Java并发学习

**线程管理**

# 一、实现线程的两种方式

## 1 扩展javalangThread类

继承该类后,覆盖run方法.通过调用start()方法调用线程. start()方法的调用后并不是立即执行多线程代码，而是使得该线程变为可运行态（Runnable），什么时候运行是由操作系统决定的。跟操作系统的cpu资源(时间片)分配相关. start()方法不能重复调用.

重复调用会报错但是线程会正常运行

Exception in thread "main" java.lang.IllegalThreadStateException  
    at java.lang.Thread.start(Unknown Source)  
    at com.multithread.learning.Main.main(Main.java:31)

想重复调用的话可以用第二种方法,接口编程是java亲儿子啊,哈哈。

## 2 [实现javalangRunnable接口](http://blog.csdn.net/evankaka/article/details/44153709#t1)

Thread2类通过实现Runnable接口，使得该类有了多线程类的特征。run（）方法是多线程程序的一个约定。所有的多线程代码都在run方法里面。Thread类实际上也是实现了Runnable接口的类。在启动的多线程的时候，需要先通过Thread类的构造方法Thread(Runnable target) 构造出对象，然后调用Thread对象的start()方法来运行多线程代码。这边可以用new Thread(Runnable target).start重复调用。其实是重新启动了新的线程（性能的话没有做测试）。

## 3两者的区别

如果一个类继承Thread，则不适合资源共享。但是如果实现了Runable接口的话，则很容易的实现资源共享。也就是说第一种方式资源是相互独立的，而runable种资源是共享的，这其实也能解释为啥继承thead类的方式中start()为什么不能重复调用而实现Runnable则可以。

好像大家都比较推荐后者。

# 二、线程状态



1、新建状态（New）：新创建了一个线程对象。

2、就绪状态（Runnable）：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线程池中，变得可运行，等待获取CPU的使用权。

3、运行状态（Running）：就绪状态的线程获取了CPU，执行程序代码。

4、阻塞状态（Blocked）：阻塞状态是线程因为某种原因放弃CPU使用权，暂时停止运行。直到线程进入就绪状态，才有机会转到运行状态。阻塞的情况分三种：

（一）、等待阻塞：运行的线程执行wait()方法，JVM会把该线程放入等待池中。

（二）、同步阻塞：运行的线程在获取对象的同步锁时，若该同步锁被别的线程占用，则JVM会把该线程放入锁池中。

（三）、其他阻塞：运行的线程执行sleep()或join()方法，或者发出了I/O请求时，JVM会把该线程置为阻塞状态。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。

# 三、线程调度

线程的调度

1、调整线程优先级：Java线程有优先级，优先级高的线程会获得较多的运行机会。

 （这边有比较多的争议，我觉得java的多线程是基于操作系统的多线程机制实现的，所以谁先执行估计还是要操作系统来决定）

Java线程的优先级用整数表示，取值范围是1~10，Thread类有以下三个静态常量：

static int MAX\_PRIORITY

          线程可以具有的最高优先级，取值为10。

static int MIN\_PRIORITY

          线程可以具有的最低优先级，取值为1。

static int NORM\_PRIORITY

          分配给线程的默认优先级，取值为5。

Thread类的setPriority()和getPriority()方法分别用来设置和获取线程的优先级。

每个线程都有默认的优先级。主线程的默认优先级为Thread.NORM\_PRIORITY。

线程的优先级有继承关系，比如A线程中创建了B线程，那么B将和A具有相同的优先级。

JVM提供了10个线程优先级，但与常见的操作系统都不能很好的映射。如果希望程序能移植到各个操作系统中，应该仅仅使用Thread类有以下三个静态常量作为优先级，这样能保证同样的优先级采用了同样的调度方式。

2、线程睡眠：Thread.sleep(long millis)方法，使线程转到阻塞状态。millis参数设定睡眠的时间，以毫秒为单位。当睡眠结束后，就转为就绪（Runnable）状态。sleep()平台移植性好。

3、线程等待：Object类中的wait()方法，导致当前的线程等待，直到其他线程调用此对象的 notify() 方法或 notifyAll() 唤醒方法。这个两个唤醒方法也是Object类中的方法，行为等价于调用 wait(0) 一样。

4、线程让步：Thread.yield() 方法，暂停当前正在执行的线程对象，把执行机会让给相同或者更高优先级的线程。

5、线程加入：join()方法，等待其他线程终止。在当前线程中调用另一个线程的join()方法，则当前线程转入阻塞状态，直到另一个进程运行结束，当前线程再由阻塞转为就绪状态。

6、线程唤醒：Object类中的notify()方法，唤醒在此对象监视器上等待的单个线程。如果所有线程都在此对象上等待，则会选择唤醒其中一个线程。选择是任意性的，并在对实现做出决定时发生。线程通过调用其中一个 wait 方法，在对象的监视器上等待。 直到当前的线程放弃此对象上的锁定，才能继续执行被唤醒的线程。被唤醒的线程将以常规方式与在该对象上主动同步的其他所有线程进行竞争；例如，唤醒的线程在作为锁定此对象的下一个线程方面没有可靠的特权或劣势。类似的方法还有一个notifyAll()，唤醒在此对象监视器上等待的所有线程。

**线程同步**

1. synchronized关键字的作用域有二种：   
   1）是某个对象实例内，synchronized aMethod(){}可以防止多个线程同时访问这个对象的synchronized方法（如果一个对象有多个synchronized方法，只要一个线程访问了其中的一个synchronized方法，其它线程不能同时访问这个对象中任何一个synchronized方法）。这时，不同的对象实例的synchronized方法是不相干扰的。也就是说，其它线程照样可以同时访问相同类的另一个对象实例中的synchronized方法；   
   2）是某个类的范围，synchronized static aStaticMethod{}防止多个线程同时访问这个类中的synchronized static 方法。它可以对类的所有对象实例起作用。   
     
   2、除了方法前用synchronized关键字，synchronized关键字还可以用于方法中的某个区块中，表示只对这个区块的资源实行互斥访问。用法是: synchronized(this){/\*区块\*/}，它的作用域是当前对象；   
     
   3、synchronized关键字是不能继承的，也就是说，基类的方法synchronized f(){} 在继承类中并不自动是synchronized f(){}，而是变成了f(){}。继承类需要你显式的指定它的某个方法为synchronized方法；