### 项目介绍

基于C++实现的WEB服务器，实现了GET、POST请求。功能上主要实现了**用户注册、登录，文件上传、下载，远程目录浏览**。

项目主要包括**reactor、线程池、HTTP、数据库连接池、日志系统、时钟系统**等模块。

基于epoll IO复用技术，ET边缘触发+非阻塞+忙轮询网络模型实现高并发处理。

线程池通过任务队列，多线程异步处理网络请求。

HTTP模块对连接进行维护，利用主从状态机对报文进行解析，实现GET/POST请求处理。

RAII机制维护数据库连接池，避免重复连接/断开数据库。

基于单例模式实现的日志系统，异步写入日志信息。

基于双向链表实现定时器时钟系统，处理非活跃连接。

部署环境： Linux、Mysql

1. 配置数据库

**create database dbname;**

**USE dbname;**

**CREATE TABLE user(**

**username char(50) NULL,**

**password char(50) NULL**

**);**

**INSERT INTO user(username, password) VALUES('name', 'password');**

config.cpp 中，修改默认配置，3306是mysql默认端口，dbName为自建表名

**sqlPort\_ = 3306 ;**

**sqlUsr\_ = "root";**

**sqlPSWD\_ = "root" ;**

**dbName\_ = "dbname" ;**

****

1. 编译程序

bin 目录下

