Android9 拨号流程

G:\code-pj\source\t710\packages\apps\Dialer\java\com\android

拨号应用：

G:\code-pj\source\t710\packages\apps\Dialer

Telecom : InCallController: Components updated: [ComponentInfo{com.android.dialer/com.android.incallui.InCallServiceImpl}]

am\_set\_resumed\_activity: [0,com.android.dialer/com.android.incallui.InCallActivity,minimalResumeActivityLocked]

Dialer : AudioModeProvider.getApproximatedAudioRoute - Routing to earpiece

D ImsPhoneCallTracker: [0] updatePhoneState pendingMo = DIALING, fg= DIALING(1), bg= IDLE(0)

BroadcastQueue: Finished with ordered broadcast BroadcastRecord{203b078 u0 android.intent.action.NEW\_OUTGOING\_CALL} receivers:(2) take

Telecom : Logging.Events: Non-call EVENT: BIND\_CS, ComponentInfo{com.android.phone/com.android.services.telephony.TelephonyConnectionService}: NOCBIR.oR->CSFM.rF@Aq

Telecom : CallerInfoLookupHelper: CI query for handle tel:\*\*\* \*\*\*\* \*\*77 has completed; notifying all listeners.: TSI.pC->CILH.sL->CILH.oQC@AqU

I Telecom : CallerInfoLookupHelper: CI query for handle tel:\*\*\*\*\*\*\*\*\*77 has completed; notifying all listeners.: TSI.pC->CILH.sL->CILH.oQC@AqU

I Telecom : Call: CallerInfo received for tel:\*\*\*\*\*\*\*\*\*77: com.android.internal.telephony.CallerInfo@5145e31 { name null, phoneNumber non-null }: TSI.pC->CI

I Telecom : Event: RecordEntry TC@2: START\_CONNECTION, tel:\*\*\*\*\*\*\*\*\*77: SBC.oSC@Aqw

Telecom : CallerInfoLookupHelper: CI query for handle tel:\*\*\*86 has completed; notifying all listeners.: TSI.pC->CILH.sL->CILH.oQC@AKI

Dialer : AudioModeProvider.getApproximatedAudioRoute - Routing to earpiece

Dialer : InCallPresenter.onCallListChange - Phone switching state: NO\_CALLS -> NO\_CALLS

ImsPhoneCallTracker: [0] isUsim = true isUtEnable = true

GsmCdmaPhone: type = 0

GsmCdmaPhone: [GsmCdmaPhoneEx] useImsForCall=true, useImsForEmergency=false, useImsForUt=true, isUt=false, isWpsCall=false, allowWpsOverIms=true, imsPhone=Handler (com.android.internal.telephony.imsphone.ImsPhone) {84178c1}, imsPhone.isVolteEnabled()=true, imsPhone.isVowifiEnabled()=false, imsPhone.isVideoEnabled()=true, imsPhone.getServiceState().getState()=0

[InCallUIHdAudioHelper]: getInstance()

GsmCdmaPhone: [GsmCdmaPhoneEx] Trying IMS PS call

am\_on\_create\_called: [0,com.android.incallui.InCallActivity,performCreate]

Dialer : InCallFragment.onCreateView

ImsService: createCallSession-> startVoLteCall

RIL : onRequest: DIAL radioState = RADIO\_ON

REQ\_THDS: get Channel1

RIL-AT : Channel1: AT> ATD10086;//打10086

ImsPhoneCallTracker: [0] updatePhoneState pendingMo = DIALING, fg= DIALING(1), bg= IDLE(0)

audio\_hw\_voice: Start pcm\_modem\_ul start

DataPhoneManager[0]: phone0 call start

static struct RIL\_Env s\_rilEnv = {

RIL\_onRequestComplete,

RIL\_onUnsolicitedResponse,

RIL\_requestTimedCallback,

RIL\_onRequestAck

};

ImsPhoneCallTracker.java (frameworks\opt\telephony\src\java\com\android\internal\telephony\imsphone) 179838 2022/2/28

updatePhoneState

Rild.c (hardware\ril\rild) 7045 2022/2/28

Main

--->rilLibPath = argv[i + 1]; //读取ril库路径strcmp(argv[i], "-l")

--->hasLibArgs = 1;//strcmp(argv[i], "--" 库参数

--->clientId = argv[i+1];

--->property\_get(LIB\_PATH\_PROPERTY, libPath, NULL)//如何rilLibPath没有设置 根据属性控制

--->dlHandle = dlopen(rilLibPath, RTLD\_NOW); //打开库并加载

--->RIL\_startEventLoop();

Ril.cpp (hardware\ril\libril) 48228 2022/2/28

--->pthread\_create(&s\_tid\_dispatch, &attr, eventLoop, NULL);

--->eventLoop

--->ril\_event\_init();

Ril.cpp (hardware\ril\libril) 48228 2022/2/28

--->FD\_ZERO(&readFds);

init\_list(&timer\_list);

init\_list(&pending\_list);

memset(watch\_table, 0, sizeof(watch\_table));

--->s\_started = 1;

--->pipe(filedes);

--->s\_fdWakeupRead = filedes[0];

--->s\_fdWakeupWrite = filedes[1];

--->fcntl(s\_fdWakeupRead, F\_SETFL, O\_NONBLOCK);

---> ril\_event\_set (&s\_wakeupfd\_event, s\_fdWakeupRead, true,

processWakeupCallback, NULL);

Ril.cpp (hardware\ril\libril) 48228 2022/2/28

---> ev->fd = fd;--->s\_fdWakeupRead

**--->ev->index = -1;**

**--->ev->persist = persist; --->true**

**--->ev->func = func;--->**processWakeupCallback

--->do {

ret = read(s\_fdWakeupRead, &buff, sizeof(buff));

--->do {

ret = read(s\_fdWakeupRead, &buff, sizeof(buff));

} while (ret > 0 || (ret < 0 && errno == EINTR));

} while (ret > 0 || (ret < 0 && errno == EINTR));

**--->ev->param = param; --->null**

**--->fcntl(fd, F\_SETFL, O\_NONBLOCK);**

---> rilEventAddWakeup (&s\_wakeupfd\_event);

--->ril\_event\_add(ev);

--->for (int i = 0; i < MAX\_FD\_EVENTS; i++) {

if (watch\_table[i] == NULL) {

watch\_table[i] = ev;

ev->index = i;

dlog("~~~~ added at %d ~~~~", i);

dump\_event(ev);

FD\_SET(ev->fd, &readFds);

if (ev->fd >= nfds) nfds = ev->fd+1;

dlog("~~~~ nfds = %d ~~~~", nfds);

break;

}

}

--->triggerEvLoop();

--->do {

ret = write (s\_fdWakeupWrite, " ", 1);

} while (ret < 0 && errno == EINTR);

--->ril\_event\_loop();

Ril\_event.cpp (hardware\ril\libril) 10100 2022/2/28

--->for (;;)

--->select(nfds, &rfds, NULL, NULL, ptv);

--->processReadReadies(&rfds, n);

---> for (int i = 0; (i < MAX\_FD\_EVENTS) && (n > 0); i++) {

struct ril\_event \* rev = watch\_table[i];

if (rev != NULL && FD\_ISSET(rev->fd, rfds)) {

addToList(rev, &pending\_list);

if (rev->persist == false) {

removeWatch(rev, i);

}

n--;

}

}

---> processTimeouts();

--->firePending();

--->while (ev != &pending\_list) {

struct ril\_event \* next = ev->next;

removeFromList(ev);

ev->func(ev->fd, 0, ev->param);

ev = next;

}

--->rilInit =(const RIL\_RadioFunctions \*(\*)(const struct RIL\_Env \*, int, char \*\*))dlsym(dlHandle, "RIL\_Init");//打开ril库中的RIL\_Init函数 通过这个接口调用vendor-ril

--->s\_rilenv = env;-->

--->rilUimInit =(const RIL\_RadioFunctions \*(\*)(const struct RIL\_Env \*, int, char \*\*))dlsym(dlHandle, "RIL\_SAP\_Init");

--->funcs = rilInit(&s\_rilEnv, argc, rilArgv);

--->s\_rilenv = env;-->s\_rilEnv

--->传入:s\_rilEnv

--->RIL\_onRequestComplete

--->RIL\_onUnsolicitedResponse ---》void RIL\_onUnsolicitedResponse(int unsolResponse, const void \*data,

--->RIL\_requestTimedCallback

--->RIL\_onRequestAck

--->ret = pthread\_create(&s\_tid\_mainloop, &attr, mainLoop, NULL);

--->fd = open (s\_device\_path, O\_RDWR);

--->s\_closed = 0;

--->ret = at\_open(fd, onUnsolicited);

--->RIL\_requestTimedCallback(initializeCallback, NULL, &TIMEVAL\_0);

--->setRadioState (RADIO\_STATE\_OFF);

--->at\_send\_command("AT+CMGF=0", NULL);

--->s\_rilenv->RequestTimedCallback(a,b,c)

--->return &s\_callbacks;//vendor ril

--->RIL\_VERSION

--->onRequest

--->currentState

--->onSupports

--->onCancel

--->getVersion

--->RIL\_register(funcs);//funcs=s\_callbacks

Ril.cpp (hardware\ril\libril) 48228 2022/2/28

--->memcpy(&s\_callbacks, callbacks, sizeof (RIL\_RadioFunctions));

--->s\_registerCalled = 1;

--->radio::registerService(&s\_callbacks, s\_commands);--->ril\_commands.h

Ril\_service.cpp (hardware\ril\libril) 331337 2022/2/28

--->s\_vendorFunctions = callbacks;

--->s\_commands = commands;

--->configureRpcThreadpool(1, true /\* callerWillJoin \*/);

--->radioService[i]->mSlotId = i;

--->status = radioService[i]->registerAsService(serviceNames[i]);//这里是注册hidl服务

--->RIL\_register\_socket(rilUimInit, RIL\_SAP\_SOCKET, argc, rilArgv);

--->rilc\_thread\_pool();

T710 android10 ril分析：

Impl\_ril.c (g:\code-pj\source\t710-android10\t710-android10-ril\rild\impl-ril) 67472 2022/5/30

RIL\_Init

--->s\_rilEnv = env;

--->requestHandlerInit(processRequest, SIM\_COUNT);

