1. **任何1个网络框架在逻辑上要解决哪几个问题？**

* server socket的绑定（bind ip,port）+ 启动监听动作 (这是为了让client可以连上来)
* 连接上来的socket，启动IO监听动作，按需进行IO读写操作
* 针对读到的数据，执行业务逻辑操作，业务逻辑操作得到的数据，执行写操作，通过步骤2写回去

任何1个网络程序都要解决这3个问题！！！比如mina netty thrift redis kafka无不例外

**2什么是 可Accept ？**

内核为每个服务端socket维护一个连接队列，容量由backlog参数决定，当这个连接队列不为空时，内核会通知应用层满足OP\_ACCEPT事件，应用层执行accept方法就可以拿到请求的客户端socket对象

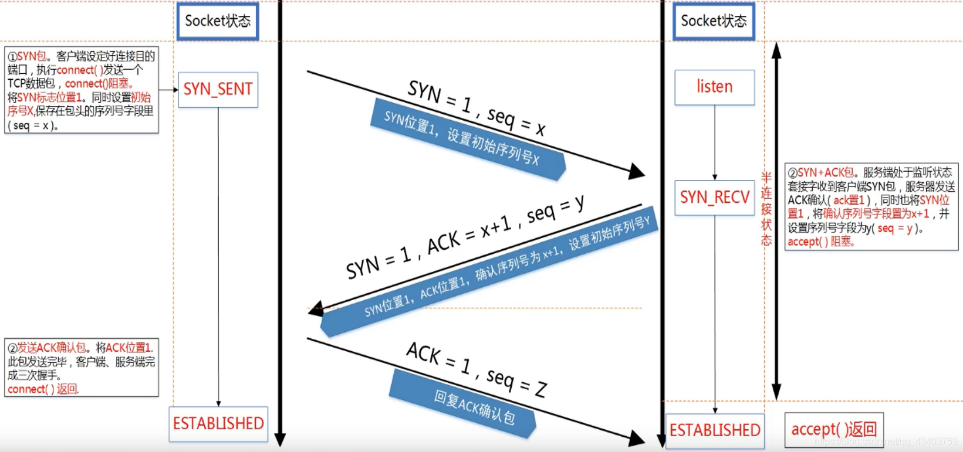
**3什么是backlog? 连接队列到底是个啥？**

内核会为 listen 状态的 socket 维护两个[队列](https://so.csdn.net/so/search?q=%E9%98%9F%E5%88%97&spm=1001.2101.3001.7020)：不完全连接请求队列（SYN\_RECV）和

等待 accept() 建立的队列（ESTABLISHED），Linux 内核2.2之后，backlog 参数的行为发生了改变，现在它指等待 accept() 的完全建立的 socket 的队列长度（已经完成三次握手），而不是不完全连接请求的数量。

不完全连接请求队列长度由/proc/sys/net/ipv4/tcp\_max\_syn\_backlog指定：





**4网络框架在接受到1个socket连接后，处理socket，一直到响应最终写回client,结合以前学过的TCPIP知识，整个数据处理流程是什么样的，谈谈你的理解**

1. 应用层分发socket给IO线程组
2. IO线程组将此新socket纳入某个线程的监听可读可写事件的socket集合(每个线程托管1部分socket)
3. 对于某个线程来说，获得其监管下socket的可读/可写的socket子集，然后遍历子集的每一个socket
4. 可读时，执行读操作，应用层长度完整后，抛给后面的业务线程池
5. 业务逻辑处理结果再返回给之前的线程
6. IO线程监听可写事件，满足条件时，写(复制)到内核态发送缓冲区
7. 剩下的交给内核的网络协议栈，想一下以前学的滑动窗口协议
8. 不停循环

**5耗时长的业务逻辑为什么不能放在IO线程？**

常规情况下，当一个进程处理十几万个socket后,通过轮询,任何一个IO线程都会处理很多比如1万个socket,如果业务逻辑操作放在IO线程，因为业务逻辑很耗时，这个线程就没时间去处理别的socket的IO处理了，造成其他socket即使可读可写，也只能干等！

**6粘包拆包问题如何产生？如何解决？Netty是如何解决这类问题的？**

操作系统在发送TCP数据的时候，底层会有一个缓冲区，例如1024个字节大小，如果一次请求发送的数据量比较小，没达到缓冲区大小，TCP则会将多个请求合并为同一个请求进行发送，这就形成了粘包问题；如果一次请求发送的数据量比较大，超过了缓冲区大小，TCP就会将其拆分为多次发送，这就是拆包，也就是将一个大的包拆分为多个小包进行发送。

常见解决方案:

1. 客户端在发送数据包的时候，每个包都固定长度，比如1024个字节大小，如果客户端发送的数据长度不足1024个字节，则通过补充空格的方式补全到指定长度；
2. 客户端在每个包的末尾使用固定的分隔符，例如\r\n，如果一个包被拆分了，则等待下一个包发送过来之后找到其中的\r\n，然后对其拆分后的头部部分与前一个包的剩余部分进行合并，这样就得到了一个完整的包；
3. 将消息分为头部和消息体，在头部中保存有当前整个消息的长度，只有在读取到足够长度的消息之后才算是读到了一个完整的消息；
4. 通过自定义协议进行粘包和拆包的处理。

Netty 提供相关解码器类解决

**7市面常用几种软件的网络模型有细节区别，谈谈你的理解**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **框架** | **监听（accept）** | **IO事件监听** | **IO读写操作【应用层完整报文提取+发送应用层响应数据】** | **业务逻辑** |
| mina | 监听线程池 | IO线程池 | | 业务线程池【需要用户自己创建】 |
| Netty(比如dubbo) | 监听线程池 | IO线程池 | | 业务线程池【需要用户自己创建】 |
| thrift | 监听线程池 | IO线程池 | | 业务线程池【框架自己创建好了，对业务开发透明，开发填充业务逻辑代码即可】 |
| Kafka | 监听线程池 | IO线程池 | | 业务线程池【Kafka自己创建，运行自己的代码逻辑-读写文件】 |
| Tomcat(比如springboot) | 监听线程池 | IO监听线程池 | 业务线程池【框架自己创建好了，对业务开发透明】 | |
| redis | 1个线程打天下【早期一个线程，6.X多个线程】备注：响应客户请求的主体线程是1个，整个进程內还有别的线程（比如持久化相关线程） | | | |