

Java2 Lab 16

(多线程)

[Experimental Objective]

掌握多线程的基本使用方法

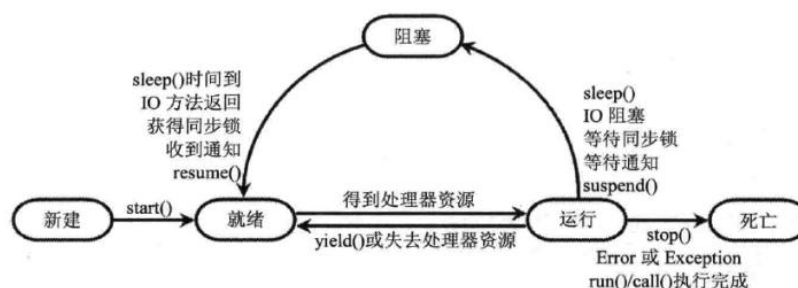
[多线程]

多线程就是一个程序中有多个线程在同时执行。

线程的生命周期：

有新建(new),就绪(Runnable),运行(Running),阻塞(Blocked)和死亡(Dead)5种状态。

启动线程使用 start()方法，而不是 run()方法！永远不要调用线程对象的 run()方法！调用 start()方法来启动线程，系统会把该 run()方法当成线程执行体来处理；但如果直接调用线程对象的 run()方法，则 run()方法立即就会被执行，而且在 run()方法返回之前其他线程无法并发执行——也就是说，如果直接调用线程对象的 run()方法，系统把线程对象当成一个普通对象，而 run()方法也是一个普通方法，而不是线程执行体。



【案例1 多线程创建】

(1) 继承 Thread 类的方法创建线程如下：

- 1、定义一个继承自 Thread 类的子类，并且重写 run()方法，run(),这个 run 方法，就是未来新线程要运行的具体任务或者叫做功能。
- 2、实例化(new)出刚才定义的子类
- 3、运行这个新对象的 start 方法。务必记住是 start 方法，只有这样才会启动一个新的线程。如果是运行 run 方法，那么仍然是简单的单线程执行

```
package multithread;
```

```
public class CreateThread1 extends Thread {
    public void run() {
        for (int i = 0; i <= 100; i++) {
```

```

        System.out.println(getName());
    }
}

public static void main(String[] args) {
    for (int i = 0; i < 100; i++) {
        if (i % 10 == 0) {
            new CreateThread1().start();
            new CreateThread1().start();
        }
    }
}
}

```

可以看到创建的两个线程在同时进行：



```

<terminated> CreateThread1 [Java Application] C:\Program Files\Java\jre1.8.0_181\bin
Thread-0
Thread-0
Thread-0
Thread-0
Thread-5
Thread-5
Thread-6
Thread-6

```

(2) 实现 Runnable 接口，创建线程如下：

- 1、定义一个类，这个类需要实现 Runnable 接口，仍然需要在该类中重写接口中的 run 方法，与方法 1 一样，这个 run 方法也是未来的线程执行体
- 2、实例化(new)出刚才定义的类
- 3、实例化(new)出一个 Thread 类，并以 A 作为 target，运行 start 方法

```
package multithread;
```

```

public class CreateThread2 implements Runnable {
    public void run() {
        for (int i = 0; i <= 100; i++) {
            System.out.println(Thread.currentThread().getName());
        }
    }

    public static void main(String[] args)
    {
        for(int i=0;i<100;i++)
        {
            if(i%10==0)
            {
                CreateThread2 st=new CreateThread2();
                new Thread(st,"name1").start();
                new Thread(st,"name2").start();
            }
        }
    }
}

```

【练习 1 多线程创建】

请分别使用这两种方式创建 2 个线程。其中使用继承 Thread 类方法创建线程 1 用来打印数字 1-52, 使用实现 Runnable 接口方法创建线程 2 用来打印字母 A-Z。(顺序暂时不要求)

