线程 进程

# **线程和进程的区别是什么？**

<https://www.zhihu.com/question/25532384/answer/81152571>

<https://zhuanlan.zhihu.com/p/34678757>

<https://zhuanlan.zhihu.com/p/35773962>

<https://www.jianshu.com/p/2dc01727be45>

1. 进程是系统进行资源分配(assignment)和调度(dispatching)的独立单位。每一个进程都有它自己的内存空间和系统资源

(1)**创建进程**，系统在创建一个进程时，必须为它分配其所必需的、除处理机以外的所有资源，如内存空间、I/O设备，以及建立相应的PCB；

(2)**撤消进程**，系统在撤消进程时，又必须先对其所占有的资源执行回收操作，然后再撤消PCB；

(3)**进程切换**，对进程进行上下文切换时，需要保留当前进程的CPU环境，设置新选中进程的CPU环境，因而须花费不少的处理机时间。

1. 在同一个进程内又可以执行多个任务，而这每一个任务我就可以看出是一个线程。一个进程会有1个或多个线程的！
2. 进程作为**资源分配**的基本单位。线程作为**资源调度**的基本单位，是程序的执行单元，执行路径(单线程：一条执行路径，多线程：多条执行路径)。是程序使用CPU的最基本单位。

线程有**3个基本状态**：执行、就绪、阻塞

线程有**5种基本操作**：派生、阻塞、激活、 调度、 结束

线程的**属性**：

1)轻型实体；

2)独立调度和分派的基本单位；

3)可并发执行；

4)共享进程资源。

1) lightweight entity;

2) basic units for independent dispatching

3) it can be executed concurrently;

4) sharing process resources.

线程有两个**基本类型**：

1. **用户级线程**：管理过程全部由用户程序完成，操作系统内核心只对进程进行管理
2. **系统级线程**(核心级线程)：由操作系统内核进行管理。操作系统内核给应用程序提供相应的系统调用和应用程序接口API，以使用户程序可以创建、执行以及撤消线程

4.值得注意的是：多线程的存在，**不是提高程序的执行速度**。其实是为了**提高应用程序的使用率**，**程序的执行其实都是在抢CPU的资源，CPU的执行权**。多个进程是在抢这个资源，**而其中的某一个进程如果执行路径比较多，就会有更高的几率抢到CPU的执行权**

**5.并行与并发Parallelism and concurrency**

并行：

并行性是指**同一时刻内**发生两个或多个事件。

并行是在**不同实体**上的多个事件

并发：

并发性是指**同一时间间隔内**发生两个或多个事件。

并发是在**同一实体上**的多个事件

由此可见：**并行是针对进程的，并发是针对线程的**。

那么大概是这样的：进办公室有两个门(两CPU)，如果两同学分别从不同的门进入，不管先后性，两者互相独立，那么是并行；如果两同学不管以什么方式进入，在路人甲看来，他两同时都在办公室内，就是并发

