# NetWork

## 变量

running 运行状态

loginAddress 登陆服务器地址,手动填写

pingdt 登陆到游戏服务器后发送ping消息的间隔,初始设置为30则立即发送一次ping

pong 游戏服务器的ping回应, 发送之后设为false,如果下一次发送ping时间到了还没被设为true就断开连接返回登陆界面

needPing 是否开启ping超时返回登陆界面,默认为false

State 连接到的服务器状态,因服务器情况差异而定,一般来说分NLS\_LOGINSERVER(登陆服务器)和NLS\_GAMESERVER(游戏服务器)两种状态

Reconnect 是否重连中的标识

Loop 处理网络状态和接收消息的loop函数

Handlers 自定义的消息处理函数

## 接口

send(packet)

向服务器发送消息包,会在发送之前判断下网络运行状态,如果network.running为false,则会进入reconncet流程

disconnect()

断开连接

hasPacket()

是否收到了包

getPacket()

获得当前需要解析的包

popPacket(packet)

释放当前包

isLoginState(state)

isState(state)

连接状态.

NLS\_LOGINSERVER 为连接登陆服务器中

NLS\_GAMESERVER 为连接游戏服务器中

两种状态的消息码可能相同,但是处理函数却不相同

ConnectLoginServer(address)

连接登陆服务器,address为地址字符串,连接成功后network.state切换为NLS\_LOGINSERVER

SetUserNameAndPassWord(user,psw)

设置用户输入的账号名和密码,以便注册或者登陆使用

SendLoginPacket(packet)

保存登陆服务器返回的一些用作验证的信息

LoginRealmListResponse(packet)

处理游戏服务器列表消息

目前处理完成后会自动去连接服务器列表首个服务器,连接状态切换到NLS\_GAMESERVER

ConnectGameServer()

连接游戏服务器,连接成功之后networ.state切换为NLS\_GAMESERVER

GameAuthChallengeResponse(packet)

处理游戏登陆挑战返回的一些信息

setVersion(version)

设置服务器版本,登陆的时候需要用到,必须设置

SetUserNameAndPassWord(user,psw)

设置用户名user和密码psw

sendAuthSessionMessage()

发送登陆信息给服务器

SendSignInPacket(packet)

发送注册消息给服务器

ConnectFailed()

判断是否断开连接用

IsConnected(state)

不仅判断当前连接状态,还会判断连接是否断开

SetPingCode(opcode)

设置发送的ping的消息码

Ping()

发送ping消息

OnRecvPong(time)

接收服务器的pong消息

addHandler(opcode,state,n,h)

设置不同连接状态下的消息处理函数,opcode为消息码,state为连接状态,n为函数说明,h为处理函数

reConnect()

根据当前连接状态重新连接服务器

reSend()

发送上次Send不成功的一个消息

update()

在C++处理一些逻辑

startup()

启动network.

会根据填写的loginAddress去连接登陆服务器,

然后注册一个逐帧监控网络包的接收状态并处理包的函数

shutdown()

关闭network

断开连接,并且关闭network.loop函数

## 状态处理函数

以下函数可以由使用者重定义处理函数

onNeedReConnect

当网络断开无法发送消息包时调用

onSendMessage

当消息包push到发送队列时调用

onSendSuccess

当消息包被成功发送时调用

onRecvMessage(opcode)

当接收到某个opcode的包时调用

onShutDown

当network关闭时调用

onReConnect

当断开重连成功之后调用

onPingFailed

当ping失败之后调用,默认会将network shutdown

onConnectFailed

当网络失效时调用,默认会将network shutdown

## 消息包

游戏内的消息包使用自定义的WorldPacket

因为lua中没有64位的数据类型,所以lua里的U64是使用Guid这个对象来保存的.

构建

new(opcode,size)

opcode 消息码

size 消息长度,不用精确匹配,在写入流的时候会自动改变

setU8(int) 写入一个unsigned char

setI8(int) 写入一个 char

setU16(int) 写入一个 unsigned short

setI16(int) 写入一个 short

setU32(int) 写入一个 unsigned int

setI32(int) 写入一个 int

setFloat(float) 写入一个 float

setStr(const char\*) 写入一个字符串

setU64(Guid) 写入一个 unsigned int64

例:

local send\_packet = WorldPacket:new(CMSG\_GAME\_CHOOSE\_CLASS,1)

send\_packet:setU8(tag)

network.send(send\_packet)

解析

getU8() 返回一个unsigned char

getI8() 返回一个 char

getU16() 返回一个 unsigned short

getI16() 返回一个 short

getU32() 返回一个 unsigned int

getI32() 返回一个 int

getFloat() 返回一个 float

getStr() 返回一个 字符串

getU64() 返回一个 guid对象

例:

function Handle\_questUpdateComplete( packet )

local questId = packet:getU32()

local rewardGold = packet:getU32()

local rewardBag = packet:getU32()

local rewArane = packet:getU32()

local rewCardSizes = packet:getU8()

local rewCardTable = {}

for i = 1,rewCardSizes do

rewCardTable[i].rewCardId = packet:getU32()

rewCardTable[i].rewCardCounts = packet:getU32()

rewCardTable[i].reCardType = packet:getU8()

end

end

## Opcode的编写

见附件里的opcodes.lua