1. 作用：

控制访问某个共享资源的线程的个数。

2. 原理：

Semaphore内部维护了一个计数器，其值为可以访问的共享资源的个数。一个线程要访问共享资源，先获得信号量，如果信号量的计数器值大于1，意味着有共享资源可以访问，则使其计数器值减去1，再访问共享资源。

如果计数器值为0,线程进入休眠。当某个线程使用完共享资源后，释放信号量，并将信号量内部的计数器加1，之前进入休眠的线程将被唤醒并再次试图获得信号量。

就好比一个厕所管理员，站在门口，只有厕所有空位，就开门允许与空侧数量等量的人进入厕所。多个人进入厕所后，相当于N个人来分配使用N个空位。为避免多个人来同时竞争同一个侧卫，在内部仍然使用锁来控制资源的同步访问。

3. 源码

acquire源代码

public void acquire() throws InterruptedException {

sync.acquireSharedInterruptibly(1);

}

public final void acquireSharedInterruptibly(int arg)

throws InterruptedException {

if (Thread.interrupted())

throw new InterruptedException();

if (tryAcquireShared(arg) < 0)

doAcquireSharedInterruptibly(arg);

}

final int nonfairTryAcquireShared(int acquires) {

for (;;) {

int available = getState();

int remaining = available - acquires;

if (remaining < 0 ||

compareAndSetState(available, remaining))

return remaining;

}

}

可以看出，如果remaining <0 即获取许可后，许可数小于0，则获取失败，在doAcquireSharedInterruptibly方法中线程会将自身阻塞，然后入列。

release源代码

public void release() {

sync.releaseShared(1);

}

public final boolean releaseShared(int arg) {

if (tryReleaseShared(arg)) {

doReleaseShared();

return true;

}

return false;

}

protected final boolean tryReleaseShared(int releases) {

for (;;) {

int current = getState();

int next = current + releases;

if (next < current) // overflow

throw new Error("Maximum permit count exceeded");

if (compareAndSetState(current, next))

return true;

}

}

可以看出释放许可就是将AQS中state的值加1。然后通过doReleaseShared唤醒等待队列的第一个节点。可以看出Semaphore使用的是AQS的共享模式，等待队列中的第一个节点，如果第一个节点成功获取许可，又会唤醒下一个节点，以此类推。

4. 构造函数

//构造一个指定数量的许可

**public** Semaphore(**int** permits)

//构造一个指定数量的许可，fair决定了公平性，true为公平获得，false为非公平获得

**public** Semaphore(**int** permits, **boolean** fair)

4. demo

**public static void** main(String[] args){  
 *//定义10个许可的信号量* Semaphore semaphore = **new** Semaphore(10);  
 *//线程池* ExecutorService executor = Executors.*newFixedThreadPool*(50);  
 **for**(**int** i=0;i<50;i++){  
 Thread thread = **new** Thread( ()-> {  
 **try** {  
 **synchronized** (executor){  
 *//获得许可* semaphore.acquire();  
 System.***out***.println(Thread.*currentThread*().getName()+**"获得了信号量"**);  
 System.***out***.println(**"当前可用的资源为:"**+semaphore.availablePermits());  
 }  
 *//do something* Thread.*sleep*(2000);  
 **synchronized** (semaphore){  
 *//释放许可* semaphore.release();  
 System.***out***.println(Thread.*currentThread*().getName()+**"释放了信号量"**);  
 System.***out***.println(**"当前可用的资源为:"**+semaphore.availablePermits());  
 }  
 } **catch** (Exception e){  
 }  
 });  
 executor.submit(thread);  
 }  
 executor.shutdown();  
}

