Bug修改记录 11.13

1.18819： LTcpSocket的客户端不绑定socket地址 先给服务端发送数据后 服务端再发送数据给客户端 客户端无法接收数据

- 原因：用了测试的代码，发现我修改过后不是我的问题，是测试代码有点小问题

```
// 客户端类
class TestClient : public LThread {
public:
    bool flag = false;
    int bytes;
    LString m_msg;
    ClientTcpSocket m_cl;
    void run() override {
        LTcpSocket socket(LAbstractSocket::IPv4Protocol);
        socket.setBufferSize(1024);
        LString ip = "127.0.0.1";
        // 直接连接服务端,并发送和接收消息
        flag = socket.connectToHost(LHostAddress(ip), uint16_t(50200));
        socket.sends("please call back if you received my text!");  // 发送
        LThread::msleep(50);
        std::cout << "client starts receiving!" << std::endl;
        bytes = socket.receives();
        std::cout << "client received!" << std::endl;
        m_msg = LString(socket.pData->bytebuffer);  // 这里给注释掉了 ... ,怪不得不对
        socket.closes();                            // 关闭套接字
    }
};
```

这里测试的代码是注释掉的，因此正确获得数据之后没有赋值，导致下面的判断出现了错误...

- 测试：代码在单元测试`Test`里面的`ltcpsocket_test`的`CommunicateTest`，结果如下：

```
[ RUN      ] LTcpSocketTest.CommunicateTest
server_bind: 1
server_listen: 1
 == start waiting client's connection! ==
server_recevice_message: please call back if you received my text!
client_ip: 127.0.0.1:46186
client starts receiving!
sever starts closing!
client received!
sever closed!
[       OK ] LTcpSocketTest.CommunicateTest (1650 ms)
```

2.18811：LUdpSocket的sends(unsigned char* data,int length) 数据长度设置为0 在接收时程序崩溃

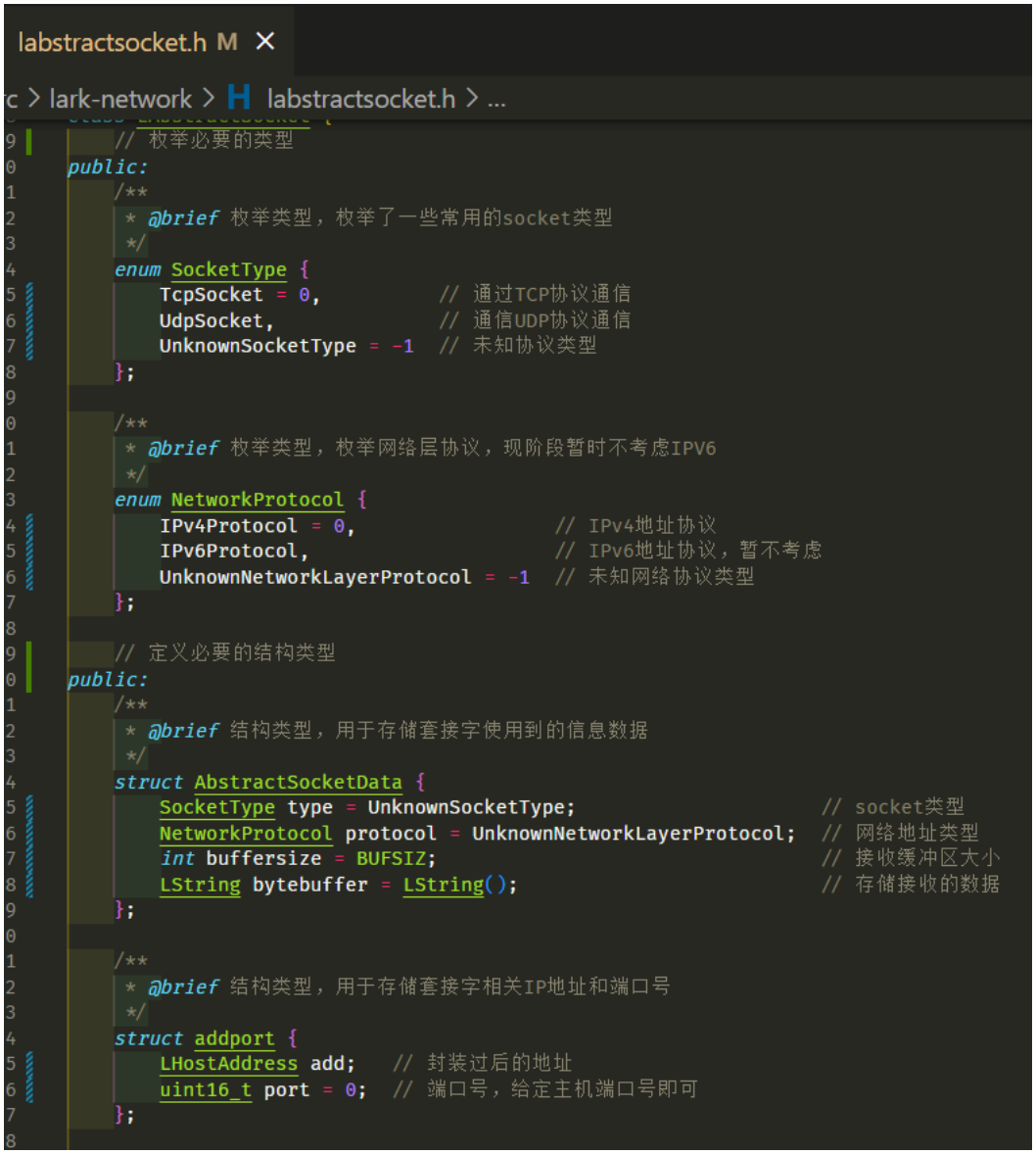
- 解决：跑出来结果是对的，代码在单元测试`Test`里面的`ludpsocket_test`的`CommunicateTest`，结果如下：

```
[ RUN      ] LUdpSocketTest.CommunicateTest
server_bind: 1
client starts sending!!!!
client starts closing!!!! -- 1
server is receiving
get_adress: 127.0.0.1:53547
[       OK ] LUdpSocketTest.CommunicateTest (1131 ms)
```

