# Telephony模块总结

|  |  |  |
| --- | --- | --- |
| Date  日期 | Version  版本 | Comments  备注 |
| 2018-02-18 | 0.1 | First version |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Function  职位 | Name  姓名 | Date  日期 | Signature  签名 |
| Written by  拟定 | SW | 曾灿炫 | 2018-02-18 | 曾灿炫 |
| Verified by  审核 |  |  |  |  |
| Verified by  审核 |  |  |  |  |
| Approved by  批准 |  |  |  |  |

目录

[Telephony模块总结 1](#_Toc1851044)

[1. 概述 3](#_Toc1851045)

[**1.1.** **本文档的目的** 3](#_Toc1851046)

[**1.2.** **背景** 3](#_Toc1851047)

[**1.3.** **缩略语清单** 3](#_Toc1851048)

[**1.4.** **参考文献** 3](#_Toc1851049)

[2. Telephony整体架构 4](#_Toc1851050)

[2.1. 简介 4](#_Toc1851051)

[2.2.1. 图解 4](#_Toc1851052)

[2.2.2. 架构组成 4](#_Toc1851053)

[2.2. Telephony Framework 5](#_Toc1851054)

[2.2.1. CatService 5](#_Toc1851055)

[2.2.2. RILJ 5](#_Toc1851056)

[2.2.3. 完整流程 5](#_Toc1851057)

[2.3. RIL层的实现 5](#_Toc1851058)

[2.3.1. RIL层结构 5](#_Toc1851059)

[2.3.2. RILC 5](#_Toc1851060)

[3. modem和RIL层的消息传递 6](#_Toc1851061)

[3.1. qcril+qmi 6](#_Toc1851062)

[3.2. 关键流程 6](#_Toc1851063)

[3.2.1. 监听信息 6](#_Toc1851064)

[3.2.2. 运作流程 6](#_Toc1851065)

[4. 总结 7](#_Toc1851066)

# 概述

* 1. **本文档的目的**

*本文档作为一个学习输出，通过分析不同层级间命令的传递方式，总结STK模块在Telephony层的知识。*

* 1. **背景**

*STK即SIM Tool Kit，它提供一系列用于移动设备与SIM卡间交互的机制。通过这些机制，支持STK的手机可以操作SIM卡里的应用。其Framework层部分（主要涉及Telephony）负责消息的转化和传递，实现modem与应用间的交互。*

* 1. **缩略语清单**

*列出文中使用的术语的定义和缩略（语）词的英文全名和中文解释。*

| Term | Explanation |
| --- | --- |
| CAT | Universal Integrated Circuit Card |
| TR | Terminal Response |
|  |  |

* 1. **参考文献**

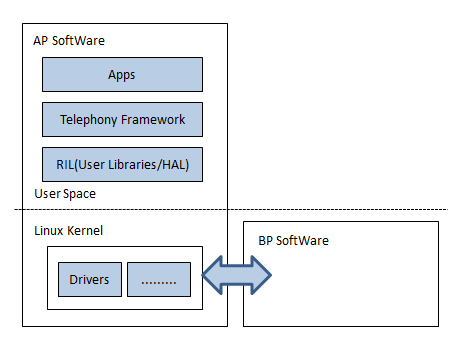
| Document | Explanation |
| --- | --- |
| Android中的Uicc框架 | https://blog.csdn.net/u010961631/article/category/2430627 |
|  |  |
|  |  |

# Telephony整体架构

## 简介

### 2.2.1. 图解

目前的手机硬件结构大多采用双处理器架构，分为AP端与BP端。AP端运行开放式操作系统以及各种基于操作系统的应用，BP端负责无线通信基本能力，Telephony的业务跨越了AP与BP，是我们平时工作主要涉及的内容，具体的结构图如图1。