--->initStk(&s\_stkFunctions);

--->s\_stkFunctions

--->onRILRequest

--->requestSetupDataConnection

--->requestDeactiveDataConnection

--->getDefaultBearerNetAccessName

--->getEthNameByCid

--->onUnsolResponse

--->sendPsDataOffToExtData

--->at\_set\_on\_reader\_closed(onATReaderClosed);

---> ps\_service\_init();

--->pthread\_create(&tid, &attr, (void \*)listenExtDataThread, NULL);

--->pthread\_create(&tid, &attr, detectModemState, NULL);

--->pthread\_create(&s\_mainLoopTid[simId], &attr, mainLoop,

(void \*)&s\_socketId[simId]);

--->init\_channels(socket\_id);

--->start\_reader(socket\_id);

--->pthread\_create(&s\_readerThread[socket\_id].readerTid, &attr,

readerLoop, (void\*)&s\_socketId[socket\_id]);

--->readerLoop

--->select(open\_ATchs + 1, &rfds, NULL, NULL, NULL);

--->read(s\_fdReaderLoopWakeupRead[socket\_id], buf, sizeof(buf));

---> setChannelInitialized(socket\_id);

--->pthread\_create(&tid, &attr, initializeCallback,

(void \*)&s\_socketId[socket\_id]);

--->setCESQGlobalArray(s\_rsrp, s\_ecno, s\_rscp, s\_ber, s\_rxlev, s\_ss\_rsrp);

--->pthread\_create(&tid, &attr, (void \*)signalProcess, (void \*)&s\_networkFunctions);

s\_networkFunctions

--->onRILRequest

--->onSignalStrengthUnsolResponse

--->setHwVerPorp();

--->return &s\_callbacks;

s\_callbacks

--->onRequest

--->currentState

--->onSupports

--->onCancel

--->getVersion

--->sendCmdSync

--->initVaribales

--->requestTimedCallback

--->&s\_seCallbacks

--->initForSeService

--->getAtrForSeService

--->isCardPresentForSeService

--->transmitForSeService

--->openLogicalChannelForSeService

--->openBasicChannelForSeService

--->closeChannelForSeService

Ril\_service.cpp (g:\code-pj\source\t710-android10\t710-android10-ril\rild\libril) 683588 2022/5/30

Ril\_services分析

radio::registerService

--->获取服务名：android::RIL\_getServiceName()

--->多个sim卡 增加服务名：RIL2\_SERVICE\_NAME

--->s\_vendorFunctions = callbacks;//将vendor ril的接口赋值给vendorFunctions

--->s\_commands = commands;//

--->configureRpcThreadpool(1, true /\* callerWillJoin \*/);

---> radioService[i] = new RadioImpl;

--->extRadioService[i] = new ExtRadioImpl;

--->android::status\_t status = radioService[i]->registerAsService(serviceNames[i]);//注册为hidl服务 绑定式服务

--->status = extRadioService[i]->registerAsService(serviceNames[i]);

--->status = oemHookService[i]->registerAsService(serviceNames[i]);

Ril.cpp

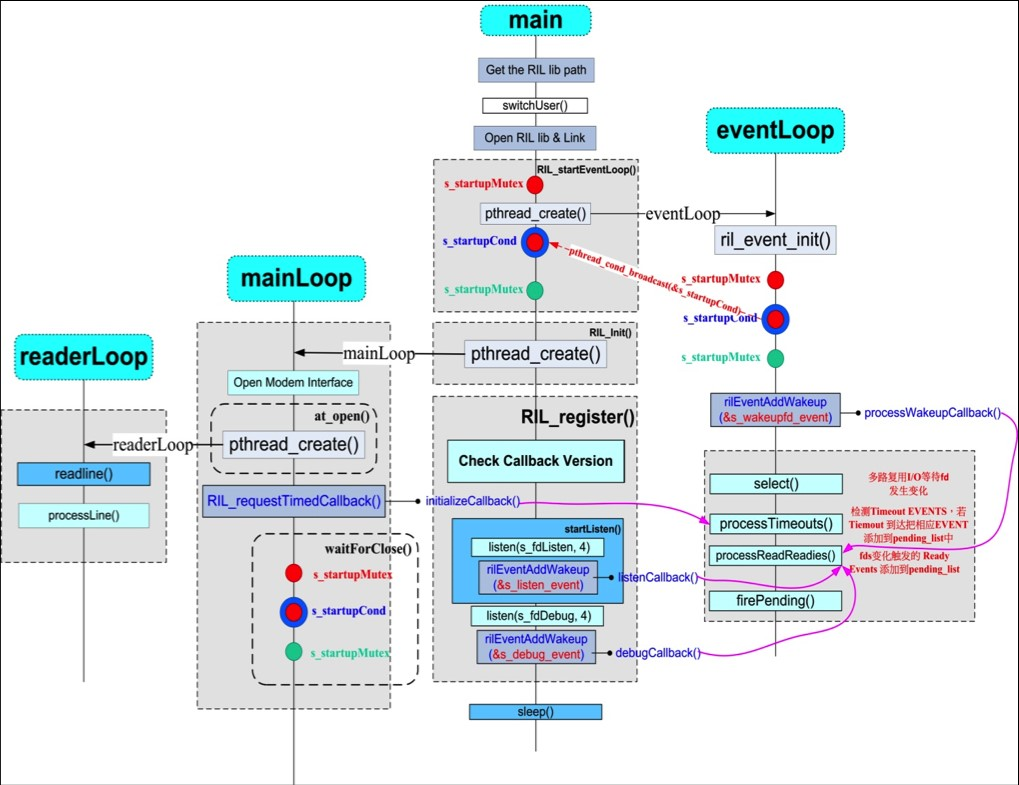
ril\_event\_loop

**--->n = select(nfds, &rfds, NULL, NULL, ptv);**

**--->processTimeouts();**

**--->processReadReadies(&rfds, n);**

**--->firePending();**



**s\_ATSender**

**Ril\_mch.cpp (f:\work\2022-work\t710-pad\work\714资料\ril\_tt\ril\libril) 216064 2022/8/24**

**Ril\_requestAtSender s\_ATSender[RIL\_DEVICE\_MAX\_COUNT];**

**RIL\_startEventLoop**

**--->for(n = 0; n < RIL\_DEVICE\_MAX\_COUNT; n++) {**

**s\_ATSender[n].m\_at\_ch\_name = at\_ch\_name[n];**

**s\_ATSender[n].m\_fd = -1; // fix 2011-09-22**

**//RLOGD("%d, %s, n = %d\n",\_\_LINE\_\_,\_\_FUNCTION\_\_, n);**

**}**

**--->for(n = 0; n < (RIL\_DEVICE\_MAX\_COUNT-1); n++) {**

**ret = pthread\_create(&(s\_ATSender[n].m\_thread\_id), &attr\_request, request\_handler\_thread, &(s\_ATSender[n]));**

**}**

--->**request\_handler\_thread**

**--->while(1)**

**--->processCommandBuffer(pRI->m\_buffer, pRI->m\_buff\_len);**

**--->p.setData((uint8\_t \*) buffer, buflen);**

**--->sendResponse(pErr);**

**--->pRI = (RequestInfo \*)calloc(1, sizeof(RequestInfo));**

**---> pRI->token = token;**

**--->pRI->pCI = &(s\_commands[requestIdx]);**

**--->pRI->pCI->dispatchFunction(p, pRI);**

Android4.4 ril

Rild.c (f:\work\2022-work\t710-pad\work\714资料\anroid4.4\rild) 7980 2022/10/4

int main(int argc, char \*\*argv)

--->switchUser();

--->dlHandle = dlopen(rilLibPath, RTLD\_NOW);//打开vendor ril库

//Ril.cpp (f:\work\2022-work\t710-pad\work\714资料\anroid4.4\libril) 130646 2022/10/4

--->RIL\_startEventLoop();

--->s\_started = 0;

--->pthread\_create(&s\_tid\_dispatch, &attr, eventLoop, NULL);

eventLoop

---> ril\_event\_init();

--->FD\_ZERO(&readFds);

--->init\_list(&timer\_list);

--->init\_list(&pending\_list);

--->memset(watch\_table, 0, sizeof(watch\_table));

--->s\_started = 1;

--->pthread\_cond\_broadcast(&s\_startupCond);

--->pthread\_mutex\_unlock(&s\_startupMutex);

--->ret = pipe(filedes);

--->s\_fdWakeupRead = filedes[0];

---> s\_fdWakeupWrite = filedes[1];

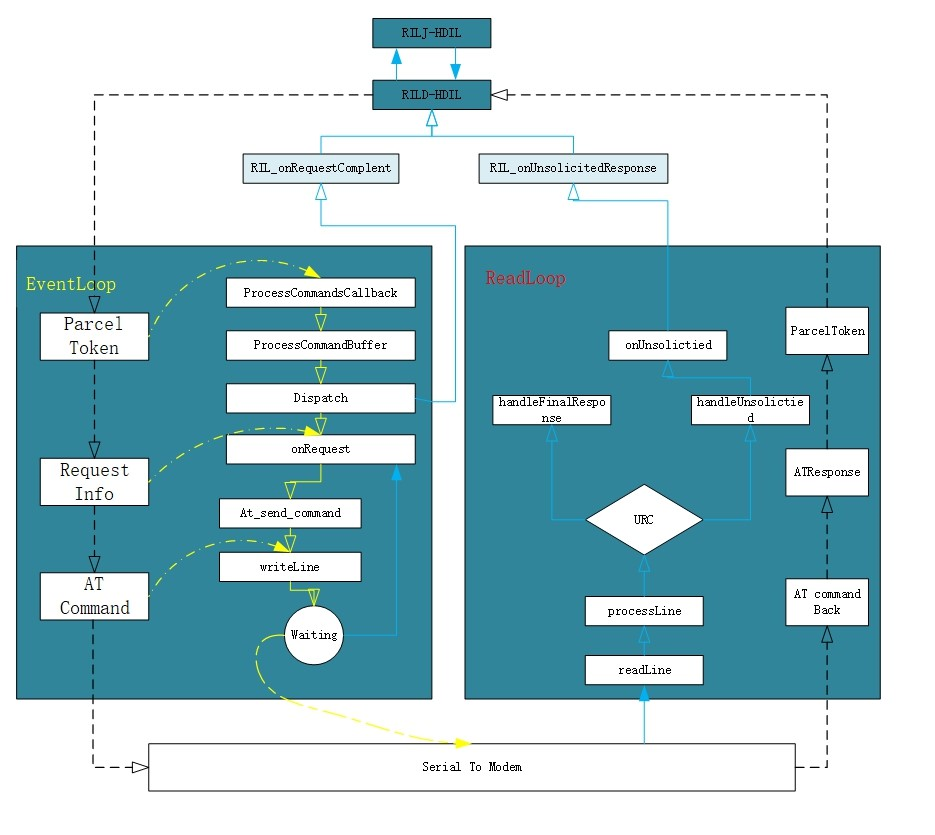
--->fcntl(s\_fdWakeupRead, F\_SETFL, O\_NONBLOCK);

