# TODO

由于我们的串口设备多，完全可以用java来实现一套kjilink干的事情

我们完全可以封装适配框架层，权限控制

通讯应该按照网络分层来设计

# 串口服务的基本框架

fw几乎没有什么封装，其实也就是一个文件读写而已，没有必要封装吧

这里将了解Android下的串口系统框架，仍然使用上一篇博客中的tiny\_serial.c作为本例的驱动，本例实现的功能不变，无论应用程序写任何数据到串口，都能从该串口中读回。  
Android串口(serial port)服务框架。

(A) 串口服务的基本框架  
    1. 注册驱动  
    2. 注册hal  
    3. 注册JNI  
    4. 注册和添加SerialService  
    5. 串口管理SerialManager  
  
(B) 打开串口  
(C) 设置权限  
(D) 测试串口

## 1. 注册驱动

    通过加载kernel/driver/tty/serial/tiny\_serial.c驱动，生成/dev/ttytiny0节点，应用通过访问该节点，实现的对串口的open、read、write、close等操作。

## 注册hal

在Android串口服务中，省略了hal层，即通过JNI直接访问驱动。

## 3. 注册JNI

  通过frameworks/base/services/core/jni/com\_android\_server\_SerialService.cpp和  
frameworks/base/core/jni/android\_hardware\_SerialPort.cpp两个JNI文件对驱动访问，  
为JAVA提供底层驱动访问的接口，如：

private native ParcelFileDescriptor native\_open(String path);

private native void native\_open(FileDescriptor pfd, int speed) throws IOException;

private native void native\_close();

private native int native\_read\_array(byte[] buffer, int length) throws IOException;

private native int native\_read\_direct(ByteBuffer buffer, int length) throws IOException;

private native void native\_write\_array(byte[] buffer, int length) throws IOException;

private native void native\_write\_direct(ByteBuffer buffer, int length) throws IOException;

private native void native\_send\_break();

## 注册和添加SerialService

frameworks/base/services/core/java/com/android/server/SerialService.java的SerialService类  
中提供了服务端通过JNI放问驱动的接口，如：

public String[] getSerialPorts()

public ParcelFileDescriptor openSerialPort(String path)

通过向frameworks/base/services//java/com/android/server/SystemServer.java的服务管理器ServiceManager  
添加服务：

serial = new SerialService(context);

ServiceManager.addService(Context.SERIAL\_SERVICE, serial);

并且，通过frameworks/base/core/java/android/app/SystemServiceRegistry.java

[复制代码](javascript:void(0);)

1 registerService(Context.SERIAL\_SERVICE, SerialManager.class,

2 new CachedServiceFetcher<SerialManager>() {

3 @Override

4 public SerialManager createService(ContextImpl ctx) {

5 IBinder b = ServiceManager.getService(Context.SERIAL\_SERVICE);

6 return new SerialManager(ctx, ISerialManager.Stub.asInterface(b));

7 }});

[复制代码](javascript:void(0);)

可以访问串口服务SerialService，客户端通过获得SerialService，就能远程调用getSerialPorts()和  
openSerialPort(String path)接口

## 5. 串口管理SerialManager

    frameworks/base/core/java/android/hardware/SerialManager.java中SerialManager对串口操作进一步管理，  
客户端只要实例化一个SerialManager对象，使用该对象的方法访问串口。不过，该对象只对  
openSerialPort(String name, int speed)管理。

# (B) 打开串口

    通过以上简单分析Android串口服务框架之后，这里进一步分析串口的打开open操作流程。在串口服务中，客户端也调用JNI直接访问底层驱动。服务端只负责对串口的open操作，在frameworks/base/services/core/jni/com\_android\_server\_SerialService.cpp中

[复制代码](javascript:void(0);)

1 static jobject android\_server\_SerialService\_open(JNIEnv \*env, jobject /\* thiz \*/, jstring path)

2 {

3 const char \*pathStr = env->GetStringUTFChars(path, NULL);

4 //打开/dev/tty\*节点

5 int fd = open(pathStr, O\_RDWR | O\_NOCTTY);

6 if (fd < 0) {

7 ALOGE("could not open %s", pathStr);

8 env->ReleaseStringUTFChars(path, pathStr);

9 return NULL;

10 }

11 env->ReleaseStringUTFChars(path, pathStr);

12

13 jobject fileDescriptor = jniCreateFileDescriptor(env, fd);

