Bug修改记录 11.11

1.19292：LAbstractSocket的crateSocket建议私有化 用户不会直接使用

- 解决：已私有化

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

public:
```

2.19289和19290：关于TCP和UDP的send的问题

先声明，我没看懂测试给的测试啥意思，我就按照`bug`的标题来进行处理了哈!测试里面的错误我会在后面做陈述。

- 先看`TCP`：19289，`LTcpSocket`当某一端`buffer`有限时 接收的数据有问题
  - 原因：`TCP`是可靠的数据传输，所以发送的数据大于接收缓冲区的大小的时候，应该要做循环读的处理来保证数据正常的，因为这个时候缓冲区满了，接收方需要等我们从缓冲区中读取数据，让缓冲区有空间，才会继续发送剩下的数据，这也是`TCP`可靠传输的简要原理
  - 解决：做循环处理

```
 */
int LTcpSocket::receives() {
    if (bufferSize() <= 0) {
        std::string message = std::string("缓冲区大小 ") + std::to_string(bufferSize()) + std::string(" 不

        throw LException(LString(message));
    }

    // 循环读取，防止数据丢失...
    char buf[pData->buffersize + 1] = {0};  // 多开一个'\0'
    int ret = -1;
    while (1) {
        bzero(buf, sizeof(buf));

        ret = recv(localfd(), buf, pData->buffersize, 0);
        if (-1 == ret) {
            perror("recv");
            exit(-1);
        }

        // 加入接收字符串
        pData->bytebuffer.append(std::string(buf));

        // 缓冲区没满则退出
        if (ret != pData->buffersize)
            break;
    }

    return ret;
}
```

关于`send`，我没有听说过关于发送缓冲区的事情，当然在我们的类当中也没有相应的定义，`send`函数底层应该是做了可靠的处理的，就是数据量太大分批次发出去，保证可靠性，所以`send`这边就不管了，但是`recv`我们需要做处理，如上

- 测试：`LTCPDemo`的`test3()`

```
lark5@DavidingPlus:~/Lark5/larksdk/build$ ./snippet/LTCPServerDemo/LTCPServerDemo
recv: hello,i am client,got it...1231
lark5@DavidingPlus:~/Lark5/larksdk/build$
```

```
lark5@DavidingPlus:~/Lark5/larksdk/build$ ./snippet/LTCPClientDemo/LTCPClientDemo
sends: hello,i am client,got it...1231
lark5@DavidingPlus:~/Lark5/larksdk/build$
```

关于禅道上的测试，第一个用例和第二个用例没有接受完整其实都是因为没有做循环处理，后面数据有多余的字符我还真不知道了，但是做了这个之后现在我测试过后不会有了

- 再看`UDP`，`UDP`一般用于实时的通信，所以丢包肯定是常有的事情，因此发送的数据太长的话，接收方收到之后只会截取缓冲区的大小部分，其余部分丢失，这才是合理的处理
  - 解决：我对代码做了细微处理

```
 */
int LUdpSocket::receives() {
    if (bufferSize() <= 0)
        throw LException("接收缓冲区大小不对");

    struct sockaddr_in peer_addr;
    socklen_t len = sizeof(struct sockaddr_in);

    // UDP没有连接的建立，很容易出现丢包，所以服务端发送的数据如果一次性超过接收缓冲区大小，那么肯定会丢失后面的
    // 所以这里直接接收即可，否则会阻塞导致程序出现问题
    char buf[pData->buffersize + 1] = {0};
    int ret = recvfrom(localfd(), buf, pData->buffersize, 0, (struct sockaddr*)&peer_addr, &len
    if (-1 == ret) {
        perror("recvfrom");
        exit(-1);
    }

    pData->bytebuffer.append(std::string(buf));

    setPeerAddress(new LHostAddress(peer_addr.sin_addr));
    setPeerPort(ntohs(peer_addr.sin_port));

    return ret;
}
```

  - 测试：`LUDPDemo`的`test3()`

可见服务端只收到了10个字节的数据，这是预料之中的

```
lark5@DavidingPlus:~/Lark5/larksdk/build$ ./snippet/LUDPServerDemo/LUDPServerDemo
receives: "hello,i am"
lark5@DavidingPlus:~/Lark5/larksdk/build$
```

```
lark5@DavidingPlus:~/Lark5/larksdk/build$ ./snippet/LUDPClientDemo/LUDPClientDemo
sends: hello,i am client,got it!1231
lark5@DavidingPlus:~/Lark5/larksdk/build$
```

关于禅道中`UDP`数据处理不稳定，下面的问题是"仅支持小于等于三字节的`UTF8`字符"，额...，这不关我事吧，这是`LString`的锅了，测试生成的随机字符串的某个字符是多字节的，那不就出问题了嘛

当然，如果想要一劳永逸的话，可以把`byteBuffer`的类型改为`std::string`，但是测试之前让我改为`LString`，所以这没办法...

3.18819：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

- 问题：在我这边是正常的，我没看懂测试的程序怎么写的，给的不是很清楚
- 测试：在`LTCPDemo`的`test3()`中，结果如下：

