类设计思想

- `Thread`

  线程功能的封装

  - `pthread_t  pthreadId_` 和线程ID绑定
  - `Thread::start()` 创建线程
  - 对象成员 `ThreadFunc func_` 用于指定线程运行的函数
- `Socket`

  socket功能的封装

  - `const int sockfd_` 和套接字描述符绑定
  - `void listen()` 开始套接字监听
  - `void bindAddress(const InetAddress& localaddr)` 和监听地址和端口绑定
  - `int accept(InetAddress* peeraddr)` 接受连接，返回新的套接字
- `EPollPoller`

  对I/O复用的抽象

  - `EventLoop* ownerLoop_` **Poller所属的EventLoop**
  - `EventList events_` 用于返回活跃事件
  - `ChannelMap channels_`
  - `poll(int timeoutMs, ChannelList* activeChannels)` 在EventLoop的loop函数中调用
  - `updateChannel(Channel* channel)` 更新Poller中的channels（EventLoop中调用）
  - `removeChannel(Channel* channel)` 移除Poller中的某个channels（EventLoop中调用）
- `Channel`

  对I/O事件注册和封装的响应，套接字绑定，负责管理套接字关注的事件，负责将socket关注的事件，注册到Poller中或者从Poller中删除，这些操作都是通过EventLoop中的函数实现的

  - `EventLoop* loop_` Channel所属EventLoop
  - `const int  fd_` 和套接字文件描述符绑定
  - `int events_ 和 int revents_` 关注的时间和poll/epoll返回的事件
  - `int index_` 该channel在Poller中ChannnelMap的序号
  - 各种回调函数
    - `ReadEventCallback readCallback_` **构造Acceptor时设置，Acceptor::handleRead()**
    - `EventCallback writeCallback_` `EventCallback closeCallback_;` `EventCallback errorCallback_`
- `EventLoop`

对事件循环的抽象

- `const pid_t threadId_` 当前EventLoop对象所属线程ID
- `boost::scoped_ptr<Poller> poller_`
- `int wakeupFd_ 和 boost::scoped_ptr<Channel> wakeupChannel_`

  用于唤醒当前进程，如调用queueInLoop的线程不是当前I/O线程，需要唤醒当前I/O线程，以便及时执行任务；或者调用queueInLoop的线程不是当前I/O线程，并且此时正在调用pending functor；只有当前IO线程的事件回调中调用queueInLoop才不需要唤醒，因为在执行完handleEvent之后，会执行doPendingFunctors
- `loop()` 调用poller的poll函数，返回活跃channel，调用其事件处理回调函数
- `void updateChannel(Channel* channel` 和 `void removeChannel(Channel* channel)` 在poller中添加或移除channel

- 关系
  - `EventLoop和Poller`
    - 组合关系，一个eventloop包含一个poller
    - 并且**poller的生命期由eventloop来控制**
    - eventloop的loop函数实际上调用poller的poll函数实现
  - `EventLoop和Channel`
    - 聚合关系，一个EventLoop包含多个Channel，通过Poller实现这种聚合关系
    - 调用Channel的update函数，实际上调用了eventloop函数的updateChannel，进而调用Poller的updateChannel，将channel注册到Poller中
    - EventLoop不负责Channel的生命周期
  - `Channel和socketfd`
    - Channel不拥有文件描述符，是关联关系，一个fd绑定，channel对象销毁时不关闭文件描述符
    - 文件描述符为套接字所拥有，其生存期由套接字控制，即套接字销毁时关闭文件描述符
    - channel是TcpConnection、Acceptor、Connector的成员，关系为组合，**channel的生存期由TcpConnection、Acceptor、Connector来控制**
- `Acceptor`

对主动套接字的抽象

- acceptor关注监听套接字的可读对象，由channel来注册，可读对象发生后，channnel调用事件处理回调函数
- `EventLoop* loop_` 所属EventLoop
- `Socket acceptSocket_` 监听的套接字
- `Channel acceptChannel_` socket绑定的channel
- `NewConnectionCallback newConnectionCallback_` 连接建立回调函数，**构造TcpServer时设置，TcpServer::newConnection**
- `listen()`：调用socket的listen()，启动套接字监听功能，使能对应channel的可读事件监听

    - `handleRead()`：回调连接建立回调函数
- `TcpServer`
    - `EventLoop* loop_`
    - `boost::scoped_ptr<Acceptor> acceptor_`
    - `boost::scoped_ptr<EventLoopThreadPool> threadPool_`
    - `ConnectionMap connections_` 连接列表
    - 回调函数：`ConnectionCallback connectionCallback_` `MessageCallback messageCallback_` `WriteCompleteCallback writeCompleteCallback_` `ThreadInitCallback threadInitCallback_` **这些回调函数由EchoServer给出**
    - `start()`：启动线程池，构造指定数目的线程并初始化，启动acceptor的监听函数，Acceptor::listen，此时只是使能了监听，执行eventloop.loop()时才真正开始监听
    - `newConnection()`：获取一个线程，并新建一个链接TcpConnection，设置各种回调函数，将其连接建立回调函数加入到线程的任务列表中
- `EventLoopThreadPool`
    - `EventLoop* baseLoop_;`
    - `boost::ptr_vector<EventLoopThread> threads_` 和 `std::vector<EventLoop*> loops_;` 线程池中线程及其对应的eventloop
    - `start()`：创建相应数目的线程，并且初始化线程
    - `getNextLoop()`：轮叫的方式
- `TcpConnection`

  TcpConnection由Acceptor创建，有与其绑定的eventloop、socket、channel，和acceptor相同
    - `EventLoop* loop_`
    - `boost::scoped_ptr<Socket> socket_`
    - `boost::scoped_ptr<Channel> channel_`
    - `connectEstablished()` 关注可读事件
    - 回调函数

      连接到来时，调用Acceptor的连接到来回调函数，进而调用TcpServer::newConnection，该函数中接收一个连接，并且创建一个TcpConnection，将EchoServer传递的回调函数设置到TcpConnection中（可能没有高水位标回调函数）
        - `ConnectionCallback connectionCallback_`
        - `MessageCallback messageCallback_`
        - `WriteCompleteCallback writeCompleteCallback_`
        - `CloseCallback closeCallback_`
        - `HighWaterMarkCallback highWaterMarkCallback_`
    - 输入缓冲区和输出缓冲区：`Buffer inputBuffer_` 和 `Buffer outputBuffer_`
- `Buffer`
    - `std::vector<char> buffer_`

- `size_t readerIndex_`
- `size_t writerIndex_`

- 关系
  - `TcpConnection和Buffer`：每个TcpConnection包含两个Buffer，一个输入缓冲区和一个输出缓冲区
  - `TcpServer和Acceptor`：TcpServer包含一个Acceptor，并且设置其连接建立回调函数
  - `TcpServer和EventLoopThreadPool`：每个TcpServer包含一个线程池
  - `TcpServer和TcpConnection`：每个TcpServer包含一个TcpConnection列表，在Acceptor回调TcpServer::newConnection()创建连接并且设置其各种回调函数

以上内容整理于 幕布文档