# JAVA多线程

## 线程与进程

1. 线程：

进程中负责程序执行的执行单元。

线程本身依靠程序进行运行。

线程是程序中的顺序控制流，只能使用分配给程序的资源和环境。

1. 进程：

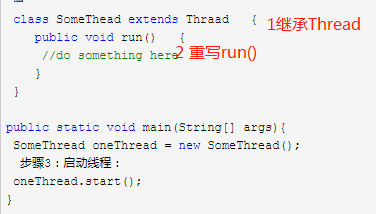
执行中的程序。

一个进程至少包括一个线程。

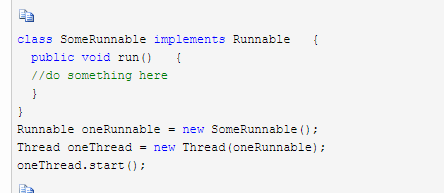
1. 线程比进程的优势有：
2. 进程之间不能共享数据，线程可以。
3. 系统创建进程需要为该进程重新分配系统资源，故创建线程代价比较 小。

## 创建线程和启动

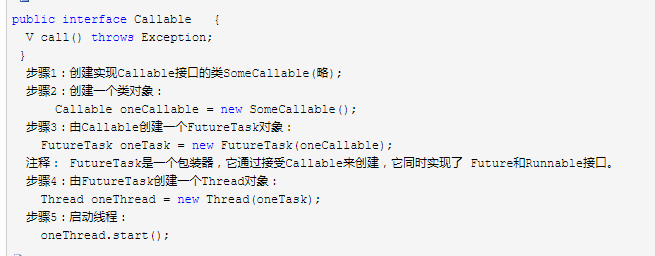
1. 继承Thread类创建线程类



1. 实现Runnable接口创建线程类



1. 通过Callable和Future创建线程
2. 创建Callable接口的实现类，并实现call()方法，该call()方法将作为线 程执行体，并且有返回值。
3. 创建Callable实现类的实例，使用FutureTask类来包装Callable对象， 该FutureTask对象封装了该Callable对象的call()方法的返回值。
4. 使用FutureTask对象作为Thread对象的target创建并启动新线程。
5. 调用FutureTask对象的get方法来获得子线程执行结束后的返回值。



## 线程的生命周期

1. 新建状态

用new关键字和Thread类或其子类建立一个线程对象后，该线程对象就处于新生状态。处于新生状态的线程有自己的内存空间，通过调用start方法进入就绪状态。

注意：不能对已经启动的线程再次调用start()方法，否则会出现异常。

1. 就绪状态

处于就绪状态的线程已经具备了运行条件，但还没与分配到CPU，处于线程就绪队列(尽管是采用队列形式，事实上，把它称为可运行池而不是可运行队列。因为cpu的调度不一定是按照先进先出的顺序来调度的)，一旦获得cpu，线程就进入运行状态并自动调用自己的run()方法。

1. 运行状态

处于运行状态的线程最为复杂，他可以变为阻塞状态，就绪状态和死亡状态。

失去cpu资源后，会从运行状态变为就绪状态。重新等待系统分配资源。也可以运行中的线程调用yield()方法，他就会让出cpu资源，再次变为就绪状态。让运行状态变为阻塞状态的方法：

1. 线程调用sleep方法主动放弃所占用的系统资源。
2. 线程调用一个阻塞式IO方法，在该方法返回之前，该线程被阻塞。
3. 线程在等待某个通知(notify)

线程run()方法执行完，或者被强制性终止例如抛出异常、stop()、destory()会让线程从运行状态变为死亡状态。

1. 阻塞状态

处于运行状态的线程在某些情况下，如执行了sleep()方法，或等待I/O设备等资源，将让出CPU并暂时停止自己的运行，进入阻塞状态。

在阻塞状态的线程不能进入就绪队列。只有当引起阻塞的原因消除时便 会转入就绪状态，重新导就绪队列中等待。被系统选中后从原来停止的 位置继续运行。

1. 死亡状态

当run()执行完，或被强制性中指。就认为他死去，也许该线程对象是活的，但是，他已经不是一个单独执行的线程。如果在一个死去的线程上调用start()方法，会抛出异常。

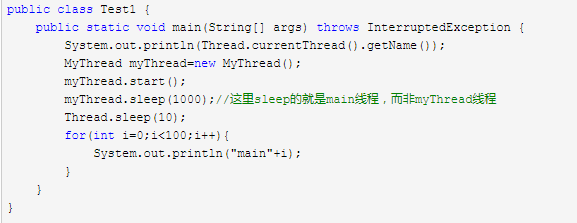
## 线程管理

1. 线程睡眠 sleep

如果我们需要让当前正在执行的线程暂停一段时间，并进入阻塞状态，则可以通过Thread的sleep方法。

注：

1. sleep是静态方法，只用Thread.sleep()调用就好了。



1. sleep()无法实现精准调度。
2. 线程让步 yield

yield()方法和sleep()方法有点相似，它也是Thread类提供的一个静态的方法，也可以让当前执行的线程暂停，让出cpu资源给其他线程。但该方法执行不会使线程进入阻塞状态，而是进入就绪状态。

sleep()和yield()区别：

1. Sleep方法暂停当前线程后，会进入阻塞状态，只有当睡眠时间到了， 才会转入就绪状态。而yield方法调用后，是直接进入就绪状态，所以有 可能刚进入就绪状态，又被调度到运行状态。
2. Sleep方法声明抛出了IneruptedException，所以调用该方法要捕获该 异常。而yield方法没有异常抛出。
3. Sleep方法比yield方法有更好的可移植性，通常不要依靠yield方法 来控制并发线程的执行。
4. 线程合并 join