



个人资料



happy\_8

原创  
21

粉丝  
4

等级：博客 已

积分：420

排名：11万+

勋章：恒

关于 招聘 广告服务

度

©1999-2018 CSDN版权所有  
京ICP证09002463号

经营性网站备案信息

网络110报警服务

中国互联网举报中心

北京互联网违法和不良信息举报中心

最新文章

微分享-高并发下的缓存实战

微分享-工作中常用到的java enum实现形式

微分享-快速高效的从一个list转化到另一个list

微分享-spring线程池实战

不惑JAVA之JAVA基础 - Concurrent 概述

个人分类

Java基础 15篇

工作积累 2篇

项目管理-PMP 1篇

开发常用工具 1篇

系统运维 1篇

展开

归档

2017年8月 4篇

2016年5月 12篇

2016年4月 1篇

2015年1月 1篇

2014年12月 1篇

展开



不惑JAVA之JAVA基础 - 锁 -synchronized、Lock解析

2016年05月07日 21:54:17

阅读数：1468

锁主要解决线程安全问题。而线程安全问题，即多个线程同时访问一个资源时，会导致程序运行结果并不是想看到的结果。

synchronized



如果对临界资源加上互斥锁，当一个线程在访问该临界资源时，其他线程便只能等待。

在Java中，每一个对象都拥有一个锁标记（monitor），也称为监视器，多线程同时访问某个对象时，线程只有获取了该对象的锁才能访问。

在Java中，可以使用synchronized关键字来标记一个方法或者代码块，当某个线程调用该对象的synchronized方法或者访问synchronized代码块时，这个线程便获得了该对象的锁，其他线程暂时无法访问这个方法，只有等待这个方法执行完毕或者代码块执行完毕，这个线程才会获得该对象的锁，其他线程才能执行这个方法或者代码块。

## Synchronized使用

synchronized使用方式：

- 同步方法；
- 同步块。

**同步方法**我们以StringBuffer源码为例（StringBuffer是现成安全的）：

```
1 public synchronized StringBuffer append(String str) {
2     super.append(str);
3     return this;
4 }
```

### 同步块

```
1 synchronized(synObject) {
2     // 允许访问控制的代码
3 }
```

synchronized 块是这样一个代码块，其中的代码必须获得对象 syncObject 。由于可以针对任意代码块，且可任意指定上锁的对象，故灵活性较高。

```
1 synchronized (this) {
2     for(int i=0;i<100;i++){
3         System.out.println(thread.getName()+"在插入数据"+i);
4         arrayList.add(i);
5     }
6 }
```

### 对synchronized (this)的一些理解

1. 当两个并发线程访问同一个对象object中的这个synchronized(this)同步代码块时，一个时间内只能有一个线程得到执行。另一个线程必须等待当前线程执行完这个代码块以后才能执行该代码块。
2. 当一个线程访问object的一个synchronized(this)同步代码块时，其他线程对object中所有其它synchronized(this)同步代码块的访问将被阻塞。
3. 当一个线程访问object的一个synchronized(this)同步代码块时，另一个线程仍然可以访问该object中的除synchronized(this)同步代码块以外的部分。
4. 当一个线程访问object的一个synchronized(this)同步代码块时，它就获得了这个object的对象锁。结果，其它线程对该object对象所有同步代码部分的访问都被暂时阻塞。

**我理解的是当一个线程获取了object锁，其他线程就不能过去object锁，如果想获取只能阻塞等待，但可以访问此object中其他没有被锁定的方法。而如果是同步方法的话，一个线程执行一个对象的非static synchronized方法，另外一个线程需要执行这个对象所属类的static synchronized方法，此时不会发生互斥现象，因为访问static synchronized方法占用的是类锁，而访问非static synchronized方法占用的是对象锁，所以不存在互斥现象。** 看个例子就明白了：

**同步方法实例代码：**

```
1 public class Test {
2
3     public static void main(String[] args) {
4         final InsertData insertData = new InsertData();
5         new Thread(){
6             @Override
7             public void run() {
8                 insertData.insert();
9             }
10        }.start();
11        new Thread(){
12            @Override
13            public void run() {
14                insertData.insert1();
15            }
16        }.start();
17    }
18 }
19
20 class InsertData {
21     public synchronized void insert(){
22         System.out.println("执行insert");
23         try {
24             Thread.sleep(5000);
25         } catch (InterruptedException e) {
```

