最全面的Java多线程用法解析

晴明 · 21 天前

最全面的java多线程用法解析，如果你对Java的多线程机制并没有深入的研究，那么本文可以帮助你更透彻地理解Java多线程的原理以及使用方法。

1.创建线程

在Java中创建线程有两种方法：使用Thread类和使用Runnable接口。在使用Runnable接口时需要建立一个Thread实例。因此，无论是通过Thread类还是Runnable接口建立线程，都必须建立Thread类或它的子类的实例。 Thread构造函数：

- public Thread( );
- public Thread(Runnable target);
- public Thread(String name);
- public Thread(Runnable target, String name);
- public Thread(ThreadGroup group, Runnable target);
- public Thread(ThreadGroup group, String name);
- public Thread(ThreadGroup group, Runnable target, String name);
- public Thread(ThreadGroup group, Runnable target, String name, long stackSize);

方法一：继承Thread类覆盖run方法

```
public class ThreadDemo1 {
public static void main(String[] args){
        Demo d = new Demo();
        d.start();
for(int i=0;i<60;i++){
            System.out.println(Thread.currentThread().getName()+i);
        }

    }
 }
class Demo extends Thread{
public void run(){
for(int i=0;i<60;i++){
            System.out.println(Thread.currentThread().getName()+i);
        }
    }
 }
```

方法二：

```
public class ThreadDemo2 {
public static void main(String[] args){
       Demo2 d =new Demo2();
       Thread t = new Thread(d);
       t.start();
for(int x=0;x<60;x++){
           System.out.println(Thread.currentThread().getName()+x);
```

```
        }
    }
}
class Demo2 implements Runnable{
public void run(){
for(int x=0;x<60;x++){
            System.out.println(Thread.currentThread().getName()+x);
        }
    }
}
```

2.线程的生命周期

与人有生老病死一样，线程也同样要经历开始（等待）、运行、挂起和停止四种不同的状态。这四种状态都可以通过Thread类中的方法进行控制。下面给出了Thread类中和这四种状态相关的方法。

- // 开始线程
- publicvoid start( );
- publicvoid run( );
- // 挂起和唤醒线程
- publicvoid resume( ); // 不建议使用
- publicvoid suspend( ); // 不建议使用
- publicstaticvoid sleep(long millis);
- publicstaticvoid sleep(long millis, int nanos);
- // 终止线程
- publicvoid stop( ); // 不建议使用
- publicvoid interrupt( );
- // 得到线程状态
- publicboolean isAlive( );
- publicboolean isInterrupted( );
- publicstaticboolean interrupted( );

- // join方法
- publicvoid join( ) throws InterruptedException;

线程在建立后并不马上执行run方法中的代码，而是处于等待状态。线程处于等待状态时，可以通过Thread类的方法来设置线程不各种属性，如线程的优先级（setPriority）、线程名(setName)和线程的类型（setDaemon）等。

当调用start方法后，线程开始执行run方法中的代码。线程进入运行状态。可以通过Thread类的isAlive方法来判断线程是否处于运行状态。当线程处于运行状态时，isAlive返回true，当isAlive返回false时，可能线程处于等待状态，也可能处于停止状态。下面的代码演示了线程的创建、运行和停止三个状态之间的切换，并输出了相应的isAlive返回值。

一但线程开始执行run方法，就会一直到这个run方法执行完成这个线程才退出。但在线程执行的过程中，可以通过两个方法使线程暂时停止执行。这两个方法是suspend和sleep。在使用suspend挂起线程后，可以通过resume方法唤醒线程。而使用sleep使线程休眠后，只能在设定的时间后使线程处于就绪状态（在线程休眠结束后，线程不一定会马上执行，只是进入了就绪状态，等待着系统进行调度）。

在使用sleep方法时有两点需要注意：

1. sleep方法有两个重载形式，其中一个重载形式不仅可以设毫秒，而且还可以设纳秒(1,000,000纳秒等于1毫秒)。但大多数操作系统平台上的Java虚拟机都无法精确到纳秒，因此，如果对sleep设置了纳秒，Java虚拟机将取最接近这个值的毫秒。

2. 在使用sleep方法时必须使用throws或try{…}catch{…}。因为run方法无法使用throws，所以只能使用try{…}catch{…}。当在线程休眠的过程中，使用interrupt方法中断线程时sleep会抛出一个InterruptedException异常。sleep方法的定义如下：

