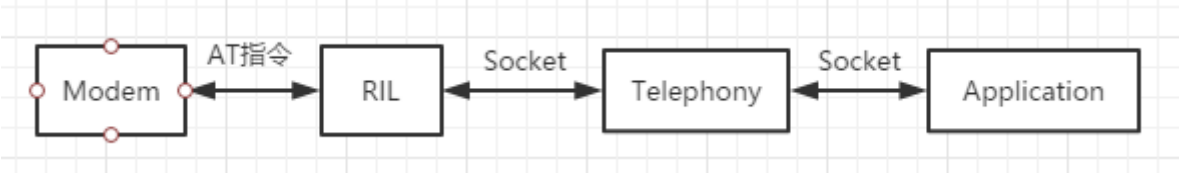
Telephony-Linux 蜂窝状态机

Telephony: Telephony是通信框架中数据通信服务的接口层. Telephony层维护和保存数据通信的连接状态,通过Socket和RIL层进行交互,接受上层应用程序的事件请求并转发给RIL层,并将RIL层返回的响应数据上传给应用程序

通信框架图:



- Modem

  通讯的硬件基础

- RIL

  适配不同Modem芯片而抽象出来的中间层,和Modem通过AT指令进行交互,主要功能为将Telephony层的请求事件转换为AT指令发送给Modem,并将Modem的响应信息和主动上报事件上报给Telephony

- Telephony

  数据通信服务的接口层,通过Socket和RIL进行事件处理命令的交互

- Application

  应用程序

Telephony 的主要业务:

