

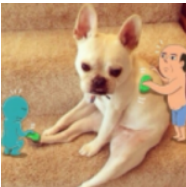
u014386544的博客

目录视图

摘要视图

RSS 订阅

个人资料



我是普通人

关注 发私信

访问： 45529次

积分： 709

等级：

BLOG > 3

排名： 千里之外

原创： 14篇 转载： 88篇
译文： 0篇 评论： 8条

文章搜索

文章分类

- UICC (7)
- 编译出错相关 (2)
- Framework (7)
- 数据连接 (5)
- View事件处理 (1)
- TelephonyRegistry (1)
- TelephonyManager (1)
- spn plmn (3)
- PhoneInterfaceManager (0)
- Telephony (8)
- Android 多用户问题 (2)
- HandlerThread (1)
- Android内存泄漏相关 (0)
- Android应用相关 (3)
- Phone通话相关 (4)

图灵赠书——程序员11月书单 【力荐】写给想成为前端工程师的同学们！ 异步赠书:kafka/TensorFlow机器学习 每周荐书：OpenCV、自然语言、SpringBoot2

Qualcomm平台qcril初始化及消息处理流程

2016-09-20 11:01 702人阅读 评论(0)

目录(?) [+]

本节主要来介绍Qcril的初始化流程以及消息在Qcril中如何传递。

Android平台不同厂商的AP侧可以相同，但是Modem侧肯定会有很大的差异，RIL层要解决的问题就是适配不同厂商的Modem，为了达到兼容性要求，Android在AP与Modem之间搭建了RILC的框架，由不同的Modem厂商将自己的协议连接到AP侧。**对于高通平台来说，他的RILC就是QCRIL。**

在Qcril中保存一个静态表单，里面保存了所有RILC中下发请求的ID以及相应的处理函数，表单内容简要如下：

[cpp]

```
01. static qcril_dispatch_table_entry_type qcril_event_table[] = {
02.     /* QCRIL_EVT_UIM_QMI_COMMAND_CALLBACK */
03.     { QCRIL_REG_ALL_STATES( QCRIL_EVT_UIM_QMI_COMMAND_CALLBACK, qcril_uim_process_c
04.     /* QCRIL_EVT_UIM_QMI_INDICATION */
05.     { QCRIL_REG_ALL_STATES( QCRIL_EVT_UIM_QMI_INDICATION, qcril_uim_process_qmi_inc
06.     /* QCRIL_EVT_INTERNAL_UIM_VERIFY_PIN_COMMAND_CALLBACK */
07.     { QCRIL_REG_ALL_STATES( QCRIL_EVT_INTERNAL_UIM_VERIFY_PIN_COMMAND_CALLBACK, qcr
08.     /* QCRIL_EVT_INTERNAL_MMGSDI_CARD_POWER_UP */
09.     { QCRIL_REG_ALL_STATES( QCRIL_EVT_INTERNAL_MMGSDI_CARD_POWER_UP, qcril_uim_pro
10.     /* 0x90007 - QCRIL_EVT_HOOK_OEM_ENG_MODE */
11.     { QCRIL_REG_ALL_ACTIVE_STATES( QCRIL_EVT_HOOK_OEM_ENG_MODE, qcril_qmi_nas_reque
12.     /* 1 - RIL_REQUEST_GET_SIM_STATUS */
13.     { QCRIL_REG_ALL_ACTIVE_STATES( RIL_REQUEST_GET_SIM_STATUS, qcril_uim_request_ge
14.     /* 2 - RIL_REQUEST_ENTER_SIM_PIN */
15.     { QCRIL_REG_ALL_ACTIVE_STATES( RIL_REQUEST_ENTER_SIM_PIN, qcril_uim_request_ent
16.     /* 105 - RIL_REQUEST_ISIM_AUTHENTICATION */
17.     { QCRIL_REG_ALL_ACTIVE_STATES( RIL_REQUEST_ISIM_AUTHENTICATION, qcril_uim_reque
18. }
```

里面每一项都包含两个元素：事件ID和处理函数，在发生事件时，会调用对应的处理函数。

比如，对于得到当前SIM卡状态这个请求，对应的ID为RIL_REQUEST_GET_SIM_STATUS，其处理函数为：qcril_uim_request_get_sim_status()。

一、Qcril初始化流程



Unable to Conn

The Proxy was unable to connect to the remote site. responding to requests. If you feel you have reached please submit a ticket via the link provided below.

