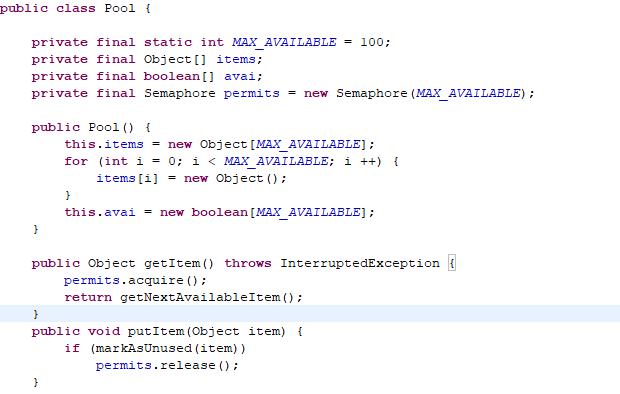
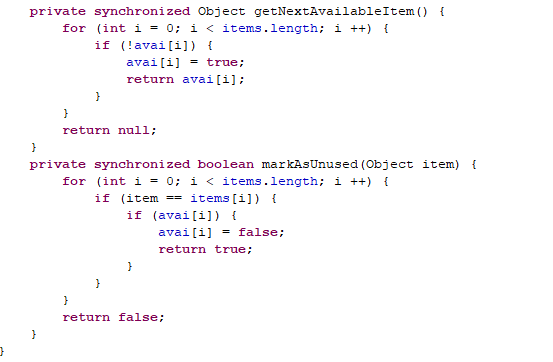
# Semphore

Semaphore表示一个计数信号量，Semahore内部持有一组许可，调用acquire的线程会获取到一个许可，如果Semaphore中没有许可可用，acquire会一直阻塞直到调用release的线程归还许可。

Semahore经常被用来限制对共享资源的访问，例如：





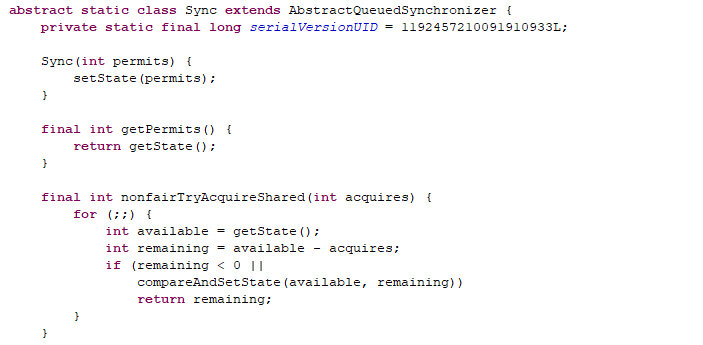
以上代码实现了一个资源池，当每次从资源池中获取元素时，首先要获得许可以保证资源池中有资源可用，当释放资源时，先标记资源可用，然后在返回许可。

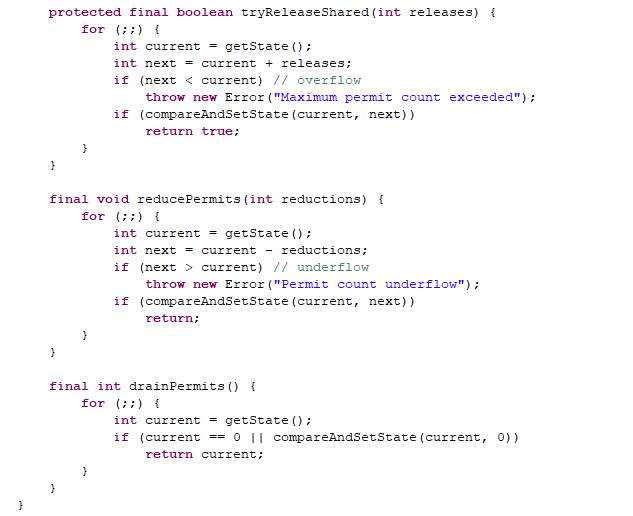
Semaphore可以作为一个独占锁来使用当初始化为一个许可的时候，但更多的是被当作一个二元信号量，因为只有两种状态，一个许可和零个许可。当Semaphore被当作二元信号量使用时，可以实现生产者消费者模式，这时，消费者获取的许可可以由生产者释放。

