状态机

游戏里的状态机其实也是用c++代码中的定时器来驱动的 每帧调用update

GameStateMachine.ChangeState

该方法会调用当前状态的Enter方法

GameStateMachine.Update

该方法会调用当前状态的Excute方法

RAGameLoadingState:

==>excute方法中调用startSDK

RASDKInitManager.loginSDK中发一个消息“SDKLogin”到平台

平台通过C++回调到注册类的对应方法“onSDKLogin”调回到到lua层

RAGameLoadingState:包含六个子状态：sdk init、hot update、init base environment、login server 、login finish 、max （login server 和 max 代码里没有用到）

1 RAGameLoadingState.enter==>sdk init

2 RAGameLoadingState.startSDK==>RASDKInitManager.getServerList==>hot update

3 RAUpdateManger.endDownload==>RASDKInitManager.doneUpdate

==>RASDKInitManager：EnterInitprecedure==》init base environment

isInLogin被设置我true，在状态机的excute方法里，

==》RAGameLoadingState.Execute()

==》 RASDKInitManager:Execute();

==》

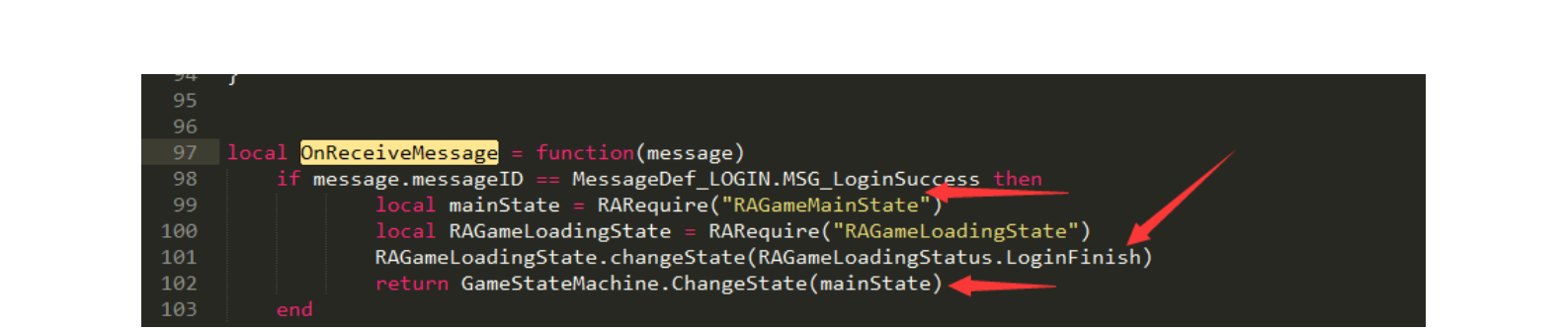
```
103  end
104
105  function RASDKInitManager:Execute()
106      if isInUpdate then
107          RAUpdateManager:Execute()
108      end
109
110      --
111      if isInLogin then
112          local RAInitPrecedure = RARequire("RAInitPrecedure")
113          RAInitPrecedure.Execute()
114      end
```

sdk登录后发消息 在消息里

```
150  --end
151
152  function RALoginPrecedure:addHandler()
153      RALoginPrecedure.platformSDKListener = platformSDKListener:new(self)--注册SDK回调处理tabel
154      RALoginPrecedure.packetManagerListener = ScriptPacketManagerListener:new(RALoginPrecedure.packetManagetHandler);--添加网络回调
155      MessageManager.registerMessageHandler(MessageDef_LOGIN.MSG_LoginSuccess, OnReceiveMessage)
156  end
157
```

4 登录成功服务器后发消息设置状态为login finish 并且切换状态机为RAGameMainState

```
95
96
97  local OnReceiveMessage = function(message)
98      if message.messageID == MessageDef_LOGIN.MSG_LoginSuccess then
99              local mainState = RARequire("RAGameMainState")
100             local RAGameLoadingState = RARequire("RAGameLoadingState")
101             RAGameLoadingState.changeState(RAGameLoadingStatus.LoginFinish)
102             return GameStateMachine.ChangeState(mainState)
103     end
```



login finish后 进入RAGameMainState状态