--->ril\_event\_set (&s\_wakeupfd\_event, s\_fdWakeupRead, true,

processWakeupCallback, NULL);

--->memset(ev, 0, sizeof(struct ril\_event));

---> ev->fd = fd;--->s\_fdWakeupRead



--->ev->index = -1;

--->ev->persist = persist;

--->ev->func = func;--->processWakeupCallback

--->do {

ret = read(s\_fdWakeupRead, &buff, sizeof(buff));

} while (ret > 0 || (ret < 0 && errno == EINTR));

--->ev->param = param;

--->fcntl(fd, F\_SETFL, O\_NONBLOCK);

--->rilEventAddWakeup (&s\_wakeupfd\_event);

--->ril\_event\_add(ev);--->s\_wakeupfd\_event

--->for (int i = 0; i < MAX\_FD\_EVENTS; i++) {

if (watch\_table[i] == NULL) {

watch\_table[i] = ev;

ev->index = i;

dlog("~~~~ added at %d ~~~~", i);

dump\_event(ev);

FD\_SET(ev->fd, &readFds);

if (ev->fd >= nfds) nfds = ev->fd+1;

dlog("~~~~ nfds = %d ~~~~", nfds);

break;

}

}

--->triggerEvLoop();

--->if (!pthread\_equal(pthread\_self(), s\_tid\_dispatch)) {

/\* trigger event loop to wakeup. No reason to do this,

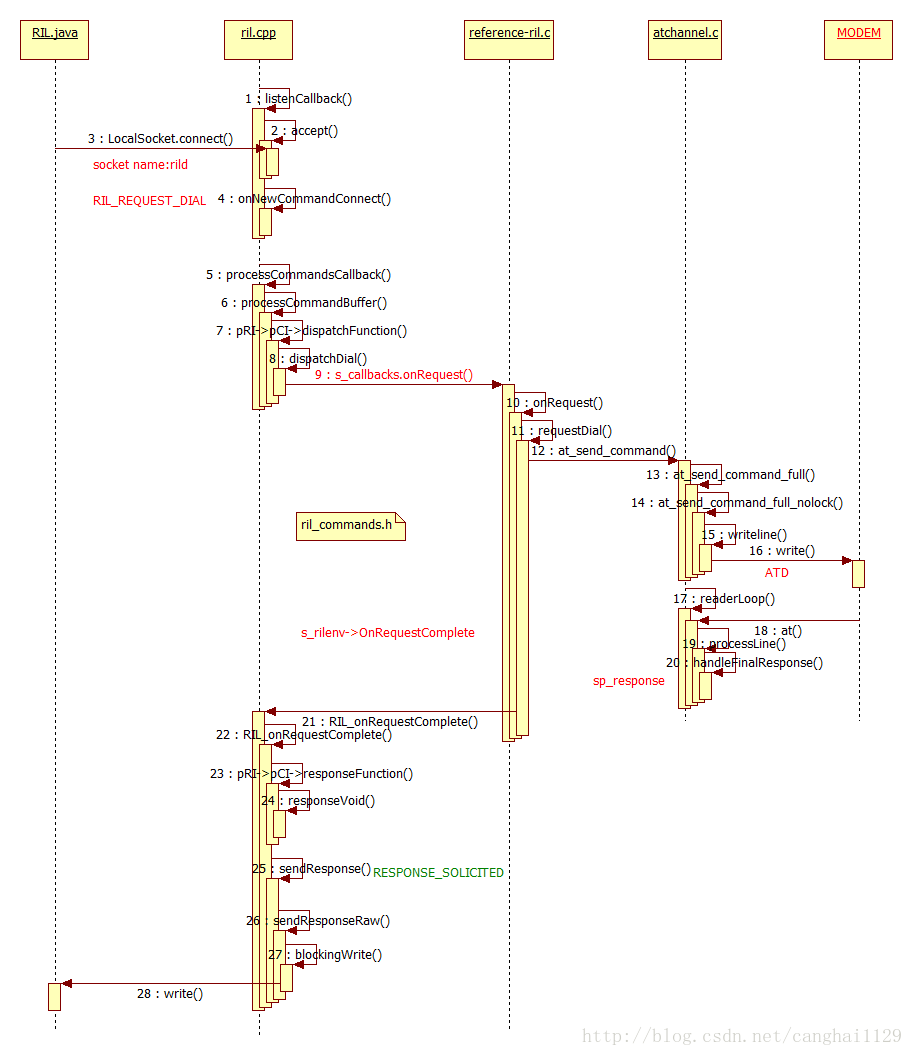
\* if we're in the event loop thread \*/

do {

ret = write (s\_fdWakeupWrite, " ", 1);

} while (ret < 0 && errno == EINTR);

}



--->ril\_event\_loop();

--->

for (;;) {

// make local copy of read fd\_set

memcpy(&rfds, &readFds, sizeof(fd\_set));

if (-1 == calcNextTimeout(&tv)) {

// no pending timers; block indefinitely

dlog("~~~~ no timers; blocking indefinitely ~~~~");

ptv = NULL;

} else {

dlog("~~~~ blocking for %ds + %dus ~~~~", (int)tv.tv\_sec, (int)tv.tv\_usec);

ptv = &tv;

}

printReadies(&rfds);

n = select(nfds, &rfds, NULL, NULL, ptv);

printReadies(&rfds);

dlog("~~~~ %d events fired ~~~~", n);

if (n < 0) {

if (errno == EINTR) continue;

RLOGE("ril\_event: select error (%d)", errno);

// bail?

return;

}

// Check for timeouts

processTimeouts();

--->struct ril\_event \* tev = timer\_list.next;

---> struct ril\_event \* next;

--->while ((tev != &timer\_list) && (timercmp(&now, &tev->timeout, >))) {

// Timer expired

dlog("~~~~ firing timer ~~~~");

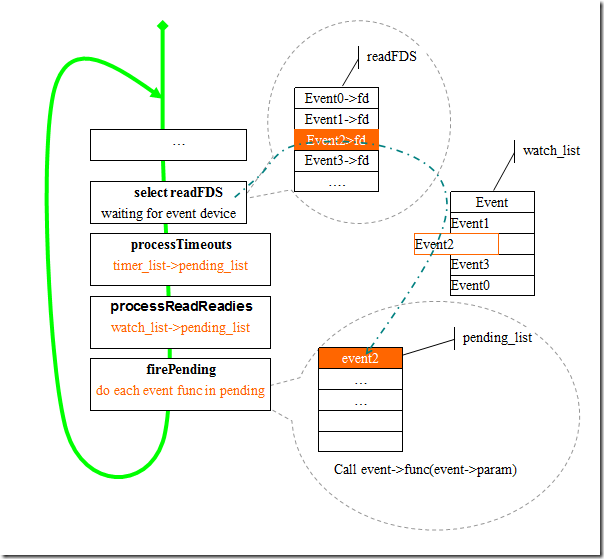
next = tev->next;

removeFromList(tev);

addToList(tev, &pending\_list);

tev = next;

}



// Check for read-ready

processReadReadies(&rfds, n);

--->for (int i = 0; (i < MAX\_FD\_EVENTS) && (n > 0); i++) {

struct ril\_event \* rev = watch\_table[i];

if (rev != NULL && FD\_ISSET(rev->fd, rfds)) {

addToList(rev, &pending\_list);

if (rev->persist == false) {

removeWatch(rev, i);

}

n--;

}

}

// Fire away

firePending();

--->struct ril\_event \* ev = pending\_list.next;

while (ev != &pending\_list) {

struct ril\_event \* next = ev->next;

removeFromList(ev);

ev->func(ev->fd, 0, ev->param);

ev = next;

}

}

获取vendor ril库中的RIL\_Init接口

--->rilInit = (const RIL\_RadioFunctions \*(\*)(const struct RIL\_Env \*, int, char \*\*))dlsym(dlHandle, "RIL\_Init");

--->funcs = rilInit(&s\_rilEnv, argc, rilArgv);//获取vendor ril的func ，vendor ril的初始化参数s\_rilEnv

Reference-ril.c (f:\work\2022-work\t710-pad\work\714资料\anroid4.4\reference-ril) 103111 2022/10/4

--->RIL\_Init

--->ret = pthread\_create(&s\_tid\_mainloop, &attr, mainLoop, NULL);

--->mainLoop

--->fd = open (s\_device\_path, O\_RDWR);

--->ret = at\_open(fd, onUnsolicited);

--->RIL\_requestTimedCallback(initializeCallback, NULL, &TIMEVAL\_0);

