**服务器架构图：**



**服务器启动流程：**



**发包流程：**

1. 从发送队列中取出[1,N]个消息节点，填充发送缓冲，直到发送缓冲满或者发送队列空为止。

2. IOCP线程把发送缓冲中的数据用WSASend()进行异步发送

3. IOCP线程收到发送结果后，循环执行步骤1

注：同一时间，服务器一次只发送一个数据包，一个包发送成功后才会发送下一个



**收包流程：**

1．多个小包

对于由几个小包组成的接收缓冲，根据包头的大小挨个分解并放入接收队列，如果最后剩下的数据不满包头的大小，那么将其放入一个临时的缓冲，等待合并新的数据。

2. 大包（数据包长度大于一个接收缓冲）

对于一个接收缓冲放不下一个包的情况，需要用一个变量指明接收队列中最后一个节点是否处理完成。



**断线处理**

1. 客户端正常断开

服务器直接进行下线流程（客户端主动断线，服务器踢客户端断线）

2. 客户端异常断开

如果拔网线等服务器检测不到客户端断线，服务器会对每个连接进行计数，每秒递增，直到到达阈值才执行下线流程

**AccountManager(账号模块)**

账号结构:

struct SAccount

{

uint32 m\_unAccountID; //账号ID

char m\_account[MAX\_ACCOUNT\_LEN]; //账号名

uint32 m\_unGateID; //所在GateServer的ID

uint32 m\_unChallengeID; //登陆ID

uint8 m\_isOnline; //是否在线

};

账号管理：

std::map<uint32, SAccount\*> m\_mapAccounts; //所有登陆的账号

void OnJoin(uint32 accountID, char\* account, uint32 gateID, uint32 challengeID); //账号登陆处理

void OnLeave(uint32 accountID); //账号登出处理

SAccount\* GetAccount(uint32 accountID); //根据账号ID获取一个账号信息

SAccount\* GetAccount(const XString& strAccount);//根据账号名获取一个账号信息

**CharactorManager(人物模块)**

玩家结构:

class CGamePlayer

{

uint32 m\_unPlayerID; //玩家ID

uint32 m\_unGateID; //所在GateServer的ID

SPlayerBaseInfo m\_baseInfo; //玩家基本信息

SRelation m\_playerRelation; //玩家好友信息

SETPLAYERGROUPS m\_playerGroups; //玩家群信息

};

玩家管理：

void OnPlayerEnterWorld(DBMsgAckEnterWorld\* dbAckEnterWorldMsg);//玩家进入世界处理

void OnPlayerLeaveWorld(uint32 unPlayerID); //玩家离开世界处理

CGamePlayer\* GetPlayer(uint32 playerID); // 根据玩家ID查找玩家信息

**ChatManager(聊天模块)**

聊天信息结构：

struct SChatData

{

uint32 m\_unTime; //发送的时间

uint32 m\_unTargetID; //发送目标ID

uint32 m\_unSenderID; //发送者ID

XString m\_strSenderName; //发送人昵称

uint8 m\_unChannel; //单聊or群聊

XString m\_strContent; //内容

};

**GroupManager(群模块)**

群成员信息结构：

struct SGroupMemeber

{

uint32 m\_unMemeberID; //成员ID

XString m\_strName; //成名名称

bool m\_bIsOnline; //是否在线

};

struct SGroupData

{

uint32 m\_unGroupID; //群ID

XString m\_strGroupName; //群名称

MAPGROUPMEMBERS m\_mapMembers;//所有群成员

};

struct SGroups

{

MAPGROUPS m\_mapGroups; //某个玩家所有的群

};

