IM protocol-draft V 0.1

Date: 2010-06-29 version: 0.1 Author: William Lv

Description: Message protocol.

Msg : Message

Seg: Segment

Len: Length

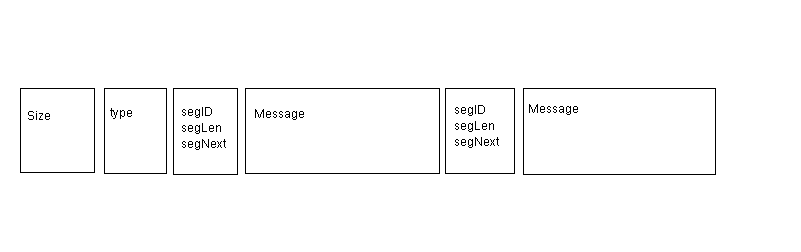
quint8 足够，按照type的定义，估计只有8种

1. Msg protocol

Head content:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Data type | Max Len | Description |
| Size | quint32 | 2^32 | Whole message length |
| Type | quint8 | 2^8 | Message Type |
| SegID | quint8 | 2^8 | Segment ID |
| SegLen | quint16 | 2^16 | Segment length |
| SegNext | quint8 | 2^8 | If has next segment |
| Message |  |  | Message input by user/server |

为啥需要多个segment？只有一个是不是让协议更简单些，程序处理也方便些。



我们有多种消息，如text, file, picture, or others，怎么区分？

可以在Message Fragment中的Type区分。

1. Size (total size in byte)
2. Type :

0x00是什么意思，或者是否放在这里而已，没有什么意义？

删除

~~Default value: 0x00~~

|  |  |  |
| --- | --- | --- |
| 0x00 | Default value |  |
| 0x01 | Syn message, sent by server and client automaitcly to check connection status.More details inform written in first Message segment. |  |
| 0x02 | Login message, sent by client and the server sent 0x03 as response. |  |
| 0x04 | Sent by server to tell client that connection accpected and server is ready for next message. | Mutex, can’t appear with 0x08 in one message. |
| 0x08 | Sent by server to tell client that connection build failed. Failed reason should be written in Message segment. | Mutex, can’t appear with 0x04 in one message. |
| 0x10 | User message. |  |
| 0x20 | Reserve for next version. |  |
| 0x30 | Reserve for next version. |  |
| 0x40 | Reserve for next version. |  |

除了标成黄色的不能同时出现，其他的0Xxx可以同时出现，多种类型直接相与。提高传输效率。

但这样会增加编程的复杂性

同意你的观点.

例如：

一条消息包含：同步+用户发送的信息 格式如下：

0X01 & 0X10 = 0X11

0X11 --十六进制 00010001 -- 二进制

C． SegID

标识每一段Message segment 。

minimum：0x00

maximum：0x0F

1. SegLen

Message segment length.

E. SegNext

0XFF means no message segments behind.

Otherwise more message segment follow behind.

Example:

void MainWindow::sendData()

{

QByteArray block;

QDataStream out(&block, QIODevice::WriteOnly);

out.setVersion(QDataStream::Qt\_4\_6);

quint32 blockSize = 0;

quint8 blockType = 0;

quint8 segID = 0;

quint32 segLen = 0;

quint8 segNext = 0xFF;

QString content = msgEdit->toPlainText();

segLen = content.size() + sizeof(segLen) + sizeof(segID);

out<<blockSize<<blockType<<segID<<segLen<<segNext<<content;

out.device()->seek(0);

out<<quint32(block.size() - sizeof(quint32));

tcpSocket->write(block);

Answer: 一个Message固定大小, 但可能有填充, 需要删除这些填充数据

目前开发就设置成0。

}

Question：在这里对消息进行细化？如果这些字段是固定的，是不是放在整个数据包的定义里更明显？

1. Message

Question：P2P和P2M在实际应用过程中有什么用处？

Length: Message segment length (size in byte)

