## 1、线程与进程

1. 进程是一个实体。每一个进程都有它自己的地址空间，一般情况下，包括文本区域（text region）、数据区域（data region）和堆栈（stack region）。文本区域存储处理器执行的代码；数据区域存储变量和进程执行期间使用的动态分配的内存；堆栈区域存储着活动过程调用的指令和本地变量。

2. 一个标准的线程由线程ID，当前指令指针(PC），寄存器集合和堆栈组成。另外，线程是进程中的一个实体，是被系统独立调度和分派的基本单位，线程自己不拥有系统资源，只拥有一点儿在运行中必不可少的资源，但它可与同属一个进程的其它线程共享进程所拥有的全部资源。

3. 区别不同   
a,地址空间:进程内的一个执行单元;进程至少有一个线程;它们共享进程的地址空间;而进程有自己独立的地址空间;   
b,资源拥有:进程是资源分配和拥有的单位,同一个进程内的线程共享进程的资   
c,线程是处理器调度的基本单位,但进程不是.   
d,二者均可并发执行.

## 2、 守护线程

在Java中有两类线程：用户线程 (User Thread)、守护线程 (Daemon Thread)。   
守护线程和用户线程的区别在于：守护线程依赖于创建它的线程，而用户线程则不依赖。举个简单的例子：如果在main线程中创建了一个守护线程，当main方法运行完毕之后，守护线程也会随着消亡。而用户线程则不会，用户线程会一直运行直到其运行完毕。在JVM中，像垃圾收集器线程就是守护线程。

## 3、java thread状态

1. NEW 状态是指线程刚创建, 尚未启动

2. RUNNABLE 状态是线程正在正常运行中, 当然可能会有某种耗时计算/IO等待的操作/CPU时间片切换等, 这个状态下发生的等待一般是其他系统资源, 而不是锁, Sleep等

3. BLOCKED 这个状态下, 是在多个线程有同步操作的场景, 比如正在等待另一个线程的synchronized 块的执行释放, 也就是这里是线程在等待进入临界区

4. WAITING 这个状态下是指线程拥有了某个锁之后, 调用了他的wait方法, 等待其他线程/锁拥有者调用 notify / notifyAll 一遍该线程可以继续下一步操作, 这里要区分 BLOCKED 和 WATING 的区别, 一个是在临界点外面等待进入, 一个是在理解点里面wait等待别人notify, 线程调用了join方法 join了另外的线程的时候, 也会进入WAITING状态, 等待被他join的线程执行结束

5. TIMED\_WAITING 这个状态就是有限的(时间限制)的WAITING, 一般出现在调用wait(long), join(long)等情况下, 另外一个线程sleep后, 也会进入TIMED\_WAITING状态

6. TERMINATED 这个状态下表示 该线程的run方法已经执行完毕了, 基本上就等于死亡了(当时如果线程被持久持有, 可能不会被回收)

## 4、请说出与线程同步以及线程调度相关的方法。

1. wait()：使一个线程处于等待（阻塞）状态，并且释放所持有的对象的锁；

2. sleep()：使一个正在运行的线程处于睡眠状态，是一个静态方法，调用此方法要处理InterruptedException异常；

3. notify()：唤醒一个处于等待状态的线程，当然在调用此方法的时候，并不能确切的唤醒某一个等待状态的线程，而是由JVM确定唤醒哪个线程，而且与优先级无关；

4. notityAll()：唤醒所有处于等待状态的线程，该方法并不是将对象的锁给所有线程，而是让它们竞争，只有获得锁的线程才能进入就绪状态；

## 5、进程调度算法

实时系统：FIFO(First Input First Output，先进先出算法)，SJF(Shortest Job First，最短作业优先算法)，SRTF(Shortest Remaining Time First，最短剩余时间优先算法）。   
交互式系统：RR(Round Robin，时间片轮转算法)，HPF(Highest Priority First，最高优先级算法)，多级队列，最短进程优先，保证调度，彩票调度，公平分享调度。

