Java多线程

于策

Java 多线程

- 多线程的概念；
- 线程的生命周期；
- 多线程编程中的常量和方法；
- 线程调度方法；
- 资源冲突与协调；
- 线程之间的通信。

Java中多线程的基本概念

- 在多线程模型中，多个线程共存于同一块内存中，且共享资源。
- 每个线程分配有限的时间片来处理任务。由于CPU在各个线程之间的切换速度非常快，用户感觉不到，从而认为并行运行。

多线程的特点

- 多个线程在运行时，系统自动在线程之间进行切换；
- 由于多个线程共存于同一块内存，线程之间的通信非常容易；
- Java将线程视为一个对象。线程要么是Thread类的对象，要么是接口Runnable的对象。

多线程的特点(续)

- 当多个线程并行执行时，具有较高优先级的线程将获得较多的CPU时间片；
- 优先级是从0到10的整数，并且它仅表示线程之间的相对关系；
- 多个线程共享一组资源，有可能在运行时产生冲突。必须采用synchronized关键字协调资源，实现线程同步。

多线程编程中常用的常量和方法

- 线程类Thread定义在java.lang包中；
- Thread类包含的常量有：

1. public static final int MAX_PRIORITY： 最大优先级，值是10。
2. public static final int MIN_PRIORITY： 最小优先级，值是1。
3. public static final int NORM_PRIORITY：缺省优先级，值是5。

多线程编程中常用的常量和方法(续)

- 常用方法：

currentThread( )：返回当前运行的线程对象，是一个静态的方法。

sleep(int n)：使当前运行的线程睡n个毫秒，然后继续执行，也是静态方法。

yield( )：使当前运行的线程放弃执行，切换到其它线程，是一个静态方法。

isAlive( )：判断线程是否处于执行的状态，返回值true表示处于运行状态，false表示已停止。

多线程编程中常用的常量和方法(续)

start( )：使调用该方法的线程开始执行。

run( )：该方法由start( )方法自动调用。

stop( )：使线程停止执行，并退出可执行状态。

suspend():使线程暂停执行，不退出可执行态。

resume( )：将暂停的线程继续执行。

setName(String s)：赋予线程一个名字。

getName( )：获得调用线程的名字。

多线程编程中常用的常量和方法(续)

