/\*

1，进程和线程

**进程**：正在进行中的程序(直译).

**线程**：就是进程中一个负责程序执行的控制单元(执行路径)

1，一个进程中可以多执行路径，称之为多线程。

2，一个进程中至少要有一个线程。

3，开启多个线程是为了同时运行多部分代码。

4，每一个线程都有自己运行的内容。这个内容可以称为线程要执行的任务。

5，多线程好处：解决了多部分同时运行的问题。

6，多线程的弊端：线程太多回到效率的降低。

7，其实应用程序的执行都是cpu在做着快速的切换完成的。这个切换是随机的。

**Example：** 比如360杀毒软件，360已启动，就相当于一个进程，然后他可以杀毒，可以修复，这就是有多个执行路劲，这就是多线程

2，并发和并行

1，你吃饭吃到一半，电话来了，你一直到吃完了以后才去接，这就说明你不支持并发也不支持并行。

2，你吃饭吃到一半，电话来了，你停了下来接了电话，接完后继续吃饭，这说明你支持并发。  （不一定是同时的）

3，你吃饭吃到一半，电话来了，你一边打电话一边吃饭，这说明你支持**并行**。

**4，联系：**

**并发：**的关键是你有处理多个任务的能力，不一定要同时。

**并行：**的关键是你有同时处理多个任务的能力。

所以我认为它们最关键的点就是：**是否是『同时』**

**5，理解：**

解释一：并行是指两个或者多个事件在同一时刻发生；而并发是指两个或多个事件在同一时间间隔发生。  
解释二：并行是在不同实体上的多个事件，并发是在同一实体上的多个事件。

并发：交替做不同事情的能力  
并行：同时做不同事情的能力  
6，专业术语：  
并发：不同的代码块交替执行  
并行：不同的代码块同时执行

7，并发和并行的意义：

并发和并行都可以处理“多任务”，二者的主要区别在于是否是“同时进行”多个的任务。

8，首先理解并行和并发的区别：

并行：指在同一时刻，有多条指令在多个处理器上同时执行；

并发：指在同一时刻，只能有一条指令执行，但多个进程指令被快速轮换执行，使得在宏观上具有多个进程同时执行的效果。

**大部分操作系统都支持多进程并发运行**：程序员一边使用开发工具写代码，一边打开网页在查API，同时还开着网易云听歌······还没完，每台电脑运行时还有大量的底层支撑性程序在后台运行······这些进程看**上去像是在同时工作**。真相是，对于一个CPU而言，他在某个时间点只能执行一个程序，即一个进程，其**实CPU是不断地在这些进程间轮换执行的**，由于执行速度相对人的感觉来说过快，所以我们感觉不到而已。

**进程**：正在进行中的程序(直译).

**线程**：就是进程中一个负责程序执行的控制单元(执行路径)

一个进程中可以多执行路径，称之为多线程。

一个进程中至少要有一个线程。

开启多个线程是为了同时运行多部分代码。

每一个线程都有自己运行的内容。这个内容可以称为线程要执行的任务。

**多线程好处**：解决了多部分同时运行的问题。

**多线程的弊端**：线程太多回到效率的降低。

**1、同步**就是指一个线程要等待上一个线程执行完之后才开始执行当前的线程。

**2、异步**是指一个线程去执行，它的下一个线程不必等待它执行完就开始执行。

**通俗的解释一下多线程**

多线程用于堆积处理，就像一个大土堆，一个推土机很慢，那么10个推土机一起来处理，当然速度就快了，不过由于位置的限制，如果20个推土机，那么推土机之间会产生相互的避让，相互摩擦，相互拥挤，反而不如10个处理的好，所以，多线程处理，线程数要开的恰当，就可以提高效率。

## 场景一:

**一个业务逻辑有很多次的循环，每次循环之间没有影响，比如验证1万条url路径是否存在，正常**情况要循环1万次，逐个去验证每一条URL，这样效率会很低，假设验证一条需要1分钟，总共就需要1万分钟，有点恐怖。这时可以用多线程，将1万条URL分成50等份，开50个线程，没个线程只需验证200条，这样所有的线程执行完是远小于1万分钟的

**最典型的如：**  
1、用户注册完成送大礼包/积分之类，且积分等也是另一个系统并比较耗时；且这类任务即使失败也不是特别重要的。  
2、后台线程：比如定期执行一些特殊任务，如定期更新配置文件，任务调度（如quartz），一些监控用于定期信息采集等。