# 源码分析

Semaphore与ReentrantLock的实现思想相同，内部实现了一个AbstractQueuedSychronizer同步器，接口的实现都是直接委托给同步器直接调用，同样的，Semaphore也提供了公平与非公平两种模式，也是通过构造器传参来指定使用哪种方式，默认使用非公平模式。非公平模式不会保证许可的获取顺序，当许可可被获取时，新来的线程可能抢先在同步队列中的线程之前获取到许可，而公平模式下许可的获取是严格按照线程到来的顺序获取的。

## 1.1Sync





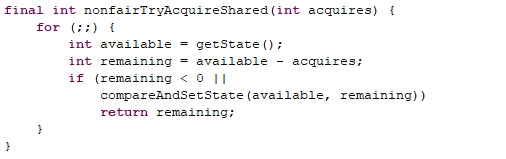
通过方法名可以看出Sync主要是实现了共享模式下同步状态的获取与释放，下面挨个方法来看一下。

### 1.1.1getPermits



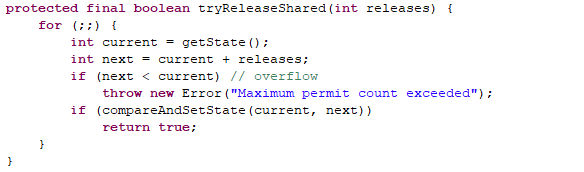
获取可用许可的数量。

### 1.1.2nonfairTryAcquireShared



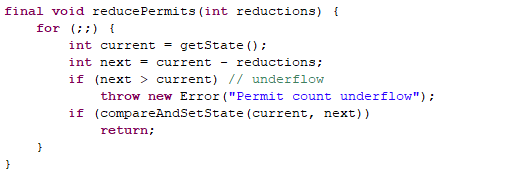
非公平模式下获取许可。如果剩余许可的数量满足请求许可的数量，那么更新许可数量，更新成功后或者许可数量不满足时返回剩余许可与请求许可数量的差值。

### 1.1.3tryReleaseShared



返回许可。首先检查许可数量是否溢出，然后更新许可数量，更新成功后返回，否则重复算法。

### 1.1.4reducePermits

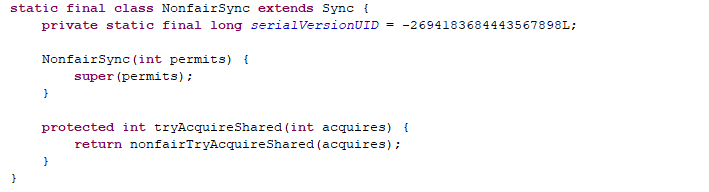


该方法与nonfairTryAcquireShared方法类似，但该方法不会检查许可数量，该方法应该在许可数量已经满足的情况下使用。

### 1.1.5drainPermits

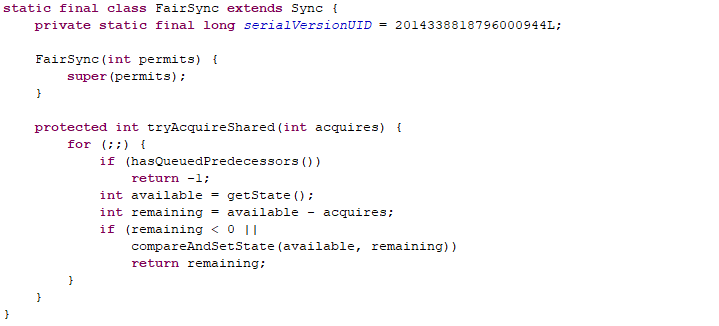
获取剩下所有的许可。

## 1.2NonfairSync



非公平模式的实现，tryAcquireShared直接调用了父类的nonfairTryAcquireShared。tryAcquireShared如果返回负数将阻塞调用线程，只有在获取许可大于可用许可时，nonfairTryAcquireShared会返回负数。

## 1.3FairSync



公平模式的实现，与非公平模式的区别在请求许可之前首先会检查同步队列中是否存在等待获取许可的线程，如果存在直接返回-1，阻塞调用线程。

## 1.4Semaphore

### 1.4.1构造器



当不指定fair时，使用非公平模式。

### 1.4.2acquire



Semaphore在阻塞获取许可中会响应中断。该方法只有在成功获取到许可或者当前线程被中断后返回。

### 1.4.3acquireUninterruptibly



不响应中断的阻塞获取许可版本。只有在成功获取到许可之后才会返回。

### 1.4.4tryAcquire



尝试获取许可，成功获取到许可返回true，否则返回false。该方法只有非公平版本。

### 1.4.5release



释放许可，并唤醒阻塞的获取许可的线程。