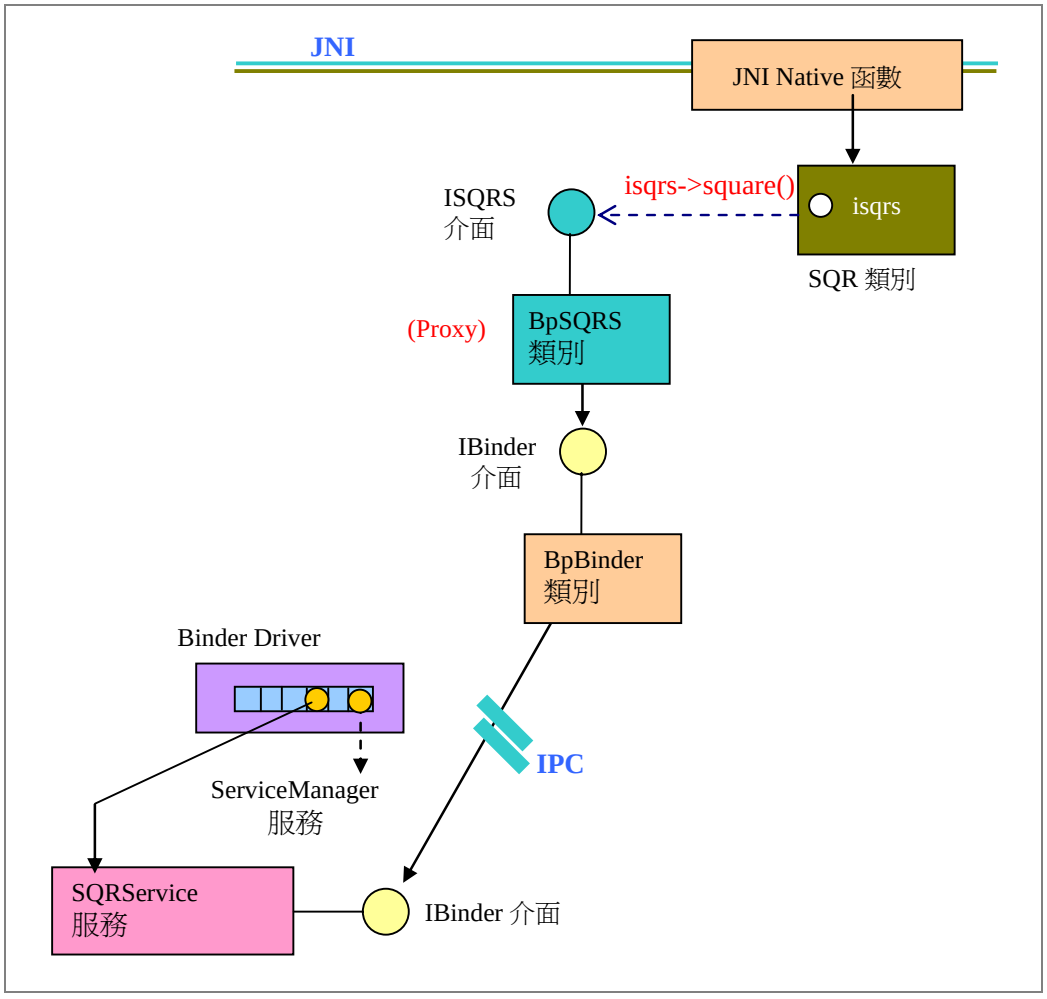
開發 Android C++層核心服務

第 1 步：撰寫核心服務，調用驅動程式、並加入 Binder Kernel

本範例裡，將在 BpBinder 外加一個 Proxy 類別：BpSQRS，來封裝 IBinder 介面，並且提供一個新的介面：ISQRS。如下圖：



撰寫核心服務

**** SQRService 類別之定義 ****

```
// SQRService.h
#include <stdint.h>
#include <sys/types.h>
#include <utils/Parcel.h>

#ifndef ANDROID_MISOO_SQRSERVICE_H
#define ANDROID_MISOO_SQRSERVICE_H
#include <utils.h>
```

```
#include <utils/KeyedVector.h>
#include <ui/SurfaceComposerClient.h>

namespace android {
class SQRService : public BBinder
{
public:
        static    int       instantiate();
        virtual status_t onTransact(uint32_t, const Parcel&, Parcel*, uint32_t);
                                        SQRService();
        virtual                         ~SQRService();
};
};
#endif
```

**** SQRService 類別之實作****

```
// SQRService.cpp
#include <utils/IServiceManager.h>
#include <utils/IPCThreadState.h>
#include <utils/RefBase.h>
#include <utils/IInterface.h>
#include <utils/Parcel.h>
#include "SQRService.h"

namespace android {
enum {
      SQUARE = IBinder::FIRST_CALL_TRANSACTION,
};

int SQRService::instantiate(){
      LOGE("SQRService instantiate");
      int r = defaultServiceManager()->addService(
                  String16("misoo.sqr"), new SQRService());
      LOGE("SQRService r = %d\n", r);
      return r;
}
SQRService::SQRService(){
      LOGV("SQRService created");
}
SQRService::~SQRService(){
      LOGV("SQRService destroyed");
}
status_t SQRService::onTransact(
      uint32_t code, const Parcel& data, Parcel* reply, uint32_t flags)
{
      switch(code) {
            case SQUARE: {
                  int num = data.readInt32();
                  reply->writeInt32(num * num);
```

```
                LOGE("onTransact::CREATE_NUM.. n=%d\n", num);
                return NO_ERROR;
            }
            break;
            default:
               LOGE("onTransact::default\n");
               return BBinder::onTransact(code, data, reply, flags);
        }
}
}; // namespace android
```

