I/O分类：

第一种方法就是最传统的多进程并发模型(每进来一个新的I/O流会分配一个新的进程管理。)

第二种方法就是I/O多路复用(单个线程，通过记录跟踪每个I/O流(sock)的状态，来同时管理多个I/O流。)

多进程并发模型：Apache

I/O 多路复用:Nginx(有多少个core 多少个worker-process**进程**)

I/O 多路复用：select,poll,epoll

同步/异步：是消息通信机制

同步：发出调用后没有结果不返回

异步：发出调用后就返回

阻塞和非阻塞：程序在等待调用结果（消息，返回值）时的状态.

阻塞调用是指调用结果返回之前，当前线程会被挂起。

调用线程只有在得到结果之后才会返回。 非阻塞调用指在不能立刻得到结果之前，该调用不会阻塞当前线程。

同步异步指的是通信过程，而阻塞不阻塞指的是调用通信的线程。

Nio:同步非阻塞(epoll):**线程不挂起，以其中某种方式获取（如轮询）获得资源。**

**（core：不挂起，不切换线程消耗资源）**

Aio：异步非阻塞

Bio：同步阻塞 selct/poll/

Fd：FileDescriptor是文件描述符，一个socket句柄，可以看做是一个文件，在socket上收发数据，相当于对一个文件进行读写，所以一个socket句柄，通常也用表示文件句柄的fd来表示

实现原理

Select:

所有的文件句柄初始化放到一个数组里，

Select遍历看是否可读，可读则都，不可读挂起等有io时间再继续轮询尝试。

Poll：

与select类似，poll的实现和select非常相似，只是描述fd集合的方式不同，poll使用pollfd结构而不是select的fd\_set结构，poll没有最大文件描述符数量的限制

Epoll

1. Epoll\_create建立线程内核缓冲区（红黑树，链表）
2. 将文件句柄放到红黑树上，并注册句柄的回调函数
3. 当句柄（读/写）可读可写则将句柄放到就绪列表
4. Epoll\_wait监控就绪列表（同样需要轮询，本质非异步）

0 epollfd = epoll\_create(FDSIZE);

1 //添加监听描述符事件

2 add\_event(epollfd,listenfd,EPOLLIN);

3 for ( ; ; )

4 {

5 //获取已经准备好的描述符事件

6 ret = epoll\_wait(epollfd,events,EPOLLEVENTS,-1);

7 handle\_events(epollfd,events,ret,listenfd,buf);

8 }

9 close(epollfd);

Nio/aio

nio需要自己去查看是否返回结果，epoll

**java.nio**

NIO的类库和API繁杂，使用麻烦，你需要熟练掌握Selector、ServerSocketChannel、SocketChannel、ByteBuffer等

也可参考rpc章节netty相关内容