[Android RIL概述](http://blog.csdn.net/jason_wzn/article/details/53232022)

<http://blog.csdn.net/jason_wzn/article/details/53392526>

RILD运行机制

<http://blog.csdn.net/jason_wzn/article/details/53232022>

RIL概述

<http://blog.163.com/machao_163job/blog/static/2087753220121020101316838/>

SubscriptionManager描述

**前言**

[**Android**](http://lib.csdn.net/base/android)作为一个通用的移动平台，其首要的功能就是通话、短信以及上网等通信功能。那么，从系统的角度来看，Android究竟是怎么实现与网络的交互的了？ 这篇文章里，就来看一看Android中负责通信功能的Telephony中间层，通常也被称之为RIL(Radio Interface Layer)的具体实现原理与[**架构**](http://lib.csdn.net/base/architecture)。

Android手机要实现与网络端的通信，需要跨越两个层：

* RIL Java(RILJ)：负责将上层APP的通信请求发送给HAL层；
* RIL C++(RILD)： 系统守护进程，负责将RILJ的请求命令发送给CP(Communication Processor)

**什么是RIL**

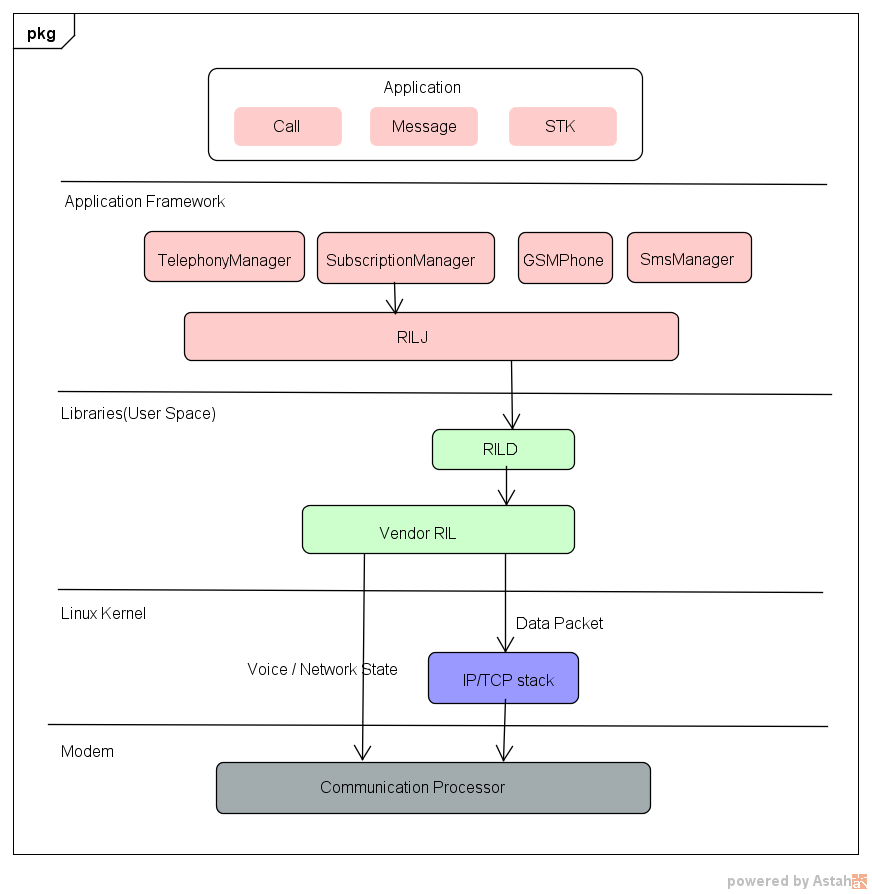
简单的说，RIL(Radio Interface Layer)，就是将应用程序的通信请求发送给CP的中间层，其包括两个部分，一个是[**Java**](http://lib.csdn.net/base/javase)层RILJ,一个是C++层（不妨看作是CP对应的HAL层）RILD。

RILJ属于系统Phone进程的一部分，随Phone进程启动而加载；而RILD守护进程是通过Android的Init进程进行加载的。

**RIL结构**

下图是一个Android RIL的一个结构图。整个通信过程有四个层：

* 最上层的是应用程序，如通话，短信以及SIM卡管理，它们主要负责将用户的指令发送到RIL Framework(以后统称RILJ）；
* RILJ为上层提供了通用的API，如TelephonyManager(包括通话，网络状态； SubscriptionManager(卡状态）以及SmsManager等，同时RILJ还负责维持与RILD的通信，并将上层的请求发送给RILD；
* RILD是系统的守护进程，对于支持通话功能的移动平台是必不可少的。RILD的功能主要功能是将RILJ发送过来的请求继续传递给CP，同时会及时将CP的状态变化发送给RILJ；
* Linux驱动层：kernel驱动层接受到数据后，将指令传给CP，最后由CP发送给网络端，等网络返回结果后，CP将传回给RILD；