調用驅動程式

本範例沒有調用驅動程式。

產生*.so 共享程式庫

// Android.mk

```
LOCAL_PATH:= $(call my-dir)
include $(CLEAR_VARS)

LOCAL_SRC_FILES:= \
        SQRService.cpp
LOCAL_C_INCLUDES := \
     $(JNI_H_INCLUDE)

LOCAL_SHARED_LIBRARIES := \
        libutils
LOCAL_PRELINK_MODULE := false

LOCAL_MODULE := libSQRS01
include $(BUILD_SHARED_LIBRARY)
```

執行這 Android.mk 檔，就能產出 libSQRS01.so 共享程式庫。

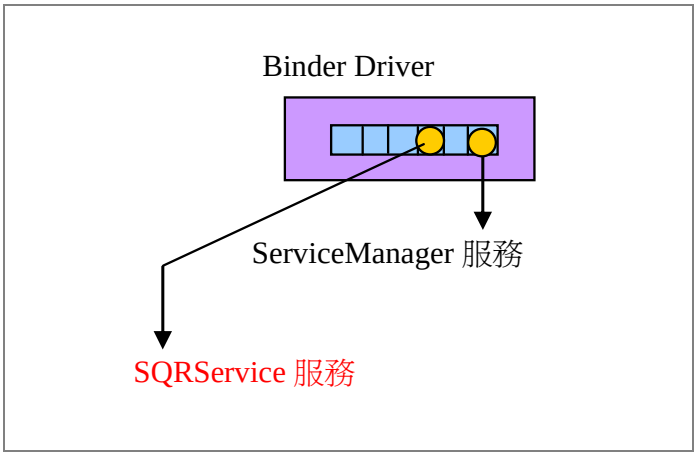
將服務加入 Binder Dirver

*** *addserver.cpp 程式****

```
// addserver.cpp
#include <sys/types.h>
#include <unistd.h>
#include <grp.h>

#include <utils/IPCThreadState.h>
#include <utils/ProcessState.h>
#include <utils/IServiceManager.h>
#include <utils/Log.h>
#include <private/android_filesystem_config.h>
```

```
#include "../libadd/SQRService.h"
using namespace android;

int main(int argc, char** argv){
    sp<ProcessState> proc(ProcessState::self());
    sp<IServiceManager> sm = defaultServiceManager();
    LOGI("ServiceManager: %p", sm.get());
    SQRService::instantiate();
    ProcessState::self()->startThreadPool();
    IPCThreadState::self()->joinThreadPool();
}
```

執行後，成功加入到 Binder Driver，如下：



撰寫 BpSQRS 類別

利用 BpInterface<T>樣板來實現 ISQRS 介面。例如，讓 BpSARS 來繼承 BpInterface<ISQRS>父類別。

**** BpSQRS 類別之定義 ****

```
// ISQRS.h
#include <utils/RefBase.h>
#include <utils/IInterface.h>
#include <utils/Parcel.h>
#ifndef ANDROID_MISOO_ISQRS_SERVICE_H
#define ANDROID_MISOO_ISQRS_SERVICE_H

namespace android {
class ISQRS: public IInterface{
public:
        DECLARE_META_INTERFACE(SQRS);
        virtual int square(const int& n) = 0;
};
//---------------------------------------------------------------
class BpSQRS: public BpInterface<ISQRS>{
public:
```

```
        BpSQRS(const sp<IBinder>& impl): BpInterface<ISQRS>(impl){}
        virtual int square(const int& n);
};
}; // namespace android
#endif
```

*** *BpSQRS 類別之實作****

```
// ISQRS.cpp
#include "ISQRS.h"
namespace android {
enum {
        SQUARE = IBinder::FIRST_CALL_TRANSACTION,
};
int BpSQRS::square(const int& n) {
                Parcel data, reply;
                data.writeInt32(n);
                   LOGE("BpSQRService::create remote()->transact()\n");
                remote()->transact(SQUARE, data, &reply);
                   LOGE("BpSQRService::create n=%d\n", n);
                int num = reply.readInt32();
                   LOGV("tom's num = %d", num);
                return num;
        }
IMPLEMENT_META_INTERFACE(SQRS, "android.misoo.IAS");
}; // namespace android
```

從 SQR 類別調用到 BpSQRS 的 ISQRS 介面，再調用 BpBinder 的 IBinder 介面。

第 2 步：撰寫 JNI Native 程式來使用核心服務、確保核心服務及 Native 程式的線程安全

撰寫 SQR 類別來使用核心服務

SQR 類別之定義

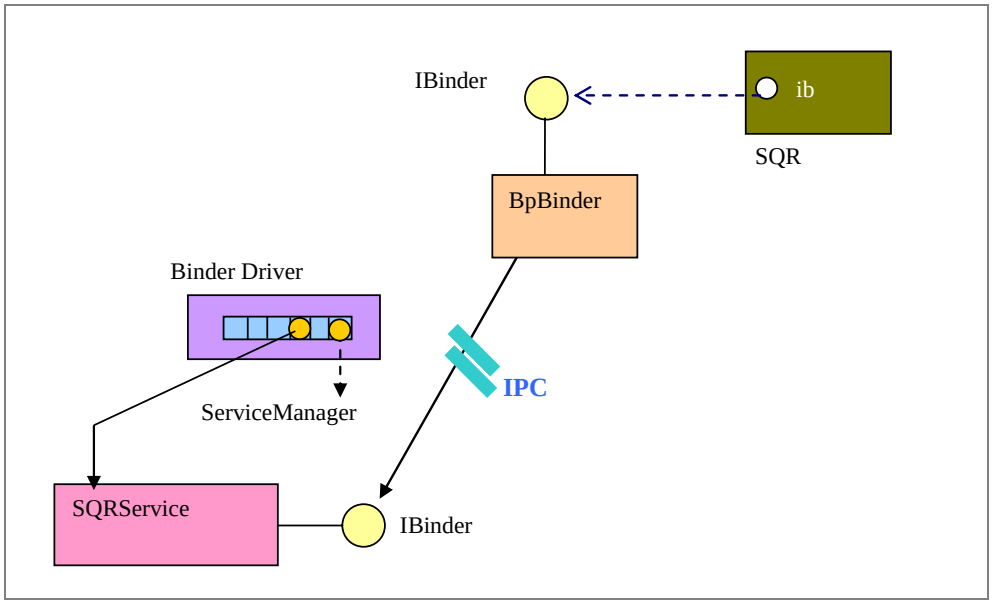
```
//SQR.h
#ifndef ANDROID_MISOO_SQR_H
#define ANDROID_MISOO_SQR_H
#include "../core_service/ISQRS.h"
```

```
namespace android {
class SQR{
public:
     int    execute(int n);
private:
        static const sp<ISQRS>& getSQRService();
        static    sp<ISQRS>         sSQRService;
};
}; // namespace android
#endif
```