---> s\_rilenv->RequestTimedCallback(a,b,c) --->initializeCallback

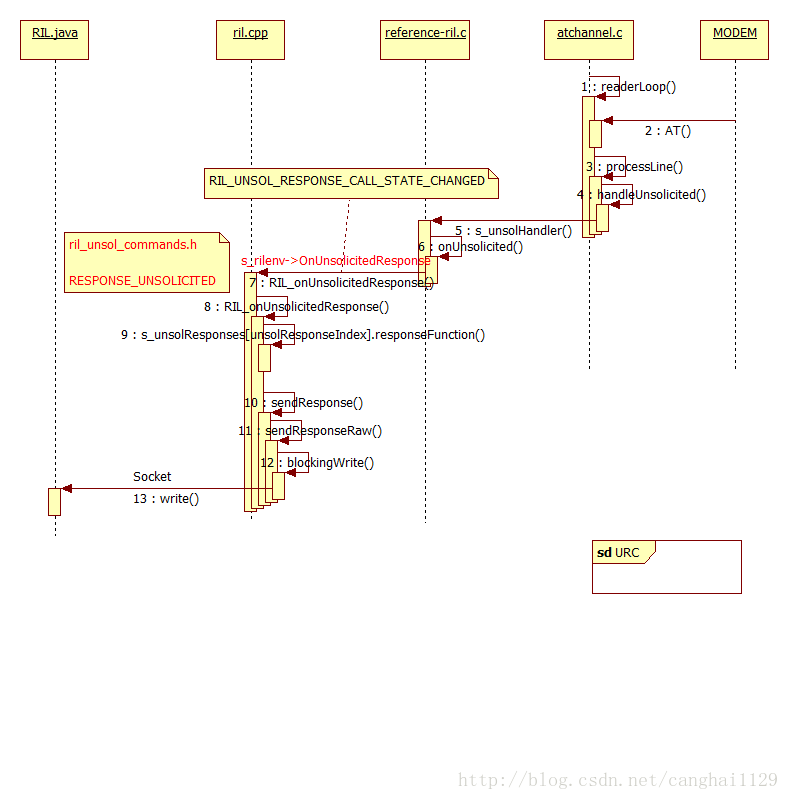
--->setRadioState (RADIO\_STATE\_OFF);

---> at\_send\_command("AT+CGEREP=1,0", NULL);

--->setRadioState (RADIO\_STATE\_ON);

---> waitForClose();

---> return &s\_callbacks;



--->RIL\_VERSION

--->onRequest

--->currentState

--->onSupports

--->onCancel

--->getVersion

--->RIL\_register(funcs);

//Ril.cpp (f:\work\2022-work\t710-pad\work\714资料\anroid4.4\libril) 130646 2022/10/4

RIL\_register

--->memcpy(&s\_callbacks, callbacks, sizeof (RIL\_RadioFunctions));//将vendor ril的callback函数给s\_callbacks

s\_callbacks.onRequest(request, data, len, pRI);调用的是vendor ril中的文件

---> s\_registerCalled = 1;

--->if (s\_started == 0) {

RIL\_startEventLoop();

--->pthread\_create(&s\_tid\_dispatch, &attr, eventLoop, NULL);

--->eventLoop

--->ret = pipe(filedes);

--->s\_fdWakeupRead = filedes[0];

--->s\_fdWakeupWrite = filedes[1];

--->fcntl(s\_fdWakeupRead, F\_SETFL, O\_NONBLOCK);

--->ril\_event\_set (&s\_wakeupfd\_event, s\_fdWakeupRead, true,

processWakeupCallback, NULL);

--->rilEventAddWakeup (&s\_wakeupfd\_event);

--->ril\_event\_loop();

}

--->s\_fdListen = android\_get\_control\_socket(SOCKET\_NAME\_RIL);//创建socket

--->ret = listen(s\_fdListen, 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);

--->ret = fcntl(s\_fdCommand, F\_SETFL, O\_NONBLOCK);

--->p\_rs = record\_stream\_new(s\_fdCommand, MAX\_COMMAND\_BYTES);

---> ril\_event\_set (&s\_commands\_event, s\_fdCommand, 1,

processCommandsCallback, p\_rs);

--->processCommandsCallback

--->for (;;)

--->ret = record\_stream\_get\_next(p\_rs, &p\_record, &recordlen);

--->processCommandBuffer(p\_record, recordlen);

--->pRI->token = token;

--->pRI->pCI = &(s\_commands[request]);//根据request 获取ril\_commands.h的命令

--->pRI->p\_next = s\_pendingRequests;

--->s\_pendingRequests = pRI;

--->pRI->pCI->dispatchFunction(p, pRI);

---->调用vendor ril的函数

--->rilEventAddWakeup (&s\_commands\_event);

--->onNewCommandConnect();

--->rilEventAddWakeup (&s\_listen\_event);

Socket :

#define SOCKET\_NAME\_RIL "rild"

ril\_event有5个

static struct ril\_event s\_commands\_event;

static struct ril\_event s\_wakeupfd\_event;

static struct ril\_event s\_listen\_event;

static struct ril\_event s\_wake\_timeout\_event;

static struct ril\_event s\_debug\_event;

Ril\_event.cpp (f:\work\2022-work\t710-pad\work\714资料\anroid4.4\libril) 10437 2022/10/4

static struct ril\_event \* watch\_table[MAX\_FD\_EVENTS];

static struct ril\_event timer\_list;

static struct ril\_event pending\_list;

714 天通 ril:

F:\work\2022-work\t710-pad\work\714资料\ril\_tt\ril\rild\rild.c

int main(int argc, char \*\*argv)

--->int rilID = -1;

--->根据-s设置相关属性

--->switch (index) {

case 0:

property\_set(RILD\_MSMS\_SIM\_COUNT\_PROPERTY, &ch);

break;

case 1:

property\_set(RILD\_MSMS\_STANDBY\_COUNT\_PROPERTY, &ch);

break;

case 2:

property\_set(RILD\_MSMS\_COMMUNICATION\_COUNT\_PROPERTY, &ch);

break;

}

--->设置Modem类型

--->if(property\_get(PROPERTY\_PERSIST\_SYS\_XT\_MODEM\_TYPE, property\_value\_a, NULL)) {

g\_modemtype = property\_value\_a[0] - '0';

}

--->dlHandle = dlopen(rilLibPath, RTLD\_NOW);

--->RIL\_startEventLoop\_ex(rilID);

Ril\_mch.cpp (f:\work\2022-work\t710-pad\work\714资料\ril\_tt\ril\libril) 216064 2022/8/24

--->sprintf(socket\_ril, "%s%d", SOCKET\_NAME\_RIL, rilID);

--->memcpy(&s\_callbacks, callbacks, sizeof (RIL\_RadioFunctions));

--->s\_registerCalled = 1;

--->if (s\_started == 0) {

-RIL\_startEventLoop\_ex(rilID);

--->for(n = 0; n < RIL\_DEVICE\_MAX\_COUNT; n++) {

if(0 == mrilID) {

s\_ATSender[n].m\_at\_ch\_name = at\_ch\_name[n];

}

else {

s\_ATSender[n].m\_at\_ch\_name = at\_ch\_name[n+RIL\_DEVICE\_MAX\_COUNT];

}

}

--->pthread\_create(&(s\_ATSender[0].m\_thread\_id), &attr\_request, request\_handler\_thread, &(s\_ATSender[0]));

--->request\_handler\_thread(void \*param)

--->Ril\_requestAtSender \*mATSender = (Ril\_requestAtSender \*)param;

--->//接收multiRequstNotify 通知

---> processCommandBuffer(pRI->m\_buffer, pRI->m\_buff\_len);

--->pRI->token = token;

--->pRI->pCI = &(s\_commands[requestIdx]);

--->pRI->pCI->dispatchFunction(p, pRI);--->vendor ril处理

}

--->// start listen socket

--->s\_fdListen = android\_get\_control\_socket(socket\_ril);

--->ret = listen(s\_fdListen, 4);

--->ril\_event\_set (&s\_listen\_event, s\_fdListen, false,

listenCallback, NULL);

--->rilEventAddWakeup (&s\_listen\_event);

--->rilInit = (const RIL\_RadioFunctions \*(\*)(const struct RIL\_Env \*, int, char \*\*))dlsym(dlHandle, "RIL\_Init");

--->shutdownLmiDevices = (void (\*)(int))dlsym(dlHandle, "ata\_ps\_close\_lmi\_in\_shutdown");

--->funcs = rilInit(&s\_rilEnv, argc, rilArgv);

--->signal(RESET\_PROCESS\_SIGNAL, signal\_treatment);

--->RIL\_register\_ex(funcs, rilID);

---> memcpy(&s\_callbacks, callbacks, sizeof (RIL\_RadioFunctions));

--->s\_registerCalled = 1;

--->if (s\_started == 0) {

RIL\_startEventLoop\_ex(rilID);

}

--->s\_fdListen = android\_get\_control\_socket(socket\_ril);

--->ret = listen(s\_fdListen, 4);

--->ril\_event\_set (&s\_listen\_event, s\_fdListen, false,

listenCallback, NULL);

--->s\_fdCommand = accept(s\_fdListen, (sockaddr \*) &peeraddr, &socklen);

---> err = getsockopt(s\_fdCommand, SOL\_SOCKET, SO\_PEERCRED, &creds, &szCreds);

--->ret = fcntl(s\_fdCommand, F\_SETFL, O\_NONBLOCK);

--->p\_rs = record\_stream\_new(s\_fdCommand, MAX\_COMMAND\_BYTES);

--->ril\_event\_set (&s\_commands\_event, s\_fdCommand, 1,

processCommandsCallback, p\_rs);

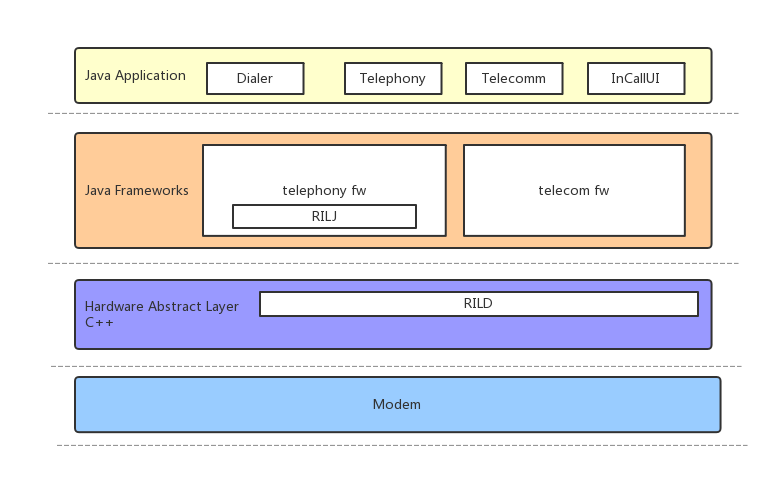
--->rilEventAddWakeup (&s\_commands\_event);