URL: http://pos.baidu.com/s?hei=250&wid=300&di=u%2Fblog.csdn.net%2Fu014386544%2Farticle%2Fde

文章存档

2017年07月 (4)

2017年04月 (2)

2017年03月 (5)

2017年02月 (12)

2017年01月 (1)

展开

阅读排行

SubscriptionInfo框架解析 (2155)

Android--多用户模式 (1650)

VoLTE的前世今生...说清楚VoL... (1585)

Android反射机制实现与原理 (1580)

Android6.0的phone应用源码... (1478)

Android页面返回上一级的三... (1448)

VoLTE技术中的会话持续性-SR... (1349)

Android Call分析(一) ---- Call... (1246)

Android Phone系统架构1 (955)

Android SystemProperties设... (921)

评论排行

Android6.0的phone应用源码... (2)

Telephony之PhoneInterface... (2)

6.0视频电话流程 (1)

Android运营商名称显示之PL... (1)

Android页面返回上一级的三... (1)

Android7.0 IMS开机初始化 (1)

Android N 通话界面_CallButt... (0)

《JAVA与模式》之状态模式 (0)

Android中Service的使用详解 (0)

Android Phone架构设计 (0)

推荐文章

* CSDN邀请您来GitChat赚钱啦！

* 改做人工智能之前，90%的人都没能给自己定位

* TensorFlow 人脸识别网络与对抗网络搭建

* Vue 移动端项目生产环境优化

* 面试必考的计算机网络知识点梳理

* Node 企业项目大规模实践

* TCP/IP 和 HTTP不了解？看完这篇文章，网络知识就全懂了

最新评论

Android页面返回上一级的三种方式

weixin_38638959 : @Override public void

初始化流程需要完成EventLoop消息循环的建立、各个模块的初始化等工作。先看RILD部分。

```
[cpp]
01. @rild.c
02. int main(int argc, char **argv)
03. {
04.     //动态加载reference-ril.so或者qcril.so
05.     dlHandle = dlopen(rilLibPath, RTLD_NOW);
06.     //创建Loop监听Socket事件
07.     RIL_startEventLoop();
08.     rilInit = (const RIL_RadioFunctions *) (const struct RIL_Env *, int, char **);
09.     funcs = rilInit(&s_rilEnv, argc, rilArgv);
10.     RIL_register(funcs);
11. }
```

在RILD中会通过dlsym查找ril库中的RIL_Init函数地址，然后通过rilInit调用，对数在qcril.c中：

```
[cpp]
01. @qcril.c
02. const RIL_RadioFunctions *RIL_Init ( const struct RIL_Env *env, int argc, char **ar
03.     //设置线程名字为rild
04.     qmi_ril_set_thread_name( pthread_self(), QMI_RIL_QMI_RILD_THREAD_NAME);
05.     qmi_ril_fw_android_request_flow_control_init();
06.     //初始化unsol的eventlist pending_unsol_resp_list, 为其分配内存
07.     qmi_ril_init_android_unsol_resp_pending_list();
08.     //初始化接收Modem消息的EventLoop
09.     qcril_event_init();
10.     //初始化qcril中的各个模块
11.     qcril_init();
12.     //开启EventLoop
13.     qcril_event_start();
14.     //其他初始化
15.     qmi_ril_initiate_bootup();
16.     //返回RILD对RILC的接口函数
17.     return &qcril_request_api[ QCRIL_DEFAULT_INSTANCE_ID ];
18. }
```

下面我们分别来分析上面的过程。

1.1、初始化EventLoop过程

在Qcril中搭建了EventLoop循环用于检测Modem上报的消息，而EventLoop机制的初始化工作是在qcril_event_init()中完成的。

```
[cpp]
01. @qcril_event.c
02. void qcril_event_init( void ) {
03.     pthread_attr_t attr;
04.     int ret;
05.     qcril_event.started = 0;
06.     MI_RIL_UTF
07.     pthread_attr_init (&attr);
08.     ret = utf_pthread_create_handler(&
09.     qcril_event.tid, &attr, qcril_event_main, NULL);
10.     pthread_attr_destroy( &attr );
11.     pthread_attr_init (&attr);
12.     pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
13.     //创建EventLoop线程，线程入口是qcril_event_main
```