**** SQR 類別之實作****

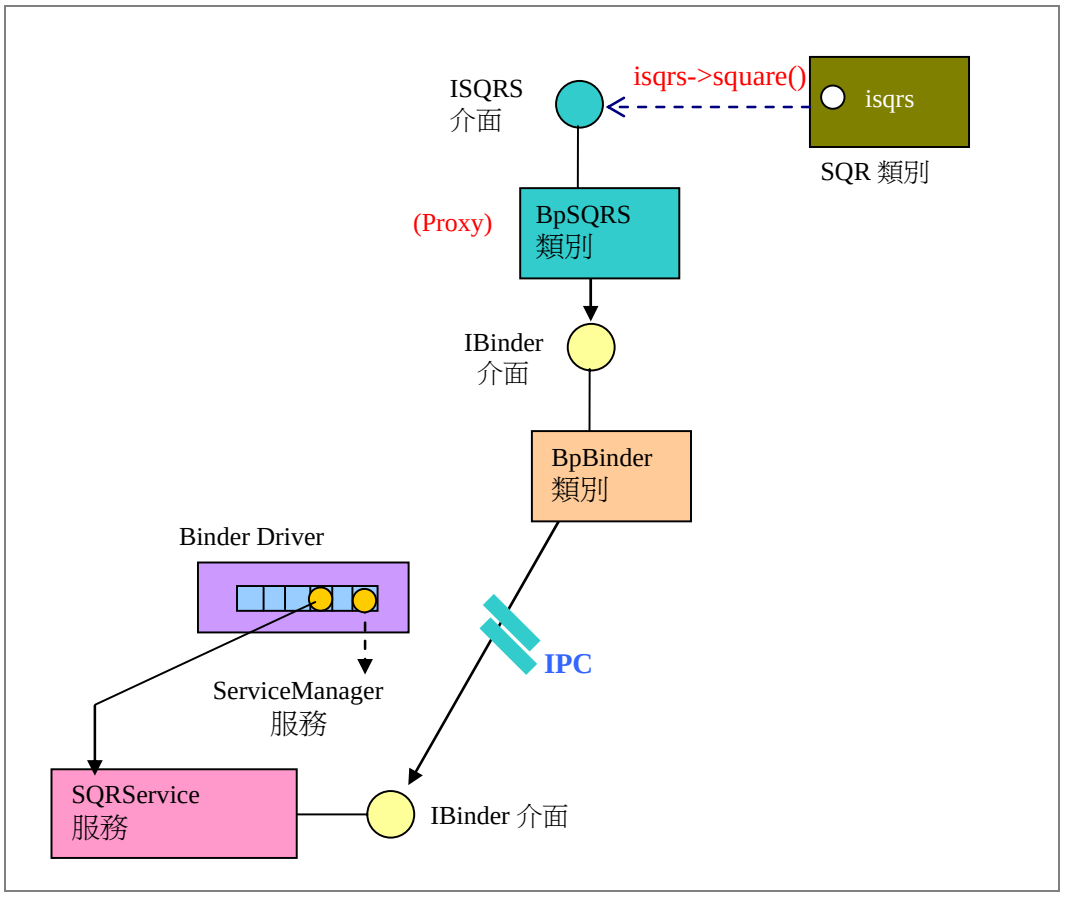
```
//SQR.cpp
#include <utils/IServiceManager.h>
#include <utils/IPCThreadState.h>
#include "SQR.h"

namespace android {
const sp<ISQRS>& SQR::getSQRService(){
          sp<IServiceManager> sm = defaultServiceManager();
          sp<IBinder> ib = sm->getService(String16("misoo.sqr"));
               LOGE("SQR::getSQRService\n");
          sp<ISQRS> sService = interface_cast<ISQRS>(ib);
          return sService;
}
int SQR::execute(int n){
       int k =0;
       const sp<ISQRS>& isqrs(getSQRService());
       if (isqrs != 0)
             k = isqrs->square(n);
       return k;
}}; // namespace android
```

- execute()函數使用ServiceManager的函數：getSQRService()來獲得服務。
- 取得服務時 ServiceManager 傳回 BpBinder 的 IBinder 介面。



轉換出 ISQRS 新介面

接著，使用樣板 interface_cast<T>來轉換出 ISQRS 新介面。如下圖：



定義撰寫 JNI Native 程式來使用核心服務

**** JNI Native函數之定義****

```
/* com_misoo_service_sqr01.h
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class com_misoo_service_sqr01 */

#ifndef _Included_com_misoo_service_sqr01
#define _Included_com_misoo_service_sqr01
#ifdef __cplusplus

extern "C" {
#endif
/*
  * Class:          com_misoo_service_sqr01
  * Method:        execute
  * Signature: (II)I
  */
JNIEXPORT jint JNICALL Java_com_misoo_service_sqr01_execute
    (JNIEnv*, jobject, jint);

#ifdef __cplusplus
}
#endif
#endif
```

