

## 笔记总链接:http://bbs.itheima.com/thread-200600-1-1.html

5、多线程 5.3 线程间通信

```
5.3.1 线程间通信涉及的方法
```

多个线程在处理统一资源,但是任务却不同,这时候就需要线程间通信。

监视器其实就是锁。

22. 23.

24.

26.

```
等待/唤醒机制涉及的方法:
1. wait():让线程处于冻结状态,被wait的线程会被存储到线程池中。
```

2. notify():唤醒线程池中的一个线程(任何一个都有可能)。 3. notifyAll():唤醒线程池中的所有线程。 P.S. 1、这些方法都必须定义在同步中,因为这些方法是用于操作线程状态的方法。

2、必须要明确到底操作的是哪个锁上的线程! 3、wait和sleep区别?

1) wait可以指定时间也可以不指定。sleep必须指定时间。 2)在同步中时,对CPU的执行权和锁的处理不同。 wait:释放执行权,释放锁。 sleep:释放执行权,不释放锁。

锁可以是任意的对象,任意的对象调用的方式一定在object类中。 生产者-消费者问题: class Resource{ 02. private String name ; private String sex ; 03.

为什么操作线程的方法wait、notify、notifyAll定义在了object类中,因为这些方法是监视器的方法,

private boolean flag = false; 05. public synchronized void set(String name,String sex){ 06. if(flag ) 07. try{ 08.

this.wait(); 09. } catch(InterruptedException e){ 10. e.printStackTrace(); 11. }

this.name = name; 13. 14. this.sex = sex;

this.wait();

} catch(InterruptedException e){ e.printStackTrace();

System. out.println(name + "..." + sex);

15. flag = true ; this.notify(); 16. 17. 18. public synchronized void out(){ 19. if(!flag ) 20. try{ 21.

}

flag = false ; 27. this.notify(); 28. 29. } 30. } 31. //输入 32. class Input implements Runnable{ 33. Resource r; 34. Input(Resource r){ this.r = r; 36. 37. } 38. public void run(){ 39. int x = 0; 40. while(true ){ 41.  $if(x == 0){$ 42. r.set( "mike", "男" ); 43. } else{ 44. r.set( "lili", "女" ); 45. 46. x = (x + 1)%2;47. 48. } 49. } } 50. 51. //输出 52. class Output implements Runnable{ Resource r; 54. 55. Output(Resource r){ 57. this.r = r; 58. } 59. public void run(){ 60. while(true ){ r.out(); 62. 63. 64. } } 65. 66. class ResourceDemo { 67.

public static void main(String[] args){ 68. //创建资源 69. Resource r = new Resource();70. //创建任务 71. Input in = new Input(r); 72. Output out = new Output(r); 73. //创建线程,执行路径 Thread t1 = new Thread(in); 75. Thread t2 = new Thread(out); 76. //开启线程 77. t1.start(); 78. 79. t2.start(); } 80. 81. } 复制代码 运行结果: \_ 0 X 📷 管理员: C:\Windows\system32\cmd.exe D:\code\day14>javac ResourceDemo.java D:∖code∖day14>java ResourceDemo mike...男 lili...女 mike...男 lili....安 mike...男 mike.. lili... 多生产者-多消费者问题: class Resource{ private String name ; 02. 03. private int count = 1; 04. private boolean flag = false; 05. public synchronized void set(String name){ if(flag ) 07. 08. try{

wait();

this.name = name + count;

count++;

notify();

if(!flag )

}

flag = false ;

notify();

class Producer implements Runnable{

Producer(Resource r){

this.r = r;

while(true ){

r.set( "烤鸭");

private Resource r ;

public void run(){

者..." + this. name);

}

}

flag = true ;

public synchronized void out(){

wait();

} catch(InterruptedException e){

e.printStackTrace();

try{

者..." + this. name);

} catch(InterruptedException e){

e.printStackTrace();

System.out.println(Thread.currentThread().getName() + "...生产

System.out.println(Thread.currentThread().getName() + "...消费

Ξ

## 40. } 41. 42. }

}

09.

10.

11. 12.

13.

15.

16.

17. 18. 19.

20.

21.

22.

