

**Android4.4 Telephony-Framework相关功能底层接口使用说明**

版本： V1.1

日期：2018-07-11

**目 录**

[版本历史 1](#_Toc516765751)

[1](#_Toc516765752) 获取网络类型[接口](#_Toc516765752) 5

[2 获取SIM卡存在状态接口](#_Toc516765753) 6

[3 获取信号强度接口](#_Toc516765754) 7

4 开关协议栈的接口.............................................................................................................................................8

5 数据连接激活的接口.........................................................................................................................................9

6 数据连接断开的接口.........................................................................................................................................11

**版本历史**

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 日期 | 作者 | 备注 |
| V1.0 | 2018.6.15 | Bingjie.chen |  |
| V1.1 | 2018.7.11 | Bingjie.chen |  |
|  |  |  |  |
|  |  |  |  |

声明

本文件所含数据和信息都属于紫光展锐机密及紫光展锐财产，紫光展锐保留所有相关权利。当您接受这份文件时，即表示您同意此份文件内含机密信息，且同意在未获得紫光展锐同意前，不使用或复制、整个或部分文件。紫光展锐有权在未经事先通知的情况下，对本文件做任何修改。紫光展锐对本文件所含数据和信息不做任何保证，在任何情况下，紫光展锐均不负责任何与文件相关的直接或间接的、任何伤害或损失。

1. 获取网络类型接口

**1.1 接口调用方式如下：**

#define RIL\_REQUEST\_VOICE\_REGISTRATION\_STATE 20

void getVoiceNetworkRegistrationState(){

Parcel p;

p.writeInt32(RIL\_REQUEST\_VOICE\_REGISTRATION\_STATE);

p.writeInt32(s\_token++);

sendRequest(p.data(), p.dataSize(), socket\_fd);

}

其中：

1). RIL\_REQUEST\_VOICE\_REGISTRATION\_STATE宏是固定值，不可随意更改。

2). sendRequest( ， ， )方法在上次提供的demo code中存在。

3). s\_token的值是唯一的，**该方法是异步接口**，在receive responsed时，需要用到此值。

**1.2 接收到responsed的处理过程**

1） 首先从Parcel中依次读取出message-type, token及error值。

Message-type: 此值目前可以不用关注

token：即我们调用接口时传递下去的值，根据此值来判断是哪个请求ID（比如这里的token值对应的请求ID值就是20）。

Error = 0 表示请求处理的结果是success，其它值表示fail

2） 接下来继续从Parcel中读取数据类型是string类型的数组，

首先读出的元素是数组的长度(15)，其中数组中第四个元素值即是网络类型。

值分布如下：

1，2： 表示2G网络

3，9，10，11，15 ： 表示3G网络

14 ： 表示4G网络。

1. 获取SIM卡存在状态接口

**2.1 接口调用方式如下**

#definde RIL\_REQUEST\_GET\_SIM\_STATUS 1

void getIccCardState(){

Parcel p;

p.writeInt32(RIL\_REQUEST\_GET\_SIM\_STATUS);

p.writeInt32(s\_token++);

sendRequest(p.data(), p.dataSize(), socket\_fd);

}

其中：

1). RIL\_REQUEST\_GET\_SIM\_STATUS 宏是固定值，不可随意更改。

2). sendRequest( ， ， )方法在上次提供的demo code中存在。

3). s\_token的值是唯一的，不同与上面的接口对应token值。**该方法也是异步接口**，在receive responsed时，需要用到此值。

**2.2 接收到responsed的处理过程**

1） 首先从Parcel中依次读取出message-type, token及error值。

Message-type: 此值目前可以不用关注

token：即我们调用接口时传递下去的值，根据此值来判断是哪个请求ID（比如这里的token值对应的请求ID值就是1）。

Error = 0 表示请求处理的结果是success, 其它值表示fail

2） 接下来继续从Parcel中读取一个数据类型是int32的数据就可以了，后面的数据目前可以不用关心，读取出的值表示如下：

0：表示sim卡absent。

1：表示sim卡present。

1. 获取信号强度的接口

**3.1 接口调用方式如下**

#define RIL\_REQUEST\_SIGNAL\_STRENGTH 19

void getSignalStrength(){

Parcel p;

p.writeInt32(RIL\_REQUEST\_SIGNAL\_STRENGTH);

p.writeInt32(s\_token++);

sendRequest(p.data(), p.dataSize(), socket\_fd);

}

其中：

1). RIL\_REQUEST\_SIGNAL\_STRENGTH 宏是固定值，不可随意更改。

2). sendRequest( ， ， )方法在上次提供的demo code中存在。

3). s\_token的值是唯一的，不同与上面的两个接口对应token值。**该方法也是异步接口**，在receive responsed时，需要用到此值。

**3.2 接收到responsed的处理过程**

1） 首先从Parcel中依次读取出message-type, token及error值。

Message-type: 此值目前可以不用关注

token：即我们调用接口时传递下去的值，根据此值来判断是哪个请求ID（比如这里的token值对应的请求ID值就是19）。

Error = 0 表示请求处理的结果是success, 其它值表示fail

2） 接下来继续从Parcel中反序列化出下面signalStrength类型的结构体数据：

typedef struct｛

int32 mGsmSignalStrength;

int32 mGsmBitErrorRate;

int32 mCdmaDbm;

int32 mCdmaEcio;

int32 mEvdoDbm;

int32 mEvdoEcio;

int32 mEvdoSnr;

int32 mLteSignalStrength;

int32 mLteRsrp;

int32 mLteRsrq;

int32 mLteRssnr;

int32 mLteCqi;

｝signalStrength;

目前可以只关注 mLteRsrp的值(即信号强度的值)，它的单位是dBm.