群信息管理：

void UpdateGroupMember(SGroupMemeber& groupMember, uint8 oper = SGroupData::EOPER\_JoinGroup); //更新一个群成员的信息

void UpdateGroup(uint32 memberID = 0, uint8 oper = SGroups::EOPER\_AddGroup);//更新一个群信息

void SaveGroup(); //保存一个群信息

void SendMsgToAllMembers(SMsg\* pMsg);//发送消息到所有群成员

CGroup\* GetGroup(uint32 unGroupID); //根据群ID获取群信息

void OnMemberEnterWorld(CGamePlayer& rPlayer); //群成员上线处理

void OnMemberLeaveWorld(CGamePlayer& rPlayer); //群成员下线处理

void UpdateMemberAllGroups(CGamePlayer& rPlayer); //更新玩家所有群信息

void SaveMemberAllGroups(CGamePlayer& rPlayer); //保存玩家所有群信息

void OnLoadGroup(SMsgDynamic\* pMsg); //Manager加载群信息

void OnPrepareCreateGroup(MsgReqCreateGroup\* pMsg);//准备创建群

void OnCreateGroup(SMsgDynamic\* pMsg); //创建群

void OnJoinGroup(MsgReqJoinGroup\* pMsg); //玩家加入群

void OnLeaveGroup(MsgReqLeaveGroup\* pMsg); //玩家离开群

**RelationManager(好友模块)**

好友结构：

struct SRelationData

{

uint32 m\_unCharactorID; //好友ID

XString m\_strNickname; //好友昵称

bool m\_bIsOnline; //是否在线

uint8 m\_unFriendGroupID; //好友所在分组

};

struct SRelation

{

std::map<uint32, SRelationData> m\_mapRelations; //所有好友信息

std::map<uint32, SFriendGroupData> m\_mapFriendGroups; //所有好友分组信息

};

好友信息处理：

void OnPlayerEnterWorld(CGamePlayer& rPlayer);//玩家上线处理（通知好友上线）

void OnPlayerLeaveWorld(CGamePlayer& rPlayer);//玩家下线处理（通知好友下线）

void UpdateSingleRelation(CGamePlayer& rPlayer, SRelationData& relationData, uint8 oper = SRelation::EOPER\_AddRelation);//更新单个好友信息

void UpdateAllRelation(CGamePlayer& rPlayer);//更新全部好友信息

void SaveAllRelation(CGamePlayer& rPlayer);//保存所有好友信息

void OnAddRelation(MsgReqAddRelation\* pMsg); //添加好友处理

void OnDelRelation(MsgReqDelRelation\* pMsg);//删除好友处理

**FileTransManager(文件传输模块)**

单个文件结构：

struct SFile

{

XString m\_strFilePath; //文件路径

uint32 m\_untotalLen; //总大小

uint32 m\_unTotalSent; //总发送量

uint32 m\_unTotalReceived; //总接收量

bool m\_bIsSender; //是否是发送者

bool m\_bEnableSend; //是否允许发送

bool m\_bHasReqSent; //是否已发送请求

uint32 m\_unLocalFileNO; //本地文件号

uint32 m\_unOtherFileNO; //对方文件号

std::fstream m\_fileOP; //文件对象

};

注：一个玩家可以同时发送和接收文件，这时候需要把每个文件标号来区分当前处理的文件对象。

bool SendFileReq(uint32 unTargetID, const XString& strFilePath);//发送文件请求

void SendFileAck(uint32 unSenderID, uint32 unSenderFileNO, const XString& strFilePath, uint32 unTotalLen);//发送文件应答

MAPFILELIST\* GetFileList(uint32 unPlayerID);//根据玩家ID获取当前发送和接收的文件列表

uint32 GetFileSize(std::fstream& fileStream);//获取文件大小

uint32 GenerateFileNO();//生成一个文件号

void CheckFiles(); //每一个TICK检测当前的文件列表，发送未完成的文件

**ChatHistoryManager(聊天记录模块)**

struct SChatHistoryData

{

uint32 m\_unTime; //发送时间

uint32 m\_unSenderID; //发送者ID

XString m\_strSenderName; //发送者昵称

uint32 m\_unTargetID; //目标ID

bool m\_bIslocalSent; //是否是本地发送

uint8 m\_unType; //单聊or群聊

XString m\_strContent; //内容

};

void LoadHistory(); //读取历史记录

void Save(SChatData& chatData, bool bIsLocalSent); //记录

LSTHISTORYLIST\* GetPlayerHistory(uint32 unPlayerID); //获取好友聊天记录

LSTHISTORYLIST\* GetGroupHistory(uint32 unGroupID);//获取群聊天记录

void LoadFriendsChatHistory(); //从数据库读取好友聊天记录

void LoadGroupsChatHistory(); //从数据库读取群聊天记录