### 多线程应用场景：

多线程最多的场景：web服务器本身；各种专用服务器（如[游戏服务器](https://www.baidu.com/s?wd=%E6%B8%B8%E6%88%8F%E6%9C%8D%E5%8A%A1%E5%99%A8&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)）；多线程的常见应用场景：

1、**后台任务**，例如：定时向大量（100w以上）的用户发送邮件；

**2、异步处理**，例如：发微博、记录日志等；

3、分布式计算

# 3，宏观概念和微观概念

1，宏观就是看得见摸得着的，

2，微观则是深入到分子原子水平的。  
3，我用化学举个例吧：元素是指一类物质，例如铁元素，所以元素是个宏观的名称  
。而分子、原子则是微观的，看不到的。应该很好判断吧？  
那在物理上，[热胀冷缩](https://www.baidu.com/s?wd=%E7%83%AD%E8%83%80%E5%86%B7%E7%BC%A9&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)就是宏观的，而[分子动理论](https://www.baidu.com/s?wd=%E5%88%86%E5%AD%90%E5%8A%A8%E7%90%86%E8%AE%BA&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)则是微观的

4，为什么要用到并发

通过**并发编程的形式可以将多核CPU的计算能力发挥到极致，性能得到提升**。

另外，在特殊的业务场景下先天的就适合于并发编程。比如在图像处理领域，一张1024X768像素的图片，包含达到78万6千多个像素。即时将所有的像素遍历一边都需要很长的时间，面对如此复杂的计算量就需要充分利用多核的计算的能力。又比如当我们在网上购物时，为了提升响应速度，需要拆分，减库存，生成订单等等这些操作，就可以进行拆分利用多线程的技术完成。**面对复杂业务模型，并行程序会比串行程序更适应业务需求，而并发编程更能吻合这种业务拆分** 。正是因为这些优点，使得多线程技术能够得到重视，也是一名CS学习者应该掌握的：

* 充分利用多核CPU的计算能力；
* 方便进行业务拆分，提升应用性能

**，把一个任务分解为多个任务，并发的执行这些分解的任务，这样就能减少总的运行时间。这叫做任务并行**

5，为什么要使用多线程？多线程的优点和缺点是什么？

**首先说下多线程出现的原因:**

为了解决负载均衡问题,充分利用CPU资源.为了提高CPU的使用率,采用多线程的方式去同时完成几件事情而不互相干扰.为了处理大量的IO操作时或处理的情况需要花费大量的时间等等,比如:读写文件,视频图像的采集,处理,显示,保存等

**5.2多线程的好处:**

1.使用线程可以把占据时间长的程序中的任务放到后台去处理

2.用户界面更加吸引人,这样比如用户点击了一个按钮去触发某件事件的处理,可以弹出一个进度条来显示处理的进度

3.程序的运行效率可能会提高

4.在一些等待的任务实现上如用户输入,文件读取和网络收发数据等,线程就比较有用了.

**多线程的缺点:**

1.如果有大量的线程,会影响性能,因为操作系统需要在它们之间切换.

2.更多的线程需要更多的内存空间

3.线程中止需要考虑对程序运行的影响.