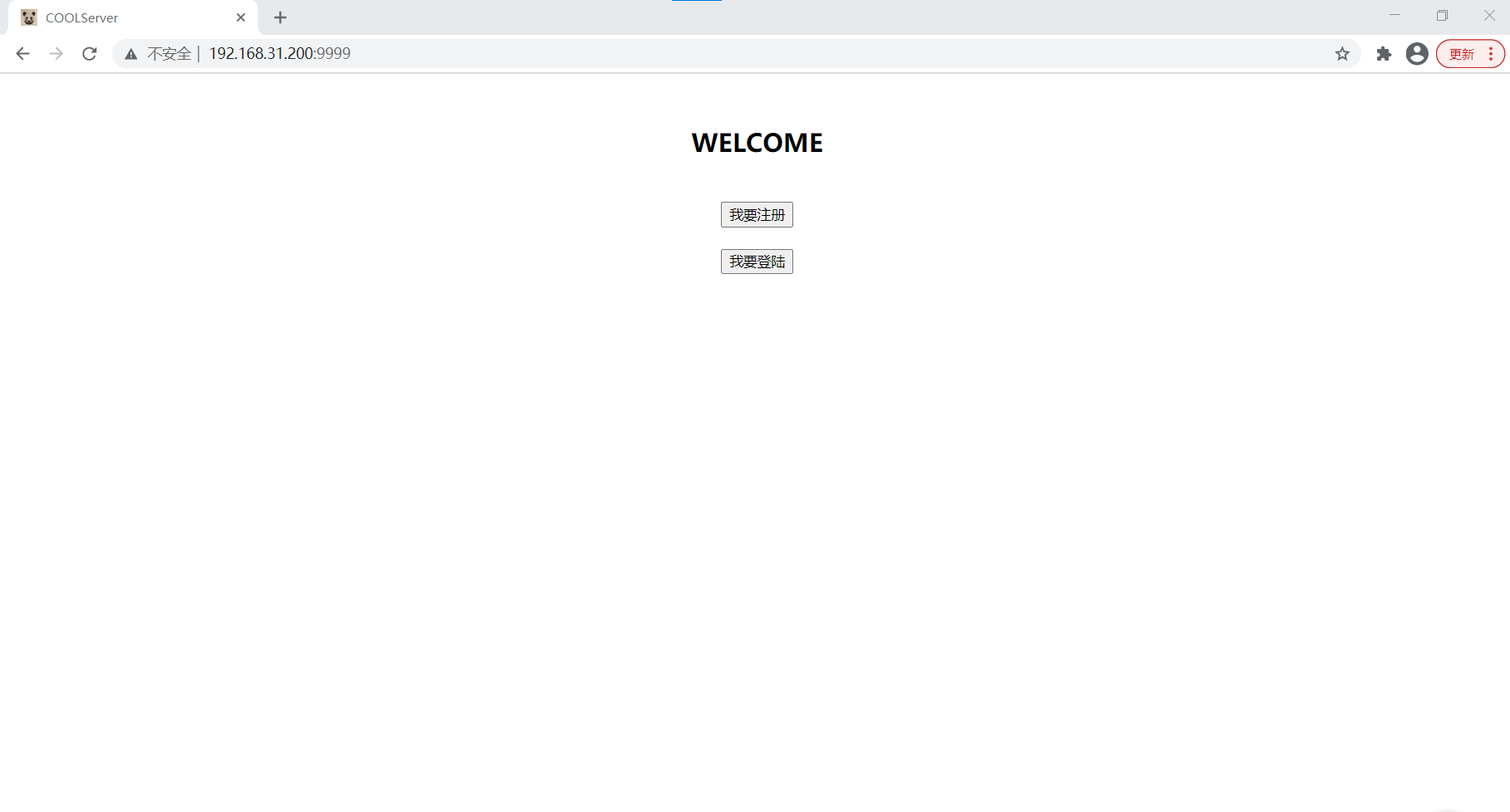
**# sh build.sh**

1. 运行

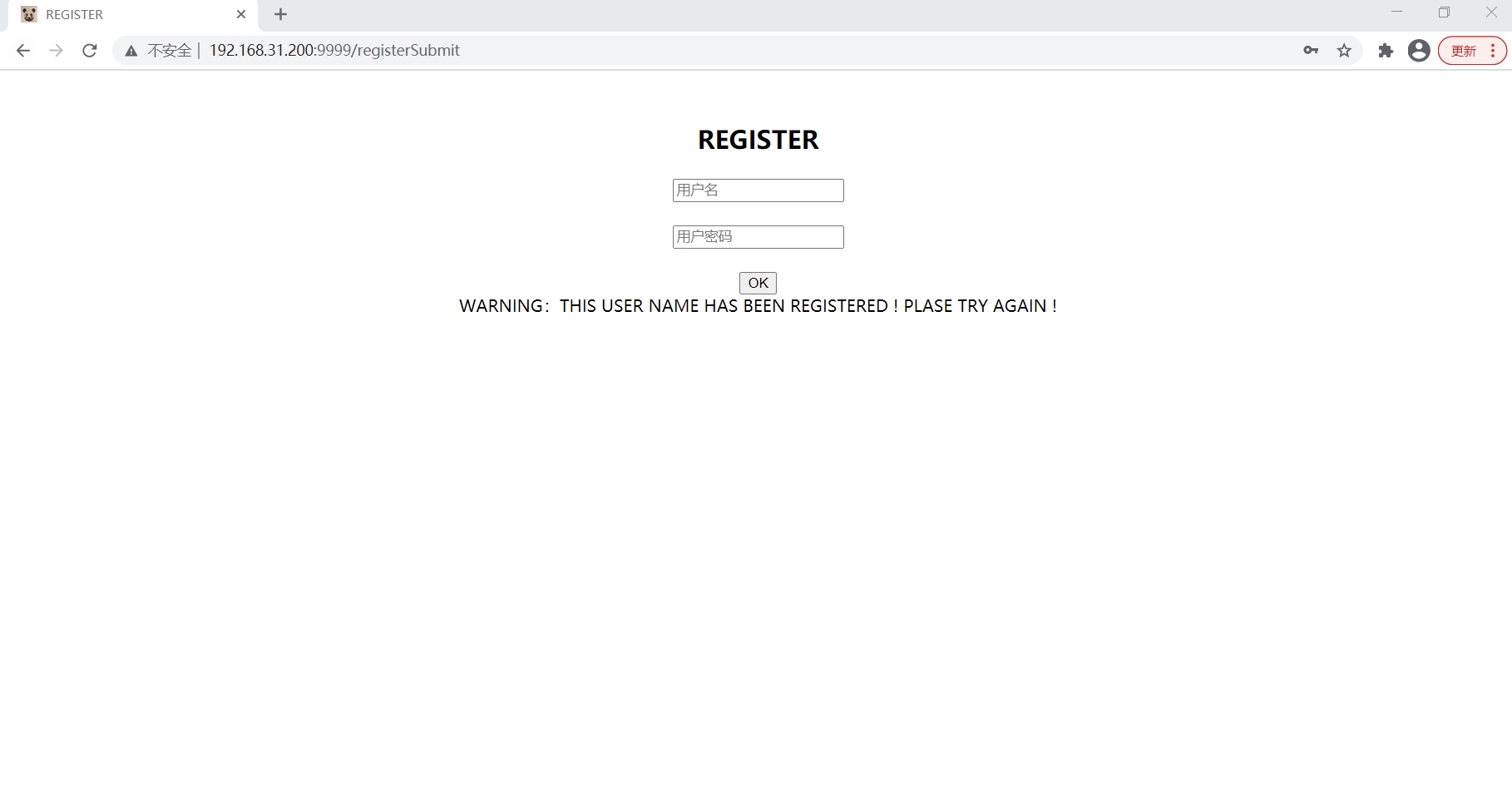
**# ./COOLServer**

**界面展示**

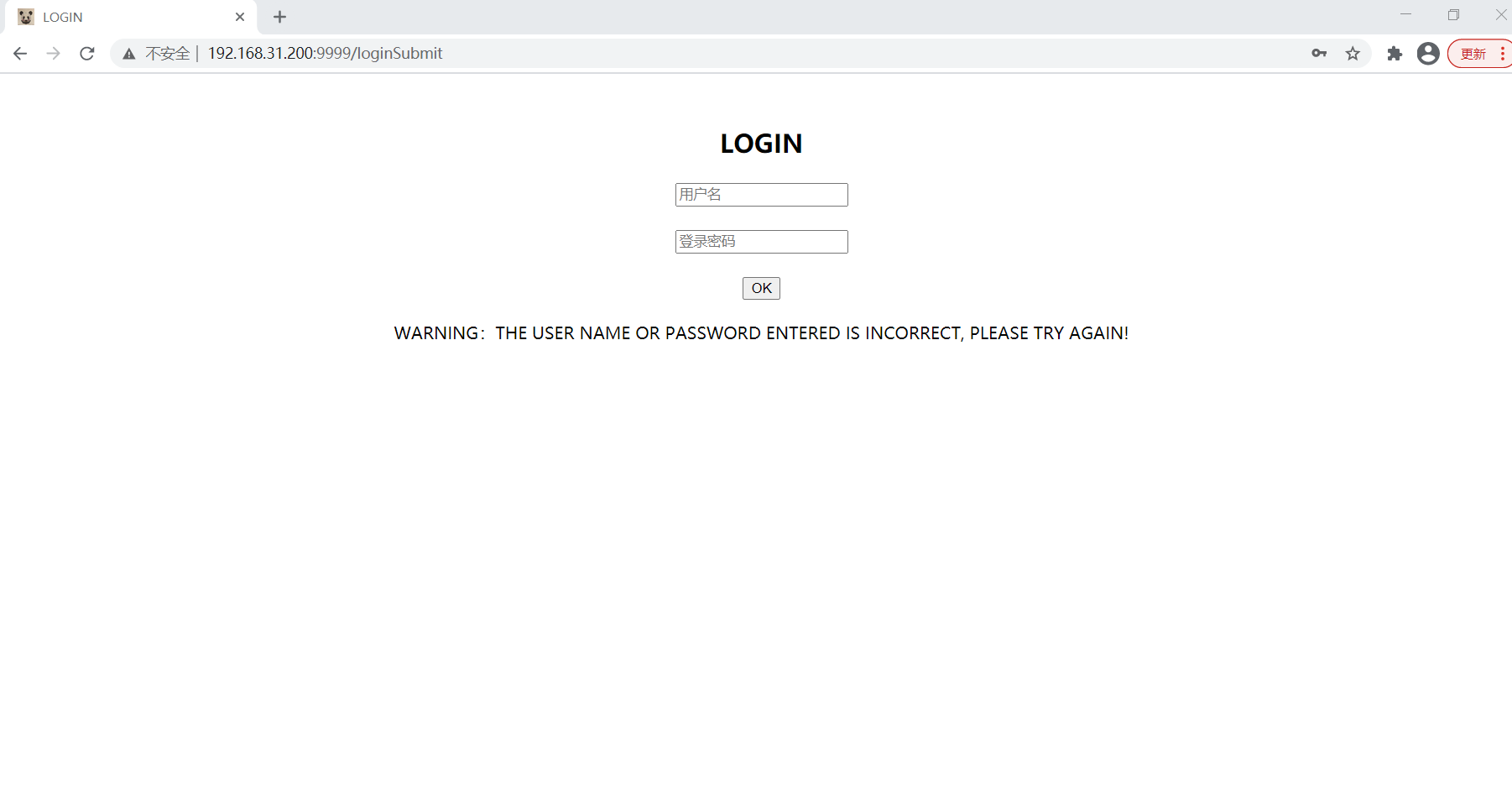
入口界面：



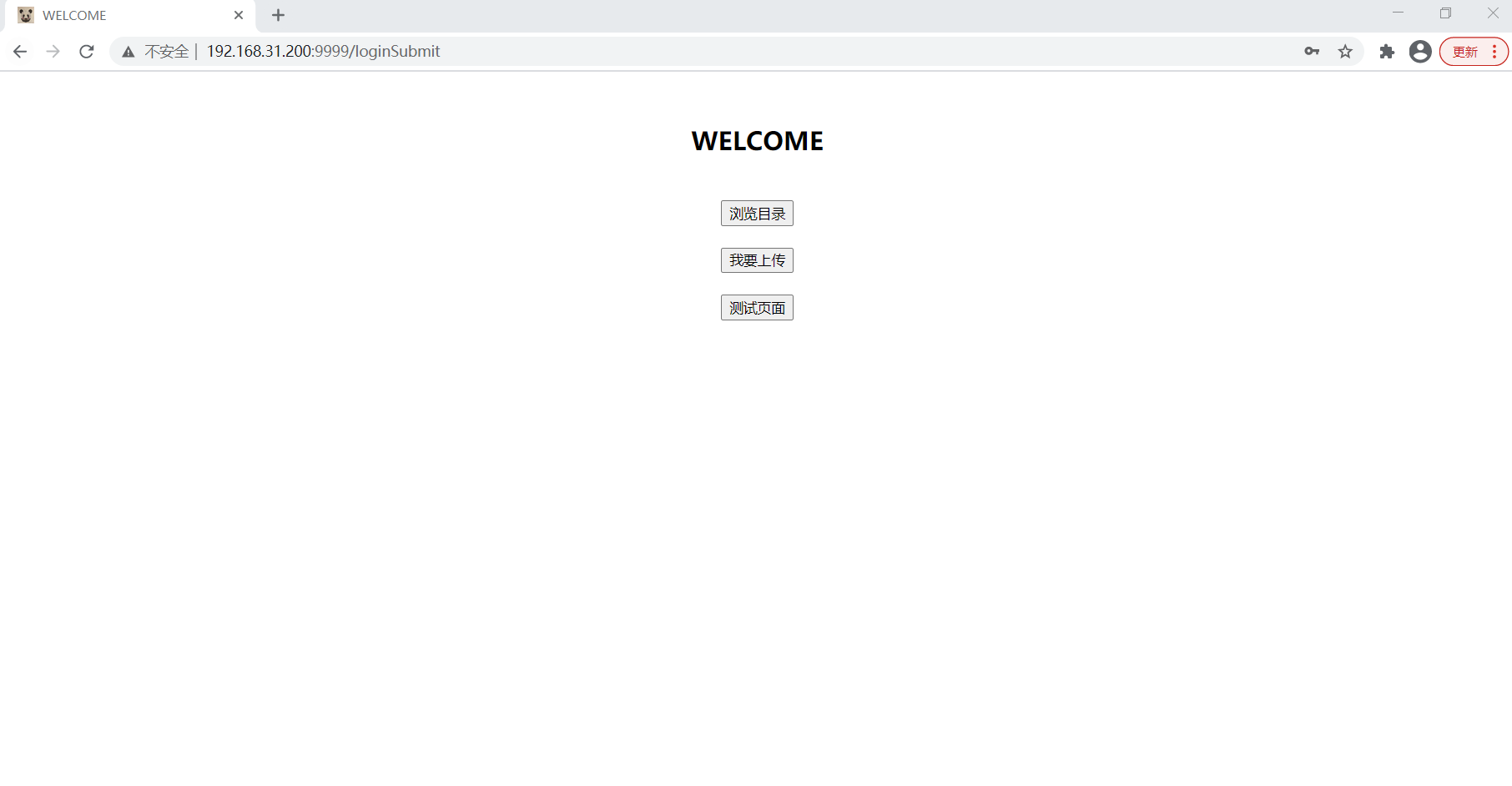
注册界面：



