Bug修改记录 11.14

1.19234： LHttpRequest头文件中headerToRawHeader函数，注释不完整

- 已修改

```
    // 静态私有成员函数
private:
    /**
     * @brief 对枚举的已知标头做一个到LString的映射
     *
     * @param header 枚举类型的已知标头
     * @return LString 返回字符串类型的原始标头
     */
    static LString headerToRawHeader(KnownHeaders header);
```

2.19323：LHttpReply头文件中，sedata函数注释错误

- 已修改

```
    /**
     * @brief 设置数据。
     * @param data 数据
     */
    void setData(const LString &data);
```

3.19322：请检查所有代码，禁止使用printf等输出错误

已修改

- `labsstractsocket.cpp`

```
LAbstractSocket::LAbstractSocket(SocketType socketType, NetworkProtocol networkProtocol) {
    pData.type = socketType;
    pData.protocol = networkProtocol;
    socketfd = -1;

    if (false == createSocket())
        throw LException("socket creat error : please creat again");
}
```

```cpp
bool LAbstractSocket::createSocket() {
    // 设置端口复用
    int optval = 1;
    setsockopt(socketfd, SOL_SOCKET, SO_REUSEPORT, &optval, sizeof(optval));

    // IPv4
    if (pData.protocol == IPv4Protocol) {
        if (pData.type == TcpSocket)
            socketfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
        else if (pData.type == UdpSocket)
            socketfd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    }
    // IPv6
    else if (pData.protocol == IPv6Protocol) {
        if (pData.type == TcpSocket)
            socketfd = socket(AF_INET6, SOCK_STREAM, IPPROTO_TCP);
        else if (pData.type == UdpSocket)
            socketfd = socket(AF_INET6, SOCK_DGRAM, IPPROTO_UDP);
    } else {
        std::cerr << "UnknownNetworkLayerProtocol : please make a socket again" << std::endl;
        return false;
    }

    if (-1 == socketfd) {
        perror("socket");
        return false;
```

```cpp
        ret = bind(socketfd, (struct sockaddr *)&bindadd, sizeof(bindadd));
    } else {
        std::cerr << "Binds error : please set NetworkProtocol" << std::endl;
        return false;
    }

    if (-1 == ret) {
        std::cerr << "Binds error : creat failed" << std::endl;
        return false;
    }
    return true;
```

- `lhostaddress.cpp`

```cpp
LHostAddress::LHostAddress(const LString& address) : LHostAddress() {
    if (false == setAddress(address))
        throw LException("warning : wrong IPV4/IPV6 (string),LHostAddress init failed");
}
```

```cpp
bool LHostAddress::setAddress(const LString& address) {
    std::string std_address = address.toStdString();

    // 并未进行一些对传参的正确性的处理
    if (std_address.find('.') != std::string::npos) {
        // 地址修改正确之后再进行后续的修改
        if (1 != inet_pton(AF_INET, std_address.c_str(), &pAddr.ipv4)) {
            std::cerr << "Failed to convert IPv4 address from string to bytes" << std::endl;
            return false;
        }

        pAddr.type = IPv4addr;

        return true;
    } else if (std_address.find(':') != std::string::npos) {
        if (1 != inet_pton(AF_INET6, std_address.c_str(), &pAddr.ipv6)) {
            std::cerr << "Failed to convert IPv6 address from string to bytes" << std::endl;
            return false;
        }

        pAddr.type = IPv6addr;

        return true;
    } else {
        std::cerr << "warning : wrong IPV4/IPV6 (string)" << std::endl;
        return false;
    }
}
```

- lhttpcontrol.cpp

```cpp
void LHttpControl::dealSigAlrm(int num) {
    std::cerr << "connect连接超时!请检查您输入的地址!" << std::endl;
    exit(-1);
}
```

- lhttprequest.cpp

```cpp
LString LHttpRequest::url() const {
    if (!m_pData.m_host.isEmpty()) {
        // url需要加上协议类型，目前只支持http类型
        LString url("http://");
        url.append(m_pData.m_host);

        // 不是默认端口80需要指明端口
        if (80 != m_pData.m_port)
            url.append(":" + std::to_string(m_pData.m_port));

        url.append(m_pData.m_path);
        return url;
    } else {
        std::cerr << "请检查是否设置host" << std::endl;
        return LString();
    }
}
```

- ltcpsocket.cpp

```cpp
bool LTcpSocket::listens(int backlog = 5) {
    int ret = listen(localfd(), backlog);
    if (ret < 0 || backlog < 0) {
        std::cerr << "listens error" << std::endl;
        return false;
    }
    return true;
}
```

```cpp
bool LTcpSocket::accepts() {
    struct sockaddr_in acceptadd;
    socklen_t len = sizeof(struct sockaddr_in);
    int newfd = accept(localfd(), (struct sockaddr *)&acceptadd, &len);
    if (newfd < 0) {
        std::cerr << "accepts error" << std::endl;
        return false;
    }

    setlocalfd(newfd);
    setPeerAddress(LHostAddress(acceptadd.sin_addr));
    setPeerPort(ntohs(acceptadd.sin_port));

    isConnected = true;

    return true;
}
```

```cpp
bool LTcpSocket::disconnect() {
    if (!isConnected) {
        std::cerr << "socket未连接!" << std::endl;
        return false;
    }

    if (closes())
        return true;
    else
        return false;
}
```

- ludpsocket.cpp

```
    if (ret < 0) {
        std::cerr << "sendto error" << std::endl;
        return false;
    }
    return true;
} else if (pData.protocol == IPv6Protocol) {
    struct sockaddr_in6 sendadd;
    sendadd.sin6_family = AF_INET6;
    sendadd.sin6_port = htons(peerPort());
    sendadd.sin6_addr = peerAddress().getIPv6();
    int ret = sendto(localfd(), data, length, 0, (str
    (struct sockaddr_in6));
    if (ret < 0) {
        std::cerr << "sendto error" << std::endl;
        return false;
    }
    return true;

} else {
    std::cerr << "sends error" << std::endl;
    return false;
}
```