4.通常块模型数据是在多个线程间共享的,需要防止线程死锁情况的发生

**多线程的实际应用：**

（1）一个浏览器必须能同时下载多个图片

（2）一个web服务器必须能同时相应多个用户请求

（3）Java虚拟机本身就在后台提供了一个超级线程来进行垃圾回收

**线程是独立运行的**，它并不知道进程中是否还有其他线程存在。线程的执行时抢占式的，也就是说，当前运行的线程在任何时候**都可能被挂起**，以便另外一个线程可以运行。

6，线程之间的状态

https://blog.csdn.net/freedomfanye/article/details/79812783

**进程的状态**

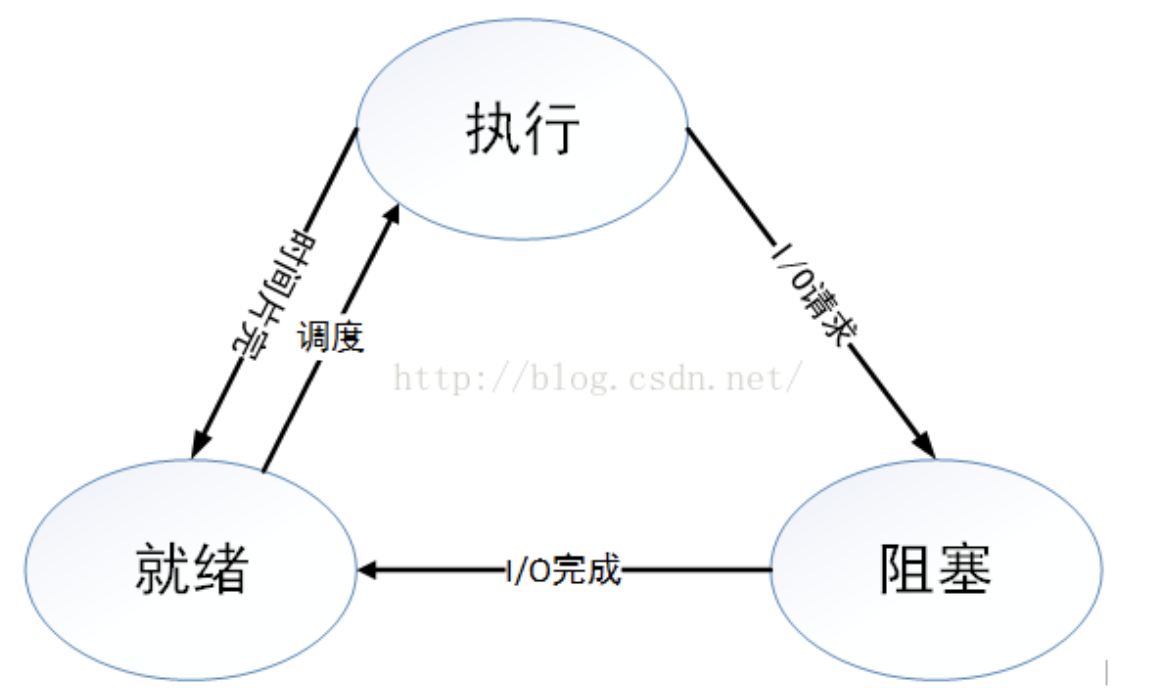
进程是系统进行资源分配和调度的基本单位。线程是CPU调度的基本单位。

一般来说，进程有三个状态：

**1、运行态**：进程占用CPU，并在CPU上运行。

**2、就绪态**：进程已经具备运行条件，但是CPU还没有分配过来。

**3、阻塞态**：进程因等待某件事发生而暂时不能运行。



7，线程的生命周期

https://blog.csdn.net/xingjing1226/article/details/81977129

1. 新建(NEW)：当用new操作符创建一个线程时。此时程序还没有开始运行线程中的代码。

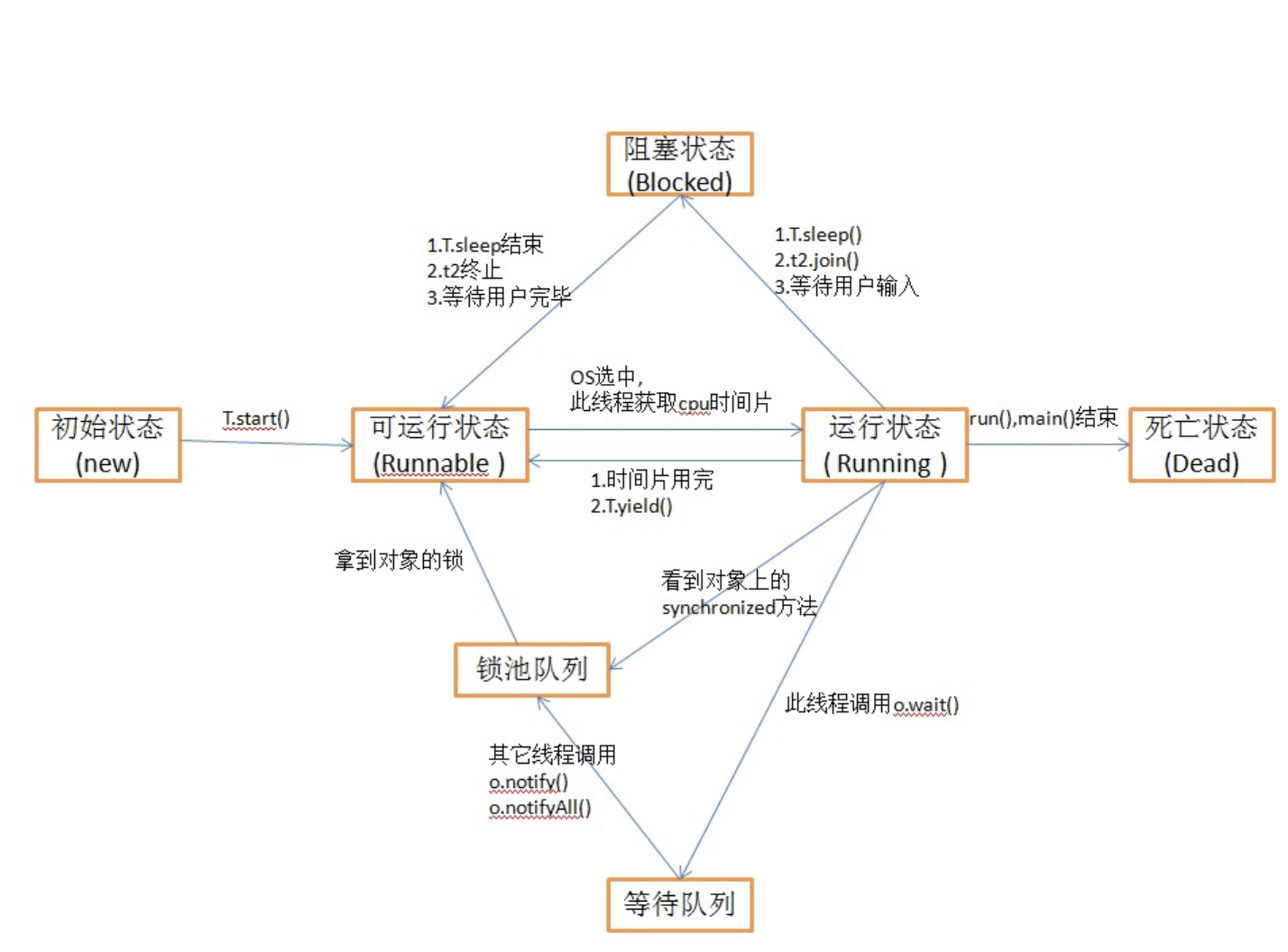
2.就绪状态(可运行(RUNNABLE)