**** JNI Native函數之實作****

```
// com_misoo_service_sqr01.cpp
#include <utils/Log.h>
#include <utils/IPCThreadState.h>
#include <utils/ProcessState.h>
#include "com_misoo_service_sqr01.h"
#include "SQR.h"

using namespace android;
JNIEXPORT jint JNICALL Java_com_misoo_service_sqr01_execute(JNIEnv *env,
jobject thiz, jint x)
{
        SQR* sqrObj = new SQR();
        LOGE("sqr1Obj = %p\n", sqrObj);
        int k = sqrObj->execute(x);
        return k;
}
```

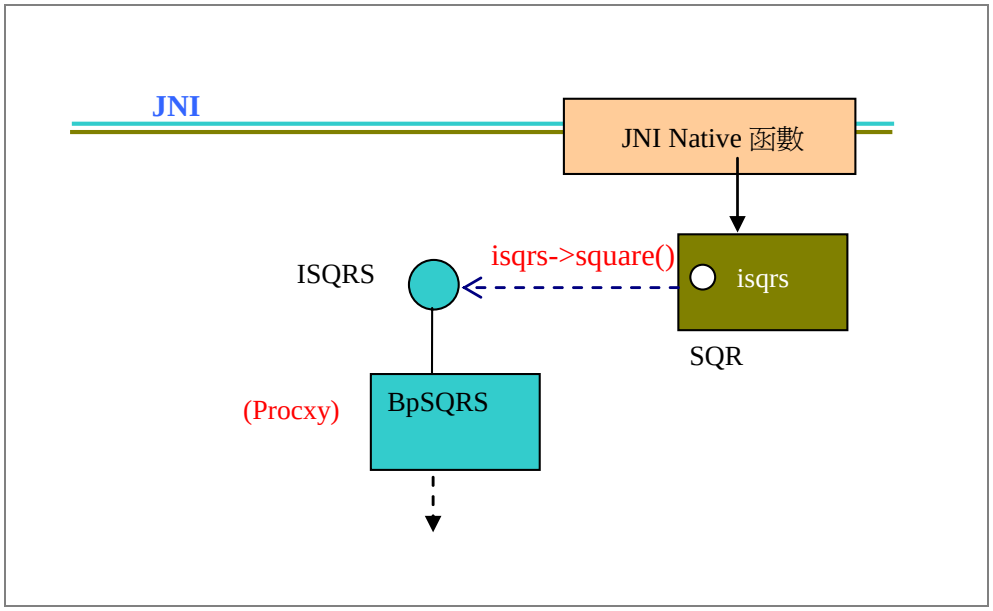
**** makefile 檔案****

// Android.mk

```
LOCAL_PATH:= $(call my-dir)
include $(CLEAR_VARS)

LOCAL_SRC_FILES:= \
```

```
        com_misoo_service_sqr01.cpp \
        SQR.cpp \
        ISQRS.cpp

base := $(LOCAL_PATH)/../../..
LOCAL_C_INCLUDES := \
        $(JNI_H_INCLUDE) \
        $(base)/CS01

LOCAL_SHARED_LIBRARIES := \
        libutils \
        libSQRS01

LOCAL_PRELINK_MODULE := false

LOCAL_MODULE := libSQRS01_jni

include $(BUILD_SHARED_LIBRARY)
```

執行此 Android.mk 檔，產出 libSQRS01_jni.so 共享程式庫，就可以載入手機或模擬器裡執行了。執行時，Native 函數調用 SQR，轉而調用 BpSQRS，如下圖：



確保核心服務及 Native 程式的線程安全

如果會有多個線程可能同時來執行同一個 Native 函數或 C++函數時，請依據上一節(Section-11)所介紹的線程安全機制，修改 Native 程式和核心服務程式碼。

第 3 步：擴充應用框架--即撰寫 Java 抽象類別、調用 JNI Native 程式

撰寫 Native 的對應 Java 類別

```
//sqr01.java
package com.misoo.service;
public class sqr01
{
	static
		{ System.loadLibrary("SQRS01_jni"); }
	public static native int execute(int x);
}
```

