**文件结构**

Source

+-Common

+-Common.h

+-CommStruct.h

+-PackageStruct.h

+-PackageStruct\_XX.h

+-Protocol.h

+-Protocol\_XX.h

+-PlayerGoods.h

+-PlayerGoods.cpp

+-PlayerTasks.h

+-PlayerTasks.cpp

+-Utility

+-Utility.h

+-Singleton.h

+-FunctionMap.h

+-FunctionMap.cpp

+-PublicObject.h

+-PublicObject.cpp

+-MemoryFactory.h

+-MemoryFactory.cpp

+-Yond\_MD5.h

+-Yond\_XML.h

+-Yond\_XML.cpp

+-Yond\_Log.h

+-Yond\_log.cpp

+-Yond\_Thread.h

+-Yond\_Thread.h

+-Yond\_Timer.h

+-Yond\_Timer.cpp

+-Yond\_ini.h

+-Yond\_ini.h

+-Network

+-Network.h

+-Epoll\_for.h

+-Epoll\_for.cpp

+-NetworkObject.h

+-NetworkObject.cpp

+-SessionObject.h

+-SessionObject.cpp

+-SyncMessage.h

+-SyncMessage.cpp

+-AgentSrv

+-AgentMain.cpp

+-AgentServer.h

+-AgentServer.cpp

+-AgentPlayer.h

+-AgentPlayer.cpp

+-AgentFactory.h

+-AgentFactory.cpp

+-Handler\_FromClientServer.h

+-Handler\_FromClientServer.cpp

+-Handler\_FromLoginServer.h

+-Handler\_FromLoginServer.cpp

+-Handler\_FromGameServer.h

+-Handler\_FromGameServer.cpp

+-Handler\_FromChatServer.h

+-Handler\_FromChatServer.cpp

+-GameSrv

+-GameMain.cpp

+-GameServer.h

+-GameServer.cpp

+-GamePlayer.h

+-GamePlayer.cpp

+-GameFactory.h

+-GameFactory.cpp

+-Handler\_FromAgentServer.h

+-Handler\_FromAgentServer.cpp

+-Handler\_FromDBServer.h

+-Handler\_FromDBServer.cpp

+-Handler\_FromAIServer.h

+-Handler\_FromAIServer.cpp

+-LoginSrv

+-LoginMain.cpp

+-LoginServer.h

+-LoginServer.cpp

+-LoginPlayer.h

+-LoginPlayer.cpp

+-LoginFactory.h

+-LoginFactory.cpp

+-Handler\_FromAgentServer.h

+-Handler\_FromAgentServer.cpp

+-Handler\_FromGameServer.h

+-Handler\_FromGameServer.cpp

+-ChatSrv

+-ChatMain.cpp

+-ChatServer.h

+-ChatServer.cpp

+-ChatPlayer.h

+-ChatPlayer.cpp

+-ChatFactory.h

+-ChatFactory.cpp

+-Handler\_FromAgentServer.h

+-Handler\_FromAgentServer.cpp

+-AISrv

+-WorldSrv

+-DBSrv

+-DBMain.cpp

+-DBServer.h

+-DBServer.cpp

+-DBPlayer.h

+-DBPlayer.cpp

+-DBFactory.h

+-DBFactory.cpp

+-Handler\_FromLoginServer.h

+-Handler\_FromLoginServer.cpp

+-Handler\_FromGameServer.h

+-Handler\_FromGameServer.cpp

**Common模块说明**

**1, PackageStruct\_XX 网络数据包结构；**

Struct MSG\_BASE {

Pubilc MSG\_BASE();

Public Init();

Public U\_short m\_byCategory;

Public U\_short m\_byProtocal;

};

Struct MSG\_SERVER\_TYPE : public MSG\_BASE; （服务器类型包）

Struct MSG\_BASE\_FORWARD : public MSG\_BASE; （通信前缀包）

Struct MSG\_TEST\_PACK : public MSG\_BASE; （测试包）

**2, ProtocoXX 网络协议类型枚举；**

消息命名规则：

1）(Request) \_REQ （请求）

2）(Answer) \_ANS （应答）

3）Client -> Server (SYN) \_SYN （同步）

4）Server -> Client (CMD) \_CMD （命令）

5）Server -> Client (CMD) \_BRD （广播）

6）Server -> Client (CMD) \_NTF （通知）

7） \_ACK （允许）

8） \_NAK （失败）

**3, Protoco\_CL 客户端协议类型枚举（命名以主动连联者为先，**Client->Login简称CL**）**

enum eCL\_Category {

CL\_Connect = 10,

};

enum eCL\_Connect {

CL\_Heratbeat\_SYN = 1000,

CL\_Heratbeat\_ANC,

CL\_Login\_REQ, （登入请求：【用户名，MD5(日期+密码)】）

CL\_Login\_ANC, （登入应答：【 临时Key值，指定Agent服务器{IP, Port}】)

