**Binder服务应用实践**

1. **AIDL接口**

这个文档用于记录一个Binder服务的编写过程，用于Linux环境中用到了Android Binder的状况。所有语言实现用C++实现。

Demo服务，起于一个AIDL接口文档，我们定义的接口如下：

//IHelloService.aidl

/\*

Cy AIDL TEST

\*/

package fce.aidl.test;

import fce.aidl.test.IHelloCallback;

interface IHelloService {

void hello(int param);

void registerCb(IHelloCallback cb);

}

//IHelloCallback.aidl

/\*

Just for test

\*/

package fce.aidl.test;

interface IHelloCallback {

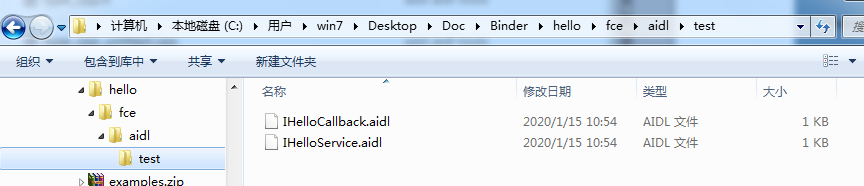
void onData(int data);

}

比较简单的一个例子，只有一个hello接口和回调注册registerCb，其他复杂的还可以定义数据，需要的可以参考aidl-cpp工具里的test例子。

1. **生成cpp接口文件**

我们需要创建一个目录，然后在里面按照包名顺序创建好子目录，将aidl文件放到对应的目录位置，本文例子见下图：



然后在\hello目录下执行命令：

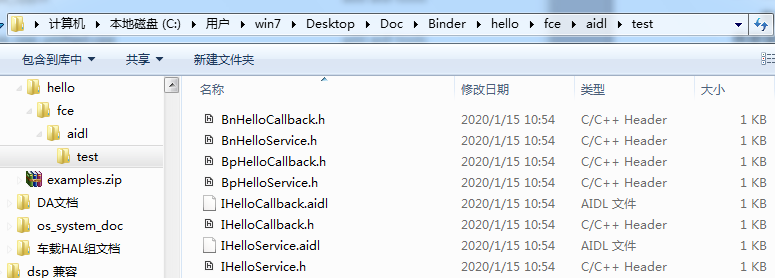
aidl-cpp -I./ ./fce/aidl/test/IHelloService.aidl ./

./helloservice.cpp

aidl-cpp -I./ ./fce/aidl/test/IHelloCallback.aidl ./

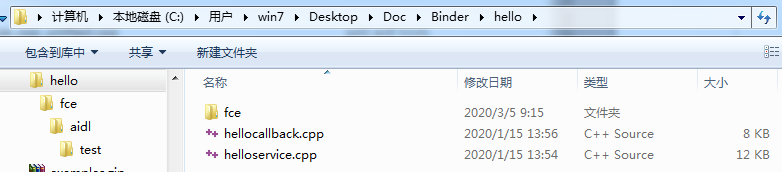
./hellocallback.cpp

则对应的目录下将生成下图的接口文件：



接口文件

以及实现文件：



实现文件

这里的实现文档只是生成了对应的Bn，Bp类。其中，在BnHelloService上，我们需要实现具体的服务端程序，为此，自己新建一个HelloImp类，以下为添加的两个文件：

//HelloImp.h

#ifndef \_\_HELLO\_IMP\_H

#define \_\_HELLO\_IMP\_H

using namespace android;

#include "BnHelloService.h"

#include "IHelloCallback.h"

using namespace fce::aidl::test;

using android::binder::Status;

namespace hello

{

class HelloImp :public BnHelloService

{

public:

HelloImp();

Status hello(int param);

Status registerCb(const sp<IHelloCallback>& cb);

private:

sp<IHelloCallback> callback;

};

}

#endif //#ifndef \_\_HELLO\_IMP\_H

//HelloImp.cpp

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <binder/IServiceManager.h>

#include "helloImp.h"

using namespace hello;

HelloImp::HelloImp()

{

}

Status HelloImp::hello(int param)

{

printf("hello ~~~ %d \n",param);

if (callback!=NULL)

callback->onData(param + 10);

return Status::ok();

}

Status HelloImp::registerCb(const sp<IHelloCallback>& cb)

{

callback = cb;

return Status::ok();

}

这样，关于Binder的功能实现部分，就完成了。

1. **编写service和client**

为了让Binder服务运行起来，需要自己编写一个service程序，将服务注册到系统里。还需要编一个client端，用来测试服务是否正常。

为此，添加service.cpp和client.cpp两个文件，如下：

//service.cpp

#include <binder/IPCThreadState.h>

#include <binder/ProcessState.h>

#include <binder/IServiceManager.h>

#include "helloImp.h"

using namespace hello;

int main(int argc, char \*argv[])

{

printf("just for test\n");

sp<IServiceManager> sm = defaultServiceManager();

HelloImp \* sp = new HelloImp();

sm->addService(String16("fce.hello"),sp);

//Start Service

ProcessState::self()->startThreadPool();