联系我们



请扫描二维码

微信扫码关注公众号

QQ客服

客服论坛

关于 招聘 广告服务

度

©1999-2018 CSDN版权所有  
京ICP证09002463号

经营性网站备案信息

网络110报警服务

中国互联网举报中心

北京互联网违法和不良信息举报中心

```
28         System.out.println("执行insert完毕");
29     }
30
31     public synchronized static void insert1() {
32         System.out.println("执行insert1");
33         System.out.println("执行insert1完毕");
34     }
35 }
```

结果是

执行insert

执行insert1

执行insert1完毕

执行insert完毕

两个方法互不影响。但要注意如果是在public synchronized void insert()添加一个static结果就不一样了，及代码

```
1  public class Test {
2
3      public static void main(String[] args) {
4          final InsertData insertData = new InsertData();
5          new Thread(){
6              @Override
7              public void run() {
8                  insertData.insert();
9              }
10         }.start();
11         new Thread(){
12             @Override
13             public void run() {
14                 insertData.insert1();
15             }
16         }.start();
17     }
18 }
19
20 class InsertData {
21 // ----- 这里添加了static修复 -----
22     public synchronized static void insert(){
23         System.out.println("执行insert");
24         try {
25             Thread.sleep(5000);
26         } catch (InterruptedException e) {
27             e.printStackTrace();
28         }
29         System.out.println("执行insert完毕");
30     }
31
32     public synchronized static void insert1() {
33         System.out.println("执行insert1");
34         System.out.println("执行insert1完毕");
35     }
36 }
```

记过就是：

执行insert

执行insert完毕

执行insert1

执行insert1完毕

也就是说：

- 某个对象实例内，synchronized aMethod(){} 可以防止多个线程同时 访问这个对象的synchronized方法，这时，不同的对象实例的 synchronized方法是不相干扰的。也就是说，其它线程照样可以同时访问相同类的另一个对象实例中的synchronized方法；
- 是某个类的范围，synchronized static aStaticMethod{} 防止多个线程同时访问这个类中的synchronized static 方法 。它可以对类的所有对象实例起作用。

来看看代码块：

```
1  public class Test1 {
2
3      public static void main(String[] args) {
4          final InsertData1 insertData = new InsertData1();
5          new Thread(){
6              @Override
7              public void run() {
8                  insertData.insert();
9              }
10         }.start();
```

二维

ister

·660·

客服  
坛

关于 招聘 广告服务  
度

©1999-2018 CSDN版权所有  
京ICP证09002463号

经营性网站备案信息

网络110报警服务

中国互联网举报中心

录举报

```
13         public void run() {
14             insertData.insert1();
15         }
16     }.start();
17 }
18 }
19
20 class InsertData1 {
21     public void insert(){
22         synchronized(this){
23             System.out.println("执行insert");
24             try {
25                 Thread.sleep(5000);
26             } catch (InterruptedException e) {
27                 e.printStackTrace();
28             }
29             System.out.println("执行insert完毕");
30         }
31     }
32
33     public void insert1() {
34         synchronized(this){
35             System.out.println("执行insert1");
36             System.out.println("执行insert1完毕");
37         }
38     }
39 }
```

执行结果：

执行insert

执行insert完毕

执行insert1

执行insert1完毕

这就证实了上面总结的。

### synchronized底层

可以大致了解一下synchronized的底层，通过反编译看一下。

```
1  public class InsertData {
2      private Object object = new Object();
3
4      public void insert(Thread thread){
5          synchronized (object) {
6
7              }
8      }
9
10     public synchronized void insert1(Thread thread){
11
12     }
13
14     public void insert2(Thread thread){
15
16     }
17 }
```

```
E:\Workspace\Test\bin\com\cxh\test1>javap -c InsertData
Compiled from "InsertData.java"
public class com.cXH.test1.InsertData extends java.lang.Object{
  public com.cXH.test1.InsertData();
    Code:
       0:   aload_0
       1:   invokespecial   #10; //Method java/lang/Object."<init>":()V
       4:   aload_0
       5:   new           #3; //class java/lang/Object
       8:   dup
       9:   invokespecial   #10; //Method java/lang/Object."<init>":()V
      12:   putfield       #12; //Field object:Ljava/lang/Object;
      15:   return

  public void insert(java.lang.Thread);
    Code:
       0:   aload_0
       1:   getfield       #12; //Field object:Ljava/lang/Object;
       4:   dup
       5:   monitorenter
       6:   monitorexit
       7:   return

  public synchronized void insert1(java.lang.Thread);
    Code:
       0:   return

  public void insert2(java.lang.Thread);
    Code:
       0:   return

}
```