- SIM Control
- Data Connection
- Service State

SIM Control

管理SIM卡相关信息，启动后会监听SIM卡变化事件，在SIM卡READY后，获取 ICCID、MSISDN、IMSI、MCC、MNC 等相关信息。

Data Connection

用于APN的初始化、数据链路的建立以及状态维护，能够在数据链路不通的情况下自行恢复。

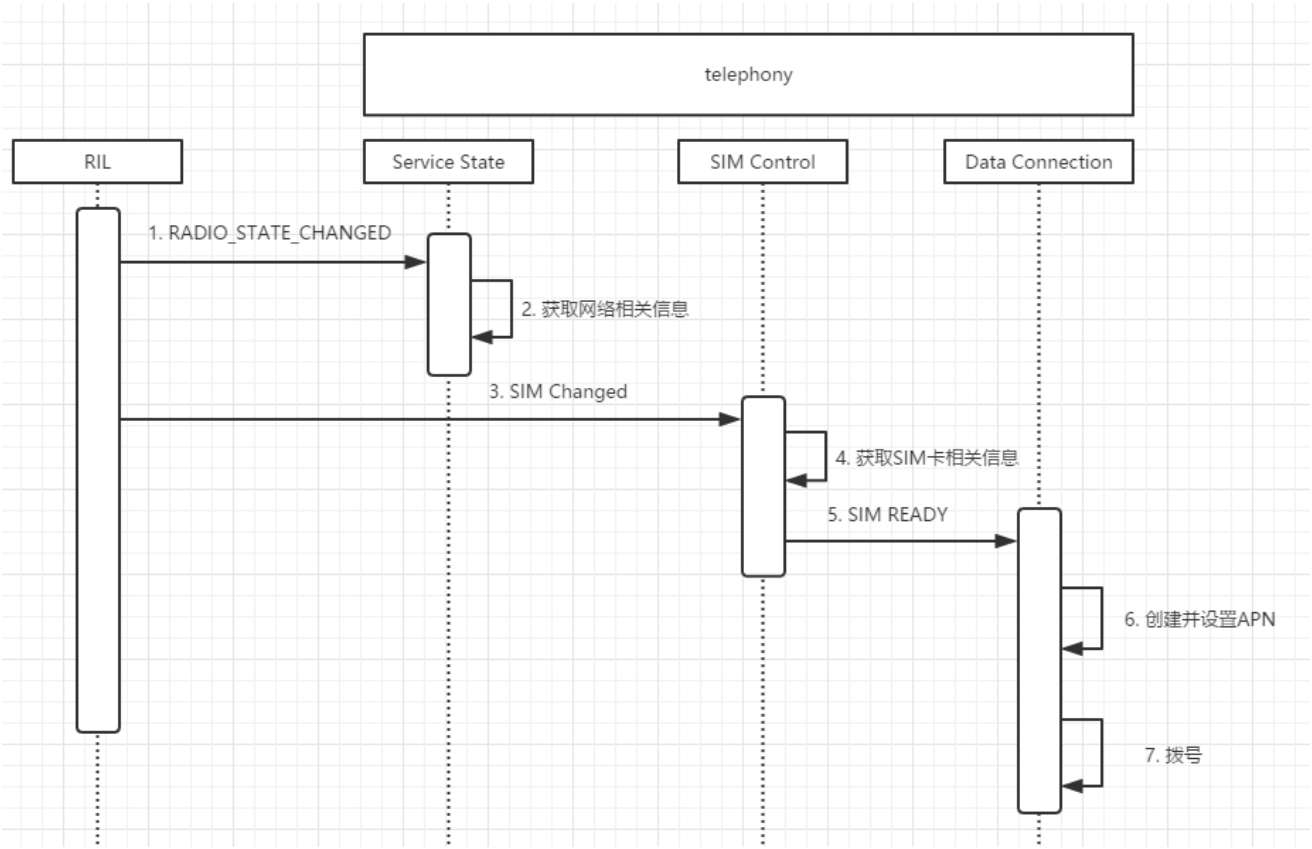
Service State

处理与网络相关的状态与信息，通过监听RIL层来获取网络状态的变化。主要包含以下状态：

- CS, PS 域注册状态
- 漫游情况
- 运营商信息
- 当前网络模式
- 信号格的变化

- 注册小区信息

Telephony大致初始化流程:



1. 在RIL成功打开AT端口后,会上报 `RADIO_STATE_CHANGED` 事件,表示RIL已正常工作
2. Service State 在获取到 RIL 正常工作后,就可以通过 RIL 获取所需的网络状态信息

```lua
-- telephony/status.lua
udp_map["RIL_UNSOL_RESPONSE_RADIO_STATE_CHANGED"] = function()
    if not queryDataEnable then
        ski.go(query_data)  -- 定时(15s)获取网络状态信息
    end

    if not collectRoutingDataEnable then
        ski.go(wan4g_interface_collect_routing) -- 定时(30s)获取网口流量状态(空闲,上传,下载,上传和下载)
    end
end
```

3. 在 SIM 卡发生变化时，RIL 会上报一个 `SIM_STATUS_CHANGED` 事件。SIM Control 也可自行轮询检测 SIM 卡状态。

```lua
-- telephony/UiccController.lua
udp_map["RIL_UNSOL_RESPONSE_SIM_STATUS_CHANGED"] = function()
    log.debug("recv RIL_UNSOL_RESPONSE_SIM_STATUS_CHANGED")
    getIccCardStatus()
end
```

4. SIM Control 在获取到此事件后，去获取 SIM 卡的状态，在 SIM READY 后可以读取其中的文件。

5. 在读取完文件后，发出 SIM_READY 事件，通知其他模块。
6. 接收到 SIM_READY 事件后，根据 SIM 卡信息创建APN List，并设置APN。

```lua
-- telephony/DcTracker.lua
local function onRecordsLoaded()
    createAllApnList()
    setInitialAttachApn()
    SimRecordsLoaded = true
    setupDataOnConnectableApns()
end
```