Type:

|  |  |  |
| --- | --- | --- |
| 0x00 | Default value |  |
| 0x01 | P2P Message | Person to person message. Include simple text message and multi-media message. |
| 0x02 | P2M | One to multi message. |
| 0x04 | SCM | Server control message. |
| Other for reservation |  |  |
|  |  | Question: 会在什么场景下用到Friend SegID? |
|  |  |  |

Answer：用作分别群发和点对点发送。比如私聊和群聊

Friend SegID:

One message segment may be not enough to carry all data. Friend SegID tell client that this message should be operated with following message segments identified with Friend SegID.

Answer：比如一个Message Fragment实际数据大小只有 64/128Byte，如果一次更新200个好友在线（以及其他信息）的信息，可能会超过64/128Byte，这就需要多个Message Fragment 来装这些信息。

又如，传送文件时，也可能会占用很多Message Fragment，不过传送文件可以另设新协议。

再如：实际一条信息有3种：Text、Picture和HttpLink，简单处理就用Friend Fragment去装。

Extension: Reserve for next version.

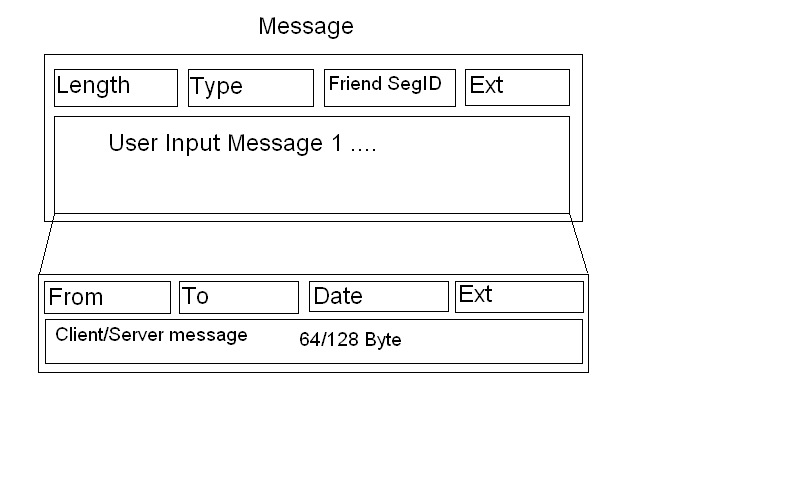
1. May used as link priority.
2. May used as connection method. ( A. One time connection; B. Temporary/short connection; C. Long time connection )

什么场景下用到这些？

1. May used as contain more control/other information

基于TCP/UDP的协议，已经知道了From和To，Date也应该知道了，除非是离线消息.

关于Ext最初考虑用来 给未来扩展做接口。昨天（0629）会议讨论到需要应用层抽象UDP长连接，connection method为其留了一个接口。



To可以不用IP地址（或者其他ID），可以是简短的由服务器分配的ID（或者特定的名字），发送的消息可以是群发（P2M）, 就需要对应的GroupID , 这就需要To来标记了.

Message maximum size (in bytes) =

4(size) + 1(type) + 1(Segment ID) + 2(Segment length) +

[2(Length in message segment) + 1(type) + 1(Friend SegID) + 4(Ext) +

4(From) + 4(To) + 4(Date) + 4(Ext) + 128(User/Server input message)] \* 0x0F = 8 + 2432 = 2440Byte

16 Client (Server) messages / Maximum Size with 16 message fragments

= 2048 / 2440 = 0.84

Message minimum size (in bytes) =

4(size) + 1(type) + 1(Segment ID) + 2(Segment length) +

2(Length in message fragment) + 1(type) + 1(Friend SegID) + 4(Ext) +

4(From) + 4(To) + 4(Date) + 4(Ext) + 128(User/Server input message)

= 8 + 8 + 16 + 128 = 160

One Client (Server) message / Minimum Size with one message fragment

= 128 / 160 = 0.8