第9章 Java中的多线程

部分内容摘自

《Java面向对象编程》，孙卫琴

《Java 程序设计》，唐大仕

Java中的线程

- 目前Java程序都是串行执行的，也就是每个语句依次执行。
- 有时候需要程序并行执行：

1 有事系统有多个工作要同时完成：例如一边打字、一遍听歌；一边开视频会议，一边看网页。

2 利用CPU多线程的特性。

- 一个进程可以有多个线程在执行。

1 创建线程的方法1

- 1．通过继承Thread类创建线程
- ch09.TestThread1
- 注意启动线程用t1.start()，不能用t1.run();

```
public class TestThread1 {
    public static void main(String args[]) {
        Thread t = new MyThread1(100);
        t.start();
        System.out.println("Main thead end!");
    }
}

class MyThread1 extends Thread {
    private int n;
    public MyThread1(int n) {
        this.n = n;
    }
    public void run() {
        for (int i = 0; i < n; i++) {
```

创建线程对象的两种方法

- 1．通过向Thread()构造方法传递Runnable对象来创建线程 ch09.TestThread2
- 注意启动线程用t.start()，不能用t.run();

```
public class TestThread2 {
    public static void main(String args[]) {
        MyThread2 mytask = new MyThread2(100);
        Thread t = new Thread(mytask);
        t.start();
    }
}

class MyThread2 implements Runnable {
    private int n;
    public MyThread2(int n) {
        this.n = n;
    }
    public void run() {
        for (int i = 0; i < n; i++) {
```

两种方法的比较

- 使用**Runnable**接口
- 直接继承**Thread**类
- 无论使用哪种方法，注意启动线程的方法应该调用**start()**，不能调用**run()**。

2 一个使用多线程的例子

- ch09.sync.sum. SumMain
- thread.jion(),等待这个线程结束。
- 多线程可以显著提升系统性能。
- 比较SumMain中1/2/3/4/5个线程时候耗时比较。

使用多线程的另一个例子

- 一个搞笑的排序算法：ch09.join. MySort
  （工作中大家千万不要使用这种方法排序）

Daemon线程和非Daemon线程

- ch09.TestThreadDaemon
- Thread. setDaemon(**true/false);**
- **Deamon 线程 主程序终止 线程终止**
- 非**Deamon 线程 主程序终止 线程**不**终止**

3 多线程访问的冲突问题

因为多线程的原因，同一个对象，同一个时刻可能有多个线程访问，进而导致冲突。

- ch09.counter0. SyncCounter0
- ch09.sync.stack0. MyStackUserThread

解决方式 使用 **synchronized**

synchronized void method() 锁加在这个对象上。

static synchronized void method() 锁加在这个类上。

多线程访问的冲突的解决

关键字**synchronized** 用来与对象的互斥锁联系

- **synchronized void method()** 锁加在这个对象上。
- **static synchronized void method()** 锁加在这个类上。

解决了多线程访问冲突之后的例子：

- ch09.counter0. SyncCounter0
- ch09.sync.stack0. MyStackUserThread

多线程访问的冲突的解决

关键字**synchronized** 用来与对象的互斥锁联系

- **synchronized void method()** 锁加在这个对象上。
- **static synchronized void method()** 锁加在这个类上。

解决了多线程访问冲突之后的例子：

- ch09.counter0. SyncCounter0
- ch09.sync.stack0. MyStackUserThread

synchronized 带来的死锁问题

- ch09.lock. DeadLockTest
- 运行程序，看一下为什么卡住了？

死锁

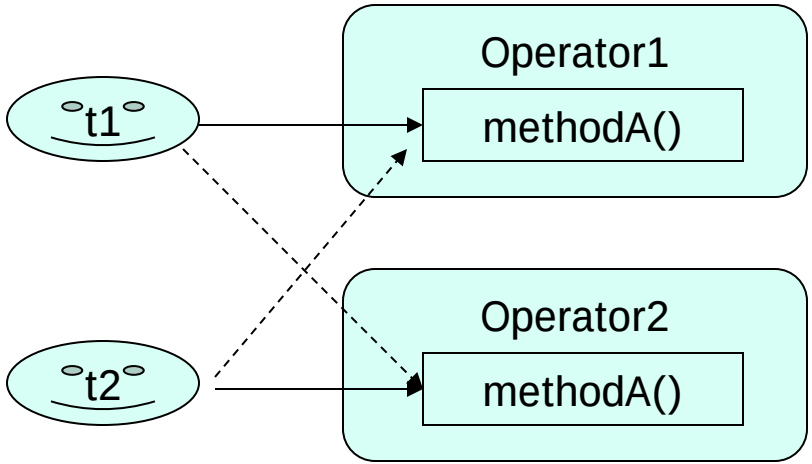
- 当一个线程等待由另一个线程持有的锁，而后者正在等待已被第一个线程持有的锁时，就会发生死锁。
- Java不监测也不试图避免这种情况。因而保证不发生死锁就成了程序员的责任。
- 参见ch09.lock. DeadLockTest