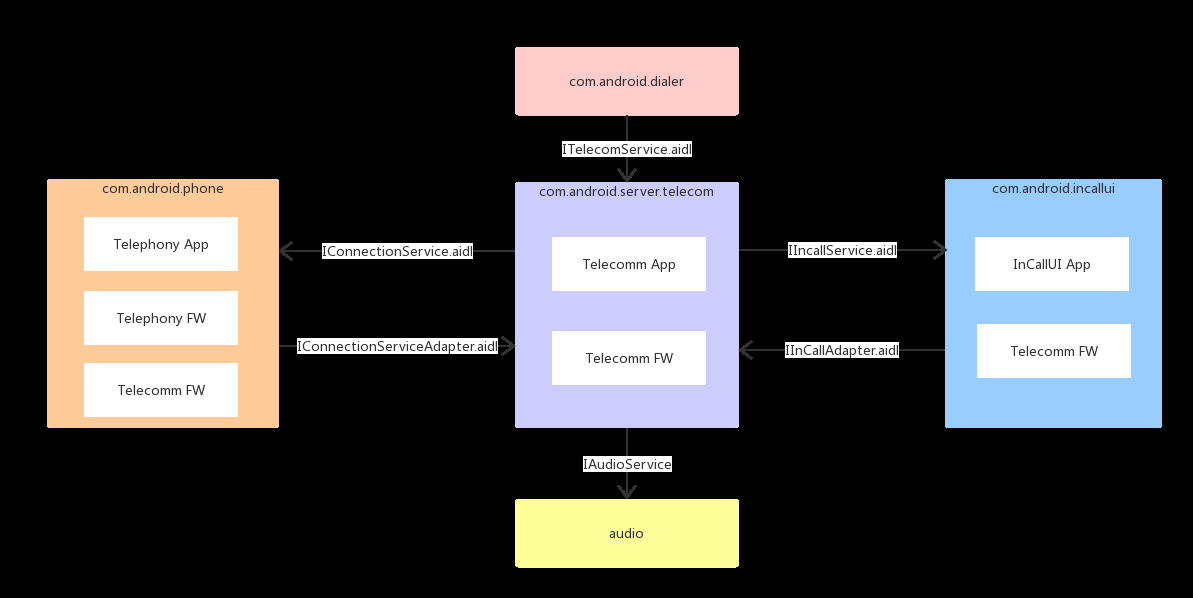
--->onNewCommandConnect();

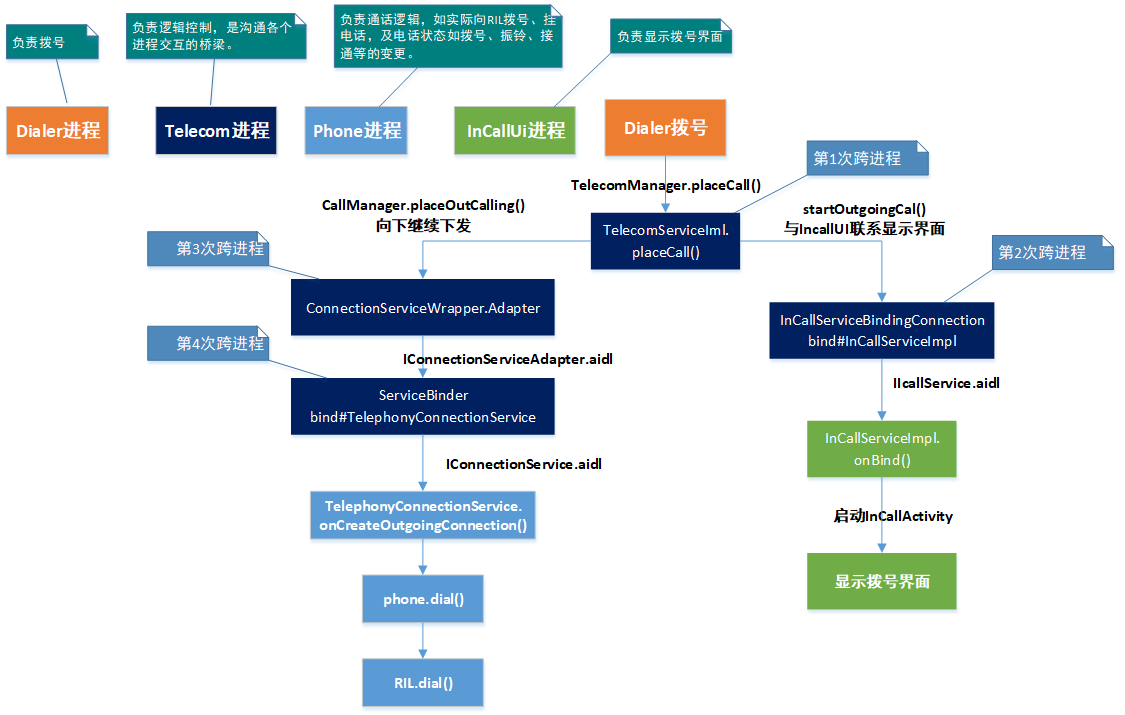
--->rilEventAddWakeup (&s\_listen\_event);

--->create\_parse\_plmn\_th(rilID);

--->







DialpadFragment.java (packages\apps\dialer\java\com\android\dialer\dialpadview) 109651 2022/2/28

拨号按键处理:

public void onClick(View view) {

int resId = view.getId();

if (resId == R.id.dialpad\_floating\_action\_button) {

view.performHapticFeedback(HapticFeedbackConstants.VIRTUAL\_KEY);

/\* SPRD: add for bug731278 & 782352 & 897080 @{ \*/

if (mIsDualVoLTEActive) {

if (mIsSecSimIncall && !ImsManagerEx.isReadyForDualActiveCall()) {

Toast.makeText(getActivity(),

R.string.toast\_not\_dual\_volte\_regist, Toast.LENGTH\_SHORT).show();

return;

}

handleDialButtonPressed(INDEX\_SIM1);

} else {

handleDialButtonPressed();

}

} else if (resId == R.id.dialpad\_floating\_action\_secondary\_button) {

view.performHapticFeedback(HapticFeedbackConstants.VIRTUAL\_KEY);

if (mIsFirstSimIncall && !ImsManagerEx.isReadyForDualActiveCall()) {

Toast.makeText(getActivity(),

R.string.toast\_not\_dual\_volte\_regist, Toast.LENGTH\_SHORT).show();

return;

}

handleDialButtonPressed(INDEX\_SIM2);

/\* @} \*/

} else if (resId == R.id.deleteButton) {

keyPressed(KeyEvent.KEYCODE\_DEL);

} else if (resId == R.id.digits) {

if (!isDigitsEmpty()) {

digits.setCursorVisible(true);

}

} else if (resId == R.id.dialpad\_overflow) {

overflowPopupMenu.show();

/\* SPRD: FEATURE\_WAIT\_PAUSE\_ON\_LONG\_CLICK @{ \*/

} else if (resId == R.id.pound) {

if (!digitsFilledByIntent && getActivity() != null && digits != null

&& digits.getText() != null) {

SpecialCharSequenceMgr.handleAdnEntry(getActivity(), digits.getText()

.toString(), digits);

}

/\* @} \*/

} else {

LogUtil.w("DialpadFragment.onClick", "Unexpected event from: " + view);

}

}

---->

DialpadFragment.java (packages\apps\dialer\java\com\android\dialer\dialpadview) 109651 2022/2/28

private void handleDialButtonPressed(int phoneId) {

List<PhoneAccountHandle> subscriptionAccountHandles =

PhoneAccountUtils.getSubscriptionPhoneAccounts(getActivity());

PhoneAccountHandle account;

int size = 0;

if (subscriptionAccountHandles != null) {

size = subscriptionAccountHandles.size();

}

Log.i(TAG, "handleDialButtonPressed->phoneId:" + phoneId + " size:" + size);

if (subscriptionAccountHandles != null && size > 1 && phoneId < size && mIsDoubleSim) {

account = subscriptionAccountHandles.get(phoneId);

} else {

handleDialButtonPressed();

return;

}

if (isDigitsEmpty()) {

handleDialButtonClickWithEmptyDigits();

} else {

final String number = digits.getText().toString();

if (number != null

&& !TextUtils.isEmpty(prohibitedPhoneNumberRegexp)

&& number.matches(prohibitedPhoneNumberRegexp)) {

Log.i(TAG, "The phone number is prohibited explicitly by a rule.");

if (getActivity() != null) {

DialogFragment dialogFragment = ErrorDialogFragment.newInstance(

R.string.dialog\_phone\_call\_prohibited\_message);

dialogFragment.show(getFragmentManager(), "phone\_prohibited\_dialog");

}

clearDialpad();

} else {

final Intent intent = new CallIntentBuilder(number, CallInitiationType.Type.DIALPAD)

.setPhoneAccountHandle(account)

.build();

DialerUtils.startActivityWithErrorToast(getActivity(), intent);

clearDialpad();

}

}

}

/\* @} \*/

---->

DialerUtils.java (packages\apps\dialer\java\com\android\dialer\util) 23969 2022/2/28

public static void startActivityWithErrorToast(Context context, Intent intent) {

/\* SPRD: Add feature of low battery for Reliance @{ \*/

if (intent.getExtras() != null && VideoProfile.isBidirectional(intent.getExtras().getInt(

TelecomManager.EXTRA\_START\_CALL\_WITH\_VIDEO\_STATE))

&& CallUtil.isBatteryLow(context)) {

CallUtil.showLowBatteryDialDialog(context, intent, true);

} else {

startActivityWithErrorToast(context, intent,

R.string.activity\_not\_available);

}/\* @} \*/

}

--->DialerUtils.java (packages\apps\dialer\java\com\android\dialer\util) 23969 2022/2/28

public static void startActivityWithErrorToast(

final Context context, final Intent intent, int msgId) {

try {

if ((Intent.ACTION\_CALL.equals(intent.getAction()))) {

// All dialer-initiated calls should pass the touch point to the InCallUI

Point touchPoint = TouchPointManager.getInstance().getPoint();

if (touchPoint.x != 0 || touchPoint.y != 0) {

Bundle extras;

// Make sure to not accidentally clobber any existing extras

if (intent.hasExtra(TelecomManager.EXTRA\_OUTGOING\_CALL\_EXTRAS)) {

extras = intent.getParcelableExtra(TelecomManager.EXTRA\_OUTGOING\_CALL\_EXTRAS);

} else {

extras = new Bundle();