//Loop for waiting for request

IPCThreadState::self()->joinThreadPool();

return 0;

}

//client.cpp

#include <binder/IPCThreadState.h>

#include <binder/ProcessState.h>

#include <binder/IServiceManager.h>

using namespace android;

#include "IHelloService.h"

#include "BnHelloCallback.h"

using namespace fce::aidl::test;

using android::binder::Status;

namespace hello

{

class HelloCallback : public BnHelloCallback

{

public:

Status onData(int data);

};

Status HelloCallback::onData(int data)

{

printf("recive callback st = %d\n",data);

return Status::ok();

}

}

using namespace hello;

int main(int argc, char \*argv[])

{

sp<IServiceManager> sm = defaultServiceManager();

sp<IBinder> binder;

sp<IHelloService> helloservice;

binder = sm->getService(String16("fce.hello"));

helloservice = interface\_cast<IHelloService>(binder);

if( helloservice.get() == NULL )

{

printf("[%s:%d] IHelloService proxy object is still NULL!\n", \_\_FUNCTION\_\_, \_\_LINE\_\_);

return -1;

}

sp<IHelloCallback> cb = new HelloCallback();

//Start Service

ProcessState::self()->startThreadPool();

helloservice->registerCb(cb);

int count = 0;

while(true)

{

printf("say hello to service %d ~ \n",count);

helloservice->hello(count++);

sleep(2);

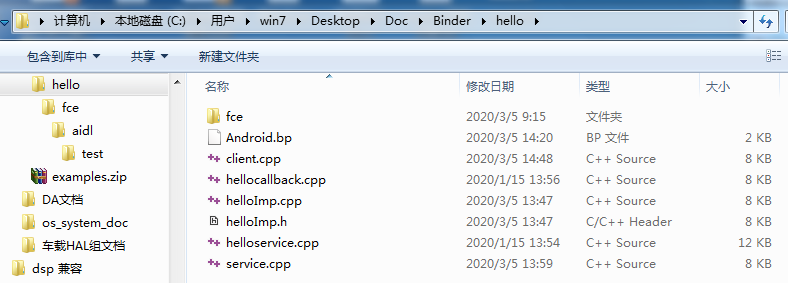
}

return 0;

}

1. **在Android环境编译**

将文件夹复制到AndroidP的工作目录里，再编写一个Android.bp的编译文件。最后的文件截图如下：



其中，Android.bp文件内容如下：

//Android.bp

cc\_library\_static {

name: "demo-static",

vendor: true,

// relative\_install\_path: "hw",

cflags: [

"-Wall",

"-Wextra",

"-Werror",

"-Wno-unused-parameter",

],

srcs: [

"hellocallback.cpp",

"helloservice.cpp",

"helloImp.cpp",

],

shared\_libs: [

"libbinder",

"liblog",

"libutils",

],

include\_dirs: [

"vendor/FCE/demo",

"vendor/FCE/demo/fce/aidl/test",

"frameworks/native/libs/binder/include/binder"

],

export\_include\_dirs: ["."],

}

cc\_binary {

name: "helloService",

vendor: true,

srcs: ["service.cpp"],

include\_dirs: ["vendor/FCE/demo","vendor/FCE/demo/fce/aidl/test","frameworks/native/libs/binder/include/binder"],

shared\_libs: [

"liblog",

"libbinder",

"libcutils",

"libutils",

],

static\_libs: [

"demo-static",

],

}

cc\_binary {

name: "helloClient",

vendor: true,

srcs: ["client.cpp"],

include\_dirs: ["vendor/FCE/demo","vendor/FCE/demo/fce/aidl/test","frameworks/native/libs/binder/include/binder"],

shared\_libs: [

"liblog",

"libbinder",

"libcutils",

"libutils",

],

static\_libs: [

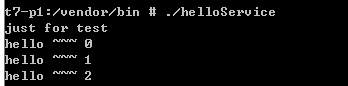
"demo-static",

],

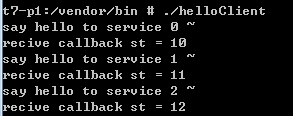
}

其中，将生成的Bp，Bn类和服务实现编译成一个静态库，再将service和client各自编译成执行文件。编译完成后，会生成helloService和helloClient执行文件，将两个文件push到安卓机器里运行测试，结果如下：

先执行helloService:



再运行helloClient：



可以看到，运行的结果是符合预期的。

1. **源码分析**

**5.1 service端**

//service.cpp

sp<IServiceManager> sm = defaultServiceManager();

HelloImp \* sp = new HelloImp();

sm->addService(String16("fce.hello"),sp);

这里就是生成一个服务实现，然后由ServiceManager注册到服务列表里面。

其中的关键内容在IServiceManager的addService函数里：

//android\frameworks\native\libs\binder\IServiceManager.cpp

virtual status\_t addService(const String16& name, const sp<IBinder>& service,

bool allowIsolated, int dumpsysPriority) {

Parcel data, reply;

data.writeInterfaceToken(IServiceManager::getInterfaceDescriptor());

data.writeString16(name);

data.writeStrongBinder(service);

data.writeInt32(allowIsolated ? 1 : 0);

data.writeInt32(dumpsysPriority);

status\_t err = remote()->transact(ADD\_SERVICE\_TRANSACTION, data, &reply);

return err == NO\_ERROR ? reply.readExceptionCode() : err;

