JAVA程序员培训-4

讲师：陈伟俊

第十五章

线　　程

本章内容

- 线程的概念模型
- 线程的创建和启动
- 线程的状态控制
- 线程的互斥和同步

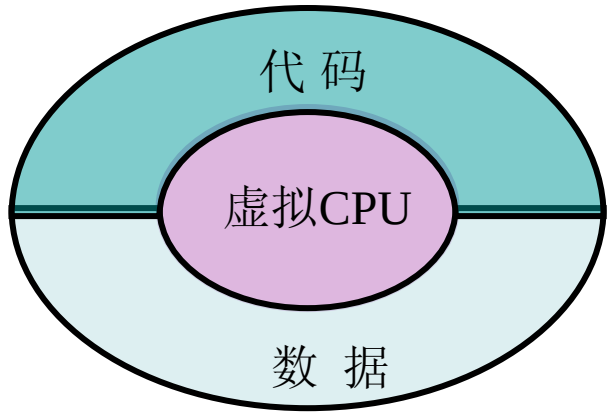
什么是线程

- 线程是一个程序内部的顺序控制流。
- 线程和进程
  - 每个进程都有独立的代码和数据空间(进程上下文)，进程切换的开销大。
  - 线程：轻量的进程，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

线程的概念模型

- 虚拟的CPU，由java.lang.Thread类封装和虚拟
- CPU所执行的代码，传递给Thread类对象。
- CPU所处理的数据，传递给Thread类对象。

Java线程模型



创建线程方式----继承Thread类

```
public class TestThread6 {
   public static void main(String args[]){
       Thread t = new Runner6();
       t.start();
    }
}

class Runner6 extends Thread {
   public void run() {
       for(int i=0;i<50;i++) {
             System.out.println("SubThread: " + i);
       }
   }
}
```

创建线程 ----通过实现Runnable接口

```
public class TestThread1 {
   public static void main(String args[]) {
            Runner1 r = new Runner1();
            Thread t = new Thread(r);
            t.start();
   }
      }

class Runner1 implements Runnable {
   public void run() {
              for(int i=0; i<30; i++) {
                    System.out.println("No. " + i);
              }
   }
}
```

两种创建线程方法的比较

- 使用Runnable接口

  可以将CPU，代码和数据分开，形成清晰的模型；

  还可以从其他类继承；

  保持程序风格的一致性。

- 直接继承Thread类

  不能再从其他类继承；

  编写简单，可以直接操纵线程，无需使用Thread.currentThread()。

线程体run()方法

- Java的线程是通过java.lang.Thread类来实现的。
- 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。

启动线程

- 使用start() 方法启动线程
- 启动线程是使线程进入到可运行(runnable)状态，并不一定立即开始执行该线程

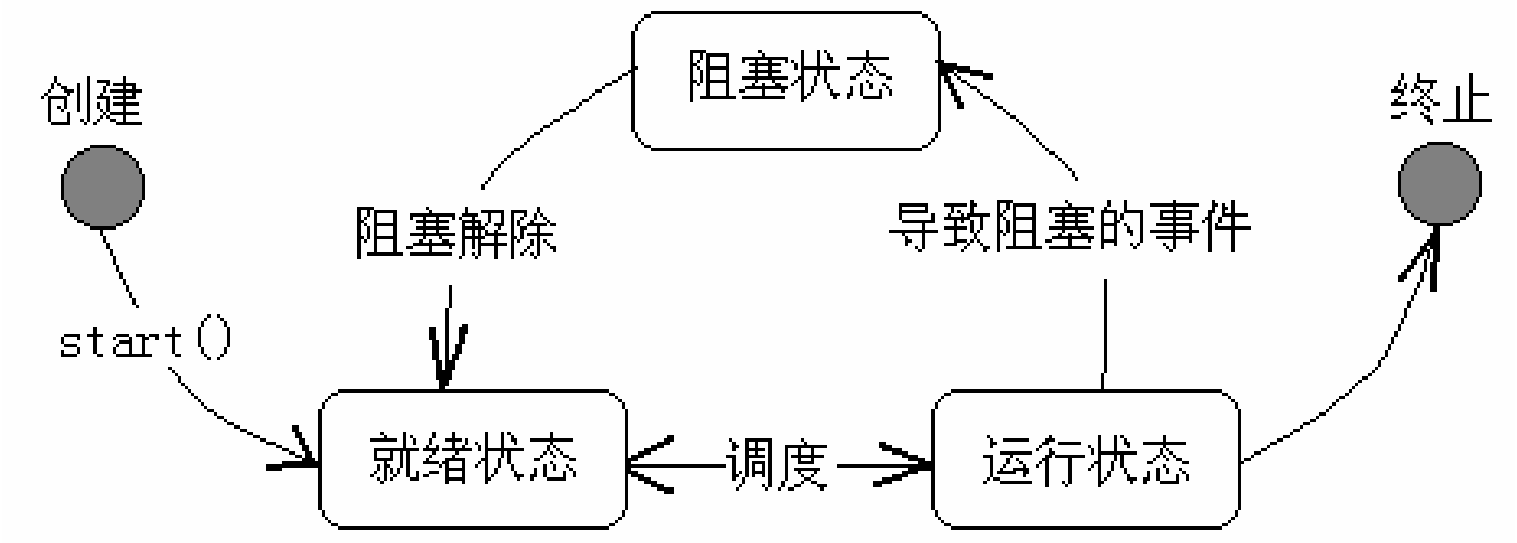
```
public class TestThread1 {
      public static void main(String args[]) {
          Runner1 r = new Runner1();
          Thread t = new Thread(r);
          t.start();
      }
}
```