一个新创建的线程并不自动开始运行，要执行线程，必须调用线程的start()方法。当线程对象调用start()方法即启动了线程，start()方法创建线程运行的系统资源，**并调度线程运行run()方法。当start()方法返回后，线程就处于就绪状态。**

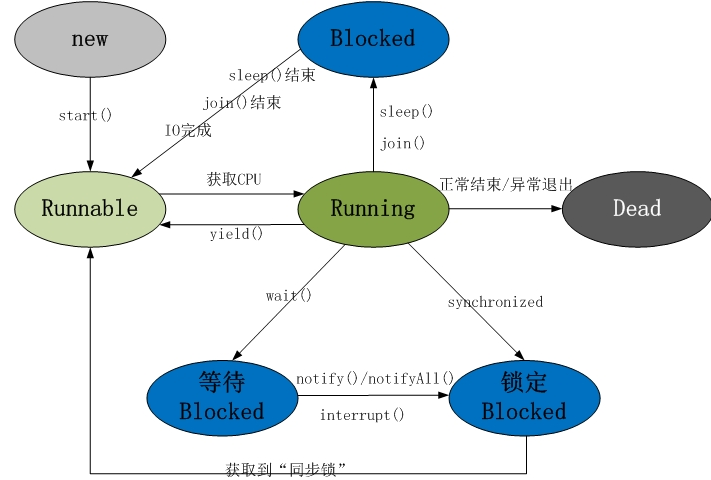
**处于就绪状态的线程并不一定立即运行run()方法，线程还必须同其他线程竞争CPU时间，**只有获得CPU时间才可以运行线程。因为在单CPU的计算机系统中，不可能同时运行多个线程，一个时刻仅有一个线程处于运行状态。因此此时可能有多个线程处于就绪状态。对多个处于就绪状态的线程是由[Java](http://lib.csdn.net/base/javase)运行时系统的线程调度程序来调度的。

3， 运行(RUNNABLE)：**当线程获得CPU时间后，它才进入运行状态，真正开始执行run()方法。**线程对象创建后，其他线程(比如main线程）调用了该**对象的start()**方法。该状态的线程位于可运行线程池中，等待被线程调度选中**，获取cpu 的使用权 。**  
4. 阻塞(BLOCKED)：阻塞状态是指线程因为某种原因放弃了cpu 使用权，也即让出了cpu timeslice，暂时停止运行。直到线程进入可运行(runnable)状态，才有机会再次获得cpu timeslice 转到运行(running)状态。阻塞的情况分三种：

