[网络包 2](#_Toc2054346934)

[message.cpp 3](#_Toc166843272)

[一、创建 3](#_Toc1668713169)

[1、交换机 3](#_Toc2073285210)

[2、队列 4](#_Toc634868248)

[3、绑定 4](#_Toc1531885840)

[4、生产者 5](#_Toc223868997)

[5、消费者 5](#_Toc174883035)

[6、订阅 6](#_Toc1501540149)

[7、客户端拉取的消息 6](#_Toc1304948346)

[8、服务回复消息 7](#_Toc16364411)

[9、客户端确认消息 7](#_Toc158748861)

[10、服务端确认消息 8](#_Toc917417253)

[11、客户端退出消息 8](#_Toc99185711)

[二、删除 9](#_Toc566934705)

[1、交换机 9](#_Toc86645196)

[2、队列 9](#_Toc251896506)

[3、绑定 9](#_Toc934308105)

[mq\_util.cpp 10](#_Toc515893871)

[进程、文件、事件创建 10](#_Toc1236806958)

[内存拷贝 10](#_Toc1520323793)

[套接字的发送和接收 11](#_Toc1321556945)

[设置时间 11](#_Toc2131697341)

[文件 11](#_Toc967527286)

[client.cpp 11](#_Toc484920718)

[客户端 12](#_Toc362067061)

[初始化 12](#_Toc1439922276)

[连接与通信 12](#_Toc780474689)

[创建与删除 13](#_Toc607982147)

[数据发送 13](#_Toc628752203)

[生产者 14](#_Toc1818732581)

[初始化 14](#_Toc156257469)

[数据包 14](#_Toc1994264849)

[消费者 15](#_Toc1832038414)

[logger.cpp 16](#_Toc449093412)

[使用例子 16](#_Toc1655439926)

[消息确认 16](#_Toc180705750)

[ack\_producer.cpp 16](#_Toc579663392)

[ack\_consumer.cpp 16](#_Toc1416806552)

[持久化 17](#_Toc969041528)

[durable\_producer.cpp 17](#_Toc164982248)

[durable\_consumer.cpp 17](#_Toc455253859)

[测试最大连接 17](#_Toc2114857599)

[max\_connect\_test.cpp 17](#_Toc1409824896)

[多个生产者和消费者 ？？没有理解 17](#_Toc1739949721)

[multi\_producer.cpp 17](#_Toc1050139648)

[multi\_consumer.cpp 18](#_Toc1676452890)

[普通生产者和消费者 18](#_Toc1158273590)

[normal\_producer.cpp 18](#_Toc164967075)

[normal\_consumer1.cpp 18](#_Toc200241248)

[normal\_consumer2.cpp 18](#_Toc347780287)

[有优先级的生产者和消费者 18](#_Toc1840280122)

[priority\_producer.cpp 18](#_Toc1528526360)

[priority\_consumer.cpp 19](#_Toc1743147106)

[互斥锁 19](#_Toc1101498168)

[sem\_lock.cpp 19](#_Toc1570672436)

[共享内存队列 20](#_Toc1422642928)

[shm\_queue.cpp 20](#_Toc276765198)

[客户端服务 21](#_Toc143103384)

[客户端连接对象 21](#_Toc2104373895)

[struct ClientConnect 22](#_Toc1303870822)

[连接池 23](#_Toc1233771366)

[clientConnect.cpp 23](#_Toc2040736577)

[和客户端通信 24](#_Toc1198323402)

[connectServer.cpp 24](#_Toc1119775848)

[接收接入层包 25](#_Toc1673478675)

[logicServer.cpp 25](#_Toc562765966)

[MessageQueue类 25](#_Toc889609174)

[Exchange类 26](#_Toc880237004)

[LogicServer类 27](#_Toc128482045)

[持久化服务 28](#_Toc1176665080)

[PersistenceServer.cpp 28](#_Toc33094337)

# 网络包

定义网络包头部

#pragma pack(1) //调整结构体的边界对齐，让其以一个字节对齐；<使结构体按1字节方式对齐>，类中所有的成员变量都紧密的连续分布

typedef struct clientPackageHead\_s

{

unsigned short m\_iPackLen; //整个包长度

int m\_iClientIndex; //客户端在数组中的下标

unsigned short m\_iCmdId; //消息类型

}ClientPackageHead;

//网络包结构

typedef struct client\_package\_s

{

ClientPackageHead m\_ClientPackageHead;

char \*m\_pPackageBody;

}ClientPackage;

#pragma pack() //恢复默认的内存对齐（与文件开头的指令配对使用）

}

# message.cpp

## 一、创建

### 1、交换机

（1）

CreateExchangeMessage::CreateExchangeMessage(const string &istrName,unsigned short iExchangeType,bool ibDurable/\*=false\*/,bool ibAutoDel/\*=true\*/)

:Message(CMD\_CREATE\_EXCNANGE), m\_strExchangeName(istrName),m\_iExchangeType(iExchangeType),m\_bDurable(ibDurable),m\_bAutoDel(ibAutoDel)

输入：

CMD\_CREATE\_EXCNANGE：设置消息类型为1（默认）

const string &istrName 交换机名称

unsigned short iExchangeType 交换机类型（fanout,idrect,toppic）

bool ibDurable：是否持久化

bool ibAutoDel：没有消费者时是否自动删除

:Message(CMD\_CREATE\_EXCNANGE), m\_strExchangeName(istrName),m\_iExchangeType(iExchangeType),m\_bDurable(ibDurable),m\_bAutoDel(ibAutoDel)

输出：无输出

功能：初始化交换机

（2）int CreateExchangeMessage::GetMessagePack(char \*ipBuf,int \*iopBuffLen)

