Bug修改记录 10.31

1.18919：LHttpControl的base64Encode 对用户信息编码后进行认证 认证失败

- 原因：第一，测试代码对用户名和密码的格式规范有问题，用户名和密码之间应该用 `':'` 连接；第二，`get`方法的请求有问题，处在对用户名和密码的编码操作当中
- 解决：我做了一些规范，我们设置请求报文的原始标头，传入身份验证的时候就直接传入用户名和密码，不需要手动调用`base64Encode`进行编码，这个东西我们在get请求内部进行，传入用户名和密码使用的是`setCredentials`方法

```
// 我们通过setCredentials接口设置用户名和密码
lr.setCredentials("huahua", "123456");
```

在`get`请求当中对这一块的处理如下：

```
// 处理用户名和密码
if (!request.userName().isEmpty() && !request.password().isEmpty()) {
    // 使用Base64编码用户名和密码
    std::string auth = request.userName().toStdString() + ":" + request.password().toStdString();
    std::string encodedAuth = base64Encode(LString(auth)).toStdString();
    sendMessage.append("Authorization: Basic ");
    sendMessage.append(encodedAuth);
    sendMessage.append("\r\n");
}
```

- 测试：在`HttpControl`中的`testGet3()`，结果如下：

```
Basic aHVhaHVhOjEyMzQ1Ng==

{
  "authenticated": true,
  "user": "huahua"
}


GET /basic-auth/huahua/123456 HTTP/1.1
Host: 192.168.1.211
Authorization: Basic aHVhaHVhOjEyMzQ1Ng==
Accept: */*
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8,en-GB;q=0.7,en-US;q=0.6
Accept-Encoding: gzip, deflate
Connection: keep-alive



HTTP/1.1 200 OK
Server: gunicorn/19.9.0
Date: Tue, 31 Oct 2023 11:31:31 GMT
Connection: keep-alive
Content-Type: application/json
Content-Length: 49
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true

{
  "authenticated": true,
  "user": "huahua"
}
```

2.18917：建议完善LHttpRequest的base64Encode函数的注释

- 原因：是之前没有对用户名和密码到底怎么传入的说的不是很清楚，结合1查看；这个东西我更倾向于是一个私有函数，因为他是被我们的`get`调用的，然后用户使用`setCredentials`方法进行设置用户名和密码，在`get`请求当中会自动调用这个方法进行编码并且加入`Http`请求头
- 解决：已完善

```
/**
 * @brief 对用户名和密码进行正确编码
 * @param LString ，封装用户名和密码信息的LString类型的字符串，其中用户名和密码之间用':'连接，整体为一个LString类型字符串，用户使用setCrede
 * @return 编码之后的字符串对象
 */
LString base64Encode(const LString &str);
```

3.18908：LHttpRequest的setRawHeader设置短链接 设置失败

- 原因：在`get`请求封装`HttpReply`对象的时候，`HttpReply`类带有指针，但是没有做拷贝构造的重写，没有做深拷贝的操作

```
reply->setController(reply, this);
reply->setRequest(reply, request);
reply->setUrl(reply, request);
reply->setPort(reply, request);
reply->setOperation(1);
```

对于`setRequest`函数，传的是一个值，这样就会创建一个新对象并且调用拷贝构造，这样在浅拷贝下就会导致两个对象公用一个数据段，因此当这个对象消亡后，`rawHeaders`中的数据自然就会被释放，这样原来的`rawHeaders`就没了...

```
/**
 * @brief 匹配请求与响应。
 * @param reply 响应对象
 * @param request 请求对象
 */
static inline void setRequest(LHttpReply *reply, LHttpRequest request) { reply->m_request = request; }
```

数据段指针和里面的数据，里面存储了`rawHeaders`

```
private:
    struct LHttpDataStruct* m_pData = nullptr;

    LString m_userName = "";
    LString m_passWord = "";
};
```

```
struct LHttpDataStruct {
    LString m_host = LString();
    LString m_path = LString();
    int m_port = -1;
    LMap<LString, LString> m_rawHeaders;
    LMap<KnownHeaders, LString> m_headers;
    LMap<Attribute, LString> m_attributes;
    int m_redirectCount = 0;
    int m_maxRedirects = 50;
};
```

- 解决：对`HttpRequest`类重写拷贝构造函数和拷贝赋值函数

```
LHttpRequest::LHttpRequest(const LHttpRequest& other) : LHttpRequest() {
    *this = other;
}

LHttpRequest& LHttpRequest::operator=(const LHttpRequest& other) {
    // 检测自我赋值
    if (*this == other)
        return *this;

    delete m_pData;
    m_pData = new LHttpDataStruct;

    m_pData->m_host = other.m_pData->m_host;
    m_pData->m_path = other.m_pData->m_path;
    m_pData->m_port = other.m_pData->m_port;
    m_pData->m_rawHeaders = other.m_pData->m_rawHeaders;
    m_pData->m_headers = other.m_pData->m_headers;
    m_pData->m_attributes = other.m_pData->m_attributes;
    m_pData->m_redirectCount = other.m_pData->m_redirectCount;
    m_pData->m_maxRedirects = other.m_pData->m_maxRedirects;
    return *this;
}
```

