状态机

项目从gamePreceudre开始

```
GamePrecedure::Get()->enterGame();
```

```
164
165
166 void GamePrecedure::enterGame()
167 {
168     HawkUtil::Init();
169     init();
170
171     if (mIsInitScriptEngine)
172     {
173         cocos2d::CCLuaEngine* pEngine = cocos2d::CCLuaEngine::defaultEngine();
174         if(pEngine)
175         {
176             pEngine->executeGlobalFunction("GamePrecedure_enterGame");
177         }
178     }
179
180 }
```

注册帧刷新

```
89
90 void GamePrecedure::init()
91 {
92     if(mInitialized)
93         return;
94
95     // add
96     CCDirector *pDirector = CCDirector::sharedDirector();
97     CCDirectorHandler* directorHandler = pDirector->getHandler();
98     if (directorHandler){
99         directorHandler->onCreateGame();
100    }
101
102    CCLOG("GamePrecedure::init begin");
103
104    TimeCalculator::Get()->init();
105    //PlatformSDK::Get()->init();//the listener of platform is not init here, so change the init pos. add by xinghui
106
107    m_pScheduler = CCDirector::sharedDirector()->getScheduler();
108    m_pScheduler->retain();
109    m_pScheduler->scheduleUpdateForTarget(this, 0, false);
```

initScriptEngine方法里开始初始化lua引擎，调用GamePrecedure.lua文件，提前加载一些lua文件

```cpp
void GamePrecedure::initScriptEngine()
{
    if(!cocos2d::CCScriptEngineManager::sharedManager()->getScriptEngine() && !mIsInitScriptEngine)
    {
        mIsInitScriptEngine = true;
        cocos2d::CCLuaEngine* pEngine = cocos2d::CCLuaEngine::defaultEngine();
        cocos2d::CCScriptEngineManager::sharedManager()->setScriptEngine(pEngine);
        pEngine->getLuaStack()->setLogLuaErrorFun(catch_log_lua_error_message);
        lua_State *L = pEngine->getLuaStack()->getLuaState();

        lua_pushinteger(L, CC_TARGET_PLATFORM);
        lua_setfield(L, LUA_GLOBALSINDEX, "CC_TARGET_PLATFORM_LUA");

#ifdef COCOS2D_DEBUG
        lua_pushinteger(L, COCOS2D_DEBUG);
#else
        lua_pushinteger(L, 0);
#endif // COCOS2D_DEBUG
        lua_setfield(L, LUA_GLOBALSINDEX, "COCOS2D_DEBUG");

        tolua_Gamelua_open(pEngine->getLuaStack()->getLuaState());
        luaopen_pb(pEngine->getLuaStack()->getLuaState());
        luaopen_cjson(pEngine->getLuaStack()->getLuaState());
        luaopen_cjson_safe(pEngine->getLuaStack()->getLuaState());

        //pEngine->executeString("CCLuaLog(package.path)");
        pEngine->executeString("require \"BeforeUpate_Core/GamePrecedure\"");

        //修改lua堆栈大小
        int stack_size = 32;
        while (lua_checkstack(pEngine->getLuaStack()->getLuaState(), stack_size))
            stack_size <<= 1;

        CCLOG("lua stack size: %d", stack_size >> 1);
    }
}
```

```lua
require('BeforeUpate_Core.RAApi')

RARequire('GameStateMachine')
```

回到update方法里

```cpp
void GamePrecedure::update( float dt )
{
```

第一次调用firstUpdate

```cpp
    else{
        if (mIsInitScriptEngine)
        {
            if (isFristUpdate)
            {
                fristUpdate();
                isFristUpdate = false;
            }
            cocos2d::CCLuaEngine* pEngine = cocos2d::CCLuaEngine::defaultEngine();
            if (pEngine)
            {
                pEngine->executeGlobalFunction("GamePrecedure_update");
            }

            if (mPushInitOver)
            {
                mPushInitOver = false;
                pushInitOver();
            }
        }
    }
```

firstUpdate==》调用lua全局方法，启动状态机 GameStateMachine.Run()，并且设置游戏为RAGameLoadingState状态

```cpp
    if(pEngine)
    {
        pEngine->executeGlobalFunction("GamePrecedure_fristUpdate");
    }
}
```

```lua
end

function GamePrecedure_fristUpdate()
    RALogRelease("GamePrecedure_fristUpdate: onFirstUpdate begin")
    GameStateMachine.Run()
end