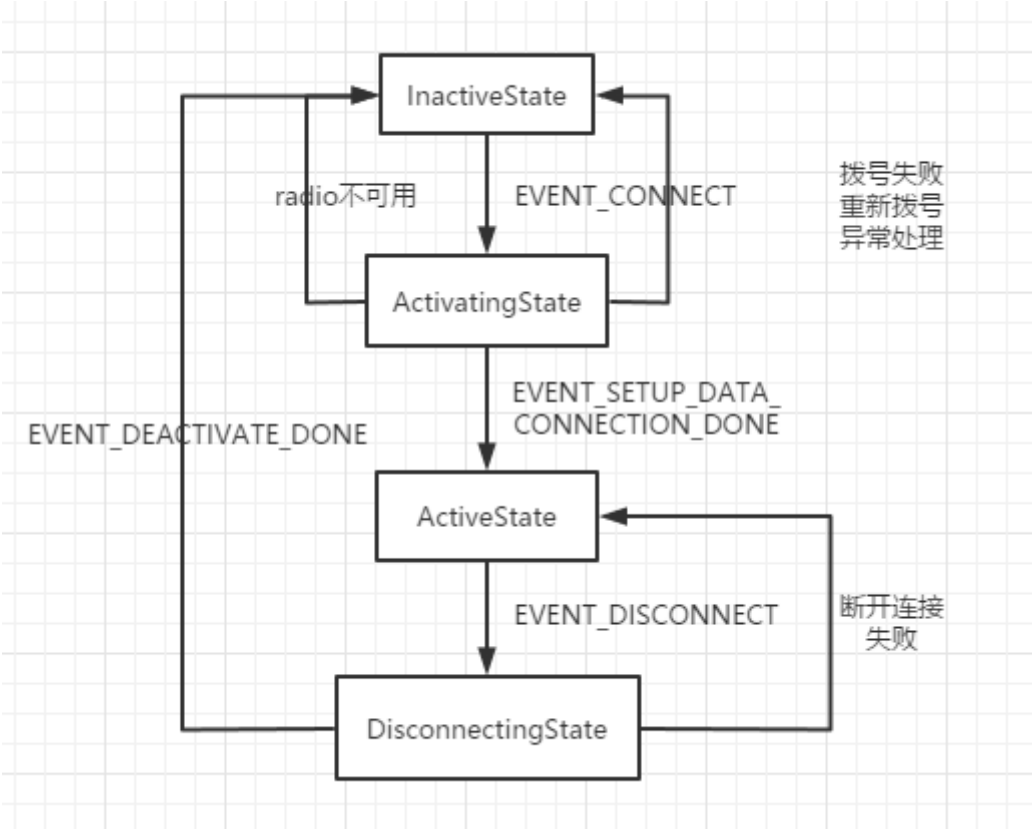
7. 在设置 APN 后，开始进行拨号操作。

```lua
-- telephony/DcTracker.lua
local function userDataEnableEvent()
    if not checkIfAllowConnect() then
        dc.SendMeassage({what=event.EVENT_DISCONNECT})
        dataStallAlarmOn = false
        return
    end

    local param = getCurrentApn()
    dc.SendMeassage({what=event.EVENT_CONNECT, param=param})     -- 开始拨号
    dataStallAlarmOn = true
end
```

数据拨号状态机

状态转换图：



- InactiveState

  拨号断开状态,初始化状态,在不使用,断开数据连接或数据连接异常时,最终回到此状态,在开启数据/漫游和APN切换时会发起拨号

- ActivatingState

  拨号连接过渡状态,进行拨号连接处理,在拨号失败后进行重新拨号,拨号失败2次后进行异常处理

- ActiveState

  拨号成功状态,在该状态会开启异常检测机制,在关闭数据/漫游和APN切换时会关闭拨号

- DisconnectingState

  拨号断开过渡状态,关闭异常检测机制,在断开连接失败时会返回到`ActiveState`状态

拨号状态对于拨号事件消息的处理

拨号事件消息 \ 拨号状态	InactiveState	ActivatingState	ActiveState	DisconnectingState
EVENT_CONNECT	处理	推后处理	忽略	推后处理
EVENT_DISCONNECT	忽略	推后处理	处理	推后处理
EVENT_DEACTIVATE_DONE	忽略	忽略	忽略	处理
EVENT_SETUP_DATA_ CONNECTION_DONE	忽略	处理	忽略	忽略