1. 开关协议栈的接口

**4.1 接口调用方式如下：**

#define RIL\_REQUEST\_RADIO\_POWER 23

void radioPower(bool isOn) {

int ret;

int request = RIL\_REQUEST\_RADIO\_POWER;

Parcel p;

p.writeInt32(request);

p.writeInt32(s\_token++);//token

p.writeInt32(1);

p.writeInt32(isOn?1:0);

RLOGD("write 'radio power on' into socket");

ret = sendRequest(p.data(), p.dataSize(), s\_socket\_fd);

}

其中：

1). RIL\_REQUEST\_RADIO\_POWER 宏是固定值，不可随意更改。

2). sendRequest( ， ， )方法在上次提供的demo code中存在。

3). s\_token的值是唯一的，不同与上面的两个接口对应token值。**该方法也是异步接口**，在receive responsed时，需要用到此值。

4). 参数isOn：true表示开协议栈，false表示关协议栈。

**4.2 接收到responsed的处理流程**

1） 首先从Parcel中依次读取出message-type, token及error值。

Message-type: 此值目前可以不用关注

token：即我们调用接口时传递下去的值，根据此值来判断是哪个请求ID（比如这里的token值对应的请求ID值就是23）。

Error = 0 表示请求处理的结果是success, 其它值表示fail

2） 反馈的数据部分是NULL，不需要关注。

1. 数据连接激活的接口

**5.1 接口调用方式如下：**

#define RIL\_REQUEST\_SETUP\_DATA\_CALL 27

void setupDataCall(const char\* radioTechnology, const char\* profile, const char\* apn, const char\* user, const char\* password, const char\* authType, const char\* protocol) {

int ret;

int request = RIL\_REQUEST\_SETUP\_DATA\_CALL;

Parcel p;

p.writeInt32(request);

p.writeInt32(s\_token++;);//token

p.writeInt32(7);

writeStringToParcel(p, radioTechnology);

writeStringToParcel(p, profile);

writeStringToParcel(p, apn);

writeStringToParcel(p, user);

writeStringToParcel(p, password);

writeStringToParcel(p, authType);

writeStringToParcel(p, protocol);

RLOGD("setup data call");

ret = sendRequest(p.data(), p.dataSize(), s\_socket\_fd);

}

其中：

1). RIL\_REQUEST\_SETUP\_DATA\_CALL 宏是固定值，不可随意更改。

2). sendRequest( ， ， )方法在上次提供的demo code中存在。

3). s\_token的值是唯一的，不同与上面的两个接口对应token值。**该方法也是异步接口**，在receive responsed时，需要用到此值。

4). 参数信息如下：

**@param** radioTechnology

\* indicates whether to setup connection on radio technology CDMA

\* (0) or [GSM](http://10.0.1.79:8081/source/s?path=GSM/)/[UMTS](http://10.0.1.79:8081/source/s?path=GSM/UMTS) (1)

\* **@param** profile

\* Profile Number or NULL to indicate default profile

\* **@param** apn

\* the APN to connect to if radio technology is [GSM](http://10.0.1.79:8081/source/s?path=GSM/)/[UMTS](http://10.0.1.79:8081/source/s?path=GSM/UMTS).

\* Otherwise null for CDMA.

\* **@param** user

\* the username for APN, or NULL

\* **@param** password

\* the password for APN, or NULL

\* **@param** authType

\* the PAP / CHAP auth type. Values is one of SETUP\_DATA\_AUTH\_\*

\* **@param** protocol

\* one of the PDP\_type values in TS 27.007 section 10.1.1.

\* For example, "IP", "IPV6", "IPV4V6", or "PPP".

**5.2 接收到responsed的处理流程**

1） 首先从Parcel中依次读取出message-type, token及error值。

Message-type: 此值目前可以不用关注

token：即我们调用接口时传递下去的值，根据此值来判断是哪个请求ID（比如这里的token值对应的请求ID值就是27）。

Error = 0 表示请求处理的结果是success, 其它值表示fail

2） 反馈的数据部分按照之前提供的demo code进行解析。

注：如果error值表示失败，那么需要查询失败的cause，根据cause值来确定是否需要重试操作。相关的demo code也已经提供。

1. 数据连接断开的接口

**6.1 接口调用方式如下：**

#define RIL\_REQUEST\_DEACTIVATE\_DATA\_CALL 41

void deactivateDataCall(int cid) {

int ret;

int request = RIL\_REQUEST\_DEACTIVATE\_DATA\_CALL;

Parcel p;

p.writeInt32(request);

p.writeInt32(s\_token++);//token

p.writeInt32(2);

writeStringToParcel(p, cid);

writeStringToParcel(p, "");//reason

RLOGD("deactivate data call");

ret = sendRequest(p.data(), p.dataSize(), s\_socket\_fd);

if (ret < 0) {

RLOGE("send at ERROR!");

}

RLOGD("deactivate data call OVER!");

}

其中：

1). RIL\_REQUEST\_DEACTIVATE\_DATA\_CALL 宏是固定值，不可随意更改。

2). sendRequest( ， ， )方法在上次提供的demo code中存在。

3). s\_token的值是唯一的，不同与上面的两个接口对应token值。**该方法也是异步接口**，在receive responsed时，需要用到此值。

4). 参数信息如下：

**@param** cid

\* indicates which one pdp link.

**6.2 接收到responsed的处理流程**

1） 首先从Parcel中依次读取出message-type, token及error值。

Message-type: 此值目前可以不用关注

token：即我们调用接口时传递下去的值，根据此值来判断是哪个请求ID（比如这里的token值对应的请求ID值就是41）。

Error = 0 表示请求处理的结果是success, 其它值表示fail

2） 反馈的数据部分是NULL，不用关注。