## 6、wait()和sleep()的区别

1. sleep来自Thread类，和wait来自Object类

2.调用sleep()方法的过程中，线程不会释放对象锁。而 调用 wait 方法线程会释放对象锁

3.sleep睡眠后不出让系统资源，wait让出系统资源其他线程可以占用CPU

4.sleep(milliseconds)需要指定一个睡眠时间，时间一到会自动唤醒

## 7、ThreadLocal,以及死锁分析

threadLocal为每个线程维护一个本地变量。   
采用空间换时间，它用于线程间的数据隔离，为每一个使用该变量的线程提供一个副本，每个线程都可以独立地改变自己的副本，而不会和其他线程的副本冲突。   
ThreadLocal类中维护一个Map，用于存储每一个线程的变量副本，Map中元素的键为线程对象，而值为对应线程的变量副本。   
[彻底理解ThreadLocal](https://www.cnblogs.com/xzwblog/p/7227509.html" \t "_blank)

## 8、Synchronized 与Lock

1. ReentrantLock 拥有Synchronized相同的并发性和内存语义，此外还多了 锁投票，定时锁等候和中断锁等候   
线程A和B都要获取对象O的锁定，假设A获取了对象O锁，B将等待A释放对O的锁定，   
如果使用 synchronized ，如果A不释放，B将一直等下去，不能被中断   
如果 使用ReentrantLock，如果A不释放，可以使B在等待了足够长的时间以后，中断等待，而干别的事情

2. ReentrantLock获取锁定与三种方式：   
a) lock(), 如果获取了锁立即返回，如果别的线程持有锁，当前线程则一直处于休眠状态，直到获取锁   
b) tryLock(), 如果获取了锁立即返回true，如果别的线程正持有锁，立即返回false；   
c)tryLock(long timeout,TimeUnit unit)， 如果获取了锁定立即返回true，如果别的线程正持有锁，会等待参数给定的时间，在等待的过程中，如果获取了锁定，就返回true，如果等待超时，返回false；   
d) lockInterruptibly:如果获取了锁定立即返回，如果没有获取锁定，当前线程处于休眠状态，直到或者锁定，或者当前线程被别的线程中断

总体的结论先摆出来：

synchronized：   
在资源竞争不是很激烈的情况下，偶尔会有同步的情形下，synchronized是很合适的。原因在于，编译程序通常会尽可能的进行优化synchronized，另外可读性非常好，不管用没用过5.0多线程包的程序员都能理解。   
ReentrantLock:   
ReentrantLock提供了多样化的同步，比如有时间限制的同步，可以被Interrupt的同步（synchronized的同步是不能Interrupt的）等。在资源竞争不激烈的情形下，性能稍微比synchronized差点点。但是当同步非常激烈的时候，synchronized的性能一下子能下降好几十倍。而ReentrantLock确还能维持常态。

[详解synchronized与Lock的区别与使用](https://blog.csdn.net/u012403290/article/details/64910926?locationNum=11&fps=1" \t "_blank)

## 9、Volatile和Synchronized

Volatile和Synchronized四个不同点：

1. 粒度不同，前者针对变量 ，后者锁对象和类
2. syn阻塞，volatile线程不阻塞
3. syn保证三大特性，volatile不保证原子性
4. syn编译器优化，volatile不优化   
   要使 volatile 变量提供理想的线程安全，必须同时满足下面两个条件：
5. 对变量的写操作不依赖于当前值。
6. 该变量没有包含在具有其他变量的不变式中。

[JAVA多线程之volatile 与 synchronized 的比较](https://www.cnblogs.com/hapjin/p/5492880.html" \t "_blank)

## 10、CAS

CAS是乐观锁技术，当多个线程尝试使用CAS同时更新同一个变量时，只有其中一个线程能更新变量的值，而其它线程都失败，失败的线程并不会被挂起，而是被告知这次竞争中失败，并可以再次尝试。CAS有3个操作数，内存值V，旧的预期值A，要修改的新值B。当且仅当预期值A和内存值V相同时，将内存值V修改为B，否则什么都不做。

## 11、Java中Unsafe类详解

1.通过Unsafe类可以分配内存，可以释放内存；类中提供的3个本地方法allocateMemory、reallocateMemory、freeMemory分别用于分配内存，扩充内存和释放内存，与C语言中的3个方法对应。

