Runnable接口与Thread类

# Runnable接口

## Runnable接口

**@FunctionalInterface**

public interface Runnable

Runnable接口存在于**java.lang**包中，是一个功能接口(**Functional Interface**)。

## 功能

Runnable接口是一个功能接口，线程中执行的实例都必须实现该接口，且该实例必须实现无参数的run()方法。

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. The class must define a method of no arguments called run.

Functional Interface:This is a functional interface and can therefore be used as **the assignment target for a lambda expression or method reference**.

This interface is designed **to provide a common protocol for objects** that wish to execute code while they are active. For example, Runnable is implemented by class **Thread**. Being active simply means that a thread has been started and has not yet been stopped.

该接口主要是规定了线程执行任务的标准。

In addition, **Runnable provides the means for a class to be active while not subclassing Thread**. A class that implements Runnable can run without subclassing Thread by instantiating a Thread instance and passing itself in as the target. In most cases, the Runnable interface should be used if you are only planning to override the run() method and no other Thread methods. This is important because classes should not be subclassed unless the programmer intends on modifying or enhancing the fundamental behavior of the class.

要想创建一个新的线程，必须**直接或间接的实现**Runnable接口且重写其中的run方法，其中**run方法**是用来封装线程所要的任务。

## 方法

Runnable接口中只有一个方法：**run()**。

**void run()**

When an object implementing interface Runnable **is used to create a thread**, starting the thread causes the object's run method to be called in that separately executing thread. The general contract of the method run is that it may take any action whatsoever.

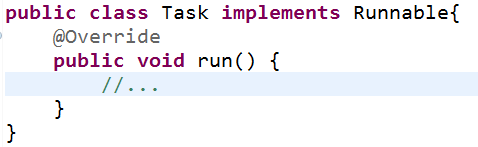
作用：run()方法用于**封装新线程要执行的任务**。

## 实现类

**Runnable接口**有许多直接实现类，其中最重要的是**Thread类**。

All Known Implementing Classes:AsyncBoxView.ChildState, ForkJoinWorkerThread, **FutureTask**, RenderableImageProducer, SwingWorker, **Thread**, **TimerTask**

**另外，**创建线程的常用方式，创建一个封装任务的Task类，即实现Runnable接口的一个类。



## 继承的子接口

All Known Subinterfaces: **RunnableFuture<V>**, **RunnableScheduledFuture<V>。**

# Thread类(静态代理设计模式)

## 继承关系：

public class Thread extends **Object** implements **Runnable**

Thread类：存在于**java.lang包**中，直接继承于Object，实现了java.lang.Runnable接口，并覆写了Runnable接口中的**run()方法**。

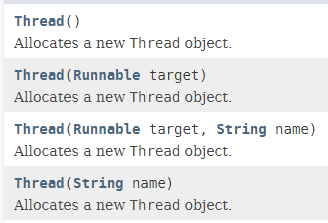
## 构造方法

一个Thread线程初始化参数：①线程名字String name;②**线程任务:Runnable task**；③线程所在组：ThreadGroup；④栈大小：long stackSize。

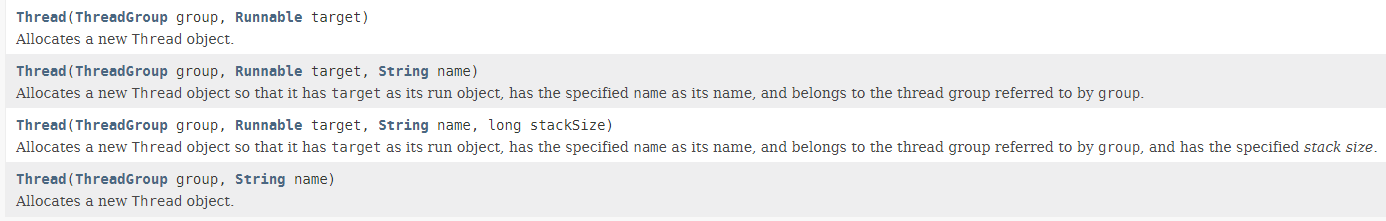
注意：线程是存在**栈**中的。

### 常用的

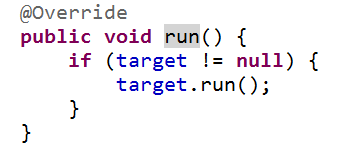
创建一个线程，Runnable task一般不可缺少；给线程命名有利于辨别该线程。



