


首页

登录 注册



潇竹zling

15

文章

1073

浏览

Android RIL源码梳理(1) ——ril启动流程

潇竹zling

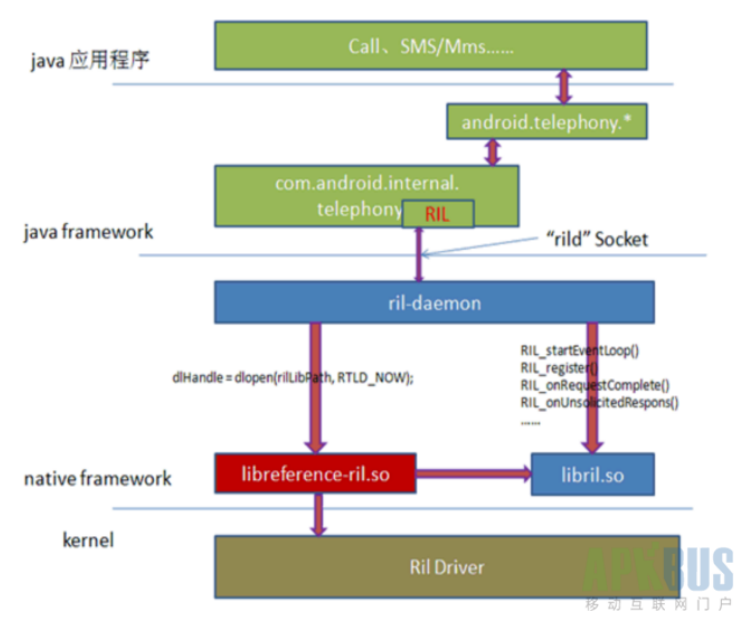
2013-9-3 20:08

(0)

(0)

一、RIL的基本架构

Android RIL (Radio Interface Layer)提供了Telephony服务和Radio硬件之间的抽象层。RIL负责数据的可靠传输、AT命令的发送以及response的解析。一般的，应用处理器（AP）通过AT命令集与无线通讯模块（基带/BP）通信。通信的方式又分为主动请求的request(诸如拨号、发短信……)，以及Modem主动上报的例如信号强度、基站信息、来电、来短信等，称之为unsolicited response。



二、ril-daemon的启动:

首先我们来看一下ril-daemon进程的启动:ril-daemon进程是由init进程在系统开机时负责启动的，该进程在我们系统启动之后就一直存在在系统里面了。在init.rc(system/core/rootdir/)中我们可以看到如下代码：  
//ril-daemon守护进程指的是system/bin/下面的可执行程序rild（此处的rild可执行程序，又是指的什么呢？）

```
service ril-daemon /system/bin/rild
class main
socket rild stream 660 root radio
socket rild-debug stream 660 radio system
user root
group radio cache inet misc audio sdcard_rw log
```

1

帖子

0

听众

0

收听

收听

发消息

加为好友

个人专栏

个人资料

分类

- ril源码分析
- app功能开发
- adb源码分析
- 反编译
- 学习总结
- 紧急通话
- 应用程序暗转
- 应用程序安装
- PhoneStateListener
- Phone

