**EPOLLONESHOT事件(修复的bug)**

**（测试时，get试了试超长请求看看会不会影响服务器，结果get同一个存在的文件有时却返回错误数据）**

**向epoll中添加需要监听的文件描述符，**

**即使可以使用 ET 模式，一个socket 上的某个事件还是可能被触发多次。一个socket连接在任何时刻都只被一个线程处理，可以使用 epoll 的 EPOLLONESHOT 事件实现。注册了 EPOLLONESHOT 事件的 socket 一旦被某个线程处理完毕， 该线程就应该立即重置这个socket 上的 EPOLLONESHOT 事件，**

**locker.h 线程同步机制封装类 => 互斥锁 条件变量 信号量**

**http\_conn.h 封装用户连接**

**重载process()线程池中可以调用**

**threadpool.hpp 模板线程池实现**

**创建thread\_number个线程，并将他们设置为脱离线程**

**线程中运行的函数先wait()唤醒后lock()操作工作队列取出一个任务执行->process()，它不断从工作队列中取出任务并执行**

**操作工作队列时一定要加锁，因为它被所有线程共享**

**main.cpp**

**创建http\_conn数组，建立unordered\_map维护用户链接socked的connfd与http\_conn映射关系**

**创建线程池**

**创建socket设置端口复用后bind，listen，创建epoll对象，和接收epoll事件数组，epoll\_ctl向epoll中添加需要监听的文件描述符**

**while循环epoll\_wait：**

**1.如果事件发生在listenfd，则accept接收用户连接并初始化http\_conn加入http\_conn数组**

**2.如果发生EPOLLRDHUP | EPOLLHUP | EPOLLERR事件则关闭链接，init对应的http\_conn**

**3.** **如果发生EPOLLIN事件则read，read成功后将对应的http\_conn插入请求队列，睡眠在请求队列上的某个工作线程被唤醒，它获得请求对象并处理客户请求**

**4.** **如果发生EPOLLOUT事件则write**

**若退出while循环则记得释放资源**

**close(epollfd);**

**close(listenfd);**

**read:**

**recv为-1，errno == EAGAIN || errno == EWOULDBLOCK读完**

**recv为0 对方关闭连接**

**write:**

**writev分散写，写入buffer内容或用mmap内存映射的用户请求文件，写完要munmap**

**若errno == EAGAIN**

**如果TCP写缓冲没有空间，则等待下一轮EPOLLOUT事件，虽然在此期间，服务器无法立即接收到同一客户的下一个请求，但可以保证连接的完整性。**

**异步日志**

**用timing wheel 踢掉非活跃链接，使用boost::circle\_buffer与shared\_ptr实现；8 个桶，每秒内接收的链接放在同一个桶内，8 秒钟没有数据就超时断开连接，先清空指针所指桶内的所有链接，再把这秒内接收的链接放在里面**

**支持定时器处理定时任务，TimerQueue 使用 timerfd\_\*（在定时器超时的那一刻变得可读） 系列函数来处理定时，可以融入到 select/poll 框架中用统一的方式来处理 IO 事件和超时事件；**