关闭

```
onBackPressed(){...
```

6.0视频电话流程

jioumk : 无耻 盗别人博主“W歹匕示申W”原创。垃圾~<http://blog.csdn.net/csh8627...>

Telephony之PhoneInterfaceManager

popuwal : 我觉得应该是TelephonyManager作为Telephony框架对外的“接口人”，经过Phone...

Telephony之PhoneInterfaceManager

popuwal : 我觉得应该是TelephonyManager作为Telephony框架对外的“接口人”，经过Phone...

Android运营商名称显示之PLMN的读取

beibai12345 : 你好，如果Android没有插入Sim卡，如何获取plmn？

Android 7.0 IMS开机初始化

android_xiaozhao : 格式没有弄好....

Android 6.0的phone应用源码分析(4) —...

我是普通人 : @spanklebobo:慢慢看就好了，呵呵

Android 6.0的phone应用源码分析(4) —...

sparklebobo : 好难懂啊。。

Android 6.0 IMS流程(二)——接口扩展(从...

我是普通人 : @linyongan:好的，马上删除

Android 6.0 IMS流程(二)——接口扩展(从...

我是普通人 : @linyongan:为啥啊

```
14.         ret = pthread_create(&qcril_event.tid, &attr, qcril_event_main, NULL);
15.         pthread_attr_destroy(&attr);
16.
17.         //设置线程名字为"event"
18.         qmi_ril_set_thread_name(qcril_event.tid, QMI_RIL_EVENT_THREAD_NAME);
19.         pthread_mutexattr_init(&qcril_event.activity_lock_mutex_attr);
20.         pthread_mutex_init(&qcril_event.activity_lock_mutex, &qcril_event.activity_lock_mutexattr);
21.         while (qcril_event.started == 0)
22.         {
23.             pthread_cond_wait(&qcril_event_startupCond, &qcril_event.startup_mutex);
24.         }
25.     }
```

在初始化过程中，通过pthread_create()函数创建了EventLoop线程，并且指出数为qcril_event_main()，我们从线程的入口开始分析：

```
[cpp]
01. static void *qcril_event_main ( void *param) {
02.     int ret;
03.     int filedes[2];
04.     int n;
05.     fd_set rfd;
06.     qcril_event_type *ev;
07.     char buff[16];
08.     int errnoType err_no;
09.     int go_on;
10.
11.
12.     param = param;
13.     pthread_mutex_init(&qcril_event.list_mutex, NULL);
14.     //初始化qcril_event.list链表
15.     qcril_event_init_list(&qcril_event.list);
16.     FD_ZERO(&qcril_event.readFds); /* Needed to use select() system call */
17.     QCRIL_MUTEX_LOCK(&qcril_event.startup_mutex, "[Event Thread] qcril_event.startup_mutex");
18.     qcril_event.started = 1;
19.     //创建管道
20.     ret = pipe(filedes);
21.     qcril_event.fdWakeupRead = filedes[0];
22.     qcril_event.fdWakeupWrite = filedes[1];
23.
24.
25.     fcntl(qcril_event.fdWakeupRead, F_SETFL, O_NONBLOCK);
26.     FD_SET(qcril_event.fdWakeupRead, &qcril_event.readFds);
27.     pthread_cond_broadcast(&qcril_event_startupCond);
28.     while (qcril_event.started < 2)
29.     {
30.         //阻塞等待qcril初始化
31.         pthread_cond_wait(&qcril_event_startupCond, &qcril_event.startup_mutex);
32.     }
33.
34.
35.     for (;;)
36.     {
37.         /* Make a local copy of read fd_set; Don't ask whv. */
38.         memcpy(&rfd, &qcril_event.readFds, sizeof(fd_set));
39.         //阻塞等待接收内容
40.         n = select(qcril_event.fdWakeupRead + 1, &rfd, NULL, NULL, NULL);
41.         if (n < 0)
42.         {
43.             if (errno == EINTR) continue;
44.             QCRIL_LOG_ERROR("QCRIL event select error (%d)", errno);
45.             qmi_ril_clear_thread_name(pthread_self());
46.             return NULL;
47.         }
48.         /* Empty the socket */
49.         do
50.         {
51.             //读取内容
52.             ret = read(qcril_event.fdWakeupRead, &buff, sizeof(buff));
```

