14 Reactor 反应堆模式

参考代码：https://gitee.com/whb-helloworld/linux-plus-meal/tree/master/reactor

主要以代码为主，下面张贴部分核心结构

附录：

Connection.hpp

```C++
#pragma once

#include <iostream>
#include <string>
#include <functional>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

class Connection;
class TcpServer;

using func_t = std::function<void(Connection *)>;

class Connection
{
public:
    Connection(int sockfd, uint32_t events, TcpServer *R)
        : _sockfd(sockfd), _events(events), _R(R)
    {
    }
    void RegisterCallback(func_t recver, func_t sender, func_t
excepter)
    {
        _recver = recver;
        _sender = sender;
        _excepter = excepter;
    }
    void AddInBuffer(std::string buffer)
    {
```

```
        _inbuffer += buffer; // 追加到 inbuffer 中
    }
    void AddOutBuffer(const std::string &buffer)
    {
        _outbuffer += buffer;
    }
    bool OutBufferEmpty()
    {
        return _outbuffer.empty();
    }
    int SockFd()
    {
        return _sockfd;
    }
    uint32_t Events()
    {
        return _events;
    }
    void SetEvents(uint32_t events)
    {
        _events = events;
    }
    void SetClient(const struct sockaddr_in c)
    {
        _client = c;
    }
    std::string &InBuffer()
    {
        return _inbuffer;
    }
    std::string &OutBuffer()
    {
        return _outbuffer;
    }
    void Close()
    {
        ::close(_sockfd);
    }
    ~Connection()
    {
    }

private:
    // 对应的 sockfd
```

```
    int _sockfd;
    // 对应的缓冲区
    std::string _inbuffer;  // _sockfd 接受缓冲区，暂时用 string 代替
    std::string _outbuffer; // _sockfd 发送缓冲区

    // 关心的事件
    uint32_t _events;
    // 维护一下 client 的 ip 和 port 信息
    struct sockaddr_in _client;

public:
    // 对特定 connection 进行处理的回调函数
    func_t _recver;
    func_t _sender;
    func_t _excepter;
    // TcpServer 的回指指针 - TODO
    TcpServer *_R;
};

class ConnectionFactory
{
public:
    static Connection *BuildListenConnection(int listensock,
func_t recver, uint32_t events, TcpServer *R)
    {
        Connection *conn = new Connection(listensock, events, R);
        conn->RegisterCallback(recver, nullptr, nullptr);
        return conn;
    }
    static Connection *BuildNormalConnection(int sockfd,
                                             func_t recver,
                                             func_t sender,
                                             func_t excepter,
                                             uint32_t events,
                                             TcpServer *R)
    {
        Connection *conn = new Connection(sockfd, events, R);
        conn->RegisterCallback(recver, sender, excepter);
        return conn;
    }
};
```

Accepter.hpp

```C++
#pragma once

#include <iostream>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <cerrno>
#include "Common.hpp"
#include "Log.hpp"
#include "Connection.hpp"
#include "HandlerConnection.hpp"

class Accepter // 连接管理器
{
public:
    Accepter()
    {
    }
    // ET 模式，你怎么知道，只有一条连接到来了呢？
    void AccepterConnection(/*this, */ Connection *conn) // conn: listensocket 对应的 conn
    {
        errno = 0;
        while (true)
        {
            struct sockaddr_in peer;
            socklen_t len = sizeof(peer);
            int sockfd = ::accept(conn->SockFd(), (struct sockaddr *)&peer, &len);
            if (sockfd > 0)
            {
                lg.LogMessage(Info, "get a new link, sockfd is: %d\n", sockfd);
                SetNonBlock(sockfd);
                auto recver = std::bind(&HandlerConnection::Recver, std::placeholders::_1);
                auto sender = std::bind(&HandlerConnection::Sender, std::placeholders::_1);
                auto excepter = std::bind(&HandlerConnection::Excepter, std::placeholders::_1);
                Connection *normal_conn = ConnectionFactory::BuildNormalConnection(sockfd,
```

```
                    recver, sender, excepter, EPOLLIN|EPOLLET,
conn->_R);
                conn->_R->AddConnection(normal_conn);
            }
            else
            {
                if (errno == EAGAIN)
                    break;
                else if (errno == EINTR)
                    continue;
                else
                {
                    lg.LogMessage(Warning, "get a new link
error\n");
                    break;
                }
            }
        }
    }
    ~Accepter()
    {
    }
};
```