RILJ与RILD（RILD与CP的通信）都是通过一个个消息进行数据传递。消息主要分两种：一种是RILJ主动发送的请求（solicited），常见的有RIL\_REQUEST\_GET\_SIM\_STATUS(获取SIM卡状态）， RIL\_REQUEST\_DIAL(拨打电话），RIL\_REQUEST\_SEND\_SMS（发送短信）， RIL\_REQUEST\_GET\_CURRENT\_CALLS（获取当前通话状态），RIL\_REQUEST\_VOICE\_REGISTRATION\_STATE（获取网络状态）； 另一种则是从CP主动上报给RIL的消息（unsolicited)，如网络状态发生变化时，CP会上报RIL\_UNSOL\_RESPONSE\_VOICE\_NETWORK\_STATE\_CHANGED，有新短信时，会上报RIL\_UNSOL\_RESPONSE\_NEW\_SMS，有来电时会上报RIL\_UNSOL\_CALL\_RING。

RIL相关的请求命令与[**数据结构**](http://lib.csdn.net/base/datastructure)都定义在/android/hardware/ril/include/telephony/ril.h

在整个过程中，有几个关键问题：

1. 上层是如何得知RILJ状态变化的；
2. RILJ与RILD是怎么进行通信的？
3. RILJD与CP又是如何进行通信的？

围绕这三个问题点，我们来看一下具体的细节。

### ****上层如何得知RILJ状态变化****

为方便上层实时监听网络状态、通话状态以及CP的状态变化，RIL提供了一个专门的监听接口IPhoneStateListener.aidl，上层需要监听上述状态变化时，只需要实现上述接口,并在Android系统服务TelephonyRegistry中对上述接口实现进行注册：

public void listen(String pkgForDebug, IPhoneStateListener callback, int events, boolean notifyNow);

另外，也可以在TelephonyManager中对RIL状态进行监听：

public void listen(PhoneStateListener listener, int events)

源代码:

/android/frameworks/base/telephony/java/com/android/internal/telephony/IPhoneStateListener.aidl

oneway interface IPhoneStateListener {

void onServiceStateChanged(in ServiceState serviceState);

void onSignalStrengthChanged(int asu);

void onMessageWaitingIndicatorChanged(boolean mwi);

void onCallForwardingIndicatorChanged(boolean cfi);

// we use bundle here instead of CellLocation so it can get the right subclass

void onCellLocationChanged(in Bundle location);

void onCallStateChanged(int state, String incomingNumber);

void onDataConnectionStateChanged(int state, int networkType);

void onDataActivity(int direction);

void onSignalStrengthsChanged(in SignalStrength signalStrength);

void onOtaspChanged(in int otaspMode);

void onCellInfoChanged(in List<CellInfo> cellInfo);

void onPreciseCallStateChanged(in PreciseCallState callState);

void onPreciseDataConnectionStateChanged(in PreciseDataConnectionState dataConnectionState);

void onDataConnectionRealTimeInfoChanged(in DataConnectionRealTimeInfo dcRtInfo);

void onVoLteServiceStateChanged(in VoLteServiceState lteState);

void onOemHookRawEvent(in byte[] rawData);

void onCarrierNetworkChange(in boolean active);

void onFdnUpdated();

void onVoiceRadioBearerHoStateChanged(int state);

}22

### ****RILJ与RILD如何通信****

RILJ在创建过程中，会启动两个线程：RILSender和RILReceiver,RILSender负责将指令发送给RILD,而RILReceiver则负责从读取从RILD发送过来的数据。RILJ与RILD的通信通道就是在RILReceiver中建立起来的。

我们来看一看RILReciver的代码：

class RILReceiver implements Runnable {

byte[] buffer;

RILReceiver() {

...

@Override

public void

run() {

int retryCount = 0;

String rilSocket = "rild";

// 尝试与RILD建立连接

try {for (;;) {

LocalSocket s = null;

LocalSocketAddress l;

if (mInstanceId == null || mInstanceId == 0 ) {

rilSocket = SOCKET\_NAME\_RIL[0];

} else {

rilSocket = SOCKET\_NAME\_RIL[mInstanceId];

}

try {

s = new LocalSocket();

l = new LocalSocketAddress(rilSocket,

LocalSocketAddress.Namespace.RESERVED);

s.connect(l);

} catch (IOException ex){

...

