Tinywebserver——服务器常问面试题！

在Tinywebserver这个项目最后，社长提出了一些可能的面试问题，读者学习完该项目之后可以试着回答一下，看是否究竟对这个项目了如指掌：：包括**项目介绍，线程池相关，并发模型相关，HTTP报文解析相关，定时器相关，日志相关，压测相关，综合能力**等。

项目介绍

为什么要做这样一个项目?

对后台开发的工作及知识非常感兴趣 因此想学习有关服务器后台开发的相关知识

介绍下你的项目

使用**线程池+非阻塞socket+epoll（ET和LT均实现）**+ 事务处理（reactor和proacter均实现）的并发模型

使用**状态机解析**HTTP请求报文 **支持解析GET和POST请求**

访问服务器数据库实现**web用户注册 登录功能 可以请求服务器图片和视频文件**

实现**同步/异步日志系统** 记录服务器运行状态

经webbench压力测试可以实现上万的并发连接数据交换

- 利用C++11编写的web服务器项目
- 使用了线程池技术 采用非阻塞socket和epoll的IO复用技术 使用reactor和模拟proactor实现的半同步半反应堆事件处理模式实现的并发模型
- 使用了状态机解析http请求报文 支持解析get和post请求 可以请求服务器图片与视频等文件
- 利用数据库连接池实现了与数据库的连接 实现了web用户的注册 登录功能
- 统一事件源 利用升序链表维护定时任务 利用定时器处理非活动连接
- 实现了同步/异步日志的系统 记录服务器运行状态
- 经webbench压力测试可以实现上万的并发连接数据交换

线程池相关

手写线程池

线程的同步机制有哪些

信号量 条件变量 互斥量等 读写锁等

线程池中的工作线程是一直等待吗

是的 等待新任务的唤醒

你的线程池工作线程处理完一个任务后的状态是什么

如果请求队列为空 则该线程进入线程池中等待 若不为空 则该线程跟其他线程一起进行任务的竞争

如果1000个客户端进行访问请求 线程数不多 怎么能及时响应处理每一个呢

该项目是基于IO复用的并发模式 需要注意的是 不是一个客户连接就对应一个线程 如果真是如此 淘宝双12服务器早就崩了 当客户连接有事件需要处理时 epoll会进行事件提醒 **而后对应的任务加入请求队列 等待工作线程竞争执行 如果速度还是慢 那就只能增大线程池容量** 或者考虑集群分布式的做法

如果一个客户请求需要占用很久的时间 会不会影响接下来的客户请求呢 有什么好的策略呢

会 因为线程的数量是固定的 如果一个客户请求长期占据着线程资源 势必会影响到服务器对外的整体响应速度

解决的策略是可以给每一个线程处理任务设定一个时间阈值 当某一个客户请求时间过长 则将其置于请求任务最后 或者断开连接

并发模型相关

简要说一下服务器使用的并发模型?

该项目选用模拟proactor实现的**半同步半反应堆**的并发模型

- 异步线程只有一个 由主线程来充当
- 它负责监听所有的socket上的事件 如果监听的socket上有新的连接请求 那么就接受它以得到新的连接socket 然后往epoll的读写事件表上注册该socket的上的读写事件
- 如果有读写事件发生 那么主线程完成数据的读写工作 然后将数据与任务等封装插入请求队列
- 睡眠在请求队列的工作线程被唤醒 取得任务对象后 进行工作逻辑的处理

也可以使用reactor实现的半同步半反应堆的并发模型

- 异步线程只有一个 由主线程来充当
- 它负责监听所有的socket上的事件 如果监听的socket上有新的连接请求 那么就接受它以得到新的连接socket 然后往epoll的读写事件表上注册该socket的上的读写事件
- 如果有读写事件发生 那么主线程就将该连接socket插入到请求队列中
- 睡眠在请求队列的工作线程被唤醒 获得连接socket 然后从socket上执行读写工作 接着根据获取的数据 进行逻辑处理