HandlerConnction.hpp

```
C++
#pragma once

#include <iostream>
#include <cerrno>
#include "Connection.hpp"
#include "Protocol.hpp"
#include "Calculate.hpp"
#include "Log.hpp"

using namespace Protocol;
using namespace CalCulateNS;

const static int buffer_size = 1024;

class HandlerConnection
{
public:
```

```
    static void HandlerRequest(Connection *conn)
    {
        std::string &inbuffer = conn->InBuffer();
        std::string message; // 表示一个符合协议的一个完整的报文
        Calculate calulate;  // 负责业务处理
        Factory factory;
        auto req = factory.BuildRequest();
        // 1. 明确报文边界，解决粘报问题
        while (Decode(inbuffer, &message))
        {
            // message 一定是一个完整的报文，符合协议的!
            // 2. 反序列化
            if (!req->Deserialize(message))
                continue;
            // 3. 业务处理
            auto resp = calulate.Cal(req);
            // 4. 对相应进行序列化
            std::string responseStr;
            resp->Serialize(&responseStr);
            // 5. 封装完整报文
            responseStr = Encode(responseStr);
            // 6. 将应答全部追加到 outbuffer 中
            conn->AddOutBuffer(responseStr);
        }
        // 考虑发送的问题了
        if (!conn->OutBufferEmpty())
        {
            conn->_sender(conn); // 对写事件，直接发！！！--- 不代表能全部发完!
        }
    }
    // 在这里读取的时候，我们关系数据是什么格式？协议是什么样子的吗？
    // 不关心！！！我们只负责把本轮属于完全读取完毕 --- 把读到的字节流数据，交给上层 --- 由上层进行分析处理
    static void Recver(Connection *conn)
    {
        errno = 0;
        // 读取流程
        char buffer[1024];
        while (true)
        {
            ssize_t n = recv(conn->SockFd(), buffer, sizeof(buffer) - 1, 0); // 非阻塞读取
```

```
            if (n > 0)
            {
                buffer[n] = 0;
                conn->AddInBuffer(buffer);
            }
            else
            {
                // std::cout << "................... errno:" <<
errno << std::endl;
                if (errno == EAGAIN)
                    break;
                else if (errno == EINTR)
                    continue;
                else
                {
                    // 真正的读取错误
                    conn->_excepter(conn); // 直接回调自己的异常处理
就可以了!
                    return;
                }
            }
        }
        std::cout << "sockfd# " << conn->SockFd() << ":\n"
                  << conn->InBuffer() << std::endl;
        // 尝试分析处理报文 -- 半个，一个半， 10 个，11 个半
        HandlerRequest(conn);
    }
    static void Sender(Connection *conn)
    {
        errno = 0;
        std::string &outbuffer = conn->OutBuffer();
        while (true)
        {
            ssize_t n = send(conn->SockFd(), outbuffer.c_str(),
outbuffer.size(), 0);
            if (n >= 0)
            {
                outbuffer.erase(0, n); // 已经发给 OS 的，就直接移除
了 // conn->remove(n);
                if (outbuffer.empty())
                    break;
            }
            else
            {
```

```
                if (errno == EAGAIN)
                    break; // 只有这里，才会正常退出
                else if (errno == EINTR)
                    continue;
                else
                {
                    conn->_excepter(conn);
                    return;
                }
            }
        }

        // 走到这里，意味着什么？我们本轮发满了，但是数据可能没发完，为什么没发完呢？
        // 开启对 conn->SockFd() EPOLLOUT 的关心！！！！！， 如何开启对于特定一个 connection 对应的写事件关心呢？？？
        if (!conn->OutBufferEmpty())
        {
            conn->_R->EnableReadWrite(conn->SockFd(), true, true);
        }
        else
        {
            conn->_R->EnableReadWrite(conn->SockFd(), true, false);
        }
    }
    static void Excepter(Connection *conn)
    {
        lg.LogMessage(Info, "connection erase done, who: %d\n", conn->SockFd());
        errno = 0;
        // 从 epoll 中移除对 conn->Sockfd 的关心
        // unordered_map 移除 conn
        conn->_R->RemoveConnection(conn->SockFd());
        // 关闭 conn->Sockfd
        conn->Close();
        // delete conn
        delete conn;
    }
};
```