## 网络服务器设计原则

使用socket进行网络通信，需要做到高并发、稳定；

### Q1.socket网络通信是什么？

A1.在单机时代，我们需要的通信是进程(线程)与进程(线程)之间的通信，每一个进程的标识可以是一个进程ID号，可以通过Pipe和FIFO等实现数据通信，在网络时代需要的是各个不同主机间的信息传递和共享，我们需要一个东西来描述不同的主机上的某一个进程，socket的返回值是一个文件描述符，不仅仅可以用于描述一个个文件，也可以是某一个主机上的某一个端口；

socket网络应用关联的是这样一对值[host01:port01, host02:port02]，每一个端口在操作系统里面都会有分配数据缓冲区，当数据从host01:port01出来，经过网络传输到达host02的网卡的时候，网卡驱动程序会将数据根据TCP/IP协议栈进行拆解包，由操作系统内核层为新到来的数据进行缓冲和推送，API里面提供了一些设置参数，包括超时时间、QoS、设置缓冲区大小、是否禁用缓冲区、是否启用Nagle算法(根据数据多少进行发送的一种算法)等；

### Q2.socket的应用特性？API特性？

A2.根据不同的应用场景，socket可以分为阻塞IO和非阻塞IO；

阻塞IO：属于同步IO，在socket链接建立起来之后，在我们调用send()和recv()时，在这些操作完成之前，所有的发送和接收都是阻塞的，如果对方一直不发送数据过来，recv()没有设置超时的话，整个线程/进程都将阻塞，这种情况下的收发效率是很慢的；

非阻塞IO：send()和recv()操作不会阻塞，这种操作数据有可能不会及时被内核处理，还停留在内核给的数据缓冲区里，如果我们需要知道某个端口上数据的处理情况，需要借助操作系统提供的select机制，这个函数可以对socket描述符对应的端口进行扫描，将数据的处理情况，通过引用参数传出，函数声明如下：

int select(int maxfdp,fd\_set \*readfds,fd\_set \*writefds,fd\_set \*errorfds,struct timeval \*timeout);

运用select进行操作的时候，需要使用一个独立的线程不停的进行轮询；这种机制下面最常使用的是反应堆模式Reactor处理，即事件注册和响应，当某个我感兴趣事件发生的时候，我会收到通知然后去处理这个事件；

### Q3.操作系统对socket支持的区别？

A3.基本上每一种操作系统对异步IO都提供select支持，但是select在大量并发(成千上万个客户同时连接并发请求)的情况下，表现效果不是很理想，因为是单线程，所以CPU的使用率在大并发下面会迅速上升；

基于以上的考虑，每一个操作系统又都根据自身的内核结构，提供了更高效的异步IO机制，如windows IOCP, linux epoll, MacOS kqueue等；

### Q4.服务器如何充分利用socket特性？

A4.服务器需要大的并发量(1.需要使用非阻塞IO; 2.根据服务器运行的平台选择对应的机制，如windows IOCP，linux下面当然epoll是最好的选择; 3.合理运用多线程)

### Q5.服务器设计中需要考虑的问题？(单机考量)

#### 1.网络并发模型选择

结合平台特性，选取最适合的网络并发模型

#### 2.服务器的应用场景

1).延时：

不同的服务器场景对服务器处理的延时和速度有不同的要求，如棋牌类服务器的延迟可以是5s左右，而一些实时性要求较高的情况下，必须是在1s以内，如股票交易后台，甚至只能容忍ms级的延时；

2).客户端数量、请求数据的大小和频率：

这一点考量很重要，如果同时在线的客户端数量上万，那么对请求数据的大小可以做相应限制，4K或者更小，否则会很耗内存，有得必有失，服务器的设计也是如此，没有最好的服务器，只有最优的服务器；

#### 3.TCP模式下的粘包处理

如果对每个socket都是采用一问一答单线程的方式的话，可以不用考虑(因为在同一个socket对上面只会同时存在一个请求)，如果是对同一个socket可以同时发送多个请求的话，那么就需要做粘包处理了，粘包处理会增加一点服务器处理的复杂度;每一个完整的请求需要做封装和解析；

非法包的处理，如果接收到非法包的话，直接断开与这个客户端的通信，不接收来自它的任何数据，判断方式是否以合法的包头开始，大小是否超过了约定的包的大小；

如果封装中有合法消息本身校验的话，最好将消息的校验放放在处理消息业务的地方校验，如果数据内容本身校验出错，可以返回给客户端;

#### 4.线程并发处理

服务器需要最快的响应客户端的请求，一种简单的方法，每一个请求进来都开启一个线程做处理，这种方式在客户端数量小的时候效果最好，当客户端的数量成百上千的增加的时候，这个时候的消耗不仅仅是线程资源分配的消耗，而且线程间的上下文切换的消耗基本上服务器就不能干什么事儿了。