```

```lua
GameStateMachine.stateListener= nil
function GameStateMachine.Run()
    --RAGameInitState
    --RAGameUpdateState
    local RASDKLoginConfig = RARequire("RASDKLoginConfig")
    if CC_TARGET_PLATFORM_LUA == CC_PLATFORM.CC_PLATFORM_IOS and RASDKLoginConfig.OpenLogoMovie then
        GameStateMachine.ChangeState(RARequire('RAGameLogoFrameState'))
    else
        local editMode = SetupFileConfig:getInstance():getSectionString("editMode")
        if editMode == 'true' then
            GameStateMachine.ChangeState(RARequire('RAEditingState'))
        else
            GameStateMachine.ChangeState(RARequire('RAGameLoadingState'))
        end
        -- GameStateMachine.ChangeState(RARequire('RAEditingState'))
    end

    --³õÊ¼»¯uuidµÄseed£¬ÕâÀïapplication Ò»´Î³õÊ¼»¯ÔÚ×Ó
    local uuid = RARequire("uuid")
    uuid.seed()
end
```

GameStateMachine.ChangeState(RARequire('RAGameLoadingState'))

==》退出上一个状态，进入当前状态并且调用当前状态的Enter方法

```lua

function GameStateMachine.ChangeState(state,data)
    RALogRelease("GameStateMachine.ChangeState")
    if GameStateMachine.curState then
        if GameStateMachine.curState == state then
            return
        else
            RALogRelease("GameStateMachine.curState.Exit")
            GameStateMachine.curState.Exit()
        end
    end
    GameStateMachine.curState = state
    if GameStateMachine.curState then
        if data then
            GameStateMachine.curState.Enter(data)
        else
            RALogRelease("GameStateMachine.curState.Enter")
            GameStateMachine.curState.Enter()
        end
    end
end
```

状态机启动后下一帧开始调用lua的全局方法GamePrecedure_update，开启状态机的更新GameStateMachine.Update()，这个是每一帧都会调用

```
if (mIsInitScriptEngine)
{
    if (isFristUpdate)
    {
        fristUpdate();
        isFristUpdate = false;
    }
    cocos2d::CCLuaEngine* pEngine = cocos2d::CCLuaEngine::defaultEngine();
    if (pEngine)
    {
        pEngine->executeGlobalFunction("GamePrecedure_update");
    }

    if (mPushInitOver)
    {
        mPushInitOver = false;
        pushInitOver();
    }
}
```

```
187
188  function GamePrecedure_update()
189      debugStrPostUpdate()
190      GameStateMachine.Update()
191  end
```

这里说下状态的更新：每次根据判断会一次调用当前状态的Execute的方法

```
25
26  function GameStateMachine.Update()
27      if GameStateMachine.stateListener then
28          GameStateMachine.stateListener.beforeUpdate()
29      end
30
31      local curState = GameStateMachine.curState
32      if curState and curState.Execute then
33          curState.Execute()
34      end
35
36      if GameStateMachine.stateListener then
37          GameStateMachine.stateListener.afterUpdate()
38      end
39  end
```

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

```
616
617   function RASDKInitManager:EnterInitPrecedure()
618       RALogRelease("RASDKInitManager:EnterInitPrecedure")
619       local RAGameLoadingState = RARequire("RAGameLoadingState")
620       RAGameLoadingState.changeState(RAGameLoadingStatus.InitBasic)
621       local RAInitPrecedure = RARequire("RAInitPrecedure")
622       isInLogin = true
623       RAInitPrecedure.Enter()
624
625   end
626
```