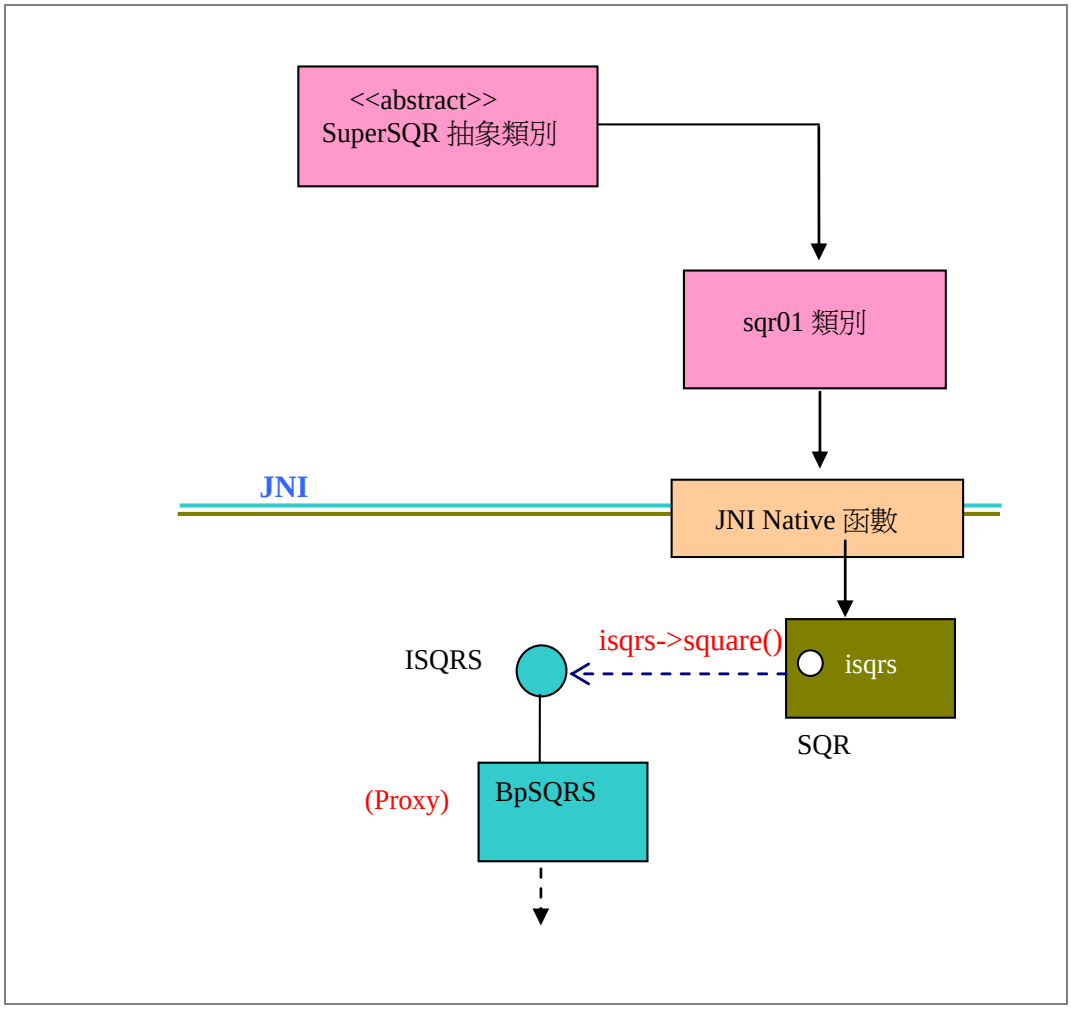
即撰寫 Java 抽象類別、調用 JNI Native 程式

Android 框架裡已經有許多種抽象類別了，例如 Activity、Serviced 等都是抽象類別。當然，我們也能將 SuperSQR 抽象類別加到 Android 框架裡，等於擴充了框架的內涵。SQR 抽象類別的程式碼如下：

*** *Java抽象類別：SuperSQR* ***

```
//SuperSQR.java
package com.misoo.service;
abstract public class SuperSQR {
		public int exec(){
			return sqr01.execute(onGetX());
		  }
		abstract public int onGetX();
}
```

這抽象類別調用 Native 函數，如下圖：



第 4 步：撰寫應用範例--撰寫 Java 子類別、繼承抽象類別

雖然 Java 子類別經常不是核心服務開發者的主要工作，但是核心服務開發者通常需要撰寫應用範例(Sample)來示範核心服務的功能和特性。例如，從 SuperSQR 抽象類別衍生出子類別如下：

撰寫 SuperSQR 的子類別

****mySQR類別之定義****

```
//mySQR.java
package com.misoo.pk01;
import com.misoo.service.SuperSQR;
public class mySQR extends SuperSQR {
	@Override
	public int onGetX() {
		return 60;
	}
```

```
}
```