}

可以看到，这里调用了ServiceManager的ADD\_SERVICE\_TRANSACTION功能，在ServiceManager的实现里，Binder驱动会创建这个binder对象（HelloService）的引用handle，并保存到ServiceManager自己的handle链表里。然后服务端就可以等待客户端的调用了。

这里说明一下remote()->transact这个调用，这里实际是调用BpBinder.cpp里的transact函数：

//android\frameworks\native\libs\binder\BpBinder.cpp

status\_t BpBinder::transact(

uint32\_t code, const Parcel& data, Parcel\* reply, uint32\_t flags)

{

// Once a binder has died, it will never come back to life.

if (mAlive) {

status\_t status = IPCThreadState::self()->transact(

mHandle, code, data, reply, flags);

if (status == DEAD\_OBJECT) mAlive = 0;

return status;

}

return DEAD\_OBJECT;

}

可以看到这里最终就是靠mHandle来传给Binder驱动，最终驱动靠handle号来找到对应的服务本身，调用其onTransact接口，再根据code来决定执行哪一个函数命令，code先匹配自定义的那些，不符合最后会调用BBinder::onTransact。

**5.2 client端**

//client.cpp

sp<IServiceManager> sm = defaultServiceManager();

sp<IBinder> binder;

sp<IHelloService> helloservice;

binder = sm->getService(String16("fce.hello"));

helloservice = interface\_cast<IHelloService>(binder);

可以看到，通过ServiceManager的getService接口，传服务的名字"fce.hello"，可以获取到HelloService的引用handle。可以参照android\frameworks\native\libs\binder\include\binder\IServiceManager.cpp的getService函数实现，实际最后调用的是：

//android\frameworks\native\libs\binder\IServiceManager.cpp

virtual sp<IBinder> checkService( const String16& name) const

{

Parcel data, reply;

data.writeInterfaceToken(IServiceManager::getInterfaceDescriptor());

data.writeString16(name);

remote()->transact(CHECK\_SERVICE\_TRANSACTION, data, &reply);

return reply.readStrongBinder();

}

这里获取到的binder，会传入interface\_cast做参数，这里其实是生成了一个BpHelloService对象，可以看下：

//android\frameworks\native\libs\binder\include\binder\IInterface.h

template<typename INTERFACE>

inline sp<INTERFACE> interface\_cast(const sp<IBinder>& obj)

{

return INTERFACE::asInterface(obj);

}

其中asInterface做的事情是IMPLEMENT\_META\_INTERFACE(INTERFACE, NAME)这个宏定义里定义了具体的函数，实际就是new 一个BpXXXXService。详细可以看看IInterface.h的具体写法。

1. **在Linux系统的实践**

在Linux车机平台方案里，有采用Android Binder作为RPC实现的方式，实践证明，将Android Binder移植到Linux系统中单独运行，是可行且有效的。现在Sunplus的车机方案里就移植了Android Binder，为了测试上述的demo，我也将代码放到了该系统上编译测试，结果也是可以的。

在这个过程中，看到有几个问题需要注意：

1. 要在Linux系统上移植Binder，需要BSP这里移植Binder的驱动，并挂载驱动；还有就是需要编译Binder库，供binder应用编译链接使用。
2. Android Binder虽然稳定，但是随着Android版本的更新，它本身也有一些改动。Sunplus移植的Binder版本应该比较低，所以与新版的Binder有些接口差异，这样aidl-cpp工具生成的文件，不能直接在上面编译通过，需要做一些小修改：

a、缺少Errors.h，Status.h，可以将Android代码的加入替一下；

b、readStrongBinder代码有差异，这个应该要改下aidl的代码：

sunplus Parcel: sp<IBinder> readStrongBinder() const;

androidP Parcel:status\_t readStrongBinder(sp<T>\* val) const;

c、其他编不过的按错误调整一下。

总结就是如果是自己移植Binder，为减少不必要的麻烦，可以直接用AndroidP的Binder来移植。

1. **备用**

aidl-cpp的例子里，展示了很多参数的写法，基本看它的例子就可以知道别的参数类型怎么实现了。

之前是在GitHub找的aidl-cpp工具，其实在Android的源码里就带着这个工具，具体目录是android/system/tools/aidl。对应hidl也有工具，具体目录是android/system/tools/hidl。

android parcel传输复杂数据结构:

先传送结构体大小，再传数据。注意结构体内存对齐。

/\*send\*/

Parcel data, reply;

char \*p = (char \*)(&obj)

data.writeInt(sizeof(obj));

for (int i = 0; i < sizeof(obj);++i) {

   data.writeByte(p[i]);

}

/\*recv\*/

Parcel data, reply;

int size = data.readInt();

char \*buf = (char \*)(&obj);

for (int i = 0; i < size; ++i) {

   buf[i] = data.readByte();

}