第八章 多线程

本章内容

- 线程的基本概念
- 线程的创建和启动
- 线程的调度和优先级
- 线程的状态控制
- 线程同步

线程的基本概念

- 线程是一个程序内部的顺序控制流。
- 线程和进程的区别
  - 每个进程都有独立的代码和数据空间(进程上下文)，进程间的切换会有较大的开销。
  - 线程可以看成时轻量级的进程，同一类线程共享代码和数据空间，每个线程有独立的运行栈和程序计数器(PC)，线程切换的开销小。
  - 多进程：在操作系统中能同时运行多个任务(程序)
  - 多线程：在同一应用程序中有多个顺序流同时执行

- Java的线程是通过java.lang.Thread类来实现的。
- VM 启动时会有一个由主方法（public static void main() {}）所定义的线程。
- 可以通过创建 Thread 的实例来创建新的线程。
- 每个线程都是通过某个特定Thread对象所对应的方法run()来完成其操作的，方法run()称为线程体。
- 通过调用Thead类的start()方法来启动一个线程。

线程的创建和启动

- 可以有两种方式创建新的线程。
  - 第一种
    - 定义线程类实现Runnable接口
    - Thread myThread = new Thead（target）//target为Runnable接口类型。
    - Runnable中只有一个方法：
      - public void run（）；用以定义线程运行体。
    - 使用Runnable接口可以为多个线程提供共享的数据。
    - 在实现Runnable接口的类的run方法定义中可以使用Thread的静态方法：
      - public static Thread currentThread() 获取当前线程的引用。
  - 第二种
    - 可以定义一个Thread的子类并重写其run方法如：

```
class MyThread extends Thead {
    public void run(){…}
}
```