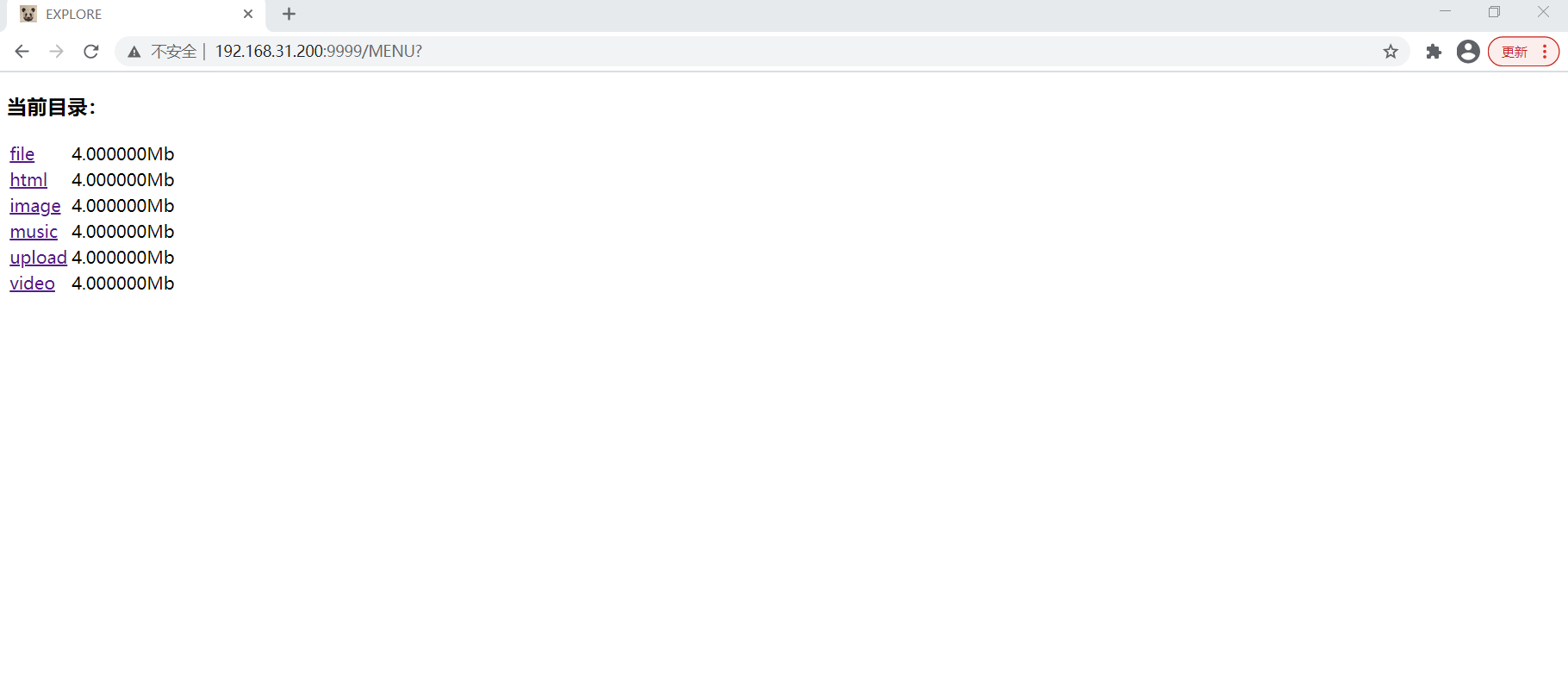
登录界面：



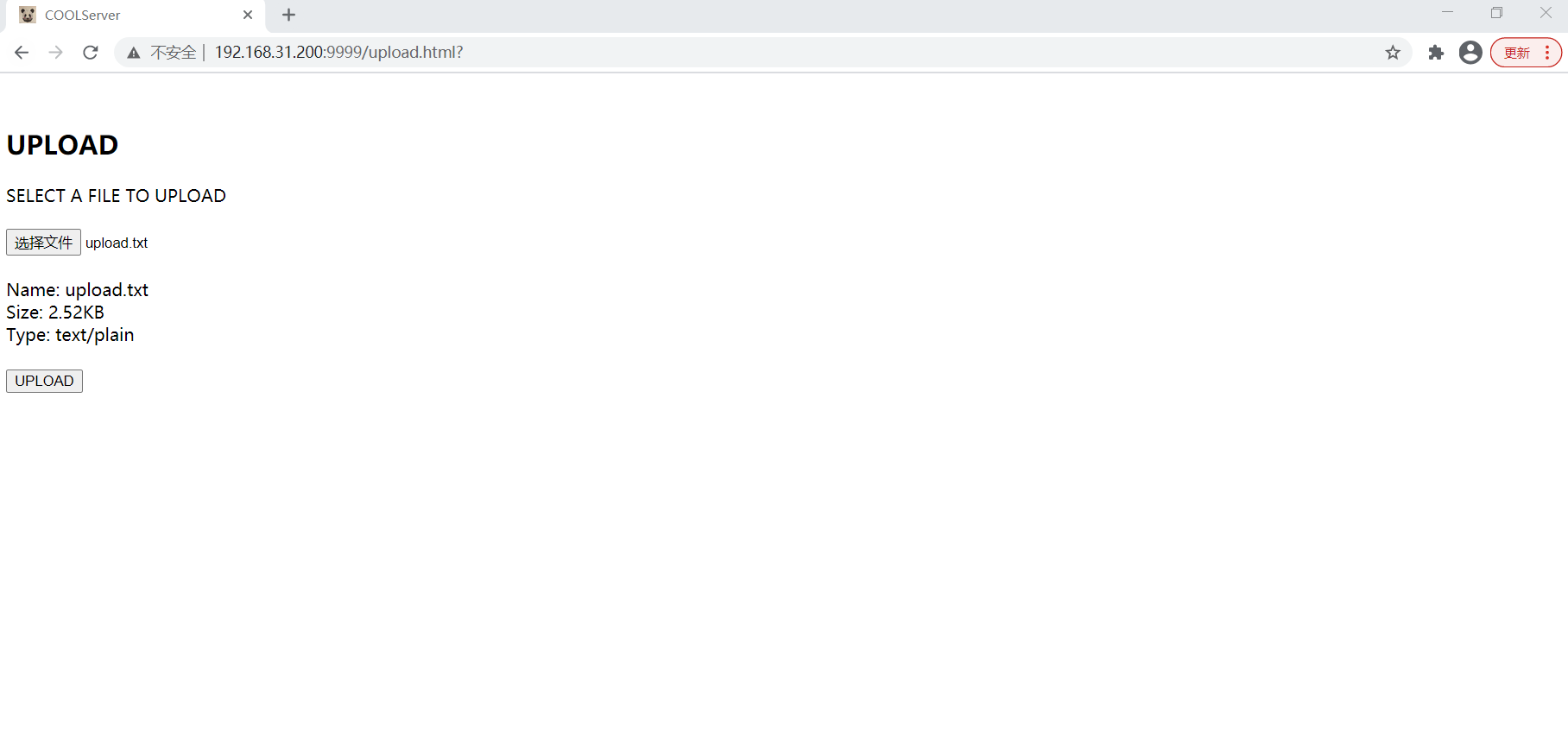
导航界面：



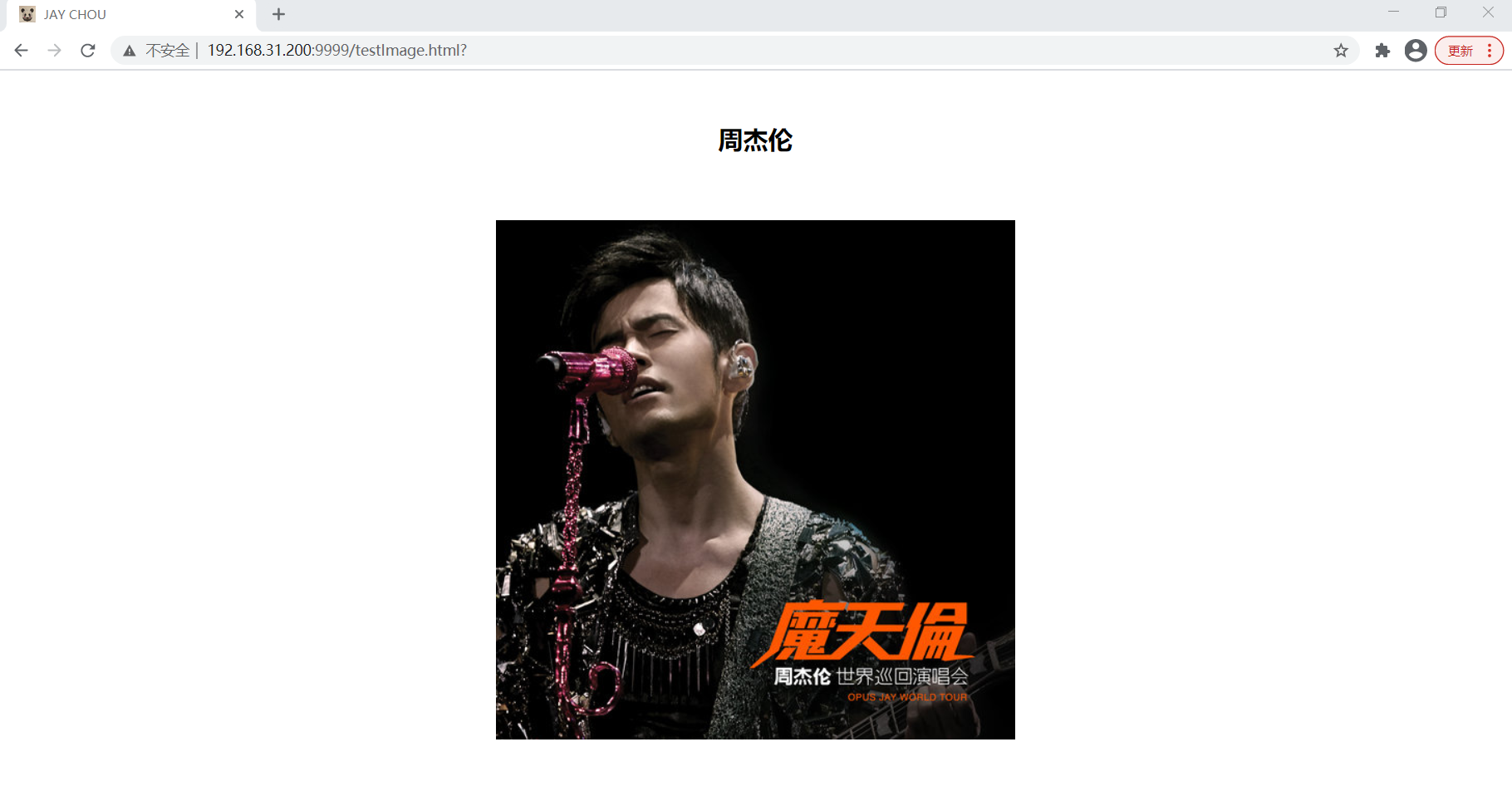
浏览目录：



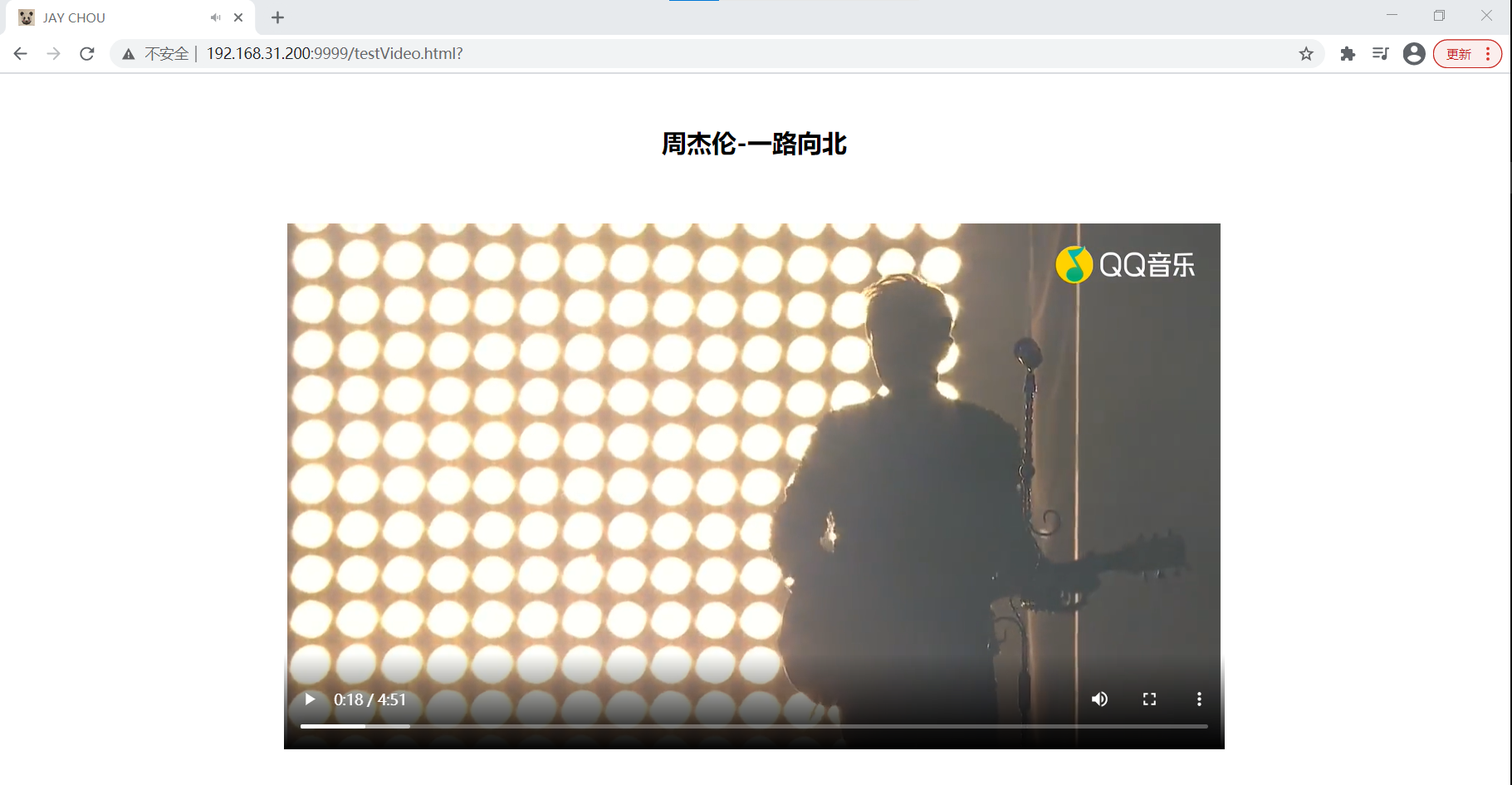
上传测试：



下行测试：







### 压力测试

测试环境为八代i5-8400机器，2核4G内存ubuntu虚拟机