}

extras.putParcelable(TouchPointManager.TOUCH\_POINT, touchPoint);

intent.putExtra(TelecomManager.EXTRA\_OUTGOING\_CALL\_EXTRAS, extras);

}

if (shouldWarnForOutgoingWps(context, intent.getData().getSchemeSpecificPart())) {

LogUtil.i(

"DialUtils.startActivityWithErrorToast",

"showing outgoing WPS dialog before placing call");

AlertDialog.Builder builder = new AlertDialog.Builder(context);

builder.setMessage(R.string.outgoing\_wps\_warning);

builder.setPositiveButton(

R.string.dialog\_continue,

new OnClickListener() {

@Override

public void onClick(DialogInterface dialog, int which) {

placeCallOrMakeToast(context, intent);

}

});

builder.setNegativeButton(android.R.string.cancel, null);

builder.create().show();

} else {

placeCallOrMakeToast(context, intent);

}

} else {

context.startActivity(intent);

}

} catch (ActivityNotFoundException e) {

Toast.makeText(context, msgId, Toast.LENGTH\_SHORT).show();

/\*\*UNISOC:modify for the bug 943770 @{\*/

} catch(SecurityException e) {

Toast.makeText(context, R.string.activity\_not\_security, Toast.LENGTH\_SHORT).show();

}

/\*\*@}\*/

}

TelecomUtil.java (packages\apps\dialer\java\com\android\dialer\telecom) 15212 2022/2/28

public static boolean placeCall(Context context, Intent intent) {

// UNISOC: add for bug917377 982631

Uri handle = intent.getData();

boolean isEmergency = PhoneNumberUtils.isEmergencyNumber(handle == null ? "" : handle.getSchemeSpecificPart());

if (!isEmergency && CallList.getInstance().getIncomingCall() != null) {

Toast.makeText(context, R.string.call\_cannot\_be\_sent, Toast.LENGTH\_SHORT).show();

return false;

}

if (hasCallPhonePermission(context)) {

getTelecomManager(context).placeCall(intent.getData(), intent.getExtras());

return true;

}

return false;

}

TelecomManager.java (frameworks\base\telecomm\java\android\telecom) 76994 2022/2/28

public boolean isInManagedCall() {

try {

if (isServiceConnected()) {

return getTelecomService().isInManagedCall(mContext.getOpPackageName());

}

} catch (RemoteException e) {

Log.e(TAG, "RemoteException calling isInManagedCall().", e);

}

return false;

}

private ITelecomService getTelecomService() {

if (mTelecomServiceOverride != null) {

return mTelecomServiceOverride;

}

return ITelecomService.Stub.asInterface(ServiceManager.getService(Context.TELECOM\_SERVICE));

}

InCallService.java (frameworks\base\telecomm\java\android\telecom) 33066 2022/2/28

class InCallServiceImpl extends InCallService

类InCallServiceImpl 继承 InCallService

服务如何绑定 如何通信

InCallServiceImpl 的作用：接收Telecom组件的的电话更新 服务绑定telecom

public class InCallServiceImpl extends InCallService {

private NewReturnToCallController newReturnToCallController;

private CallList.Listener feedbackListener;

private SpeakEasyCallManager speakEasyCallManager;

@Override

public void onBringToForeground(boolean showDialpad) {

Trace.beginSection("InCallServiceImpl.onBringToForeground");

InCallPresenter.getInstance().onBringToForeground(showDialpad);

Trace.endSection();

}

@Override

public void onCallAdded(Call call) {

Trace.beginSection("InCallServiceImpl.onCallAdded");

InCallPresenter.getInstance().onCallAdded(call);

Trace.endSection();

}

@Override

public IBinder onBind(Intent intent) {

Trace.beginSection("InCallServiceImpl.onBind");

final Context context = getApplicationContext();

final ContactInfoCache contactInfoCache = ContactInfoCache.getInstance(context);

AudioModeProvider.getInstance().initializeAudioState(this);

InCallPresenter.getInstance()

.setUp(

context,

CallList.getInstance(),

new ExternalCallList(),

new StatusBarNotifier(context, contactInfoCache),

new ExternalCallNotifier(context, contactInfoCache),

contactInfoCache,

new ProximitySensor(

context, AudioModeProvider.getInstance(), new AccelerometerListener(context)),

new FilteredNumberAsyncQueryHandler(context));

InCallPresenter.getInstance().onServiceBind();

InCallPresenter.getInstance().maybeStartRevealAnimation(intent);

TelecomAdapter.getInstance().setInCallService(this);

if (NewReturnToCallController.isEnabled(this)) {

newReturnToCallController =

new NewReturnToCallController(this, ContactInfoCache.getInstance(context));

}

feedbackListener = FeedbackComponent.get(context).getCallFeedbackListener();

CallList.getInstance().addListener(feedbackListener);

IBinder iBinder = super.onBind(intent);

Trace.endSection();

return iBinder;

}

}

启动服务:

启动服务方式一:

Intent intentOne = new Intent(this, TestOneService.class);

startService(intentOne);

启动服务方式二:

Intent intent = new Intent(this, TestTwoService.class);

intent.putExtra("from", "ActivityB");

bindService(intent, conn, BIND\_AUTO\_CREATE);

Class InCallServiceImpl 的绑定:

UiCallManager.java (packages\apps\car\dialer\src\com\android\car\dialer\telecom) 28334 2022/2/28

private UiCallManager(Context context) {

Intent intent = new Intent(context, InCallServiceImpl.class);

intent.setAction(InCallServiceImpl.ACTION\_LOCAL\_BIND);

context.bindService(intent, mInCallServiceConnection, Context.BIND\_AUTO\_CREATE);

}

Class:TelecomServiceImpl

--->interface:SubscriptionManagerAdapter //这是一个接口 实现一个方法：**getDefaultVoiceSubId**

**--->getDefaultVoiceSubId();**

**--->class:SubscriptionManagerAdapterImpl 继承SubscriptionManagerAdapter接口 并具体实现该接口**

**public int getDefaultVoiceSubId() {**

**return SubscriptionManager.getDefaultVoiceSubscriptionId();**

**}**

**---> private final ITelecomService.Stub mBinderImpl = new ITelecomService.Stub() { //对服务的具体实现**

**--->getDefaultOutgoingPhoneAccount**

**--->getUserSelectedOutgoingPhoneAccount**

**--->setUserSelectedOutgoingPhoneAccount**

**--->getCallCapablePhoneAccounts**

**--->getSelfManagedPhoneAccounts**

**--->getPhoneAccountsSupportingScheme**

**--->getPhoneAccountsForPackage**

**--->getPhoneAccount**

**--->getAllPhoneAccountsCount**

**--->getAllPhoneAccounts**

**--->getAllPhoneAccountHandles**

**--->getSimCallManager**

**--->getSimCallManagerForUser**

**--->registerPhoneAccount**

**--->unregisterPhoneAccount**

**--->clearAccounts**

**--->isVoiceMailNumber**

**--->getVoiceMailNumber**

**--->getLine1Number**

**--->silenceRinger**

**--->getDefaultPhoneApp**

**--->getDefaultDialerPackage**

**--->getSystemDialerPackage**

**--->isInCall**

**--->isInManagedCall**

**--->isRinging**

**--->getCallState**

**--->endCall**

**--->acceptRingingCall**

**--->acceptRingingCallWithVideoState**

**--->showInCallScreen**

**--->cancelMissedCallsNotification**

**--->handlePinMmi**

**--->handlePinMmiForPhoneAccount**

**--->getAdnUriForPhoneAccount**

**--->isTtySupported**

**---->getCurrentTtyMode**

**--->addNewIncomingCall**

**--->acceptHandover**

**--->addNewUnknownCall**

**--->placeCall**

**--->enablePhoneAccount**

**--->setDefaultDialer**

**--->TelecomAnalytics**

**--->createManageBlockedNumbersIntent**

**--->isOutgoingCallPermitted**

**--->waitOnHandlers**

**--->**

**}**

enforceAnswerCallPermission

enforceAcceptHandoverPermission

TelecomServiceImpl类的实例://完成初始化工作

public TelecomServiceImpl(

Context context,

CallsManager callsManager,

PhoneAccountRegistrar phoneAccountRegistrar,

CallIntentProcessor.Adapter callIntentProcessorAdapter,

UserCallIntentProcessorFactory userCallIntentProcessorFactory,

DefaultDialerCache defaultDialerCache,

SubscriptionManagerAdapter subscriptionManagerAdapter,

TelecomSystem.SyncRoot lock) {

mContext = context;

mAppOpsManager = (AppOpsManager) mContext.getSystemService(Context.APP\_OPS\_SERVICE);

mPackageManager = mContext.getPackageManager();

mCallsManager = callsManager;

mLock = lock;

mPhoneAccountRegistrar = phoneAccountRegistrar;

mUserCallIntentProcessorFactory = userCallIntentProcessorFactory;

mDefaultDialerCache = defaultDialerCache;

mCallIntentProcessorAdapter = callIntentProcessorAdapter;

mSubscriptionManagerAdapter = subscriptionManagerAdapter;

}

public ITelecomService.Stub getBinder() {

return mBinderImpl;//有完成具体方法的实现

}

--->isPhoneAccountHandleVisibleToCallingUser

--->isCallerSystemApp

--->isPackageSystemApp

--->acceptRingingCallInternal

--->endCallInternal

--->enforcePhoneAccountIsRegisteredEnabled

--->enforcePhoneAccountModificationForPackage

--->enforcePermissionOrPrivilegedDialer

--->enforceCallingPackage

--->enforceConnectionServiceFeature

--->

private TelephonyManager getTelephonyManager() {

return (TelephonyManager) mContext.getSystemService(Context.TELEPHONY\_SERVICE);

}

TelecomService.java (packages\services\telecomm\src\com\android\server\telecom\components) 10447 2022/2/28

TelecomService.java (packages\services\telecomm\src\com\android\server\telecom\components) 10447 2022/2/28

Class : public class TelecomService extends Service implements TelecomSystem.Component: 作用:Implementation of the ITelecom interface.