关闭

```
53.         } while (ret > 0 || (ret < 0 && errno == EINTR));
54.     do
55.     {
56.         if ( ( NULL != ( ev = qcril_event.list.next ) && ( ev != &qcril_event.)
57.         {
58.             qcril_event_remove_from_list( ev );
59.             QCRIL_MUTEX_UNLOCK( &qcril_event.list_mutex, "[Event Thread] qcril_
60.             //处理Modem发送的请求
61.             err_no = qcril_process_event( ev->instance_id, ev->modem_id, ev->ev
62.             QCRIL_MUTEX_LOCK( &qcril_event.list_mutex, "[Event Thread] qcril_ev
63.             if ( ev->data_must_be_freed && ev->data )
64.             {
65.                 qcril_free( ev->data );
66.             }
67.             qcril_free( ev );
68.         }
69.         go_on = ( ( NULL != ( ev = qcril_event.list.next ) && (
70.     } while ( go_on );
71.
72.
73.     }
74.     qmi_ril_clear_thread_name(pthread_self());
75.     return NULL;
76. }
```

在以上过程中，完成qcril_event.list链表的初始化，然后通过pthread_cond_wait进入阻塞状态，当被解锁后以及进入EventLoop循环，检测到事件后，通过qcril_process_event处理。

1.2、初始化qcril各个模块

Qcril在接到RILC的请求后，需要根据请求的类型将消息派发给不同的负责模块，而qcril_init()就是完成各个模块的初始化工作。

```
[cpp] C
01.     void qcril_init ( void ) {
02.         qcril_arb_init();
03.         qcril_init_state();
04.         qmi_ril_oem_hook_init();
05.         qcril_db_init();
06.         //初始化Event table
07.         qcril_init_hash_table();
08.         qcril_reqlist_init();
09. #ifdef FEATURE_QCRIL_PLMN_LIST
10.         qcril_qmi_nas2_init();
11. #endif
12.         qcril_request_suppress_list_init();
13.         qmi_ril_qmi_client_pre_initialization_init();
14.         qmi_ril_qmi_client_pre_initialization_acquire();
15.         qcril_qmi_nas_dms_common_pre_init();
16.         qcril_qmi_voice_pre_init();
17. #ifndef QMI_RIL_UTF
18.         qcril_am_pre_init();
19. #else
20.         qmi_ril_rat_enable_option = QMI_RIL_FTR_RAT_UNKNOWN;
21.         qmi_ril_baseband_ftr_info = QMI_RIL_FTR_BASEBAND_UNKNOWN;
22. #endif
23.         qcril_qmi_imsa_pre_init();
24.         qcril_qmi_sms_pre_init();
25.         QCRIL_LOG_FUNC_RETURN();
26.     }
```

[关闭](#)

在这里对qcril的各个模块进行初始化。其中完成了很重要的一步就是将qcril_event_table表拷贝

给qcril_hash_table，用于onRequest时对各种请求进行处理，我们来看具体操作：

```
[cpp]
01. static void qcril_init_hash_table( void ) {
02.     uint32 reg_index, hash_index; /*!< index into hash table */
03.     qcril_dispatch_table_entry_type *temp_entry_ptr;
04.     for (reg_index = 0; reg_index < QCRIL_ARR_SIZE( qcril_event_table ); reg_index++)
05.     {
06.         hash_index = qcril_hash( qcril_event_table[reg_index].event_id, QCRIL_HT_ENTRIES_MAX);
07.         if(hash_index < QCRIL_HT_ENTRIES_MAX)
08.         {
09.             if (qcril_hash_table[hash_index] == NULL)
10.             {
11.                 //将qcril_event_table拷贝给qcril_hash_table
12.                 qcril_hash_table[hash_index] = &qcril_event_table[reg_index];
13.             }
14.             else
15.             {
16.                 temp_entry_ptr = qcril_hash_table[hash_index];
17.                 while (temp_entry_ptr->next_ptr != NULL)
18.                 {
19.                     temp_entry_ptr = temp_entry_ptr->next_ptr;
20.                 }
21.                 temp_entry_ptr->next_ptr = &qcril_event_table[reg_index];
22.             }
23.         }
24.     }
25. }
```