2.可以定位对象某字段的内存位置，也可以修改对象的字段值，即使它是私有的；

3.挂起与恢复:将一个线程进行挂起是通过park方法实现的，调用 park后，线程将一直阻塞直到超时或者中断等条件出现。unpark可以终止一个挂起的线程，使其恢复正常。整个并发框架中对线程的挂起操作被封装在 LockSupport类中，LockSupport类中有各种版本pack方法，但最终都调用了Unsafe.park()方法。

4.cas   
[Java中Unsafe类详解](http://blog.csdn.net/bluetjs/article/details/52758095" \t "_blank)

## 12、线程池

线程池的作用：   
在程序启动的时候就创建若干线程来响应处理，它们被称为线程池，里面的线程叫工作线程   
第一：降低资源消耗。通过重复利用已创建的线程降低线程创建和销毁造成的消耗。   
第二：提高响应速度。当任务到达时，任务可以不需要等到线程创建就能立即执行。   
第三：提高线程的可管理性。   
常用线程池：ExecutorService 是主要的实现类，其中常用的有  
Executors.newSingleThreadPool(),newFixedThreadPool(),newcachedTheadPool(),

newScheduledThreadPool()。

## 13、ThreadPoolExecutor

构造方法参数说明

corePoolSize:核心线程数，默认情况下核心线程会一直存活，即使处于闲置状态也不会受存keepAliveTime限制。除非将allowCoreThreadTimeOut设置为true。   
maximumPoolSize:线程池所能容纳的最大线程数。超过这个数的线程将被阻塞。当任务队列为没有设置大小的LinkedBlockingDeque时，这个值无效。   
keepAliveTime:非核心线程的闲置超时时间，超过这个时间就会被回收。   
unit:指定keepAliveTime的单位，如TimeUnit.SECONDS。当将allowCoreThreadTimeOut设置为true时对corePoolSize生效。   
workQueue:线程池中的任务队列.   
常用的有三种队列，SynchronousQueue,LinkedBlockingDeque,ArrayBlockingQueue。

threadFactory:线程工厂，提供创建新线程的功能。ThreadFactory是一个接口，只有一个方法

原理

1. 如果当前池大小 poolSize 小于 corePoolSize ，则创建新线程执行任务。
2. 如果当前池大小 poolSize 大于 corePoolSize ，且等待队列未满，则进入等待队列
3. 如果当前池大小 poolSize 大于 corePoolSize 且小于 maximumPoolSize ，且等待队列已满，则创建新线程执行任务。
4. 如果当前池大小 poolSize 大于 corePoolSize 且大于 maximumPoolSize ，且等待队列已满，则调用拒绝策略来处理该任务。
5. 线程池里的每个线程执行完任务后不会立刻退出，而是会去检查下等待队列里是否还有线程任务需要执行，如果在 keepAliveTime 里等不到新的任务了，那么线程就会退出。

## 13、Executor拒绝策略

1. AbortPolicy:为java线程池默认的阻塞策略，不执行此任务，而且直接抛出一个运行时异常，切记ThreadPoolExecutor.execute需要try   
2. catch，否则程序会直接退出.

3. DiscardPolicy:直接抛弃，任务不执行，空方法

4. DiscardOldestPolicy:从队列里面抛弃head的一个任务，并再次execute 此task。

5. CallerRunsPolicy:在调用execute的线程里面执行此command，会阻塞入

用户自定义拒绝策略:实现RejectedExecutionHandler，并自己定义策略模式

## 14、CachedThreadPool 、 FixedThreadPool、SingleThreadPool

1. newSingleThreadExecutor :创建一个单线程化的线程池，它只会用唯一的工作线程来执行任务， 保证所有任务按照指定顺序(FIFO, LIFO, 优先级)执行   
适用场景：任务少 ，并且不需要并发执行

2. newCachedThreadPool :创建一个可缓存线程池，如果线程池长度超过处理需要，可灵活回收空闲线程，若无可回收，则新建线程.   
线程没有任务要执行时，便处于空闲状态，处于空闲状态的线程并不会被立即销毁（会被缓存住），只有当空闲时间超出一段时间(默认为60s)后，线程池才会销毁该线程（相当于清除过时的缓存）。新任务到达后，线程池首先会让被缓存住的线程（空闲状态）去执行任务，如果没有可用线程（无空闲线程），便会创建新的线程。   
适用场景：处理任务速度 > 提交任务速度,耗时少的任务(避免无限新增线程)