拨号稳定状态只会处理其对应的事件消息

拨号过渡状态对于`EVENT_CONNECT`和`EVENT_DISCONNECT`消息会推后交给拨号稳定状态处理

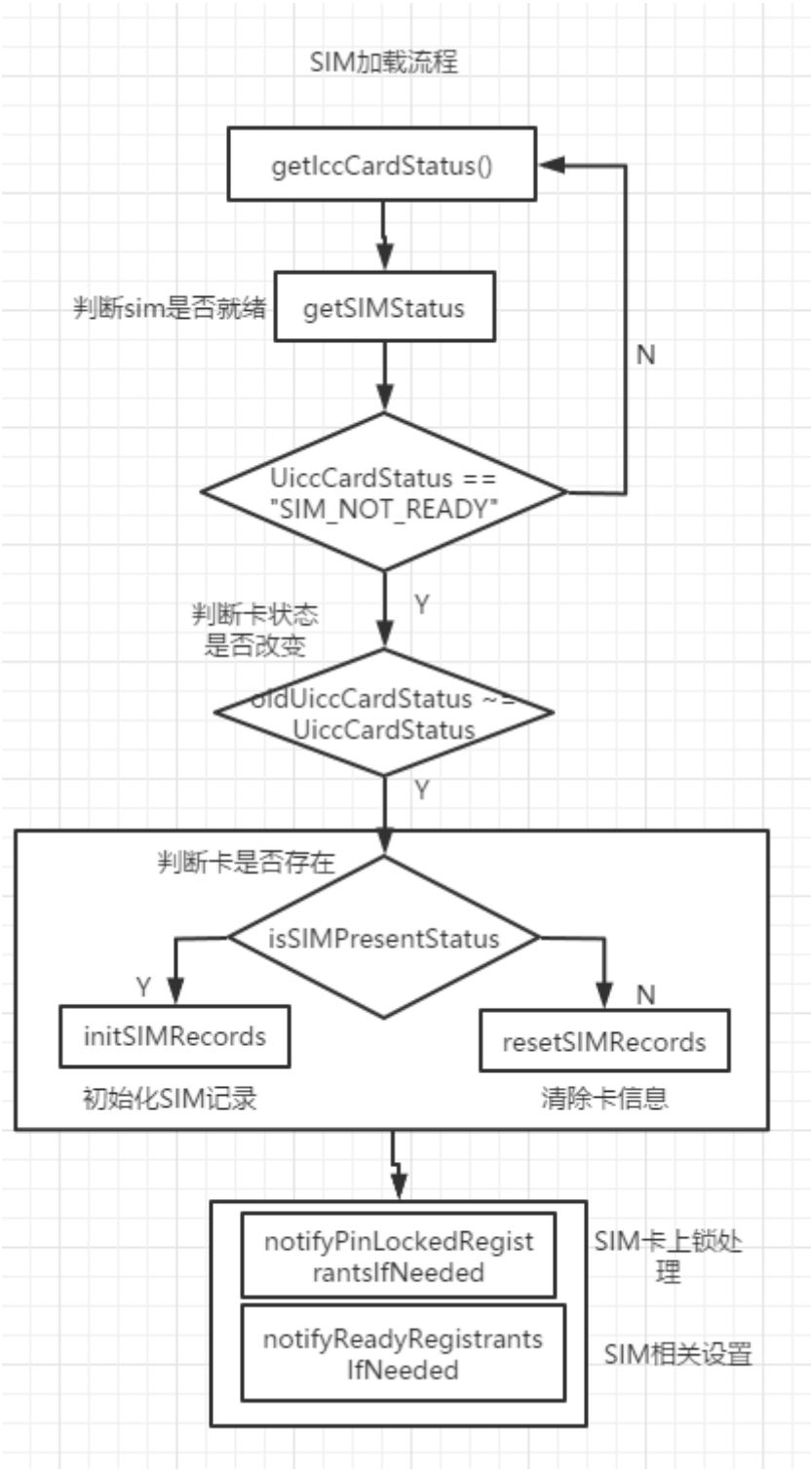
异常检测