经过上面的拷贝，qcril_event_table中就保存了所有Request的id和处理方法。

1.3、开启EventLoop

在1.1中介绍过，初始化EventLoop时，在完成其链表的初始化过程后，通过pthread_cond_wait()将其阻塞，而现在要做的就是取消其阻塞状态，使其进入消息检测循环。

这是在qcril_event_start()中完成的：

```
[cpp]
01. void qcril_event_start( void )
02. {
03.     QCRIL_MUTEX_LOCK( &qcril_event.startup_mutex, "[Main Thread] qcril_event.startup_mutex" );
04.     //更新状态
05.     qcril_event.started = 2;
06.     //释放EventLoop锁
07.     pthread_cond_broadcast(&qcril_event_startupCond);
08.     QCRIL_MUTEX_UNLOCK( &qcril_event.startup_mutex, "[Main Thread] qcril_event.startup_mutex" );
09.
10.
11. }
```

关闭

由于EventLoop被初始化后一直处于阻塞状态，所以在这里将started状态置为2后，对qcril_event_startupCond进行解锁，从而使EventLoop进入循环。

1.4、其他初始化过程

在qmi_ril_initiate_bootup()中完成了一些其他的初始化流程。

```

[cpp]
01. void qmi_ril_initiate_bootup(void)
02. {
03.     qcril_setup_timed_callback( QCRIL_DEFAULT_INSTANCE_ID, QCRIL_DEFAULT_MODEM_ID,
04. }

```

继续看qmi_ril_bootup_perform_core_or_start_polling()过程：

```

[cpp]
01. void qmi_ril_bootup_perform_core_or_start_polling(void * params)
02. {
03.     RIL_Errno init_res;
04.     int ril_version;
05.     qcril_undef_params_type undef_params;
06.     qmi_ril_main_thread_id = pthread_self();
07.     qmi_ril_set_thread_name( qmi_ril_fw_get_main_thread_id(), QMI_RIL_THREAD_NAME );
08.     qmi_ril_wave_modem_status(); // this should result in "modem unavailable"
09.     qmi_ril_set_operational_status( QMI_RIL_GEN_OPERATIONAL_STATUS_IN_TRANSACTION );
10.     qmi_ril_set_operational_status( QMI_RIL_GEN_OPERATIONAL_STATUS_INITIALIZING );
11.     //qmi初始化
12.     init_res = qmi_ril_core_init();
13. }

```

上面通过qmi_ril_core_init()完成了qmi的初始化：

```

[cpp]
01. RIL_Errno qmi_ril_core_init(void)
02. {
03.     RIL_Errno res = RIL_E_GENERIC_FAILURE;
04.     QCRIL_LOG_FUNC_ENTRY();
05.     qcril_event_suspend(); // to ensure atomic init flow cross sub domains
06.     do
07.     {
08.         //qcril client的初始化
09.         res = qcril_qmi_client_init();
10.         if ( RIL_E_SUCCESS != res )
11.             break;
12.         qcril_other_init();
13.         qcril_uim_init();
14.         qcril_gstk_qmi_init();
15. #ifndef QMI_RIL_UTF
16.         qcril_data_init();
17. #endif
18.         qcril_qmi_nas_dms_common_post_init();
19.         if ( qmi_ril_is_feature_supported(QMI_RIL_FEATURE_OEM_SOCKET))
20.         {
21.             QCRIL_LOG_INFO( "%s Init OEM socket thread", __FUNCTION__ );
22.             qcril_qmi_oem_socket_init();
23.         }
24.
25.
26.     } while (FALSE);
27.     qcril_event_resume();
28.     QCRIL_LOG_FUNC_RETURN_WITH_RET(res);
29.     return res;
30. }

