****
本 demo 程序由自 32 班开发，目的是为自 32 同学的计网大作业提供互通的基础协议。
如需转载或使用请联系朱天奕，并在代码中附带本文件，并声明陈炜祥、刘天宜、欧阳
*佳弘、万琳莉和朱天奕为 demo 做出的贡献。.本 demo 程序在 vs2012 下开发用 c++11 *
*开发。 *
****

1.协议介绍

本 demo 的目的是实现一个协议的基础，方便同学们之后的调试。该协议可以和 c# demo，java demo 完成通信。在移植和拓展自己的程序时，请按照推荐的方式进行拓展，不要改变基础协议的通信方式和协议构造，否则将失去和其他人协议使用者通信的能力。该协议（下称 32 协议）是应用层协议，其依赖于传输层的 TCP/IP，使用套接字实现通信功能。协议本身可以再分成字节流层，封装层和表意层。

1.1 表意层:作为协议内部的最顶层，包含了实际发送的文字字符串信息，发送人昵称，日期，信息类型等。可以在表意层拓展更多的表意项，在 demo 中只实现了文本的表意。

1.2 封装层：将表意项进行封装的层。其作用是合理可靠地分割、紧密经济地排布表意项，并且可以容易地实现和表意层的互相转换。封装层采用的格式是 JSON（http://baike.baidu.com/view/136475.htm），这是一种广泛使用的简单传输格式，容易实现我们需要的封装功能。请严格遵循 JSON 的封装规则，请不要改动 demo 中的 json 读、写、解析部分代码和静态链接库，否则将不能实现和任何人的通信。具体的 JSON 格式中的必选项在协议中有规定，可选项已经预留了拓展的接口，见 1.4 项的样例。

1.3 字节流层：封装完毕的封装层实际上是一串字节。为了便于调用下层的协议和实现可靠、安全的传输，字节流层进一步将上层处理完毕的字节流进行转码，以 utf-8 的格式形成字节流。在数据字节流的前方添加首部“0x32,0xA0”,在首部之后添加八个字节长的字节流长度字段。这里的字节流长度是包括了首部、长度字段、数据字段、结尾标识符的。长度用一个 8 byte 的无符号整数表示，以低位在前高位在后地方式存储在 8 个字节当中。在数据字节流之后天界结尾标识符“0x42,0xF0”。请不要修改字节流层的封装形式。

1.4 具体的协议形式：

1.4.1JSON 示例：

从这个冒号开始：

```
{
    "id":1,
    "type":"text",
    "time":"2015.11.23 23:54:24",
    "else":{},
    "data":{
        "name":"sd\"",
        "text":"sd/n    s\r\n\""
    },
    "md5":"c99108bdd8b73dba288c3adb60e22f79"
}
```

：到冒号结束。

JSON 必须包含在一对{}之间，用“”来标记表意项，用：来分割表意项和表意值，用，来分割不同的表意项。必选表意项“id”整型，可拓展用途；“type”字符串，作为枚举类型，已有“text”、“respond”、“exit”三类；“time”字符串，时间；“data”，Json 结构，“respond”、

“exit”为空，“text”必须包含“text”和“name”，二者都是字符串，并且都使用 md5 进行编码后才存入；“else”，json 结构，目前为空，可供拓展；“md5”，校验码，对之前的所有除“else”新增项的表意内容进行校验。校验方式是依次串接所有的内容的字节流，也就是内存中的形式，然后进行 md5 运算。md5 可以将不定长的字节串哈希映射到一个 32 位长的字节串，这样做可以有更大的把握将错误检出。尤其注意转义字符，如果输入了转移字符，在字节流层的体现可能是\加转移字符，即为了保证 JSON 的格式，需要二次转义。在 md5 校验时，我们对最终封装好的封装层进行编码校验，也就是二次转义后的字节流进行校验。二次转义理应在封装层完成。