今日头条

关闭

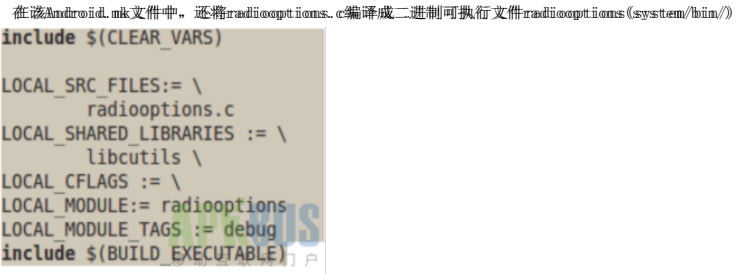
APP安全体检，赢万元好礼！



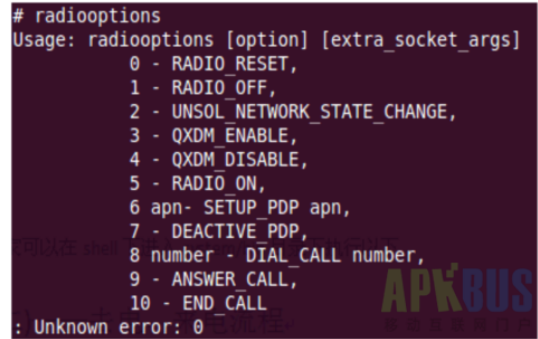
【安卓巴士会员活动】8月3日-9月1日，阿里聚安全与PP助手安卓开放平台联合活动  
“APP安全体检”火热进行，万元好礼等你来赢！！

查看 »

```
从代码中我们可以看到ril可执行程序是由hardware/ril/rild/目录下的rild.c文件编译生成的。
LOCAL_PATH:= $(call my-dir)
include $(CLEAR_VARS)
#编译的资源文件
LOCAL_SRC_FILES:= \
rild.c
LOCAL_SHARED_LIBRARIES := \
    libcutils \
    libril
ifeq ($(TARGET_ARCH),arm)
LOCAL_SHARED_LIBRARIES += libdl
endif # arm
LOCAL_CFLAGS := -DRIL_SHLIB
#编译生成ril可执行程序
LOCAL_MODULE:= rild
include $(BUILD_EXECUTABLE)
```



我们可以在shell下进入system/bin目录，执行以下radiooptions这个命令，我们就可以得到如下结果：



(注意：MTK平台对该模块进行了重新封装，因此在MTK手机的system/bin目录下我们找不到radiooptions可执行文件，如果有兴趣的同学想试着执行以下该命令，可以创建一个模拟器，然后在进入模拟器的/system/bin下面就可以看到上面的结果了)。

三、ril启动流程分析。

前面我们讲了，ril-daemon其实就是ril可执行程序，ril又是由init进程启动的，那么它具体又完成了哪些工作呢？这就需要我们来具体看看ril的入口函数rild.c(hardware/ril/rild/)的实现了。这之前我们先来了解一下几个动态库文件。

- 1、ril、libreference-ril.so、libril.so、radiooptions的作用：
  - 1)、rild(hardware/ril/rild/rild.c)：仅实现一main函数作为整个ril层的入口点，负责完成初始化。
  - 2)、libril.so(hardware/ril/libril/\*)：与ril结合相当紧密，是其共享库，编译时就已经建立了这一关系(感兴趣的同学可以看看hardware/ril/libril/Android.mk文件中动态库的调用关系)。组成部分为ril.cpp，ril\_event.cpp。libril.so驻留在ril这一守护进程中，主要完成同上层通信的工作，接受ril请求并传递给libreference\_ril.so，同时把libreference\_ril.so的反馈回传给调用进程。
  - 3)、libreference\_ril.so(hardware/ril/libreference\_ril/\*)：ril通过手动的dlopen方式加载，结合稍微松散，这也是因为libreference.so主要负责跟Modem硬件通信的缘故。这样
- 做更方便替换或修改以适配更多的Modem种类。它转换来自libril.so的请求为AT命令，同时监控Modem的反馈信息，并传递回libril.so。在初始化时，ril通过符号RIL\_Init获取一组函数指针并以此与之建立联系。