输入：当前消息指向和长度

char \*ipBuf 当前消息指向

int \*iopBuffLen 当前消息长度

输出：返回是否创建成功

功能：创建交换机消息头部

（1）判断创建名称是否符合要求

（2）计算当前创建交换机消息头部大小：网络包头部+名称+类型+是否持久化+是否自动删除

（3）判断当前创建的能否放入

（4）根据当前消息指向分别加上创建交换机消息头部大小、数组下标、消息类型、交换机名称、持久化信息、自动删除信息

### 2、队列

CreateQueueMessage::CreateQueueMessage(const string &istrName,short iPriority/\*=-1\*/,bool ibDurable/\*=false\*/,bool ibAutoDel/\*=true\*/)

:Message(CMD\_CREATE\_QUEUE), m\_strQueueName(istrName),m\_iPriority(iPriority),m\_bDurable(ibDurable),m\_bAutoDel(ibAutoDel)

输入：const string &istrName 队列名称

short iPriority/\*=-1\*/ 是否是优先级队列

bool ibDurable/\*=false\*/ 是否持久化

bool ibAutoDel/\*=true\*/ 没有消费者时是否自动删除

输出：无

功能：初始化队列

int CreateQueueMessage::GetMessagePack(char \*ipBuf,int \*iopBuffLen)

输入：当前消息指向和长度

输出：是否创建成功

功能：创建队列消息头部

### 3、绑定

CreateBindingMessage::CreateBindingMessage(const string &istrExName, const string &istrQueueName, const string &istrKey)

:Message(CMD\_CREATE\_BINDING),m\_strExchangeName(istrExName),m\_strQueueName(istrQueueName),m\_strBindingKey(istrKey)

输入 const string &istrExName 绑定的交换机名称

const string &istrQueueName 交换机需要传递消息的队列

const string &istrKey 路由键

输出：无

输入：初始化绑定

int CreateBindingMessage::GetMessagePack(char \*ipBuf,int \*iopBuffLen)

输入：当前消息指向和长度

输出：是否创建成功

功能：创建绑定消息头部

（需要判断和消息需要加上三个：交换机，队列，路由键）

### 4、生产者

CreatePublishMessage::CreatePublishMessage(const string &istrExName,const string &istrKey,

const string &istrMsgBody,short iPriority/\*=-1\*/,bool ibDurable/\*=false\*/,unsigned char iConfirmLevel/\*=0\*/)

:Message(CMD\_CREATE\_PUBLISH),m\_strExchangeName(istrExName),m\_strRoutingKey(istrKey),m\_strMsgBody(istrMsgBody),

m\_iPriority(iPriority),m\_bDurable(ibDurable),m\_iMsgSeq(-1),m\_iConfirmLevel(iConfirmLevel)

输入：const string &istrExName 交换机名称 **队列名称不需要传入吗**

const string &istrKey 路由键

const string &istrMsgBody 消息主体

short iPriority/\*=-1\*/ 优先级

bool ibDurable/\*=false\*/ 是否持久化

unsigned char iConfirmLevel/\*=0\*/ 消息确认等级

输出：无

功能：初始化生产者消息

int CreatePublishMessage::GetMessagePack(char \*ipBuf,int \*iopBuffLen)

输入：当前消息指向和长度

输出：是否创建成功

功能：创建生产者消息头部

### 5、消费者

CreateRecvMessage::CreateRecvMessage(const string &istrQueueName,unsigned char iConfirmLevel/\*=0\*/)

:Message(CMD\_CREATE\_RECV),m\_strQueueName(istrQueueName),m\_iConfirmLevel(iConfirmLevel)

输入：const string &istrQueueName 队列名称

unsigned char iConfirmLevel/\*=0\*/ 消息确认等级

输出：无

功能：初始化消费者消息

int CreateRecvMessage::GetMessagePack(char \*ipBuf,int \*iopBuffLen)

输入：当前消息指向和长度

输出：是否创建成功

功能：创建消费者消息头部

### 6、订阅

CreateSubscribeMessage::CreateSubscribeMessage(const string &istrQueueName,unsigned char iConfirmLevel/\*=0\*/)

:Message(CMD\_CREATE\_SUBCRIBE),m\_strQueueName(istrQueueName),m\_iConfirmLevel(iConfirmLevel)

输入：const string &istrQueueName 队列名称

unsigned char iConfirmLevel/\*=0\*/ 消息确认等级

输出：无

功能：初始化消费者消息

int CreateSubscribeMessage::GetMessagePack(char \*ipBuf,int \*iopBuffLen)

输入：当前消息指向和长度

输出：是否创建成功

功能：创建订阅消息头部

### 7、客户端拉取的消息

SeverStoreMessage::SeverStoreMessage(const string &istrMsgBody, short iPriority/\*=-1\*/,bool ibDurable/\*=false\*/)

:Message(CMD\_CLIENT\_PULL\_MESSAGE),m\_strMsgBody(istrMsgBody),m\_iPriority(iPriority),m\_bDurable(ibDurable)

{

m\_iConfirmLevel=CONSUMER\_NO\_ACK;

m\_iMsgSeq=-1;

m\_iDurableIndex=-1;

}

输入：const string &istrMsgBody 消息主体

short iPriority/\*=-1\*/ 优先等级

bool ibDurable/\*=false\*/ 是否持久化

输出：无

功能：初始化客户端拉取信息

设置消息确认等级为CONSUMER\_NO\_ACK= 0

设置消息下标和持久化下标 ？？

int SeverStoreMessage::GetMessagePack(char \*ipBuf,int \*iopBuffLen)

输入：当前消息指向和长度

输出：是否创建成功

功能：创建客户端推送消息头部

int SeverStoreMessage::SerializeDurableToString(char \*ipBuffer,int &iBuffLen)

？？函数功能

输入：当前消息指向和长度

输出：是否创建成功

功能：初始化可序列持久化字符串？？

1. 设置消息大小

前面加上两个unsigned short大小、名字、持久化下标、优先等级、消息主体

依次累加到当前消息后

### 8、服务回复消息

ActReplyMessage::ActReplyMessage(bool ibSucceed,const string &istrMsgBody)

