====================

更新逻辑(iOS)

====================

didFinishLaunchingWithOptions //KnightClientAppDelegate.mm

viewDidAppear //UpdateEngineViewController.mm

UpdateMethod (run on thread) //UpdateEngineViewController.mm

UpdateEngine::Run //UpdateEngineViewController.mm

Run //UpdateEngine.cpp

UpdateManager::Run

mount

patcher

CheckCopyAppResToDoc() rootDir —> cacheDir

HasNewVersion()

UnInitialize()

DownloadOtherFiles()

GlobalNotifyEnd() (back to main thread)

ReturnToGame //UpdateEngineViewController.mm

returnToGameController //KnightClientAppDelegate.mm

RunGameApplication() //GameApplication.cpp

=================

游戏启动

=================

游戏启动流程：游戏引擎(Engine)创建后，开启cocos2d引擎(EngineApp)，创建cocos2d场景(EngineLayer)，场景创建后(EngineLayer::onEnter)，先初始化游戏(GameApplication::OnInit)，然后进入引擎主循环(Engine::MainLoop)，每一帧会调用Engine::Render()对游戏进行渲染，点击事件通过EngineLayer接收并分发

GameApplication::RunGameApplication() //GameApplication.cpp

Run //engine.cpp

EngineApp app(this) [CCApplication]

cocos2d::CCApplication::sharedApplication().run() [EngineApp:run()]

EngineApp::applicationDidFinishLaunching() //cocos2d\_wraper.cpp

场景起来后先调用onEnter()，然后每帧会调用draw()

..EngineLayer:onEnter()

runAction() ..EngineTicker::step()

Engine::OnIdle()

。先初始化 GameApplication::OnInit()

InitSDXLLog()

InitNetModule()

InitLuaScriptModule()

XMLCONFIG::Initialize()

XMLCONFIG::InitialAsync()

SetXmlBeanReady(true);

InitMaxDisplayPlayNum()

InitGlobalStringRes()

CGameUImanager::InitGameUI()

cocos2d::CCScriptEngineManager::sharedManager()->getScriptEngine()->executeGlobalFunction("SoundEnable.Handle")

CLoginManager()::Init()

。MainLoop()

CTimerManager::Tick()

Engine::RunUserTask()

FileIOManager::OnUpdate()

Effect::OnTick()

GameApplication::OnTick()

World::OnTick()

Engine::UpdateSpriteAction()

effects update

Engine::UpdateBeforeRender()

AniManager::Run()

AniManager::GarbageCollection()

..EngineLayer:Draw() //every frame

Engine::Draw()

Engine::Render()

。先初始化 Engine::InitNotify()

GameApplication::OnRenderInit() //GameApplication.cpp

CLoginQuickDialog::GetSingletonDialogAndShowIt();

cocos2d::CCScriptEngineManager::sharedManager()->getScriptEngine()->executeScriptFile(L"main.lua"); //脚本入口 LoginQuickDialog.cpp

local dlg = LoginDialog.getInstance() 登陆界面

dlg:SetVisible(true)

。World::Render()

GameApplication::OnRenderUI()

================

进入游戏

================

点击登陆按钮

CGameApplication::CreateConnection()

authc::newLogin()

LoginImpl::start()

aio::Engine::getInstance().connect(c)

OnlineAnnounce::Process() //authc.cpp

CNetConnection::onAuthOk() //NetConnection.cpp

CLoginManager::LoginIn()

send CRoleList 请求角色列表

收到角色列表

SRoleList::Process()

if (在战斗托管中)

send CEnterWorld

else

CLoginManager::UpdateRoleList()

if (角色数==1)

send CEnterWorld

elseif (角色数>1)

选择角色，然后send CEnterWorld

elseif (角色数==0)

创建角色成功后send CEnterWorld

。。local CreateRoleCmd = require "protocoldef.knight.gsp.ccreaterole":new()

CreateRoleCmd.name = editname

CreateRoleCmd.school = self.m\_iSelectedSchool

CreateRoleCmd.shape = self.m\_iRoleId[self.m\_iModelID]

require "manager.luaprotocolmanager":send(CreateRoleCmd)

收到进入游戏通知

void knight::gsp::SEnterWorld::Process(Manager \* manager, Manager::Session::ID sid ) GameApplication.cpp

GameApplication::StartGame()

...

