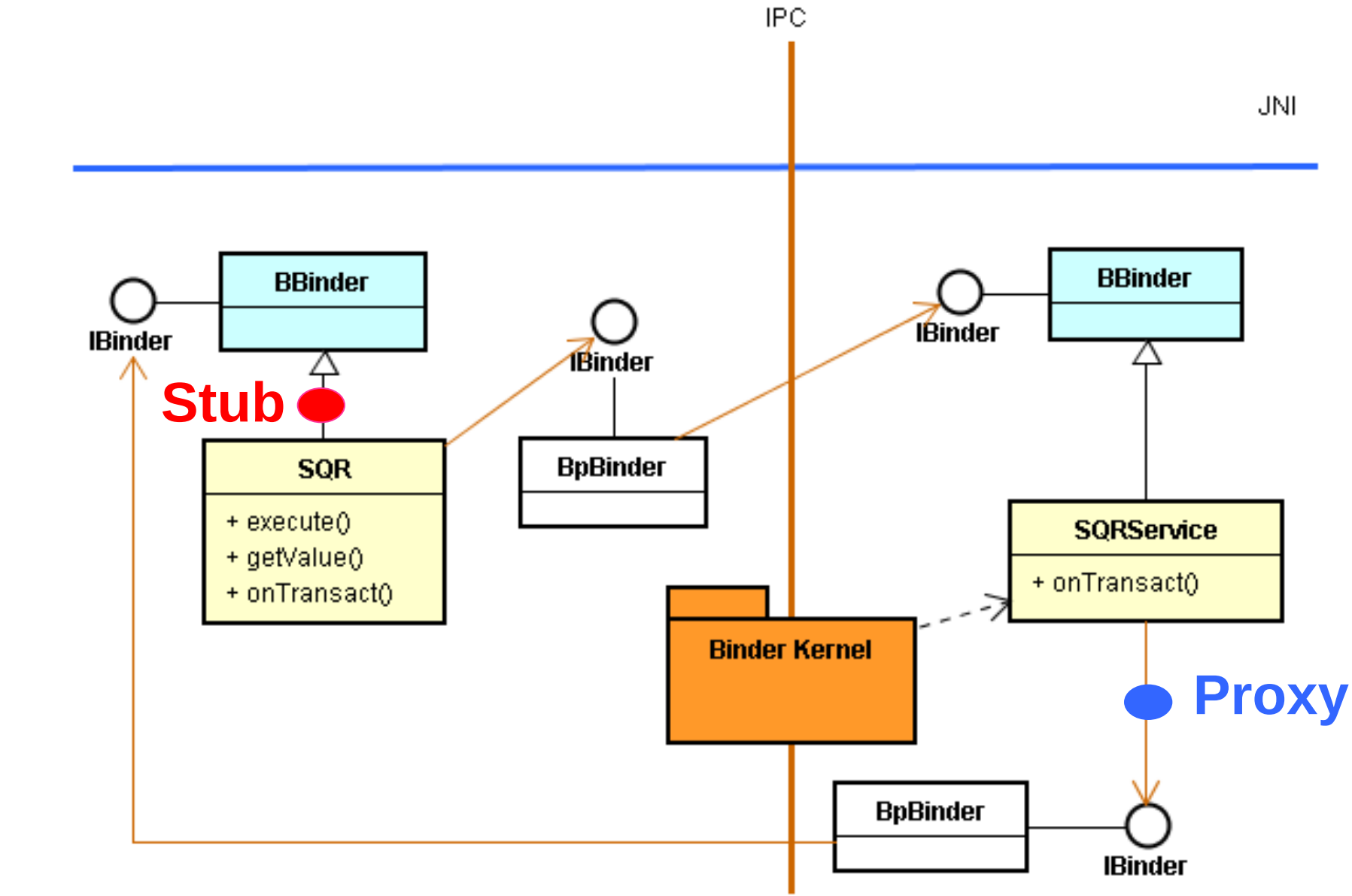
- 在本范例里的核心服务则是在上述同步回传之后，必要时(或突发状况下)才由核心服务主动回调，来将数据回传给Client模块，这是通称的「异步」数据回传。


IPC
JNI
BBinder
IBinder
IBinder
SQR
+ execute()
+ getValue()
+ onTransact()
BpBinder
IBinder
BBinder
SQRService
+ onTransact()
Binder Kernel
BpBinder
IBinder

设计回调(Callback)接口

- 在此范例里，其回调机制仅使用到IBinder接口。我们也可以藉由BpInterface<T>和BnInterface<T>模板产生Proxy和Stub类，来将上述回调机制的IBinder接口包装起来，以便提供好用的接口。


IPC
JNI
SQR
+ execute()
+ getValue()
+ onTransact()
Proxy
IBinder
BpBinder
IBinder
BBinder
Stub
SQRService
Binder Kernel


IPC
JNI
BBinder
IBinder
Stub
SQR
+ execute()
+ getValue()
+ onTransact()
IBinder
BpBinder
IBinder
BBinder
SQRService
+ onTransact()
Binder Kernel
Proxy
BpBinder
IBinder

Thanks...

高煥堂