 5,死亡(DEAD)：线程run()、main() 方法执行结束，或者因异常退出了run()方法则该线程结束生命周期。死亡的线程不可再次复生。



<https://blog.csdn.net/xiaosheng900523/article/details/82964768>



**Java线程具有五中基本状态**

**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();

**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；

2、就绪状态：线程对象创建后，其他线程调用了该对象的start（）方法。该状态的线程位于“可运行的线程池”中，变得可运行，只等待获取CPU的使用权。即在就绪状态的进程除了CPU之外，其他的运行所需资源都已全部获得。

处于就绪状态的场景：

2.1、就绪状态只是说有资格运行，调度程序没有挑选到你，你就永远是可运行状态。

2.2、调用线程的start（）方法，此线程进入就绪状态。

2.3、当前线程sleep（）方法结束，其他线程join（）结束，等待用户输入完毕，某个线程拿到对象锁，这些线程也将进入就绪状态。

2.4、当前线程时间片用完了，调用当前线程的yeild（）方法，当前线程进入就绪方法。

2.5、锁池里的线程拿到对象锁后，进入就绪状态

**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；

**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；运行的线程执行wait（）方法，该线程会释放占用的所有资源，JVM会把该线程放入”等待池“中。进入这个状态后，是不能自动唤醒的，必须依靠其他线程调用notify（）或者notifyAll（）方法才能被唤醒。

**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。

**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

**举个通俗一点的例子来解释上面五种状态，比如上厕所:**

**你平时去商城上厕所，准备去上厕所就是新建状态（new），上厕所要排队，排队就是就绪状态（Runnable），有坑位了，轮到你了，拉屎就是运行状态（Running），你拉完屎发现没有手纸，要等待别人给你送纸过来，这个状态就是阻塞（Blocked），等你上完厕所出来，上厕所这件事情结束了就是死亡状态了。**

**注意：便秘也是阻塞状态，你便秘太久了，别人等不及了，把你赶走，这个就是挂起，还有一种情况，你便秘了，别人等不及了，跟你说你先出去酝酿一下，5分钟后再过来拉屎，这就是睡眠。**

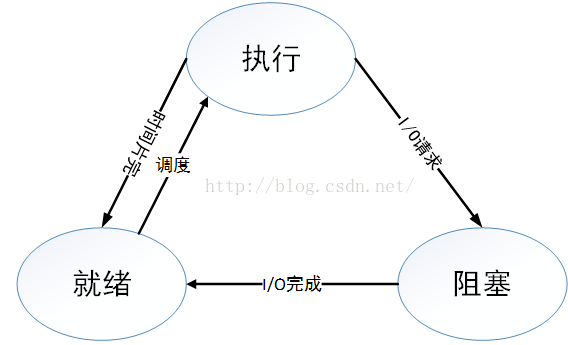
**二. Java多线程的就绪、运行和死亡状态**

就绪状态转换为运行状态：当此线程得到处理器资源；

运行状态转换为就绪状态：当此线程主动调用yield()方法或在运行过程中失去处理器资源。

运行状态转换为死亡状态：当此线程线程执行体执行完毕或发生了异常。

此处需要特别注意的是：**当调用线程的yield()方法时，线程从运行状态转换为就绪状态，**但接下来CPU调度就绪状态中的哪个线程具有一定的随机性，因此，可能会出现A线程调用了yield()方法后，接下来CPU仍然调度了A线程的情况。



**上述的三种状态之间转换分为六种情况：**

1、运行——>就绪：主要是进程占用CPU的时间过长，而系统分配给该进程占用CPU的时间是有限的；在采用抢先式优先级调度算法的系统中,当有更高优先级的进程要运行时，该进程就被迫让出CPU，该进程便由执行状态转变为就绪状态。

2、就绪——>运行：运行的进程的时间片用完，调度就转到就绪队列中选择合适的进程分配CPU

3、运行——>阻塞：正在执行的进程因发生某等待事件而无法执行，则进程由执行状态变为阻塞状态，如发生了I/O请求

4、阻塞——>就绪:进程所等待的事件已经发生，就进入就绪队

**以下两种状态是不可能发生的：**

1、阻塞——>运行：即使给阻塞进程分配CPU，也无法执行，操作系统在进行调度时不会从阻塞队列进行挑选，而是从就绪队列中选取

2、就绪——>阻塞：就绪态根本就没有执行，谈不上进入阻塞态。

8，[Java多线程中start()和run()的区别](https://www.cnblogs.com/sunflower627/p/4816821.html)