从反编译获得的字节码可以看出，synchronized代码块实际上多了monitorenter和monitorexit两条指令。monitorenter指令执行时会让对象的锁计数加1，而monitorexit指令执行时会让对象的锁计数减1，其实这个与操作系统里面的PV操作很像，操作系统里面的PV操作就是用来控制多个线程对临界资源的访问。对于synchronized方法，执行中的线程识别该方法的 method\_info 结构是否有 ACC\_SYNCHRONIZED 标记设置，然后它自动获取对象的锁，调用方法，最后释放锁。如果有异常发生，线程自动释放锁。

还有两点说明：

- 1. 对于synchronized方法或者synchronized代码块，当出现异常时，JVM会自动释放当前线程占用的锁，因此不会由于异常导致出现死锁现象；
- 2. synchronized关键字是不能继承的，也就是说，基类的方法synchronized f(){} 在继承类中并不自动是synchronized f(){}，而是变成了f(){}。继承类需要你显式的指定它的某个方法为synchronized方法。

## Lock

先来说说synchronized缺陷和lock与它的不同

缺陷：

- 1. 获取锁的线程执行完了该代码块，然后线程释放对锁的占有；
- 2. 线程执行发生异常，此时JVM会让线程自动释放锁。

那么如果这个获取锁的线程由于要等待IO或者其他原因（比如调用sleep方法）被阻塞了，但是又没有释放锁，其他线程便只能干巴巴地等待，试想一下，这多么影响程序执行效率。

不同：

- 1. Lock不是Java语言内置的，synchronized是Java语言的关键字，因此是内置特性。Lock是一个类，通过这个类可以实现同步访问；
- 2. Lock和synchronized有一点非常大的不同，采用synchronized不需要用户去手动释放锁，当synchronized方法或者synchronized代码块执行完之后，系统会自动让线程释放对锁的占用；而Lock则必须要用户去手动释放锁，如果没有主动释放锁，就有可能导致出现死锁现象；
- 3. 除此之外还多了一下如下知识点：
  - 1. **等待可中断**，在持有锁的线程长时间不释放锁的时候,等待的线程可以选择放弃等待. tryLock(long timeout, TimeUnit unit)；
  - 2. **公平锁**，按照申请锁的顺序来一次获得锁称为公平锁.synchronized的是非公平锁,ReentrantLock可以通过构造函数实现公平锁. new RenentrantLock(boolean fair)；
  - 3. **绑定多个Condition**（这个在堵塞队列中用到），通过多次newCondition可以获得多个Condition对象,可以简单的实现比较复杂的线程同步的功能.通过await(),signal()。

## ReentrantLock

Lock是一个接口，这里重点介绍ReentrantLock。ReentrantLock，意思是“可重入锁”，ReentrantLock是唯一实现了Lock接口的类，并且ReentrantLock提供了更多的方法。

### 可重入锁

在学习ReentrantLock之前我们先来了解一下可重入锁。

在JAVA环境下 **ReentrantLock 和synchronized 都是 可重入锁。**

**可重入锁**，也叫做递归锁，指的是同一线程 外层函数获得锁之后，内层递归函数仍然有获取该锁的代码，但不受影响。简单的说：当一个线程请求得到一个对象锁后再次请求此对象锁,可以再次得到该对象锁,就是说在一个synchronized方法/块或ReentrantLock的内部调用本类的其他synchronized方法/块

#### 联系我们



请扫描二维码



webmaster



400-660



QQ客服  
客服论坛

[关于](#) [招聘](#) [广告服务](#)