### 规定线程组ThreadGroup和栈大小



## Thread类中的run()方法

如果创建Thread对象时没有传入Runnable的实现类对象，则该Thread对象的run方法什么也不执行。所以，要创建Thread对象，一般需要传入Runnable的实现类对象。

结论：Thread类默认实现的run方法相当于是空的。可以通过继承Thread类或传入Runnable实现对象来覆盖Thread中默认的run()方法。

## Thread中线程的优先级Priority

**对线程优先级的理解**：对线程赋予不同的优先级，只是优先级大的线程执行到的**概率大**些，优先级小的线程执行到的**概率小**些。但是，优先级大的线程不一定比优先级小的线程先执行，优先级低的线程有可能先执行。**优先级并不能代表执行的顺序，只是一种概率大小的含义，优先级高的线程先执行的概率较大，但是低优先级的线程也有可能先执行**。

**Java虚拟机**中通过**线程调度器**来管理线程，调度器把线程分成10个等级；Thread中规定线程优先级范围为1—10；默认为5。把优先级为1、5、10封装成了字段。线程（包括主线程）被创建时，默认的优先级级别都是5。

static int **MAX\_PRIORITY 值为10**

The maximum priority that a thread can have.

static int **MIN\_PRIORITY 值为1**

The minimum priority that a thread can have.

static int **NORM\_PRIORITY 值为5**

The default priority that is assigned to a thread.

Thread中也定义了get和set方法，获取和设置线程的优先级：

**int getPriority()** Returns this thread's priority.

**void setPriority(int newPriority)** Changes the priority of this thread.

## Thread的静态方法

**一个类的静态方法**体现了该类的具有的共性，获取该类的共同性质。

Thread类的静态方法有：

### static Thread currentThread() 获取当前的线程对象。

Returns a reference to **the currently executing thread object**.

### static int activeCount() 获取当前线程组中active的线程个数，返回int值。

Returns an estimate of the number of active threads in the current thread's thread group and its subgroups.

### static boolean holdsLock(Object obj) 判断当前线程是否持有obj锁

Returns true **if and only if** the current thread holds the monitor lock on the specified object.

### static boolean interrupted() 判断当前线程是否已经中断。

Tests whether the current thread has been interrupted.

### static void yield()

A hint to the scheduler that the current thread is willing to yield its current use of a processor.