3.代码的规范

- 按照要求对所有代码做了规范，以下是对`bug`里面的进行呈现
- **19310：请检查所有头文件的结构体和枚举内容是否存在描述**

  `labstractsocket.h`：



```
labstractsocket.h M
c > lark-network > H labstractsocket.h > ...
    // 枚举必要的类型
public:
    /**
     * @brief 枚举类型，枚举了一些常用的socket类型
     */
    enum SocketType {
        TcpSocket = 0,          // 通过TCP协议通信
        UdpSocket,              // 通信UDP协议通信
        UnknownSocketType = -1  // 未知协议类型
    };

    /**
     * @brief 枚举类型，枚举网络层协议，现阶段暂时不考虑IPV6
     */
    enum NetworkProtocol {
        IPv4Protocol = 0,                   // IPv4地址协议
        IPv6Protocol,                       // IPv6地址协议，暂不考虑
        UnknownNetworkLayerProtocol = -1  // 未知网络协议类型
    };

    // 定义必要的结构类型
public:
    /**
     * @brief 结构类型，用于存储套接字使用到的信息数据
     */
    struct AbstractSocketData {
        SocketType type = UnknownSocketType;                    // socket类型
        NetworkProtocol protocol = UnknownNetworkLayerProtocol; // 网络地址类型
        int buffersize = BUFSIZ;                                // 接收缓冲区大小
        LString bytebuffer = LString();                         // 存储接收的数据
    };

    /**
     * @brief 结构类型，用于存储套接字相关IP地址和端口号
     */
    struct addport {
        LHostAddress add;   // 封装过后的地址
        uint16_t port = 0;  // 端口号，给定主机端口号即可
    };
```