度

©1999-2018 CSDN版权所有  
京ICP证09002463号

经营性网站备案信息

网络110报警服务

中国互联网举报中心

北京互联网违法和不良信息举报中心

```
1 public class ReentrantLockTest implements Runnable {
2     ReentrantLock lock = new ReentrantLock();
3
4     public void get() {
5         lock.lock();
6         System.out.println(Thread.currentThread().getId());
7         set();
8         lock.unlock();
9     }
10
11    public void set() {
12        lock.lock();
13        System.out.println(Thread.currentThread().getId());
14        lock.unlock();
15    }
16
17    @Override
18    public void run() {
19        get();
20    }
21
22    public static void main(String[] args) {
23        ReentrantLockTest ss = new ReentrantLockTest();
24        new Thread(ss).start();
25        new Thread(ss).start();
26        new Thread(ss).start();
27    }
28 }
```

结果是：

11  
11  
12  
12  
13  
13

（可以每次执行的结构不太一样）但可以看到同一个线程id被连续输出两次。

ReentrantLock的正确使用

```
1 public class Test {
2     private ArrayList<Integer> arrayList = new ArrayList<Integer>();
3     public static void main(String[] args) {
4         final Test test = new Test();
5
6         new Thread(){
7             public void run() {
8                 test.insert(Thread.currentThread());
9             };
10        }.start();
11
12        new Thread(){
13            public void run() {
14                test.insert(Thread.currentThread());
15            };
16        }.start();
17    }
18
19    public void insert(Thread thread) {
20        Lock lock = new ReentrantLock();    //注意这个地方
21        lock.lock();
22        try {
23            System.out.println(thread.getName()+"得到了锁");
24            for(int i=0;i<5;i++) {
25                arrayList.add(i);
26            }
27        } catch (Exception e) {
28            // TODO: handle exception
29        }finally {
30            System.out.println(thread.getName()+"释放了锁");
31            lock.unlock();
32        }
33    }
34 }
```

结果：

Thread-0得到了锁



Thread-1释放了锁

为什么会是这么呢？原因在于，在insert方法中的lock变量是局部变量，每个线程执行该方法时都会保存一个副本，那么理所当然每个线程执行到lock.lock()处获取的是不同的锁，所以就不会发生冲突。

正确使用方式是，只需要将lock声明为类的属性即可：

```
1 public class Test {
2     private ArrayList<Integer> arrayList = new ArrayList<Integer>();
3     private Lock lock = new ReentrantLock();    //注意这个地方
4     public static void main(String[] args) {
5         final Test test = new Test();
6
7         new Thread(){
8             public void run() {
9                 test.insert(Thread.currentThread());
10            };
11        }.start();
12
13        new Thread(){
14            public void run() {
15                test.insert(Thread.currentThread());
16            };
17        }.start();
18    }
19
20    public void insert(Thread thread) {
21        lock.lock();
22        try {
23            System.out.println(thread.getName()+"得到了锁");
24            for(int i=0;i<5;i++) {
25                arrayList.add(i);
26            }
27        } catch (Exception e) {
28            // TODO: handle exception
29        }finally {
30            System.out.println(thread.getName()+"释放了锁");
31            lock.unlock();
32        }
33    }
34 }
```

## ReentrantLock可重入锁的使用场景

**场景1**：如果发现该操作已经在执行中则不再执行（有状态执行）

```
1 private ReentrantLock lock = new ReentrantLock();
2 if (lock.tryLock()) { //如果已经被lock，则立即返回false不会等待，达到忽略操作的效果
3     try {
4
5         //操作
6
7     } finally {
8         lock.unlock();
9     }
10
11 }
```

**场景2**：如果发现该操作已经在执行，等待一个一个执行（同步执行，类似synchronized）

```
1 private ReentrantLock lock = new ReentrantLock(); //参数默认false，不公平锁
2 // private ReentrantLock lock = new ReentrantLock(true); //公平锁
3 try {
4     lock.lock(); //如果被其它资源锁定，会在此等待锁释放，达到暂停的效果
5
6     //操作
7
8 } finally {
9     lock.unlock();
10 }
```

**不公平锁与公平锁的区别**：

公平情况下，操作会排一个队按顺序执行，来保证执行顺序。（会消耗更多的时间来排队）  
不公平情况下，是无序状态允许插队，jvm会自动计算如何处理更快速来调度插队。（如果不关心顺序，这个速度会更快）