**krain@krain-virtual-machine:~/webbench-1.5$ webbench -c 10000 -t 10 http://192.168.31.200:9999/**

**Webbench - Simple Web Benchmark 1.5**

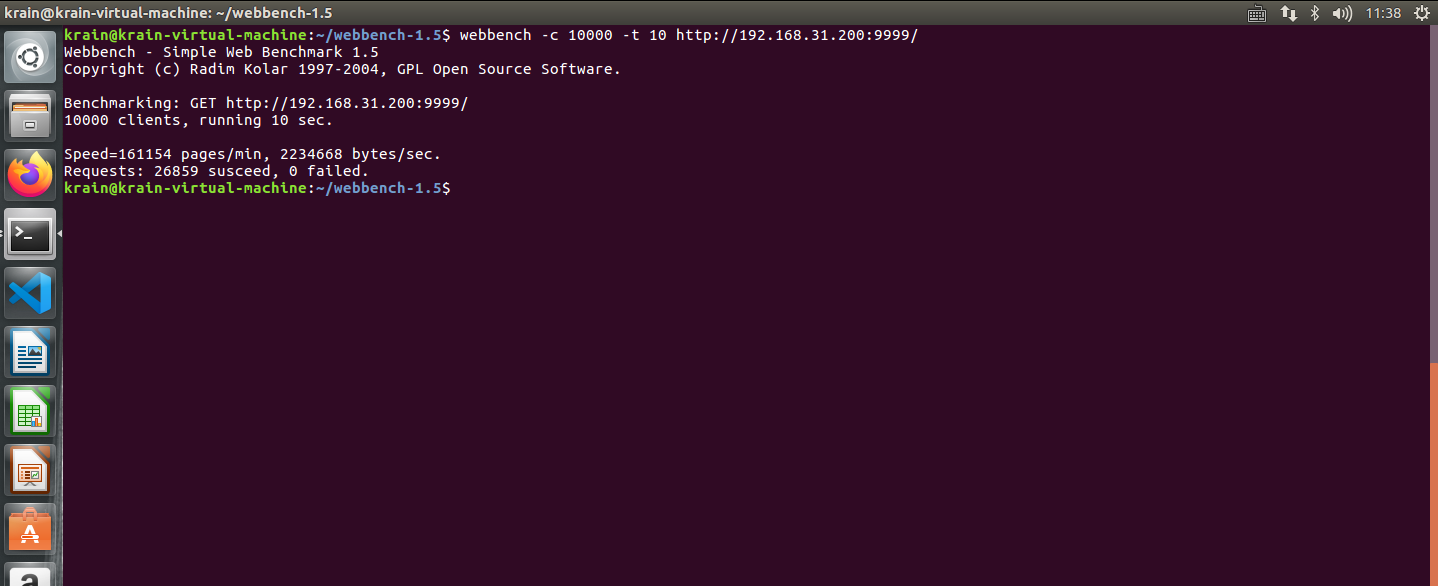
**Copyright (c) Radim Kolar 1997-2004, GPL Open Source Software.**