今日头条

关闭

APP安全体检，赢万元好礼！

APP安全体检

【安卓巴士会员活动】8月3日-9月1日，阿里聚安全与PP助手安卓开放平台联合活动“APP安全体检”火热进行，万元好礼等你来赢！！

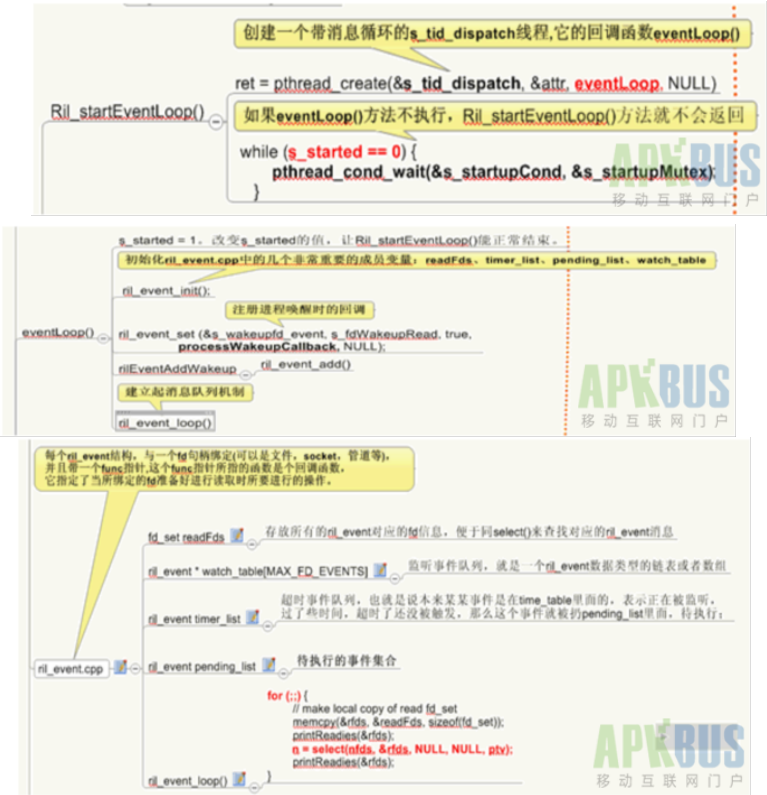
查看 >

4)、radiooptions(hardware/ril/rild/radiooptions.c): radiooptions通过获取启动参数，利用socket与rild通信，可供调试时配置Modem参数。

2、下面我们就来介绍一下rild的启动的流程，首先来看看rild.c的main方法。在rild.c的main方法中，其实主要就完成了3件事，下面我们来一一解析这3件事。



1)、RIL\_startEventLoop()



而ril\_event\_loop就是一个for的无限循环，在这个for内部看到select函数了，其实select只监测read的fd\_set，所要监听的fd都存放在全局变量readFds中，ptv决定select block的形态，

要么设定时间block直到到期，要么无限block直到有监听fd上数据可读，当select返回后就会查找是哪个事件的fd的触发的，然后通过firePending()呼叫该事件的callback。注意这是循环的内部，也

就是说每当select返回并执行其他动作之后，又会重新把readFds加到select中。熟悉Linux的同学应该很清楚这种IO多路复用的select机制（我们也可以在命令提示行下执行man select来

查看select函数的详细介绍）。这样就完成了我们rild启动的第一步就完成了，接下来我们再来看看第二步。

2)、RIL\_Init

在rild.c的main方法是通过如下方式来调用RIL\_Init方法的：

```
dlHandle = dlopen(rilLibPath, RTLD_NOW); //加载libreference-ril.so动态库文件

rilInit = (const RIL_RadioFunctions (*)(const struct RIL_Env *, int, char **))dlsym(dlHandle, "RIL_Init");
```

此处的rilLibPath是通过property属性值的方式来获取的。它的值为：“/system/lib/libreference-ril.so”“动态库文件的属性值。下面就会通过调用动态库的方式来调用

今日头条 关闭

APP安全体检，赢万元好礼！

【安卓巴士会员活动】8月3日-9月1日，阿里聚安全与PP助手安卓开放平台联合活动

“APP安全体检”火热进行，万元好礼等你来赢！！

查看 »

reference-ril.c中RIL\_Init()方法。

创建一个s\_tid\_mainloop的线程，回调函数为mainLoop()

```
RIL_init()
{
    ret = pthread_create(&s_tid_mainloop, &attr, mainLoop, NULL);
}
```