**场景3**：如果发现该操作已经在执行，则尝试等待一段时间，等待超时则不执行（尝试等待执行）

联系我们



请扫描二维码

webmaster

400-660-

客服  
论坛

务

有

举报

0

目录

收藏

评论

微信

微博

QQ

```
4         //操作
5     } finally {
6         lock.unlock();
7     }
8 }
9 } catch (InterruptedException e) {
10     e.printStackTrace(); //当前线程被中断时(interrupt)，会抛InterruptedException
11 }
```

**场景4**：如果发现该操作已经在执行，等待执行。这时可中断正在进行的操作立刻释放锁继续下一操作。这种情况主要用于取消某些操作对资源的占用。如：（取消正在同步运行的操作，来防止不正常操作长时间占用造成的阻塞）,该操作的开销也很大一般不建议用。

```
1 try {
2     lock.lockInterruptibly();
3     //操作
4
5 } catch (InterruptedException e) {
6     e.printStackTrace();
7 } finally {
8     lock.unlock();
9 }
```

### 下期会介绍volatile 和 ThreadPool

- 参考：
- Java并发编程：synchronized
  - Synchronized关键字总结
  - Java并发编程：Lock
  - Java锁的种类以及辨析（四）：可重入锁

版权声明：转载请标明原文链接。 [https://blog.csdn.net/happy\\_85/article/details/51332966](https://blog.csdn.net/happy_85/article/details/51332966)

文章标签： [java](#) [多线程](#) [线程安全](#) [线程](#)

个人分类： [Java基础](#) [不惑JAVA](#)

查看更多>>

想对作者说点什么？ 

我来说两句

## PHP全栈工程师特训班

课程包含PHP零基础入门、PHP基础编程、数据库、项目实战、linux、框架开发到高级+企业项目实战，深入浅出地剖析和分解了PHP企业级开发项目在实际工作中的应用

学院    2018年04月26日 14:35

## java 中Lock的使用

ReentrantLock Condition

 jihaitaowangyi    2016-10-18 21:04:10    阅读数：5114

## Java中Lock框架学习笔记

锁在多线程编程中有很重要的作用,synchronized比较常见也很常用，但是Lock提供了更广泛的锁操作，处理多线程同步的问题也更加优雅和灵活，Java从Java SE 5之后在并发包中提供Lock...

 qq\_25160969    2017-05-07 16:36:33    阅读数：520

## 《Java中Lock和synchronized的比较和应用》

《Java中Lock和synchronized的比较和应用》 尽管synchronized在语法上已经足够简单了，在JDK 5之前只能借助此实现，但是由于是独占锁，性能却不高，因此JDK 5以后就开始借...

 u010412719    2016-07-30 20:07:46    阅读数：1926

## Java并发编程：Lock



👤 u012554102    2016-07-06 23:23:27    阅读数：1066

### Java中的锁-Lock接口解析

提到java中的锁，相信大家都知道它是用来控制多个线程访问共享资源的方式(即锁能防止多个线程同时访问空享资源而出现线程安全问题)。在实践过程中使用最多的也最常见的锁就是 synchronized 在j...

👤 canot    2016-07-27 23:51:13    阅读数：14731

### 为什么互联网公司都在做小程序开发？

微信请来了硅谷大佬一起开发了门小程序课，了解一下



### java lock的底层实现原理

关于java lock的底层实现原理，讲的有点深，转载学习！ Lock完全用Java写成，在java这个层面是无关JVM实现的。 在java.util.concurrent.locks包中...

👤 abcd1430    2016-10-06 19:08:39    阅读数：2476

### 深入研究 Java Synchronize 和 Lock 的区别与用法

在分布式开发中，锁是线程控制的重要途径。Java为此也提供了2种锁机制，synchronized和lock。做为Java爱好者，自然少不了对比一下这2种机制，也能从中学到些分布式开发需要注意的地方。 ...

👤 natian306    2014-01-19 15:22:20    阅读数：40290

### java5 Lock用法

java5 Lock用法 锁是控制多个线程对共享资源进行访问的工具。通常，锁提供了对共享资源的独占访问。一次只能有一个线程获得锁，对共享资源的所有访问都需要首先获得锁。不过，某些锁可能允许对共享...

👤 wanglha    2016-04-01 09:38:31    阅读数：709

### Java线程(八)：锁对象Lock-同步问题更完美的处理方式

Lock是java.util.concurrent.locks包下的接口，Lock 实现提供了比使用synchronized 方法和语句可获得的更广泛的锁定操作，它能以更优雅的方式处理线程同步问题，我...

