Semaphore类

# Semaphore-信号量--介绍

## Semophore信号量的包及继承关系

单词介绍： **semaphore** 英[ˈseməfɔ:(r)] 美[ˈsɛməˌfɔr, -ˌfor]

n. 臂板信号系统，（铁道）臂板信号装置; vt. **发出信号，打旗语**

public class **Semaphore** extends **Object** implements **Serializable**

**Semaphore**类存在于java.util.concurrent包中，直接继承于Object类，实现了Serializable接口。

## 功能介绍

**Semaphore**又称信号量，是操作系统中的一个概念，在**Java并发编程**中，信号量控制的是**线程并发的数量**。Semaphore类用于**控制并发线程数**。

public **Semaphore**(int permits)；其中参数permits就是允许**同时运行的线程数目**。

**Semaphore（信号量）**是**用来控制同时访问特定资源的线程数量**，它通过协调各个线程，以保证合理的使用公共资源。很多年以来，我都觉得从字面上很难理解Semaphore所表达的含义，只能把它比作是控制流量的红绿灯，比如XX马路要限制流量，只允许同时有一百辆车在这条路上行使，其他的都必须在路口等待，所以前一百辆车会看到绿灯，可以开进这条马路，后面的车会看到红灯，不能驶入XX马路，但是如果前一百辆中有五辆车已经离开了XX马路，那么后面就允许有5辆车驶入马路，这个例子里说的车就是线程，驶入马路就表示线程在执行，离开马路就表示线程执行完成，看见红灯就表示线程被阻塞，不能执行。

**A counting semaphore**. Conceptually, a semaphore maintains **a set of permits**. Each acquire() blocks if necessary until a permit is available, and then takes it. Each release() adds a permit, potentially releasing a blocking acquirer. However, no actual permit objects are used; the Semaphore just keeps a count of the number available and acts accordingly.

Semaphores are often used **to restrict the number of threads** than can access some (physical or logical) resource.

A semaphore **initialized to one**, and which is used such that it only has at most one permit available, can serve as a mutual exclusion lock. This is more commonly known as a binary semaphore, because it only has two states: one permit available, or zero permits available. When used in this way, the binary semaphore has the property (unlike many Lock implementations), that the "lock" can be released by a thread other than the owner (as semaphores have no notion of ownership). This can be useful in some specialized contexts, such as deadlock recovery.

This class also provides convenience methods to acquire and release multiple permits at a time. Beware of the increased risk of indefinite postponement when these methods are used without fairness set true.

Memory consistency effects: Actions in a thread prior to calling a "**release**" method such as **release**() happen-before actions following a successful "**acquire**" method such as **acquire**() in another thread.

## 构造方法—Constructors

### Semaphore(int permits)

Creates a Semaphore with **the given number of permits** and nonfair fairness setting.

Parameters:**permits** - the initial number of permits available. This value may be negative, in which case **releases** must occur **before** any **acquires** will be granted.

### Semaphore(int permits, boolean fair)

Creates a Semaphore with the given number of permits and **the given fairness setting**.

**Parameters**: **permits** - the initial number of permits available. This value may be negative, in which case releases must occur before any acquires will be granted.

**fair** - true if this semaphore will guarantee **first-in first-out**(FIFO) granting of permits under contention, else false

对于fair的详细解释：

The constructor for this class optionally accepts **a fairness parameter**. ①When set false, this class makes no guarantees about the order in which threads acquire permits. In particular, barging is permitted, that is, a thread invoking acquire() can be allocated a permit ahead of a thread that has been waiting - logically the new thread places itself at the head of the queue of waiting threads. ②When fairness is set true, the semaphore guarantees that threads invoking any of the acquire methods are selected to obtain permits in the order in which their invocation of those methods was processed (**first-in-first-out; FIFO**). Note that FIFO ordering necessarily applies to specific internal points of execution within these methods. So, it is possible for one thread to invoke acquire before another, but reach the ordering point after the other, and similarly upon return from the method. Also note that the untimed **tryAcquire** methods do not honor the fairness setting, but will take any permits that are available.

Generally, semaphores used to control resource access should be initialized **as fair**, to ensure that no thread is starved out from accessing a resource. When using semaphores for other kinds of synchronization control, the throughput advantages of non-fair ordering often outweigh fairness considerations.当用于控制资源连接的时候，建议使用公平，对其他用途，不公平更好一些。

## 方法介绍

最重要的方法就是acquire和release

### acquire():获取一个permit许可证。

void **acquire()**

Acquires a permit from this semaphore, blocking until one is available, or the thread is interrupted.

具体见acquire(int permits)，就是个数不同。

### acquire(int permits)：获取指定个数个permit许可证。

void acquire(int permits)

**Acquires the given number of permits** from this semaphore, blocking until all are available, or the thread is interrupted.

Acquires the given number of permits, **if they are available,** and returns immediately, reducing the number of available permits by the given amount.

**If insufficient permits are available** then the current thread becomes disabled for thread scheduling purposes and lies dormant until one of two things happens:

1. Some other thread invokes one of the release methods for this semaphore, the current thread is next to be assigned permits and the number of available permits satisfies this request; or
2. Some other thread interrupts the current thread.