接著，撰寫 Activity 的子類別。

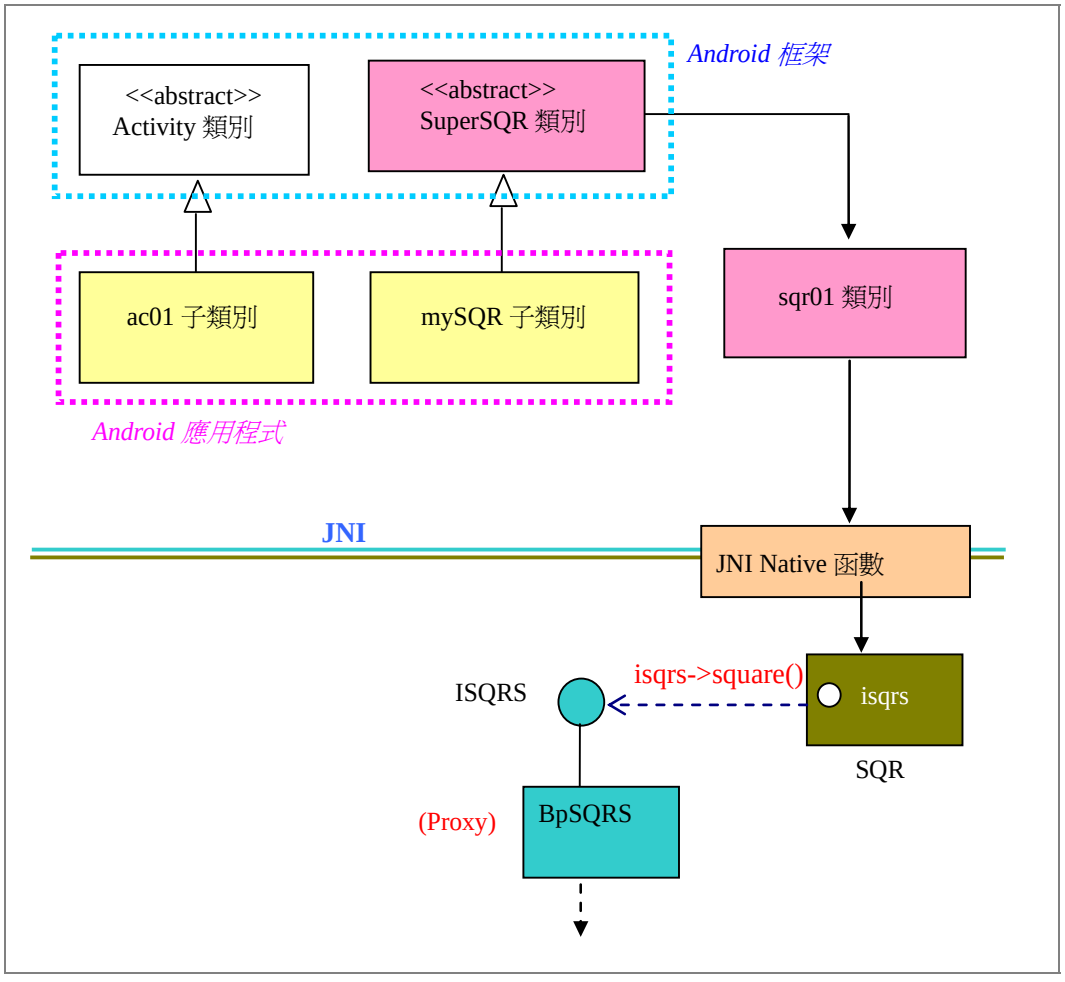
撰寫 Activity 的子類別

****ac01子類別之定義****

```
// ac01.java
package com.misoo.pk01;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.LinearLayout;

public class ac01 extends Activity implements OnClickListener {
      /** Called when the activity is first created. */
        private Button btn, btn2;
        @Override
      public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);

            LinearLayout layout = new LinearLayout(this);
            layout.setOrientation(LinearLayout.VERTICAL);
            btn = new Button(this);
            btn.setBackgroundResource(R.drawable.heart);
            btn.setId(101);
            btn.setText("run");
            btn.setOnClickListener(this);
            LinearLayout.LayoutParams param =
                  new LinearLayout.LayoutParams(120, 55);
            param.topMargin = 10;
            layout.addView(btn, param);

            btn2 = new Button(this);
            btn2.setBackgroundResource(R.drawable.ok);
            btn2.setId(102);
            btn2.setText("exit");
            btn2.setOnClickListener(this);
            layout.addView(btn2, param);
            setContentView(layout);
         }
 public void onClick(View v) {
    switch(v.getId()){
       case 101:
              mySQR sqr = new mySQR();
              int k = sqr.exec();
              setTitle("Value = " + String.valueOf(k));
```

```
                break;
        case 102:
                finish();
                break;
        }
  }
}
```

於是，總共開發了兩個 Java 層的子類別。如下圖所示：



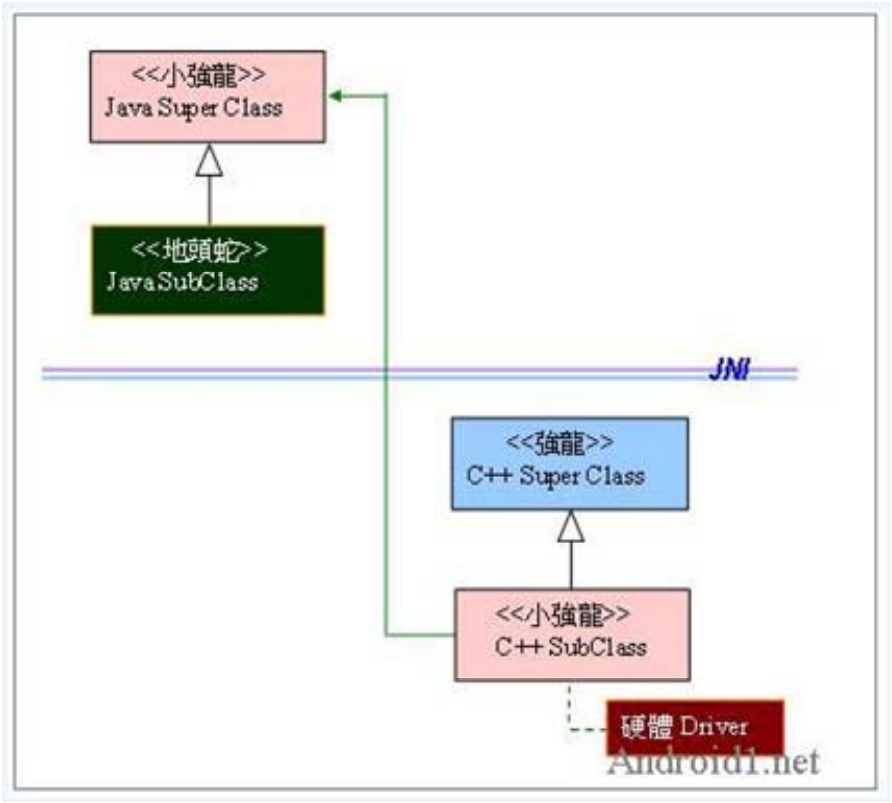
結論

從上述過程中，可以看到：

- C++層的 SQRService 繼承 BBinder。
- C++層的 BpSQRS 繼承 BpInterface<T>。
- Java 層的 mySQR 繼承 SuperSQR。

其中，核心服務開發者的主要工作是：撰寫 SQRService、BpSQRS 和 SuperSQR。這種角色就稱爲「小強龍」。而撰寫 BBinder、BpInterface<T>者，稱爲「大強龍」。

而撰寫 mySQR、Activity 者，稱爲「地頭蛇」。三種角色的智慧銜接如下圖：



核心服務的 Stub 類別

回顧 Proxy 類別

在 Java 層裡，使用 aidl.exe 工具來建立 Proxy-Stub 機制來封裝 IBinder 介面。在 C++層也有 IBinder 介面，所以也需要 Proxy-Stub 機制。再上一小節裡，已經使用 BpInterface<T>樣板來定義出 C++核心服務的 Proxy 類別了。例如：