:Message(CMD\_SERVER\_REPLY\_MESSAGE),m\_bSucceed(ibSucceed),m\_strMsgBody(istrMsgBody)

输入：bool ibSucceed 是否回复成功

const string &istrMsgBody 消息主体

输出：无

功能：初始化服务回复消息

int ActReplyMessage::GetMessagePack(char \*ipBuf,int \*iopBuffLen)

输入：当前消息指向和长度

输出：是否创建成功

功能：创建服务回复消息头部

### 9、客户端确认消息

ClientAckMessage::ClientAckMessage(unsigned char iConfirmLevel,int iAckSeq):Message(CMD\_CLIENT\_ACK\_MESSAGE),m\_iConfirmLevel(iConfirmLevel),m\_iAckSeq(iAckSeq)

输入：

unsigned char iConfirmLevel 确认级别

int iAckSeq 确认号

输出：无

功能：初始化客户端确认消息

int ClientAckMessage::GetMessagePack(char \*ipBuf,int \*iopBuffLen)

输入：当前消息指向和长度

输出：是否创建成功

功能：创建客户端确认消息头部

### 10、服务端确认消息

ServerAckMessage::ServerAckMessage(int iAckSeq):Message(CMD\_SERVER\_ACK\_MESSAGE),m\_iAckSeq(iAckSeq)

输入：

int iAckSeq 确认号

输出：无

功能：初始化服务端确认消息

int ServerAckMessage::GetMessagePack(char \*ipBuf,int \*iopBuffLen)

输入：当前消息指向和长度

输出：是否创建成功

功能：创建服务端确认消息头部

### 11、客户端退出消息

ClientExitMessage::ClientExitMessage(int iClientIndex):Message(CMD\_CLIENT\_EXIT)

输入：

int iClientIndex客户端在数组中的下标

输出：无

功能：初始化客户端推出消息

设置客户端在数组中的下标

int ClientExitMessage::GetMessagePack(char \*ipBuf,int \*iopBuffLen)

输入：当前消息指向和长度

输出：是否创建成功

功能：创建客户端推出消息头部

## 二、删除

### 1、交换机

int DeleteExchangeMessage::GetMessagePack(char \*ipBuf,int \*iopBuffLen)

输入：当前消息指向和长度

char \*ipBuf 当前消息指向

int \*iopBuffLen 当前消息长度

输出：返回是否删除成功

功能：删除交换机消息头部

（因为消息地址是累加的，即使是删除也是要把删除消息累加在当期消息后）

（1）判断删除的名称是否符合要求

（2）计算当前删除交换机消息头部大小：网络包头部+名称

（3）判断当前删除的能否删

（4）根据当前消息指向分别加上删除交换机消息头部大小、当前数组下标，消息类型、名称

### 2、队列

int DeleteQueueMessage::GetMessagePack(char \*ipBuf,int \*iopBuffLen)

输入：当前消息指向和长度

输出：是否删除成功

功能：删除队列消息头部

### 3、绑定

int CancelSubscribeMessage::GetMessagePack(char \*ipBuf,int \*iopBuffLen)

输入：当前消息指向和长度

输出：是否删除成功

功能：删除绑定消息头部

# mq\_util.cpp

## 进程、文件、事件创建

int FuncTool::DaemonInit()

输入：无

输出：返回是否创建成功

功能：创建两个进程

int FuncTool::SetNonBlock(int iFd)

输入：设置的文件

输出：返回是否创建成功

功能：设置非阻塞模式

int FuncTool::MakeEpollEvent(struct epoll\_event &iEvent,void \*const ptr)

输入：注册事件，字符串指针

输出：事件数据

功能：创建epoll事件

void \*FuncTool::GetEventDataPtr(const struct epoll\_event &iEvent)

输入：注册的事件

输出：返回时间数据

功能：获取事件数据

## 内存拷贝

从存储区 str2 复制 n 个字节到存储区 str1

int FuncTool::ReadBool(const void \*ipBuffer, bool &ibVal)

int FuncTool::WriteBool(void \*ipBuffer,bool ibVal)

int FuncTool::ReadByte(const void\* ipBuffer, unsigned char &iVal)

int FuncTool::ReadByte(const void\* ipBuffer, char &iVal)

int FuncTool::WriteByte(void\* ipBuffer, unsigned char iVal)

int FuncTool::WriteByte(void\* ipBuffer, char iVal)

int FuncTool::ReadBuf(const void\* ipSrc, void \*ipDest, int iLen)

int FuncTool::WriteBuf(void\* ipDest, const void \*ipSrc, int iLen)

内存拷贝+主机字节顺序和网络字节顺序互换

int FuncTool::ReadShort(const void \*ipBuffer, unsigned short &oVal, int iToHostOrder/\* = 1\*/)

int FuncTool::ReadShort(const void \*ipBuffer, short &oVal, int iToHostOrder/\* = 1\*/)

int FuncTool::WriteShort(void \*ipBuffer, unsigned short iVal, int iToNetOrder/\* = 1\*/)

int FuncTool::WriteShort(void \*ipBuffer, short iVal, int iToNetOrder/\* = 1\*/)

int FuncTool::ReadInt(const void \*ipBuffer, unsigned int &iVal, int iToHostOrder/\* = 1\*/)

int FuncTool::ReadInt(const void \*ipBuffer, int &iVal, int iToHostOrder/\* = 1\*/)

int FuncTool::WriteInt(void \*ipBuffer, unsigned int iVal,int iToNetOrder/\* = 1\*/)

int FuncTool::WriteInt(void \*ipBuffer, int iVal, int iToNetOrder/\* = 1\*/)

## 套接字的发送和接收

int FuncTool::Sendn(int iSockfd,const void \*ipBuffer,int iLen)

int FuncTool::Recvn(int iSockfd,void \*opBuffer,int iLen)

## 设置时间

const char \*FuncTool::GetLogHeadTime(time\_t tSec, time\_t tUsec)

## 文件

int FuncTool::CheckDir(const char \* ipPath) 文件是否存在，不存在则创建

int FuncTool::MakeDir(const char \* ipPath,bool bIsFilePath/\*=false\*/) 创建目录（调用checkdir）

bool FuncTool::IsFileExist(const char \* ipPath) 文件是否存在

int FuncTool::RemoveDir(const char \* ipPath) 删除目录及文件

int FuncTool::RemoveFile(const char \* ipPath) 删除文件

# client.cpp

三次握手：

第一次:客户端发送连接请求给服务器，服务器接收;

第二次:服务器返回给客户端一个确认码,附带一个从服务器到客户端的连接请求,客户机接收,确认客户端到服务器的连接.