<https://www.cnblogs.com/f-ck-need-u/p/11161481.html>

<https://segmentfault.com/a/1190000012754925>

1. Java实现多线程

Java实现多线程是使用Thread这个类的

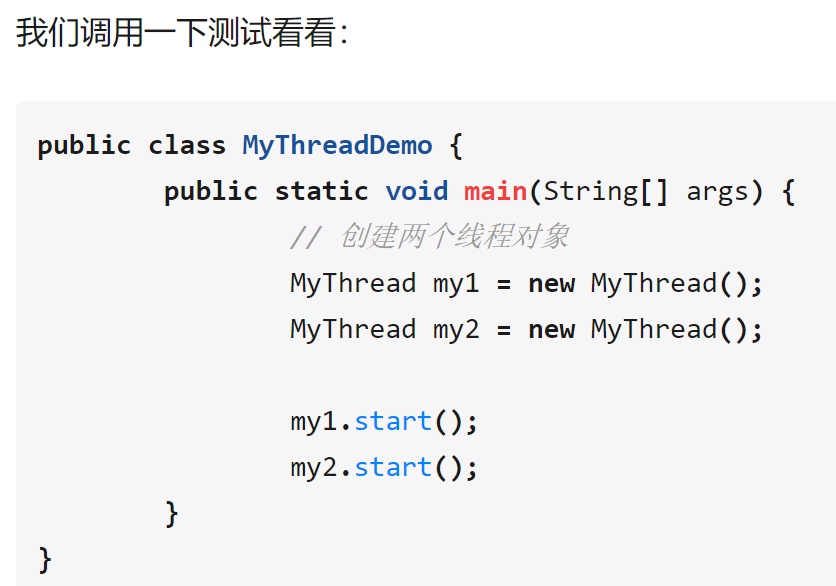
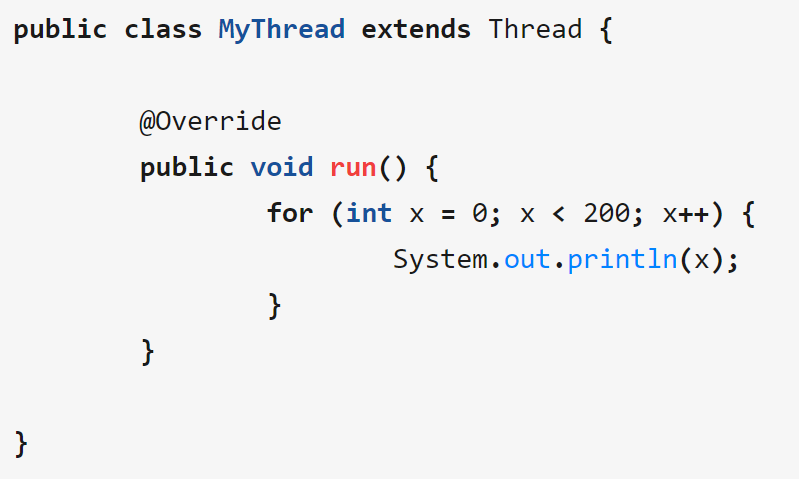
创建多线程有两种方法：