多线程原理：相当于玩游戏机，只有一个游戏机（cpu），可是有很多人要玩，于是，start是排队！等CPU选中你就是轮到你，你就run（），当CPU的运行的时间片执行完，这个线程就继续排队，等待下一次的run（）。

**调用start（）后，线程会被放到等待队列，等待CPU调度，并不一定要马上开始执行，只是将这个线程置于可动行状态。然后通过JVM，线程Thread会调用run（）方法，执行本线程的线程体。先调用start后调用run，这么麻烦，为了不直接调用run？**就是为了实现多线程的优点，没这个start不行。

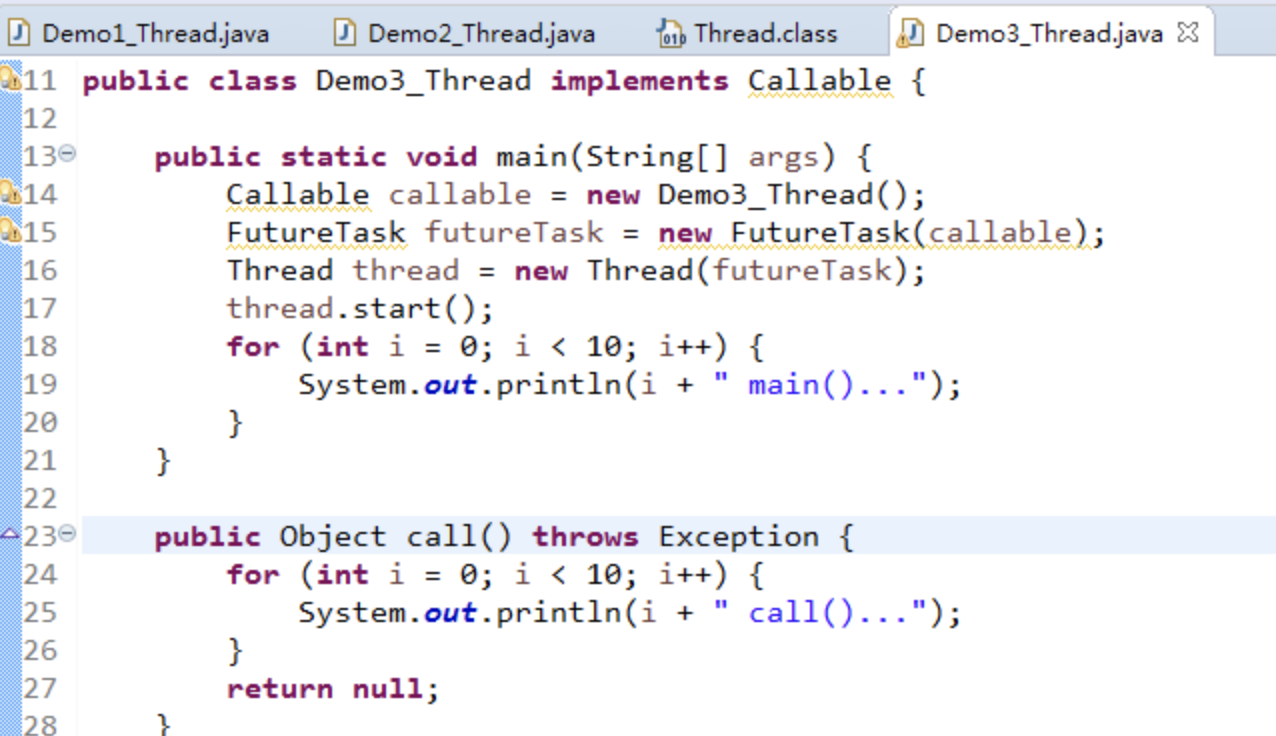
1. start（）方法来启动线程，真正实现了多线程运行。这时无需等待run方法体代码执行完毕，可以直接继续执行下面的代码；通过调用Thread类的start()方法来启动一个线程， 这时此线程是处于就绪状态， 并没有运行。 **然后通过此Thread类调用方法run()来完成其运行操作的， 这里方法run()称为线程体，它包含了要执行的这个线程的内容， Run方法运行结束， 此线程终止。然后CPU再调度其它线程。**  
   2.run（）方法当作普通方法的方式调用。程序还是要顺序执行，**要等待run方法体执行完毕后，才可继续执行下面的代码；** 程序中只有主线程——这一个线程， 其程序执行路径还是只有一条， 这样就没有达到写线程的目的。

9，[java多线程开启的三种方式](https://www.cnblogs.com/xiaostudy/p/9800601.html)