```
lark5@DavidingPlus:~/Lark5/larksdk/build$ ./snippet/LTCPServerDemo/LTCPServerDemo
recv: hello, i am client.
server_ip: 127.0.0.1
server_port: 9999
client_ip: 127.0.0.1
client_port: 38770
sends: hello, i am server.
lark5@DavidingPlus:~/Lark5/larksdk/build$
```

```
lark5@DavidingPlus:~/Lark5/larksdk/build$ ./snippet/LTCPClientDemo/LTCPClientDemo
server_ip: 127.0.0.1
server_port: 9999
sends: hello,i am client.
server_ip: 127.0.0.1
server_port: 9999
recv: hello, i am server.
server_ip: 127.0.0.1
server_port: 9999
lark5@DavidingPlus:~/Lark5/larksdk/build$
```

4.18811：LUdpSocket的sends(unsigned char* data,int length) 数据长度设置为0在接收时程序崩溃

- 问题：同样，在我这边能正确获得
- 测试：在`LUDPDemo`的`test2()`中，结果如下：

```
lark5@DavidingPlus:~/Lark5/larksdk/build$ ./snippet/LUDPServerDemo/LUDPServerDemo
receives:
peeraddress: 127.0.0.1
peerport: 52880
lark5@DavidingPlus:~/Lark5/larksdk/build$
```

```
lark5@DavidingPlus:~/Lark5/larksdk/build$ ./snippet/LUDPClientDemo/LUDPClientDemo
sends: 您好，这里是客户端！
lark5@DavidingPlus:~/Lark5/larksdk/build$
```

5.19286和19288：关于Http请求报文的原始标头问题

- 分析：原来写的一坨大便，我不知道为什么要写那么多`if`，`else`，明明都存到`Map`中了直接遍历这个`Map`不好吗，所以我对这个做了架构上的调整，下面我将一一阐述改动之处(当然有些细微的地方就没有提到哈)
    - `HttpRequest`

      删除了`m_Headers`成员，`Headers`属性在内部经过处理都存到`m_rawHeaders`中，并且给`m_rawHeaders`给定一些初始值，如下

`setCredentials`接口，设置了用户名密码后就立即做转换，然后存到`rawHeader`中，哦，`base64Encode`私有转换函数从`LHttpControl`移动到`LHttpRequest`中

```
void LHttpRequest::setCredentials(const LString& username, const LString& password) {
    this->m_userName = username;
    this->m_passWord = password;

    // 我们需要在这里就把就把用户名和密码的验证加入rawheaders中
    std::string auth = m_userName.toStdString() + ":" + m_passWord.toStdString();
    std::string encodedAuth = base64Encode(LString(auth)).toStdString();
    std::string headerValue = "Basic " + encodedAuth;

    setRawHeader("Authorization", LString(headerValue));
}
```

关于`header`和`rawheader`的一系列函数，`header`在内部都会走`rawheader`的处理方式

```
void LHttpRequest::setRawHeader(const LString& headerName, const LString& headerValue) {
    if (!headerValue.isEmpty()) {
        if (!m_pData->m_rawHeaders.contains(headerName))  // key不存在则插入
            m_pData->m_rawHeaders.insert(headerName, headerValue);
        else
            m_pData->m_rawHeaders[headerName] = headerValue;  // 存在则更新
    }
}

LString LHttpRequest::rawHeader(const LString& headerName) const {
    auto iter = m_pData->m_rawHeaders.find(headerName);
    if (iter != m_pData->m_rawHeaders.end())
        return iter->value();
    else
        return LString();
}

void LHttpRequest::setHeader(KnownHeaders header, const LString& headerValue) {
    setRawHeader(header_to_rawheader(header), headerValue);
}

LString LHttpRequest::header(KnownHeaders header) {
    return rawHeader(header_to_rawheader(header));
}
```

- `HttpControl`

将每个请求函数的设置原始标头的代码简化

```cpp
// 从request中的rawHeaders中读取，并且设置到请求报文中
for (auto &item : request.m_pData->m_rawHeaders) {
    sendMessage.append(item.key());
    sendMessage.append(": ");
    sendMessage.append(item.value());
    sendMessage.append("\r\n");
}
```

- 解决`bug`

  两个测试代码都在`HttpControl`的`testGet2()`中

  可以验证成功，并且后面的设置会覆盖前面的设置(合理)，这里从我修改的代码也可以看出，并且设置了`LocationHeader`在请求报文中也携带了

```
lark5@DavidingPlus:~/Lark5/larksdk/build$ ./snippet/HttpControlTest/HttpControlTest
------------------------
GET /basic-auth/huahua/123456 HTTP/1.1
Host: 192.168.1.211
Accept: */*
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8,en-GB;q=0.7,en-US;q=0.6
Authorization: Basic aHVhaHVhOjEyMzQ1Ng==
Connection: keep-alive
Location: http://192.168.1.211/response-headers
User-Agent: Mozilla/5.0


------------------------
HTTP/1.1 200 OK
Server: gunicorn/19.9.0
Date: Sat, 11 Nov 2023 08:01:01 GMT
Connection: keep-alive
Content-Type: application/json
```