Semaphore

信号量

PV操作

- P 信号量值减1，结果不可为负
- V 信号量值加1

- PV来自荷兰语， Passeren, Vrijgeven 通过和释放

java.util.concurrent.Semaphore

构造函数	Semaphore(int permits)	permits许可数，资源数量，可为负值
	Semaphore(int permits, boolean fair)	fair 是否使用公平机制；按照等待时间分发
P操作	acquire()	
	acquire(int permits)	
	tryAcquire(int permits)	
	tryAcquire(long timeout, TimeUnit unit)	
	tryAcquire(int permits, long timeout, TimeUnit unit)	
P/V	drainPermits()	许可数改为0；返回获取/释放的数目
V操作	release()	释放一个许可
	release(int permits)	释放多个许可

信号量

- 许可数目可多个
- 许可数目可动态调整
- acquire release 无先后次序
- 用于资源池管理

锁

- 只能有一个持有锁（读锁例外）
- 只有一个
- 必须先lock后unlock
- 原子操作/控制并发

```java
public class SampleConnectionPool {
    private static final int MAX = 10;
    private final Semaphore available = new Semaphore(MAX, true);

    protected Connection[] items = new Connection[MAX];
    protected boolean[] used = new boolean[MAX];

    public Connection getOne() throws InterruptedException {
        available.acquire();
        return createOne();
    }
    private synchronized Connection createOne(){
        for (int i = 0; i < MAX; ++i) {
            if (!used[i]) {
                used[i] = true;
                if(items[i] == null){
                   // items[i] = new ...
                }
                return items[i];
            }
        }
        return null;
    }
```

```java
    public void putOne(Connection conn) {
        if (markAsAvailable(conn)) {
            available.release();
        }
    }
    private synchronized boolean markAsAvailable(
                                  Connection conn){
        for (int i = 0; i < MAX; ++i) {
            if (conn == items[i]) {
                if (used[i]) {
                    used[i] = false;
                    return true;
                }else{
                    return false;
                }
            }
        }
        return false;
    }
}
```

java.util.concurrent.Semaphore