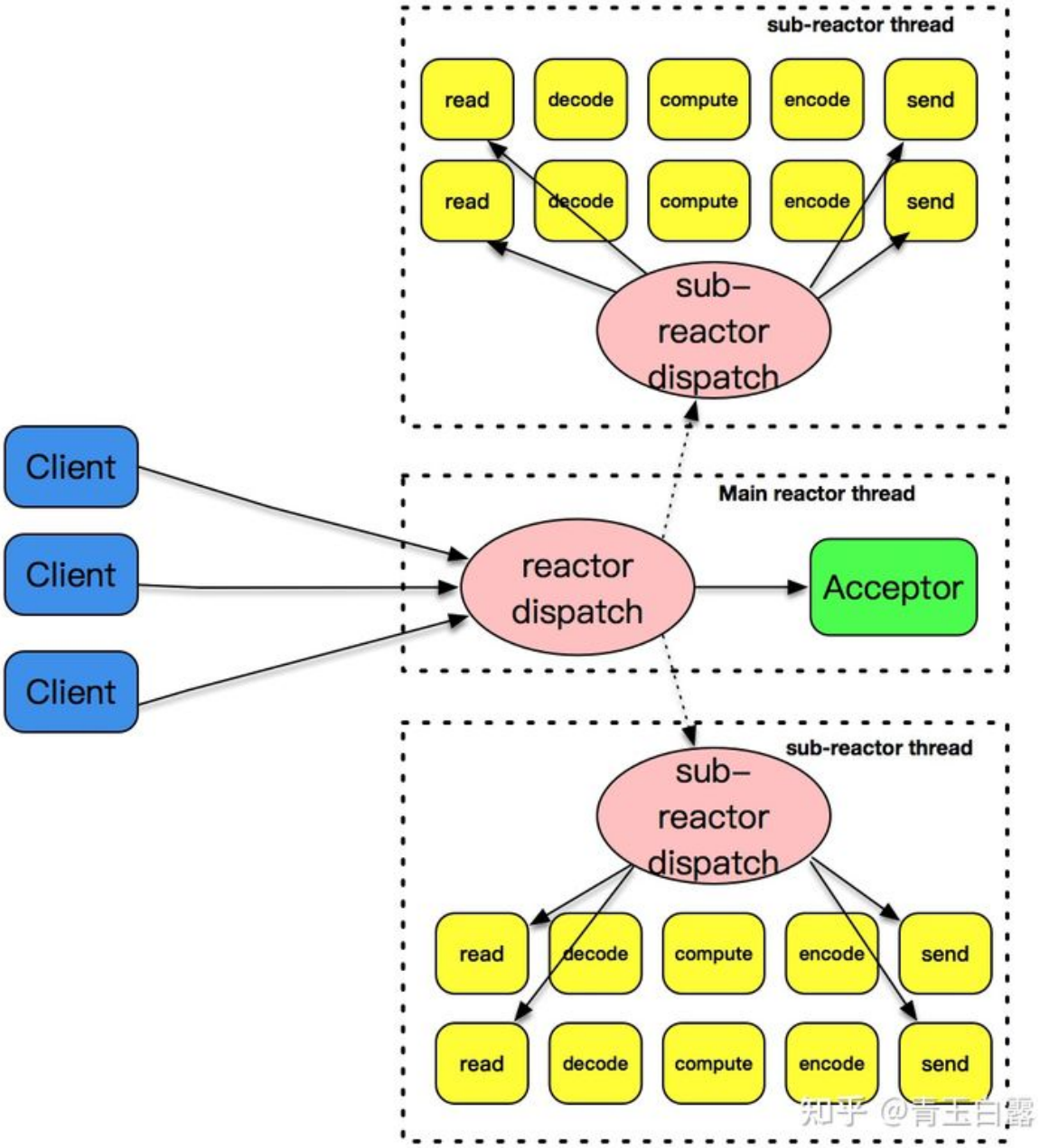
reactor(反应堆)、poractor 主从reactor模型的区别?

- **同步IO用来实现reactor 异步IO用来实现proactor模式**
- reactor模式：要求主线程（I/O线程）**只负责监听文件描述符上是否有事件发生**（可读 可写） **若有 则立即通知工作线程 工作线程处理除此之外的所有工作**------ **将socket可读可写事件放入请求队列 读写数据 接受新连接及处理客户请求均在工作线程中完成** （需要区别读和写事件） reactor将IO事件分配给对应的handler acceptor处理客户端的新连接 并分派请求到处理器链中 handler执行非阻塞读写任务

- proactor模式：**利用了异步IO proactor模式把IO操作全都给了内核处理 主线程和内核负责处理读写数据、接受新连接等I/O操作 工作线程仅负责业务逻辑（给予相应的返回url） 如处理客户请求 解耦了其他部分**
- 模拟proactor模式 因为linux对异步IO的支持不是很完美 故一般用**同步IO**来模拟proactor模式 前面说了 proactor让工作线程只解决业务逻辑 这里模拟的proactor模式也是 **把IO任务(read write)交**

