# IO流

## 同步和异步

所谓同步，指的是协同步调。既然叫协同，所以至少要有2个以上的事物存在。协同的结果就是： **多个事物不能同时进行，必须一个一个的来，上一个事物结束后，下一个事物才开始。 那当一个事物正在进行时，其它事物都在干嘛呢？**严格来讲这个并没有要求，但一般都是处于一种“等待”的状态，因为通常后面事物的正常进行都需要依赖前面事务的结构或前面事务正在使用的资源。因此，可以认为，同步更希望关注的是从宏观整体来看，多个事务是一种逐个逐个串行化关系，绝对不会出现交叉的情况。所以，自然也不太会去关注某个瞬间某个事务是处于一个什么状态。

把这个理论应用的出神入化的非“排队”莫属。凡是在资源少需求多的场景下都会用到排队。

比如排队买火车票这件事： 其实售票大厅更在意的是旅客一个一个的到窗口去买票，因为一次只能卖一张票。

即使大家一窝蜂的都围上去，还是一次只能卖一张票，何必呢？挤在一起又不安全。

只是有些人素质太差，非要往上挤，售票大厅迫不得已，采用排队这种形式来达到自己的目的，即一个一个的买票。

关于同步还需知道两个小的点： 一是范围，并不需要在全局范围内都去同步，只需要在某些关键的点执行同步即可。 比如食堂只有一个卖饭窗口，肯定是同步的，一个人买完，下一个人再买。但吃饭的时候也是一个人吃完，下一个人才开始吃吗？当然不是啦。 二是粒度，并不是只有大粒度的事物才有同步，小粒度的事物也有同步。

## 阻塞和非阻塞

所谓阻塞，指的就是阻碍堵塞。它的本意可以理解为遇到了障碍而造成的动弹不已。

所谓非阻塞，自然是和阻塞相对，可以理解为没有遇到阻碍而畅通无阻。对这两个词最好的诠释就是，当今中国一大交通难题，堵车： 汽车可以正常通行时，就是非阻塞。一旦堵上了，全部趴窝，一动不动，就是阻塞。因此阻塞关注的是不能动，非阻塞关注的是可以动。

不能动的结果就是只能等待，可以动的结果就是继续前行。 因此和阻塞搭配的词一定是等待，和非阻塞搭配的词一定是进行。

回到程序里，阻塞同样意味着停下来等待，非阻塞表明可以继续向下执行。

## 阻塞和等待

等待只是阻塞的一个副作用而已，表明随着时间的流逝，没有任何意义的事务发生或者进行。阻塞的真正含义是你关心的事务由于某些原因无法进行，因此让你等待。但没必要干等，你可以做一些其它无关的事情，因为这并不影响你对相关事务的等待在堵车时，你可以干等。也可以玩手机、和别人聊天，或者打牌、甚至先去吃饭都行。因为这些事物并不影响你对堵车的等待。不过你的车必须呆在原地。

在计算机里，是没有人这么灵活的，一般在阻塞时，选在干等，因为这最容易实现，只需要挂起线程，让出CPU即可。在条件满足时，会重新调度该线程。

## 两两组合

所谓同步/异步，关注的是能不能同时开工。

所谓阻塞/非阻塞，关注的是能不能动。

通过推理进行组合：

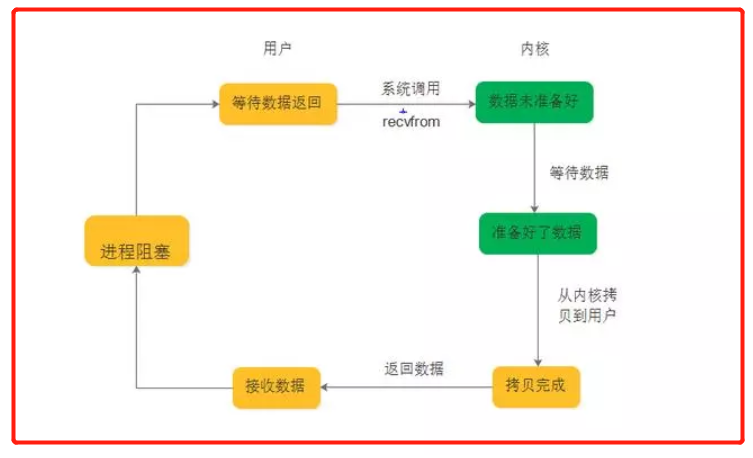
* 同步阻塞：就是不能同时开工，也不能动。只有一条小道，一次只能过一辆车，还TMD的堵上了
* 异步阻塞：就是可以同时开工，但却不能动。有很多条车道，但是却全部堵车了，不能动
* 同步非阻塞：不能同时开工，可以动。只有一条车道，而车辆可以正常通行
* 异步非阻塞：可以同时开工，可以东。很多条车道，且全部都可以通行