23.

24. 25.

26.

27. 28.

29.

30.

31. 32.

33.

34.

36.

37.

39.

43.

44.

}

代码:

class Resource{

private String name ;

private int count = 1;

while(flag )

count++;

flag = true ;

notifyAll();

while(!flag )

}

flag = false ;

notifyAll();

class Producer implements Runnable{

private Resource r ; Producer(Resource r){

者..." + this. name);

}

public synchronized void out(){

try{

者..." + this. name);

}

private boolean flag = false;

try{

public synchronized void set(String name){

this.name = name + count;

this.wait();

this.wait();

} catch(InterruptedException e){ e.printStackTrace();

} catch(InterruptedException e){

e.printStackTrace();

System.out.println(Thread.currentThread().getName() + "...生产

System.out.println(Thread.currentThread().getName() + "...消费

01.

02.

03. 04.

07.

08.

09.

10.

11.

14.

15.

16. 17.

20.

21. 22.

23.

24.

26.

27.

28.

29.

30.

31. 32. 33.

34.

35.

54.

55. 56.

58. 59.

61.

62.

63. 64.

65.

66. 67.

68.

69.

70. 71. 72.

}

运行结果:

复制代码

画 管理员: C:\Windows\system32\cmd.exe

D: Gode May147 Java Product
Thread-0...生产者...烤鸭1
Thread-2...消费者...烤鸭2
Thread-1...生产者...烤鸭2
Thread-1...生产者...烤鸭3
Thread-2...消费者...烤鸭3
Thread-0...生产者...烤鸭3
Thread-0...生产者...烤鸭4

Thread-3...消费者...烤鸭8 Thread-1...牛产者...烤鸭9

活,可以一个锁上加上多组监视器。

封装,变成Condition监视器对象,可以任意锁进行组合。

Condition接口中的await方法对应于Object中的wait方法。

lock(): 获取锁。

同步代码块就是对于锁的操作是隐式的。

JDK1.5以后将同步和锁封装成了对象,并将操作锁的隐式方式定义到了该对象中,将隐式动作变成了

Lock接口:出现替代了同步代码块或者同步函数,将同步的隐式操作变成显示锁操作。同时更为灵

unlock():释放锁,为了防止异常出现,导致锁无法被关闭,所以锁的关闭动作要放在finally中。

Condition接口:出现替代了Object中的wait、notify、notifyAll方法。将这些监视器方法单独进行了

Thread-2...消费 Thread-1...生产

Thread-3...消费 Thread-1...生产

Thread-2...消费 Thread-0...生产

Thread-2...消费 Thread-1...生产

5.3.2 JDK1.5新特性

显示动作。

}

}

57. class ProducerConsumerDemo {

public static void main(String[] args){

Resource r = new Resource(); Producer pro = new Producer(r);

Thread t0 = new Thread(pro);

Thread t1 = new Thread(pro); Thread t2 = new Thread(con);

Thread t3 = new Thread(con);

\_ O X

t0.start();

t1.start();

t2.start(); t3.start();

D:∖code∖day14>javac ProducerConsumerDemo.java

D:\code\day14>java ProducerConsumerDemo

Consumer con = new Consumer(r);