给了主线程去做 完成后再把数据交给工作线程

- 主从reactor模式 核心思想是 **主反应堆线程只负责分发acceptor连接建立 已连接套接字上的I/O事件交给sub-reactor负责分发** 其中sub-reactor的数量 可以根据CPU的核数来灵活设置



- 主反应堆线程一直在感知连接建立的事件 如果有连接成功建立 主反应堆线程通过accept方法获取已连接套接字 接下来会按照一定的算法选取一个从反应堆线程 并把已经连接套接字加入到选择好的从反应堆线程中 **主反应堆线程唯一的工作 就是调用accept获取已经连接的套接字** 以及将已经连接套接字加入到从反应堆线程中

半同步半反应堆?

- 为啥说是半同步半反应堆呢 因为**用到了reactor模式 IO线程只监听连接到来事件 而工作线程要执行读写**
- 而半同步半反应堆模式可以用模拟proactor来模拟完成 **即让IO线程完成读写 IO线程向工作线程分发已经接受的数据**

你用了epoll 说一下为什么用epoll 还有其他的复用方式吗 区别是什么?

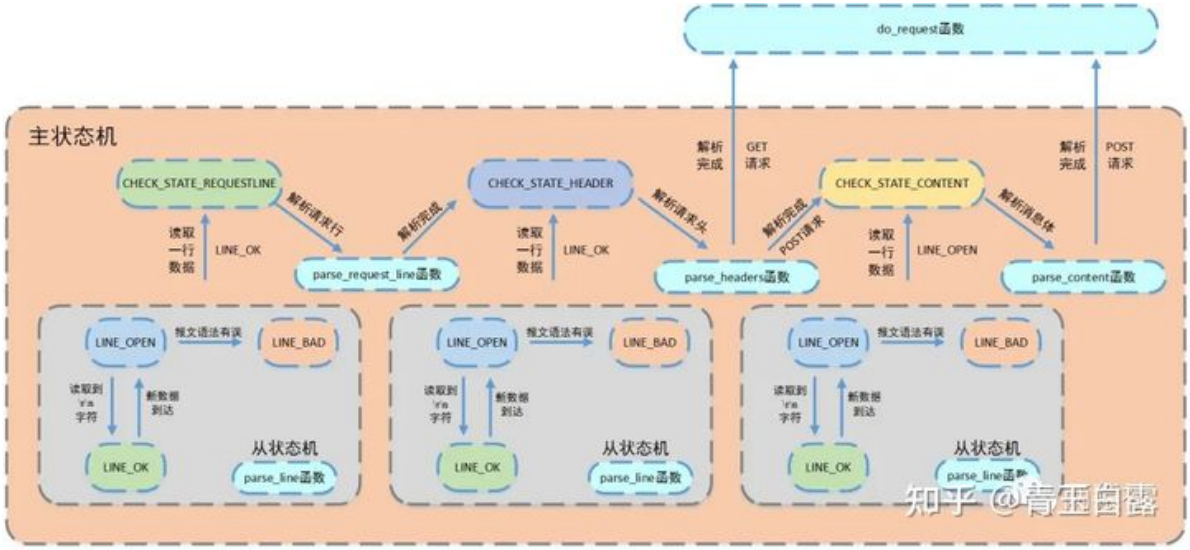
比较常用的服用方式有三种 select/poll/epoll。本项目之所以采用epoll

- 对于select和poll来说 所有文件描述符都是在用户态被加入到其文件描述符集合的 每次调用都需要将整个集合拷贝到内核态 **epoll则将整个文件描述符集合维护在内核态 每次添加文件描述符的时候都需要执行一个系统调用** 系统调用的开销时很大的 **而且有很多短期活跃连接的情况下 epoll可能会慢于select和poll由于这些大量的系统调用的开销**
- select使用线性描述文件描述符集合 **文件描述符有上限 poll使用链表来描述 epoll底层通过红黑树来描述 并且维护一个ready_list 将事件表中已经就绪的事件添加到这里 在使用epoll_wait调用时仅观察这个list有没有数据即可**
- **select和poll的最大开销来自内核判断是否有文件描述符就绪这一过程** 每次执行select或poll调用时 他们会采用遍历的方式 遍历整个文件描述符集合去判断各个文件描述符是否有活动 epoll则不需要去以这种方式检查 当有活动产生时 会自动触发epoll回调函数通知epoll文件描述符 **然后内核将这些就绪的文件描述符放到了之前提到的ready_list中等待epoll_wait调用后被处理**
- select和poll都只能工作在相对**低效的LT模式下 而epoll同时支持LT和ET模式**
- 综上 当**监测的fd数量较小 且各个fd都很活跃的情况下 建议使用select和poll 当监听的fd数量较多且单位时间仅仅部分fd活跃的情况下 使用epoll会明显提升性能**

