sleep

1.sleep是帮助其他线程获得运行机会的方法(让出CPU时间片),但是如果当前线程获取的的有锁,sleep不会让出锁

2.线程到时间自动苏醒,并返回到可运行状态,不是运行状态

3.优先线程的调用,现在苏醒之后,并不会立即执行,

4.sleep是静态方法,只能控制当前正在运行的线程.

yield

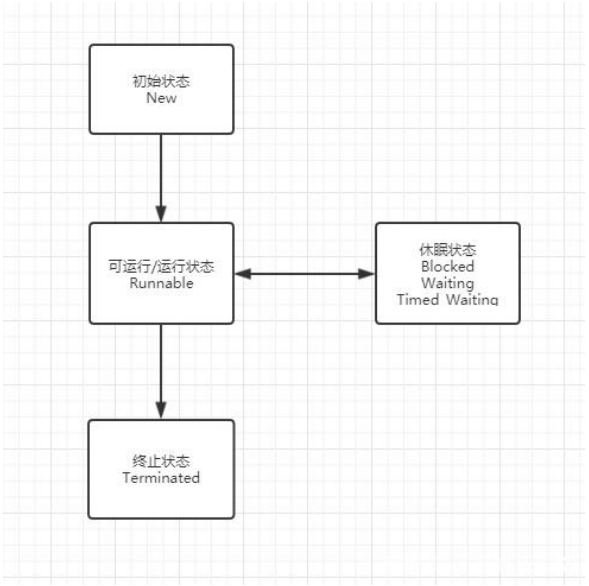
1.线程让步,会把自己的时间片让出,然后和其他线程一起抢时间片

2.不会释放锁

# 1.什么是线程

线程是进程内的执行单元,是轻量级进行,是程序执行的最小单位,线程之间切换调度的成本比较低

线程的声明周期:




通过wait()方法等待的线程在等待notify(),而通过join()方法等的线程则会等待目标线程的终止

# 2.初始线程

通过start()方法开启新线程

## 2.2终止线程

正常线程执行完毕就会结束,无须手动关闭,

stop()方法停止线程是有坏处的,过于暴力,会直接终止线程,并立即释放这个线程持有的锁,而锁是保证对象一致性的,不能保证数据的完整性,线程中断是更强大的支持,需要注意:

## 2.3.线程中断

1.线程中断不会使线程立即退出,而是给线程发送一个通知,告知目标线程,有人希望你退出了,,至于何时退出,则完全由目标线程自行决定.如果中断后,线程立即退出,则和使用stop()方法一样,又会遇到老问题.

```
public void Thread.interrupt()  //中断线程
public boolean Thread.isInterrupted() //判断是否被中断
public static boolean Thread.intertupted() //判断是否被中断,并清除当前中断状态
```

Thread.intertupt()方法是一个实例方法,通知目标线程中断,也就是设置中断标志位.中断标志位表示当前线程已经被中断了.Thread.isInterrupted()方法也是实例方法,检查中断标志位,判断是否被中断.

需要增加中断处理代码,否则中断没有效果.

如果在循环体中,出现了类似于wait()方法,或者sleep()方法这样的操作,则只能通过中断来识别了.

Thread.sleep()方法会让当前线程休眠若干时间,它会抛出一个InterruptedException中断异常,这个异常不是运行时异常,也就是程序必须捕获处理它,当线程sleep()休眠时,如果被中断,这个异常就会产生.

2.sleep()

最好处理中断的异常,Thread.sleep()方法由于中断而抛出异常,此时,它会清除中断标记,如果不加处理,那么下一次循环开始,就无法捕获这个中断,

## 2.4 等待(wait)和通知(notify)

目的:支持多线程之间的协作

等待wait()和notify()方法,这两个方法不是在Thread中,而是Object类,这意味着任何对象都可以调用这两个方法.

线程A调用了obj.wait()方法,那么线程A就会停止继续执行,转为等待状态.会一直等待到其他线程调用了obj.notify()方法为止.

obj.wait()方法不能随意调用,必须在synchronized语句中,无论是wait()还是notify()方法都需要首先获得目标对象的一个监视器,

## 2.5 挂起(suspend)和继续执行(resume)

被挂起的线程,必须要等到resume()方法操作后,才能继续执行.已经被标注为废弃.

不推荐使用suspend()方法去挂起线程是因为suspend()导致线程暂停的同时,不释放任何锁资源.此时,其他任何线程想要访问被它占用的锁时,都会被牵连,导致无法继续运行.直

到对应的线程上执行了resume()方法操作,

对于被挂起的线程,从它的线程状态看,居然还是Runnable,这也会严重影响我们对系统当前状态的判断.

# 4.线程的基本操作

## 1.挂起(suspend)和继续(resume)

-suspend()不会释放锁

-如果加锁发生在resume()之前,则死锁发生

## 2.等待线程结束(join)和谦让(yeild)

## 3.守护线程

-在后台默默的完成一些系统性的服务,比如垃圾回收,JIT线程就可以理解为守护线程

-当一个Java应用内,只有守护线程时,Java虚拟机就会自然退出

## 4.基本的线程同步操作

1.synchromized

-指定加锁对象,对给定对象加锁,进入同步代码前要获得给定对象的锁

-直接作用于实例方法:相当于对当前实例加锁,进入同步代码前要获得当前实例的锁

-直接作用于静态方法,相当于对当前类加锁,进入同步代码前需要获得当前类的锁

2.Object.wait(),Objcet.notify();