3. newFixedThreadPool :创建一个定长线程池，可控制线程最大并发数，超出的线程会在队列中等待。

4. newScheduledThreadPool:创建一个定长线程池，支持定时及周期性任务执行

## 15、CopyOnWriteArrayList

CopyOnWriteArrayList : 写时加锁，当添加一个元素的时候，将原来的容器进行copy，复制出一个新的容器，然后在新的容器里面写，写完之后再将原容器的引用指向新的容器，而读的时候是读旧容器的数据，所以可以进行并发的读，但这是一种弱一致性的策略。   
使用场景：CopyOnWriteArrayList适合使用在读操作远远大于写操作的场景里，比如缓存。

## 16、AQS

AQS使用一个int成员变量来表示同步状态，通过内置的FIFO队列来完成获取资源线程的排队工作。

private volatile int state; //共享变量，使用volatile修饰保证线程可见性

1. 2种同步方式：独占式，共享式。独占式如ReentrantLock，共享式如Semaphore，CountDownLatch，组合式的如ReentrantReadWriteLock
2. 节点的状态   
   CANCELLED，值为1，表示当前的线程被取消；   
   SIGNAL，值为-1，表示当前节点的后继节点包含的线程需要运行，也就是unpark；   
   CONDITION，值为-2，表示当前节点在等待condition，也就是在condition队列中；   
   PROPAGATE，值为-3，表示当前场景下后续的acquireShared能够得以执行；   
   值为0，表示当前节点在sync队列中，等待着获取锁。
3. 模板方法模式   
   　protected boolean tryAcquire(int arg) : 独占式获取同步状态，试着获取，成功返回true，反之为false   
   　protected boolean tryRelease(int arg) ：独占式释放同步状态，等待中的其他线程此时将有机会获取到同步状态；   
   　protected int tryAcquireShared(int arg) ：共享式获取同步状态，返回值大于等于0，代表获取成功；反之获取失败；   
   　protected boolean tryReleaseShared(int arg) ：共享式释放同步状态，成功为true，失败为false   
   AQS维护一个共享资源state，通过内置的FIFO来完成获取资源线程的排队工作。该队列由一个一个的Node结点组成，每个Node结点维护一个prev引用和next引用，分别指向自己的前驱和后继结点。双端双向链表。
4. 独占式:乐观的并发策略   
    acquire   
   　 a.首先tryAcquire获取同步状态，成功则直接返回；否则，进入下一环节；   
    b.线程获取同步状态失败，就构造一个结点，加入同步队列中，这个过程要保证线程安全；   
    　c.加入队列中的结点线程进入自旋状态，若是老二结点（即前驱结点为头结点），才有机会尝试去获取同步状态；否则，当其前驱结点的状态为SIGNAL，线程便可安心休息，进入阻塞状态，直到被中断或者被前驱结点唤醒。   
    release   
    release的同步状态相对简单，需要找到头结点的后继结点进行唤醒，若后继结点为空或处于CANCEL状态，从后向前遍历找寻一个正常的结点，唤醒其对应线程。
5. 共享式:   
   共享式地获取同步状态.同步状态的方法tryAcquireShared返回值为int。   
   a.当返回值大于0时，表示获取同步状态成功，同时还有剩余同步状态可供其他线程获取；   
   　b.当返回值等于0时，表示获取同步状态成功，但没有可用同步状态了；   
   　c.当返回值小于0时，表示获取同步状态失败。
6. AQS实现公平锁和非公平锁   
   非公平锁中，那些尝试获取锁且尚未进入等待队列的线程会和等待队列head结点的线程发生竞争。公平锁中，在获取锁时，增加了isFirst(current)判断，当且仅当，等待队列为空或当前线程是等待队列的头结点时，才可尝试获取锁。   
   　[Java并发包基石-AQS详解](https://www.cnblogs.com/chengxiao/archive/2017/07/24/7141160.html)