HTTP报文解析相关

用了状态机啊 为什么要用状态机?

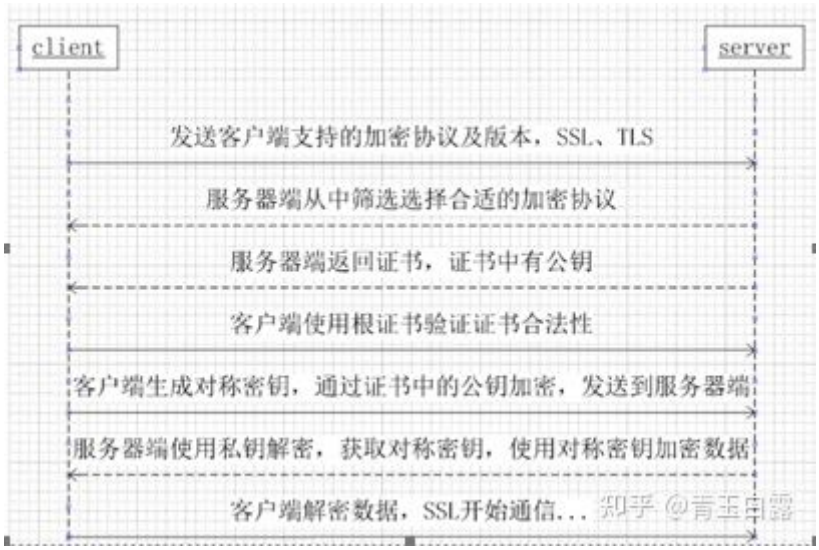
有限状态机是一种抽象的理论模型 它能够把有限个变量描述的变化过程 以可构造可验证的方式呈现出来 比如封闭的有向图 有限状态机可以通过if-else switch-case和函数指针来实现 从软件工程角度来看 主要为了封装逻辑 有限状态机一种逻辑单元内部的一种高效编程方法 在服务器编程中 服务器可以根据不同状态或者消息类型进行相应的处理逻辑 使得程序逻辑清晰易懂



https协议为什么安全

连接建立阶段基于ssl安全验证 数据传输阶段加密

https的ssl连接过程?



GET和POST报文的区别?

最直观的区别就是GET把参数包含在URL中 POST通过消息体传递参数

get请求参数会被完整保留在浏览器历史记录里面 而post中的参数不会被保留

get请求在URL中传送的参数是有长度限制 大多数浏览器会现在url长度在2k个子节 而大多数服务器最多处理64k大小的url

get产生一个TCP数据包 而POST产生两个数据包 对于get方式的请求 浏览器会把http header和data一并发送出去 服务器响应200（返回数据） 而对于post 浏览器会先发送header 服务器响应100（指示信息表示已接收 继续处理）continue 浏览器再发送data 服务器响应200 ok（返回数据）

实际上

如果我告诉你GET和POST本质上没有区别你信吗?

让我们扒下GET和POST的外衣，坦诚相见吧!

get和post是http协议中两种发送请求的方法 http的底层实tcp/ip 所以get和post的底层也是tcp/ip 也就是说 GET/POST都是TCP连接 get和post能做的事情是一样的 比如要给get加上消息体 给post带上url参数 技术上是完全行得通的

但是由于浏览器与服务器会形成一些限制 一些不成文规定 （大多数）浏览器通常都会限制url长度在2K个字节，而（大多数）服务器最多处理64K大小的url。超过的部分，恕不处理

如果你用GET服务，在request body偷偷藏了数据，不同服务器的处理方式也是不同的，有些服务器会帮你卸货，读出数据，有些服务器直接忽略，所以，虽然GET可以带request body，也不能保证一定能被接收到哦

数据库登陆注册相关

登录说一下?