**Benchmarking: GET http://192.168.31.200:9999/**

**10000 clients, running 10 sec.**

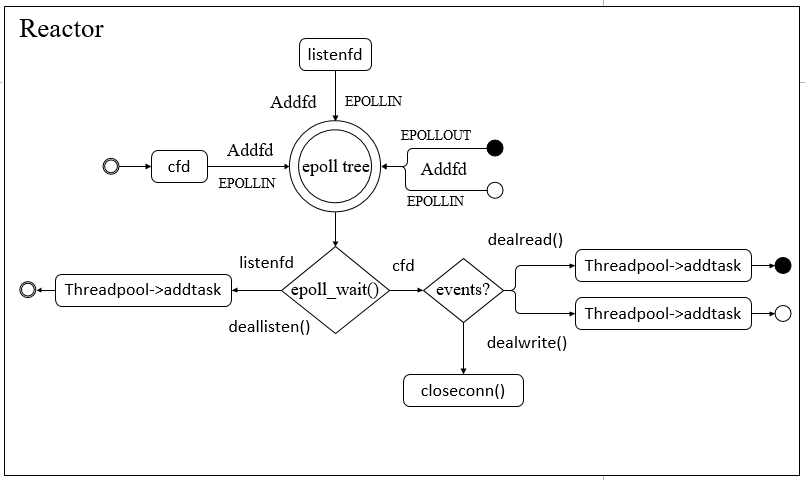
**Speed=161154 pages/min, 2234668 bytes/sec.**

**Requests: 26859 susceed, 0 failed.**



### Reactor

单reactor多线程设计，IO复用Epoll并发模型，主线程中创建一颗epoll红黑监听树，红黑树监听树用于监听所有读写事件。初始化一个listenfd用于监听客户端连接事件；每个连接事件对应一个cfd，用于监听对应客户端的读/写事件。当监听到文件描述符上有事件到达时，主线程不做实际处理，而是将处理操作作为任务添加到线程池任务队列，由线程池自主地唤醒一个空闲线程，并从任务队列取任务执行。对同一个cfd，交替监听其读写事件，即处理完读操作（读缓冲区，解析数据），将cfd添加到epoll树监听写事件，处理完写事件，将cfd添加到epoll树监听读事件。listenfd用于监听连接事件，只监听其读事件。



**ET边缘触发+非阻塞+忙轮询**

边缘触发通常指的是文件描述符在有数据到达时才触发；另一种触发方式叫做水平触发LT，只要有数据都会触发，效率较低。

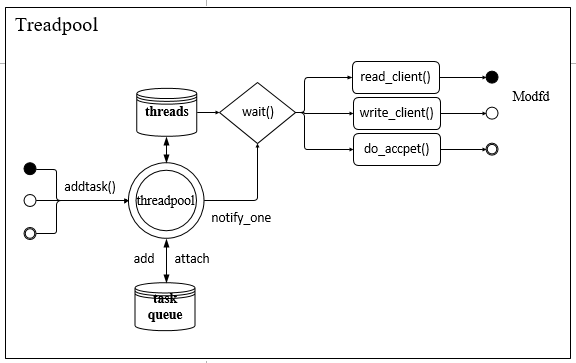
**epoll的边缘触发只支持非阻塞、忙轮询的工作模式**

ET模式下每次读写IO需要循环调用read、write直到返回EADAIN错误（没有数据可读），以读操作为例，epoll\_wait只在数据到来时才触发可读事件，如果不一次性读完所有数据，IO不会再次触发，epoll\_wait不会再次收到通知，会造成数据的丢失，所以**忙轮询**的工作方式是必要的。

同时，如果ET模式下使用阻塞IO，则程序在轮询调用最后一次read或write操作时必阻塞。

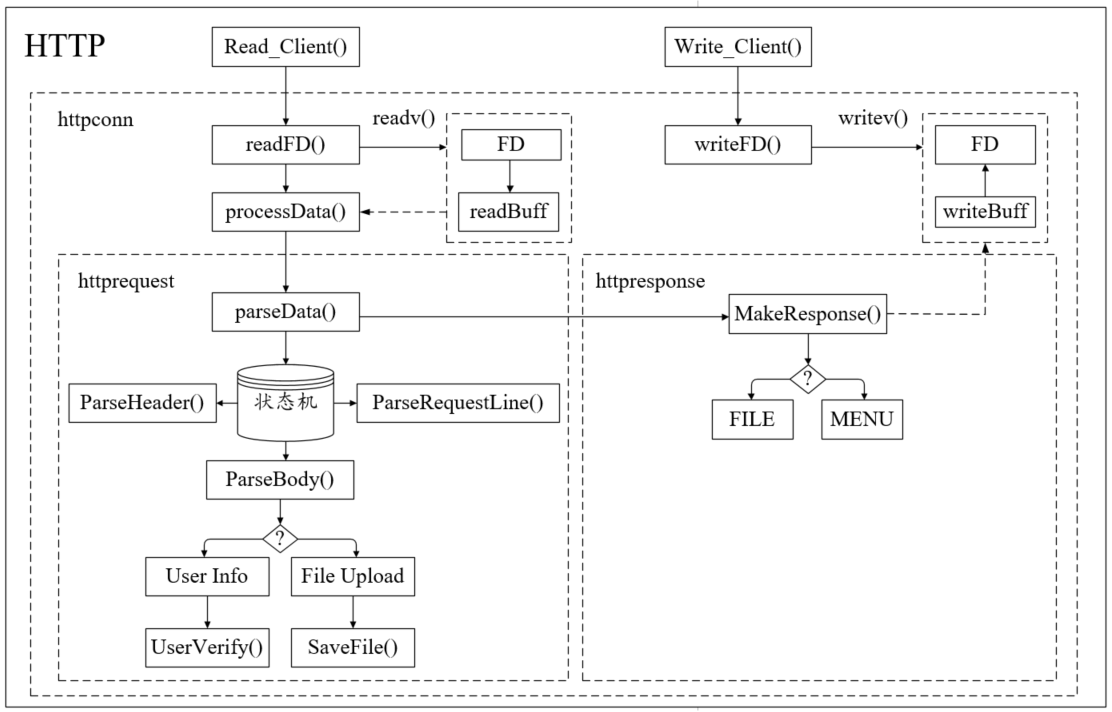
### 线程池

线程池初始化时一次性创建多个工作线程，工作线程阻塞等待任务队列中的可执行任务。主线程只负责监听文件描述符上是否有事件发生，具体的任务处理交给线程池，由线程池自主分配工作线程取任务执行。主线程每次向任务队列添加新的任务即唤醒一个阻塞线程，工作线程被唤醒时从任务队列取出任务执行，添加到队列的任务是一个函数对象的形式。