*** *Proxy 類別之定義* ***

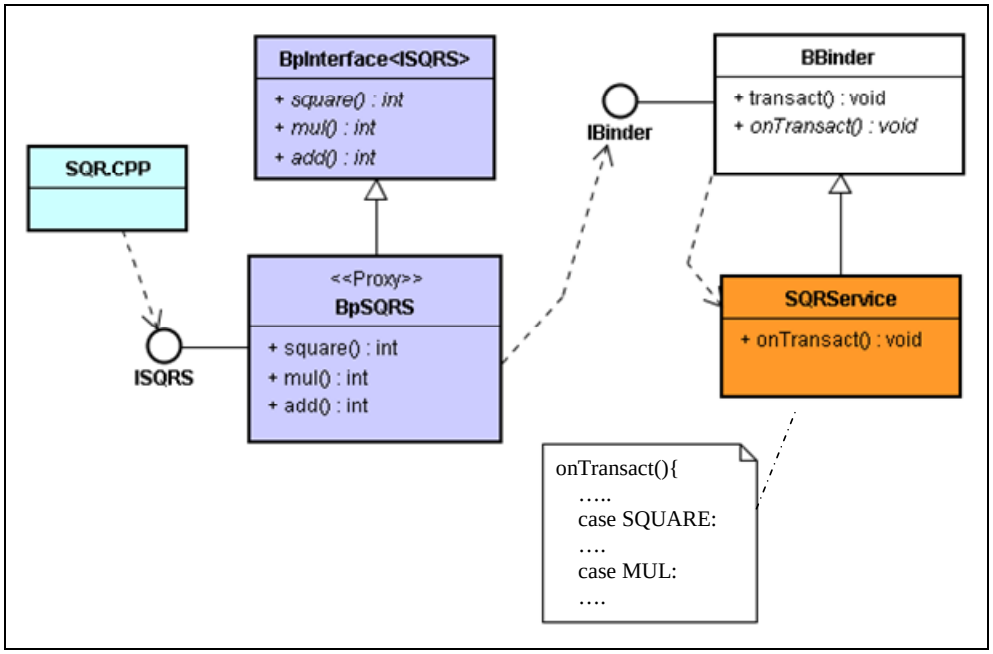
```
// ISQRS.h
#include <utils/RefBase.h>
#include <utils/IInterface.h>
#include <utils/Parcel.h>
#ifndef ANDROID_MISOO_ISQRS_SERVICE_H
#define ANDROID_MISOO_ISQRS_SERVICE_H

namespace android {
class ISQRS: public IInterface{
public:
        DECLARE_META_INTERFACE(SQRS);
        virtual int square(const int& n) = 0;
        virtual int mul(const int& n, const int& m ) = 0;
        virtual int add(const int& x, const int& y) = 0;
};
```

```
//-------------------------------------------------------------
class BpSQRS: public BpInterface<ISQRS>{
public:
	BpSQRS(const sp<IBinder>& impl): BpInterface<ISQRS>(impl){}
	virtual int square(const int& n);
	virtual int mul(const int& n, const int& m);
	virtual int add(const int& x, const int& y);
};
};
#endif
```

**** Proxy 類別之實作****

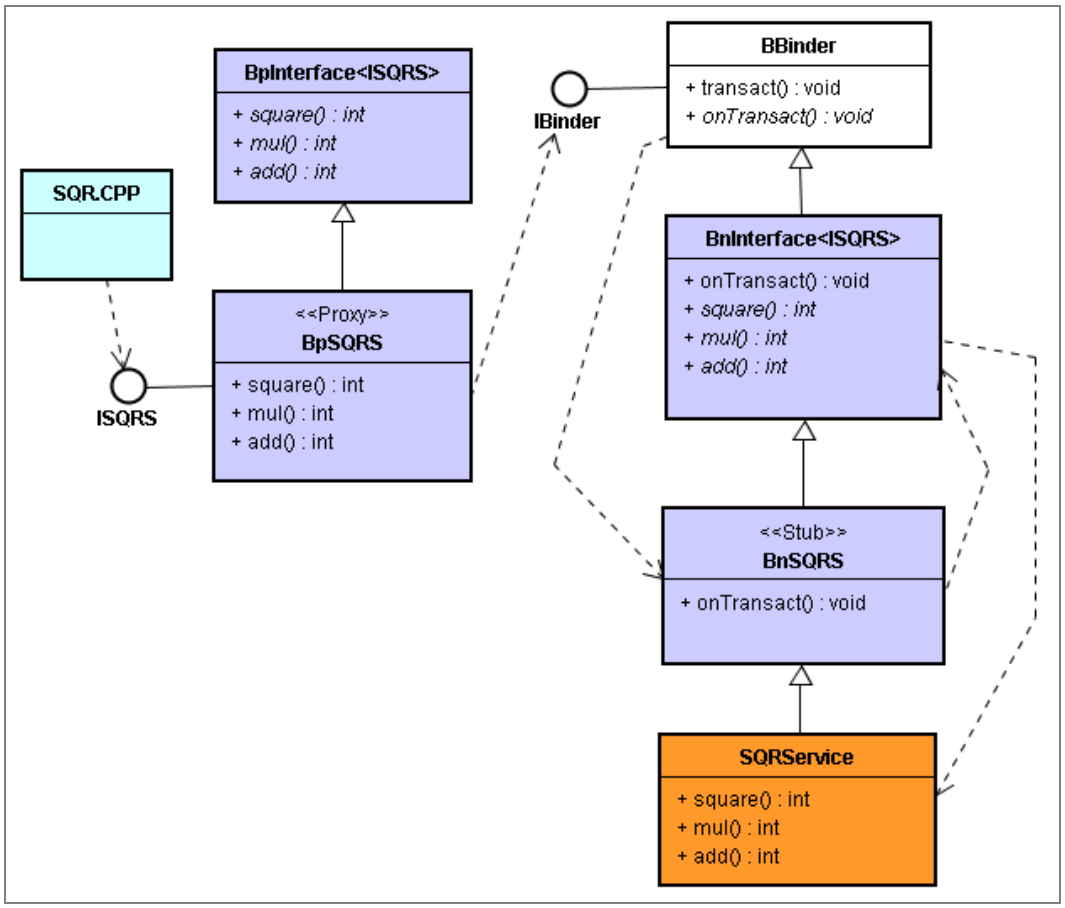
```
// ISQRS.cpp
#include "ISQRS.h"
namespace android {
enum {
	SQUARE = IBinder::FIRST_CALL_TRANSACTION,
	MUL = IBinder::FIRST_CALL_TRANSACTION + 1,
	ADD = IBinder::FIRST_CALL_TRANSACTION + 2,
};
int BpSQRS::square(const int& n) {
		Parcel data, reply;
		data.writeInt32(n);
		remote()->transact(SQUARE, data, &reply);
		int num = reply.readInt32();
		return num;
	}
int BpSQRS::mul(const int& n, const int& m) {
		Parcel data, reply;
		data.writeInt32(n);	data.writeInt32(m);
		remote()->transact(MUL, data, &reply);
		int num = reply.readInt32();
		return num;
	}
int BpSQRS::add(const int& x, const int& y) {
		Parcel data, reply;
		data.writeInt32(x);	data.writeInt32(y);
		remote()->transact(MUL, data, &reply);
		int num = reply.readInt32();
		return num;
	}
IMPLEMENT_META_INTERFACE(SQRS, "android.misoo.IAS");
};
```