// don't print an error message after the the first time

// or after the 8th time

if (retryCount == 8) {

Rlog.e (RILJ\_LOG\_TAG,

"Couldn't find '" + rilSocket

+ "' socket after " + retryCount

+ " times, continuing to retry silently");

} else if (retryCount >= 0 && retryCount < 8) {

Rlog.i (RILJ\_LOG\_TAG,

"Couldn't find '" + rilSocket

+ "' socket; retrying after timeout");

}

...

retryCount++;

continue;

}

retryCount = 0;

mSocket = s;

// 从socket读取数据

int length = 0;

try {

InputStream is = mSocket.getInputStream();

for (;;) {

Parcel p;

length = readRilMessage(is, buffer);

if (length < 0) {

// End-of-stream reached

break;

}

p = Parcel.obtain();

p.unmarshall(buffer, 0, length);

p.setDataPosition(0);

processResponse(p);

p.recycle();

}

} catch (java.io.IOException ex) {

Rlog.i(RILJ\_LOG\_TAG, "'" + rilSocket + "' socket closed",

ex);

} catch (Throwable tr) {

Rlog.e(RILJ\_LOG\_TAG, "Uncaught exception read length=" + length +

"Exception:" + tr.toString());

}

//无法读取数据，将CP状态设置为不可用

setRadioState (RadioState.RADIO\_UNAVAILABLE);

...

mSocket = null;

RILRequest.resetSerial();

// Clear request list on close

clearRequestList(RADIO\_NOT\_AVAILABLE, false);

}} catch (Throwable tr) {

Rlog.e(RILJ\_LOG\_TAG,"Uncaught exception", tr);

}

}

}

RILReceiver启动时，会建立一个UNIX Domain socket(LocalSocket，kernel层对应/dev/socket/rild)，与RILD进行通信，然后一直从socket中读取数据,并将数据传给上层。连接成功后，RILD会发送一个消息给RILJ，表示连接成功了，这样RILJ就可以将请求数据发送给RILD，进行通信了。

### ****RILD与CP如何进行通信****

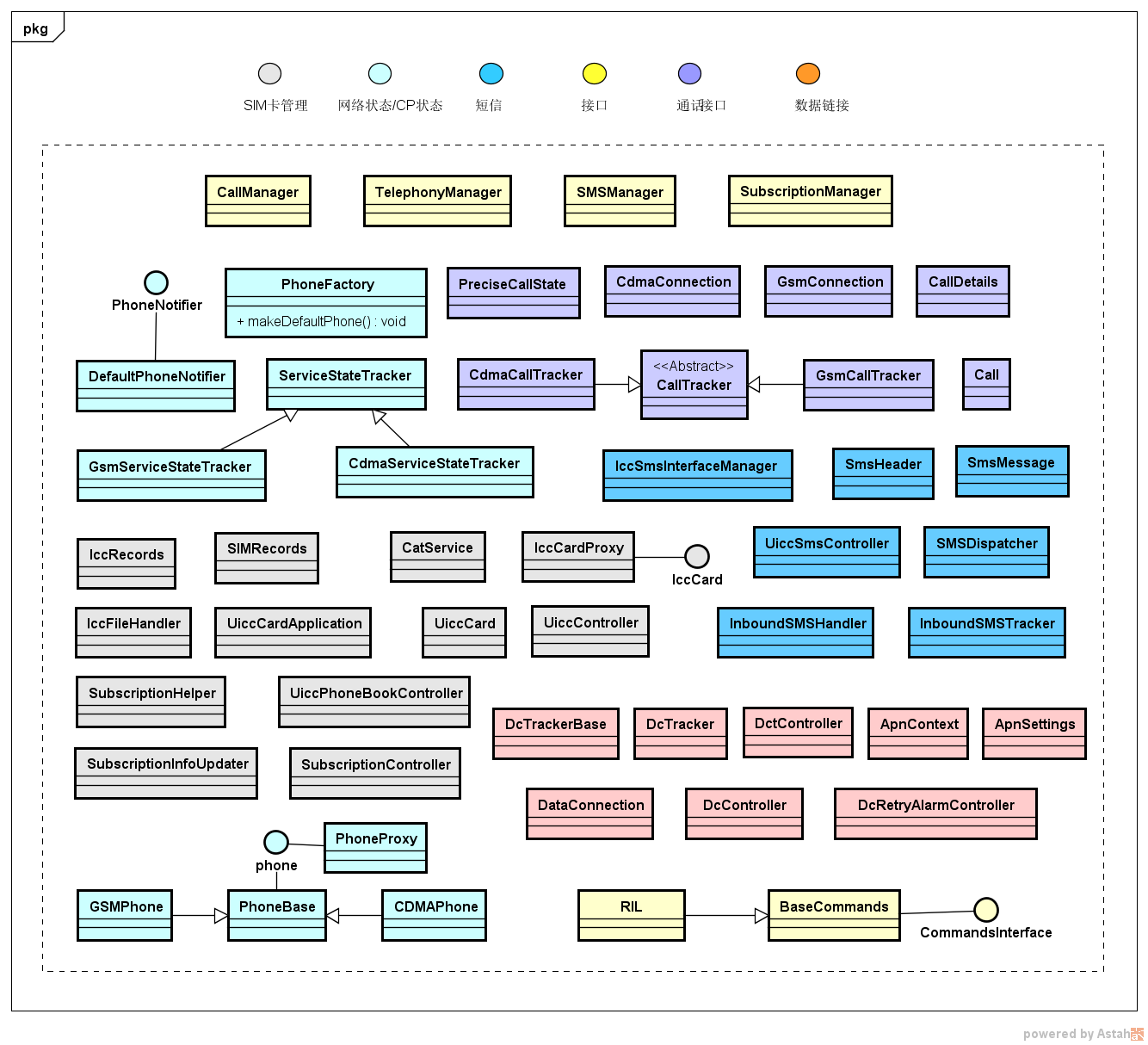
RILD与CP（可以看做是两个运行在不同CPU上的进程通信）交换数据方式一般有两种情况。如果AP与CP集中在一个芯片上，如高通的平台就是将AP与CP集中在一块芯片上，这时通常采用共享内存的方式实现跨进程通信；而如果不是在同一块芯片，而是AP与CP分别采用不同厂商的平台，则一般采用字符设备(character devices) 进行通信。总的说来，共享内存的方式在速度上要优于字符设备。