RIL\_Init首先通过参数获取硬件接口的设备文件或模拟硬件接口的socket。接下来便新一个线程继续初始化，即mainLoop。mainLoop的主要任务是建立起与硬件的通信，然后通过read方法阻塞

等待硬件的主动上报或响应在注册一些基础回调（timeout,readerclose）后，mainLoop首先打开硬件设备文件，建立起与硬件的通信，s\_device\_path和s\_port是前面获取的设备路径参数，将其打开。

创建一个s\_tid\_reader线程，AT命令都是以\r\n或\r\n\r\n的换行符来作为分隔符的，所以readerLoop是line驱动的，除非出错，超时等，否则会读到一行完整的响应或主动上报，才会返回。这个循环跑起来以后，我们基本的AT响应机制已经建立了起来

```
at_open()
{
    ret = pthread_create(&s_tid_reader, &attr, readerLoop, &attr);
}
```

有了响应的机制，通过RIL\_requestTimedCallback(initializeCallback,NULL,&TIMEVAL\_0)，跑到initializeCallback中，执行一些Modem的初始化命令，主要都是AT命令的方式。

```
for (;;) { //无线循环
    line = readline(); //AT命令都是以\r\n结束的，都是以一行为单位读取的
    processLine(line);
}
```

上面我们就完成了RIL\_Init的整个过程了。

3)、RIL\_register()

在RIL\_init结束时的返回值为rilInit（RIL\_RadioFunctions结构体类型），先来看看RIL\_RadioFunctions结构体的构成，其中最重要的是onRequest，上层来的请求都由这个函数进行映射

后转换成对应的AT命令发给硬件。

```
typedef struct {
    int version; /* set to RIL_VERSION */
    RIL_RequestFunc onRequest;
    RIL_RadioStateRequest onStateRequest;
    RIL_Supports supports;
    RIL_Cancel onCancel;
    RIL_GetVersion getVersion;} RIL_RadioFunctions;
```

```
funcs = rilInit(&s_rilEnv, argc, rilArgv);
RIL_register(funcs); //ril通过RIL_register注册这一指针。
```

RIL\_register的另外一个重要的作用是：打开和上层通信的socket管道。有了这样一个管道，上层App就可以通过这个管道来和Modem端通信，Modem端也可以通过这个管道向上层App发送响应消息了。

此处即监听我们上层RIL.java中创建的那个“ril”的Socket

```
RIL_Register()
{
    s_fdListen = android_get_control_socket(SOCKET_NAME_RIL);
    ret = listen(s_fdListen, 4);

    s_fdDebug = android_get_control_socket(SOCKET_NAME_RIL_DEBUG);
    ret = listen(s_fdDebug, 4);

    ril_event_set(&s_listen_event, s_fdListen, false, listenCallback, NULL);

    listenCallback()
    {
        s_fdCommand = accept(s_fdListen, (sockaddr *)&peeraddr, &socklen);
        err = getsockopt(s_fdCommand, SOL_SOCKET, SO_PEERCRED, &creds, &szCreds);
    }
}
```

我们也可以看到“ril” socket的通信处理，是在它的回调函数listenCallback中进行处理的，有兴趣的同学可以自行学习一下。到此，ril的启动的流程我们就已经分析完了。

以上的分析都是基于google 4.0的源码的基础上进行的，和我们目前开发的MTK平台有一些差异，MTK自己重新实现了rildlibreference-ril.so、libril.so……，但是它基本的流程任然是一样的，