. Android Telephony框架结构

从这个结构图可以看出：

1. Telephony主要运行在Linux Kernel上的用户空间。
2. **与Android系统分层对应**。
3. BP端是处理无线通信部分，即modem部分。

### 2.2.2. 架构组成

**1.** RIL层指的是工程目录下hardware/ril部分，又称RILC，由C代码组成，与上层java进行通信，具体的消息传递方式在2.3部分讲述。

**2.** Telephony Framework分为telecom和telephony，相关目录为:

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

.../frameworks/base/telecomm/java/com/android/internal/telecom/

.../frameworks/opt/telephony/src/java/com/android/internal/telephony/

Stk模块相关代码为第三个目录下的CatService.java以及RIL.java，其中CatService.java负责RIL层与应用层间的消息传递；RIL.java为RIL层的上层结构，又称RILJ，与Telephony做最直接的交互，作为RIL的消息入口。

**3.**Apps层存放在目录/packages/services/Telephony/与/packages/services/Telecom/，不涉及STK模块。

接下来根据Telephony的分层结构进行单独的分析。

## Telephony Framework

这一部分实现了STK APP与RIL层的信息同步，在modem端上传SIM卡命令后，通过广播的方式将命令发送至APP处理，同时将APP下发的TR传至RILC，最终实现消息的转换与传递。

### CatService

#### 2.2.1.1. 类图



.CatService

CatService继承Handler,实现了AppInterface接口。Handler的运作原理见图3。（图片取自<http://gityuan.com/2015/12/26/handler-message-framework/>）

通过Handler的消息队列原理来接收来自应用层以及RILJ的消息。Handler的运作原理见图3。

1．STK APP通过调用接口函数onCmdResponse()发送消息，具体代码实现：

public synchronized void onCmdResponse(CatResponseMessage resMsg) {

if (resMsg == null) {

return;

}

// queue a response message.

Message msg = obtainMessage(MSG\_ID\_RESPONSE, resMsg);

msg.sendToTarget();

}

可以看到最终通过sendToTarget将消息发至队列。（这里用法与直接new Message 再调用sendMessage不同，跟进前者代码发现最终走的还是sendMessage，区别在于没有新构造一个消息对象，根据注释解释，前者更有效率）

2．回到CatService的构建函数，可以看到：

// Register ril events handling.

mCmdIf.setOnCatSessionEnd(this, MSG\_ID\_SESSION\_END, null);

mCmdIf.setOnCatProactiveCmd(this, MSG\_ID\_PROACTIVE\_COMMAND, null);

mCmdIf.setOnCatEvent(this, MSG\_ID\_EVENT\_NOTIFY, null);

mCmdIf.setOnCatCallSetUp(this, MSG\_ID\_CALL\_SETUP, null);

//mCmdIf.setOnSimRefresh(this, MSG\_ID\_REFRESH, null);

这里调用了CommandInterface的函数去注册实现观察者模式，具体实现在RILJ的父类BaseCommands当中：

public void setOnCatProactiveCmd(Handler h, int what, Object obj) {

mCatProCmdRegistrant = new Registrant (h, what, obj);

}

这里的Hnalder即为CatService自身。



3. Handler

#### 2.2.1.2.关键函数

1. getInstance:：向外提供CatService的单例。

2. handlemessage：处理来自APP以及RILJ的消息。

3. sendStartDecodingMessageParams：接口函数，将RIL Message解码成CommandParams Objects。

4. handleRilMsg：处理解码器转换后的RIL层消息。

5. handleCommand：处理具体的命令。

6. handleCmdResponse：处理APP发来的Repsonse。

7. sendTerminalResponse：发送TR。

那么CatService的具体处理流程图下图：



### 2.2.2. RILJ

#### 2.2.2.1.类图

#### 2.2.2.2.关键函数

### 2.2.3. 完整流程

时序图

## RIL层

进入RILC的目录，可以看到代码结构分为四部分，分别是include、libril、reference-ril、rild。

Include：

ril.h定义了RIL\_Init、RIL\_register等函数和其他结构体，以及RIL消息RIL\_REQUEST\_XXX和RIL\_UNSOL\_XXX。