}

class Consumer implements Runnable{ 45. private Resource r ; 46. Consumer(Resource r){ 47. 48. this.r = r; 49. } public void run(){ 50. while(true ){ 51. r.out(); 52. 54. } } 55. 56. 57. class ProducerConsumerDemo { public static void main(String[] args){ 58. Resource r = new Resource(); 59. Producer pro = new Producer(r); 60. Consumer con = new Consumer(r); 62. Thread t0 = new Thread(pro); 63. Thread t1 = new Thread(pro); 64. Thread t2 = new Thread(con); 65. Thread t3 = new Thread(con); 66. t0.start(); 67. 68. t1.start(); 69. t2.start(); 70. t3.start(); 71. } 72. } 复制代码 运行结果:以上代码存在安全问题。 画 管理员: C:\Windows\system32\cmd.exe D:\code\day14>javac ProducerConsumerDemo.java D:∖code∖day14>java ProducerConsumerDemo Thread-0...生产者...烤鸭1 Thread-3...消费者...烤鸭1 Thread-1...生产者...烤鸭2 Thread-3...消费者...烤鸭2 Thread-2...消费者...烤鸭2 原因分析: 得到以上结果的过程分析如下: 1. 线程Thread-0获取到CPU执行权及锁,生产了烤鸭3298,将flag设置为true。然后,Thread-0又 重新获取到CPU执行权,由于flag为true,故执行wait方法,阻塞。Thread-1接着获取到CPU执行权, 由于flag为true,故执行wait方法,也阻塞。 public synchronized void set(String name) { if (flag) try { wait(); } catch (InterruptedException e) { e.printStackTrace(); this .name = name + count; count ++; System.out .println(Thread.currentThread().getName() + "...生产者..." + this.name ); flag = true ; notify(); 2. 线程Thread-3获取到CPU执行权及锁,消费了烤鸭3298,将flag设置为false。然后,线程Thread-0被唤醒,但是并没有获取到锁,而是线程Thread-3接着获取到CPU执行权及锁,然而此时flag为false, 所以Thread-3阻塞。下面线程Thread-2接着获取到CPU执行权及锁,然而此时flag为false,所以 Thread-2也阻塞。 public synchronized void out() { if (!flag) try { } catch (InterruptedException e) { e.printStackTrace(); } flag = false; notify(); System.out .println(Thread.currentThread().getName() + "...消费者..." + this.name ); 3. 线程Thread-0获取到CPU执行权及锁,不需要if语句判断,直接生产烤鸭3299,然后又唤醒线程 Thread-1获取到CPU执行权及锁,不需要if语句判断,直接生产烤鸭3300。从而造成了烤鸭3299还没有 被消费,就直接生产了烤鸭3300的情况。 由于if判断标记,只有一次,会导致不该运行的线程运行了,出现了数据错误的情况。故修改成while 判断标记,线程获取CPU执行权及锁后,将重新判断是否具备运行条件。 notify方法只能唤醒一个线程,如果本方唤醒了本方,没有意义。而且while判断标记+notify会导致死 锁。notifyAll解决了本方线程一定会唤醒对方线程的问题。 P.S. while判断标记+notify会导致死锁的示例: 如果将上面的代码中的if判断标记修改成wile判断标记,就会出现死锁的现象,前2步与原来是一致 的。第3步如下: 3. 线程Thread-0获取到CPU执行权及锁,通过了while语句判断,直接生产烤鸭3299,将flag设置为

true。然后又唤醒线程Thread-1获取到CPU执行权及锁,没有通过while语句判断,阻塞。线程Thread-0

又获取到CPU执行权及锁,通不过while语句判断,也阻塞,此时Thread-0、1、2、3都阻塞,故死锁。

Ξ

36. this.r = r; 37. public void run(){ while(true ){ r.set( "烤鸭"); 40. 41. } 42. } 43. 44. 45. class Consumer implements Runnable{ private Resource r ; 47. Consumer(Resource r){ this.r = r;48. 49. } public void run(){ 50. while(true ){ 51. 52. r.out(); }

Condition接口中的signal方法对应于Object中的notify方法。 Condition接口中的signalAll方法对应于Object中的notifyAll方法。 使用一个Lock、一个Condition修改上面的多生产者-多消费者问题。 代码: import java.util.concurrent.locks.\*; 01. class Resource{ 02. private String name; 03. private int count = 1; 04. private boolean flag = false; 05. //创建一个锁对象 07. Lock lock = new ReentrantLock(); 08. 09. //通过已有的锁获取该锁上的监视器对象 10. Condition con = lock .newCondition(); 11. 12. public void set(String name){ 13. lock.lock(); 15. try{ while(flag) 16. 17. try{ con.await(); 18. } catch(InterruptedException e){ 19. e.printStackTrace(); 20. 21. this.name = name + count; 22. 23. count++; 24. name); flag = true ; 25. 26. con.signalAll(); }finally{ 27. lock.unlock(); 28. 29. }