1. **publicstaticvoid** sleep(**long** millis) **throws** InterruptedException
2. **publicstaticvoid** sleep(**long** millis, **int** nanos) **throws** InterruptedException

有三种方法可以使终止线程。

1. 使用退出标志，使线程正常退出，也就是当run方法完成后线程终止。

2. 使用stop方法强行终止线程（这个方法不推荐使用，因为stop和suspend、resume一样，也可能发生不可预料的结果）。

3. 使用interrupt方法中断线程。

1. 使用退出标志终止线程

当run方法执行完后，线程就会退出。但有时run方法是永远不会结束的。如在服务端程序中使用线程进行监听客户端请求，或是其他的需要循环处理的任务。在这种情况下，一般是将这些任务放在一个循环中，如while循环。如果想让循环永远运行下去，可以使用while(true){…}来处理。但要想使while循环在某一特定条件下退出，最直接的方法就是设一个boolean类型的标志，并通过设置这个标志为true或false来控制while循环是否退出。下面给出了一个利用退出标志终止线程的例子。

join方法的功能就是使异步执行的线程变成同步执行。也就是说，当调用线程实例的start方法后，这个方法会立即返回，如果在调用start方法后后需要使用一个由这个线程计算得到的值，就必须使用join方法。如果不使用join方法，就不能保证当执行到start方法后面的某条语句时，这个线程一定会执行完。而使用join方法后，直到这个线程退出，程序才会往下执行。下面的代码演示了join的用法。

3.多线程安全问题

问题原因：当多条语句在操作同一个线程共享数据时，一个线程对多条语句只执行了一部分，还没执行完，另一个线程参与进来执行，导致共享数据的错误。

解决办法：对多条操作共享数据的语句，只能让一个线程都执行完，在执行过程中，其他线程不执行。

同步代码块：

```
public class ThreadDemo3 {
public static void main(String[] args){
        Ticket t =new Ticket();
        Thread t1 = new Thread(t,"窗口一");
        Thread t2 = new Thread(t,"窗口二");
        Thread t3 = new Thread(t,"窗口三");
        Thread t4 = new Thread(t,"窗口四");
        t1.start();
        t2.start();
        t3.start();
        t4.start();
    }
```

```
}
class Ticket implements Runnable{
private int ticket =400;
public void run(){
while(true){
synchronized (new Object()) {
try {
                    Thread.sleep(1);
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
if(ticket<=0)
break;
                System.out.println(Thread.currentThread().getName()+"---卖出"+ticket--);
            }
        }
    }
}
```

同步函数

```
public class ThreadDemo3 {
public static void main(String[] args){
        Ticket t =new Ticket();
        Thread t1 = new Thread(t,"窗口一");
        Thread t2 = new Thread(t,"窗口二");
        Thread t3 = new Thread(t,"窗口三");
        Thread t4 = new Thread(t,"窗口四");
        t1.start();
        t2.start();
        t3.start();
        t4.start();
    }
}
class Ticket implements Runnable{
private int ticket = 4000;
public synchronized void  saleTicket(){
if(ticket>0)
            System.out.println(Thread.currentThread().getName()+"卖出了"+ticket--);

    }
```

```
public void run(){
while(true){
            saleTicket();
        }
    }
}
```

同步函数锁是this 静态同步函数锁是class

线程间的通信

```
public class ThreadDemo3 {
public static void main(String[] args){
class Person{
public String name;
private String gender;
public void set(String name,String gender){
this.name =name;
this.gender =gender;
            }
public void get(){
                System.out.println(this.name+"...."+this.gender);
            }
        }
        final Person p =new Person();
new Thread(new Runnable(){
public void run(){
int x=0;
while(true){
if(x==0){
                        p.set("张三", "男");
                    }else{
                        p.set("lili", "nv");
                    }
                    x=(x+1)%2;
                }
            }
        }).start();
new Thread(new Runnable(){
public void run(){
while(true){
                    p.get();
```

```
                }
            }
        }).start();
    }
}
/*
张三....男
张三....男
lili....nv
lili....男
张三....nv
lili....男
*/
```