第三次:客户机返回服务器上次发送请求的确认码,服务器接收,确认服务器到客户端的连接.

## 客户端

### 初始化

构造函数

Client::Client()

初始化进程好、确认等级、输入输出缓冲区等

析构函数

Client::~Client()

根据进程号关闭socket连接，清空输入输出缓冲区

### 连接与通信

客户端创建连接

Int Client::BuildConnection(const char \*ipSeverIp/\*=SERVER\_DEFAULT\_IP\_ADDR\*/,unsigned short iServerPort/\*=SERVER\_DEFAULT\_PORT\*/)

m\_iSockfd=socket(AF\_INET,SOCK\_STREAM,0); //创建套接字 通信域 通信类型 通信协议

sockaddr\_in serverAddr; //sin\_port和sin\_addr都必须是网络字节序

int ret=connect(m\_iSockfd,(struct sockaddr \*)&serverAddr,sizeof(serverAddr));//第一个参数是客户端的套接字（表明即将发起连接请求），第二个参数是服务端的套接字所在的“地方”（“地方”是我自定义的专有名词），第三个参数是该“地方”的大小。

//接收缓冲区

//发送缓冲区

//SO\_RCVBUF和SO\_SNDBUF每个套接口都有一个发送缓冲区和一个接收缓冲区，使用这两个套接口选项可以改变缺省缓冲区大小。

setsockopt(m\_iSockfd,SOL\_SOCKET,SO\_SNDBUF,(char \*)&iSendBufSize,sizeof(int));

setsockopt(m\_iSockfd,SOL\_SOCKET,SO\_RCVBUF,(char \*)&iRevBufSize,sizeof(int))

### 创建与删除

创建交换机

int Client::CreateExchange(const string &istrName,unsigned short iExchangeType,bool ibDurable/\*=false\*/,bool ibAutoDel/\*=true\*/)

创建交换机消息头部（判断是否创建成功）

发送一个数据包

接收一个回复数据包

解析回复包

创建队列

int Client::CreateQueue(const string &istrName,short iPriority/\*=-1\*/,bool ibDurable/\*=false\*/,bool ibAutoDel/\*=true\*/)

创建绑定

int Client::CreateBinding(const string &istrExName, const string &istrQueueName, const string &istrKey)

删除交换机

int Client::DeleteExchange(const string &istrName)

删除队列

int Client::DeleteQueue(const string &istrName)

### 数据发送

等待请求

int Client ::WaitServerReply()

使用poll根据各个描述符状态返回是否成功

发送数据包

int Client::SendOnePack(const char \*ipBuffer,int iLen)

等待套接字可写：使用poll根据各个描述符状态返回是否成功

在缓冲区发送数据包： int ret=FuncTool::Sendn(m\_iSockfd,ipBuffer,iLen);//s-已建立连接的套接字；buf-存放将要发送的数据的缓冲区指针；len-发送缓冲区中的字符数；flags-控制数据传输方式

缓冲区

int Client::SendPack()

缓冲区没满则发送，同时更新缓冲区头部

接收数据包

int Client:: RecvOnePack(char \*opBuffer,int \*iopLen,unsigned short &oRelyType)

一直监听缓冲区内有无发送过来的数据，有则分别读取头部和数据部分

缓冲区

int Client::RecvPack()

根据缓冲区大小判断是否接收（缓冲区大小小于头部，或者是大于头部小于包长度，说明有数据），当要接受的时候等待请求，并且通过recv接收，收到后查看缓冲区是否能再放入数据（能放就结束，证明符合要求）

## 生产者

## 初始化

构造函数

Producer::Producer()

初始化消息序列号

析构函数

Producer::~Producer()

释放接收缓冲区

### 数据包

生产者消息

int Producer:: PuslishMessage(const string &istrExName,const string &istrKey,

const string &istrMsgBody, short iPriority/\*=-1\*/,bool ibDurable/\*=false\*/)

创建生产者消息头部（如果需要确认，设置消息序号），获取消息包，发送数据（若要确认，则将消息保存）

接收数据包

int Producer::RecvServerAck()

一直监听缓冲区内有无发送过来的数据

如果收到的包太小（小于数据包头部大小），改变缓冲区数据位置（不读取包）；

反之读取包，如果数据包小于给定数据长度，更新缓冲区位置，反之读取客户端下标、消息类型、确认号，准备读下一个包

单个确认直接删除对应消息，批量确认则删除所有序号小于等于该序号的消息

消息确认与重发

int Producer::ReSendMsg()

从待确认链表中找到对应消息进行移除，获取等待确认花费时间，3秒未确认则重发

### 消费者

创建消费者消息

int Consumer::RecvMessage(const string &istrQueueName,char \*opBuffer,int \*iopLen,unsigned char iConfirmLevel/\*=CONSUMER\_NO\_ACK\*/)

创建消费者消息头部，获取消息包，发送数据

接收回包，若是服务器回复消息，说明出错（解析回复包），反之读确认级别、读取消息序号，若是自动确认，则进行确认（发送确认消息），如果是是手动确认，则保存确认级别和确认

若消息是之前已经收到的，打印重复消息

如果确认级别大于0，则把数据进行保存

创建订阅

int Consumer::CreateSubscribe(const string &istrQueueName,unsigned char iConfirmLevel/\*=0\*/)