14 if (fileDescriptor == NULL) {

15 return NULL;

16 }

17 //返回文件描述符，用于跨进程访问文件

18 return env->NewObject(gParcelFileDescriptorOffsets.mClass,

19 gParcelFileDescriptorOffsets.mConstructor, fileDescriptor);

20 }

[复制代码](javascript:void(0);)

而read、write、close等相关操作在frameworks/base/core/jni/android\_hardware\_SerialPort.cpp中，  
其中android\_hardware\_SerialPort\_open只对串口进行相关设置，如波特，数据位等。由此看出  
frameworks/base/services/core/jni/com\_android\_server\_SerialService.cpp和  
frameworks/base/core/jni/android\_hardware\_SerialPort.cpp可能处于两个不同的线程中，这两个不  
同的线程对同一个文件进行访问，需要对文件描述符进行转换。  
在frameworks/base/core/jni/android\_hardware\_SerialPort.cpp中

[复制代码](javascript:void(0);)

1 static void android\_hardware\_SerialPort\_open(JNIEnv \*env, jobject thiz, jobject fileDescriptor, jint speed)

2 {

3 ......

4 int fd = jniGetFDFromFileDescriptor(env, fileDescriptor);

5 // duplicate the file descriptor, since ParcelFileDescriptor will eventually close its copy

6 fd = dup(fd);

7 if (fd < 0) {

8 jniThrowException(env, "java/io/IOException", "Could not open serial port");

9 return;

10 }

11 ......

12 }

[复制代码](javascript:void(0);)

# (C) 设置权限

***device/sprd/scx35l/common/rootdir/root/ueventd.sc8830.rc***

1 ......

2 /dev/ttytiny0 0660 system system

3 ......

***device/sprd/scx35l/common/sepolicy/file\_contexts***

1 ......

2 /dev/ttytiny0 u:object\_r:serial\_device:s0

3 ......

***device/sprd/scx35l/common/sepolicy/system\_app.te***

......

allow system\_app serial\_device:chr\_file { open read write ioctl};

......

# (D) 测试串口

    这里引用Android6.0源码目录下的一个串口测试app，还要添加一些权限等设置才能正常使用。串口测试APP源码：frameworks/base/tests/SerialChat

## 1. 设置权限：

***frameworks/base/tests/SerialChat/Android.mk***

1 ......

2 LOCAL\_CERTIFICATE := platform

3 ......

***frameworks/base/tests/SerialChat/AndroidManifest.xml***

1 ......

2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"

3 package="com.android.serialchat"

4 android:sharedUserId="android.uid.system">

5 ......

## 添加串口节点名称

应用程序根据这个名称来打开对应的串口。

***frameworks/base/core/res/res/values/config.xml***

1 ......

2 <string-array translatable="false" name="config\_serialPorts">

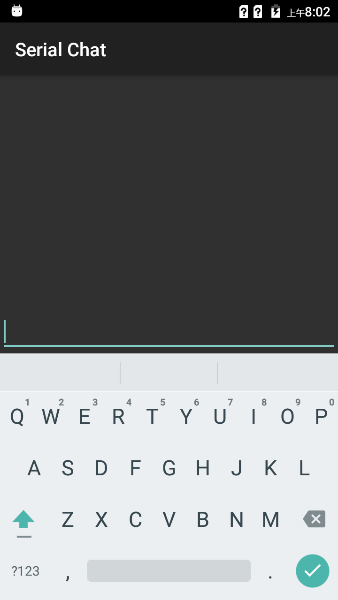
3 <item>"/dev/ttytiny0"</item>

4 </string-array>

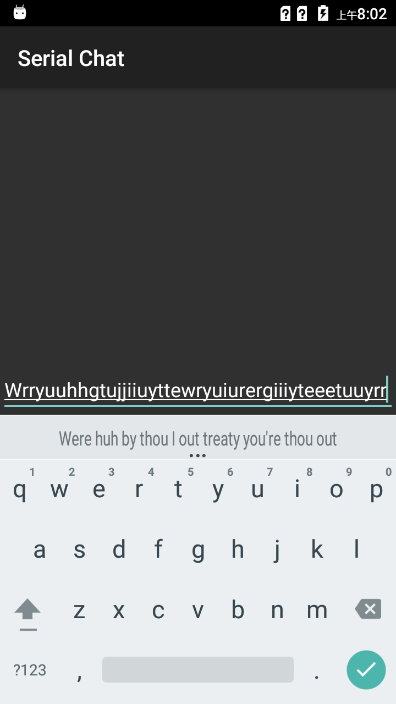
5 ......