- 测试：`HttpRequest`中的`test3()`，禅道上对应了两种情况，我的代码标注了，两个返回的值都是`close`

```
lzx0626@DavidingPlus:~/DavidingPlus/Lark5/larksdk/build$ snippet/HttpRequestTest/HttpRequestTest
close

GET /get HTTP/1.1
Host: 192.168.1.211
Connection: close



HTTP/1.1 200 OK
Server: gunicorn/19.9.0
Date: Tue, 31 Oct 2023 12:10:14 GMT
Connection: close
Content-Type: application/json
Content-Length: 162
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true

{
  "args": {},
  "headers": {
    "Connection": "close",
    "Host": "192.168.1.211"
  },
  "origin": "192.168.7.217",
  "url": "http://192.168.1.211/get"
}


close
```

4.18895：LHttpRequest的setAttribute多次设置失败

- 原因：`Attribute`的`code`和`value`是用键值对`Map`存储的

```
 */
enum Attribute {
    HttpStatusCodeAttribute = 0,    ///< 从HTTP服务器接收的HTTP状态码
    HttpReasonPhraseAttribute = 1   ///< 从HTTP服务器接收的HTTP原因短语
};

struct LHttpDataStruct {
    LString m_host = LString();
    LString m_path = LString();
    int m_port = -1;
    LMap<LString, LString> m_rawHeaders;
    LMap<KnownHeaders, LString> m_headers;
    LMap<Attribute, LString> m_attributes;
    int m_redirectCount = 0;
    int m_maxRedirects = 50;
};
```

而在`setAttribute`当中，原来没有考虑多次设置的情况，只是盲目的插入，因此可能`LMap`检测到对已经有的`key`进行插入就舍弃了这组数据，因此设置失败

- 解决：分情况讨论，当没有`key`的时候插入，有的时候就覆盖这个值

```cpp
void LHttpRequest::setAttribute(Attribute code, const LString& value) {
    if (!value.isEmpty()) {
        if (!m_pData->m_attributes.contains(code))  // key不存在则插入
            m_pData->m_attributes.insert(code, value);
        else
            m_pData->m_attributes[code] = value;  // 存在则更新
    }
}
```

- 测试：`HttpRequest`中的`test5()`，运行结果：

```
lzx0626@DavidingPlus:~/DavidingPlus/Lark5/larksdk/build$ snippet/HttpRequestTest/HttpRequestTest
中文中文中文中文
111
EnglishEnglish
~~~
lzx0626@DavidingPlus:~/DavidingPlus/Lark5/larksdk/build$
```

5.18861：确认 LHttpControl不设置任何标头值直接发送请求后的请求报文数据的完整性

- 解决：根据禅道上`Qt`返回的结果对相应请求标头设置了默认值

```cpp
    } else {
        // 根据Qt返回的结果设置了一些默认值
        sendMessage.append("User-Agent: Mozilla/5.0\r\n");
    }

    if (!request.rawHeader("Accept").isEmpty()) {
        sendMessage.append("Accept: ");
        sendMessage.append(request.rawHeader("Accept"));
        sendMessage.append("\r\n");
    }
    if (!request.rawHeader("Accept-Language").isEmpty()) {
        sendMessage.append("Accept-Language: ");
        sendMessage.append(request.rawHeader("Accept-Language"));
        sendMessage.append("\r\n");
    } else {
        // 根据Qt返回的结果设置了一些默认值
        sendMessage.append("Accept-Language: en-US,*\r\n");
    }

    if (!request.rawHeader("Accept-Encoding").isEmpty()) {
        sendMessage.append("Accept-Encoding: ");
        sendMessage.append(request.rawHeader("Accept-Encoding"));
        sendMessage.append("\r\n");
    } else {
        // 根据Qt返回的结果设置了一些默认值
        sendMessage.append("Accept-Encoding: gzip, deflate\r\n");
    }
```

- 测试：在`HttpRequest`的`test2()`中，结果如下，根据禅道上设置了一些请求标头的默认值：

```
lzx0626@DavidingPlus:~/DavidingPlus/Lark5/larksdk/build$ snippet/HttpRequestTest/HttpRequestTest
url1: http://192.168.1.211/get
url2: http://192.168.1.211/get
buffersize: 65536
operation: GET
{
  "args": {},
  "headers": {
    "Accept-Encoding": "gzip, deflate",
    "Accept-Language": "en-US,*",
    "Connection": "keep-alive",
    "Host": "192.168.1.211",
    "User-Agent": "Mozilla/5.0"
  },
  "origin": "192.168.7.1",
  "url": "http://192.168.1.211/get"
}



GET /get HTTP/1.1
Host: 192.168.1.211
User-Agent: Mozilla/5.0
Accept-Language: en-US,*
Accept-Encoding: gzip, deflate
Connection: keep-alive
```