基于上述情况，最好的方式使用队列，有限的线程都关联需要处理的数据的队列，典型的如请求处理线程+队列、请求响应线程+队列，至于数量也是需要根据服务器的应用场景决定的。

#### 5.稳定性保证

1. 服务器基本上都是7x24小时运转的，不能允许一点内存泄露；
2. 服务器规格限制，不能无限制的接受客户端的链接请求，需要设置客户端链接上限，超过链接上线采用集群协商机制解决，直接关闭客户端的链接，客户端重新请求别的服务器；
3. 服务器对客户端的管理，每一个客户端都有一个请求周期，如果一个客户端链接在周期内没有发生任何数据请求的话，服务器可以强制关闭客户端链接；

#### 6.运行产生的内存碎片

使用内存池，减少碎片的产生

### Q6. Windows下的 IOCP工作流程

#### 1.操作流程图

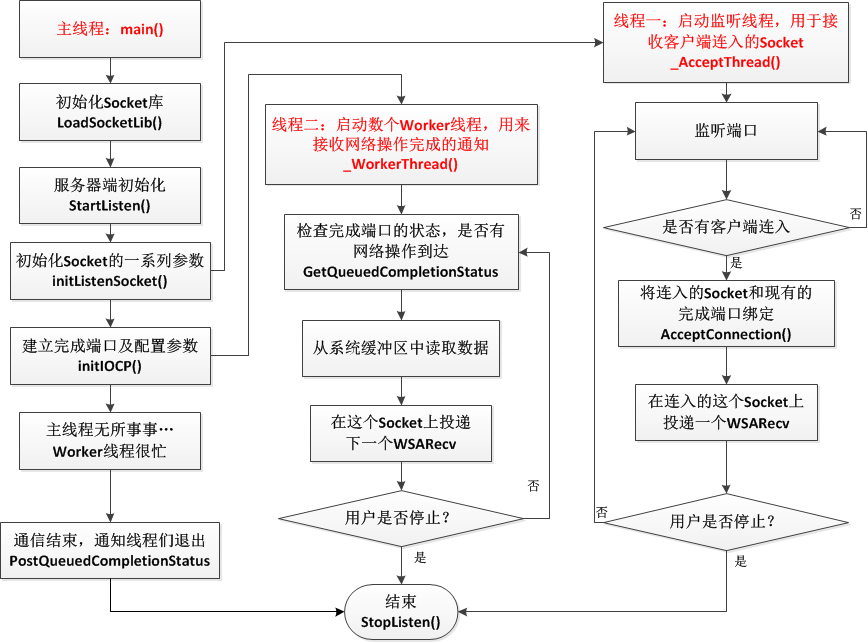


Figure 1 使用Accept进行的IOCP流程

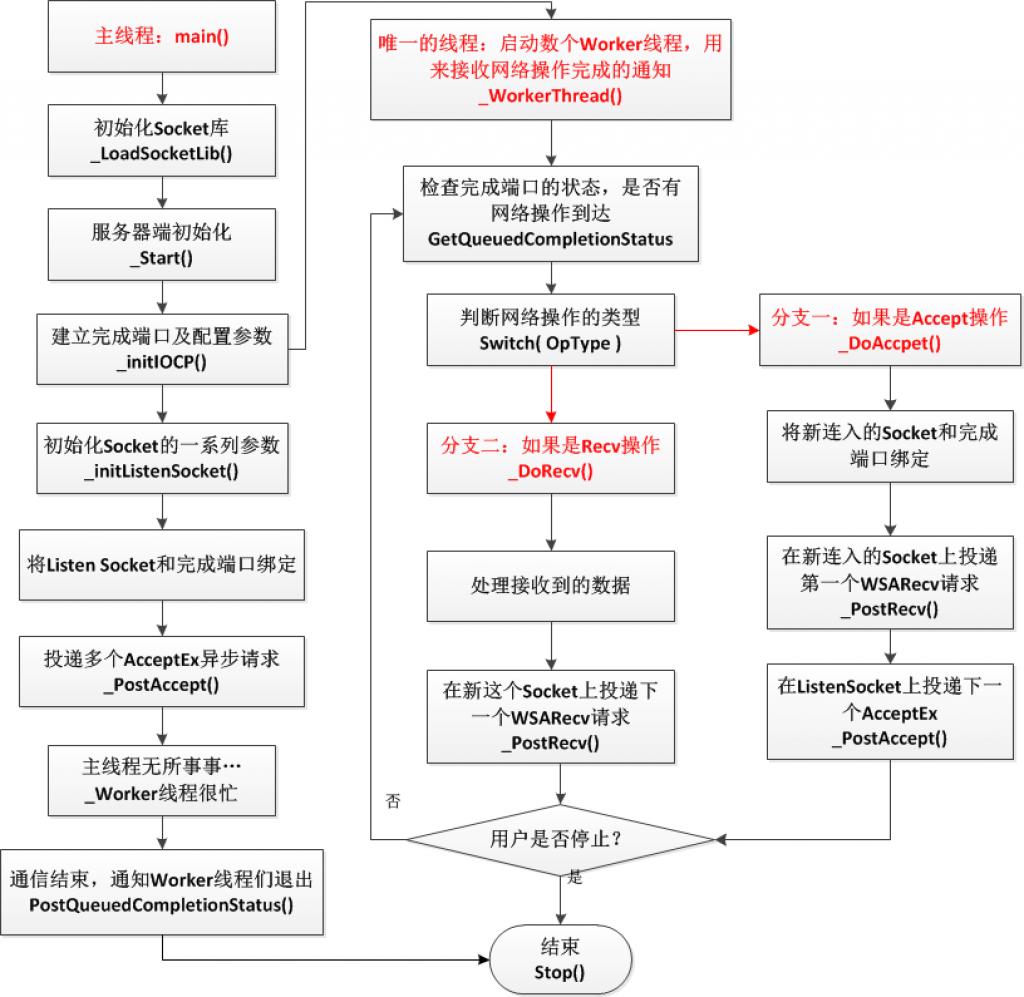


Figure 2 使用AcceptEx的IOCP流程

两个图中最大的相同点是什么？是的，最大的相同点就是主线程无所事事，闲得蛋疼……

         为什么呢？因为我们使用了异步的通信机制，这些琐碎重复的事情完全没有必要交给主线程自己来做了，只用在初始化的时候和Worker线程交待好就可以了，用一句话来形容就是，主线程永远也体会不到Worker线程有多忙，而Worker线程也永远体会不到主线程在初始化建立起这个通信框架的时候操了多少的心……

         图a中是由 \_AcceptThread()负责接入连接，并把连入的Socket和完成端口绑定，另外的多个\_WorkerThread()就负责监控完成端口上的情况，一旦有情况了，就取出来处理，如果CPU有多核的话，就可以多个线程轮着来处理完成端口上的信息，很明显效率就提高了。

         图b中最明显的区别，也就是AcceptEx和传统的accept之间最大的区别，就是取消了阻塞方式的accept调用，也就是说，AcceptEx也是通过完成端口来异步完成的，所以就取消了专门用于accept连接的线程，用了完成端口来进行异步的AcceptEx调用；然后在检索完成端口队列的Worker函数中，根据用户投递的完成操作的类型，再来找出其中的投递的Accept请求，加以对应的处理。