`lhostaddress.h`：

lhostaddress.h M

> lark-network > lhostaddress.h > ...

```cpp
#include <cstdint>
#include <string>

#include "lstring.h"

class LHostAddress {
    // 枚举必要的类型
public:
    /**
     * @brief 枚举类型，枚举了一些常用的特殊ip地址
     */
    enum SpecialAddress {
        Null = 0,        // 空地址
        Broadcast,       // 广播地址
        LocalHost,       // 本地链路地址(IPv4)
        LocalHostIPv6,   // 本地链路地址(IPv6)
        // Any,             // 双栈任意地址。与此地址绑定的套接字将在IPv4和IPv6接
        AnyIPv4,   // 任意的IPv4地址
        AnyIPv6    // 任意的IPv6地址
    };

    /**
     * @brief 枚举类型，枚举了IP地址的类型
     */
    enum AddressType {
        Unknown = 0,  // 未知地址类型
        IPv4addr,     // IPv4地址类型
        IPv6addr      // IPv6地址类型
        // AnyIPaddr // 任意地址，暂不考虑

    };
    /**
     * @brief 枚举类型，枚举了端序类型
     */
    enum Endian {
        NBO = 0,  // 网络字节序，即大端字节序
        HBO       // 主机字节序
    };

    // 定义必要的结构类型
public:
    /**
     * @brief 结构类型，用于存储IP地址并且存储了其类型，IP地址使用网络字节序存储
     */
    struct Address {
        in_addr ipv4;      // 存储IPv4地址(IPv4地址使用)
        in6_addr ipv6;     // 存储IPv6地址(IPv6地址使用)
        AddressType type;  // 存储地址类型
    };
```

`lhttpcontrol.h`:

```cpp
class LHttpControl : public LObject {
    // 枚举必要的类型
public:
    /**
     * @enum Operation
     * @brief 指示答复正在处理的操作。
     */
    enum Operation {
        UnknownOperation = 0,   // 未知请求
        GetOperation,           // GET请求
        HeadOperation,          // HEAD请求
        PostOperation,          // POST请求
        PutOperation,           // PUT请求
        DeleteOperation         // DELETE请求
    };
```

`lhttpreply.h`：

```
    // 枚举必要的类型
public:
    /**
     * @enum NetworkError
     * @brief 网络错误。
     */
    enum NetworkError {
        NoError = 0,              // 没有任何错误
        ConnectionRefusedError,   // 远程服务器拒绝连接
        RemoteHostClosedError,    // 远程服务器在处理整个答复之前过早地关闭了连接
        HostNotFoundError,        // 找不到远程主机
        TimeoutError              // 与远程服务器的连接超时
    };

    // 其余成员函数
```

`lhttprequest.h`：

```
    // 枚举必要的类型
public:
    /**
     * @enum KnownHeaders
     * @brief 已知标头。
     */
    enum KnownHeaders {
        ContentDispositionHeader = 0,  // HTTP Content-Disposition标头
        ContentTypeHeader,             // HTTP Content-Type标头
        ContentLengthHeader,           // HTTP Content-Length标头
        LocationHeader,                // HTTP Location标头
        LastModifiedHeader,            // HTTP Last-Modified标头
        IfModifiedSinceHeader,         // HTTP If-Modified-Since标头
        UserAgentHeader,               // HTTP客户端发送的用户代理标头
        ServerHeader                   // HTTP客户端接收的服务器标头
    };

    /**
     * @enum Attribute
     * @brief 属性。
     */
    enum Attribute {
        HttpStatusCodeAttribute = 0,   // 从HTTP服务器接收的HTTP状态码
        HttpReasonPhraseAttribute      // 从HTTP服务器接收的HTTP原因短语
    };
```

`ltcpsocket.h`和`ludpsocket.h`无

- **19309：LHttpRequest头文件中，header_to_rawheader函数命名不规范**

  替换为`headToRawHeader`

```
    // 私有成员函数
private:
    /**
     * @brief 对枚举的已知标头做一个到LString的映射
     *
     * @param header
     * @return LString
     */
    static LString headerToRawHeader(KnownHeaders header);
```

- **19308：LHttpControl头文件中，overtime变量与函数混淆存放**

  已放到`private`成员的位置

```
    // 私有成员变量
private:
    /**
     * @brief 存储请求报文
     *
     */
    LString m_sendMessage = LString();

    /**
     * @brief 存储响应报文
     *
     */
    LString m_recvMessage = LString();

    /**
     * @brief 定义connect连接我们设定的超时时间
     *
     */
    static int overtime;
    /**
     * @brief 存储http状态码到状态短语的映射
     *
     */
    static std::unordered_map<int, std::string> Http_Attribute;
};
```

- **19307：LHttpControl头文件中read_message，deal_redirect，common_request、deal_sigalrm函数命名、参数命名不规范，函数功能描述不完整**