Thread.yield() 线程让步。

## Thread.yield()方法详解

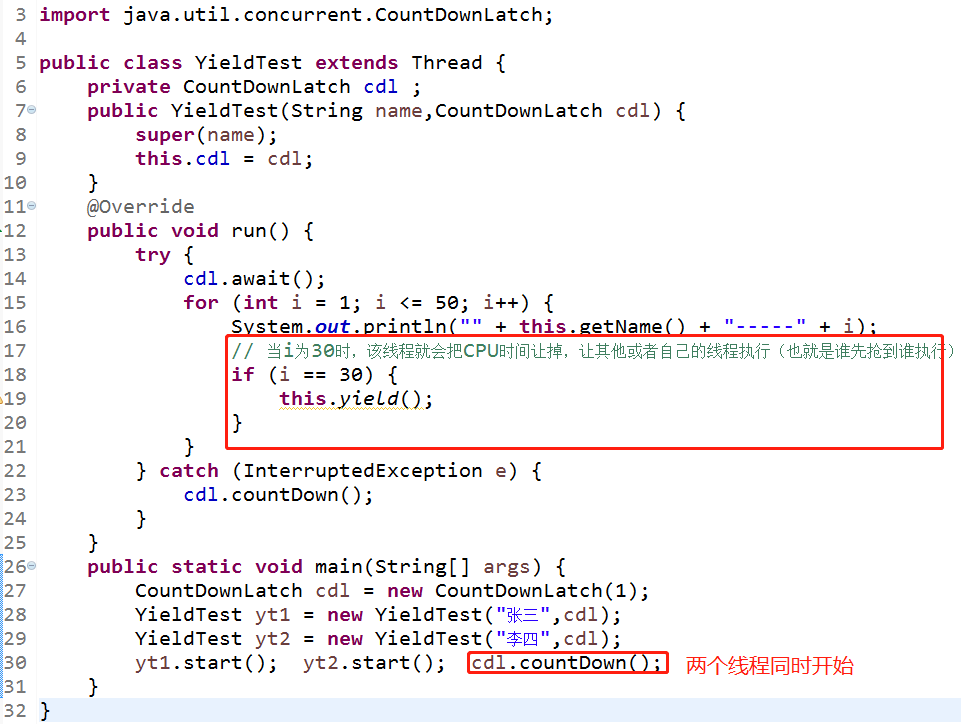
**yield**美[jild]vt.屈服，投降，让步; 不再反对;vi.放弃，屈服; 退让，退位;

**使当前线程从执行状态（运行状态）变为可执行态（就绪状态）**。cpu会从众多的可执行态里选择，也就是说，当前也就是刚刚的那个线程还是有可能会被再次执行到的，并不是说一定会执行其他线程而该线程在下一次中不会执行到了。

Java线程中有一个Thread.yield( )方法，很多人翻译成**线程让步**。顾名思义，就是说当一个线程使用了这个方法之后，它就会把自己CPU执行的时间让掉，让自己或者其它的线程运行。

打个比方：现在有很多人在排队上厕所，好不容易轮到这个人上厕所了，突然这个人说：“我要和大家来个竞赛，看谁先抢到厕所！”，然后所有的人在同一起跑线冲向厕所，有可能是别人抢到了，也有可能他自己有抢到了。我们还知道线程有个优先级的问题，那么手里有优先权的这些人就一定能抢到厕所的位置吗? 不一定的，他们只是概率上大些，也有可能没特权的抢到了。

示例：



运行结果：

第一种情况：李四（线程）当执行到30时会CPU时间让掉，这时张三（线程）抢到CPU时间并执行。

第二种情况：李四（线程）当执行到30时会CPU时间让掉，这时李四（线程）抢到CPU时间并执行。

## Thread.join()方法

**join方法:线程联合**。作用：当前运行的线程进入阻塞状态，等待**调用join方法的线程**运行完毕后，再继续执行，注意主线程只等调用join方法的线程运行完毕，调用join方法也会抛出异常。临时加入一个线程可以使用join方法。

## 守护线程daemon

**守护线程**： **setDaemon(boolean b)**方法可以将此线程设置为守护线程或用户线程。This method must be invoked before the thread is started.

守护线程与一般线程运行、等待都一样，只不过结束的时候不一样，只要一般线程消失，守护线程会自动消失，无论其处于何种状态。当所有的线程都是守护线程时，此时程序会自动退出。

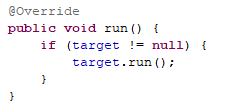
## Thread代理：典型的静态代理设计模式

真实角色(真正的线程任务)：class **Task** implements Runnable

代理角色：class **Thread** implements Runnable

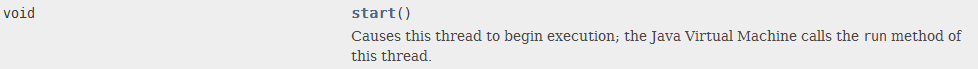
Task 与 Thread都实现了**Runnable**接口。

通过Thread thread = new Thread(**new Task()**);将线程任务task交给Thread代理thread执行，Thread就是作为Task的代理角色。  
Thread类run()方法的源码：