{

public IBinder onBind(Intent intent) {

Log.d(this, "onBind");

initializeTelecomSystem(this);

synchronized (getTelecomSystem().getLock()) {

return getTelecomSystem().getTelecomServiceImpl().getBinder(); //实例化TelecomServiceImpl

}

}

static void initializeTelecomSystem(Context context) {

elecomSystem.setInstance(

new TelecomSystem(

---->new MissedCallNotifierImpl.MissedCallNotifierImplFactory()

--->new CallerInfoAsyncQueryFactory()

--->new HeadsetMediaButtonFactory()

--->new ProximitySensorManagerFactory()

--->new InCallWakeLockControllerFactory()

--->new CallAudioManager.AudioServiceFactory()

--->new BluetoothPhoneServiceImpl.BluetoothPhoneServiceImplFactory()

--->new ConnectionServiceFocusManager

.ConnectionServiceFocusManagerFactory()

--->

)

)

}

public TelecomSystem getTelecomSystem() {

return TelecomSystem.getInstance();

}

}

TelecomSystem.java (packages\services\telecomm\src\com\android\server\telecom) 15832 2022/2/28

public class TelecomSystem {

public interface Component { //这个接口需要由继承的类去实现

TelecomSystem getTelecomSystem();

}

private static final IntentFilter DIALER\_SECRET\_CODE\_FILTER;//暗码

public static TelecomSystem getInstance() {

return INSTANCE;

}

public static void setInstance(TelecomSystem instance) {

if (INSTANCE != null) {

Log.w("TelecomSystem", "Attempt to set TelecomSystem.INSTANCE twice");

}

Log.i(TelecomSystem.class, "TelecomSystem.INSTANCE being set");

INSTANCE = instance;

}

public TelecomSystem(){

--->new DefaultDialerCache.DefaultDialerManagerAdapterImpl();

---> new DefaultDialerCache(mContext,

defaultDialerAdapter, mLock);

--->new PhoneAccountRegistrar(mContext, defaultDialerCache,

--->new ContactsAsyncHelper(

--->new BluetoothDeviceManager(mContext

--->new BluetoothRouteManager(mContext, mLock,

--->new BluetoothStateReceiver(

bluetoothDeviceManager, bluetoothRouteManager);

--->wiredHeadsetManager = new WiredHeadsetManager(mContext);

--->systemStateProvider = new SystemStateProvider(mContext);

--->emergencyCallHelper = new EmergencyCallHelper(mContext,

mContext.getResources().getString(R.string.ui\_default\_package), timeoutsAdapter);

--->InCallControllerFactory inCallControllerFactory = new InCallControllerFactory() {

----> return new InCallController(context, lock, callsManager, systemStateProvider,

defaultDialerCache, timeoutsAdapter, emergencyCallHelper, clockProxy);

}

}

mCallsManager = new CallsManager;

incomingCallNotifier.setCallsManagerProxy(new IncomingCallNotifier.CallsManagerProxy()

--->mCallsManager.setIncomingCallNotifier(mIncomingCallNotifier);

--->mRespondViaSmsManager = new RespondViaSmsManager(mCallsManager, mLock);

--->mCallsManager.setRespondViaSmsManager(mRespondViaSmsManager);

---> mDialerCodeReceiver = new DialerCodeReceiver(mCallsManager);

---> mContext.registerReceiver(mDialerCodeReceiver, DIALER\_SECRET\_CODE\_FILTER,

Manifest.permission.CONTROL\_INCALL\_EXPERIENCE, null);

--->mTelecomServiceImpl = new TelecomServiceImpl(

mContext, mCallsManager, mPhoneAccountRegistrar,

new CallIntentProcessor.AdapterImpl(),

new UserCallIntentProcessorFactory() {

@Override

public UserCallIntentProcessor create(Context context, UserHandle userHandle) {

return new UserCallIntentProcessor(context, userHandle);

}

},

defaultDialerCache,

new TelecomServiceImpl.SubscriptionManagerAdapterImpl(),

mLock);

Log.endSession();

}

getPhoneAccountRegistrar

getCallsManager

getBluetoothPhoneServiceImpl

getCallIntentProcessor

getTelecomBroadcastIntentProcessor

getTelecomServiceImpl

getLock

isBootComplete

}

private final ITelecomService.Stub mBinderImpl = new ITelecomService.Stub() {

@Override

public void placeCall(Uri handle, Bundle extras, String callingPackage) {

try {

synchronized (mLock) {

final UserHandle userHandle = Binder.getCallingUserHandle();

long token = Binder.clearCallingIdentity();

try {

final Intent intent = new Intent(Intent.ACTION\_CALL, handle);

mUserCallIntentProcessorFactory.create(mContext, userHandle)

.processIntent(

intent, callingPackage, hasCallAppOp && hasCallPermission);

//这个地方的实现：

TelecomSystem.java (packages\services\telecomm\src\com\android\server\telecom) 15832 2022/2/28

new UserCallIntentProcessorFactory() {

@Override

public UserCallIntentProcessor create(Context context, UserHandle userHandle) {

return new UserCallIntentProcessor(context, userHandle);

}

}

}

}

}

}

}

UserCallIntentProcessor.java (packages\services\telecomm\src\com\android\server\telecom\components) 10677 2022/2/28

-->TelecomSystem.getInstance().getCallIntentProcessor().processIntent(intent);

--->mCallIntentProcessor = new CallIntentProcessor(mContext, mCallsManager);

**-->CallIntentProcessor.java (packages\services\telecomm\src\com\android\server\telecom) 17296 2022/2/28**

**public void processIntent(Intent intent) {**

**final boolean isUnknownCall = intent.getBooleanExtra(KEY\_IS\_UNKNOWN\_CALL, false);**

**Log.i(this, "onReceive - isUnknownCall: %s", isUnknownCall);**

**Trace.beginSection("processNewCallCallIntent");**

**if (isUnknownCall) {**

**processUnknownCallIntent(mCallsManager, intent);**

**} else {**

**processOutgoingCallIntent(mContext, mCallsManager, intent);**

**}**

**Trace.endSection();**

**}**

=

CallIntentProcessor.java (packages\services\telecomm\src\com\android\server\telecom) 17296 2022/2/28

static void processOutgoingCallIntent(

{

---->Call call = callsManager

.startOutgoingCall(handle, phoneAccountHandle, clientExtras, initiatingUser, //更新UI更新 ui 界面

intent);

--->CallsManager.java (packages\services\telecomm\src\com\android\server\telecom) 203998 2022/2/28

startOutgoingCall(){

---->call = new Call(getNextCallId(), mContext,

this,

mLock,

mConnectionServiceRepository,

mContactsAsyncHelper,

mCallerInfoAsyncQueryFactory,

mPhoneNumberUtilsAdapter,

handle,

null /\* gatewayInfo \*/,

null /\* connectionManagerPhoneAccount \*/,

null /\* phoneAccountHandle \*/,

Call.CALL\_DIRECTION\_OUTGOING /\* callDirection \*/,

false /\* forceAttachToExistingConnection \*/,

false, /\* isConference \*/

mClockProxy);

--->addCall(call);

}

---> sendNewOutgoingCallIntent(context, call, callsManager, intent);//将上层信息下发到telephony， ril

}

static void sendNewOutgoingCallIntent(Context context, Call call, CallsManager callsManager,

Intent intent) {

// Asynchronous calls should not usually be made inside a BroadcastReceiver because once

// onReceive is complete, the BroadcastReceiver's process runs the risk of getting

// killed if memory is scarce. However, this is OK here because the entire Telecom

// process will be running throughout the duration of the phone call and should never

// be killed.

final boolean isPrivilegedDialer = intent.getBooleanExtra(KEY\_IS\_PRIVILEGED\_DIALER, false);

NewOutgoingCallIntentBroadcaster broadcaster = new NewOutgoingCallIntentBroadcaster(

context, callsManager, call, intent, callsManager.getPhoneNumberUtilsAdapter(),

isPrivilegedDialer);

final int result = broadcaster.processIntent();

--->placeOutgoingCallImmediately(mCall, callingAddress, null,

speakerphoneOn, videoState);

--->mCall.setNewOutgoingCallIntentBroadcastIsDone();

--->mCallsManager.placeOutgoingCall(call, handle, gatewayInfo, speakerphoneOn, videoState);

--->call.startCreateConnection(mPhoneAccountRegistrar);

--->

final boolean success = result == DisconnectCause.NOT\_DISCONNECTED;

if (!success && call != null) {

disconnectCallAndShowErrorDialog(context, call, result);

}

}

**先执行了 startOutgoingCall 方法 ，然后再 执行 broadcaster. processCall () 方法**

**Call.java (packages\services\telecomm\src\com\android\server\telecom) 121810 2022/2/28**

**public void setNewOutgoingCallIntentBroadcastIsDone() {**

**mIsNewOutgoingCallIntentBroadcastDone = true;**

**}**

接收modem消息的流程：

RadioIndication.java (frameworks\opt\telephony\src\java\com\android\internal\telephony) 43411 2022/2/28

public class RadioIndication extends IRadioIndication.Stub { //集成HIDL

public void radioStateChanged(int indicationType, int radioState) {

mRil.processIndication(indicationType);

CommandsInterface.RadioState newState = getRadioStateFromInt(radioState);

if (RIL.RILJ\_LOGD) {

mRil.unsljLogMore(RIL\_UNSOL\_RESPONSE\_RADIO\_STATE\_CHANGED, "radioStateChanged: " +

newState);

}

mRil.setRadioState(newState);

}

}

public void callStateChanged(int indicationType) {

mRil.processIndication(indicationType);

if (RIL.RILJ\_LOGD) mRil.unsljLog(RIL\_UNSOL\_RESPONSE\_CALL\_STATE\_CHANGED);

/\*\* UNISOC: kill-stop mechanism BEGIN \*/

sendKillStopToAms();

mRil.mCallStateRegistrants.notifyRegistrants();

}

modem发送的消息后 会自动调用callStateChanged 谁会监听这个消息？

注册者调用:

@Override

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

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

mCallStateRegistrants.add(r);

}

GsmCdmaCallTracker.java (frameworks\opt\telephony\src\java\com\android\internal\telephony) 71261 2022/2/28

这个类注册了 要观察的消息

public GsmCdmaCallTracker (GsmCdmaPhone phone) {

this.mPhone = phone;

mCi = phone.mCi;

mCi.registerForCallStateChanged(this, EVENT\_CALL\_STATE\_CHANGE, null);

mCi.registerForOn(this, EVENT\_RADIO\_AVAILABLE, null);

mCi.registerForNotAvailable(this, EVENT\_RADIO\_NOT\_AVAILABLE, null);