在拨号成功后或多次拨号失败后,将会启用异常检测机制,查看网络是否正常. 若检测到异常,则依次执行以下操作直到网络恢复正常

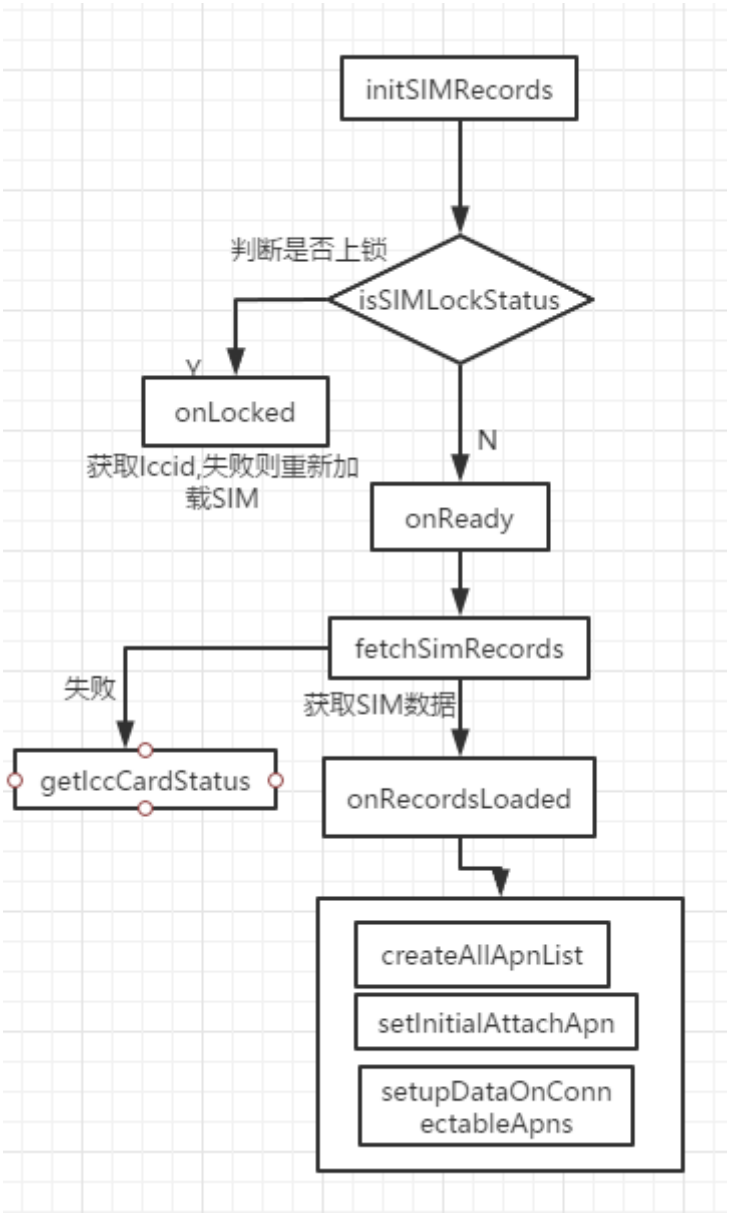
- 重新拨号
- 重新驻网
- 重启Radio, 相当于开启飞行模式
- 重启模组