具体的涉及到载入数据库表 提取用户名和密码 注册登录流程与页面跳转

- 载入数据库表 结合代码**将数据库中的数据载入到服务器中**
- 提取用户名和密码 结合代码**对报文进行解析 提取用户名和密码 、**
- 注册登录流程 结合代码对描述**服务器进行注册和登录校验的流程**
- 页面跳转 结合代码对**页面跳转机制进行详解**

你这个状态保存了吗 如果需要保存 你会怎么做?

可以利用session或者cookie的方式进行状态的保存

cookie其实就是服务器给客户分配了一串身份标识 比如 123456789这么一串字符串 每**次客户发送数据时 都在HTTP报文上附带这个字符串 服务器就知道你是谁了**

session是保存在服务器端的状态 每当一个客户发送HTTP报文过来的时候 服务器会在自己记录的用户数据中去找 **类似于核对名单**

登录中的用户名和密码你是load到本地 然后使用map匹配的 如果有十亿数据 即使load到本地后hash 也是很耗时的 你要怎么优化

这个问题的关键在于大量数据量的情况下用户的登陆验证怎么进行 将所有的用户信息加载到内存中耗时耗力 对于大量数据最便利的方法就是进行hash **利用hash建立多级索引的方式来加快用户验证**

首先将10亿的用户信息 利用大致缩小1000倍的hash算法进行hash 这时就获得了100万的hash数据 每一个hash数据代表着一个用户信息块（一级）

而后 再分别对这100万的hash数据再进行hash 例如最终剩下1000个hash数据（二级）

在这种情况下 服务器只需要保存1000个二级的hash数据 当用户请求登陆的时候 现堆用户信息进行一次hash 找到对应信息块（二级） 在读取其对应的一级信息块 最终找到对应的用户数据

定时器相关

为什么要用定时器?

处理定时任务 或者非活动连接 节省系统资源

说一下定时器的工作原理?

服务器就为各事件分配一个定时器 该项目**使用SIGALRM信号来实现定时器 首先每一个定时事件都处于一个升序链表上 通过alarm函数周期性触发SIGALRM信号** 而后信号回调函数**利用管道通知主循环 主循环接收到信号之后对升序链表上的定时器进行处理 若一定时间内无数据交换则关闭连接**

双向链表啊 删除和添加的时间复杂度说一下 还可以优化吗?

添加一般情况下都为O(N) 删除只需要O(1) 从双向链表的方式优化不太现实 可以考虑使用最小堆 或者跳表的数据结构

最小堆优化 说一下时间复杂度和工作原理?

最小堆以每个定时器的过期时间进行排序 最小的定时器位于堆顶 当SIGALRM信号触发tick函数时执行过期定时器清除 如果堆顶的定时器时间过期 则删除 并重新建堆 再判定是否过期 如此循环知道未过期为止

插入，O(logn)；

删除，O(logN)；

日志相关

说下你的日志系统的运行机制

初始化服务器的同时 利用单例模式初始化日志系统 根据配置文件确认是同步还是异步写入的方式

为什么要异步？和同步的区别事什么？

同步方式写入日志会产生比较多的系统调用 若是某条日志信息过大 会阻塞日志系统 造成系统瓶颈 **异步方式采用生产者-消费者模型 具有较高的并发能力**

现在你要监控一台服务器的状态 输出监控日志 请问如何将该日志分发到不同机器上？

为了便于故障排查 或服务器状态分析 看是否需要维护 可以使用**消息队列**进行消息的分发 例如mqtt rabitmq等等

压测相关

服务器并发量测试过吗？怎么测试的？

测试过 利用webbench 至少满足万余的并发量

webbench是什么 介绍一下原理？

是一款**轻量级的网址压力测试工具** 可以实现高达**3万**的并发测试 其原理：webbench实现的核心原理是：父进程fork若干个子进程 每个子进程在用户要求时间或者默认的时间内对目标web循环发出实际访问请求 **父子进程通过管道进行通信** 子进程通过**管道写端**向父进程传递在若干次请求访问完毕后记录到的总信息 父进程通过管道**读端**读取子进程发来的相关信息 **子进程在时间到后结束 父进程在所有子进程退出后统计并给用户显示最后的测试结果 然后退出**

综合能力

说一下前端发送请求后 服务器处理的过程 中间涉及哪些协议

HTTP协议 TCP IP 协议等 计算机网络的知识