**Muduo 的目录结构如下**

muduo

|-- base # 与网络无关的基础代码，已[提前发布](http://blog.csdn.net/Solstice/archive/2010/08/21/5829421.aspx" \t "_blank)

`-- net # 网络库

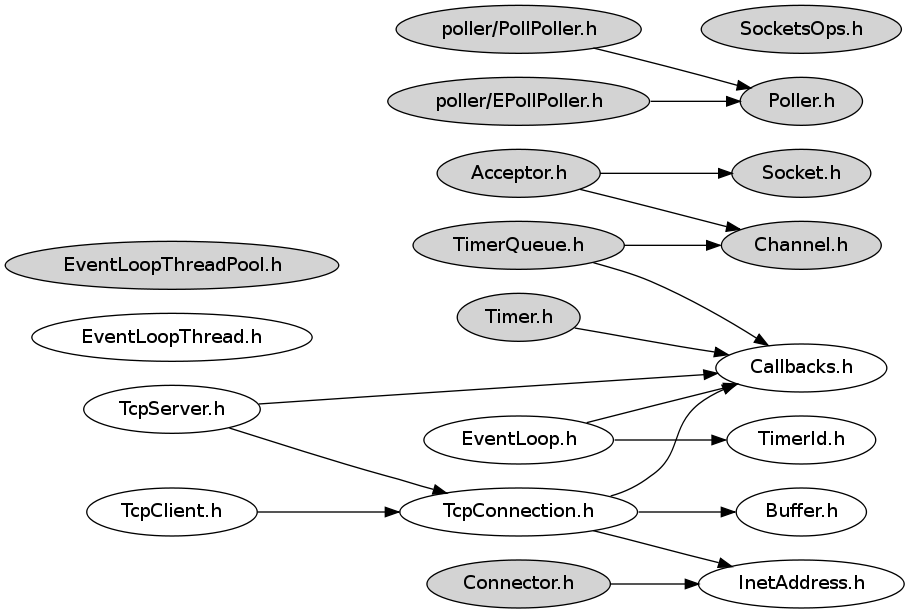
    |-- http # 一个简单的可嵌入的 web 服务器

    |-- inspect # 基于以上 web 服务器的“窥探器”，用于报告进程的状态

    `-- poller # poll(2) 和 epoll(4) 两种 IO multiplexing 后端

Muduo 是基于 Reactor 模式的网络库，其核心是个事件循环 EventLoop，用于响应计时器和 IO 事件。Muduo 采用基于对象（object based）而非面向对象（object oriented）的设计风格，其[接口多以 boost::function + boost::bind 表达](http://blog.csdn.net/Solstice/archive/2008/10/13/3066268.aspx" \t "_blank)。

Muduo 的头文件明确分为客户可见和客户不可见两类。客户可见的为白底，客户不可见的为灰底。

[](http://hi.csdn.net/attachment/201008/29/0_128309628681gp.gif)