死锁

```
public class DeadLockTest {
  public static void main(String args[]){
     Operator o1=new Operator();
     Operator o2=new Operator();

     o1.anotherOperator=o2;
     o2.anotherOperator=o1;

     Thread t1=new Thread(o1);
     Thread t2=new Thread(o2);
     t1.start();
     t2.start();
  }
}
```



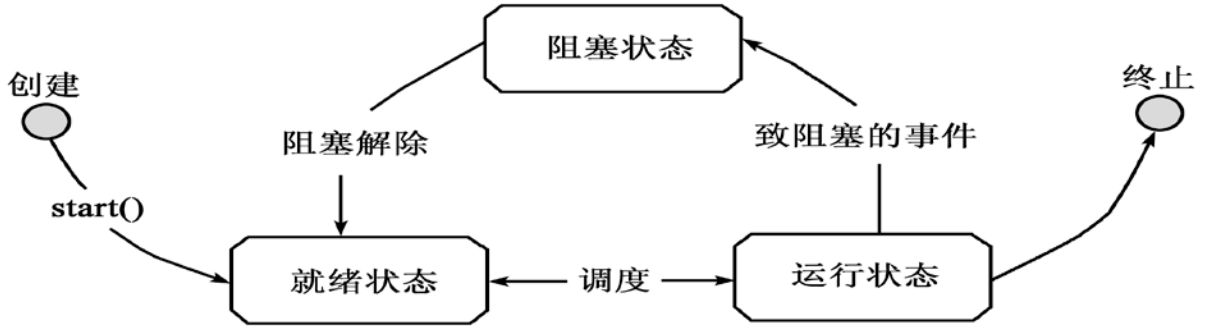

```
C:\java_base\examples\src>java mythread.deadlock.DeadLockTest
Thread-0:begin methodA
Thread-1:begin methodA
Thread-0:call another methodA
Thread-1:call another methodA
```