// Register receiver for ECM exit

IntentFilter filter = new IntentFilter();

filter.addAction(TelephonyIntents.ACTION\_EMERGENCY\_CALLBACK\_MODE\_CHANGED);

mPhone.getContext().registerReceiver(mEcmExitReceiver, filter);

updatePhoneType(true);

}

注册监听后，当Modem发送消息过来 GsmCdmaCallTracker 处理该消息的的地方？

public void handleMessage(Message msg) {

case EVENT\_CALL\_STATE\_CHANGE:

pollCallsWhenSafe();

---》mCi.getCurrentCalls(mLastRelevantPoll);//获取call的相关信息 如电话号码

--》radioProxy.getCurrentCalls(rr.mSerial);//RIL.java (frameworks\opt\telephony\src\java\com\android\internal\telephony) 225657 2022/2/28

---->等待modem返回状态

等到modem返回消息后 调用的函数：Modem返回RIL\_REQUEST\_GET\_CURRENT\_CALLS 并最终调用RadioResponse.java的getCurrentCallsResponse

{RIL\_REQUEST\_GET\_CURRENT\_CALLS, radio::getCurrentCallsResponse},

RadioResponse.java (frameworks\opt\telephony\src\java\com\android\internal\telephony) 85065 2022/2/28

--->getCurrentCallsResponse

--->esponseCurrentCalls(responseInfo, calls);

--->dc.number = calls.get(i).number;//通过底层的消息创建dcCalls 即call的状态信息

--->sendMessageResponse(rr.mResult, dcCalls);//告诉ril已经处理好了

--->mRil.processResponseDone(rr, responseInfo, dcCalls);

break;

case EVENT\_POLL\_CALLS\_RESULT:

handlePollCalls((AsyncResult)msg.obj);

其先对底层反馈的消息进行解析，获取其通话状态，判断如果是来电则发出notifyNewRingingConnection响铃消息通知，然后进行一些通话断开连接的操作及更新phone状态。继续跟进notifyNewRingingConnection响铃消息，该消息调用的是phone的方法，往上追溯。

---> mPhone.notifyNewRingingConnection(newRinging);//铃声通知 --->这个消息铃声通知到哪里？搜索技巧 notifyNewRingingConnection notifyxxxxx 是一种观察者模式 注册它的地方:notifyxxx registerForNewRingingConnection

PstnIncomingCallNotifier.java (packages\services\telephony\src\com\android\services\telephony) 16550 2022/2/28

private void registerForNotifications() {

if (mPhone != null) {

Log.i(this, "Registering: %s", mPhone);

mPhone.registerForNewRingingConnection(mHandler, EVENT\_NEW\_RINGING\_CONNECTION, null);

mPhone.registerForCallWaiting(mHandler, EVENT\_CDMA\_CALL\_WAITING, null);

mPhone.registerForUnknownConnection(mHandler, EVENT\_UNKNOWN\_CONNECTION, null);

}

}

private final Handler mHandler = new Handler() {

@Override

public void handleMessage(Message msg) {

switch(msg.what) {

case EVENT\_NEW\_RINGING\_CONNECTION:

handleNewRingingConnection((AsyncResult) msg.obj);

break;

case EVENT\_CDMA\_CALL\_WAITING:

handleCdmaCallWaiting((AsyncResult) msg.obj);

break;

case EVENT\_UNKNOWN\_CONNECTION:

handleNewUnknownConnection((AsyncResult) msg.obj);

break;

default:

break;

}

}

};

---->注册通知的地方：

mIncomingCallNotifier = new PstnIncomingCallNotifier((Phone) mPhone);

mPhoneCapabilitiesNotifier = new PstnPhoneCapabilitiesNotifier((Phone) mPhone,

this);

CallsManager.java (packages\services\telecomm\src\com\android\server\telecom) 203998 2022/2/28

public void setIncomingCallNotifier(IncomingCallNotifier incomingCallNotifier) {

if (mIncomingCallNotifier != null) {

mListeners.remove(mIncomingCallNotifier);

}

mIncomingCallNotifier = incomingCallNotifier;

mListeners.add(mIncomingCallNotifier);

} 在CallsManager实例化的时候调用

case EVENT\_NEW\_RINGING\_CONNECTION:

handleNewRingingConnection((AsyncResult) msg.obj);

--->Call call = connection.getCall()

if (call != null && call.getState().isRinging()) {

sendIncomingCallIntent(connection);

--->TelecomManager.from(mPhone.getContext()).addNewIncomingCall(handle, extras);

TelecomManager.java (frameworks\base\telecomm\java\android\telecom) 76994 2022/2/28

public void addNewIncomingCall(PhoneAccountHandle phoneAccount, Bundle extras) {

try {

if (isServiceConnected()) {

if (extras != null && extras.getBoolean(EXTRA\_IS\_HANDOVER) &&

mContext.getApplicationContext().getApplicationInfo().targetSdkVersion >

Build.VERSION\_CODES.O\_MR1) {

Log.e("TAG", "addNewIncomingCall failed. Use public api " +

"acceptHandover for API > O-MR1");

// TODO add "return" after DUO team adds support for new handover API

}

getTelecomService().addNewIncomingCall(phoneAccount, extras == null ? new Bundle() : extras);

TelecomServiceImpl.java (packages\services\telecomm\src\com\android\server\telecom) 76032 2022/2/28

----> public void addNewIncomingCall(PhoneAccountHandle phoneAccountHandle, Bundle extras) {

try {

Intent intent = new Intent(TelecomManager.ACTION\_INCOMING\_CALL);

intent.putExtra(TelecomManager.EXTRA\_PHONE\_ACCOUNT\_HANDLE,

phoneAccountHandle);

intent.putExtra(CallIntentProcessor.KEY\_IS\_INCOMING\_CALL, true);

if (extras != null) {

extras.setDefusable(true);

intent.putExtra(TelecomManager.EXTRA\_INCOMING\_CALL\_EXTRAS, extras);

}

mCallIntentProcessorAdapter.processIncomingCallIntent(

CallIntentProcessor.java (packages\services\telecomm\src\com\android\server\telecom) 17296 2022/2/28

--->callsManager.processIncomingCallIntent(phoneAccountHandle, clientExtras);

CallsManager.java (packages\services\telecomm\src\com\android\server\telecom) 203998 2022/2/28

void processIncomingCallIntent(PhoneAccountHandle phoneAccountHandle, Bundle extras) {

---> Call call = new Call(xxx);

--->call.addListener(this);//建立监听

---> call.setStartWithSpeakerphoneOn(true);

---> call.getTargetPhoneAccount())

--->call.startCreateConnection(mPhoneAccountRegistrar);//建立连接

Call.java (packages\services\telecomm\src\com\android\server\telecom) 121810 2022/2/28

void startCreateConnection(PhoneAccountRegistrar phoneAccountRegistrar) {

mCreateConnectionProcessor = new CreateConnectionProcessor(this, mRepository, this,

phoneAccountRegistrar, mContext);

mCreateConnectionProcessor.process();

CreateConnectionProcessor.java (packages\services\telecomm\src\com\android\server\telecom) 20945 2022/2/28

public void process() {

Log.v(this, "process");

clearTimeout();

mAttemptRecords = new ArrayList<>();

if (mCall.getTargetPhoneAccount() != null) {

mAttemptRecords.add(new CallAttemptRecord(

mCall.getTargetPhoneAccount(), mCall.getTargetPhoneAccount()));

}

if (!mCall.isSelfManaged()) {

adjustAttemptsForConnectionManager();

adjustAttemptsForEmergency(mCall.getTargetPhoneAccount());

}

mAttemptRecordIterator = mAttemptRecords.iterator();

attemptNextPhoneAccount();

---> mCall.setConnectionService(mService);

--->mService.createConnection(mCall, CreateConnectionProcessor.this);

ConnectionServiceWrapper.java (packages\services\telecomm\src\com\android\server\telecom) 75321 2022/2/28

--->mServiceInterface.createConnection(

call.getConnectionManagerPhoneAccount(),

callId,

connectionRequest,

call.shouldAttachToExistingConnection(),

call.isUnknown(),

Log.getExternalSession());

}

--->mBinder.bind(callback, call);

ServiceBinder.java (packages\services\telecomm\src\com\android\server\telecom) 14595 2022/2/28

void bind(BindCallback callback, Call call) {

--->mCallbacks.add(callback);

---> isBound =isBound = mContext.bindServiceAsUser(serviceIntent, connection, bindingFlags,

mUserHandle);

--->Preconditions.checkNotNull(mBinder);

--->handleSuccessfulConnection();

}

}

}

mCallsManager, intent);

} finally {

Binder.restoreCallingIdentity(token);

}

}

} catch (RemoteException e) {

Log.e(TAG, "RemoteException adding a new incoming call: " + phoneAccount, e);

}

}

}

---> mNewRingingConnectionRegistrants.notifyRegistrants(ar);

---->updatePhoneState();//更新phone状态

}

ril接收返回

mRadioResponse = new RadioResponse(this);

mRadioIndication = new RadioIndication(this);

RadioResponse.java (frameworks\opt\telephony\src\java\com\android\internal\telephony) 85065 2022/2/28

**中处理查询的返回**

**public void getSignalStrengthResponse(RadioResponseInfo responseInfo,**

**android.hardware.radio.V1\_0.SignalStrength sigStrength) {**

**responseSignalStrength(responseInfo, sigStrength);**

**}--》这个方法是对hidl的具体实现**

processResponse(Parcel)

RadioResponse.java (frameworks\opt\telephony\src\java\com\android\internal\telephony) 85065 2022/2/28

public void getSignalStrengthResponse(RadioResponseInfo responseInfo,//这个函数有底层直接调用 有数据时返回

搜索方法：发送这个请求的 一定有handleMessage 并处理EVENT\_CALL\_RING

android.hardware.radio.V1\_0.SignalStrength sigStrength) {

responseSignalStrength(responseInfo, sigStrength);

}

getSignalStrength 对应的返回是responseSignalStrength 找到getSignalStrength 所在文件 一定有handleMessage处理responseSignalStrength的消息