getPriority( )：获得调用线程的优先级。

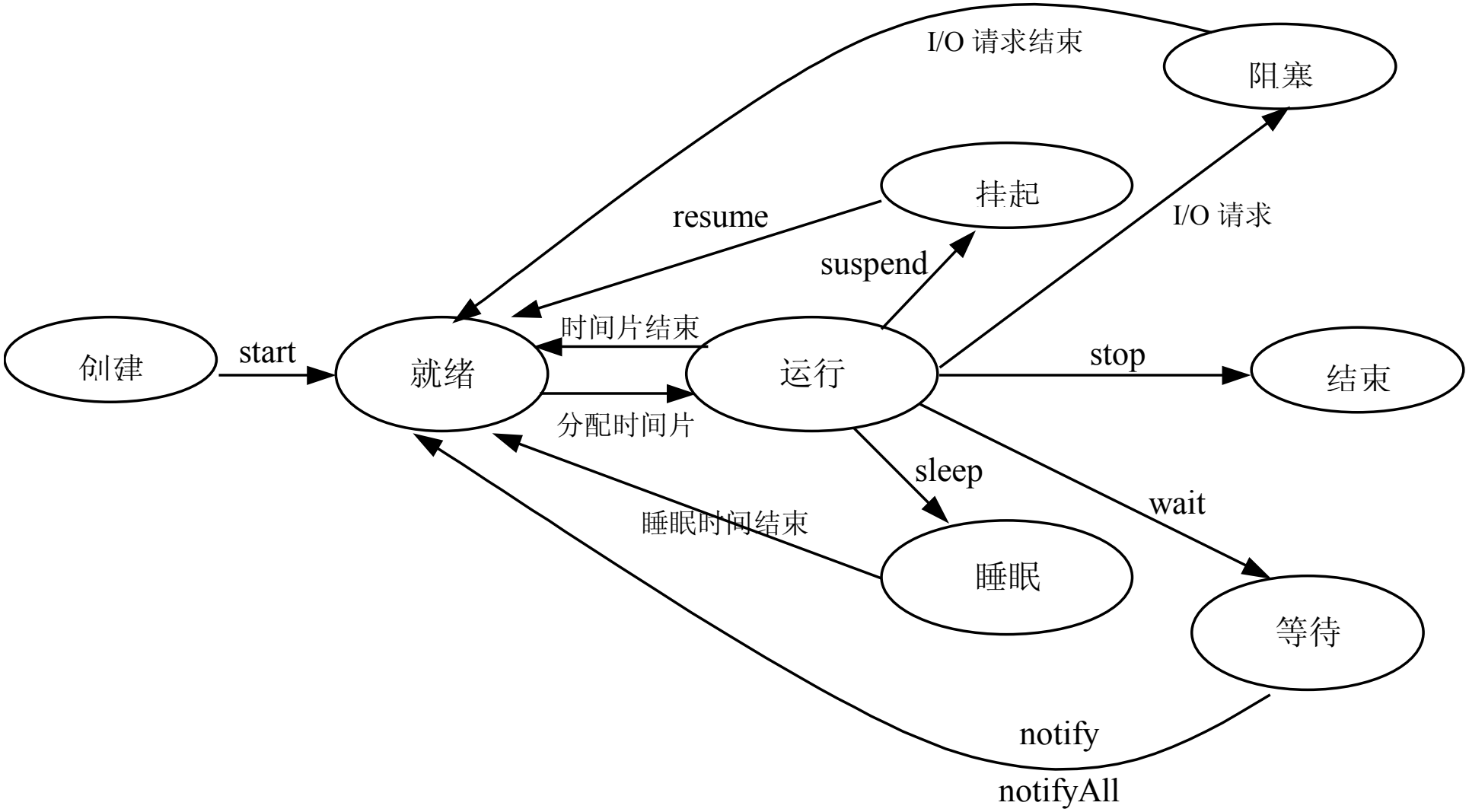
setPriority(int p)：设置线程的优先级。

join( )：等待线程死亡，若中断了该线程，将抛出异常。

多线程编程中常用的常量和方法(续)

- 注意1：在创建线程对象时，缺省的线程优先级是5，一般设置优先级4到6之间，不要设置为10，否则其它线程将执行不到。
- 注意2：Java的调度器能使高优先级的线程始终运行，一旦CPU有空闲，具有同等优先级的线程，以轮流的方式顺序使用时间片。

线程的生命周期


I/O 请求结束
阻塞
挂起
resume
suspend
I/O 请求
创建
start
就绪
时间片结束
运行
stop
结束
分配时间片
sleep
wait
睡眠时间结束
睡眠
等待
notify
notifyAll

```
class getThreadInfo {      // 单线程示例
   public static void main(String args[ ]) {
            String name;
            int p;
            Thread curr;

            curr=Thread.currentThread( );
            System.out.println("当前线程: "+curr);
            name=curr.getName( );
            p=curr.getPriority( );
            System.out.println("线程名: "+name);
            System.out.println("优先级 :"+p);
      }
}
```

程序输出结果：

当前线程: Thread[main,5,main]

线程名 : main

优先级 :5

创建多线程的方法

- 方法1：通过Thread类的子类实现多线程。
- 方法2：定义一个实现Runnable接口的类实现多线程。

创建多线程的方法(续)

- 方法1：通过创建Thread类的子类实现多线程，步骤如下 ：

1. 定义Thread类的一个子类。
2. 定义子类中的方法run( )，覆盖父类中的方法run( )。
3. 创建该子类的一个线程对象。
4. 通过start( )方法启动线程。例如程序2：

```
// 程序2
class UserThread extends Thread{
    int sleepTime;

    public UserThread(String id) {  // 构造函数
        super(id);

        sleepTime=(int)(Math.random( )*1000);
        System.out.println("线程名：  "+getName( )+
                        "，睡眠：  "+sleepTime+" 毫秒");
    }
```

```
public void run( ) {
        try{  // 通过线程睡眠模拟程序的执行
                Thread.sleep(sleepTime);
        }catch(InterruptedException e) {
                System.err.println("运行异常:" +
                        e.toString( ));
        }

        System.out.println("运行的线程是：  "+
                        getName( ));
    }
}
```

```
public class multThreadTest{
        public static void main(String args[ ]) {
                UserThread      t1,t2,t3,t4;

                t1=new UserThread("NO 1");
                t2=new UserThread("NO 2");
                t3=new UserThread("NO 3");
                t4=new UserThread("NO 4");

                t1.start( );        t2.start( );
                t3.start( );        t4.start( );
        }
}
```

程序某次的运行结果：

线程名： NO 1，睡眠： 885 毫秒
线程名： NO 2，睡眠： 66 毫秒
线程名： NO 3，睡眠： 203 毫秒
线程名： NO 4，睡眠： 294 毫秒
目前运行的线程是：NO 2
目前运行的线程是：NO 3
目前运行的线程是：NO 4
目前运行的线程是：NO 1

注意：Thread类中的run( )方法具有public属性，覆盖该方法时，前面必须带上public。

创建多线程的方法(续)