1. **继承Thread类，**新建一个当前类对象，并且运行其start()方法

**2，实现Runnable接口**，然后新建当前类对象，接着新建Thread对象时把当前类对象传进去，最后运行Thread对象的start()方法

**3，使用Callable和Future创建线程，**实现Callable接口，新建当前类对象，在新建FutureTask类对象时传入当前类对象，接着新建Thread类对象时传入FutureTask类对象，最后运行Thread对象的start()方法，**调用FutureTask对象的get()方法来获得子线程执行结束后的返回值，**这种方式可以获得线程执行完之后的返回值。



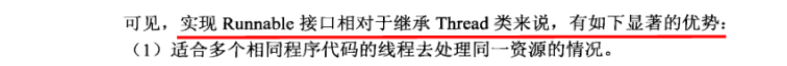
**Callable和Runnable的区别：**

返回值：Callable有返回值，Runnable没有返回值

1. java 多线程 继承Thread和实现Runnable的区别

3、继承Thread实现的模式是 **定义多个线程,各自完成各自的任务.**

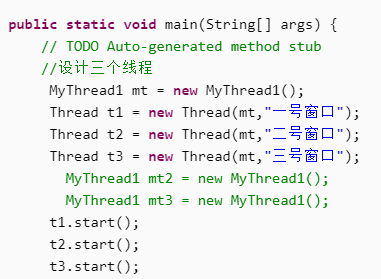
4、实现Runnable实现的模式是 **定义多个线程,实现一个任务.**

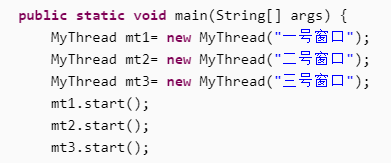


总结：**一个是多个线程分别完成自己的任务，一个是多个线程共同完成一个任务**。其实，在实现一个任务用多个线程来做也可以用继承Thread类来实现，只是比较麻烦，一般我们用实现Runnable接口来实现。

**继承Thread**: 线程代码存放Thread子类run方法中。  
**实现Runnable**，线程代码存在接口的子类的run方法。

**实现Runnable接口相对于继承Thread类来说，有如下的显著优势：**1 .适合多个相同代码的线程去处理同一个资源的情况  
2.可以避免由于java的单继承特性带来的局限  
3.增强了程序的健壮性，代码能够被多个线程共享，代码与数据时独立的





**1，为什么会出现这种结果呐。我们不妨做个比喻，其实刚的程序，**

1，继承Thread类的，我们相当于拿出三件事即三个卖票10张的任务分别分给三个窗口，他们各做各的事各卖各的票各完成各的任务，因为MyThread继承Thread类，所以在new MyThread的时候在创建三个对象的同时创建了三个线程；

2，实现Runnable的， 相当于是拿出一个卖票10张得任务给三个人去共同完成，new MyThread相当于创建一个任务，然后实例化三个Thread，创建三个线程即安排三个窗口去执行。

**2，为什么要将Runnable接口的子类对象传递给Thread的构造函数。**  
因为，自定义的run方法所属的对象是Runnable接口的子类对象。  
所以要让线程去指定指定对象的run方法。就必须明确该run方法所属对象

1. Java线程的start方法能调用多次吗?
2. 一个线程对象只能调用一次start方法.从new到等待运行是单行道,所以如果你对一个已经启动的线程对象再调用一次start方法的话,会产生:IllegalThreadStateException异常，可以被重复调用的是run()方法。**调试发现，第一个次运行start()方法时，threadStatus是0，此时if条件不满足，继续执行，会将当前线程添加到线程组中去执行。第二次运行start()方法时，threadStatus变成了2，if条件满足，于是抛出了java.lang.IllegalThreadStateException异常。**
3. 什么是线程安全和线程不安全？

**1，简单的说，**加锁的就是线程安全的，不加锁的是线程不安全的

**2，线程安全:** 当多线程访问时，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。

**3，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。

**4，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。

**如果你的代码在多线程下执行和在单线程下执行永远都能获得一样的结果，那么你的代码就是线程安全的。**

**//线程安全问题产生的原因：**

/\*1，多个线程在操作共享数据。

2，操作共享数据的线程代码有多条，

当一个线程在执行操作共享数据的多条代码过程中，其他线程参与了进来，就会导致线程安全问题的产生\*/

**//解决思路**

/\* 1，就是将多条操作共享数据的线程代码封装起来，当有线程在执行这些代码的时候，

其他线程不可以参与运算。必须要当前线程把这些代码都执行完毕后，其他线程才可以参与运算\*/

1. 什么是自旋锁？

**自旋锁（spinlock）：**是指当一个线程在获取锁的时候，**如果锁已经被其它线程获取，那么该线程将循环等待，然后不断的判断锁是否能够被成功获取，直到获取到锁才会退出循环。**

获取锁的线程一直处于活跃状态，但是并没有执行任何有效的任务，使用这种锁会造成[busy-waiting](https://en.wikipedia.org/wiki/Busy_waiting)。