接下来，主要介绍下RILJ部分的代码结构。

## ****RILJ代码结构****

RIL Framework (RILJ)的代码按照功能来划分的话，主要有以下几个组成部分：

* 管理网络状态（信号强度，网络注册状态等）:ServiceStateTracker等；
* 通话管理（拨号，接听，呼叫等待等）： CallManager,GsmCallTracker等
* SMS短信接收发送： InboundSMSHandler,SmsDispater等
* SIM卡管理： UiccController, SubscriptionsController等
* 数据链接管理： DcTracker,DctController等
* Telephony 大管家： PhoneBase,GsmPhone,PhoneProxy等



以上代码主要位于两个目录：

* /android/frameworks/opt/telephony/ （负责与RILD交互）
* /android/frameworks/base/telephony/ （对上层提供接口）

下面，以拨打电话的流程作为示例看一看RIL是如何发挥作用的。

**示例： CALL流程**

下图是一个MO(Mobile Originated) 通话流程简图：

1. APP向CallManager发送拨号请求；
2. CallManager将通话请求发送给GsmPhone；
3. GsmPhone继续将指令传递给GsmCallTracker；
4. GsmCallTracker调用RILJ，RILJ将通话请求发送给RILD；
5. RILD接收到通话指令时，发送给CP；
6. CP发送给网络，MT(Mobile Terminal)收到通话后，告知网络，由网络将该信息传递给MO已将通话信息发送给MT了（就是手机发出嘟嘟声音的时候）：通话状态由DIALING –> ALERTING；
7. RILD收到通话状态变化的消息后，发送一个UNSOL\_RESPONSE\_CALL\_STATE\_CHANGED的消息给RILJ；
8. RILJ通知GsmCallTracker通话状态变化了；
9. GsmCallTracker主动查询CALL状态：pollCallWhenSafe()，确保得到的信息是对的，没有发生变化；
10. RILJ给RILD发送getCurrentCalls()的请求；
11. RILD获取到CALL状态后，上报给RILJ，再由RILJ返回结果给GsmCallTracker
12. GsmCallTracker得到确定的CALL状态后，通知GsmPhone：notifyPreciseCallStateChanged();
13. GsmPhone将CALL状态变化的消息告知CallManager
14. 最后，CallManager发送CALL状态变化的广播给上层APP

到这一步后，通话并没有开始，如果MT接听了电话，则MO会收到CALL状态变化的信息，然后，才真正开始建立通话链接。