修改上面代码

```
public class ThreadDemo3 {
public static void main(String[] args){
class Person{
public String name;
private String gender;
public void set(String name,String gender){
this.name =name;
this.gender =gender;
            }
public void get(){
                System.out.println(this.name+"...."+this.gender);
            }
        }
        final Person p =new Person();
new Thread(new Runnable(){
public void run(){
int x=0;
while(true){
                    synchronized (p) {
if(x==0){
                            p.set("张三", "男");
                        }else{
                            p.set("lili", "nv");
                        }
                        x=(x+1)%2;
                    }
```

```
                }
            }
        }).start();
new Thread(new Runnable(){
public void run(){
while(true){
                    synchronized (p) {
                        p.get();
                    }
                }
            }
        }).start();
    }

}
/*
lili....nv
lili....nv
lili....nv
lili....nv
lili....nv
lili....nv
张三....男
张三....男
张三....男
张三....男
*/
```

等待唤醒机制

```
/*
 *线程等待唤醒机制
 *等待和唤醒必须是同一把锁
 */
public class ThreadDemo3 {
private static boolean flags =false;
public static void main(String[] args){
class Person{
public String name;
private String gender;
public void set(String name,String gender){
```

```
this.name =name;
this.gender =gender;
            }
public void get(){
                System.out.println(this.name+"...."+this.gender);
            }
        }
final Person p =new Person();
new Thread(new Runnable(){
public void run(){
int x=0;
while(true){
synchronized (p) {
if(flags)
try {
                                p.wait();
                            } catch (InterruptedException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                            };
if(x==0){
                                p.set("张三", "男");
                        }else{
                                p.set("lili", "nv");
                        }
                        x=(x+1)%2;
                        flags =true;
                        p.notifyAll();
                    }
                }
            }
        }).start();
new Thread(new Runnable(){
public void run(){
while(true){
synchronized (p) {
if(!flags)
try {
                                p.wait();
                            } catch (InterruptedException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
```

```
                        };
                    p.get();
                    flags =false;
                    p.notifyAll();
                    }
                }
            }
        }).start();
    }
}
```

生产消费机制一

```
public class ThreadDemo4 {
private static boolean flags =false;
public static void main(String[] args){
class Goods{
private String name;
private int num;
public synchronized void produce(String name){
if(flags)
try {
                    wait();
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
this.name =name+"编号: "+num++;
                System.out.println("生产了...."+this.name);
                flags =true;
                notifyAll();
            }
public synchronized void consume(){
if(!flags)
try {
                    wait();
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
                System.out.println("消费了******"+name);
                flags =false;
```

```
                notifyAll();
            }

        }
final Goods g =new Goods();
new Thread(new Runnable(){
public void run(){
while(true){
                    g.produce("商品");
                }
            }
        }).start();
new Thread(new Runnable(){
public void run(){
while(true){
                    g.consume();
                }
            }
        }).start();
    }
}
```

生产消费机制2

```
public class ThreadDemo4 {
private static boolean flags =false;
public static void main(String[] args){
class Goods{
private String name;
private int num;
public synchronized void produce(String name){
while(flags)
try {
                    wait();
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
this.name =name+"编号："+num++;
                System.out.println(Thread.currentThread().getName()+"生产了...."+this.na
                flags =true;
                notifyAll();
```

```
            }
public synchronized void consume(){
while(!flags)
try {
                        wait();
                    } catch (InterruptedException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                    }
                System.out.println(Thread.currentThread().getName()+"消费了******"+name)
                flags =false;
                notifyAll();
            }

        }
final Goods g =new Goods();
new Thread(new Runnable(){
public void run(){
while(true){
                    g.produce("商品");
                }
            }
        },"生产者一号").start();
new Thread(new Runnable(){
public void run(){
while(true){
                    g.produce("商品");
                }
            }
        },"生产者二号").start();
new Thread(new Runnable(){
public void run(){
while(true){
                    g.consume();
                }
            }
        },"消费者一号").start();
new Thread(new Runnable(){
public void run(){
while(true){
                    g.consume();
                }
```

```
            }
        },"消费者二号").start();
    }
}
/*
消费者二号消费了******商品编号：48049
生产者一号生产了....商品编号：48050
消费者一号消费了******商品编号：48050
生产者一号生产了....商品编号：48051
消费者二号消费了******商品编号：48051
生产者二号生产了....商品编号：48052
消费者二号消费了******商品编号：48052
生产者一号生产了....商品编号：48053
消费者一号消费了******商品编号：48053
生产者一号生产了....商品编号：48054
消费者二号消费了******商品编号：48054
生产者二号生产了....商品编号：48055
消费者二号消费了******商品编号：48055
*/
```

Java　Java 编程　程序员