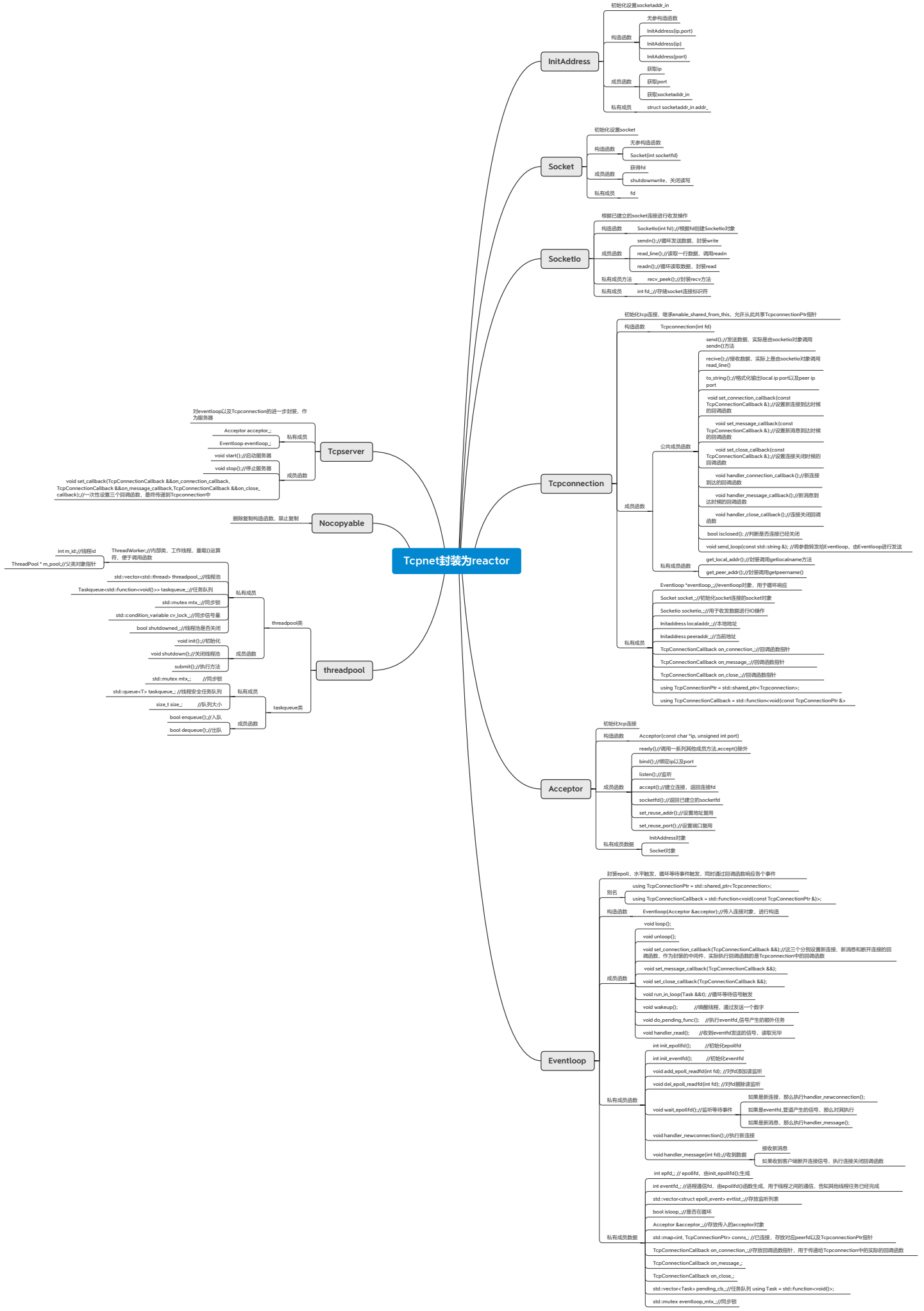
Tcpnet封装为reactor
InitAddress
初始化设置socketaddr_in
构造函数
无参构造函数
InitAddress(ip,port)
InitAddress(ip)
InitAddress(port)
成员函数
获取ip
获取port
获取socketaddr_in
私有成员
struct socketaddr_in addr_
Socket
初始化设置socket
构造函数
无参构造函数
Socket(int socketfd)
成员函数
获得fd
shutdownwrite，关闭读写
私有成员
fd
SocketIo
根据已建立的socket连接进行收发操作
构造函数
SocketIo(int fd);//根据fd创建SocketIo对象
成员函数
sendn();//循环发送数据，封装write
read_line();//读取一行数据，调用readn
readn();//循环读取数据，封装read
私有成员方法
recv_peek();//封装recv方法
私有成员
int fd_;//存储socket连接标识符
Tcpconnection
初始化tcp连接，继承enable_shared_from_this，允许从此共享TcpconnectionPtr指针
构造函数
Tcpconnection(int fd)
成员函数
公共成员函数
send();//发送数据，实际是由socketio对象调用sendn()方法
recive();//接收数据，实际上是由socketio对象调用read_line()
to_string();//格式化输出local ip port以及peer ip port
void set_connection_callback(const TcpConnectionCallback &);//设置新连接到达时候的回调函数
void set_message_callback(const TcpConnectionCallback &);//设置新消息到达时候的回调函数
void set_close_callback(const TcpConnectionCallback &);//设置连接关闭时候的回调函数
void handler_connection_callback();//新连接到达的回调函数
void handler_message_callback();//新消息到达时候的回调函数
void handler_close_callback();//连接关闭回调函数
bool isclosed(); //判断是否连接已经关闭
void send_loop(const std::string &); //待参数转发给Eventloop，由Eventloop进行发送
私有成员函数
get_local_addr();//封装调用getlocalname方法
get_peer_addr();//封装调用getpeername()
私有成员
Eventloop *eventloop_;//eventloop对象，用于循环响应
Socket socket_;//初始化socket连接的socket对象
Socketio socketio_;//用于收发数据进行IO操作
Initaddress localaddr_;//本地地址
Initaddress peeraddr_;//当前地址
TcpConnectionCallback on_connection_;//回调函数指针
TcpConnectionCallback on_message_;//回调函数指针
TcpConnectionCallback on_close_;//回调函数指针
using TcpConnectionPtr = std::shared_ptr<Tcpconnection>;
using TcpConnectionCallback = std::function<void(const TcpConnectionPtr &>
Acceptor
初始化tcp连接
构造函数
Acceptor(const char *ip, unsigned int port)
成员函数
ready()//调用一系列其他成员方法,accept()除外
bind();//绑定ip以及port
listen();//监听
accept();//建立连接，返回连接fd
socketfd();//返回已建立的socketfd
set_reuse_addr();//设置地址复用
set_reuse_port();//设置端口复用
私有成员数据
InitAddress对象
Socket对象
Eventloop
封装epoll，水平触发，循环等待事件触发，同时通过回调函数响应各个事件
别名
using TcpConnectionPtr = std::shared_ptr<Tcpconnection>;
using TcpConnectionCallback = std::function<void(const TcpConnectionPtr &)>;
构造函数