(附)具体代码流程

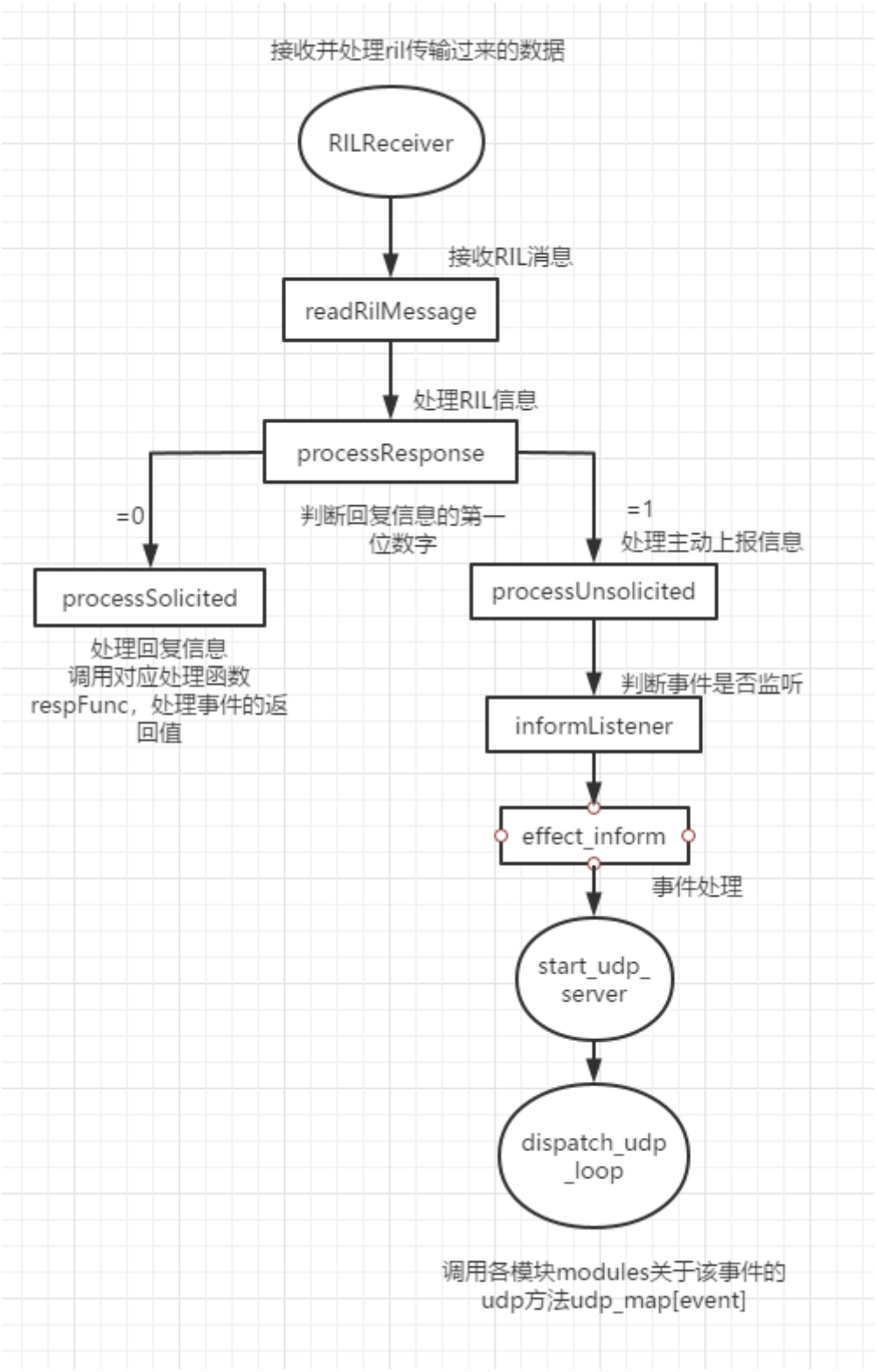
SIM加载流程


SIM加载流程
getIccCardStatus()
判断sim是否就绪
getSIMStatus
N
UiccCardStatus ==
"SIM_NOT_READY"
Y
判断卡状态
是否改变
oldUiccCardStatus ~=
UiccCardStatus
Y
判断卡是否存在
isSIMPresentStatus
Y
N
initSIMRecords
resetSIMRecords
初始化SIM记录
清除卡信息
notifyPinLockedRegist
rantsIfNeeded
SIM卡上锁处
理
notifyReadyRegistrants
IfNeeded
SIM相关设置

SIM初始化



Telephony 对于RIL 响应事件的处理


接收并处理ril传输过来的数据
RILReceiver
接收RIL消息
readRilMessage
处理RIL信息
processResponse
=0
判断回复信息的第一位数字
=1
处理主动上报信息
processSolicited
processUnsolicited
处理回复信息
调用对应处理函数
respFunc，处理事件的返回值
判断事件是否监听
informListener
effect_inform
事件处理
start_udp_server
dispatch_udp_loop
调用各模块modules关于该事件的udp方法udp_map[event]