给状态机设置全局监听

```
function RAInitPrecedure.Enter()
    --logTime
    local Utilitys = RARequire("Utilitys")
    Utilitys.LogCurTime("RAInitPrecedure.Enter Enter To RAInitPrecedure")

    -- mFrameNum = 0
    --init the global listner

    local RAGlobalListener = RARequire('RAGlobalListener')
    RAGlobalListener.init()
    GameStateMachine.setStateListener(dynamic_require('RAGlobalListener'))

    local RAGameConfig = RARequire("RAGameConfig")
    if RAGameConfig.BattleDebug == 1 then
        RAInitPrecedure._initLuaPreloadTest()
        RAInitPrecedure._initResourcePreloadTest()
    else
        RAInitPrecedure._initLuaPreload()
        RAInitPrecedure._initResourcePreload()
    end
end
```

```lua
--preload the lua
function RAInitPrecedure._initLuaPreload()
    --init the other config
    RAInitPrecedure.initLuaTable = {
        function ()
            --logTime
            local Utilitys = RARequire("Utilitys")
            Utilitys.LogCurTime("RAInitPrecedure._initLuaPreload Start To Preload Lua")

            RARequire("RANetUtil")
        end,
        function ()
            RARequire("RALoginManager")
        end,
        function ()
            RARequire("BasePage")
        end,
        function()
            RARequire("RAStringUtil"):setLanguage()
        end,
        function()
            --Ìí¼ÓpushÐÅÏ¢¼àÌý£¬²»»áÊÍ·Å
            local RAProtoPushLogic = RARequire("RAProtoPushLogic")
            RAProtoPushLogic:removePushProto();
            RAProtoPushLogic:registerPushProto();
        end,
        function()
            RARequire("RAPushRemindPageManager"):init()
        end,
        function()
            RARequire("RALoginPrecedure"):Enter()
        end,
        function()
            RARequire("RAGameLoadingState").changeState(RAGameLoadingStatus.LoginServer)
            RALogRelease("RAGameLoadingState  send login percent 100%")

            --logTime
            local Utilitys = RARequire("Utilitys")
            Utilitys.LogCurTime("RAInitPrecedure._initLuaPreload End To Preload Lua")
        end
    }
end
```

开启全局消息机制以及帧消息

```lua
function GameStateMachine.Update()
    if GameStateMachine.stateListener then
        GameStateMachine.stateListener.beforeUpdate()
    end

    local curState = GameStateMachine.curState
    if curState and curState.Execute then
        curState.Execute()
    end

    if GameStateMachine.stateListener then
        GameStateMachine.stateListener.afterUpdate()
    end
end
```

```lua

function RAGlobalListener.beforeUpdate()
--  if g_isLoadTallTables then
--      TableReader.loadAllTables()
--  end
    MessageManager.update()
    EnterFrameMananger.enterFrame()
    --RAPacketManager.Execute()
    -- RAScenesMananger.Execute()
end
```

isInLogin被设置true，在状态机的excute方法里，

==》RAGameLoadingState.Execute()

==》 RASDKInitManager:Execute();

==》RAInitPrecedure.Execute()

```
end

function RASDKInitManager:Execute()
    if isInUpdate then
        RAUpdateManager:Execute()
    end

    --
    if isInLogin then
        local RAInitPrecedure = RARequire("RAInitPrecedure")
        RAInitPrecedure.Execute()
    end
```

==》

```
RALoginPrecedure:Execute()
```

sdk登录后发消息 在消息里

```
--end

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

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

```
}


local OnReceiveMessage = function(message)
    if message.messageID == MessageDef_LOGIN.MSG_LoginSuccess then
            local mainState = RARequire("RAGameMainState")
            local RAGameLoadingState = RARequire("RAGameLoadingState")
            RAGameLoadingState.changeState(RAGameLoadingStatus.LoginFinish)
            return GameStateMachine.ChangeState(mainState)
    end
```

login finish后 进入RAGameMainState状态

RAGameMainState状态为进入游戏后的状态