System.out.println(Thread.currentThread().getName() + "...生产者..." + this. 30. } 31. public void out(){ 32. lock.lock(); 33. try{ 34. while(!flag ) 35. try{ 36. 37. con.await(); } catch(InterruptedException e){ 38. e.printStackTrace(); 39. 40. flag = false; 41. 42. con.signalAll(); System.out.println(Thread.currentThread().getName() + "...消费者..." + this. name); 44. }finally{ lock.unlock(); 46. } 47. 48. 49. class Producer implements Runnable{ 50. private Resource r; 51. Producer(Resource r){ 52. this.r = r;53. 54. public void run(){ 55. while(true){ r.set( "烤鸭"); 57. 58. 59. 60. 61. class Consumer implements Runnable{ 62. private Resource r; 63. Consumer(Resource r){ this.r = r;65. 66. } public void run(){ 67. while(true){ 68. r.out(); 69. 70. 71. 72. 73. class ProducerConsumerDemo { 74. public static void main(String[] args){ 75. Resource r = new Resource(); Producer pro = new Producer(r); 77. Consumer con = new Consumer(r); 78. 79. 80. Thread t0 = new Thread(pro); Thread t1 = new Thread(pro); 81. Thread t2 = new Thread(con);82. Thread t3 = new Thread(con);t0.start(); 84. t1.start(); 85. t2.start(); 86. t3.start(); 87. 88. } 89.

复制代码

运行结果:

```
_ 0 X
📷 管理员: C:\Windows\system32\cmd.exe
D:\code\day14>javac ProducerConsumerDemo.java
D:\code\day14>java ProducerConsumerDemo
Thread-0...生产者...烤鸭1
Thread-2...消费者...烤鸭1
Thread-0...生产者...烤鸭2
Thread-2...消费者...烤鸭2
Thread-0...生产者...烤鸭2
Thread-2...消费
Thread-0...生产
Thread-0...生产
Thread-2...消费
Thread-0...生产
Thread-0...
Thread-0... 生产
Thread-2... 消费
Thread-0... 生费
Thread-2... 消费
Thread-2... 消费
Thread-2... 消费
Thread-2... 消费
                                                                                                                               Ε
Thread-0...
Thread-2...消费
  使用一个Lock、两个Condition修改上面的多生产者-多消费者问题。
  代码:
          import java.util.concurrent.locks.*;
```

class Resource{ 02. private String name ; 03. private int count = 1; 04.

```
private boolean flag = false;
05.
             //创建一个锁对象
07.
             Lock lock = new ReentrantLock();
08.
09.
             //通过已有的锁获取该锁上的监视器对象
10.
             Condition con = lock .newCondition();
12.
             //通过已有的锁获取两组监视器,一组监视生产者,一组监视消费者
13.
             Condition producer_con = lock .newCondition();
             Condition consumer_con = lock .newCondition();
15.
16.
             public void set(String name){
17.
                   lock.lock();
18.
19.
                   try{
20.
                         while(flag )
21.
                               try{
                                    producer_con.await();
22.
                              } catch(InterruptedException e){
23.
24.
                                    e.printStackTrace();
25.
                              }
                         this.name = name + count;
26.
27.
                         System.out.println(Thread.currentThread().getName() + "...
28.
      生产者..." + this. name);
29.
                         flag = true ;
                         consumer_con.signal();
30.
                  } finally{
                         lock.unlock();
32.
33.
            }
34.
35.
            public void out(){
                  lock.lock();
37.
                   try{
38.
39.
                         while(!flag )
40.
                               try{
                                     consumer_con.await();
41.
                              } catch(InterruptedException e){
42.
                                    e.printStackTrace();
43.
                              }
                         flag = false ;
45.
                         producer_con.signal();
46.
                         System.out.println(Thread.currentThread().getName() + "...
47.
      消费者..." + this. name);
48.
                  } finally{
49.
                         lock.unlock();
50.
51.
            }
     }
52.
53.
54.
     class Producer implements Runnable{
            private Resource r;
55.
             Producer(Resource r){
56.
                   this.r = r;
57.
58.
             public void run(){
59.
                   while(true ){
60.
                        r.set( "烤鸭");
61.
                  }
62.
63.
            }
64.
65.
     class Consumer implements Runnable{
67.
             private Resource r ;
             Consumer(Resource r){
68.
                   this.r = r;
69.
70.
             }
             public void run(){
71.
                   while(true ){
72.
73.
                         r.out();
74.
            }
75.
     }
76.
77.
    class ProducerConsumerDemo {
78.
79.
             public static void main(String[] args){
                  Resource r = new Resource();
80.
                  Producer pro = new Producer(r);
81.
                  Consumer con = new Consumer(r);
83.
                  Thread t0 = new Thread(pro);
84.
                  Thread t1 = new Thread(pro);
85.
                  Thread t2 = new Thread(con);
86.
87.
                  Thread t3 = new Thread(con);
                  t0.start();
88.
89.
                  t1.start();
90.
                  t2.start();
91.
                  t3.start();
92.
            }
```

```
final Condition notEmpty = lock .newCondition();
08.
09.
         final Object[] items = new Object[100];
         int putptr, takeptr , count ;
11.
12.
         public void put(Object x) throws InterruptedException {
13.
           lock.lock();
14.
           try {
            while (count == items.length)
16.
               notFull.await();
17.
               items[putptr ] = x;
18.
               if (++putptr == items.length) putptr = 0;
19.
20.
               ++ count;
               notEmpty.signal();
21.
22.
           } finally {
              lock.unlock();
         }
24.
       }
25.
26.
         public Object take() throws InterruptedException {
27.
           lock.lock();
28.
29.
           try {
            while (count == 0)
30.
             notEmpty.await();
```

