目录

[线程 2](#_Toc516677322)

[synchronized关键字 2](#_Toc516677323)

[MultiThread 3](#_Toc516677324)

[锁的写法 4](#_Toc516677325)

[脏读 6](#_Toc516677326)

[Volatile 7](#_Toc516677327)

[线程异常 8](#_Toc516677328)

[Atomic原子类 9](#_Toc516677329)

[Notify和wait 10](#_Toc516677330)

[countDownLatch 11](#_Toc516677331)

[线程池 13](#_Toc516677332)

[自定义线程池 13](#_Toc516677333)

## 线程

### synchronized关键字

public class MyThread **extends** Thread **{**

private int count **=** 5**;**

//synchronized关键字

@Override

public synchronized void run**()** **{**

count**--;**

System**.**out**.**println**(this.**currentThread**().**getName**()** **+** " count =" **+** count**);**

**}**

public static void main**(**String**[]** args**)** **{**

引用同一个对象myThread,如果不加synchronized关键字，则每个线程都能count—操作。

MyThread myThread **=** **new** MyThread**();**

Thread t1 **=** **new** Thread**(**myThread**,** "t1"**);**

Thread t2 **=** **new** Thread**(**myThread**,** "t2"**);**

Thread t3 **=** **new** Thread**(**myThread**,** "t3"**);**

Thread t4 **=** **new** Thread**(**myThread**,** "t4"**);**

Thread t5 **=** **new** Thread**(**myThread**,** "t5"**);**

t1**.**start**();**

t2**.**start**();**

t3**.**start**();**

t4**.**start**();**

t5**.**start**();**

**}**

**}**

### MultiThread

public class MultiThread **{**

private int num **=** 0**;**

public synchronized void printNum**(**String tag**){**

方法不加static表示对象锁，加了static表示类锁

**try** **{**

**if(**tag**.**equals**(**"a"**)){**

num **=** 100**;**

System**.**out**.**println**(**"tag a, set num over!"**);**

Thread**.**sleep**(**1000**);**

**}** **else** **{**

num **=** 200**;**

System**.**out**.**println**(**"tag b, set num over!"**);**

**}**

System**.**out**.**println**(**"tag " **+** tag **+** ", num = " **+** num**);**

**}** **catch** **(**InterruptedException e**)** **{**

e**.**printStackTrace**();**

**}**

**}**

public static void main**(**String**[]** args**)** **{**

final MultiThread m1 **=** **new** MultiThread**();**

final MultiThread m2 **=** **new** MultiThread**();**

Thread t1 **=** **new** Thread**(new** Runnable**()** **{**

public void run**()** **{**

m1**.**printNum**(**"a"**);**

**}**

T1线程调用m1对象

T2线程调用m2对象

**});**

Thread t2 **=** **new** Thread**(new** Runnable**()** **{**

public void run**()** **{**

m2**.**printNum**(**"b"**);**

**}**

**});**

t1**.**start**();**

t2**.**start**();**

**}**

**}**

### 锁的写法

public class ObjectLock **{**

public void method1**(){**

synchronized **(this)** **{** //对象锁

**try** **{**

System**.**out**.**println**(**"do method1.."**);**

Thread**.**sleep**(**2000**);**

**}** **catch** **(**InterruptedException e**)** **{**

e**.**printStackTrace**();**

**}**

**}**

**}**

public void method2**(){** //类锁.class

synchronized **(**ObjectLock**.**class**)** **{**

**try** **{**

System**.**out**.**println**(**"do method2.."**);**

Thread**.**sleep**(**2000**);**

**}** **catch** **(**InterruptedException e**)** **{**

e**.**printStackTrace**();**

**}**

**}**

**}**

private Object lock **=** **new** Object**();**

public void method3**(){** //任何对象锁 Obejct lock

synchronized **(**lock**)** **{**

**try** **{**

System**.**out**.**println**(**"do method3.."**);**