## 16、Java里的阻塞队列

7个阻塞队列。分别是

ArrayBlockingQueue ：一个由数组结构组成的有界阻塞队列。   
LinkedBlockingQueue ：一个由链表结构组成的有界阻塞队列。   
PriorityBlockingQueue ：一个支持优先级排序的无界阻塞队列。   
DelayQueue：一个使用优先级队列实现的无界阻塞队列。   
SynchronousQueue：一个不存储元素的阻塞队列。   
LinkedTransferQueue：一个由链表结构组成的无界阻塞队列。   
LinkedBlockingDeque：一个由链表结构组成的双向阻塞队列。

添加元素

Java中的阻塞队列接口BlockingQueue继承自Queue接口。BlockingQueue接口提供了3个添加元素方法。   
add：添加元素到队列里，添加成功返回true，由于容量满了添加失败会抛出IllegalStateException异常   
offer：添加元素到队列里，添加成功返回true，添加失败返回false   
put：添加元素到队列里，如果容量满了会阻塞直到容量不满

删除方法

3个删除方法   
poll：删除队列头部元素，如果队列为空，返回null。否则返回元素。   
remove：基于对象找到对应的元素，并删除。删除成功返回true，否则返回false   
take：删除队列头部元素，如果队列为空，一直阻塞到队列有元素并删除

## 17、condition

对Condition的源码理解，主要就是理解等待队列，等待队列可以类比同步队列，而且等待队列比同步队列要简单，因为等待队列是单向队列，同步队列是双向队列。

[java condition使用及分](http://blog.csdn.net/bohu83/article/details/51098106" \t "_blank)

## 18、DelayQueue

队列中每个元素都有个过期时间，并且队列是个优先级队列，当从队列获取元素时候，只有过期元素才会出队列。

[并发队列-无界阻塞延迟队列delayqueue原理探究](http://ifeve.com/%E5%B9%B6%E5%8F%91%E9%98%9F%E5%88%97-%E6%97%A0%E7%95%8C%E9%98%BB%E5%A1%9E%E5%BB%B6%E8%BF%9F%E9%98%9F%E5%88%97delayqueue%E5%8E%9F%E7%90%86%E6%8E%A2%E7%A9%B6/" \t "_blank)

## 19、Fork/Join框架

　Fork/Join框架是Java 7提供的一个用于并行执行任务的框架，是一个把大任务分割成若干个小任务，最终汇总每个小任务结果后得到大任务结果的框架。Fork/Join框架要完成两件事情：

　　1.任务分割：首先Fork/Join框架需要把大的任务分割成足够小的子任务，如果子任务比较大的话还要对子任务进行继续分割

　　2.执行任务并合并结果：分割的子任务分别放到双端队列里，然后几个启动线程分别从双端队列里获取任务执行。子任务执行完的结果都放在另外一个队列里，启动一个线程从队列里取数据，然后合并这些数据。

　　在Java的Fork/Join框架中，使用两个类完成上述操作

　　1.ForkJoinTask:我们要使用Fork/Join框架，首先需要创建一个ForkJoin任务。该类提供了在任务中执行fork和join的机制。通常情况下我们不需要直接集成ForkJoinTask类，只需要继承它的子类，Fork/Join框架提供了两个子类：

　　　　a.RecursiveAction：用于没有返回结果的任务

　　　　b.RecursiveTask:用于有返回结果的任务

　　2.ForkJoinPool:ForkJoinTask需要通过ForkJoinPool来执行

任务分割出的子任务会添加到当前工作线程所维护的双端队列中，进入队列的头部。当一个工作线程的队列里暂时没有任务时，它会随机从其他工作线程的队列的尾部获取一个任务(工作窃取算法)。   
Fork/Join框架的实现原理   
　　ForkJoinPool由ForkJoinTask数组和ForkJoinWorkerThread数组组成，ForkJoinTask数组负责将存放程序提交给ForkJoinPool，而ForkJoinWorkerThread负责执行这些任务。

[Fork/Join框架详解](https://www.cnblogs.com/senlinyang/p/7885964.html)