下面我们再简单看看彼此的差异点。

四、MTK与Google原生的一些差异

1、首先我们先来看一下init.rc的差异

今日头条

关闭

APP安全体检，赢万元好礼！

APP安全体检

【安卓巴士会员活动】8月3日-9月1日，阿里聚安全与PP助手安卓开放平台联合活动

“APP安全体检”火热进行，万元好礼等你来赢！！

查看 »

```
service ril-daemon /system/bin/rild
class core
socket rild stream 660 root radio
socket rild2 stream 660 root radio
socket rild-debug stream 660 radio system
socket rild-mtk-ut stream 660 radio net_bt
socket rild-mtk-ut-2 stream 660 radio net_bt
socket rild-mtk-modem stream 660 radio system
socket rild-atci stream 660 root radio
user root
group radio cache inet misc audio sdcard_rw log cccl
disabled

service ril-daemon /system/bin/rild
class main
socket rild stream 660 root radio
socket rild-debug stream 660 radio system
user root
group radio cache inet misc audio sdcard_rw log
```

上边是MTK4.0平台的ril-daemon，下边是google4.0的源代码的。ril-daemon都是通过rild可执行程序来实现的。MTK平台实现了双卡功能，因此它在初始化时创建了两个Socket管道“rild”和“rild2”，这两个socket就是用来上层app和Modem通信的通道。同时还创建了rild-mtk-ut、rild-mtk-ut-2、rild-mtk-modem、rild-atci等socket通道，这些socket通道具体的作用，希望有了解的同学分享一下。

2、在MTK4.1的代码中，我们在hardware/ril/rild/目录下也可以看到一个rild.c文件编译生成的rild可执行文件，但是前面我们又讲到MTK是有自己的实现的，那么这又是怎么回事了？

在ril目录下有一个CleanSpec.mk文件，里面只有如下一句：`$(call add-clean-step, rm -rf $(PRODUCT_OUT)/system/bin/rild)`也就是说，hardware/ril/\*生成的rild可执行程序会被删除掉，也就是说init.rc中定义的ril-daemon的可执行文件并不是此处的rild。

那么那个init.rc中启动的那个rild在哪里呢？在vendor下，我们找到了MTK的rild，这个是以可执行程序的形式Release给我们的。因此我们也看不到它具体的实现的。

```
./mediatek/huaqin17_td_tb_jb/artifacts/out/target/product/huaqin17_td_tb_jb/system/bin/rild
./mediatek/huaqin17_td_tb_jb/artifacts/out/target/product/huaqin17_td_tb_jb/system/bin/rild3
```

3、libreference-ril.so和libril.so动态库文件

前面我们讲到，在rild.c的main方法中会调用dlopen()的方式来加载动态库文件，而动态库文件的路径又是通过property/属性文件的方式获取的，既然如此，那么我们就可以尝试在adb shell下执行getprop方法来查看这个属性

```
root@android:/ # getprop ril.libpath
/system/lib/mtk-ril.so
```

同样我们在vendor下找到了mtk-ril.so以及librilmtk.so库文件。

举报

0

0

1

加入巴单

作者的其他最新专栏

全部

今日头条

关闭

APP安全体检，赢万元好礼！

APP安全体检

【安卓巴士会员活动】8月3日-9月1日，阿里聚安全与PP助手安卓开放平台联合活动“APP安全体检”火热进行，万元好礼等你来赢！！

查看 »

Android 中系统字体、语言、主题改变的实现方式

电话状态改变图

如何实现电话、短信的拦截功能

PhoneStateListener的实现方式

Android应用程序安装过程

PackageManagerServer是如何解析AndroidManifest.xml文件

默认最新(0)

要评论？请先 登录

没有帐号？点击 注册

快捷登录

微信登录

微博登录

QQ登录

评论 (0 个评论)

[关于安卓巴士](#)[联系我们](#)[友情链接](#)[商务合作](#)[版权申明](#)

今日头条 关闭

APP安全体检，赢万元好礼！

APP安全体检

【安卓巴士会员活动】8月3日-9月1日，阿里聚安全与PP助手安卓开放平台联合活动“APP安全体检”火热进行，万元好礼等你来赢！！

查看 »