CL\_Logout\_REQ, （退出请求：【临时Key值】）

CL\_Logout\_ANC, （退出应答：【结果{0，1}】）

};

**4, Protoco\_LD 客户端协议类型枚举（命名以主动连联者为先，**Login->DB简称LD**）**

enum eLD\_Category {

LD\_Connect = 10,

LD\_ClientLogin,

LD\_Update,

};

enum eLB\_Connect {

LD\_Heratbeat\_SYN,

LD\_Heratbeat\_ANC,

};

enum eLD\_ClientLogin{

LD\_Login\_SYN, （登入请求：【玩家ID】）

LD\_Login\_ANC, （认证确认）

LD\_Logout\_REQ, （登入请求：【玩家ID】）

LD\_Logout\_ANC, （清除缓存）

};

**5, Protoco\_AL 客户端协议类型枚举（命名以主动连联者为先，**Agent->Login简称AL**）**

enum eAL\_Category {

AL\_Connect = 10,

AL\_ClientLogin,

};

enum eAL\_Connect {

AL\_Heratbeat\_SYN = 1000,

AL\_Heratbeat\_ANC,

};

enum eAL\_ClientLogin{

AL\_Login\_SYN, （登入同步：【临时Key值, 用户名ID】）

AL\_Login\_ANC, （登入应答：【 临时Key值，指定Agent服务器{IP, Port}】)

AL\_Logout\_REQ, （退出请求：【临时Key值】）

AL\_Logout\_ANC, （退出应答：【临时Key值，结果{0，1}】）

};

**6, Protoco\_CA 客户端协议类型枚举（命名以主动连联者为先，**Client->Agent简称CA**）**

enum eCA\_Category {

CA\_Connect = 10,

CA\_ClientLogin,

CA\_Game,

/\*CA\_Chat,\*/

};

enum eCA\_Connect {

CA\_Heratbeat\_SYN = 1000,

CA\_Heratbeat\_ANC,

};

enum eAL\_ClientLogin{

CA\_Login\_SYN, （登入请求：【临时Key值】）

CA\_Login\_ANC, （登入应答：【临时Key值】)

CA\_Logout\_REQ, （退出请求：【临时Key值】）

CA\_Logout\_ANC, （退出请求：【临时Key值】）

};

enum eCA\_Game {

CA\_StartGame\_REQ = 2000,

CA\_StartGame\_ANC,

CA\_JoinGame\_REQ, （加入房加：【玩家X】）

CA\_JoinGame\_ANC, （加入房加：【房间ID, 玩家{1,2,3}】）

CA\_InitCards\_BRD, （服务器给所有玩家发牌：【牌数、牌串】）

CA\_Discards\_BRD, （出牌通知：【玩家ID、开始时间，结束时间】）

CA\_Discards\_SYN, （玩家出牌：【牌数、牌值串】）

CA\_Discards\_ANC, （出牌应答：0-不合格、1-合格）

CA\_MaxCards\_BRD, （单圈胜利：0-不合格、1-合格）

CA\_WinPlayer\_BRD, （单局胜利：0-不合格、1-合格）

CA\_EndGame\_SYN,

CA\_StartGame\_ANC,

}

**7, Protoco\_GA 客户端协议类型枚举（命名以主动连联者为先，**Game->Agent简称GA**）**

enum eGA\_Category {

GA\_Connect = 10,

GA\_ClientLogin,

GA\_Game,

};

enum eGA\_Connect {

GA\_Heratbeat\_SYN = 1000,

GA\_Heratbeat\_ANC,

};