Thread**.**sleep**(**2000**);**

**}** **catch** **(**InterruptedException e**)** **{**

e**.**printStackTrace**();**

**}**

**}**

**}**

public static void main**(**String**[]** args**)** **{**

final ObjectLock objLock **=** **new** ObjectLock**();**

Thread t1 **=** **new** Thread**(new** Runnable**()** **{**

@Override

public void run**()** **{**

objLock**.**method1**();**

**}**

**});**

Thread t2 **=** **new** Thread**(new** Runnable**()** **{**

@Override

public void run**()** **{**

objLock**.**method2**();**

**}**

**});**

Thread t3 **=** **new** Thread**(new** Runnable**()** **{**

@Override

public void run**()** **{**

objLock**.**method3**();**

**}**

**});**

t1**.**start**();**

t2**.**start**();**

t3**.**start**();**

**}**

**}**

### 脏读

/\*\*

\* 脏读

\* 有两个线程 主线程和t1线程

\* 一致性读的概念

\* 如果读数据需要到9点10读完，9点05分改数据了，但是不影响9点10分查询的结果

\*

\*

\*/

public class DirtyRead **{**

private String username **=** "bjsxt"**;**

private String password **=** "123"**;**

public void setValue**(**String username**,** String password**){**

**this.**username **=** username**;**

**try** **{**

Thread**.**sleep**(**2000**);**

**}** **catch** **(**InterruptedException e**)** **{**

e**.**printStackTrace**();**

**}**

**this.**password **=** password**;**

System**.**out**.**println**(**"setValue最终结果：username = " **+** username **+** " , password = " **+** password**);**

**}**

public void getValue**(){**

System**.**out**.**println**(**"getValue方法得到：username = " **+** **this.**username **+** " , password = " **+** **this.**password**);**

**}**

public static void main**(**String**[]** args**)** **throws** Exception**{**

final DirtyRead dr **=** **new** DirtyRead**();**

Thread t1 **=** **new** Thread**(new** Runnable**()** **{**

@Override

public void run**()** **{**

dr**.**setValue**(**"z3"**,** "456"**);**

**}**

**});**

t1**.**start**();**//线程1

Thread**.**sleep**(**1000**);**

dr**.**getValue**();** //主线程

**}**

**}**

### Volatile

/\*\*

thread每一个线程都有独立空间，装主内存的引用变量，如果没有加volatile就是copy，加volatile就是引用copy

\*\*/

public class RunThread **extends** Thread**{**

private volatile boolean isRunning **=** **true;**

// private boolean isRunning = true;

private void setRunning**(**boolean isRunning**){**

**this.**isRunning **=** isRunning**;**

**}**

public void run**(){**

System**.**out**.**println**(**"进入run方法.."**);**

int i **=** 0**;**

**while(**isRunning **==** **true){**

//..

**}**

System**.**out**.**println**(**"线程停止"**);**

**}**

public static void main**(**String**[]** args**)** **throws** InterruptedException **{**

RunThread rt **=** **new** RunThread**();**

rt**.**start**();**

Thread**.**sleep**(**1000**);**

rt**.**setRunning**(false);**

System**.**out**.**println**(**"isRunning的值已经被设置了false"**);**

System**.**out**.**println**(**rt**.**isRunning**);**

**}**

**}**

### 线程异常

public class SyncException **{**

private int i **=** 0**;**

public synchronized void operation**(){**

**while(true){**

**try** **{**

i**++;**

Thread**.**sleep**(**100**);**

System**.**out**.**println**(**Thread**.**currentThread**().**getName**()** **+** " , i = " **+** i**);**