创建订阅消息头部，获取消息包，发送数据

接收回复，解析回复包

消费消息

int Consumer::ConsumeMessage(char \*opBuffer,int \*iopLen)

接收数据包，读确认级别、读取消息序号

若是自动确认，则进行确认（发送确认消息），如果是是手动确认，则保存确认级别和确认

若消息是之前已经收到的，打印重复消息

如果确认级别大于0，则把数据进行保存

取消订阅

int Consumer::CancelSubscribe(const string &istrName)

创建取消订阅消息头部，获取消息包，发送数据

接收回复，解析回复包

发送消费数据包

int Consumer::SendConsumerAck()

如果确认登记是手动确认，创建客户端确认消息头部，获取消息包，发送数据

# logger.cpp

写日志函数

# 使用例子

## 消息确认

### ack\_producer.cpp

获取字符串形式的当前时间

string GetCurTime()

主函数

初始化生产者，建立连接，创建一个队列queue2，一个交换机exchange1，将队列绑定到交换机上

开启确认

生产消息：通过交换机、路由键。每5000条打印一次消息，每发100条获取确认号并处理到期消息（接收数据包（监听缓冲区），消息的确认与重发）

删除交换机

### ack\_consumer.cpp

计算消息的延时，以毫秒为单位

double GetDelayTime(string iStrMsg)

主函数

初始化消费者，建立连接，创建两个队列queue1、queue2，创建两个订阅接收,手动确认

消费消息

取消订阅，删除队列，删除消费者

## 持久化

### durable\_producer.cpp

主函数

初始化生产者，建立连接，创建一个队列queue4，一个交换机exchange1，将队列绑定到交换机上（两者设置为持久化）

生产消息：通过交换机、路由键，消息设置为持久化 ？？如何实现持久化 ——不删除队列和交换机

删除生产者

### durable\_consumer.cpp

初始化消费者，建立连接，创建队列queue4，创建两个接收,自动确认

消费消息

删除消费者

## 测试最大连接

### max\_connect\_test.cpp

不断创建连接，直至报错，打印建立的n个连接

删除所有连接

## 多个生产者和消费者 ？？没有理解

### multi\_producer.cpp

创建exchange queue并绑定

创建指定长度的消息

注册信号处理函数

### multi\_consumer.cpp

## 普通生产者和消费者

### normal\_producer.cpp

创建生产者，建立连接，创建交换机echange1和两个队列queue6,queue7，通过不同的路由键绑定队列，将消息通过其中一个路由键传递

删除交换机和生产者

### normal\_consumer1.cpp

创建消费者，建立连接，创建队列queue6，订阅队列接收（自动确认），消费消息

删除队列和消费者

### normal\_consumer2.cpp

创建消费者，建立连接，创建队列queue7，订阅队列接收（自动确认），消费消息

删除队列和消费者

## 有优先级的生产者和消费者

### priority\_producer.cpp

创建生产者，创建交换机exchange1，队列queue3，创建队列的时候指定优先级为10，将队列绑定到交换机上，将消息通过其中路由键传递

删除交换机和生产者

### priority\_consumer.cpp

创建消费者，建立连接，创建队列queue3，订阅队列接收（自动确认），消费消息

取消订阅，删除队列和消费者

# 互斥锁

通过信号量实现互斥锁，便于加锁解锁以及初始化

## sem\_lock.cpp

构造函数

SemLock::SemLock()

信号量锁和id都设置为0

析构函数：无

SemLock::~SemLock()

初始化

int SemLock::Init(int iSemKey)

创建了一个权限为666的信号量，初始化该信号量，未曾使用或者上次op操作超过3分钟则释放锁

上锁

int SemLock::Lock()

对该信号量执行等待操作

解锁

int SemLock::UnLock()

对信号量挂起操作

# 共享内存队列

实现基于共享内存的队列，便于进程间通信和数据持久化

## shm\_queue.cpp

队列头和数据块头

//队列头，包括：队列长度，头尾数据块位置及数据块个数,使用者数目

typedef struct queue\_head\_s

{

int m\_iLen;//队列长度

int m\_iHead;//头尾数据块位置

int m\_iTail;

int m\_iBlockNum;//数据块个数

int m\_iUsedNum;//使用者数目

}QueueHead;

//数据块头，包括：当前块起始位置及数据长度

typedef struct date\_block\_head\_s

{

int m\_iIndex;

int m\_iDateLen;

}DateBlockHead;

构造函数

ShmQueue::ShmQueue()

队列头。队列大小，互斥锁，内存地址均设置为空

析构函数

ShmQueue::~ShmQueue()

删除互斥锁

初始化队列

int ShmQueue::Init(int iShmKey,int iQueueSize)

创建信号量互斥锁

创建或者打开共享内存，成功返回共享存储的id，不成功重新换位置创建

最后初始化队列头，重新调整队列指向

数据入队

int ShmQueue::Enqueue(const char \*ipDate,int iDateLen)

获取使用权（cas32）

计算剩余空间，看能否存放

如果能存放，初始化数据块头部，用队列头部长度减去尾部判断：

若是尾数据块位置在首数据块位置之前，或者尾数据块之后剩余空间足够大，可以直接全部存放（数据块）

否则分开保存数据头、数据体、尾部标志

更新尾部位置及数据块个数

释放使用权

数据出队

int ShmQueue::Dequeue(char \*opBuf,int \*iopBufLen)

获取使用权

读取数据块头

缓冲区长度不够则返回

数据块位置不是队列首位置，则查找下一个数据块位置或者置空队列，反之数据块是队首位置，读取长度不对直接置空队列或者直觉兑取数据部分和尾部部分

释放使用权

获取数据块个数

int ShmQueue::GetDateBlockNum()

加锁

读取数据块个数

解锁

# 客户端服务

## 客户端连接对象

定义客户端链接对象

### struct ClientConnect

客户端连接

typedef struct

{

//连接的套接字

int m\_iSockfd;