enum eAL\_ClientLogin{

GA\_Login\_SYN, （登入请求：【临时Key值，玩家ID】）

GA\_Login\_ANC, （登入请求：【临时Key值，玩家游戏ID】）

GA\_Logout\_REQ, （退出请求：【临时Key值】）

GA\_Logout\_ANC, （退出请求：【临时Key值】）

};

enum eGA\_Game {

GA\_StartGame\_REQ = 2000,

GA\_StartGame\_ANC,

GA\_JoinGame\_REQ, （加入房加：【玩家X】）

GA\_JoinGame\_ANC, （加入房加：【房间ID, 玩家{1,2,3}】）

GA\_InitCards\_BRD, （服务器给所有玩家发牌：【牌数、牌串】）

GA\_Discards\_BRD, （出牌通知：【玩家ID、开始时间，结束时间】）

GA\_Discards\_SYN, （玩家出牌：【牌数、牌值串】）

GA\_Discards\_ANC, （出牌应答：0-不合格、1-合格）

GA\_MaxCards\_BRD, （单圈胜利：0-不合格、1-合格）

GA\_WinPlayer\_BRD, （单局胜利：0-不合格、1-合格）

GA\_EndGame\_SYN,

GA\_StartGame\_ANC,

}

**8, Protoco\_GD 客户端协议类型枚举（命名以主动连联者为先，**Game->DB简称GD**）**

enum eGD\_Category {

GD\_Connect = 10,

GD\_ClientLogin,

GD\_Update,

};

enum eGB\_Connect {

GD\_Heratbeat\_SYN,

GD\_Heratbeat\_ANC,

};

enum eGD\_ClientLogin{

GD\_Login\_SYN, （登入请求：【玩家ID】）

GD\_Login\_ANC, （登入请求：【玩家数据包】）

GD\_Logout\_REQ, （登入请求：【玩家ID】）

GD\_Logout\_ANC, （清除缓存）

};

enum eGD\_Update { （GameServer每10分钟更新一次数据到数据库）

GD\_Update\_REQ, （退出请求：【玩家数据包】）

GD\_Update\_ANC, （退出请求：【结果{0,1}】）

}

**9, CommStruct通用类数据结构；**

Struct PLAYINFO {

Public char m\_szUserKey; （用户的Key）

Public uint m\_uiDBUserID; （用户数据库ID）

Public uint m\_uiMoney; （拥有的钱）

Public uint m\_uiPrepiad; （预付的钱）

Public uint m\_uiCards[17]; （握有的牌）

Public bool m\_bDisards[17]; （打出的牌）

Public float m\_fWonRate （胜率）

Public float m\_fEscapeRate; （逃跑）

Public uint m\_uiRankList; （排行）

Public uint m\_uiGoodsList[5]; （道具）

};

Struct TABLEINFO {

Public uint m\_szUserKey[3]; （玩家{1,2,3} - Key）

Public uint m\_szUserWons[3]; （玩家{1,2,3} - 胜率）

Public uint m\_szUserMoney[3]; （玩家{1,2,3} - 钱池）

};

**Utility模块说明**

**1 FunctionMap公共处理类;**

class FunctionMap {

Private Hash\_map<Uint, BASE\_FUNC\*> m\_mFunction;

Public SetFirst();

Public GetNext();

Public Add();

Public Find();

};

**2, PublicObject 公共处理类;**

class PublicObject {

Public SplitString();

Public GetTimeSpanLocalToUTC();

Public GetNowTime();

Public ToTimeStampByString();

Public ToOleDateTimeString();

Public IsToday();

};

**3， MemoryFactory 内存管理；**

Template<class cType> class MemoryFactory {

Protected MemBand<cType> \* m\_pBandHead;

Protected MemBand<cType> \* m\_pFreeHead;

Protected UINT m\_dwPoolExtendSize;

Public Initialize();

Public Release();

Public Alloc();

Public Free();

};

Template<class DataType> class MemBand {

Private DataType \*\* ppDataType;

Private DataType \* pDeleteArray;

Private UInt nMaxBandObjectNum;

Private Uint FreeIdx;

Public MemBand pPrev;

Public MemBand pNext;

Public AllocBand();

Public FreeBand();

Public AllocObject();

Public FreeObject();

};