#### 2.Accept和AcceptEx的区别

首先，我可以很明确的告诉各位，如果短时间内客户端的并发连接请求不是特别多的话，用accept和AcceptEx在性能上来讲是没什么区别的。

        按照我们目前主流的PC来讲，如果客户端只进行连接请求，而什么都不做的话，我们的Server只能接收大约3万-4万个左右的并发连接，然后客户端其余的连入请求就只能收到WSAENOBUFS (10055)了，因为系统来不及为新连入的客户端准备资源了。

        需要准备什么资源？当然是准备Socket了……虽然我们创建Socket只用一行SOCKET s= socket(…) 这么一行的代码就OK了，但是系统内部建立一个Socket是相当耗费资源的，因为Winsock2是分层的机构体系，创建一个Socket需要到多个Provider之间进行处理，最终形成一个可用的套接字。总之，系统创建一个Socket的开销是相当高的，所以用accept的话，系统可能来不及为更多的并发客户端现场准备Socket了。

        而AcceptEx比Accept又强大在哪里呢？是有三点：

         (1) 这个好处是最关键的，是因为AcceptEx是在客户端连入之前，就把客户端的Socket建立好了，也就是说，AcceptEx是先建立的Socket，然后才发出的AcceptEx调用，也就是说，在进行客户端的通信之前，无论是否有客户端连入，Socket都是提前建立好了；而不需要像accept是在客户端连入了之后，再现场去花费时间建立Socket。如果各位不清楚是如何实现的，请看后面的实现部分。

         (2) 相比accept只能阻塞方式建立一个连入的入口，对于大量的并发客户端来讲，入口实在是有点挤；而AcceptEx可以同时在完成端口上投递多个请求，这样有客户端连入的时候，就非常优雅而且从容不迫的边喝茶边处理连入请求了。

         (3) AcceptEx还有一个非常体贴的优点，就是在投递AcceptEx的时候，我们还可以顺便在AcceptEx的同时，收取客户端发来的第一组数据，这个是同时进行的，也就是说，在我们收到AcceptEx完成的通知的时候，我们就已经把这第一组数据接完毕了；但是这也意味着，如果客户端只是连入但是不发送数据的话，我们就不会收到这个AcceptEx完成的通知……这个我们在后面的实现部分，也可以详细看到。

         最后，各位要有一个心里准备，相比accept，异步的AcceptEx使用起来要麻烦得多……

总结：

一个好的应用系统，少不了系统中客户端和服务器的协调工作