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. 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();  
}