👤 ghsau    2012-04-14 22:29:43    阅读数：103572

### Java锁--Lock实现原理(底层实现)

关于java lock的底层实现原理，讲的有点深，转载学习！ Lock完全用Java写成，在java这个层面是无关JVM实现的。在java.util.concurrent.locks包中有很多Lock的...

👤 Luxia\_24    2016-09-01 20:11:56    阅读数：9842

### Java 并发编程（5）—— Lock

1 synchronized 的缺陷    synchronized 是 java 中的一个关键字，也就是说 Java 语言内置的特性。那么为什么会出现 Lock 呢？    在上面一篇文章中，我们了解到如...

👤 HeatDeath    2018-01-01 18:21:05    阅读数：389

### Java Lock-同步的另一种实现

通常初级的程序员喜欢使用synchronized关键字来实现同步机制，理由很简单，使用它简单，我们不用考虑更多的细节，对程序员的要求比较低。那这里我们介绍另外一种通过Lock实现的同步的方法，显然使用...

👤 dingji\_ping    2016-04-04 16:02:11    阅读数：1771

### Java中的ReentrantLock和synchronized两种锁定机制的对比

原文：http://www.ibm.com/developerworks/cn/java/j-jtp10264/index.html 多线程和并发性并不是什么新内容，但是 Java 语言设计中的创新...

👤 fw0124    2011-08-09 15:03:01    阅读数：217764

### 为什么互联网公司都在做小程序开发？

微信请来了硅谷大佬一起开发了门小程序课，了解一下



### Java多线程之Lock的使用

import java.util.concurrent.ExecutorService; import java.util.concurrent.Executors; import java.util...

👤 huang\_xw    2011-12-21 09:48:37    阅读数：63839

### java多线程之Lock类的使用

1.ReentrantLock类的使用    1.1ReentrantLock实现线程间同步 public class MyService { private Lock lock=new R...

👤 ya\_1249463314    2016-09-26 23:35:59    阅读数：3266

..... 共 4 页 1 2 3 4 下一页

摘要： 我们已经知道，synchronized 是Java的关键字，是Java的内置特性，在JVM层面实现了对临界资源的同步互斥访问，但 synchronized 粒度有些大，在处理实际问...

 u012767369 2017-02-14 14:06:01 阅读数：133

## Java 中的锁之Lock接口

锁是控制多个线程访问共享资源的方式，一般来说，一个锁能够防止多个线程同时访问共享资源（但是有些锁是允许多个并发线程的访问，如读写锁），在Lock接口出现之前，Java程序靠synchronized关键...

 jcsyl\_mshot 2018-03-28 14:58:05 阅读数：63

## Java中Synchronized的用法

synchronized是Java中的关键字，是一种同步锁。它修饰的对象有以下几种： 1. 修饰一个代码块，被修饰的代码块称为同步语句块，其作用的范围是大括号{}括起来的代码，作用的对象是调用这个代...

 luoweifu 2015-06-24 00:25:01 阅读数：295457


## 共享锁和排它锁（ ReentrantReadWriteLock ）

1、什么是共享锁和排它锁 共享锁就是允许多个线程同时获取一个锁，一个锁可以同时被多个线程拥有。 排它锁，也称作独占锁，一个锁在某一时刻只能被一个线程占有，其它线程必须等待锁被释...

 yanlinwang 2014-11-16 14:27:22 阅读数：9483

## JAVA并发编程: CAS和AQS

说起JAVA并发编程，就不得不聊聊CAS(Compare And Swap)和AQS了(AbstractQueuedSynchronizer)。 CAS(Compare And Swap) ...

 u010862794 2017-06-06 23:55:04 阅读数：3302

## 轻松学习java可重入锁(ReentrantLock)的实现原理

前言相信学过java的人都知道 synchronized 这个关键词，也知道它用于控制多线程对并发资源的安全访问，兴许，你还用过Lock相关的功能，但你可能从来没有想过java中的锁底层的机制是怎么实...

 yanyan19880509 2016-08-28 14:17:46 阅读数：26720

## 轻松掌握java读写锁(ReentrantReadWriteLock)的实现原理

前言前面介绍了java中排它锁，共享锁的底层实现机制，本篇再进一步，学习非常有用的读写锁。鉴于读写锁比其他的锁要复杂，不想堆一大波的文字，本篇会试图图解式说明，把读写锁的机制用另外一种方式阐述。 ...

