20180330_Java线程的几种状态和相关函数

```
1- 线程的4种状态：
    产生(New)，尚未启动，start() 函数之前
    可执行（Runnable）,start()之后。 可能正在运行，也可能正处于线程池中等待排程器启动它
    死亡（Dead），一个线程run() 执行完后，就进入该死亡状态
     停滞（Blocked），wait()函数之后就处于停滞状态。排程器不会调用它，只有当两次对该线程调用notify或notifyAll后它才能两次回到可执行状态

2- wait(), notify() 和 notifyAll() 三大函数
```

1- 线程的几种状态

线程有四种状态，任何一个线程肯定处于这四种状态中的一种：

1) **产生（New）**：线程对象已经产生，但尚未被启动，所以无法执行。如通过new产生了一个线程对象后没对它调用start()函数之前。

2) **可执行（Runnable）**：**每个支持多线程的系统都有一个排程器，排程器会从线程池中选择一个线程并启动它**。当一个线程处于可执行状态时，表示它可能正处于线程池中等待排程器启动它；也可能它已正在执行。如执行了一个线程对象的start()方法后，线程就处于可执行状态，但显而易见的是此时线程不一定正在执行中。

3) **死亡（Dead）**：当一个线程正常结束，它便处于死亡状态。如一个线程的run()函数执行完毕后线程就进入死亡状态。

4) **停滞（Blocked）**：当一个线程处于停滞状态时，系统排程器就会忽略它，不对它进行排程。当处于停滞状态的线程重新回到可执行状态时，它有可能重新执行。如通过对一个线程调用wait()函数后，线程就进入停滞状态，只有**当两次对该线程调用notify或notifyAll后它才能两次回到可执行状态**。

2- class　Object下常用的线程函数

wait()、notify()和notifyAll()这三个函数由java.lang.Object类提供，用于协调多个线程对共享数据的存取。

2.1- wait()、notify()和notifyAll()

1) wait()函数有两种形式：第一种形式接受一个毫秒值，用于在指定时间长度内暂停线程，使线程进入停滞状态。第二种形式为不带参数，代表waite()在notify()或notifyAll()之前会持续停滞。

2) 当对一个对象执行notify()时，**会从线程等待池中移走该任意一个线程，并把它放到锁标志等待池中；当对一个对象执行notifyAll()时，会从线程等待池中移走所有该对象的所有线程，并把它们放到锁标志等待池中。**

3) 当调用wait()后，线程会**释放掉它所占有的“锁标志”，从而使线程所在对象中的其它synchronized数据可被别的线程使用。**

例17：

下面，我们将对例11中的例子进行修改

```
class TestThreadMethod extends Thread{
    public static int shareVar = 0;
    public TestThreadMethod(String name){
    super(name);
    }

    public synchronized void run(){
        if(shareVar==0){
            for(int i=0; i<10; i++){
                shareVar++;
                if(shareVar==5){
                    try{
                        this.wait();  // (4)
                    }
                    catch(InterruptedException e){}

                }

            }

        }

        if(shareVar!=0){
            System.out.print(Thread.currentThread().getName());
            System.out.println(" shareVar = " + shareVar);
            this.notify();  // (5)
        }

    }

}

public class TestThread{
    public static void main(String[] args){
    TestThreadMethod t1 = new TestThreadMethod("t1");
    TestThreadMethod t2 = new TestThreadMethod("t2");
    t1.start();  // (1)
    //t1.start();   (2)
    t2.start();  // (3)
```

```
    }
}
```

运行结果为：

```
t2 shareVar = 5
```

因为t1和t2是两个不同对象，所以线程t2调用代码（5）不能唤起线程t1。如果去掉代码（2）的注释，并注释掉代码（3），结果为：

```
t1 shareVar = 5
t1 shareVar = 10
```

这是因为，当代码（1）的线程执行到代码（4）时，它进入停滞状态，并释放对象的锁状态。接着，代码（2）的线程执行run()，由于此时 shareVar值为5，所以执行打印语句并调用代码（5）使代码（1）的线程进入可执行状态，然后代码（2）的线程结束。当代码（1）的线程重新执行后，它接着执行for()循环一直到shareVar=10，然后打印shareVar。