【案例 2 同步与互斥】**(1) synchronized 使用:**

以下的例子中, 如果多个进程需要使用同一资源, 会出现同时访问的情况:

```
package multithread;

public class SynchronizedTest2 implements Runnable {
    public void run() {
        this.UsingResource();
    }

    public static void main(String[] args) {
        SynchronizedTest2 st = new SynchronizedTest2();
        new Thread(st, "name1").start();
        new Thread(st, "name2").start();
    }

    private void UsingResource() {
        cnt++;
        System.out.println(Thread.currentThread().getName() + " is the" +
cnt + "th using resource");
    }

    int cnt = 0;
}
```

结果发现会两个线程在同时使用资源UsingResource():

```
<terminated> SynchronizedTest2 [Java Application] C:\Program Files\Java\jre
name2 is the2th using resource
name1 is the2th using resource
```

如果在需要访问的资源前加上 synchronized 修饰, 使之互斥访问, 可以看到效果:

```
private synchronized void UsingResource() {
    cnt++;
    System.out.println(Thread.currentThread().getName() + " is the" +
cnt + "th using resource");
}
```

```
Type Hierarchy Console JUnit Servers
<terminated> SynchronizedTest2 [Java Application] C:\Program Files\Java\jre1.8.0_1
name1 is the1th using resource
name2 is the2th using resource
```

(2) lock 使用:

使用 lock 对互斥访问的变量或成员函数加锁也可达到互斥访问的目的。

```
package multithread;

import java.util.concurrent.locks.ReentrantLock;

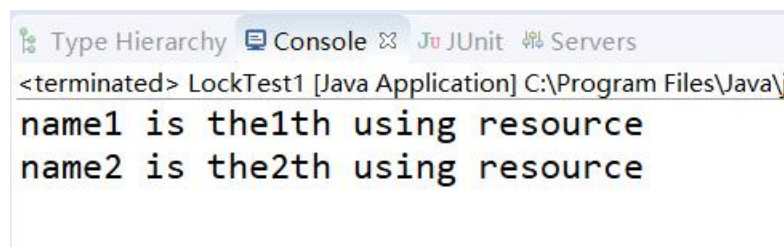
public class LockTest1 implements Runnable {
    public void run() {
        this.UsingResource();
    }

    public static void main(String[] args) {
        LockTest1 st = new LockTest1();
        new Thread(st, "name1").start();
        new Thread(st, "name2").start();
    }

    private void UsingResource() {
        lock.lock();
        cnt++;
        System.out.println(Thread.currentThread().getName() + " is the" +
cnt + "th using resource");
        lock.unlock();
    }

    int cnt = 0;
    ReentrantLock lock = new ReentrantLock();
}
```

可以看到，在访问资源时，通过加锁和解锁也可以保护资源不被多个线程访问。



```
<terminated> LockTest1 [Java Application] C:\Program Files\Java\
name1 is the1th using resource
name2 is the2th using resource
```

(3) wait 和 notify()使用

wait() 方法可以使线程进入等待状态，而 notify() 可以使等待的状态唤醒。这样的同步机制十分适合生产者、消费者模式：消费者消费某个资源，而生产者生产该资源。当该资源缺失时，消费者调用 wait() 方法进行自我阻塞，等待生产者的生产；生产者生产完毕后调用 notify/notifyAll() 唤醒消费者进行消费。

```
public class WaitNotifyTest {
    public static void main(String[] args) {
        final Object object = new Object();
        Thread t1 = new Thread() {
            public void run()
            {
                synchronized (object) {
                    System.out.println("T1 start!");
                    try {
                        object.wait();
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            }
        };
    }
}
```

```
        }
        System.out.println("T1 end!");
    }
}
};
Thread t2 = new Thread() {
    public void run()
    {
        synchronized (object) {
            System.out.println("T2 start!");
            object.notify();
            System.out.println("T2 end!");
        }
    }
};

t1.start();
t2.start();
}
}
```

【练习 2 同步与互斥】

请在练习 1 中输出 1-52 和 A-Z 线程基础上，进行适当修改。加入同步与互斥，要求输出为 “12A34B56C...5152Z” 。