Thread**.**sleep**(**500**);**

**if(**i **==** 10**){**

Integer**.**parseInt**(**"a"**);**

**}**

**}** **catch** **(**Exception e**)** **{**

**throw** **new** RuntimeException**();**

**}**

**}**

**}**

public static void main**(**String**[]** args**)** **{**

final SyncException se **=** **new** SyncException**();**

Thread t1 **=** **new** Thread**(new** Runnable**()** **{**

@Override

public void run**()** **{**

se**.**operation**();**

**}**

**},**"t1"**);**

t1**.**start**();**

**}**

**}**

### Atomic原子类

/\*\*

\* 原子性 要么完整的被执行,要么完全不执行

\* 1.volatile关键字不具备synchronized关键字的原子性（同步）

\* 2.automic具备

\* 多线程并发原子操作，操作一个对象

\*/

public class VolatileNoAtomic **extends** Thread**{**

// private static volatile int count = 0;

private static AtomicInteger count **=** **new** AtomicInteger**(**0**);**//原子类

private static void addCount**(){**

**for** **(**int i **=** 0**;** i **<** 1000**;** i**++)** **{**

// count++ ; //加1000次

count**.**incrementAndGet**();**//只能保证一个方法的原子性,多个的话需要方法加sync关键字

**}**

System**.**out**.**println**(**count**);**

**}**

public void run**(){**

addCount**();**

**}**

public static void main**(**String**[]** args**)** **{**

VolatileNoAtomic**[]** arr **=** **new** VolatileNoAtomic**[**100**];**

**for** **(**int i **=** 0**;** i **<** 10**;** i**++)** **{**

arr**[**i**]** **=** **new** VolatileNoAtomic**();** //10个线程

**}**

**for** **(**int i **=** 0**;** i **<** 10**;** i**++)** **{**

arr**[**i**].**start**();** //线程启动

**}**

**}**

**}**

### Notify和wait

Notify不释放锁

public class ListAdd2 **{**

private volatile static List list **=** **new** ArrayList**();**

public void add**(){**

list**.**add**(**"bjsxt"**);**

**}**

public int size**(){**

**return** list**.**size**();**

**}**

public static void main**(**String**[]** args**)** **{**

final ListAdd2 list2 **=** **new** ListAdd2**();**

final Object lock **=** **new** Object**();**

Thread t1 **=** **new** Thread**(new** Runnable**()** **{**

@Override

public void run**()** **{**

**try** **{**

synchronized **(**lock**)** **{** //获得锁

System**.**out**.**println**(**"t1启动.."**);**

**for(**int i **=** 0**;** i **<**10**;** i**++){**

list2**.**add**();**

System**.**out**.**println**(**"当前线程：" **+** Thread**.**currentThread**().**getName**()** **+** "添加了一个元素.."**);**

Thread**.**sleep**(**500**);**

**if(**list2**.**size**()** **==** 5**){**

System**.**out**.**println**(**"已经发出通知.."**);**

lock**.**notify**();** //发出通知 ，不释放锁 ，执行完毕才会释放

**}**

**}**

**}**

**}** **catch** **(**InterruptedException e**)** **{**

e**.**printStackTrace**();**

**}**

**}**

**},** "t1"**);**

Thread t2 **=** **new** Thread**(new** Runnable**()** **{**

@Override

public void run**()** **{**

synchronized **(**lock**)** **{** //同一个锁对象

System**.**out**.**println**(**"t2启动.."**);**

//不等于5就等待

**if(**list2**.**size**()** **!=** 5**){**

**try** **{**

lock**.**wait**();**//释放锁 阻塞t1先执行没人唤醒了

**}** **catch** **(**InterruptedException e**)** **{**

e**.**printStackTrace**();**

**}**

**}**

System**.**out**.**println**(**"当前线程：" **+** Thread**.**currentThread**().**getName**()** **+** "收到通知线程停止.."**);**

