Echo 网络库

本项目发布在 github 上：

https://github.com/guochy2012/EchoLib

概要说明

源码分析

使用方法

概要说明

1.Echo 借鉴了大量 muduo 的代码，并对其进行简化。

2.Echo 体现了现代 C++的两大风格，一是事件回调，我们使用 function/bind 实现回调机制，用户只需要向其注册回调函数即可，必要时可以封装成 class，二是采用智能指针进行资源管理，例如 TcpConnection 使用了 shared_ptr 管理其生命周期，其他类采用了 unique_ptr 借助其销毁功能

3.Echo 的不足之处：

a) 对错误的处理比较粗糙

b) 没有使用一个比较规范的日志

4.Echo 与 muduo 的主要差距

a) Echo 没有实现复用同一个 Poller 的功能，这正是 muduo 中的 EventLoop，里面封装了一个通用的 Poll、Epoll

模型

b) 我们采用的是阻塞 IO，不能成为严格意义上的 Reactor 模式，而且我们没有实现缓冲区。
c) 大量的实用技巧。

源码分析：

1.NonCopyable、Copyable 表示对象是否具有 value 语义（复制和赋值），Echo 中除了 InetAddress 之外，其余均禁用掉了 value 语义，这是为了避免潜在的 BUG。

2.Exception 相比标准库的 exception，增加了打印栈痕迹的功能

3.ThreadPool 系列，主要包含 MutexLock、Condition、Thread、ThreadPool。其中大量采用了 RAII 技术，避免资源的泄露，对于 Thread 和 ThreadPool，我们采用了 function 作为泛型技术，用户只需注册回调函数。

4.Timer，内部采用 timerfd 系列的定时器，不使用信号，而是使用 fd 可读作为定时器的触发事件，这使得 Timer 可以加入到 IO 复用模型，我们采用的是 Poll 模型。也可以单独把 Timer 放到一个线程，这就是 TimerThread 的产生。

5.TcpServer 系列：

a) Rio 封装了网络编程中的三大函数 readn、readLine、

writen，

b) Socket 则封装了基本的 socket 操作，但是不包含读写。

c) InetAddress 包装了 sockaddr_in 结构体，使之更易使用。

d) TcpConnection 则包装了 Tcp 连接的各种操作，主要是数据的收发以及获取 Tcp 连接的信息。TcpConnection 采用 shared_ptr 管理其生存期，还继承了 enable_shared_from_this，用于在类的内部获取自己的智能指针。

e) PollPoller 封装了 Poll 模型，内部存在一个 map<int, TcpConnectionPtr>实现从文件描述符到 Tcp 连接的映射。

f) TcpServer 则是对以上组件的组合。

6.用户注册事件与回调流程：

a) 先注册给 TcpServer，然后是 PollPoller，之后是 TcpConnection，这样完成了事件的注册

b) 回调函数由 PollPoller 触发，通过 map 寻找到 Tcp 连接，然后调用里面的回调函数。

7.TcpServer 实质是一个 IO 复用模型，ThreadPool 则是代笔多线程。用户在使用时，可以只选择其一。如果计算任务负担较重，可以将计算任务与 Tcp 回发封装成函数，交给线程池去计算。

8.此时，运行 TcpServer 的线程是一个 IO 线程，ThreadPool

里面的线程专注于 CPU 密集型计算。

使用方法见 example 文件夹

1.只使用 TcpServer，适用于计算任务很轻的情况

2.在代码中组合 TcpServer 和 ThreadPool，把计算任务（和 Tcp 回发）分发到线程池中，适合于计算任务较重的情况。

安装方法

执行 make，生成 echo 头文件夹和静态库 libecho.a

sudo make install 将 echo 安装到/usr/include/下，将 libecho.a 放置/usr/lib/下。

编译的时候需要加上-std=c++0x -lecho -lpthread