简单说一下服务器使用的并发模型？

在并发模型上，采用epoll边沿触发模式来实现IO复用，使用同步IO模拟Proactor事件处理模式，实现半同步/半反应堆线程池，主线程为异步线程，负责监听文件描述符，接收socket新连接，若当前监听的socket发生了读写事件，然后将任务插入到请求队列。工作线程从请求队列中取出任务，完成读写数据的处理，解除了主线程和工作线程的耦合关系

reactor、proactor、主从reactor模型的区别？

- reactor模式中，主线程(**I/O处理单元**)只负责监听文件描述符上是否有事件发生，有的话立即通知工作线程(**逻辑单元**)，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。通常由**同步I/O**实现。
- proactor模式中，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，如处理客户请求。通常由**异步I/O**实现。
- 主从reactor

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

epoll工作过程

1. **epoll_wait()调用ep_poll()**，如果就绪链表**rdlist**为空，则挂起当前进程，直到**rdlist**不为空时被唤醒，这个时候会调用**ep_send_events()将实际发生的事件revents**和**data**从内核空间拷贝到用户空间（拷贝调用的是__put_user，并不存在什么共享内存之类的）。
2. 当文件描述符fd的**状态改变**时（buffer由不可读变为可读或由不可写变为可写），导致相应fd上的回调函数**ep_poll_callback()** 被调用。
3. **ep_poll_callback()将有事件发生的文件描述符（epitem）加入到就绪链表rdlist** 中，这时候就绪链表不为空，**epoll_wait()** 进程被唤醒。
4. **ep_send_events()会扫描就绪链表**，调用每个文件描述符的**poll**函数返回**revents**，之后将**revents**和**data**从内核空间拷贝到用户空间。如果是**ET模式**, epitem是**不会再进入到就绪链表，除非fd再次发生了状态改变**, **ep_poll_callback**被调用。如果是**LT模式**，不但会将对应的数据返回给用户，并且会**将当前的epitem再次加入到rdllist中。**这样如果下次再次被唤醒就会给用户空间再次返回事件。

select poll epoll区别

调用函数、文件描述符数量、将文件描述符从用户传给内核、内核判断就绪的文件描述符、应用程序索引就绪文件描述符、工作模式、应用场景

- 调用函数
- - select和poll都是一个函数，epoll是一组函数
- 文件描述符数量
- - select通过fd_set位数组来描述文件描述符集合，文件描述符有上限，一般是1024，但可以修改源码，重新编译内核，不推荐
  - poll是链表描述，突破了文件描述符上限，最大可以打开文件的数目
  - epoll通过红黑树描述，最大可以打开文件的数目，可以通过命令ulimit -n number修改，仅对当前终端有效
- 将文件描述符从用户传给内核
- - select和poll通过将所有文件描述符拷贝到内核态，每次调用都需要拷贝
  - epoll通过epoll_create建立一棵红黑树，通过epoll_ctl将要监听的文件描述符注册到红黑树上
- 内核判断就绪的文件描述符
- - select和poll通过遍历文件描述符集合，判断哪个文件描述符上有事件发生

    - epoll_create时，内核除了帮我们在epoll文件系统里建了个红黑树用于存储以后epoll_ctl传来的fd外，还会再建立一个list链表，用于存储准备就绪的事件，当epoll_wait调用时，仅仅观察这个list链表里有没有数据即可。
    - epoll是根据每个fd上面的回调函数(中断函数)判断，只有发生了事件的socket才会主动的去调用 callback函数，其他空闲状态socket则不会，若是就绪事件，插入list
- 应用程序索引就绪文件描述符
-
    - select/poll只返回发生了事件的文件描述符的个数，若知道是哪个发生了事件，同样需要遍历
    - epoll返回的发生了事件的个数和结构体数组，结构体包含socket的信息，因此直接处理返回的数组即可
- 工作模式
-
    - select和poll都只能工作在相对低效的LT模式下
    - epoll则可以工作在ET高效模式，并且epoll还支持EPOLLONESHOT事件，该事件能进一步减少可读、可写和异常事件被触发的次数。
- 应用场景
-
    - 当所有的fd都是活跃连接，使用epoll，需要建立文件系统，红黑书和链表对于此来说，效率反而不高，不如selece和poll
    - 当监测的fd数目较小，且各个fd都比较活跃，建议使用select或者poll
    - 当监测的fd数目非常大，成千上万，且单位时间只有其中的一部分fd处于就绪状态，这个时候使用epoll能够明显提升性能

ET、LT、EPOLLONESHOT

- LT水平触发模式
-
    - epoll_wait检测到文件描述符有事件发生，则将其通知给应用程序，应用程序可以不立即处理该事件。
    - 当下一次调用epoll_wait时，epoll_wait还会再次向应用程序报告此事件，直至被处理
- ET边缘触发模式
-
    - epoll_wait检测到文件描述符有事件发生，则将其通知给应用程序，应用程序必须立即处理该事件
    - 必须要一次性将数据读取完，使用非阻塞I/O，读取到出现eagain
- EPOLLONESHOT
-
    - 一个线程读取某个socket上的数据后开始处理数据，在处理过程中该socket上又有新数据可读，此时另一个线程被唤醒读取，此时出现两个线程处理同一个socket
    - 我们期望的是一个socket连接在任一时刻都只被一个线程处理，通过epoll_ctl对该文件描述符注册epolloneshot事件，一个线程处理socket时，其他线程将无法处理，**当该线程处理完后，需要通过epoll_ctl重置epolloneshot事件**