Eventloop(Acceptor &acceptor);//传入连接对象，进行构造
成员函数
void loop();
void unloop();
void set_connection_callback(TcpConnectionCallback &&);//这三个分别设置新连接、新消息和断开连接的回调函数，作为封装的中间件，实际执行回调函数的是Tcpconnection中的回调函数
void set_message_callback(TcpConnectionCallback &&);
void set_close_callback(TcpConnectionCallback &&);
void run_in_loop(Task &&); //循环等待信号触发
void wakeup(); //唤醒线程，通过发送一个数字
void do_pending_func(); //执行eventfd_信号产生的额外任务
void handler_read(); //收到eventfd发送的信号，读取完毕
私有成员函数数
int init_epollfd(); //初始化epollfd
int init_eventfd(); //初始化eventfd
void add_epoll_readfd(int fd); //对fd添加读监听
void del_epoll_readfd(int fd); //对fd删除读监听
void wait_epollfd();//监听等待事件
如果是新连接，那么执行handler_newconnection();
如果是eventfd_管道产生的信号，那么对其执行
如果是新消息，那么执行handler_message();
void handler_newconnection();//执行新连接
void handler_message(int fd);//收到数据
接收新消息
如果收到客户端断开连接信号，执行连接关闭回调函数
私有成员数据
int epfd_; // epollfd，由init_epollfd();生成
int eventfd_; //进程通信fd，由epollfd()函数生成，用于线程之间的通信，告知其他线程任务已经完成
std::vector<struct epoll_event> evtlist_;//存放监听列表
bool isloop_;//是否在循环
Acceptor &acceptor_;//存放传入的acceptor对象
std::map<int, TcpConnectionPtr> conns_; //已连接，存放对应peerfd以及TcpconnectionPtr指针
TcpConnectionCallback on_connection_;//存放回调函数指针，用于传递给Tcpconnection中的实际的回调函数
TcpConnectionCallback on_message_;
TcpConnectionCallback on_close_;
std::vector<Task> pending_cb_;//任务队列 using Task = std::function<void()>;
std::mutex eventloop_mtx_;//同步锁
Tcpserver
对eventloop以及Tcpconnection的进一步封装，作为服务器
私有成员
Acceptor acceptor_;
Eventloop eventloop_;
成员函数
void start();//启动服务器
void stop();//停止服务器
void set_callback(TcpConnectionCallback &&on_connection_callback, TcpConnectionCallback &&on_message_callback,TcpConnectionCallback &&on_close_callback);//一次性设置三个回调函数，最终传递到Tcpconnection中
Nocopyable
删除复制构造函数，禁止复制
threadpool
threadpool类
私有成员
int m_id;//线程id
ThreadWorker;//内部类，工作线程，重载()运算符，便于调用函数
ThreadPool * m_pool;//父类对象指针
std::vector<std::thread> threadpool_;//线程池
Taskqueue<std::function<void()>> taskqueue_;//任务队列
std::mutex mtx_;//同步锁
std::condition_variable cv_lock_;//同步信号量
bool shutdowned_;//线程池是否关闭
成员函数
void init();//初始化
void shutdown();//关闭线程池
submit();//执行方法
taskqueue类
私有成员
std::mutex mtx_; //同步锁
std::queue<T> taskqueue_; //线程安全任务队列
size_t size_; //队列大小
成员函数
bool enqueue();//入队
bool dequeue();//出队