If the current thread:

has its interrupted status set on entry to this method; or

is interrupted while waiting for a permit,

then InterruptedException is thrown and the current thread's interrupted status is cleared. Any permits that were to be assigned to this thread are instead assigned to other threads trying to acquire permits, as if permits had been made available by a call to **release()**.

**Parameters**:permits - the number of permits to acquire

Throws:

InterruptedException - if the current thread is interrupted

IllegalArgumentException - if permits is negative

### release()：归还permit许可证。

void release()

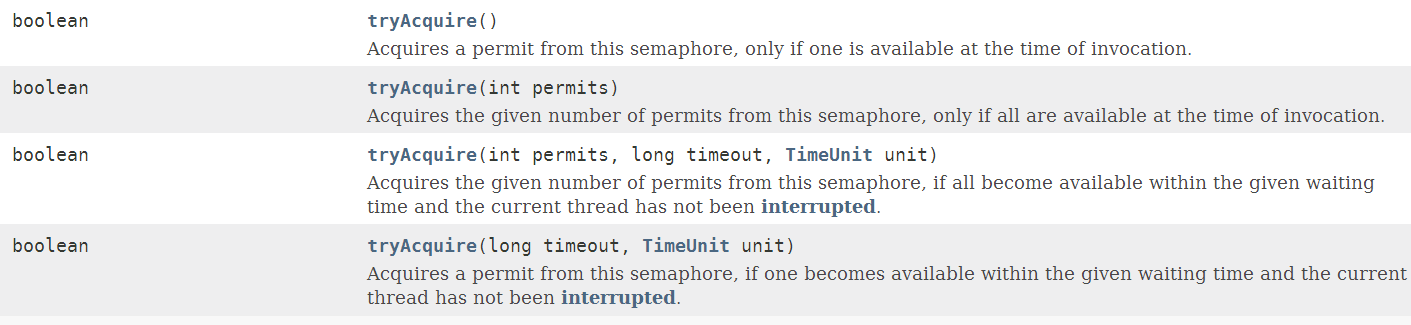
Releases a **permit**, returning it to the semaphore.

### release(int permits)：归还指定个数个permit许可证。

void release(int permits)

**Releases the given number of permits**, returning them to the semaphore.

### tryAcquire



public boolean **tryAcquire**(int permits)

Acquires the given number of permits from this semaphore, only if all are available at the time of invocation.

Acquires the given number of permits, if they are available, and returns immediately, with the value true, reducing the number of available permits by the given amount.

If insufficient permits are available then this method will return immediately with the value **false** and the number of available permits is unchanged.

Even when this semaphore has been set to use a fair ordering policy, a call to tryAcquire will immediately acquire a permit if one is available, whether or not other threads are currently waiting. This "**barging**" behavior can be useful in certain circumstances, even though it breaks fairness. If you want to honor the fairness setting, then use **tryAcquire(permits, 0, TimeUnit.SECONDS)** which is almost equivalent (it also detects interruption).

Parameters: **permits** - the number of permits to acquire

Returns:**true** if the permits were acquired and false otherwise

Throws:IllegalArgumentException - if permits is negative

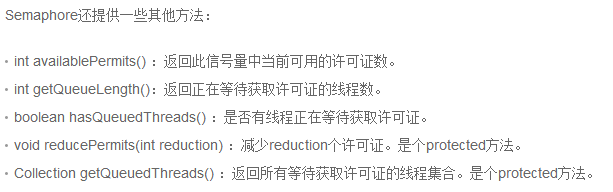
### drainPermits():获取并返回可用许可证数目，并将许可证置为0.

public int **drainPermits()**

**Acquires** and **returns** all permits that are immediately available.

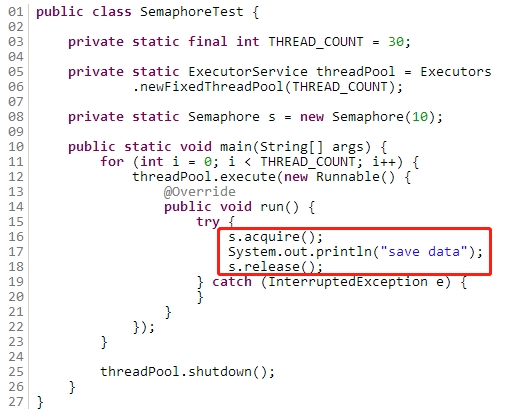
**Returns**:the number of permits acquired

### 其他方法



## 应用场景

**Semaphore**可以用于做**流量控制**，特别公用资源有限的应用场景，比如**数据库连接**。假如有一个需求，要读取几万个文件的数据，因为都是**IO密集型任务**，我们可以启动几十个线程并发的读取，但是如果读到内存后，还需要存储到数据库中，而数据库的连接数只有10个，这时我们必须控制只有十个线程同时获取数据库连接保存数据，否则会报错无法获取数据库连接。这个时候，我们就可以使用**Semaphore**来做流控。代码如下



在代码中，虽然有30个线程在执行，但是只允许10个并发的执行。Semaphore的构造方法Semaphore(int permits) 接受一个整型的数字，表示可用的**许可证数量**。Semaphore(10)表示允许10个线程获取许可证，**也就是最大并发数是10**。Semaphore的用法也很简单，首先线程**使用Semaphore的acquire()获取一个许可证**，使用完之后**调用release()归还许可证**。**还可以用tryAcquire()方法尝试获取许可证**。