多线程举例

```
public class TestThread2 {
   public static void main(String args[]) {
        Runner2 r = new Runner2();
        Thread t1 = new Thread(r);
        Thread t2 = new Thread(r);
        t1.start();
        t2.start();
   }
}

class Runner2 implements Runnable {
   public void run() {
        for(int i=0; i<30; i++) {
               System.out.println("No. " + i);
        }
   }
}
```

线程状态转换(Thread Scheduling)



线程状态转换举例

```
public class TestThread3{
   public static void main(String args[]) {
        Runner3 r = new Runner3();
        Thread t = new Thread(r);
        t.start();
   }
}

class Runner3 implements Runnable {
   public void run() {
        for(int i=0; i<30; i++) {
              if(i%10==0 && i!=0) {
                    try{
                          Thread.sleep(2000);
                    }catch(InterruptedException e){}
              }
              System.out.println("No. " + i);
        }
   }
}
```

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（1－10）
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等待池（wait pool）。
notify()/notifyAll()	唤醒对象的wait pool中的一个/所有等待线程。

join方法用法举例

```
public class TestThread5 {
   public static void main(String args[]){
        Runner5 r = new Runner5();
                Thread t = new Thread(r);
                        t.start();
                        try{
                                t.join();
                        }catch(InterruptedException e){
                        }
                        for(int i=0;i<50;i++){
                                System.out.println("主线程:" + i);
                        }
        }
}

class Runner5 implements Runnable {
   public void run() {
        for(int i=0;i<50;i++)
                System.out.println("SubThread: " + i);
   }
}
```

线程的调度

- 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些线程来执行。
  - setPriority(int)方法设置优先级
- 多数线程的调度是抢先式的。
  - 时间片方式
  - 非时间片方式

线程的调度

- 下面几种情况下，当前线程会放弃CPU:
  - 线程调用了yield()，或sleep()方法主动放弃;
  - 由于当前线程进行I/O访问，外存读写，等待用户输入等操作，导致线程阻塞;
  - 线程调用wait（）方法;
  - 抢先式系统下，有高优先级的线程参与调度;时间片方式下，当前时间片用完，有同优先级的线程参与调度。

线程的优先级

- 线程的优先级用数字来表示，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);
```

互斥锁

- 在Java语言中，引入了对象互斥锁的概念，来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，这个标记用来保证在任一时刻，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。当某个对象用synchronized修饰时，表明该对象在任一时刻只能由一个线程访问。

关键字Synchronized

- synchronized 除了象上面放在对象前面限制一段代码的执行外，还可以放在方法声明中，表示整个方法为同步方法。

```
public synchronized void push(char c){
        …
}
```

- 如果synchronized用在类声明中，则表明该类中的所有方法都是synchronized的。

```
public synchronized class Stack{
        …
}
```

本章结束