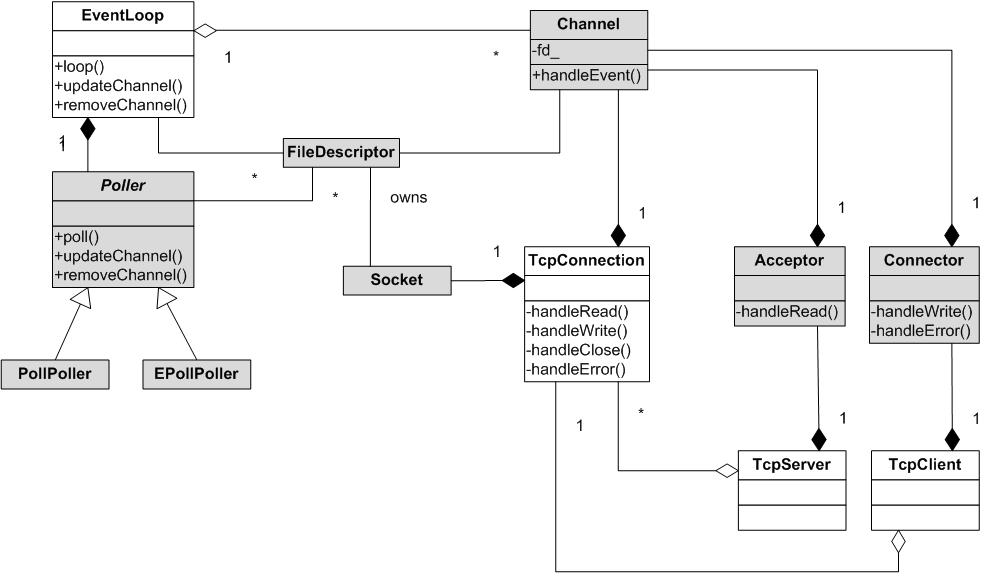
这里简单介绍各个头文件及 class 的作用，详细的介绍留给以后的博客。

**公开接口**

* Buffer 仿 Netty ChannelBuffer 的 buffer class，数据的读写透过 buffer 进行
* InetAddress 封装 IPv4 地址 (end point)，注意，muduo 目前不能解析域名，只认 IP
* EventLoop 反应器 Reactor，用户可以注册计时器回调
* EventLoopThread 启动一个线程，在其中运行 EventLoop::loop()
* TcpConnection 整个网络库的核心，封装一次 TCP 连接
* TcpClient 用于编写网络客户端，能发起连接，并且有重试功能
* TcpServer 用于编写网络服务器，接受客户的连接
* 在这些类中，TcpConnection 的生命期依靠 shared\_ptr 控制（即用户和库共同控制）。Buffer 的生命期由 TcpConnection 控制。其余类的生命期由用户控制。
* HttpServer 和 Inspector，暴露出一个 http 界面，用于监控进程的状态，类似于 Java JMX。这么做的原因是，《程序员修炼之道》第 6 章第 34 条提到“对于更大、更复杂的服务器代码，提供其操作的内部试图的一种漂亮技术是使用内建的 Web 服务器”，Jeff Dean 也说“（每个 Google 的服务器进程）Export HTML-based status pages for easy diagnosis”。

**内部实现**

* Channel 是 selectable IO channel，负责注册与响应 IO 事件，它不拥有 file descriptor。它是 Acceptor、Connector、EventLoop、TimerQueue、TcpConnection 的成员，生命期由后者控制。
* Socket 封装一个 file descriptor，并在析构时关闭 fd。它是 Acceptor、TcpConnection 的成员，生命期由后者控制。EventLoop、TimerQueue 也拥有 fd，但是不封装为 Socket。
* SocketsOps 封装各种 sockets 系统调用。
* EventLoop 封装事件循环，也是事件分派的中心。它用 eventfd(2) 来异步唤醒，这有别于传统的用一对 pipe(2) 的办法。它用 TimerQueue 作为计时器管理，用 Poller 作为 IO Multiplexing。
* Poller 是 PollPoller 和 EPollPoller 的基类，采用“电平触发”的语意。它是 EventLoop 的成员，生命期由后者控制。
* PollPoller 和 EPollPoller 封装 poll(2) 和 epoll(4) 两种 IO Multiplexing 后端。Poll 的存在价值是便于调试，因为 poll(2) 调用是上下文无关的，用 strace 很容易知道库的行为是否正确。
* Connector 用于发起 TCP 连接，它是 TcpClient 的成员，生命期由后者控制。
* Acceptor 用于接受 TCP 连接，它是 TcpServer 的成员，生命期由后者控制。
* TimerQueue 用 timerfd 实现定时，这有别于传统的设置 poll/epoll\_wait 的等待时长的办法。为了简单起见，目前用链表来管理 Timer，如果有必要可改为优先队列，这样复杂度可从 O(n) 降为O(ln n) （某些操作甚至是 O(1)）。它是 EventLoop 的成员，生命期由后者控制。
* EventLoopThreadPool 用于创建 IO 线程池，也就是说把 TcpConnection 分派到一组运行 EventLoop 的线程上。它是 TcpServer 的成员，生命期由后者控制。

类图   


**线程模型**

Muduo 的线程模型符合我主张的 [one loop per thread + thread pool](http://blog.csdn.net/Solstice/archive/2010/02/12/5307710.aspx) 模型。每个线程最多有一个 EventLoop。每个 TcpConnection 必须归某个 EventLoop 管理，所有的 IO 会转移到这个线程，换句话说一个 file descriptor 只能由一个线程读写。TcpConnection 所在的线程由其所属的 EventLoop 决定，这样我们可以很方便地把不同的 TCP 连接放到不同的线程去，也可以把一些 TCP 连接放到一个线程里。TcpConnection 和 EventLoop 是线程安全的，可以跨线程调用。TcpServer 直接支持多线程，它有两种模式：

1. 单线程，accept 与 TcpConnection 用同一个线程做 IO。

2. 多线程，accept 与 EventLoop 在同一个线程，另外创建一个 EventLoopThreadPool，新到的连接会按 round-robin 方式分配到线程池中。

**结语**

Muduo 是我对常见网络编程任务的总结，用它我能很容易地编写多线程的 TCP 服务器和客户端。Muduo 是我业余时间的作品，代码估计还有很多 bug，功能也不完善（例如不支持 signal 处理），待日后慢慢改进吧。