- 方法2：通过接口创建多线程，步骤如下：

1.定义一个实现Runnable接口的类。
2.定义方法run( )。Runnable接口中有一个空的方法run( )，实现它的类必须覆盖此方法。
3.创建该类的一个线程对象，并将该对象作参数，传递给Thread类的构造函数，从而生成Thread类的一个对象。 // 注意这一步!
4.通过start( )方法启动线程。例如：

```
// 程序3
class UserMultThread implements Runnable{
        int num;
        UserMultThread(int n) {
                num=n;
        }

        public void run( ) {
                for(int i=0;i<3;i++)
                    System.out.println("运行线程： "+num);

           System.out.println("结束 ： "+num);
        }
}
```

```
public class multThreadZero {
        public static void main(String args[ ])
                        throws InterruptedException  {
                Thread mt1=new Thread(
                                new UserMultThread(1));
                Thread mt2=new Thread(
                                new UserMultThread(2));

                mt1.start( );
                mt2.start( );
                mt1.join( );      // 等待线程死亡
                mt2.join( );
        }
}
```

程序运行某次的输出结果：

运行线程：1
运行线程：2
运行线程：1
运行线程：2
运行线程：1
运行线程：2
结束 ： 1
结束 ： 2

创建多线程的方法(续)

- 程序3中需要注意的2点：

1. mt1.join( )是等待线程死亡，对该方法必须捕捉异常，或通过throws关键字指明可能要发生的异常。

2. 对一个线程不能调用start( )两次，否则会产生IllegalThreadStateException异常。

线程调度模型

- 线程调度程序挑选线程时，将选择处于就绪状态且优先级最高的线程。
- 如果多个线程具有相同的优先级，它们将被轮流调度。
- 程序4验证了Java对多线程的调度方法。

```
class threadTest extends Thread{      // 程序4
        threadTest(String str){  super(str);  }

        public void run( ){
                try{
                        Thread.sleep(2);
                }catch(InterruptedException e) {
                        System.err.println(e.toString( ));
                }

                System.out.println(getName( )+" " +
                                        getPriority( ));
        }
}
```

```
public class multTheadOne{
    public static void main(String agrs[]){
        Thread one=new threadTest("one" );
        Thread two=new threadTest("two" );
        Thread three=new threadTest("three" );

        one.setPriority(Thread.MIN_PRIORITY);
        two.setPriority(Thread.NORM_PRIORITY);
        three.setPriority(Thread.MAX_PRIORITY);

        one.start( );
        two.start( );
        three.start( );
    }
}
```

程序输出结果：

three 10

two 5

one 1

思考：在run()方法中，通过线程睡眠2个毫秒，模拟程序的执行。如果不睡眠，将会输出什么结果？为什么？

资源冲突

- 多个线程同时运行虽然可以提高程序的执行效率，但由于共享一组资源，可能会产生冲突，例如程序5 。

```
class UserThread{      // 程序5
        void Play(int n) {
                System.out.println("运行线程 NO：  "+n);

                try{
                        Thread.sleep(3);
                }catch(InterruptedException e) {
                    System.out.println( “线程异常, NO：  "+n);
                }

                System.out.println("结束线程 NO：  "+n);
        }
}
```

```
class UserMultThread implements Runnable{
        UserThread UserObj;
        int num;

        UserMultThread(UserThread o,int n) {
                UserObj=o;
                num=n;
        }

        public void run( ) {
                UserObj.Play(num);
        }
}
```

```
public class multTheadTwo {
    public static void main(String args[ ]) {
                UserThread Obj=new UserThread( );

                Thread t1=new Thread(
                                new UserMultThread(Obj,1));
                Thread t2=new Thread(
                                new UserMultThread(Obj,2));
                Thread t3=new Thread(
                                new UserMultThread(Obj,3));

                t1.start( );   t2.start( );   t3.start( );
        }
}
```

程序输出结果：

运行线程 NO：1
运行线程 NO：3
结束线程 NO：1
结束线程 NO：3
运行线程 NO：2
结束线程 NO：2

同步方法

- Java通过关键字synchronized实现同步。
- 当对一个对象(含方法)使用synchronized，这个对象便被锁定或者说进入了监视器。在一个时刻只能有一个线程可以访问被锁定的对象。它访问结束时，让高优先级并处于就绪状态的线程，继续访问被锁定的对象，从而实现资源同步。
- 加锁的方法有两种：锁定冲突的对象，或锁定冲突的方法。

同步方法(续)

1. 锁定冲突的对象。语法格式：

```
synchronized ( ObjRef ){
        Block              // 需要同步执行的语句体
}
```

锁定对象可以出现在任何一个方法中。例如，修改程序9-5中的方法run( ) 如下：

```
public void run( ) {
        synchronized(UserObj) {
                UserObj.Play(num);
        }
}
```

运行结果如下：

运行线程 NO：1
结束线程 NO：1
运行线程 NO：2
结束线程 NO：2
运行线程 NO：3
结束线程 NO：3

同步方法(续)