  已修改

```cpp
/**
 * @brief 对捕捉到的SIGALRM信号进行处理
 * @param num ，信号的编号或者宏
 */
static void dealSigAlrm(int num);

/**
 * @brief 封装读取响应报文的函数
 *
 * @param connectFd 套接字的文件描述符
 * @param dataStart 数据段开始的下标
 * @param dataEnd 数据段结束的下标
 * @param contentLength 响应体的长度
 * @param isLocationInHeader 是否检测到重定向
 * @return LString 响应报文
 */
LString readMessage(int connectFd, size_t &dataStart, size_t &dataEnd, size_t &contentLength, bool &isLocationInHeader);

/**
 * @brief 封装处理重定向的函数
 *
 * @param readMessage 响应报文
 * @param request 请求对象
 * @param op 操作类型
 * @param data post，put请求携带的data，其他请求给空字符串即可
 * @return LHttpReply* 响应对象
 */
LHttpReply *dealRedirect(const LString &readMessage, const LHttpRequest &request, int op, const LString &data);

/**
 * @brief 封装通用的发送请求的函数
 *
 * @param request 请求对象
 * @param op 操作类型
 * @param data post，put请求携带的data，其他请求给空字符串即可
 * @return LHttpReply* 响应对象
 */
LHttpReply *commonRequest(const LHttpRequest &request, int op, const LString &data);
```

- **19305：LHostAddress头文件，删除protected**

  已删除

```
private:
    /**
     * @brief 创建socket套接字，将套接字的文件描述符存储下来
     * @return true 套接字创建成功
     * @return false 套接字创建失败
     */
    bool createSocket();

    // 公有成员变量
public:
    /**
     * @brief socket描述符
     */
    int socketfd = -1;

    /**
     * @brief addport类型，存储对应的本地地址和端口
     */
    struct addport local;

    /**
     * @brief addport类型，存储对应的对方地址和端口
     */
    struct addport peer;
};
#endif
```

- **19304：LAbstractSocket头文件中AbstractSocketData的实例跟函数写在一起**

  已修改，放到合适的位置

```
    // 公有成员变量
public:
    /**
     * @brief AbstractSocketData类型，用于指向使用的相关数据
     */
    struct AbstractSocketData *pData = nullptr;

    /**
     * @brief socket描述符
     */
    int socketfd = -1;

    /**
     * @brief addport类型，存储对应的本地地址和端口
     */
    struct addport local;

    /**
     * @brief addport类型，存储对应的对方地址和端口
     */
    struct addport peer;
};
#endif
```

- **19303：LAbstractSocket头文件AbstractSocketData结构体中bytebuffer未附初值**

  已修改

```
public:
    /**
     * @brief 结构类型，用于存储套接字使用到的信息数据
     */
    struct AbstractSocketData {
        SocketType type = UnknownSocketType;                        // socket类型
        NetworkProtocol protocol = UnknownNetworkLayerProtocol;     // 网络地址类型
        int buffersize = BUFSIZ;                                    // 接收缓冲区大小
        LString bytebuffer = LString();                             // 存储接收的数据
    };
```

4.19299：LTcpSocket的disconnect函数在未连接情况下调用，返回true

- 解决：给TCP部分增加了一个`isConnected`标记，用于做辅助判断
- 测试：在`LTCPClientDemo`的`test_3()`

```
208
209   void test_3() {
210       LTcpSocket client(LAbstractSocket::IPv4Protocol);
211       // 直接断开连接，应该返回false
212       std::cout << client.disconnect() << std::endl;
213   }
214
```

问题 输出 终端 调试控制台

```
lark5@DavidingPlus:~/Lark5/larksdk/build$ ./snippet/LTCPClientDemo/LTCPClientDemo
0
lark5@DavidingPlus:~/Lark5/larksdk/build$
```

5.19297：LTcpSocket的binds函数，同一端口号不能被多个进程绑定

- 我做了规范后自己就解决了，我也不知道为啥...
- 测试，在`LTCPClientDemo`的`test_4()`

```
lark5@DavidingPlus:~/Lark5/larksdk/build$ ./snippet/LTCPClientDemo/LTCPClientDemo
1
Bind error : creat failed
0
lark5@DavidingPlus:~/Lark5/larksdk/build$
```