回到程序里：把它们和线程关联起来：

* 线程同步阻塞，相当于一个线程在等待
* 线程异步阻塞，相当于多个线程都在等待
* 线程同步非阻塞，相当于一个线程在正常运行
* 线程异步非阻塞，相当于多个线程在正常运行

## I/O流

IO指的就是读入/写出数据的过程，和**等待**读入/写出数据的过程。一旦拿到数据后就变成了数据操作过程了，就是不I/O。拿网络IO来说，等待的过程就是数据从网络到网卡再到内核空间。读写的过程就是内核空间和用户空间相互拷贝。所以IO就包括两个过程，一个是等待数据的过程，一个是读写（拷贝）数据的过程。而且还要明白，一定**不**能包括操作数据的过程。



## 阻塞IO和非阻塞IO

应用程序都是运行在用户空间的，所以它们能操作的数据也都在用户空间。按照这样子来理解，只有数据还没到达用户空间，用户线程就操作不了。

如果此时用户线程已经参与，那它一定会被阻塞在I/O上。这就是常说的阻塞IO。用户线程被阻塞在等待数据上或拷贝数据上

非阻塞I/O就是用户线程不参与以上两个过程，即等待数据到了用户空间后，才去通知用户线程，一上来就可以直接操作数据了。

用户线程没有因为IO的事情出现阻塞，这就是常说的非阻塞IO。

## 同步IO和同步阻塞IO

按照上文中对同步的理解，同步IO是指发起IO请求后，必须拿到IO的数据才可以继续执行。按照程序的表现形式又分为两种：在等待数据的过程中，和拷贝数据的过程中，线程都在阻塞，这就是同步阻塞IO。在等待数据的过程中，线程采用死循环式轮询，在拷贝数据的过程中，线程在阻塞，这其实还是同步阻塞IO。

网上很多文章把第二种归为同步非阻塞IO，这肯定是**错误**的，它一定是阻塞IO，因为拷贝数据的过程，线程是阻塞的。 严格来讲，在IO的概念上，同步和非阻塞是不可能搭配的，因为它们是一对相悖的概念。 同步IO意味着必须拿到IO的数据，才可以继续执行。因为后续操作依赖IO数据，所以它必须是阻塞的。

非阻塞IO意味着发起IO请求后，可以继续往下执行。说明后续的执行不依赖于IO数据，所以它肯定是不同步的。因此，在IO上，同步和非阻塞是互斥的，所以不存在同步非阻塞IO。但同步非阻塞是存在的，那不叫IO，叫操作数据了。

## 异步阻塞IO和异步非阻塞IO

按照上文中对异步的理解，异步IO是指发起IO请求后，不用拿到IO的数据就可以继续执行。

用户线程的继续执行，和操作系统准备IO数据的过程是同时进行的，因此才叫做异步IO。 按照IO数据的两个过程，又可以分为两种： 在等待数据的过程中，用户线程继续执行，在拷贝数据的过程中，线程在阻塞，这就是异步阻塞IO。

在等待数据的过程中，和拷贝数据的过程中，用户线程都在继续执行，这就是异步非阻塞IO。

第一种情况是，用户线程没有参与数据等待的过程，所以它是异步的。但用户线程参与了数据拷贝的过程，所以它又是阻塞的。合起来就是异步阻塞IO。

第二种情况是，用户线程既没有参与等待过程也没有参与拷贝过程，所以它是异步的。当它接到通知时，数据已经准备好了，它没有因为IO数据而阻塞过，所以它又是非阻塞的。合起来就是异步非阻塞IO。