pool-1-thread-1获得了信号量

当前可用的资源为:9

pool-1-thread-2获得了信号量

当前可用的资源为:8

pool-1-thread-3获得了信号量

当前可用的资源为:7

pool-1-thread-4获得了信号量

当前可用的资源为:6

pool-1-thread-5获得了信号量

当前可用的资源为:5

pool-1-thread-6获得了信号量

当前可用的资源为:4

pool-1-thread-7获得了信号量

当前可用的资源为:3

pool-1-thread-8获得了信号量

当前可用的资源为:2

pool-1-thread-9获得了信号量

当前可用的资源为:1

pool-1-thread-10获得了信号量

当前可用的资源为:0

pool-1-thread-1释放了信号量

pool-1-thread-11获得了信号量

当前可用的资源为:0

当前可用的资源为:0

pool-1-thread-50获得了信号量

pool-1-thread-2释放了信号量

当前可用的资源为:0

当前可用的资源为:0

pool-1-thread-49获得了信号量

当前可用的资源为:0

pool-1-thread-6释放了信号量

当前可用的资源为:0

pool-1-thread-7释放了信号量

pool-1-thread-48获得了信号量

当前可用的资源为:0

pool-1-thread-5释放了信号量

当前可用的资源为:0

当前可用的资源为:1

pool-1-thread-47获得了信号量

pool-1-thread-3释放了信号量

当前可用的资源为:1

当前可用的资源为:1

pool-1-thread-4释放了信号量

pool-1-thread-46获得了信号量

当前可用的资源为:1

当前可用的资源为:1

pool-1-thread-45获得了信号量

当前可用的资源为:0

pool-1-thread-8释放了信号量

当前可用的资源为:0

pool-1-thread-44获得了信号量

当前可用的资源为:0

pool-1-thread-9释放了信号量

pool-1-thread-43获得了信号量

当前可用的资源为:0

当前可用的资源为:0

pool-1-thread-42获得了信号量

当前可用的资源为:0

pool-1-thread-10释放了信号量

当前可用的资源为:0

pool-1-thread-11释放了信号量

当前可用的资源为:0

pool-1-thread-41获得了信号量

当前可用的资源为:0

pool-1-thread-40获得了信号量

当前可用的资源为:0

pool-1-thread-48释放了信号量

当前可用的资源为:0

pool-1-thread-45释放了信号量

pool-1-thread-39获得了信号量

当前可用的资源为:0

当前可用的资源为:1

pool-1-thread-46释放了信号量

当前可用的资源为:1

pool-1-thread-38获得了信号量

当前可用的资源为:1

pool-1-thread-37获得了信号量

当前可用的资源为:0

pool-1-thread-47释放了信号量

当前可用的资源为:0

pool-1-thread-49释放了信号量

当前可用的资源为:1

pool-1-thread-36获得了信号量

当前可用的资源为:1

pool-1-thread-50释放了信号量

当前可用的资源为:0

pool-1-thread-44释放了信号量

当前可用的资源为:1

pool-1-thread-35获得了信号量

当前可用的资源为:1

pool-1-thread-34获得了信号量

当前可用的资源为:0

pool-1-thread-33获得了信号量

当前可用的资源为:0

pool-1-thread-42释放了信号量

当前可用的资源为:0

pool-1-thread-43释放了信号量

当前可用的资源为:1

pool-1-thread-32获得了信号量

当前可用的资源为:0

pool-1-thread-41释放了信号量

pool-1-thread-31获得了信号量

当前可用的资源为:0

当前可用的资源为:0

pool-1-thread-30获得了信号量

pool-1-thread-40释放了信号量

当前可用的资源为:0

当前可用的资源为:0

pool-1-thread-37释放了信号量

当前可用的资源为:0

pool-1-thread-29获得了信号量

当前可用的资源为:1

pool-1-thread-39释放了信号量

pool-1-thread-28获得了信号量

当前可用的资源为:0

当前可用的资源为:0

pool-1-thread-38释放了信号量

当前可用的资源为:1

pool-1-thread-27获得了信号量

当前可用的资源为:0

pool-1-thread-34释放了信号量

pool-1-thread-26获得了信号量

当前可用的资源为:0

当前可用的资源为:0

pool-1-thread-36释放了信号量

pool-1-thread-25获得了信号量

当前可用的资源为:0

当前可用的资源为:0

pool-1-thread-35释放了信号量

当前可用的资源为:0

pool-1-thread-24获得了信号量

当前可用的资源为:1

pool-1-thread-33释放了信号量

pool-1-thread-23获得了信号量

当前可用的资源为:0

当前可用的资源为:0

pool-1-thread-32释放了信号量

当前可用的资源为:1

pool-1-thread-22获得了信号量

当前可用的资源为:0