//客户端ip

unsigned int m\_iClientAddr;

//上次收到数据时间

time\_t m\_tLastRecvTime;

//连接是否被使用

bool m\_bUsed;

//连接是否被放入map

bool m\_bInMap;

//在保存连接数组中的下标

int m\_iIndex;

//接收缓冲区

char \*m\_pRecvBuff;

//接收缓冲区尾部

char \*m\_pRecvEnd;

//接收到的数据头部

char \*m\_pRecvHead;

//接收到的数据尾部

char \*m\_pRecvTail;

//发送缓冲区

char \*m\_pSendBuff;

//发送缓冲区尾部

char \*m\_pSendEnd;

//发送的数据头部

char \*m\_pSendHead;

//发送的数据尾部

char \*m\_pSendTail;

}ClientConnect;

## 连接池

### clientConnect.cpp

管理所有的客户端连接，相当于一个连接池

class ClientConnectManager

{

public:

unsigned long MIN\_POINTER\_ADDRESS;//所有连接的起始地址

unsigned long MAX\_POINTER\_ADDRESS;//所有连接的终止地址

const static int SUCCESS=0;

const static int ERROR=-1;

public:

ClientConnectManager();构造函数 初始化所有苦短连接

~ClientConnectManager();析构函数 删除所有连接

//判断地址是否有效

bool IsAddrValid(ClientConnect \*ipClientConnect);

//获取一个可用连接 连接地址需要符合要求 地址大小不可能超过客户端连接的大小

}

ClientConnect \*GetOneFreeConnect();

//客户端退出函数

void ClientExit(ClientConnect \*ipClientConnect,int iEpollfd);

//根据下标查找用户 使用map查找下标

ClientConnect \*FindClient(int iIndex);

//添加在线用户 直接添加到map中

int AddOnlineClient(ClientConnect \*ipClientConnect);

private:

//在线用户

unordered\_map<int,ClientConnect \*>m\_mOnlineClient;

//可用连接

list<ClientConnect \*>m\_lFreeList;

//所有连接

ClientConnect \*m\_pAllClientConnect;

};

## 和客户端通信

接入层，负责和客户端通信，和业务逻辑层交互

### connectServer.cpp

构造函数

ConnectServer::ConnectServer()

初始化各变量

析构函数

ConnectServer::~ConnectServer()

删除各连接

创建对象

ConnectServer\* ConnectServer::GetInstance()

创建一个和客户端通信服务对象

删除

void ConnectServer::Destroy()

删除和客户端的通信

初始化

int ConnectServer::Init()

初始化信号量

创建epoll句柄进行时间监听

创建socket,bind,listen

注册事件（传入epoll句柄、socket)

初始化和业务逻辑层的通信通道 共享内存的队列

从客户端发的包得到数据流

int ConnectServer::GetClientPackageFromBuff(char \*ipBuff,int ipBuffLen,int iClientIndex,char \*\*opNextPack,int \*opPackLen)

如果数据包长度小于定义的网络包头部长度、数据包长度小于定义的网络包长度、定义的网络包长度小于定义的网络包头部长度，则直接返回

保存客户端下标，指向下一个包位置

将数据发送到客户端

int ConnectServer::SendDataToClient(const char \*ipData,int iDataLen,ClientConnect \*ipClient)

等待套接字可写（fds[0].revents&POLLOUT），给套接字发送对应数据

处理本轮的所有事件数据

int ConnectServer::ProcessEpollData(struct epoll\_event \*ipEvents,int iNum)

监听的事件可读

事件的套接字和之前申请的套接字相同，建立接收和发送缓冲区，获取一个连接对象，添加到在线用户链，现有连接有数据过来（recv），将收到的数据分包处理，小于0，说明缓冲区无数据可读或者是告知业务逻辑层，推出客户端，等于0推出客户端，

收到其他事件，断开连接，告知业务逻辑层

运行

int ConnectServer::Run()

记录本轮是否有数据需要处理，记录处理完epoll数据的时间。

接收的数据入队，将消息放入缓冲区套接字发送缓冲区，若缓冲区空间不足，则先发送数据，再放入发送缓冲区，如果对方关闭告知业务逻辑层。

遍历所有的客户将数据发送出去

# 接收接入层包

业务逻辑层头文件，接收接入层包，执行对应操作或者回包

## logicServer.cpp

### MessageQueue类

构造函数

MessageQueue::MessageQueue(const string &istrName,short iPriority/\*=-1\*/,bool ibDurable/\*=false\*/,bool ibAutoDel/\*=true\*/)

:m\_strQueueName(istrName),m\_iPriority(iPriority),m\_bDurable(ibDurable),m\_bAutoDel(ibAutoDel)

清空普通消息、队列的订阅者、上次推送的订阅者、队列中的消息的确认级别、持久化序号

推送消息

SeverStoreMessage \*MessageQueue::PopFrontMsg(bool ibPush/\*=false\*/)

当需要推送并且队列的订阅者为空，存放的优先消息和普通股消息都为空，则直接返回空

若优先消息不为空，则推送优先消息，反之推送普通消息，推送完的数据丢弃

若消息是用来推送，从订阅者中选择一个接收，上次推送给了那一个订阅者，每次都从上次分配的下一个开始

增加消息

int MessageQueue::AddMessage(SeverStoreMessage \* ipMsg)

若队列和消息都支持优先级，则作为优先级消息存放（消息的优先级大于队列的，则设置为队列的优先级），否则作为普通消息

清除消息

int MessageQueue::ClearMessage()

把优先级队列和普通消息列表中数值一个个删除

增加订阅

int MessageQueue::AddSubscribe(int iCliIndex,unsigned char iConfirmLevel)

遍历队列的订阅者，查看是否有该客户端下标，如果没有则将其压入队列中

取消订阅

int MessageQueue::CancelSubscribe(int iCliIndex)