libril：

ril\_commands.h、ril\_unsol\_commands.h、ril.cpp、ril\_service.cpp。编译输出libril.so。

reference\_ril：

编译输出libreference-ril.so，实现了RIL AT命令的交互机制。

rild：

主要关注点，编译生成可执行文件rild。

### **2.3.1.** RILD

### 2.3.2. 与RILJ的消息传递

# 与Modem端的通信方式（高通）

## 3.1. qcril+qmi

## 3.2. 关键流程

### 3.2.1. 监听信息

### 3.2.2. 运作流程

CatService本质是一个Handle，因此通过handleMessage(Message msg)来追溯其运作流程，

**首先是来自RIL层的消息：**

case MSG\_ID\_SESSION\_END:

case MSG\_ID\_PROACTIVE\_COMMAND:

case MSG\_ID\_EVENT\_NOTIFY:

case MSG\_ID\_REFRESH:

CatLog.d(this, "ril message arrived,slotid:" + mSlotId);

String data = null;

if (msg.obj != null) {

AsyncResult ar = (AsyncResult) msg.obj;

if (ar != null && ar.result != null) {

try {

data = (String) ar.result;

// Begin added by donghai.wu for XR6154319 telecomcode on 2018/09/28

TclPluginManager.getTclStk(mContext).judgeSetupMenu(mSlotId,data);

//End added by donghai.wu for XR6154319 telecomcode on 2018/09/28

} catch (ClassCastException e) {

break;

}

}

}

mMsgDecoder.sendStartDecodingMessageParams(new RilMessage(msg.what, data));

break;

case MSG\_ID\_CALL\_SETUP:

mMsgDecoder.sendStartDecodingMessageParams(new RilMessage(msg.what, null));

可看到当接收到对应信息时，函数会调用sendStartDecodingMessageParams(new RilMessage(msg.what, data))对信息进行解码。

当收到解码的消息，则调用handleRilMsg(RilMessage rilMsg)进行消息的处理。

case MSG\_ID\_RIL\_MSG\_DECODED:

handleRilMsg((RilMessage) msg.obj);

在handleRilMsg(RilMessage rilMsg)中，当消息为MSG\_ID\_SESSION\_END时则进入handleSessionEnd()，发送SESSION END消息至应用层;当消息为MSG\_ID\_EVENT\_NOTIFY 、MSG\_ID\_PROACTIVE\_COMMAND、 MSG\_ID\_REFRESH之一且解码成功时，调用handleCommand(CommandParams cmdParams, boolean isProactiveCmd)，对命令进行下一步处理并根据情况发送对应的TR。

**来自应用层的消息：**

当收到消息为MSG\_ID\_RESPONSE则进入handleCmdResponse((CatResponseMessage) msg.obj)进行上层消息的处理：

case MSG\_ID\_RESPONSE:

handleCmdResponse((CatResponseMessage) msg.obj);

上层传来的消息都是对于命令处理的response，这里会先调用validateResponse(resMsg)确定是否为RIL上传的上一个合法命令的response，不是则不做处理。根据注释里的解释，这里是为了避免“迷惑”基带和RIL层。

根据传来的result code，函数对传来的数据做出相对应的处理，这里不做介绍。在处理完消息后，函数最终调用sendTerminalResponse(CommandDetails cmdDet,ResultCode resultCode, boolean includeAdditionalInfo, int additionalInfo, ResponseData resp)发送TR给RIL层。

# 总结

以上便是STK在Framework 层的运作流程， 这个模块是进行一个消息的转换和传递作用，不涉及命令的细节处理。对这一模块的了解主要依靠阅读源码，根据log进行追溯。了解这一模块有助于理解STK整体运作流程与相关消息的传递和处理。