继承Thread，重写run方法

实现Runnable接口，重写run方法

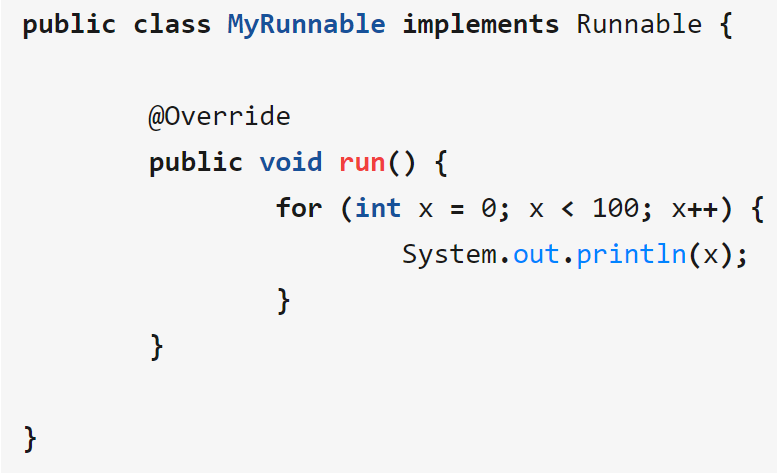
**6.1 继承Thread，重写run方法**

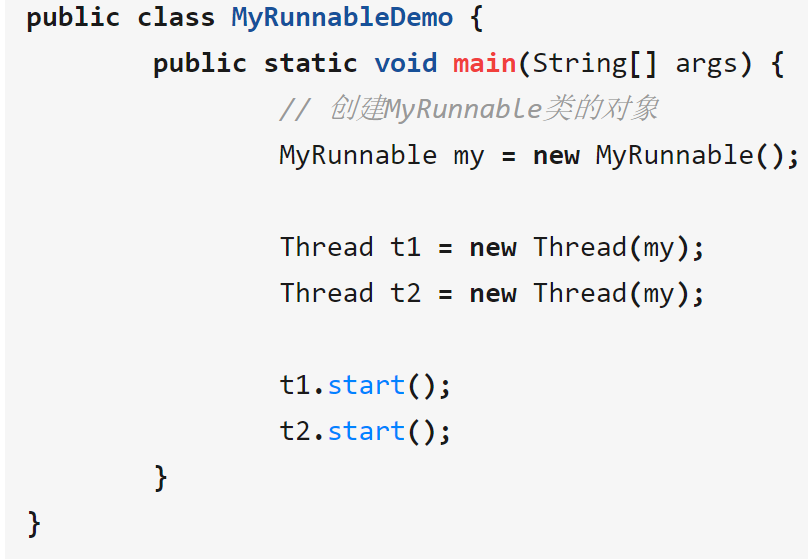
**创建一个类，继承Thread，重写run方法**

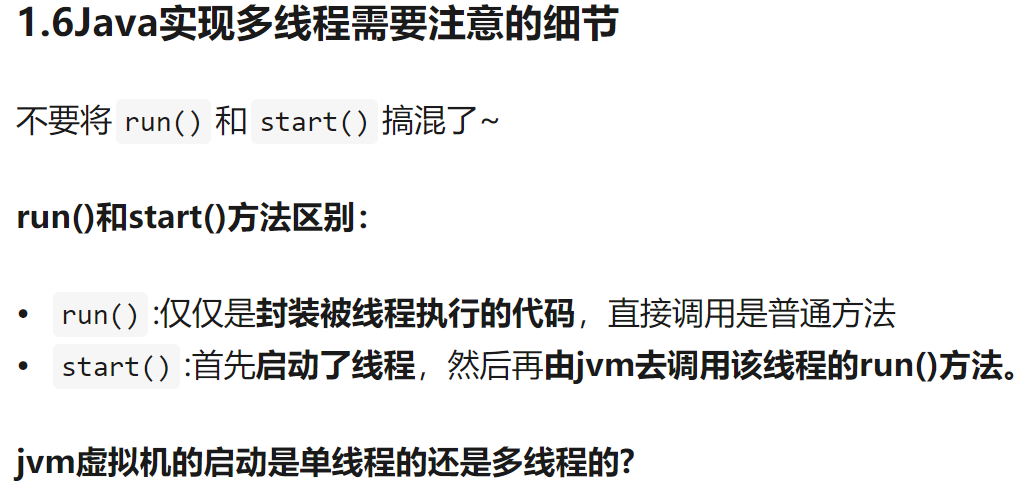


6.2 **实现Runnable接口，重写run方法**

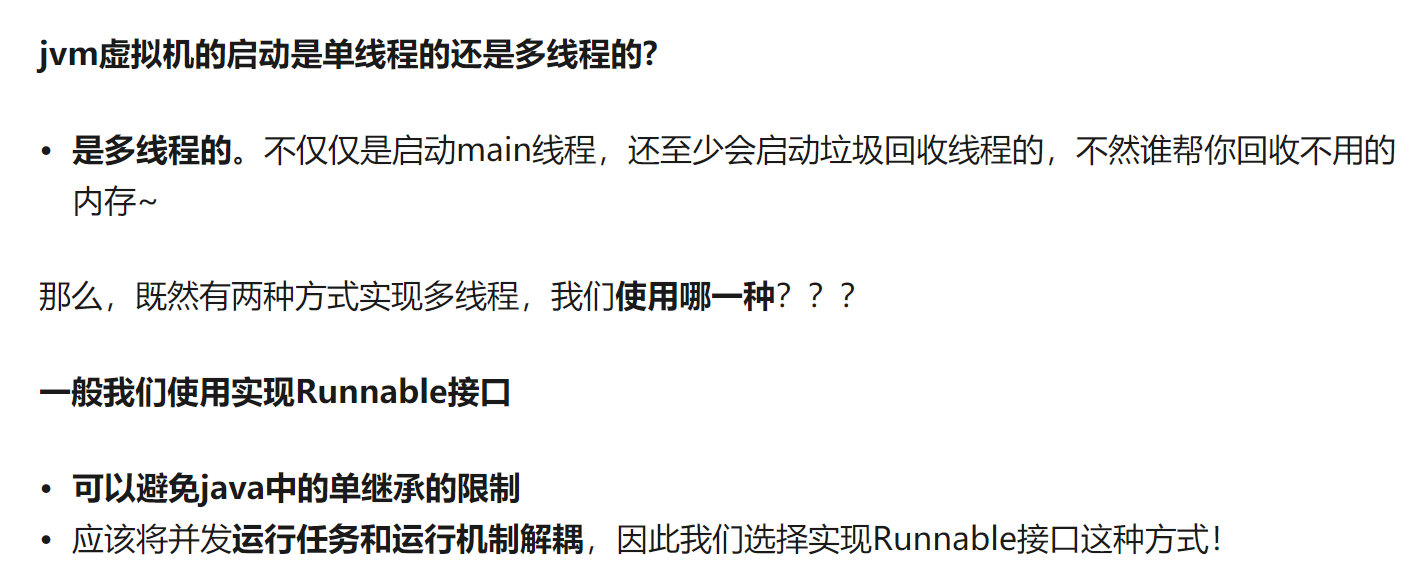
**实现Runnable接口，重写run方法**







封装encapsulation



6.3 创建线程的三种方式的对比？

1）采用实现Runnable、Callable接口的方式创建多线程。

优势是：线程类只是实现了Runnable接口或Callable接口，还可以继承其他类。

在这种方式下，多个线程可以共享同一个target对象，所以非常适合多个相同线程来处理同一份资源的情况，从而可以将CPU、代码和数据分开，形成清晰的模型，较好地体现了面向对象的思想。

劣势是：编程稍微复杂，如果要访问当前线程，则必须使用Thread.currentThread()方法。

2）使用继承Thread类的方式创建多线程

优势是：编写简单，如果需要访问当前线程，则无需使用Thread.currentThread()方法，直接使用this即可获得当前线程。

劣势是：线程类已经继承了Thread类，所以不能再继承其他父类。

3）Runnable和Callable的区别

Callable规定（重写）的方法是call()，Runnable规定（重写）的方法是run()。

Callable的任务执行后可返回值，而Runnable的任务是不能返回值的。

Call方法可以抛出异常，run方法不可以。