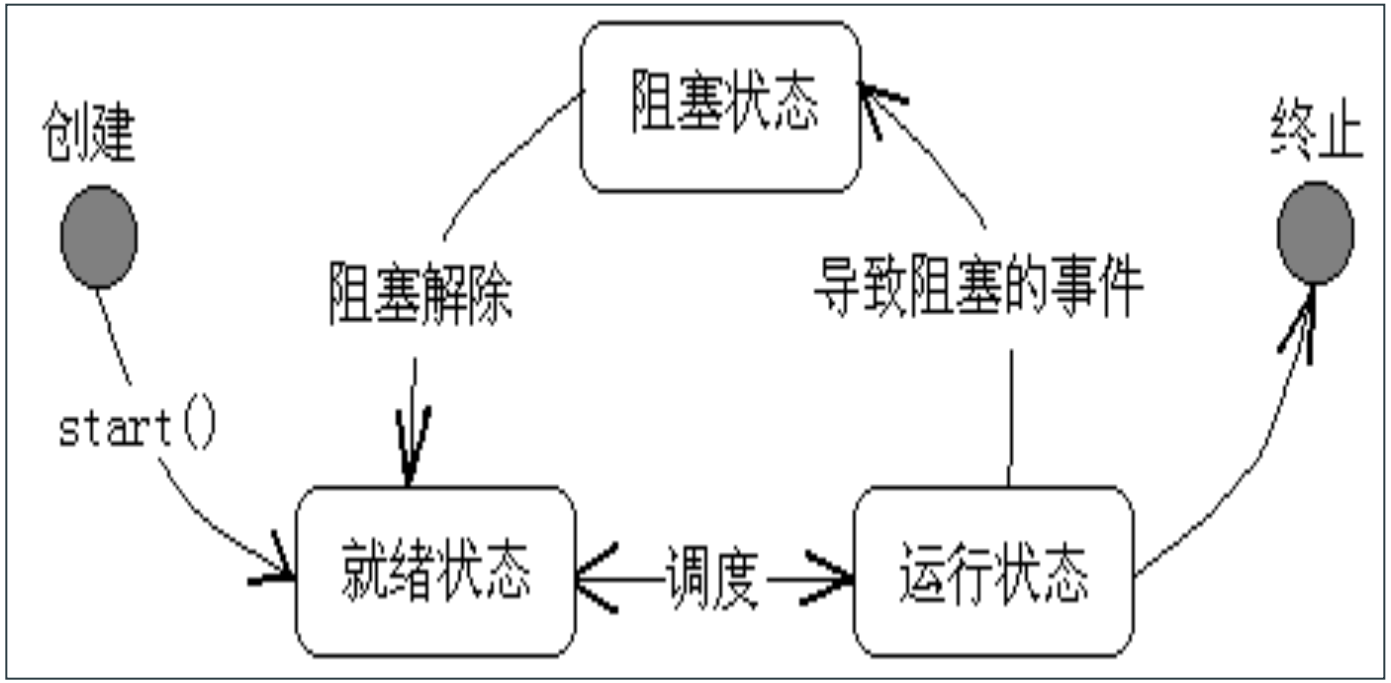
    - 然后生成该类的对象：

MyThread myThread = new MyThead(…)

TestThread1.java

- 使用那种好呢？

线程状态转换



线程控制基本方法

方 法	功 能
isAlive()	判断线程是否还“活”着，即线程是否还未终止。
getPriority()	获得线程的优先级数值
setPriority()	设置线程的优先级数值
Thread.sleep()	将当前线程睡眠指定毫秒数
join()	调用某线程的该方法，将当前线程与该线程“合并”，即等待该线程结束，再恢复当前线程的运行。
yield()	让出CPU，当前线程进入就绪队列等待调度。
wait()	当前线程进入对象的wait pool。
notify()/ notifyAll()	唤醒对象的wait pool中的一个/所有等待线程。

sleep / join / yield 方法

- sleep方法
  - 可以调用Thread的静态方法：

    public static void sleep(long millis) throws InterruptedException

    使得当前线程休眠（暂时停止执行millis毫秒）。
  - 由于是静态方法，sleep可以由类名直接调用：

    Thread.sleep(…)
- join方法
  - 合并某个线程
- yield方法
  - 让出CPU，给其他线程执行的机会

TestInterrupt.java

TestJoin.java

TestYield.java

线程的优先级别

- Java提供一个**线程调度器**来监控程序中启动后进入就绪状态的所有线程。线程调度器按照线程的优先级决定应调度哪个线程来执行。
- 线程的优先级用数字表示，范围从1到10，一个线程的缺省优先级是5。
  - Thread.MIN_PRIORITY = 1
  - Thread.MAX_PRIORITY = 10
  - Thread.NORM_PRIORITY = 5
- 使用下述线方法获得或设置线程对象的优先级。
  - int getPriority();
  - void setPriority(int newPriority);
- 不同平台上的优先级
  - Solaris：相同优先级的线程不能相互抢占对方的cpu时间。
  - windows：可以抢占相同甚至更高优先级的线程的cpu时间

TestPriority.java

例：TestThread6.java

线程同步

```
public class Test implements Runnable {
  Timer timer = new Timer();
  public static void main(String[] args) {
    Test test = new Test();
    Thread t1 = new Thread(test);
    Thread t2 = new Thread(test);
    t1.setName("t1"); t2.setName("t2");
    t1.start(); t2.start();
  }
  public void run(){
    timer.add(Thread.currentThread().getName());
  }
}
class Timer{
  private static int num = 0;
  public void add(String name){
    num ++;
    try {Thread.sleep(1);}
    catch (InterruptedException e) {}
    System.out.println(name+", 你是第"+num+"个使用timer的线程");
  }
}
```

TT.java

- 死锁TestDeadLock.java
- 活锁:相互鞠躬的问题

TestSync.java

线程同步

- 在Java语言中，引入了对象互斥锁的概念，保证共享数据操作的完整性。每个对象都对应于一个可称为“互斥锁”的标记，这个标记保证在任一时刻，只能有一个线程访问该对象。
- 关键字synchronized 来与对象的互斥锁联系。当某个对象synchronized修饰时，表明该对象在任一时刻只能由一个线程访问。

➢ **synchronized 的使用方法：**

```
                    … … …
synchronized(this){
  num ++;
  try {Thread.sleep(1);}
  catch (InterruptedException e) {}
  System.out.println
    (name+", 你是第"+num+"个使用timer的线程");
}
                    … … …
```

➢ **synchronized 还可以放在方法声明中，表示整个方法为同步方法，例如：**

```
synchronized public void add(String name){…}
```

例： ProducerConsumer.java

总结

- 线程、进程的概念
- 线程的创建和启动方式
- 线程的调度和优先级