Socket 是操作系统提供的网络编程接口，封装了对于 TCP/IP 协议栈的支持，用于进程间的通信，当有连接接入主机后，操作系统自动为其分配一个套接字，套接字绑定一个 IP 和 Port。通过 Socket 接口可以获取 TCP 连接的输入流和输出流，并进行读取和写入操作。

一般客户端使用单线程模型，有读数据时启动线程，写数据时启动线程。服务端用多线程模型，一个线程负责接收 TCP 请求，接收到一个请求后开启线程处理读写。UDP 数据报长度确定，只需要写入固定的缓存和读取固定的缓存空间即可。DatagramPacket 包装一个 UDP 数据报，DatagramSocket 发送。

IO 模型：（1）阻塞 IO，线程会阻塞在系统调用上，并且等待数据准备就绪以后返回。（2）非阻塞 IO，通过自旋忙等待的方式不断询问缓冲区是否准备就绪，不阻塞在系统调用上，避免线程阻塞的开销。（3）IO 多路复用，IO 多路复用器管理 Socket，每个 Socket 是一个文件描述符，操作系统维护 Socket 和其连接状态，一般状态分为可连接，可读和可写等。当用户程序接收到 Socket 请求，多路复用器进行监控并且可以通知用户程序，或者用户程序轮询请求，获取就绪的 Socket。需要一个线程进行轮询，多个线程处理就绪请求。

Linux 操作系统对于多路复用提供支持，有 select、poll、epoll 支持多路复用的 API。NIO 的非阻塞模型，采用 IO 多路复用方式，基于 epoll 实现。（1）select 方式主要是使用数组来存储 Socket 描述符，系统将发生时间的描述符做标记，然后 IO 复用器在轮询描述符数组时，便知道哪些请求是就绪的。缺点是数组的长度只能到 1024，并且需要不断在内核空间和用户空间之间拷贝数组。（2）poll 方式不采用数组存储文件描述符，使用独立的数据结构描述，使用 ID 来表示描述符，能支持更多的请求数量，缺点是轮询的效率低，需要拷贝数据。（3）epoll 函数会在内核空间开辟一个特殊的数据结构，红黑树，树节点中存放的是一个 Socket 描述符以及用户感兴趣的时间类型，并维护一个链表，链表存储已经就绪的 Socket 描述符节点。epoll_create 函数会执行创建红黑树的操作，epoll_ctl 函数将 Socket 和感兴趣的事件注册到红黑树中，epoll_wait 函数会等待内核空间发来的链表，执行 IO 请求。epoll 的水平触发是指，如果用户程序没有执行就绪链表中的任务，epoll 会不断通知程序，边缘触发只通知程序一次。

Netty 是一个基于事件驱动的网络编程框架。可用其实现 RPC 和 HTTP 服务。两种线程模型：reactor 和 proactor。reactor 是 Netty 采用的迷行，使用一个 acceptor 线程接收连接请求，然后开启一个线程组 reactor thread pool。server 在 endpoint 上注册一系列回调方法，然后接收 socket 请求后交给底层的 selector 进行管理，当 selector 对应的时间响应之后会通知用户进程，然后 reactor 工作进程会执行接下来的 IO 请求，执行操作写在回调处理器中。Netty 支持三种 reactor 模型。（1）reactor 单线程模型，指的是所有的 IO 操作都在同一个 NIO 线程上完成，对于小容量应用场景，使用单线程模型。（2）reactor 多线程模型，有一组 NIO 线程处理 IO 操作，用于高并发，大业务量操作。（3）主从 reactor 多线程模型，服务端用于接受客户端连接的不再是一个单独的 NIO 线程，而是一个独立的 NIO 线程池。利用主从 NIO

线程模型，可以解决一个服务端监听线程无法有效处理所有客户端连接的性能不足问题。proactor 模型为异步非阻塞模型，当 acceptor 接收到请求后，直接提交异步 IO 请求给 Linux 内核，内核完成 IO 请求后会回写消息到 proactor 提供的消息队列中，当工作线程查看到 IO 请求完成，则会继续剩下的工作，通过回调处理器来处理。