遍历队列的订阅者，查找需要取消的该客户端下标

如果取消后队列订阅者大小为空，消息确认级别设置为0

序列化所有队列订阅

int MessageQueue::SerializeSubcribeToString(char \*ipBuffer,int &iBuffLen,bool ibIsCreate)

将当前所有队列的创建或取消订阅、队列名称、消息确认级别、每个订阅者都保存下来

### Exchange类

创建绑定

int Exchange:: CreateBinding(const string &istrBindingKey,MessageQueue \*ipMsgQueue)

查找是否存在现有绑定，如果没有绑定则加入根据binding key 绑定的queue中，如果有绑定则根据binging key找到绑定的队列集，将当前队列加入其中

路由键和绑定建是否匹配

bool Exchange::IsFit(const string& istrRoutingKey,const string& iStrBindingKey)

将routingkey根据”.“分割，保存在vStrRoutingKey中， //将bindingkey根据”.“分割，保存在vStrBindingKey中。dp对两者进行匹配判断

根据交换机类型找到绑定的队列

int Exchange::FindBindingQueue(const string& istrRoutingKey,list<MessageQueue \*>&oMsgQueueList)

将消息广播 fanout 1：找到所有绑定的队列，将其存入队列列表中

根据绑定键有选择的接收消息 direct 2：根据绑定队列的map的键，找到队列存入队列列表中

基于主题接收消息 topic 3：根据两者key是否匹配找到队列存起来

删除绑定队列

int Exchange::DeleteBindingQueue(MessageQueue \*ipMsgQueue)

在当前绑定键下的所有队列查找对应队列删除，如果遍历的绑定键的队列为空直接删除绑定键

序列化所有绑定

int Exchange::SerializeBindingToString(char \*ipBuffer,int &iBuffLen)

将每个绑定下的队列的绑定名称、交换机名称、队列名称、路由键进行保存

### LogicServer类

逻辑服务

构造函数

LogicServer::LogicServer()

清空停止标志、和接入层通信的消息队列、和持久化层通信的消息队列、待消费者确认的消息编号、接收区和发送区计数

析构函数

LogicServer::~LogicServer()

删除两个队列：和接入层通信的消息队列、和持久化层通信的消息队列

创建对象

LogicServer \* LogicServer::GetInstance()

创建接入层进行逻辑服务对象

删除对象

void LogicServer::Destroy()

删除创建的对象

初始化信号处理函数

int LogicServer::InitSigHandler()

指定一个新的信号处理函数和进程信号集合

初始化

int LogicServer::Init()

初始化和接入层的通信通道，初始化和持久化层的通信通道

重启时尝试读文件回复数据

int LogicServer::OnInit()

初始化队列 int LogicServer::OnInitQueue() 对queue：查看queue文件夹是否存在，遍历文件夹中的所有子文件夹，对目录下的每一个文件检测其数据文件、持久化写位置文件、订阅文件、读订阅者，并进行数据读取

初始化exchange int LogicServer::OnInitExchange() 对exchange：查看exchange文件夹是否存在，遍历文件夹中的所有子文件夹，对目录下的每一个文件检测其数据文件、绑定文件、绑定关系、读订阅者（读绑定key和对应数目、读queue名称），并进行数据读取

初始化msg int LogicServer::OnInitMsg() 对msg：查看queue文件夹是否存在，遍历文件夹中的所有子文件夹，对目录下的每一个文件：获取文件夹中所有的index文件名称，排序保证先处理最小文件，遍历所有文件，依次进行处理

根据消息下标在文件夹中找消息所在文件组名称

string LogicServer::FindMessageFileName(int iMsgIndex,const char \*ipPath)

遍历文件夹中的所有子文件夹，找到每个文件中.index的位置，拷贝文件名称，如果在消息下标范围内，则返回该名称

将消息下标转换为字符串，不足10位的前面补0

string LogicServer::ConvertIndexToString(int iIndex)

根据消息下标找到索引文件

int LogicServer::FindPosInIndexFile(int iMsgIndex,const char \*ipFile)

打开当前索引文件，通过mmap映射物理地址到user space的虚拟地址指向索引文件，通过二分法查找消息下标对应的文件位置

处理每一个数据文件

int LogicServer::ProcessDataFile(const char \*ipFile,int &oLastIndex,int iStartPos/\*=0\*/)

打开文件，从起始位置读数据，查找队列是否存在（根据名称保存的未订阅队列对象、根据名称保存的订阅队列对象中找），处理数据（offset）,将消息保存在队列中

找到文件夹中所有的index文件名称

int LogicServer::FindIndexFiles(const char \*ipDir,vector<string>&ovIndexFiles)

遍历文件夹中的所有子文件夹，找到每个文件中.index的位置，拷贝文件名称

处理持久化消息，进行恢复

int LogicServer::ProcessDurableMsgFile(const char \*ipFile)

获取数据文件和索引文件是否存在，将索引文件（文件名+.index）和数据文件（（文件名+.data）映射到内存，将索引文件和数据文件映射到内存

将已经消费消息存入set（文件名+.consume）（在文件大小内映射内存，读取数据）

遍历所有消息，将未消费消息添加进队列（读取消息下标和位置，若消息已经被消费，则跳过，反之读取数据，存入队列中）

将消息队列数据推送到接入层

int LogicServer::PushMessage()

根据名称保存的订阅队列对象。读取所有订阅队列的对象，检查一下是否订阅，没有订阅则删除当前名称，当当前队列名称存入

有订阅者则推送消息，若是需要确认，则消息放入待确认队列，反之设置消息确认标志为-1

推送消息，如果消息确认级别等于0，若消息是持久化的，则向持久化层发包

创建交换机

int LogicServer::OnCreateExchange(char \*ipBuffer,int iLen,string &ostrMsgBody)

读名称、exchange 类型、持久化信息、自动删除信息

在根据名称保存的exchange对象中查看是否存在同名的exchange，存在则直接返回，

反之，将exchange名字存在保存exchange对象中，若是需要持久化的exchange,则重新组包发送给持久化层（将包在持久化层中入队）

创建队列

int LogicServer::OnCreateQueue(char \*ipBuffer,int iLen,string &ostrMsgBody)