运行Callable任务可以拿到一个Future对象，表示异步计算的结果。它提供了检查计算是否完成的方法，以等待计算的完成，并检索计算的结果。通过Future对象可以了解任务执行情况，可取消任务的执行，还可获取执行结果。

1. 并发编程三要素？

1）原子性 atomic

原子性指的是一个或者多个操作，要么全部执行并且在执行的过程中不被其他操作打断，要么就全部都不执行。

2）可见性

可见性指多个线程操作一个共享变量时，其中一个线程对变量进行修改后，其他线程可以立即看到修改的结果。

实现可见性的方法：

synchronized或者Lock：保证同一个时刻只有一个线程获取锁执行代码，锁释放之前把最新的值刷新到主内存，实现可见性。

3）有序性

有序性，即程序的执行顺序按照代码的先后顺序来执行

8.多线程的价值？

1）发挥多核CPU的优势

多线程，可以真正发挥出多核CPU的优势来，达到充分利用CPU的目的，采用多线程的方式去同时完成几件事情而不互相干扰。

2）防止阻塞

从程序运行效率的角度来看，单核CPU不但不会发挥出多线程的优势，反而会因为在单核CPU上运行多线程导致线程上下文的切换，而降低程序整体的效率。但是单核CPU我们还是要应用多线程，就是为了防止阻塞。试想，如果单核CPU使用单线程，那么只要这个线程阻塞了，比方说远程读取某个数据吧，对端迟迟未返回又没有设置超时时间，那么你的整个程序在数据返回回来之前就停止运行了。多线程可以防止这个问题，多条线程同时运行，哪怕一条线程的代码执行读取数据阻塞，也不会影响其它任务的执行。

3）便于建模

这是另外一个没有这么明显的优点了。假设有一个大的任务A，单线程编程，那么就要考虑很多，建立整个程序模型比较麻烦。但是如果把这个大的任务A分解成几个小任务，任务B、任务C、任务D，分别建立程序模型，并通过多线程分别运行这几个任务，那就简单很多了

9.Java线程具有五中基本状态

1）**新建状态**（New）：当线程对象对创建后，即进入了新建状态，如：Thread t = new MyThread();

