**Android无线通讯模块设计原理**

1 界面

dialpad1.xml

定义了“0~9,\*，#”数字键的布局；

dialpad\_additional\_buttons\_dial1.xml

定义了“视频”“拨号”“删除”等键的布局；

dialpad\_list\_fragment.xml

定义拨号盘界面的布局

2 拨号流程：

>DialpadFragment: onClick (R.id.dialButton)

>DialpadFragment: dialButtonPressed

>DialpadFragment: newDialNumberIntent (Intent.ACTION\_CALL\_PRIVILEGED)

>OutgoingCallBroadcaster: Start multisimdialer activity

>MsmsDialerActivity: startOutgoingCall

>OutgoingCallBroadcaster: onActivityResult

>OutgoingCallBroadcaster: processIntent (CALL\_PRIVILEGED->CALL)

>OutgoingCallBroadcaster: sendOrderedBroadcast (ACTION\_NEW\_OUTGOING\_CALL)

>OutgoingCallBroadcaster: startSipCallOptionHandler

>SipCallOptionHandler: (Intent.ACTION\_CALL,InCallScreen)

>SipCallOptionHandler: setResultAndFinish

>CallController: placeCall(finally OK to initiate the outgoing call)

>CallController: placeCallInternal(续)

.

.( communicate with ril，truly make call)

.

>PhoneApp: setBeginningCall(enable the proximity sensor)

>PhoneApp: displayCallScreen(just update call screen if already visible)

->(续)CallController: placeCallInternal

>PhoneUtils.placeCall

>CallManager: dial

> GSMPhone: dial

> GsmCallTracker: dial

> RIL: dial

> SprdRIL: dial(RIL\_REQUEST\_DIAL)

>RIL: RILSender. handleMessage(EVENT\_SEND)

.

.(socket)

.

> RIL: RILReceiver. run(**rild**)

>Rild: main

>Ril.cpp: RIL\_startEventLoop

>Reference-ril.c: RIL\_Init

>Ril.cpp: RIL\_register

>Ril\_event.cpp: ril\_event\_set listenCallback (Initialize an event non-persistent so we can accept only one connection at a time)

>ril.cpp: listenCallback( create listen socket for RIL-JAVA)

>Ril\_event.cpp: ril\_event\_set processCommandsCallback (Initialize an event: “True” means no need to delete from watchlist)

>ril.cpp: processCommandsCallback(loop until EAGAIN/EINTR, end of stream, or other error)

>ril.cpp: processCommandBuffer(add to s\_pendingRequests)

>ril.cpp: s\_commands

static CommandInfo s\_commands[] = {

#include "ril\_commands.h"

};

>ril.cpp: dispatchXXX (dispatchDial)

>Reference-ril.c: onRequest(RIL\_REQUEST\_DIAL)

>Reference-ril.c: requestDial

>atchannel.c: at\_send\_command

.

.(send at command to modem via uart)

.

>Reference-ril.c:s\_rilenv-> OnRequestComplete

rild.c main: funcs = rilInit(&s\_rilEnv, argc, rilArgv);

Reference-ril.c RIL\_Init: s\_rilenv = env;

>ril.cpp: RIL\_onRequestComplete

>ril.cpp: sendResponse

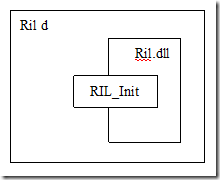
>ril.cpp: sendResponseRaw

Android电话系统之-rild

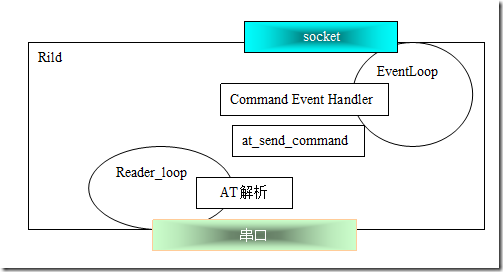
Rild是Init进程启动的一个本地服务，这个本地服务并没有使用Binder之类的通讯手段，而是采用了socket通讯这种方式。RIL(Radio Interface Layer)

Android给出了一个ril实现框架。由于Android开发者使用的Modem是不一样的，各种指令格式，初始化序列都可能不一样，GSM和CDMA就差别更大了，所以为了消除这些差别，Android设计者将ril做了一个抽象，使用一个虚拟电话的概念。这个虚拟电话对象就是GSMPhone（CDMAPhone),Phon对象所提供的功能协议，以及要求下层的支撑环境都有一个统一的描述，这个底层描述的实现就是靠RIL来完成适配。