**Yond\_MD5加密功能**

**Yond\_ini配置文件解析功能**

**Yond\_Log日志处理功能**

**Yond\_Timer定时器定时功能**

**Network模块说明**

**1, Epoll\_Handler** **在实现网络模型处理时，进行回调以用于动态扩展处理；**

Epoll\_Handler在epoll\_for.h 文件实现，是基于Epoll\_For处理机制的一个回调结构体。其中包括创建连接、接收消息、发送消息、断开连接。

struct Epoll\_Handler {

void (\*CreateSocket)(socket, epoll\_head);

void (\*DoRecv)(NetworkObject \*, MSG \*);

void (\*DoSend)(NetworkObject \*);

void (\*DestroySocket)(NetworkObject \*);

};

**2, epoll\_for，实现网络模型的处理机制；**

epoll\_for 在epoll\_for.cpp文件中实现，Epoll\_for是封装Epoll的所有原理，通过Eoll\_Handler回调结构来形成扩展处理机制。

void epoll\_for(void \* param) ;

**3, NetworkObject 实现TCP/IP的所有网络交互功能的封装；**

class NetworkObject {

Privete socket m\_socket;

Privete struct epoll\_event m\_epollEvent;

Privete u\_int m\_IP;

Privete Short m\_Port;

Privete Buffer\* RecvBuff;

Privete Buffer\* SendBuff;

Public Init();

Public StartListen();

Public OnAccept();

Public OnSend();

Public OnRecv();

Public Connect();

Public Close();

};

**4, SessionObject 实现网络实例化和异化处理（不同的网络段不同的处理方式）；**

class SessionObject {

Privete NetworkObject \* m\_socket;

Privete NetworkType m\_intType;

Privete Bool m\_bHeartbeat;

Privete Uint m\_uiLastUpdateTime;

Privete Yond\_Timer \* m\_timerHead;

Public Init();

Public Update();

};

**AgentServer模块说明**

**核心功能说明**

**1, 实例化Epoll\_Handler，四种回调处理模型；**

void callback\_CreateSocket(socket, epoll) {

NetworkObject \* obj = new NetworkObject;

obj -> Init(socket);

obj -> AddEpollEvent(epoll);

}

void callback\_DoSend(socket, epoll, msg) {

NetworkObject \* obj = new NetworkObject;

obj -> Init(socket);

obj -> AddEpollEvent(epoll);

}

void callback\_DoRecv(socket, epoll, recvLen, msg) {

PackageStruct \*Msg = msg;

FunctionMap \* func = Find(msg->first);

AgentServer a = globle;

NetworkObject obj = a->Find(sockfd);

func(obj, msg);

}

void callback\_DestroySocket(socket, epoll, recvLen, msg) {

PackageStruct \*Msg = msg;

FunctionMap \* func = Find(msg->first);

AgentServer a = globle;

NetworkObject obj = a->Find(sockfd);

func(obj, msg);

};

**2, AgentFactory 工厂类，创建和回收所需内存，内存约束和高效管理；**

class AgentFactory{

Private MemoryFactory<NetworkObject> \*m\_NetworkObjectPool;

Private MemoryFactory<SessionObject > \*m\_NetworkObjectPool;

Private MemoryFactory<AgentPlayer> \*m\_AgentPlayerPool;

Public Init();

Public NetworkObject \* AllocNetworkObject();

Public SessionObject \* AllocSessionObject();

Public AgentPlayer \* AllocAgentPlayer();

Public FreeNetworkObject(NetworkObject \*);

Public FreeSessionObject(SessionObject \*);

Public FreeAgentPlayer(AgentPlayer \*);

};

**3, AgentServer 是服务器root入口，所以资源从这里初始化、处理、释放；**

class AgentFactory : Single<AgentServer> {

Private Epoll\_Handler \* EH;

Private AgentFactory \* m\_cAgentFactory ;

Private Map<Uint, Uint>\* m\_mapSessions;

Private Map<Uint, Uint>\* m\_mapPlayers;

Public Init()

{

EH = new H;

EH->CreateSocket = callback\_CreateSocket;

EH->DoSend = callback\_DoSend;

EH->DoRecv = callback\_DoRecv;

EH->DestorySocket = callback\_DestorySocket;

}

Public Start() {

epoll\_for(\*EH);

}

Public AllocSession();

Public FindSession();

Public AllocSession();

Public AddPlayer();

Public FindPlayer();

Public FreePlayer();

};

**4 AgentPlayer记录玩家的网络信息，用于数据的转送操作；**

class AgentPlayer {

Private SessionObject \* m\_netClient;