2. 锁定冲突的方法。语法格式：

synchronized　　方法的定义

例如，修改程序5中的方法Play( ) 如下：

```
synchronized void Play(int n) {
        …… // 中间的程序代码略
}
```

注意：

1. 对方法run( )无法加锁，不可避免冲突；
2. 对构造函数不能加锁，否则出现语法错误。

线程间通信

- 多线程通信的方法有两种：

1. 把共享变量和方法封装在一个类中实现；
2. 通过wait( )和notify( )方法实现。

通过封装共享变量实现线程通信

程序6演示了通过将共享变量封装在一个类中，实现线程通信

```
class comm{      //共享类
        private int n;
        private boolean bool=false;
        void produce(int i) {
                while(bool) {    }
                n=i;
                bool=true;
                System.out.println("\n 产生数据： "+n);
        }
        void readout( ) {
                while(!bool) {   }
                bool=false;
                System.out.println(" 读取数据： "+n);
        }
}
```

```
    // 读取数据类
class dataProducer implements Runnable{
        comm cm;

        dataProducer(comm c) {
                cm=c;
        }

        public void run( ) {   // 生产者产生5个随机数
                for(int i=0;i<5;i++)
                cm.produce((int)(Math.random( )*100));
        }
}
```

```
class dataConsumer implements Runnable{
        comm cm;

        dataConsumer(comm c) {
                cm=c;
        }

        public void run( ) {
                for(int i=0;i<5;i++)
                    cm.readout( );   // 消费者读取数据
        }
}
```

```
public class multTheadThree {
   public static void main(String args[ ]) {
            comm cm=new comm( );

            Thread t1=new Thread( //无名对象做参数
                              new dataProducer(cm));
            Thread t2=new Thread(
                    new dataConsumer(cm));

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

程序的某次运行结果：

产生数据：81
读取数据：81

产生数据：15
读取数据：15

产生数据：92
读取数据：92

产生数据：98
读取数据：98

产生数据：79
读取数据：79

通过系统方法实现线程通信

- 线程同步控制的第二种方法是采用如下几个系统方法：

1.wait( )方法：使一个线程进入等待状态，直到被唤醒。

2.notify( )方法：通知等待监视器的线程，该对象的状态已经发生了改变。

3. notifyAll( )方法：唤醒从同一个监视器中用wait( )方法退出的所有线程，使它们按照优先级的顺序重新排队。例如程序7。

```
class comm{     // 程序7
  private int n;
  private boolean bool=false;
  synchronized void produce(int i) {
        if(bool) {
              try{
                    wait( );
              }catch(InterruptedException e) {
                 System.out.println("comm 出现异常");
              }
        }else{
              n=i;
              bool=true;
              System.out.println("\n产生数据：  "+n);
              notify( );  // 唤醒另外一个线程
        }
  }
```

```
synchronized void readout( ) {
        if(!bool) {
                try{
                        wait( );
                }catch(InterruptedException e) {
                        System.out.println(" comm 出现异常");
                }
        }else{
                bool=false;
                System.out.println("读取数据：  "+n);
                notify( );  // 唤醒另一个线程
        }
 }
```

```
class dataProducer  implements Runnable{
        comm cm;
        dataProducer (comm c) {   cm=c;   }

        public void run( ) {
          try{
                for(int i=0;i<5;i++) {
                    cm.produce((int)(Math.random( )*100));
                    Thread.sleep(10);
                }
           }catch(InterruptedException e) {
                System.out.println("dataProducer 出现异常");
                }
        }
}
```

```
class dataConsumer implements Runnable{
        comm cm;
        dataConsumer(comm c) {cm=c;          }

        public void run( ) {
            try{
                for(int i=0;i<5;i++) {
                    cm.readout( );
                     Thread.sleep(10);
                 }
             }catch(InterruptedException e) {
                System.out.println(" dataConsumer 出现异常");
             }
        }
}
```

```
public class multTheadFour{
   public static void main(String args[ ]) {
        comm cm=new comm( );
        Thread t1=new Thread(new dataProducer (cm));
        Thread t2=new Thread(new dataConsumer(cm));

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

程序输出结果：

产生数据： 90
读取数据： 90

产生数据： 10
读取数据： 10

产生数据： 65
读取数据： 65

产生数据： 45
读取数据： 45

线程调度

- 线程进入等待队列的方式有：一、线程a正在使用某对象，线程b调用该对象的同步方法，则b进入队列；二、调用wait( )方法使线程进入队列。
- 当一个同步方法调用返回时，或一个方法调用了wait( )方法时，另一线程就可访问冲突的对象。
- 线程调度程序在队列中选取优先级最高的线程。
- 如果一个线程因调用wait( )而进入队列，则必须调用notify( )才能“解冻”该线程。