1.5 传输层：传输层采用 TCP 协议，用 socket 进行通信。

1.6.1 应用层的 Client-Server 的建立逻辑。每台主机都拥有一个 Server，这个 Server 以一个套接字监听本机的 3232 端口。在得到了通信请求后，将转移到一个新的端口建立一个新的套接字和对方的套接字进行通信。这个新的套接字我们称之为“接收 Socket”，而监听的套接字成为“监听 Socket”。32 协议定义任何两台主机之间的通信是全双工的，因此我们为任何一个通信都要建立一个双向的 TCP 链接。因此，“接收 Socket”只接收，通过本主机的 Client 建立一个新的返回的 Socket 和对方的主机的 Server 上的“接收 Socket”建立 TCP 连接。这个新的返回通信的 Socket 称为“发送 Socket”。因此，每当有一对通话建立，就会产生一对 TCP 链接，在任意一方都有一个“接收 Socket”和一个“发送 Socket”，分别只接收和只发送，可以定义方向。值得注意的是，建立“发送 Socket”前应当查看到这个 ip 的发送方向 Socket 是否已经存在，以节约内存空间。如果要对这部分逻辑进行优化，请遵循协议的逻辑思路，以保证和协议内的通话能够正确建立。

1.6.2 应用层的 Client-Server 的接收、发送的逻辑：在建立完成之后，每当“接收 Socket”收到一个有效的“type”是“text”的包，就调用对应对话的“发送 Socket”发送一个“type”是“respond”的回包。对于这个回包，对方自然是在“接收 Socket”收到，目前的处理方法为空，也并不会等待其收到，可以进行拓展。

1.6.3 应用层的 Client-Server 的关闭逻辑：每一个 Socket 通过软状态维护。在程序关闭时，会发出“type”=“exit”的包，用来表示主动退出消息。如果出现主动退出，“接收 socket”的主机会注销相应的对话资源。

2.架构介绍

****

类 1：client

（1）通信过程：通过 socket 通信，去敲所需要通信对象的“窗口”

（2）编程功能：完成 tcp 连接和发送 message

类 2：server

（1）通信过程：等待（监听）端口敲门

（2）编程功能：长期在线和多线程

注意：通话双方都有 client 和 server

****

类 3：message

功能：校验，封装，解释字符流

****

类 4：json（标准化数据交换格式）

功能：解包和封装包成结构体，运算符重载

****

3.接口介绍

1、json 文件夹中的.h 文件均为系统自带，直接调用即可

2、server 和 client 收发

3、md5.cpp 做 md5 校验和

4、MyJson.cpp 把输入信息传成 Jason 包

5、DA32Protocol.h 所有需要的预编译文件、需要维护的 Client 列表和 Server

6、MyTson.h 定义了信息层类 MyJson

7、message.cpp 封装 Json，加头尾

8、usinginmain.h 中写了主函数中的各种功能

9、main.cpp 主函数

****

usinginmain.h（供 main 调用实现主函数功能）

bool findClient(Client *tofind)返回是否可以找到一个 Client（1 为找到）

void mianThread()是主线程，输入、验证和创建 Client

void quitFunction()实现安全退出

Client* checkClientList(SOCKADDR_IN add)检查客户端链接表，返回当前客户端

void deleteClient(Client* c)删除客户端

void initServer()各种初始化。。。

****

MyJson.cpp

tm convert_string_to_time_t(const std::string & time_string)将固定格式的字符串解析成时间结构 tm

Json 类函数的定义

bool MyJson::getJson(std::string charflow)从字符流中获得一个 Json 返回 1 为成功

bool MyJson::getJson(istream& charflow)从文件中读入一个 Json 返回 1 成功

void MyJson::showJson_in_console()输出 Json

string MyJson::PackJson(std::string input)打包

****

Message.cpp

GBKToUTF8 和 UTF8ToGBK:GBK 编码和 UTF8 编码转换

MyJson Message::getContent(string dataflow)解包，获得 Json

string Message::getWrap(string tosend)封装，返回 UTF8

****