读名称、优先级、持久化信息、自动删除信息，在根据名称保存的未订阅队列对象中查看是否存在同名队列，存在直接返回，反之新建一个MessageQueue对象，若是需要持久化的exchange,则重新组包发送给持久化层

创建绑定

int LogicServer::OnCreateBinding(char \*ipBuffer,int iLen,string &ostrMsgBody)

读exchange名称、queue名称、读binding key ,在根据名称保存的exchange对象中查看是否存在同名的exchange，在根据名称保存的未订阅队列对象中查看是否存在同名队列，存在才能创建绑定，有一个不存在则直接返回，若是需要持久化的exchange则将当前绑定者发送给持久化层

生产消息（根据exchange和绑定的队列）

int LogicServer::OnPublishMessage(char \*ipBuffer,int iLen,int iCliIndex,string &ostrMsgBody)

读exchange名称、routingkey、优先级、持久化消息、消息序号、确认级别。

检查是否是未确认的重复消息，如果消息在生产者发来的消息集合内（即时没有确认的重复消息），声明一个确认消息对象，将确认包放入和接入层通信的队列中

检查exchange是否存在。查找绑定队列

读消息体，若需要持久化，则设置持久化下标,并发包给持久化层

若是单个确认,直接确认（确认包入队），若是批量确认，则保存最新序号

发送双端确认

int LogicServer::SendMultiAck()

按照批量确认的序号将确认包入队

接收消息

int LogicServer::OnRecvMessage(char \*ipBuffer,int iLen,int iCliIndex,string &ostrMsgBody)

读取queue名称、确认级别

检查queue是否存在

若需要确认，则设置消息确认号

若消息是持久化的，则向持久化层发包

消息放入队列（和接入层通信的消息队列）

创建订阅

int LogicServer::OnCreateSubscribe(char \*ipBuffer,int iLen,int iCliIndex,string &ostrMsgBody)

读取queue名称、确认级别

检查queue是否存在

若是需要持久化的queue,则将当前订阅消费者发送给用户

若原先未订阅，增加订阅

删除交换机

int LogicServer::OnDeleteExchange(char \*ipBuffer,int iLen,string &ostrMsgBody)

读交换机名称，查看是否存在该交换机

若是需要持久化的exchange，则发送删除消息，反之则直接删除（在根据名称保存的exchange对象中）

删除队列

读queue名称，检测queue是否存在，存在则直接在未订阅队列/订阅队列中删除

若是需要持久化的queue，则发送删除消息

清空所有消息（优先消息和普通消息）

遍历所有exchange，删除对应队列的绑定（若删除后交换器为空，且交换器也是自动删除的，则删除exchange）

取消订阅

int LogicServer::OnCancelSubscribe(char \*ipBuffer,int iLen,int iCliIndex,string &ostrMsgBody)

读取queue名称

看是否已经订阅（在未订阅的队列里查找）

检查queue是否存在（在订阅的队列里查找

若是需要持久化的queue,则将当前订阅消费者发送给用户

将队列从订阅队列中取出

若是自动删除队列，则在订阅对象为空时，清空队列消息并删除队列（遍历所有exchange，删除对应队列的绑定，若删除后交换器为空，且交换器也是自动删除的，则删除exchange，若是需要持久化的exchange，则发送删除消息，若是需要持久化的queue，则发送删除消息

）

没有消费者订阅时放入未订阅队列

客户端确认

int LogicServer::OnClientAck(char \*ipBuffer,int iLen,int iCliIndex)

读确认信息

如果是单个确认，从待确认链表中找到对应消息进行移除，若消息是持久化的，则向持久化层发包

如果是批量确认，从待确认链表中找到同一客户发送的序号小于它的消息并删除，若消息是持久化的，则向持久化层发包

客户端退出

int LogicServer::OnClientExit(int iClientIndex)

（1）若退出的是生产者，则需要在已收到确认消息集合中删除掉该生产者

（2）若退出的是消费者，遍历所有队列，取消该消费者的订阅

取消之后队列订阅者大小减少说明该客户订阅了队列

若是需要持久化的queue,则将当前订阅消费者发送给持久化层

若正好是最后一个用户，现将队列从订阅集合中移除

1. 若退出是消费者，则还需要遍历待确认链表，将消息放回原先队列

获取网络包数据

int LogicServer::ProcessRecvData(char \*ipBuffer,int iLen,int &opClientIndex,string &ostrMsgBody)

读取消息长度

查找对应客户端下标

读取消息类型

按照消息类型创建

回复发送的消息

int LogicServer::ReSendMsg()

从待确认链表中找到对应消息进行移除

获取等待确认花费时间，3秒未确认则重发

运行

int LogicServer::Run()

先处理消息队列中的订阅消息，然后再处理共享内存队列中的消息

设置最多接收800个包：处理数据，回复消息

最后发送批量确认包

# 持久化服务

## PersistenceServer.cpp

unordered\_map<string,string>m\_mMsgQueueTailFile;//每个队列当前写入文件

unordered\_map<string,unordered\_map<int,string>>m\_mMsgQueueIndexFile; //每个队列消费index对应文件

unordered\_map<string,list<string>>m\_mMsgQueueList; //每个队列所有组文件名链表，从小到大排序

//初始化文件信息

int InitFileInfo();

//找到文件夹中所有的index文件名称

int FindIndexFiles(const char \*ipDir,vector<string>&ovIndexFiles);

//处理index文件

int ProcessIndexFile(string istrQueueName,const char \*ipFile);

//清理文件，将连续的消费数目超过50%的文件进行合并

int ClearUpFile(int &oClearCount);

//计算该组文件消费比率

double CalculateCondsumeRate(const char \*ipFile);

//合并文件

int MergeFiles(const char \*ipQueueName,const char \*ipFileName1,const char \*ipFileName2);