- - X

Ε

}

运行结果:

复制代码

画 管理员: C:\Windows\system32\cmd.exe

D: Node May147 Java Product
Thread-0...生产者...烤鸭1
Thread-3...消费者...烤鸭2
Thread-2...消费者...烤鸭2
Thread-0...生产者...烤鸭3
Thread-3...消费者...烤鸭3
Thread-1...生产者...烤鸭3
Thread-1...生产者...烤鸭4

利用数组解决生产者-消费者问题。

class BoundedBuffer {

import java.util.concurrent.locks.Condition;

final Lock lock = new ReentrantLock();

Object x = items[takeptr];

-- count;

return x; } finally {

} }

怎么控制线程的任务结束呢?

画 管理员: C:\Windows\system32\cmd.exe

D:\code\day14>java StopThreadDemo

■ 管理员: C:\Windows\system32\cmd.exe

Thread-0... Thread-1... main...2 Thread-1... Thread-0... Thread-1... main...3 Thread-1...

Thread-0... Thread-1... main...4 Thread-1... Thread-0...

main...42 main...43 main...44 main...45 nain...46 main...47 main...48 Thread-1... Thread-0...

D:\code\day14>javac StopThreadDemo.java

复制代码

5.3.3 停止线程

notFull.signal();

lock.unlock();

任务中都会有循环结构,只要控制住循环就可以结束任务。

if (++takeptr == items.length) takeptr = 0;

import java.util.concurrent.locks.ReentrantLock;

final Condition notFull = lock .newCondition();

import java.util.concurrent.locks.Lock;

Thread-0...生产 Thread-3...消费 Thread-1...生产 Thread-2...消费

Thread-3... 生产 Thread-3...消费 Thread-1...生产

Thread-2...消 Thread-0...生

Thread-2...消 Thread-0...生 Thread-3...消 Thread-1.. Thread-2..

代码:

01.

03.

04.

06.

07.

32.

34.

35. 36.

37.

38. 39.

40.

D:∖code∖day14>javac ProducerConsumerDemo.java

D:\code\day14>java ProducerConsumerDemo

93.

```
控制循环通常就用定义标记来完成。
示例:
     class StopThread implements Runnable{
            private boolean flag = true;
02.
             public void run(){
                  while(flag ){
04.
                       System. out.println(Thread.currentThread().getName() +
05.
     "...");
06.
                  }
08.
            public void setFlag(){
                  flag = false ;
09.
    }
11.
12.
    class StopThreadDemo{
13.
14.
            public static void main(String[] args){
15.
                  StopThread st = new StopThread();
16.
17.
                  Thread t1 = new Thread(st);
                  Thread t2 = new Thread(st);
18.
19.
20.
                 t1.start();
21.
                 t2.start();
22.
                   int num = 1;
23.
                   for(;;){
24.
                         if(++num == 50){
25.
26.
                              st.setFlag();
                              break;
27.
                        }
29.
                        System. out.println("main..." + num);
30.
                   System. out.println("over" );
31.
32.
            }
33.
     复制代码
运行结果:
                                                                      _ D X
```