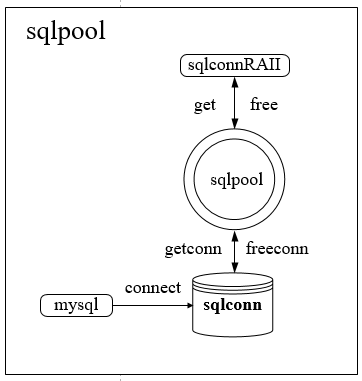
### HTTP模块

工作线程被唤醒时，从任务队列取任务执行，具体的任务主要包括读、写缓冲区以及解析数据、响应数据。HTTP模块负责从文件描述符读出数据到缓冲区，利用状态机对数据进行解析，打包待响应的数据，并将响应数据写回缓冲区。实现了主从状态机解析报文，sql数据库验证登录信息，本地保存上传文件，以及向客户端响应本地文件目录。



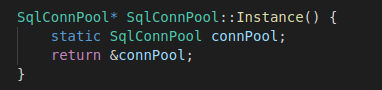
### 数据库连接池

为避免重复地创建和断开数据库连接带来的资源消耗，维护一个数据库连接池，一次性初始化多个连接并维护，当需要访问数据库时，直接从数据库连接池分配一个连接，使用完毕归还给数据库连接池。



单例化的模式设计，使得每次调用都返回同一个数据库连接池对象，并不可重复创建该类对象。

1. 将构造函数私有化，使其不能在类的外部通过new关键字实例化该类对象。
2. 封装一个静态函数，在其中创建一个静态对象，并返回其引用。这样每次通过调用这个函数调用这个唯一对象，且多次调用不会重复创建对象。



1. 通过作用域的形式调用这个静态函数，返回唯一的那个对象。

**RAII机制**

RAII全称是“Resource Acquisition is Initialization”，即 “资源获取即初始化”.

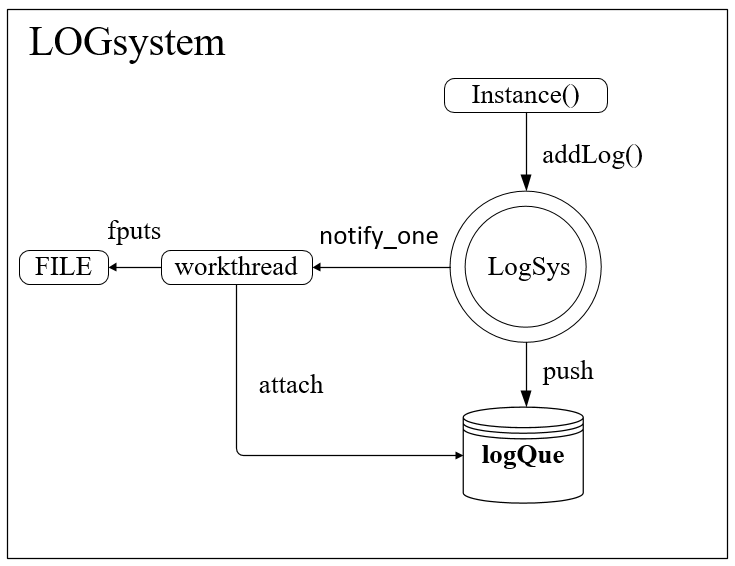
在构造函数中申请分配资源，在析构函数中释放资源。因为C++的语言机制保证了，当一个对象创建的时候，自动调用构造函数，当对象超出作用域的时候会自动调用析构函数。所以，在RAII的指导下，我们应该使用类来管理资源，**将资源和对象的生命周期绑定。**

RAII的核心思想是将资源或者状态与对象的生命周期绑定，通过C++的语言机制，实现资源和状态的安全管理，智能指针是RAII最好的例子。



### 日志系统

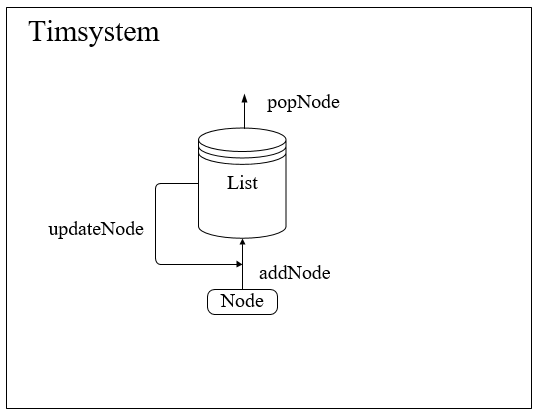
日志系统采用异步的逻辑，维护一个日志队列，主线程将需要写入的日志信息添加到日志队列，开辟一个工作线程阻塞地从日志队列取日志信息写入日志文件，日志文件句柄从初始化时即维护，避免文件重复打开关闭带来的开销。



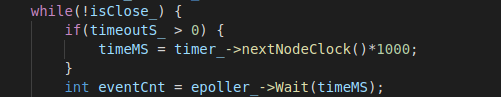
单例化模式创建日志系统，生产者-消费者模型运行，对服务器运行状态、错误信息和访问记录等数据进行记录，实现了按天记录、超行记录。

### 时钟系统

依据FIFO原则，对非活跃连接进行处理。利用双向链表维护一个定时器，每一个节点对应一个http连接，节点记录了每一个连接的预计超时时间以及超时回调函数，新加入的节点添加到链表尾部，更新时钟时从前往后清理超时连接并调用其回调函数，更新节点时首先更新其预计超时时间，然后将其移至队尾。



清理非活跃连接的逻辑：



主线程每一次epoll\_wait调用之前，对定时器非活跃连接进行清理，并返回最近一个将会超时的连接所余下的时间（nextNodeClock函数首先对已经超时的连接进行清理，然后返回最近一个将会超时的连接所余下的时间），并将这个时间作为epoll\_wait的超时时间，如果这段时间内没有IO事件触发，则进入下一个循环继续对定时器进行处理。