有了 Proxy 類別之後，Client 端的 SQR.CPP 就能透過 ISQRS 介面而間接調用 IBinder 的 transact()函數。如下圖所示：



加上 Stub 類別

於此，將加上核心服務的 Stub 類別：BnSQRS，如下圖：



這就 C++層核心服務的 Proxy-Stub 機制。

定義 Stub 類別

*** *Stub 類別之定義* ***

```
// ISQRSStub.h
#ifndef ANDROID_MISOO_ISQRSStub_H
#define ANDROID_MISOO_ISQRSStub_H
#include <utils/RefBase.h>
#include <utils/IInterface.h>
#include <utils/Parcel.h>
#include "ISQRS.h"

namespace android {

class BnSQRS: public BnInterface<ISQRS>{
public:
        virtual status_t        onTransact( uint32_t code,
                                            const Parcel& data,
                                            Parcel* reply,
                                            uint32_t flags = 0);
};
};
#endif
```

*** *Stub 類別之實作****

```
// ISQRSSTub.cpp
#include <stdint.h>
#include <sys/types.h>
#include <utils/Parcel.h>
#include "ISQRSStub.h"

namespace android {
enum {
        SQUARE = IBinder::FIRST_CALL_TRANSACTION,
        MUL = IBinder::FIRST_CALL_TRANSACTION + 1,
        ADD = IBinder::FIRST_CALL_TRANSACTION + 2,
};

//---------------------------------------------------------------
status_t BnSQRS::onTransact(
        uint32_t code, const Parcel& data, Parcel* reply, uint32_t flags)
{
          switch(code) {
             case SQUARE:
                        {
                          int num = data.readInt32();
                          int k = square(num);
                          reply->writeInt32(k);
                          return NO_ERROR;
```

```
                        }
                    break;
            case MUL:
                        {
                         int num1 = data.readInt32();
                         int num2 = data.readInt32();
                         int k = mul(num1, unm2);
                         reply->writeInt32(k);
                         return NO_ERROR;
                         }
                    break;
             case ADD:
                        {
                         int num = data.readInt32();
                         int num2 = data.readInt32();
                         int k = add(num1, num2);
                         reply->writeInt32(k);
                         return NO_ERROR;
                         }
                    break;
              default:
                    return BBinder::onTransact(code, data, reply, flags);
              }
}
}; // namespace android
```

有了 Stub 類別，核心服務的開發者就不必關心 onTransact()函數了。這會讓核心服務內容更加親切。此 SQRService 類別的內容如下所示：

簡化了核心服務

**** SQRService 類別之定義 ****

```
#ifndef ANDROID_MISOO_SQR_SERVICE_H
#define ANDROID_MISOO_SQR_SERVICE_H

#include <utils.h>
#include <utils/KeyedVector.h>
#include <ui/SurfaceComposerClient.h>
#include "ISQRSStub.h"

namespace android {

class SQRService : public BnSQRS
{
public:
        static    int       instantiate();
        virtual int       square(const int& n);
        virtual int       mul(const int& n, const int& m);
        virtual int       add(const int& x, const int& y);
```

```
  };
}; // namespace android
#endif
```

**** SQRService 類別之實作****

```
// SQRService.cpp
#include <utils/IServiceManager.h>
#include <utils/IPCThreadState.h>
#include <utils/RefBase.h>
#include <utils/IInterface.h>
#include <utils/Parcel.h>
#include "SQRService.h"

namespace android {

int SQRService::instantiate()
{
        int r = defaultServiceManager()->addService(
                String16("guilh.add"), new SQRService());
        return r;
}
int SQRService::square(const int& n)
        {
           int k = n * n;
           return k;
        }
int SQRService::mul(const int& n, const int& m)
        {
           int k = n * m;
           return k;
        }
int SQRService::add(const int& x, const int& y)
        {
           int k = x + y;
           return k;
        }
}; // namespace android
```

結論

Proxy-Stub 機制將 IPC 包裝起來。Android 的 AIDL.exe 工具可以協助建立 Java 層的 Proxy-Stub 機制。同樣地，在 C++層則可以使用 BpInterface<T>和 BnInterface<T>樣板來協助建立 Proxy-Stub 機制。

~~ END ~~