## 20、原子操作类

在java.util.concurrent.atomic包下，可以分为四种类型的原子更新类：原子更新基本类型、原子更新数组类型、原子更新引用和原子更新属性。

1. 原子更新基本类型   
   使用原子方式更新基本类型，共包括3个类：   
   AtomicBoolean：原子更新布尔变量   
   AtomicInteger：原子更新整型变量   
   AtomicLong：原子更新长整型变量
2. 原子更新数组   
   通过原子更新数组里的某个元素，共有3个类：   
   AtomicIntegerArray：原子更新整型数组的某个元素   
   AtomicLongArray：原子更新长整型数组的某个元素   
   AtomicReferenceArray：原子更新引用类型数组的某个元素   
   AtomicIntegerArray常用的方法有：   
   int addAndSet(int i, int delta)：以原子方式将输入值与数组中索引为i的元素相加   
   boolean compareAndSet(int i, int expect, int update)：如果当前值等于预期值，则以原子方式更新数组中索引为i的值为update值
3. 原子更新引用类型   
   AtomicReference：原子更新引用类型   
   AtomicReferenceFieldUpdater：原子更新引用类型里的字段   
   AtomicMarkableReference：原子更新带有标记位的引用类型。
4. 原子更新字段类   
   如果需要原子更新某个类的某个字段，就需要用到原子更新字段类，可以使用以下几个类：   
   AtomicIntegerFieldUpdater：原子更新整型字段   
   AtomicLongFieldUpdater：原子更新长整型字段   
   AtomicStampedReference：原子更新带有版本号的引用类型。   
   要想原子更新字段，需要两个步骤：   
   每次必须使用newUpdater创建一个更新器，并且需要设置想要更新的类的字段   
   更新类的字段（属性）必须为public volatile

## 21、同步屏障CyclicBarrier

CyclicBarrier 的字面意思是可循环使用（Cyclic）的屏障（Barrier）。它要做的事情是，让一组线程到达一个屏障（也可以叫同步点）时被阻塞，直到最后一个线程到达屏障时，屏障才会开门，所有被屏障拦截的线程才会继续干活。CyclicBarrier默认的构造方法是CyclicBarrier(int parties)，其参数表示屏障拦截的线程数量，每个线程调用await方法告诉CyclicBarrier我已经到达了屏障，然后当前线程被阻塞。

CyclicBarrier和CountDownLatch的区别

CountDownLatch的计数器只能使用一次。而CyclicBarrier的计数器可以使用reset() 方法重置。所以CyclicBarrier能处理更为复杂的业务场景，比如如果计算发生错误，可以重置计数器，并让线程们重新执行一次。   
CyclicBarrier还提供其他有用的方法，比如getNumberWaiting方法可以获得CyclicBarrier阻塞的线程数量。isBroken方法用来知道阻塞的线程是否被中断。比如以下代码执行完之后会返回true。

## 22、Semaphore

Semaphore（信号量）是用来控制同时访问特定资源的线程数量，它通过协调各个线程，以保证合理的使用公共资源   
Semaphore可以用于做流量控制，特别公用资源有限的应用场景，比如数据库连接。假如有一个需求，要读取几万个文件的数据，因为都是IO密集型任务，我们可以启动几十个线程并发的读取，但是如果读到内存后，还需要存储到数据库中，而数据库的连接数只有10个，这时我们必须控制只有十个线程同时获取数据库连接保存数据，否则会报错无法获取数据库连接。这个时候，我们就可以使用Semaphore来做流控，代码如下：

[控制并发线程数的Semaphore](http://ifeve.com/concurrency-semaphore/" \t "_blank)

## 23、死锁,以及解决死锁

**死锁产生的四个必要条件**

互斥条件：资源是独占的且排他使用，进程互斥使用资源，即任意时刻一个资源只能给一个进程使用，其他进程若申请一个资源，而该资源被另一进程占有时，则申请者等待直到资源被占有者释放。   
不可剥夺条件：进程所获得的资源在未使用完毕之前，不被其他进程强行剥夺，而只能由获得该资源的进程资源释放。   
请求和保持条件：进程每次申请它所需要的一部分资源，在申请新的资源的同时，继续占用已分配到的资源。   
循环等待条件：在发生死锁时必然存在一个进程等待队列{P1,P2,…,Pn},其中P1等待P2占有的资源，P2等待P3占有的资源，…，Pn等待P1占有的资源，形成一个进程等待环路，环路中每一个进程所占有的资源同时被另一个申请，也就是前一个进程占有后一个进程所深情地资源。