    用mmm frameworks/base/tests/SerialChat -B 编译出的SerialChat.apk push到机器之后，就可以  
进行测试了。

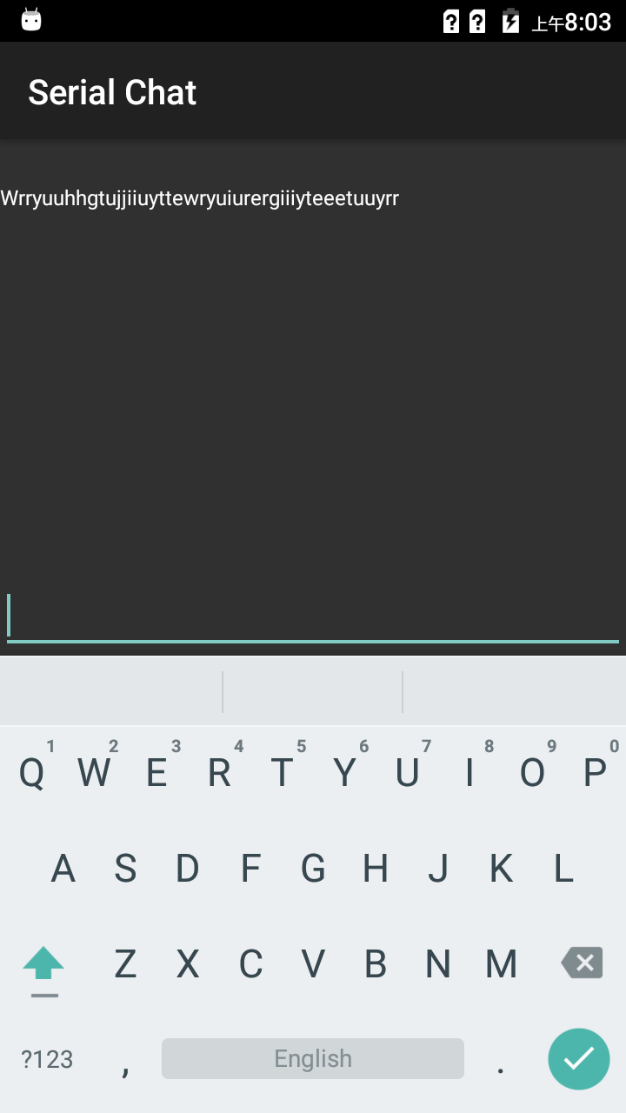
    a.点击打开SerialChat.apk：



    b.输入要发送的内容



    c.点击确定（打钩的位置）发送，并且接收显示在上方。



ervices\java\com\android\server\SystemServer.java

启动

**if** (!isWatch) {  
 *traceBeginAndSlog*(**"StartSerialService"**);  
 **try** {  
 *// Serial port support* serial = **new** SerialService(context);  
 ServiceManager.*addService*(Context.SERIAL\_SERVICE, serial);  
 } **catch** (Throwable e) {  
 Slog.*e*(***TAG***, **"Failure starting SerialService"**, e);  
 }  
 *traceEnd*();  
}

## getSerialPorts

**mSerialPorts关联**

<**string-array translatable="false" name="config\_serialPorts"**>  
</**string-array**>

*/dev/ttyUSB0*

**public** String[] getSerialPorts() {  
 **//**android.Manifest.permission.SERIAL\_PORT, **null**);  
  
 ArrayList<String> ports = **new** ArrayList<String>();  
 **for** (**int** i = 0; i < **mSerialPorts**.**length**; i++) {  
 String path = **mSerialPorts**[i];  
 **if** (**new** File(path).exists()) {  
 ports.add(path);  
 }  
 }  
 String[] result = **new** String[ports.size()];  
 ports.toArray(result);  
 **return** result;  
}

## openSerialPort

### native\_open(path);

### android\_server\_SerialService\_open

**int** fd = open(pathStr, O\_RDWR | O\_NOCTTY);

这个方法，fw几乎没有什么封装，其实也就是一个文件读写而已，没有必要封装吧

# C实现

https://www.cnblogs.com/hackfun/p/7612617.html

# 参考

https://www.cnblogs.com/hackfun/p/7683512.html