2）**就绪状态**（Runnable）：当调用线程对象的start()方法（t.start();），线程即进入就绪状态。处于就绪状态的线程，只是说明此线程已经做好了准备，随时等待CPU调度执行，并不是说执行了t.start()此线程立即就会执行；

3）**运行状态**（Running）：当CPU开始调度处于就绪状态的线程时，此时线程才得以真正执行，即进入到运行状态。注：就 绪状态是进入到运行状态的唯一入口，也就是说，线程要想进入运行状态执行，首先必须处于就绪状态中；

4）**阻塞状态**（Blocked）：处于运行状态中的线程由于某种原因，暂时放弃对CPU的使用权，停止执行，此时进入阻塞状态，直到其进入到就绪状态，才 有机会再次被CPU调用以进入到运行状态。根据阻塞产生的原因不同，阻塞状态又可以分为三种：

1.等待阻塞：运行状态中的线程执行wait()方法，使本线程进入到等待阻塞状态；

2.同步阻塞 — 线程在获取synchronized同步锁失败(因为锁被其它线程所占用)，它会进入同步阻塞状态；

3.其他阻塞 — 通过调用线程的sleep()或join()或发出了I/O请求时，线程会进入到阻塞状态。当sleep()状态超时、join()等待线程终止或者超时、或者I/O处理完毕时，线程重新转入就绪状态。

**5）死亡状态（Dead）**：线程执行完了或者因异常退出了run()方法，该线程结束生命周期。

10.什么是线程池？ 有哪几种创建方式？

线程池就是提前创建若干个线程，如果有任务需要处理，线程池里的线程就会处理任务，处理完之后线程并不会被销毁，而是等待下一个任务。由于创建和销毁线程都是消耗系统资源的，所以当你想要频繁的创建和销毁线程的时候就可以考虑使用线程池来提升系统的性能。

java 提供了一个 java.util.concurrent.Executor接口的实现用于创建线程池。

线程池 参数: <https://blog.csdn.net/u011001084/article/details/78297285>

四种线程池的创建：

（1）newCachedThreadPool创建一个可缓存线程池

（2）newFixedThreadPool 创建一个定长线程池，可控制线程最大并发数。

（3）newScheduledThreadPool 创建一个定长线程池，支持定时及周期性任务执行。

（4）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。

11.线程池的优点？

1）重用存在的线程，减少对象创建销毁的开销。

2）可有效的控制最大并发线程数，提高系统资源的使用率，同时避免过多资源竞争，避免堵塞。

3）提供定时执行、定期执行、单线程、并发数控制等功能。

1. Java中的同步集合与并发集合有什么区别？

**同步集合类：**

Vector

Stack

HashTable

Collections.synchronized方法生成

**并发集合类：**

ConcurrentHashMap

CopyOnWriteArrayList

CopyOnWriteArraySet等

**同步集合与并发集合的区别**

同步集合与并发集合都为多线程和并发提供了合适的线程安全的集合，不过并发集合的可扩展性更高。同步集合比并发集合会慢得多，主要原因是锁，同步集合会对整个May或List加锁，而并发集合例如ConcurrentHashMap，

把整个Map 划分成几个片段，只对相关的几个片段上锁，同时允许多线程访问其他未上锁的片段

1. synchronized的作用？

在Java中，synchronized关键字是用来控制线程同步的，就是在多线程的环境下，控制synchronized代码段不被多个线程同时执行。

synchronized既可以加在一段代码上，也可以加在方法上

14.什么是乐观锁和悲观锁

（1）乐观锁：就像它的名字一样，对于并发间操作产生的线程安全问题持乐观状态，乐观锁认为竞争不总是会发生，因此它不需要持有锁，将比较-替换这两个动作作为一个原子操作尝试去修改内存中的变量，如果失败则表示发生冲突，那么就应该有相应的重试逻辑。

（2）悲观锁：还是像它的名字一样，对于并发间操作产生的线程安全问题持悲观状态，悲观锁认为竞争总是会发生，因此每次对某资源进行操作时，都会持有一个独占的锁，就像synchronized，不管三七二十一，直接上了锁就操作资源了。

15.sleep方法和wait方法有什么区别?