Ξ

\_ \_ \_ X

```
Thread-1...
 main...49
 Thread-1...
 Thread-0...
 over
  P.S.
  也可以使用stop方法停止线程,不过已经过时,不再使用。
  但是如果线程处于了冻结状态,无法读取标记,如何结束呢?
  可以使用interrupt()方法将线程从冻结状态强制恢复到运行状态中来,让线程具备CPU的执行资格。
强制动作会发生InterruptedException,一定要记得处理。
  API文档叙述如下:
interrupt
public woid interrupt()
   中断线程。
   如果当前线程没有中断它自己(这在任何情况下都是允许的),则该线程的 checkAccess 方法就会被调用,这可能抛出 SecurityException。
   如果线程在调用 Object 类的 wait()、wait(long) 或 wait(long, int) 方法,或者该类的 join()、join(long)、join(long, int)、sleep(long) 或 sleep
(long, int) 方法过程中受阻,则其中断状态将被清除,它还将收到一个 InterruptedException。
   如果该线程在<u>可中断的通道上的 I/O 操作中受阻,则该通道将被关闭,该线程的中断状态将被设置并且该线程将收到</u>一个
  示例:
        class StopThread implements Runnable{
 02.
               private boolean flag = true;
               public synchronized void run(){
 03.
                     while(flag){
 04.
                           try{
 05.
                                 wait();
                          } catch(InterruptedException e){
 07.
                                 System.out.println(Thread.currentThread().getName() +
 08.
        "..." + e);
                                 flag = false;
 09.
 10.
                          System.out.println(Thread.currentThread().getName() +
 11.
        ".....");
 12.
              }
 13.
               public void setFlag(){
 14.
                    flag = false;
 15.
              }
 16.
 17.
        }
 18.
       class StopThreadDemo{
 19.
```

public static void main(String[] args){

Thread t1 = new Thread(st); Thread t2 = new Thread(st);

 $if(++num == 50){$ 

break;

System.out.println( "over");

t1.interrupt();

t2.interrupt();

System.out.println( "main..." + num);

\_ \_ \_ X

\_ \_ \_ X

Ξ

\_ \_ \_ X

t1.start();

t2.start();

for(;;){

int num = 1;

StopThread st = new StopThread();

20.

21. 22. 23.

24. 25.

26. 27.

28.

29.

30.

31.

32.

33.

34. 35.

36. 37. 38.

39. 40.

}

运行结果:

main...2 main...3 main...4 main...5 main...6 main...7 main...8 main...9 main...10 main...11 nain...12 main...13 main...14 main...15

