* **非阻塞IO**:非阻塞等待，每隔一段时间就去检测IO事件是否就绪。没有就绪就可以做其他事。非阻塞I/O执行系统调用总是立即返回，不管时间是否已经发生，若时间没有发生，则返回-1，此时可以根据errno区分这两种情况，对于accept，recv和send，事件未发生时，errno通常被设置成eagain

线程池

* 空间换时间,浪费服务器的硬件资源,换取运行效率.
* 池是一组资源的集合,这组资源在服务器启动之初就被完全创建好并初始化,这称为静态资源.
* 当服务器进入正式运行阶段,开始处理客户请求的时候,如果它需要相关的资源,可以直接从池中获取,无需动态分配.
* 当服务器处理完一个客户连接后,可以把相关的资源放回池中,无需执行系统调用释放资源.
* 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。

#### ****select/poll/epoll****

* 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通过线性表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**

### HTTP报文格式

HTTP请求报文由请求行（request line）、请求头部（header）、空行和请求数据四个部分组成。

其中，请求分为两种，GET和POST，具体的：

GET方法请求一个指定资源的表示形式，使用GET的请求应该只被用于获取数据

#### POST

POST方法用于将实体提交到指定的资源，通常导致在服务器上的状态变化或**副作用**

**日志**，由服务器自动创建，并记录运行状态，错误信息，访问数据的文件。

**同步日志**，日志写入函数与工作线程串行执行，由于涉及到I/O操作，当单条日志比较大的时候，同步模式会阻塞整个处理流程，服务器所能处理的并发能力将有所下降，尤其是在峰值的时候，写日志可能成为系统的瓶颈。

**生产者-消费者模型**，并发编程中的经典模型。以多线程为例，为了实现线程间数据同步，生产者线程与消费者线程共享一个缓冲区，其中生产者线程往缓冲区中push消息，消费者线程从缓冲区中pop消息。

**阻塞队列**，将生产者-消费者模型进行封装，使用循环数组实现队列，作为两者共享的缓冲区。

**异步日志**，将所写的日志内容先存入阻塞队列，写线程从阻塞队列中取出内容，写入日志。

**单例模式**，最简单也是被问到最多的设计模式之一，保证一个类只创建一个实例，同时提供全局访问的方法。

### 整体概述