这个问题常问，sleep方法和wait方法都可以用来放弃CPU一定的时间，不同点在于如果线程持有某个对象的监视器，sleep方法不会放弃这个对象的监视器，wait方法会放弃这个对象的监视器

16.线程的调度策略

线程调度器选择优先级最高的线程运行，但是，如果发生以下情况，就会终止线程的运行：

（1）线程体中调用了yield方法让出了对cpu的占用权利

（2）线程体中调用了sleep方法使线程进入睡眠状态

（3）线程由于IO操作受到阻塞

（4）另外一个更高优先级线程出现

（5）在支持时间片的系统中，该线程的时间片用完

17.Java死锁以及如何避免？

Java中的死锁是一种编程情况，其中两个或多个线程被永久阻塞，Java死锁情况出现至少两个线程和两个或更多资源。

Java发生死锁的根本原因是：在申请锁时发生了交叉闭环申请。

死锁的原因

1）是多个线程涉及到多个锁，这些锁存在着交叉，所以可能会导致了一个锁依赖的闭环。

例如：线程在获得了锁A并且没有释放的情况下去申请锁B，这时，另一个线程已经获得了锁B，在释放锁B之前又要先获得锁A，因此闭环发生，陷入死锁循环。

2）默认的锁申请操作是阻塞的。

所以要避免死锁，就要在一遇到多个对象锁交叉的情况，就要仔细审查这几个对象的类中的所有方法，是否存在着导致锁依赖的环路的可能性。 总之是尽量避免在一个同步方法中调用其它对象的延时方法和同步方法。

18.怎么唤醒一个阻塞的线程

如果线程是因为调用了wait()、sleep()或者join()方法而导致的阻塞，可以中断线程，并且通过抛出InterruptedException来唤醒它；如果线程遇到了IO阻塞，无能为力，因为IO是操作系统实现的，Java代码并没有办法直接接触到操作系统。

19.什么是多线程的上下文切换Context switching

多线程的上下文切换是指CPU控制权由一个已经正在运行的线程切换到另外一个就绪并等待获取CPU执行权的线程的过程。

20.Java中用到的线程调度算法是什么

抢占式。一个线程用完CPU之后，操作系统会根据线程优先级、线程饥饿情况等数据算出一个总的优先级并分配下一个时间片给某个线程执行。

<https://zhuanlan.zhihu.com/p/60950930>

上还有其他的

1. 什么是线程安全和线程不安全？

通俗的说：**加锁的就是是线程安全的，不加锁的就是是线程不安全的**

线程安全

线程安全: 就是多线程访问时，采用了**加锁机制**，当一个线程访问该类的某个数据时，进行保护，其他线程不能进行访问，直到该线程读取完，其他线程才可使用。不会出现数据不一致或者数据污染。

一个线程安全的计数器类的同一个实例对象在被多个线程使用的情况下也不会出现计算失误。很显然你可以将集合类分成两组，线程安全和非线程安全的。 Vector 是用同步方法来实现线程安全的, 而和它相似的ArrayList不是线程安全的。

线程不安全

线程不安全：**就是不提供数据访问保护，有可能出现多个线程先后更改数据造成所得到的数据是脏数据**

如果你的代码所在的进程中有多个线程在同时运行，而这些线程可能会同时运行这段代码。如果每次运行结果和单线程运行的结果是一样的，而且其他的变量的值也和预期的是一样的，就是线程安全的。

线程安全问题都是由全局变量及静态变量引起的。 若每个线程中对全局变量、静态变量只有读操作，而无写操作，一般来说，这个全局变量是线程安全的；若有多个线程同时执行写操作，一般都需要考虑线程同步，否则的话就可能影响线程安全

1. 自旋锁是SMP架构中的一种low-level的同步机制。

当线程A想要获取一把自选锁而该锁又被其它线程锁持有时，线程A会在一个循环中自选以检测锁是不是已经可用了。

自选锁需要注意：

由于自旋时不释放CPU，因而持有自旋锁的线程应该尽快释放自旋锁，否则等待该自旋锁的线程会一直在那里自旋，这就会浪费CPU时间。

持有自旋锁的线程在sleep之前应该释放自旋锁以便其它线程可以获得自旋锁