**解决死锁**

一是死锁预防，就是不让上面的四个条件同时成立。   
二是，合理分配资源。   
三是使用银行家算法，如果该进程请求的资源操作系统剩余量可以满足，那么就分配。

## 24、进程间的通信方式

1. 管道( pipe)：管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有亲缘关系的进程间使用。进程的亲缘关系通常是指父子进程关系。

2. 有名管道 (named pipe) ： 有名管道也是半双工的通信方式，但是它允许无亲缘关系进程间的通信。

3. 信号量( semophore ) ： 信号量是一个计数器，可以用来控制多个进程对共享资源的访问。它常作为一种锁机制，防止某进程正在访问共享资源时，其他进程也访问该资源。因此，主要作为进程间以及同一进程内不同线程之间的同步手段。

4. 消息队列( message queue ) ： 消息队列是由消息的链表，存放在内核中并由消息队列标识符标识。消息队列克服了信号传递信息少、管道只能承载无格式字节流以及缓冲区大小受限等缺点。

5. 信号 ( sinal ) ： 信号是一种比较复杂的通信方式，用于通知接收进程某个事件已经发生。

6. 共享内存( shared memory ) ：共享内存就是映射一段能被其他进程所访问的内存，这段共享内存由一个进程创建，但多个进程都可以访问。共享内存是最快的 IPC 方式，它是针对其他进程间通信方式运行效率低而专门设计的。它往往与其他通信机制，如信号量，配合使用，来实现进程间的同步和通信。

7. 套接字( socket ) ： 套解口也是一种进程间通信机制，与其他通信机制不同的是，它可用于不同机器间的进程通信。

## 中断

interrupt()的作用是中断本线程。   
本线程中断自己是被允许的；其它线程调用本线程的interrupt()方法时，会通过checkAccess()检查权限。这有可能抛出SecurityException异常。   
如果本线程是处于阻塞状态：调用线程的wait(), wait(long)或wait(long, int)会让它进入等待(阻塞)状态，或者调用线程的join(), join(long), join(long, int), sleep(long), sleep(long, int)也会让它进入阻塞状态。若线程在阻塞状态时，调用了它的interrupt()方法，那么它的“中断状态”会被清除并且会收到一个InterruptedException异常。例如，线程通过wait()进入阻塞状态，此时通过interrupt()中断该线程；调用interrupt()会立即将线程的中断标记设为“true”，但是由于线程处于阻塞状态，所以该“中断标记”会立即被清除为“false”，同时，会产生一个InterruptedException的异常。   
如果线程被阻塞在一个Selector选择器中，那么通过interrupt()中断它时；线程的中断标记会被设置为true，并且它会立即从选择操作中返回。   
如果不属于前面所说的情况，那么通过interrupt()中断线程时，它的中断标记会被设置为“true”。   
中断一个“已终止的线程”不会产生任何操作。

终止处于“阻塞状态”的线程   
通常，我们通过“中断”方式终止处于“阻塞状态”的线程。   
当线程由于被调用了sleep(), wait(), join()等方法而进入阻塞状态；若此时调用线程的interrupt()将线程的中断标记设为true。由于处于阻塞状态，中断标记会被清除，同时产生一个InterruptedException异常。将InterruptedException放在适当的为止就能终止线程，

终止处于“运行状态”的线程

interrupted() 和 isInterrupted()的区别

最后谈谈 interrupted() 和 isInterrupted()。   
interrupted() 和 isInterrupted()都能够用于检测对象的“中断标记”。   
区别是，interrupted()除了返回中断标记之外，它还会清除中断标记(即将中断标记设为false)；而isInterrupted()仅仅返回中断标记。   
[interrupt()和线程终止方式](https://www.cnblogs.com/skywang12345/p/3479949.html" \t "_blank)

https://blog.csdn.net/u012998254/article/details/79400549