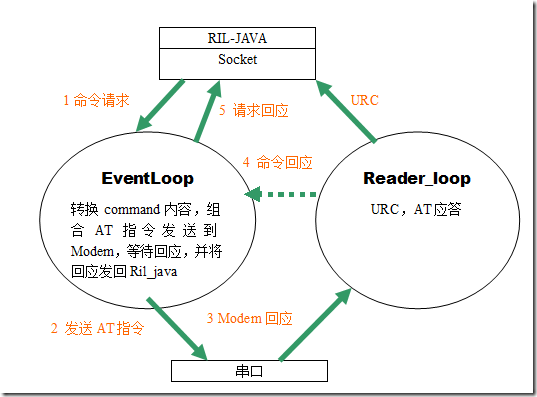
Andoid将RIL层分为两个代码空间：RILD管理框架，AT相关的xxxril.so动态链接库。将ＲＩＬ独立成一个动态链接库的好处就是Ａｎｄｒｏｉｄ系统适应不同的Ｍｏｄｅｍ，不同的Mode可以有一个独立的Ril与之对应。从这个层面上看，Rild更多是一个管理框架。

[](http://hi.csdn.net/attachment/201005/10/0_1273502394d94y.gif)

    而ril是具体的AT指令合成者和应答解析者。从最基本的功能来讲，ril建立了一个侦听Socket，等待客户端的连接，然后从该连接上读取RIL-Java成传递来的命令并转化成AT指令发送到Modem。并等待Modem的回应，然后将结果通过套接口传回到Ril-Java层。下图是Ril-D的基本框架：

[](http://hi.csdn.net/attachment/201005/10/0_1273502396C9X1.gif)

下面的数据流传递描述图表描述了RIL-JAVA层发出一个电话指令的5 步曲。

[](http://hi.csdn.net/attachment/201005/10/0_1273502399hSwI.gif)

在AT通讯的过程中有两类响应：一种是请求后给出应答，一种是通知类，即为不请自来的，例如短信通知达到，我们称该类通知为URC。在Rild中URC和一般的Response是分开处理的，概念上URC由handleUnsolicited@Atchannel.c处理，而Response由handleFinalResponse来处理。

1 Event Loop

Rild管理的真正精髓在ril.cpp,ril\_event.cpp中，在研究的过程中，可以看到设计者在抽象上所下的功夫，设计得很优美。Event Loop的基本工作就是等待在事件端口（串口，Socket），一旦有数据到达就根据登记的Event回调函数进行处理。现在来看Ril设计者是如何建立一套管理框架来完成这些工作的？

### 1.1 Event对象

Event对象构成：（fd,index,persist,func,param）

|  |  |
| --- | --- |
| fd | 事件相关设备句柄。例如对于串口数据事件，fd就是相关串口的设备句柄 |
| index |  |
| persist | 如果是保持的，则不从watch\_list中删除。 |
| func | 回调事件处理函数 |
| param | 回调时参数 |

    为了统一管理事件，Android使用了三个队列：watch\_list,timer\_list,pending\_list,并使用了一个设备句柄池readFDS。

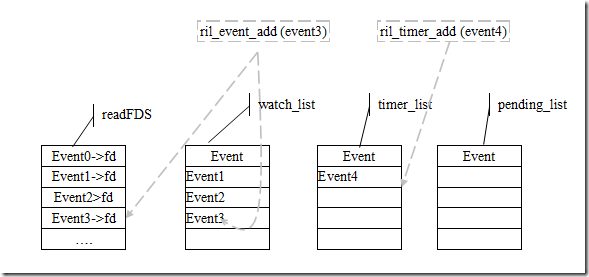
readFDS：是Linux的fd\_set，readFDS保存了Rild中所有的设备文件句柄，以便利用select函数统一的完成事件的侦听。