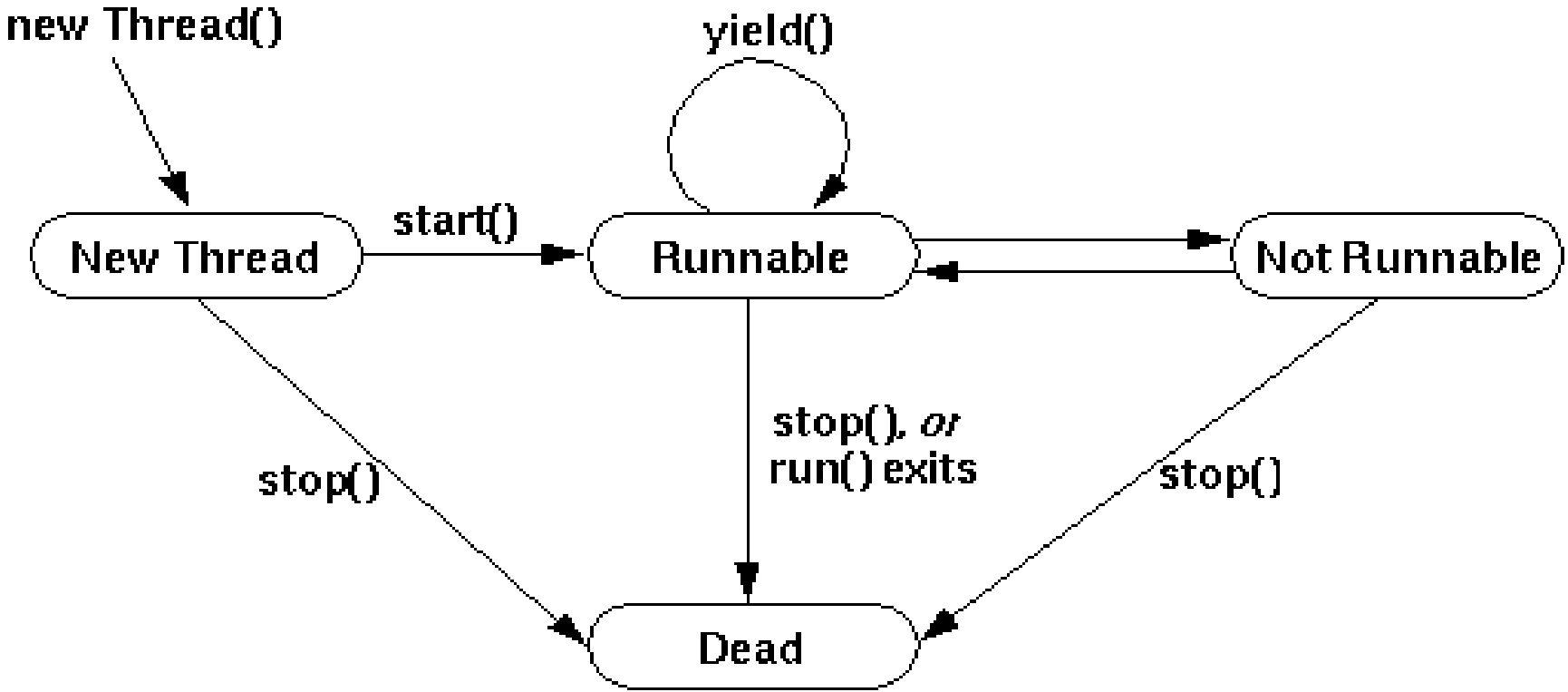
避免死锁

- 控制锁的范围
- 按序分配资源

4 线程的状态与生命周期

- 在一个线程的生命周期中，它总处于某一种状态中。
- 线程的状态表示了线程正在进行的活动以及在这段时间内线程能完成的任务。




new Thread()
yield()
New Thread
start()
Runnable
Not Runnable
stop()
stop(), or
run() exits
stop()
Dead

4 多线程的控制-生产者-消费者问题

- 不断使用某类资源的线程称为消费者
- 不断产生或释放同类资源的线程称为生产者。
- 如何协调生产者和消费者的关系，保证生产的东西及时消费掉。

生产者-消费者问题

- ch09.producer_consumer. ProducerConsumerDemo

```
public synchronized String pop() throws InterruptedException {
    while ((point + 1) <= 0) {
        this.wait();
    }
    String goods = buffer[point];
    buffer[point--] = null;
    System.out.println(" pop " + goods + " cur stack size  =" + (point + 1));
    this.notifyAll();
    return goods;
}

public synchronized void push(String goods) throws InterruptedException {
    while ((point + 1) >= SIZE) {
        this.wait();
    }
    buffer[++point] = goods;
    System.out.println(" push " + goods + " cur stack size  =" + (point + 1));
    this.notifyAll();
}
```

notify()/ notifyAll()方法。

- notify()，用来选择并唤醒等候进入监视器的线程。
- notifyAll()，唤醒所有等待的线程。
- 只有获得锁以后，才有权利notify()/ notifyAll()

wait()方法

- wait()方法使当前线程处于等待状态，直到别的线程调用notify()方法来通知它。

5 如何终止线程

- 当线程执行完run()方法，它将自然终止运行。
- Thread有一个stop()方法，可以强制结束线程，但这种方法是不安全的。因此，目前stop()方法已经被废弃。
- 实际编程中，一般是定义一个标志变量，然后通过程序来改变标志变量的值，从而控制线程从run()方法中自然退出。
- 参见ch09.MyThreadStop.java

6 线程的优先级

- Thread.setPriority(Thread.***MAX_PRIORITY/NORM_PRIORITY/MIN_PRIORITY);***

- ***ch09.*** TestThreadPriority例子

7 定时器：Timer &TimerTask

参见：

http://tool.oschina.net/uploads/apidocs/jdk_7u4/java/util/Timer.html

http://tool.oschina.net/uploads/apidocs/jdk_7u4/java/util/TimerTask.html

TimerTask例子

ch09. TimerTaskDemo

ch09. TimerTaskDemo2

TimerTask例子TimerTaskDemo.java

```
public class MyTimerTask {
        public static void main(String[] args) {
                Timer timer=new Timer();
                timer.schedule(new TimerTask(){
                        int i=0;
                        public void run() {
                                System.out.println(i++);
                        }//红色部分 匿名类
                }, 0, 1000L);
        }}
```

8 关于ThreadLocal

- 如何在不传递参数的情况下，让一个程序使用程序外的一个变量？
- 如何在不传递参数的情况下，让一个线程使用程序外的一个变量？
- 每个线程内使用的变量。
- 线程间互不干扰。
- 如果无法理解ThreadLocal的例子，想想数据库的Connection的传递问题
- ch09.thread_local.MyThreadLocal 。

关于ThreadLocal

- 每个线程内使用的变量。
- 线程间互不干扰。
- ch09.thread_local.MyThreadLocal 。