main...43 main...44 main...45 main...46 main...47 main...48 main...49

复制代码

}

画 管理员: C:∖Windows\system32\cmd.exe

D:\code\day14>java StopThreadDemo

📷 管理员: C:\Windows\system32\cmd.exe

D:\code\day14>javac StopThreadDemo.java

```
over
 Thread-0...java.lang.InterruptedException
 Thread-0.....
 Thread-1...java.lang.InterruptedException
 Thread-1.....
  5.3.4 线程类的其他方法
  setDaemon方法:
public final void setDaemon(boolean on)
    将该线程标记为守护线程或用户线程。当正在运行的线程都是守护线程时,Java 虚拟机退出。
    该方法必须在启动线程前调用。
    该方法首先调用该线程的 checkAccess 方法,且不带任何参数。这可能抛出 SecurityException(在当前线程中)。
  示例:
       class StopThread implements Runnable{
              private boolean flag = true;
 02.
              public synchronid void run(){
 03.
                    while(flag ){
 05.
                          try{
 06.
                               wait();
                         } catch(InterruptedException e){
 07.
                               System. out.println(Thread.currentThread().getName()
 08.
       + "..." + e);
                                flag = false ;
 09.
 10.
                         System. out.println(Thread.currentThread().getName() +
 11.
       ".....");
 12.
 13.
             }
              public void setFlag(){
 14.
                   flag = false ;
 15.
 16.
             }
 17.
       }
 18.
 19.
       class StopThreadDemo{
              public static void main(String[] args){
  21.
                   StopIhread st = new StopIhread();
 22.
 23.
                   Thread t1 = new Thread(st);
                   Thread t2 = new Thread(st);
 24.
 25.
 26.
                   t1.start();
                   t2.setDaemon( true);
 27.
 28.
                   t2.start();
 29.
                    int num = 1;
 30.
                    for(;;){
 31.
                          if(++num == 50){
 32.
                               t1.interrupt();
 33.
                                break;
 34.
                         }
 35.
 36.
                         System. out.println("main..." + num);
 37.
 38.
                   System. out.println("over" );
 39.
             }
 40.
       }
       复制代码
 运行结果:
                                                                        _ D X
 画 管理员: C:\Windows\system32\cmd.exe
 D:\code\day14>javac StopThreadDemo.java
 D:∖code∖day14>java StopThreadDemo
 main...2
 main...3
 main...4
 main...5
  main...6
  main...7
  nain...8
 main...9
  main...10
  main...11
  main...12
 main...13
  main...14
 main...15
```

```
Thread-0.....
 join方法:
join
public final void join()
           throws <u>InterruptedException</u>
  等待该线程终止。
    抛出:
        InterruptedException - 如果任何线程中断了当前线程。当抛出该异常时,当前线程的中断状态被清除。
 示例:
 01.
       class Demo implements Runnable{
             public void run(){
                   for(int x = 0; x < 50; x++){
 03.
 04.
                        System. out.println(Thread.currentThread().getName() +
       "..." + x);
 05.
                  }
 06.
            }
 07.
 08.
     class JoinDemo{
 09.
 10.
             public static void main(String[] args){
 11.
                  Demo d = new Demo();
 12.
 13.
                  Thread t1 = new Thread(d);
 14.
                  Thread t2 = new Thread(d);
 15.
 16.
                  t1.start();
 17.
                   try {
                        t1.join(); //t1线程要申请加入进来,运行。然后,主线程等待t1执行完
 18.
       毕。
                         //临时加入一个线程运算时可以使用join方法。
 19.
                   } catch (InterruptedException e) {
 20.
                        e.printStackTrace();
 22.
                   }
 23.
                  t2.start();
 24.
 25.
                   for(int x = 0; x < 50; x++){
                        System.out.println(Thread.currentThread().toString() +
 26.
       "..." + x);
 27.
 28.
            }
 29.
       复制代码
 运行结果:
                                                                     _ 0 X
 📷 管理员: C:\Windows\system32\cmd.exe
```

画 管理员: C:\Windows\system32\cmd.exe

Thread-0...java.lang.InterruptedException

main...43 main...44 main...45 main...46 main...47 main...48 main...49 over

Thread-0...9 Thread-0...10 Thread-0...11 Thread-0...12 \_ \_ \_ X 画 管理员: C:\Windows\system32\cmd.exe Thread[main,5,main]...44 Thread-1...42 Thread[main,5,main]...45 Thread-1...43 Thread[main,5,main]...46 Thread-1...44 Thread[main,5,main]...47 [hread-1...45 Thread[main,5,main]...48 Thread-1...46 Thread[main,5,main]...49 Thread-1...47 Thread-1...48 Thread-1...49 setPriority方法:

D:∖code∖day14>javac JoinDemo.java

D:\code\day14>java JoinDemo

Thread-0...0 Thread-0...1 Thread-0...2 Thread-0...3 Thread-0...4 Thread-0...5 Thread-0...6

Thread-0...7 Thread-0...8

setPriority

public final void setPriority(int newPriority)

```
更改线程的优先级。
  首先调用线程的 checkAccess 方法,且不带任何参数。这可能抛出 SecurityException。
  在其他情况下,线程优先级被设定为指定的 newPriority 和该线程的线程组的最大允许优先级相比较小的一个。
     newPriority - 要为线程设定的优先级
toString方法:
```