watch\_list：监测时间队列。需要检测的事件都放入到该队列中。

timer\_list：timer队列

pending\_list:待处理事件队列，事件已经触发，需要所回调处理的事件。

事件队列队列的操作：ril\_event\_add,ril\_event\_del, ril\_timer\_add



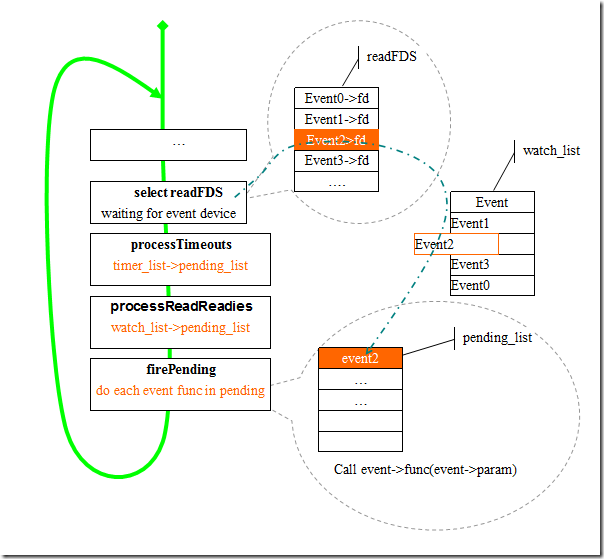
在添加操作中，有两个动作：

(1) 加入到watch\_list

(2) 将句柄加入到readFDS事件句柄池。

### 1.2 ril\_event\_loop()

   我们知道对于Linux设备来讲，我们可以使用select函数等待在FDS上,只要FDS中记录的设备有数据到来，select就会设置相应的标志位并返回。readFDS记录了所有的事件相关设备句柄。readFDS中句柄是在在AddEvent加入的。所有的事件侦听都是建立在linux的select readFDS基础上。



ril\_event\_loop 利用select等待在readFDS（fd\_set）上，当select设备有数据时，ril\_event\_loop会从select返回，在watch\_list中相应的Event放置到pend\_list，如果Event是持久性的则不从watch\_list中删除。然后ril\_event\_loop遍历pengding\_list处理Event事件，发起事件回调函数。

### 1.3 几个重要的Event

上面分析了ril-d的框架，在该框架上跑的事件有什么

（1）s\_listen\_event- （s\_fdListen,listenCallback）

listenCallback处理函数，

接收客户端连接：s\_fdCommand=accepte(..)

添加s\_commands\_event()

重新建立s\_listen\_event，等待下一次连接

（2） s\_command\_event(s\_fdCommand,ProcessCommandsCallback)

从fdCommand  Socket连接中读取StreamRecord

使用ProcessCommandBufer处理数据

s\_listen\_event在大的功能上处理客户端连接（Ril-JAVA层发起的connect）,并建立s\_commands\_event去处理Socket连接发来的Ril命令。ProcessCommandBufer实际上包含了Ril指令的下行过程。

### 1.4 下行命令翻译及其组织@ProcessCommandBuffer

RIL\_JAVA传递的命令格式：Parcel ，由命令号，令牌，内容组成。RIL\_JAVA到达RIL\_C时转为构建本地RequestInfo，并将被翻译成具体的AT指令。由于每条AT命令的参数是不同的，所以对不同的AT指令，有不同的转换函数，在此Android设计在这里做了一个抽象，做了一个分发框架，通过命令号，利用sCommand数组，获得该命令的处理函数。

sComand[]={

<...>

}

sComand 存在于Ril\_command.h中。

&sComand[]=

<

  {RIL\_REQUEST\_GET\_IMEI, dispatchVoid, responseString},

  {RIL\_REQUEST\_DIAL, dispatchDial, responseVoid},

{….}

>

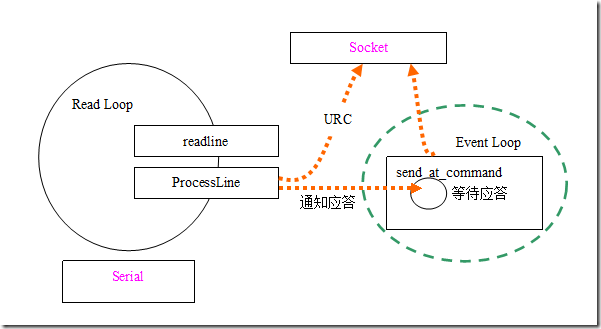
dispatchXxx函数一般都放在在Reference-ril.c中，Reference-ril.c这个就是我们需要根据不同的Modem来修改的文件。

### 1.5 send\_at\_command框架

send\_at\_command是同步的，命令发送后，send\_at\_command将等待在s\_commandcond，直到有sp\_response->finalResponse。

2 read loop@Atchannel.c

Read loop是解决的问题是：解析从Modem发过来的回应。如果遇到URC则通过handleUnsolicited上报的RIL\_JAVA。如果是命令的应答，则通过handleFinalResponse通知send\_at\_command有应答结果。



对于URC，Rild同样使用一个抽象数组@Ril.CPP.

static UnsolResponseInfo s\_unsolResponses[] = {

#include "ril\_unsol\_commands.h"

};

并利用RIL\_onUnsolicitedResponse将URC向上层发送。

### 3 Ril-d的整体数据流及其控制流示意图