```

关闭

在上面完成了qcril客户端的初始化过程：

```

[cpp]
01. RIL_Errno qcril_qmi_client_init( void )
02. {
03.     qmi_client_error_type client_err = 0;
04.     RIL_Errno res = RIL_E_GENERIC_FAILURE;
05.     QCRIL_LOG_FUNC_ENTRY();

```

```
06.      /* Start modem or vote for start modem */
07.      qcril_qmi_modem_power_process_bootup();
08.      memset(&client_info, 0, sizeof(client_info));
09.      do
10.      {
11.
12.
13.          // QMI VOICE command callback
14.          client_info.client_cbs[QCRIL_QMI_CLIENT_VOICE] = qcril_qmi_voice_command_cb;
15.
16.
17.          // Get IDL service objects
18.          client_info.service_objects[QCRIL_QMI_CLIENT_VOICE] = voice_get_service_obj;
19.          client_info.service_objects[QCRIL_QMI_CLIENT_NAS] = nas_get_
20.          client_info.service_objects[QCRIL_QMI_CLIENT_WMS] = wms_get_
21.          client_info.service_objects[QCRIL_QMI_CLIENT_WDS] = wds_get_
22.          client_info.service_objects[QCRIL_QMI_CLIENT_DMS] = dms_get_
23.          /*client_info.service_objects[QCRIL_QMI_CLIENT_UIM] = uim_ge
24.          client_info.service_objects[QCRIL_QMI_CLIENT_PBM] = pbm_get
25.          client_info.service_objects[QCRIL_QMI_CLIENT_RF_SAR] = sar_
26.          client_info.service_objects[QCRIL_QMI_CLIENT_IMS_VT] = ims_
27.          client_info.service_objects[QCRIL_QMI_CLIENT_IMS_PRESENCE] =
28.          client_info.service_objects[QCRIL_QMI_CLIENT_IMSA] = imsa_get_service_obje
29.          client_info.service_objects[QCRIL_QMI_CLIENT_RFPE] = rfrpe_ge
30.          client_info.service_objects[QCRIL_QMI_CLIENT_IMS_SETTING] = imss_get_servic
31.          if ( qmi_ril_get_process_instance_id() == QCRIL_DEFAULT_INSTANCE_ID )
32.          {
33.              client_info.service_objects[QCRIL_QMI_CLIENT_PDC] = pdc_get_service_ob
34.          }
35.
36.
37.          pthread_mutexattr_init(&client_info.cache_lock_mtx_atr);
38.          pthread_mutex_init(&client_info.cache_lock_mutex, &client_info.cache_lock_n
39.          res = qcril_qmi_init_core_client_handles();
40.          if (RIL_E_SUCCESS != res)
41.              break;
42.
43.
44.      } while (FALSE);
45.      return res;
46.  }
```

1.5、将回调函数注册给RILC

在Qcril的初始化完毕后，将自己的函数列表返回给RilC，也就是qcril_request_api:

```
[cpp] C
01. static const RIL_RadioFunctions qcril_request_api[] = {
02.     { RIL_VERSION, onRequest_rid, currentState_rid, onSupports_rid, onCancel_rid,
03.     };
```

这样的话，在RIL中调用的接口就会进入该函数列表中。

[关闭](#)

以上就是qcril的初始化流程。

二、QCRIL对请求的处理过程

当ril有请求过来时，就会调用ril库的onRequest()方法，此时就会根据当前Qcril注册的函数列表进入到qcril_request_api的onRequest_rid()函数中：

```
[cpp]
01. @qcril.c
02. static void onRequest_rid ( int request, void *data, size_t datalen, RIL-Token t)
03. {
04.     onRequest( qmi_ril_process_instance_id, request, data, datalen, t );
05. }
```

然后进入onRequest()中继续处理：

```
[cpp]
01. static void onRequest ( qcril_instance_id_e_type instance_id, int r
02.     udit_result = qmi_ril_fw_android_request_render_execution( param
03.         param.event_id,
04.         param.data,
05.         param.datalen,
06.         param.instance_id,
07.         &log_dispatch_dedicated_thrd );
08. }
```