自旋锁：线程获取锁的时候，**如果锁被其他线程持有，则当前线程将循环等待，直到获取到锁。**

1. [java中的sleep()和wait()的区别](https://www.cnblogs.com/hongten/p/hongten_java_sleep_wait.html)

1，对于**sleep()方法，我们首先要知道该方法是属于Thread类中的**。而wait()方法，则是属于Object类中的。

2，**sleep()方法导致了程序暂停执行指定的时间，让出cpu该其他线程**，但是他的监控状态依然保持者，当指定的时间到了又会自动恢复运行状态。

3，**在调用sleep()方法的过程中，线程不会释放对象锁**。而当调用wait()方法的时候，线程会放弃对象锁，进入等待此对象的等待锁定池，只有针对此对象调用notify()方法后本线程才进入对象锁定池准备，获取对象锁进入运行状态。

4，wait，notify和notifyAll只能在同步控制方法或者同步控制块里面使用，而sleep可以在任何地方使用(使用范围)

5，**sleep必须捕获异常，而wait，notify和notifyAll不需要捕获异常**

1. yield()和join()

1，**yield方法暂停当前正在执行的线程对象**。  yield()方法是停止当前线程，让同等优先权的线程或更高优先级的线程有执行的机会。如果没有的话，那么yield()方法将不会起作用，并且由可执行状态后马上又被执行。

2，**join方法是用于在某一个线程的执行过程中调用另一个线程执行，等到被调用的线程执行结束后，再继续执行当前线程**。如：t.join();//主要用于等待t线程运行结束，若无此句，main则会执行完毕，导致结果不可预测。

#### 14）如果你提交任务时，线程池队列已满。会时发会生什么？

这个问题问得很狡猾，许多程序员会认为该任务会阻塞直到线程池队列有空位。事实上如果一个任务不能被调度执行那么ThreadPoolExecutor’s submit()方法将会抛出一个RejectedExecutionException异常。

1. [多线程和异步的异同和使用场景](https://www.cnblogs.com/youmingkuang/p/10006584.html)

1**、异步**适用于IO操作（文件，网络数据修改，数据库操作、Web Service、HttpRequest以及.Net Remoting等跨进程的调用）

2、多线程适用于需要长时间CPU运算的场合（例如耗时较长的图形处理和算法执行）

**（2）异步调用**

   异步调用是为了解决同步调用可能出现阻塞，导致整个流程卡住而产生的一种调用方式。类A的方法方法a()通过新起线程的方式调用类B的方法b()，代码接着直接往下执行，这样无论方法b()执行时间多久，都不会阻塞住方法a()的执行。但是这种方式，由于方法a()不等待方法b()的执行完成，在方法a()需要方法b()执行结果的情况下（视具体业务而定，有些业务比如启异步线程发个微信通知、刷新一个缓存这种就没必要），必须通过一定的方式对方法b()的执行结果进行监听。在Java中，可以使用Future+Callable的方式做到这一点。

6、当一个线程进入一个对象的synchronized方法A之后，其它线程是否可进入此对象的synchronized方法B？

不能。其它线程只能访问该对象的非同步方法，同步方法则不能进入。因为非静态方法上的synchronized修饰符要求执行方法时要获得对象的锁，如果已经进入A方法说明对象锁已经被取走，那么试图进入B方法的线程就只能在等锁池（注意不是等待池哦）中等待对象的锁。

25、线程的sleep()方法和yield()方法有什么区别？

sleep()方法给其他线程运行机会时不考虑线程的优先级，因此会给低优先级的线程以运行的机会；yield()方法只会给相同优先级或更高优先级的线程以运行的机会；线程执行sleep()方法后转入阻塞（blocked）状态，而执行yield()方法后转入就绪（ready）状态；sleep()方法声明抛出InterruptedException，而yield()方法没有声明任何异常；sleep()方法比yield()方法（跟操作系统CPU调度相关）具有更好的可移植性。