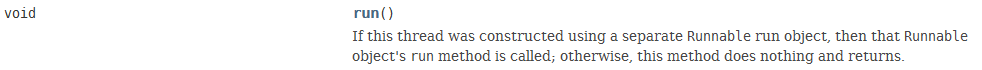


## 一般方法

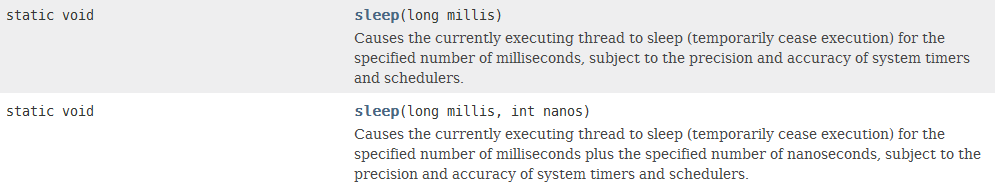
### start方法：启动此线程。



### run方法：封装线程执行的任务。



### sleep方法：



### checkAccess方法：



### getXxxx方法：

1. **getId方法**：获取线程的ID。



1. **getName()方法**：获取线程的名字。



1. **getPriority()方法**：获取线程的优先级。



### setXxx方法：

1. **setName()**



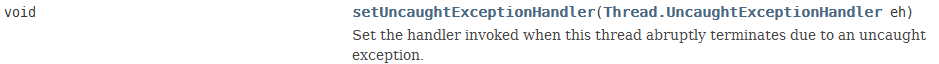
1. **setPriority()方法**: **设置优先级，数值只有1—10；默认的值是5.**

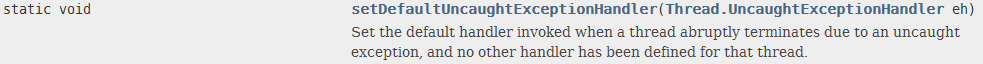


1. **setDaemon(Boolean on)**：**守护线程**。



1. 其他的setXxxx（）



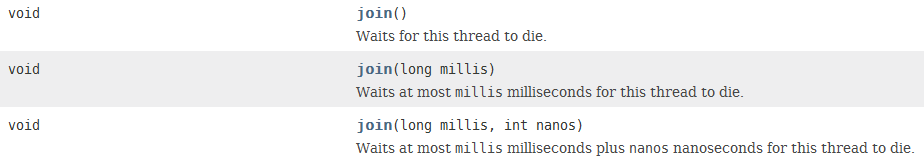




### interrupt（）：中断此线程。



### join（）方法：



### 判断方法：

1. **interrupted**：静态方法，判断此线程是否是被中断了。



1. **isInterrupted**：



1. isAlive：

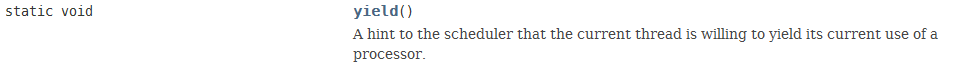


1. isDaemon():是否是后台运行。

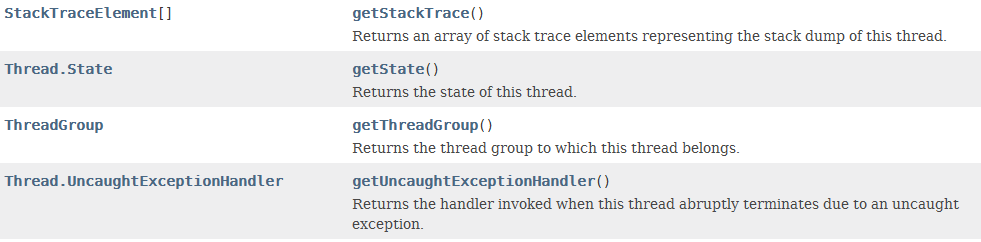
**daemon ['diːmən]n. 守护进程；后台程序**



### yield：释放cpu执行资格，线程让步(让步后仍参与竞争)。



### getXxxx（）：



### toString:返回线程的表现形式，包括线程名、优先级和线程组。