初始化游戏所需数据，如角色属性、背包物品、宠物列表、游戏设置等等

SRoleEnterScene::Process() //GameScene.cpp

CGameScene::ChangeMap()

CGameUImanager::OnGameStart()

ShowHide.OnGameStart() //showhide.lua

MainControl.getInstanceAndShow() //maincontrol.lua 主界面

=============

网络

=============

配置服务器列表

client/resource/artres/cfg/serverconfig.bin

客户端会记录上次登录的服务器ip和端口，修改serverconfig.bin后，登录时需要重新打开选择服务器界面选择

定义xml协议

详情见

[](mt-g客户端生成协议工具说明.docx)

==================

gen4d.bat

==================

1.authc 生成网络模块所需的一些协议，与游戏逻辑无关的

2.CEGUI pkg-->lua

3.ProtocolDef 生成游戏相关协议 client\FireClient\Application\protocols

4.protoluapkg 将协议文件生成pkg

5.protocoldef 协议的lua绑定，pkg-->lua

6.bean 生成表格相关的inl/cpp/lua文件

7.tolua++-pkgs 客户端类的lua绑定 --还会重新生成一遍protocolpkg

==============

表格

==============

<?xml version="1.0" encoding="utf-8"?>

<namespace name="testtable">

<bean name="CTesttable1" from="客户端提示信息表/培培提示.xlsx" genxml="client">

<variable name="id" fromCol="ID" type="int" />

<variable name="type" fromCol="提示对话框类型" type="int" />

<variable name="msg" fromCol="提示内容" type="String" />

</bean>

<bean name="CTesttable2" from="w玩法说明tips对应表.xlsx" genxml="client" gencode="lua">

<variable name="id" fromCol="id" type="int" />

<variable name="name" fromCol="名称" type="string" />

<variable name="text" fromCol="描述" type="string" />

</bean>

<bean name="CTesttable3" from="客户端提示信息表/程序内字符串-服务器.xlsx" genxml="server" priority="1">

<variable name="id" fromCol="ID" type="int" />

<variable name="msg" fromCol="字符串" type="string" />

</bean>

</namespace>

填写xml配置文件，用于生成代码

Application\beans\make\gbeans\testtable.xml 定义表格数据结构体的成员

Application\beans\make\gbeans\main.xml 注册模块

终端运行 >> call ant

生成以下文件

Application\beans\knight\gsp\testtable\CTesttable1.h 结构体、抽象类

Application\beans\knight.gsp.testtable.CTesttable1.inl 实现

client\resource\artres\video\luabean\knight\gsp\testtable\ctesttable2.lua

Application\beans\beansModule.pkg tolua，通过$cfile引用把头文件里的内容注册进lua，genode=lua的生成lua文件，否则只是注册给lua使用

资源里的表格数据文件

resource/arttes/config/autoconfig xml格式

resource/arttes/config/autobinconfig 二进制

CGameApplication::m\_bReadXmlFromBinary 确定客户端是否使用二进制表文件

\*生成表格的工具\*

E:\ Work\tools \xmlgenerator

只有 genxml=client 的表格才会生成代码

gencode=lua 只生成lua代码

priority=1 高优先级，加入到tablemanager0.cpp 里管理，其他的会平均放到tablemanager1~9里面

用法

C++: knight::gsp::title::GetCTitleConfigTableInstance().getRecorder(roleTitle);

Lua: knight.gsp.role.GetCreateRoleConfigTableInstance():getRecorder(roleId)

===========

UI

===========

CGameUIManager:AddWndToRootWindow()

parent\_window:addChildWindow()

parent\_window:removeChildWindow()

示例

E:\Work\client\resource\artres\ui\lua\_scripts\demo8.lua

CEGUI的pkg

E:\Work\client\3rd\cegui\CEGUI\src\ScriptingModules\LuaScriptModule\package

HelperFunctions.pkg 脚本里的辅助接口

mhsdutils.lua 一些公用方法

==============

其他

==============

接收点击事件的入口

CGameApplication::OnWindowsMessage()

退出游戏

CGameApplication::ExitGame()

游戏心跳

GameApplication:OnTick() c++层的游戏主心跳

CLuaTickerRegister::LuaTick()

LuaMainTick() //mainticker.lua lua层的心跳

=============

**战斗**

=============

CBattleManager::BeginBattleScene

战斗开始的特效

EffectEditor.exe打开 client\resource\artres\effect\geffect\ui\fightstart.efct.lmx

fightstart.efct.lmx用到的资源在client\resource\artres\effect\animation\ui\fightstart，用SpriteEditor.exe编辑

在client\resource\artres\config\autoconfig\knight.gsp.effectpath.ceffectpathnonedrama.xml里注册

<record id="10430" Path="geffect/ui/fightstart"/>

代码使用： MHSD\_UTILS::GetEffectPath(10430)

=============

**任务**

=============

主线任务表 client\resource\artres\config\autoconfig\knight.gsp.task.cmainmissioninfo.xml

client\resource\artres\config\autoconfig\knight.gsp.npc.cnpcshape.xml

<record id="6125" shape="xiongmao" headID="9064" littleheadID="30220" name="熊猫" dir="4" shadow="3" attack="31xiongmao\_attack1" magic="31xiongmao\_magic1"/>

client\resource\artres\config\autoconfig\knight.gsp.npc.cmonsterconfig.xml

<record id="20001" name="黑眼圈" title="雷峰塔1层" modelID="6125" becatch="1" showhpbar="1" showlevel="0" npctype="1" bodytype="5" area1colour="1" area2colour="0"/>