继续：

```
[cpp]
01. RIL_Errno qmi_ril_fw_android_request_render_execution( RIL-Token token, int android
02.     do
03.     {
04.         entry_ptr = NULL;
05.         //从hash表中查找当前的Event
06.         if ( qcril_hash_table_lookup( (uint32) param.event_id, &entry_ptr ) != E_SU
07.         {
08.             audit_result = RIL_E_REQUEST_NOT_SUPPORTED;
09.             break;
10.         }
11.         if ( dedicated_thrd_req_lookup_val == param.event_id )
12.         { // deferred thread exec
13.         }
14.         else
15.         {
16.             //派发该Event
17.             if ( qcril_dispatch_event( entry_ptr, ¶m ) == E_NOT_ALLOWED )
18.             {
19.                 audit_result = RIL_E_RADIO_NOT_AVAILABLE;
20.                 break;
21.             }
22.         }
23.     } while (FALSE);
24.     return audit_result;
25. }
```

在上面的过程中，要先通过qcril_hash_table_lookup()函数查找当前的Event，如果没有找到当前的Request，就认为非法，找到之后，进入qcril_dispatch_event()中派发该Event：

```
[cpp]
01. IxErrnoType qcril_dispatch_event ( qcril_dispatch_table_entry_type *entry_ptr, qcril
02.     if(params_ptr != NULL && (params_ptr->instance_id < QCRIL_MAX_INSTANCE_ID) )
03.     {
04.         // print the recieved date byte stream
05.         qcril_qmi_print_hex(params_ptr->data, params_ptr->datalen);
06.         instance_id = params_ptr->instance_id;
07.         s_ptr = &qcril_state->info[ instance_id ];
08.         modem_id = params_ptr->modem_id;
09.         if (E_SUCCESS == res)
10.         {
11.             //处理当前Request
12.             (entry_ptr->handler)(params_ptr, &ret);
```

关闭


```
13.         if ( ret.pri_gw_sim_state_changed || ret.pri_cdma_sim_state_changed ||
14.             ret.sec_gw_sim_state_changed || ret.sec_cdma_sim_state_changed ||
15.             ret.ter_gw_sim_state_changed || ret.ter_cdma_sim_state_changed
16.         )
17.         {
18.             qcril_state_transition( instance_id, modem_id, params_ptr->event_id );
19.         }
20.     }
21. }
22. else
23. {
24. }
25. return res;
26. }
```

上面的过程通过entry_ptr->handler调用当前Event的处理函数。这里的handler是qcril_hash_table中的某一项。从上面1.2步骤中我们将qcril_event_table表中的数据插入到qcril_hash_table，所以这里的handler可以理解为qcril_event_table中的某一项。

之后的流程就会进入到某个具体请求的处理函数中，比如对于得到当前SIM卡状态请求，其处理函数为：qcril_uim_request_get_sim_status()。

顶

0

踩

0

- [上一篇](#) Android反射机制实现与原理
- [下一篇](#) 数据业务建立流程之发起网络连接过程

相关文章推荐

framework学习之Qualcomm平台qcril初始化及消息处理流程（原）

白鹭引擎在WebAssembly中的实践

Qualcomm平台qcril初始化及消息处理流程（原）

Apache Weex：移动研发的进阶之路-董岩

Android平台按键消息处理流程一（Android4.2.2）

微信小程序开发实战-快递查询

live555 RTSP服务器建立及消息处理流程

Python内功修炼

ZigBee 中 z-Stack协议中的任务、事件、消息处理...

MySQL深入浅出

小伙伴们的ceph源码分析三——monitor消息处理...

Tensorflow项目实战-文本分类

Openfire Server presence在线状态消息处理流程

MFC消息处理流程概述

freeswitch中文本消息处理流程

live555 RTSP服务器建立及消息处理流程

关闭

查看评论

暂无评论


您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

公司简介 | 招贤纳士 | 广告服务 | 联系方式 | 版权声明 | 法律顾问 | 问题报告 | 合作伙伴 | 论坛反馈

网站客服 杂志客服 微博客服 webmaster@csdn.net 400-660-0108 | 北京创新乐知信息技术有限公司 版权所有 | 江苏知之为计算机有限公司 |

江苏乐知网络技术有限公司

京 ICP 证 09002463 号 | Copyright © 1999-2017, CSDN.NET, All Rights Reserved 

关闭