Private SessionObject \* m\_netGame ;

Public Init();

Public SendToClient(MSG \* );

Public SendToGame(MSG \* );

};

**AgentMain主函数**

#define AGENG\_CONFIG\_PATH “agent.conf”

void main(int argc, char \* argv[]) {

Yond\_ini yini;

yini.Parser(AGENG\_CONFIG\_PATH);

Yond\_Log \* yLog = new Yond\_Log(yini);

yLog.Init();

yLog.Sart();

AgentServer \* root = new AgentServer(yini);

root .init();

root .Start();

}

**GameServer模块说明**

**核心功能说明**

**1, 实例化Epoll\_Handler，四种回调处理模型；**

void callback\_CreateSocket(socket, epoll) {

NetworkObject \* obj = new NetworkObject;

obj -> Init(socket);

obj -> AddEpollEvent(epoll);

}

void callback\_DoSend(socket, epoll, msg) {

NetworkObject \* obj = new NetworkObject;

obj -> Init(socket);

obj -> AddEpollEvent(epoll);

}

void callback\_DoRecv(socket, epoll, recvLen, msg) {

PackageStruct \*Msg = msg;

FunctionMap \* func = Find(msg->first);

GameServer a = globle;

NetworkObject obj = a->Find(sockfd);

func(obj, msg);

}

void callback\_DestroySocket(socket, epoll, recvLen, msg) {

PackageStruct \*Msg = msg;

FunctionMap \* func = Find(msg->first);

GameServer a = globle;

NetworkObject obj = a->Find(sockfd);

func(obj, msg);

};

**2, GameFactory 工厂类，创建和回收所需内存，内存约束和高效管理；**

class GgentFactory{

Private MemoryFactory<NetworkObject> \*m\_NetworkObjectPool;

Private MemoryFactory<SessionObject > \*m\_NetworkObjectPool;

Private MemoryFactory<GamePlayer> \*m\_GamePlayerPool;

Public Init();

Public NetworkObject \* AllocNetworkObject();

Public SessionObject \* AllocSessionObject();

Public GamePlayer \* AllocGamePlayer();

Public FreeNetworkObject(NetworkObject \*);

Public FreeSessionObject(SessionObject \*);

Public FreeGamePlayer(GamePlayer \*);

};

**3, GameServer 是服务器root入口，所以资源从这里初始化、处理、释放；**

class GameFactory : Single<GameServer> {

Private Epoll\_Handler \* EH;

Private GameFactory \* m\_cGameFactory ;

Private Map<Uint, Uint>\* m\_mapSessions;

Private Map<Uint, Uint>\* m\_mapPlayers;

Public Init()

{

EH = new H;

EH->CreateSocket = callback\_CreateSocket;

EH->DoSend = callback\_DoSend;

EH->DoRecv = callback\_DoRecv;

EH->DestorySocket = callback\_DestorySocket;

}

Public Start() {

epoll\_for(\*EH);

}

Public AllocSession();

Public FindSession();

Public AllocSession();

Public AddPlayer();

Public FindPlayer();

Public FreePlayer();

};

**4 GamePlayer记录玩家的所有游戏数据信息，用于游戏逻辑运算操作；**

class GamePlayer {

Private PLAYINFO m\_cPlayinfo; （加入游戏）

Private TABLEINFO m\_cTableInfo; （桌面信息）

Private SessionObject \* m\_netAgent; （代理服务器）

Private SessionObject \* m\_netDB; （数据库服务器）

Public Init();

Public JoinGame(); （加入游戏）

Public InitCards(); （发牌）

Public Discards(); （出牌）

Public Pass(); （过一轮牌）

Public isWon(); （胜利结算）

Public SendToAgent(MSG \* ); （返回数据到用户手上）

Public Update(); （更新数据到DB）

Pubile StartTimer(); （开始一个定时器，定时刷新任务）

};

**GameMain主函数**

#define GAME\_CONFIG\_PATH “game.conf”

void main(int argc, char \* argv[]) {

Yond\_ini yini;

yini.Parser(GAME\_CONFIG\_PATH);

Yond\_Log \* yLog = new Yond\_Log(yini);

yLog.Init();

yLog.Sart();

GameServer\* root = new GameServer(yini);

root .init();

root .Start();

}