**throw** **new** RuntimeException**();**

**}**

**}**

**},** "t2"**);**

t2**.**start**();**

t1**.**start**();**

**}**

**}**

### countDownLatch

public class ListAdd3 **{**

private volatile static List list **=** **new** ArrayList**();**

public void add**(){**

list**.**add**(**"bjsxt"**);**

**}**

public int size**(){**

**return** list**.**size**();**

**}**

public static void main**(**String**[]** args**)** **{**

final ListAdd3 list3 **=** **new** ListAdd3**();**

// final Object lock = new Object();

//实时通知

final CountDownLatch countDownLatch **=** **new** CountDownLatch**(**1**);**//1发起一次 countDownLatch.countDown();通知

Thread t1 **=** **new** Thread**(new** Runnable**()** **{**

@Override

public void run**()** **{**

**try** **{**

// synchronized (lock) { //获得锁

System**.**out**.**println**(**"t1启动.."**);**

**for(**int i **=** 0**;** i **<**10**;** i**++){**

list3**.**add**();**

System**.**out**.**println**(**"当前线程：" **+** Thread**.**currentThread**().**getName**()** **+** "添加了一个元素.."**);**

Thread**.**sleep**(**500**);**

**if(**list3**.**size**()** **==** 5**){**

System**.**out**.**println**(**"已经发出通知.."**);**

// lock.notify(); //发出通知 ，不释放锁 ，执行完毕才会释放

countDownLatch**.**countDown**();**

**}**

**}**

// }

**}** **catch** **(**InterruptedException e**)** **{**

e**.**printStackTrace**();**

**}**

**}**

**},** "t1"**);**

Thread t2 **=** **new** Thread**(new** Runnable**()** **{**

@Override

public void run**()** **{**

// synchronized (lock) { //同一个锁对象

System**.**out**.**println**(**"t2启动.."**);**

//不等于5就等待

**if(**list3**.**size**()** **!=** 5**){**

**try** **{**

// lock.wait();//释放锁 阻塞t1先执行没人唤醒了

countDownLatch**.**await**();**

**}** **catch** **(**InterruptedException e**)** **{**

e**.**printStackTrace**();**

**}**

**}**

System**.**out**.**println**(**"当前线程：" **+** Thread**.**currentThread**().**getName**()** **+** "收到通知线程停止.."**);**

**throw** **new** RuntimeException**();**

**}**

// }

**},** "t2"**);**

t1**.**start**();**

t2**.**start**();**

**}**

**}**

### 线程池

newFixedTheadPool （固定数量 空闲执行）

newSingleThreadExecutor（一个线程池，空闲执行 没有缓存队列）

newCachedThreadPool （可调节线程池 ，空线程60秒回收）

newScheduledThreadPool （ScheduedExecutorService对象 ，返回指定线程数量）

### 自定义线程池

#### ArrayBlockingQueue 有界队列

/\*\*

\* 在使用有界队列时，若有新的任务需要执行，如果线程池实际线程数小于corePoolSize，则优先创建线程，

\* 若大于corePoolSize，则会将任务加入队列，等待第一个任务完成后才执行

\* 若队列已满，则在总线程数不大于maximumPoolSize的前提下，创建新的线程，

\* 若线程数大于maximumPoolSize，则执行拒绝策略。或其他自定义方式。

\*/

ThreadPoolExecutor pool **=** **new** ThreadPoolExecutor**(** //这时间是没有线程的，自定义线程

1**,** //coreSize 首先会加载的线程

2**,** //MaxSize 最大装载的线程

60**,** //60

TimeUnit**.**SECONDS**,**

**new** ArrayBlockingQueue**<**Runnable**>(**3**)** //指定一种队列 （有界队列）容量是3 除了coreSize运行的任务，其他任务会分配到队列中

//new LinkedBlockingQueue<Runnable>()

**,** **new** MyRejected**()**

// , new DiscardOldestPolicy() //丢弃最老的任务 拒绝策略

**);**



#### LinkedBlockingQueue 无界队列

无界队列不会拒绝，任务队列不会耗尽 ，达到core 不会增加，放在队列等待