 yanyan19880509 2016-09-04 22:08:32 阅读数：10322

## java并发-独占锁与共享锁

1 锁的独占与共享 java并发包提供的加锁模式分为独占锁和共享锁，独占锁模式下，每次只能有一个线程能持有锁，ReentrantLock就是以独占方式实现的互斥锁。共享锁，则允许多个线程同...

 wojiushiwo945you 2014-12-31 11:34:45 阅读数：11124

## 腾讯爸爸竟然和外国企业打造小程序开发？

席位有限，立即行动！抢先掌握稀缺技术，成为抢手人才



## java中synchronized和lock底层原理

JVM中锁的优化： 简单来说在JVM中monitorenter和monitorexit字节码依赖于底层的操作系统的Mutex Lock来实现的，但是由于使用Mutex Lock需要将当前线程挂起并从...

 SumResort\_LChaowei 2017-06-04 11:20:31 阅读数：719

## java synchronized与lock区别

转自：http://blog.csdn.net/liaomin416100569/archive/2010/01/11/5172652.aspxsynchronized 修饰方法时 表示同一个对象在不...

 FG2006 2011-05-08 22:56:00 阅读数：12980

## Java并行编程-lock中使用多条件condition（生产者消费者模式实例）

Java 并发包下的提供Lock，Lock相对于Synchronized可以更好的解决线程同步问题，更加的灵活和高效，并且ReadWriteLock锁还能实现读、写的分离。但线程间仅仅互斥是不够的，还...

 chenchaofuck1 2016-06-05 23:37:27 阅读数：9316

## java并发 lock锁

Java并发编程：Lock 在上一篇文章中我们讲到了如何使用关键字synchronized来实现同步访问。本文我们继续来探讨这个问题，从Java 5之后，在java.util.concurrent...

 takemetofly 2015-08-29 11:40:02 阅读数：2903

1.显示锁与内置锁对比 内置锁在代码块调用结束后会自动释放锁，但是显示锁必须自己控制加锁和释放锁，因此使用显示锁更加危险（忘记释放锁）。 内置锁不支持中断，阻塞的线程会一直等待直到拥有锁。而显示锁支持...

 u013475071    2016-04-26 23:33:13    阅读数：789

## JAVA中synchronized和lock详解

目前在Java中存在两种锁机制：synchronized和Lock，Lock接口及其实现类是JDK5增加的内容，其作者是大名鼎鼎的并发专家Doug Lea。本文并不比较synchronized与Lo  
c...

 guomei    2013-12-10 02:01:59    阅读数：2244

## 并发编程学习总结(四)：java 显式锁ReentrantLock使用详解之lock()\unlock() 加锁与释放锁

在大多数实际的多线程应用中，两个或两个以上的线程需要共享对同一数据的存取。如果两个线程存取相同的对象，并且每一个线程都调用了一个修改该对象状态的方法，那  
么线程彼此踩了对方的脚，根据各线程访问数据的次序...

 u011784767    2016-05-17 20:53:54    阅读数：4005

## Java并发编程系列之十六：Lock锁

Lock锁简介Lock锁机制是JDK 5之后新增的锁机制，不同于内置锁，Lock锁必须显式声明，并在合适的位置释放锁。Lock是一个接口，其由三个具体的实现：ReentrantLoc  
k、Reetran...

 u011116672    2016-04-05 13:50:42    阅读数：2555


## 【Java线程】锁机制：synchronized、Lock、Condition

Lock可以实现synchronized的相同功能，它能以更优雅的方式处理线程同步问题。 与互斥锁定相比，读-写锁定允许对共享数据进行更高级别的并发访问。虽然一次只有一个  
线程（writer 线程）可以...

 vking\_wang    2013-08-14 17:15:55    阅读数：80588

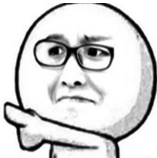
## 深入java并发Lock一

深入java并发Lock一

 zhaozhenzuo    2014-07-05 23:06:39    阅读数：10310

## 50万码农评论：英语对于程序员有多重要？

不背单词和语法，一个公式学好英语



## Lock synchronized 详细讲解

Synchronzied：我们先来看下synchronized这个关键字，Java中这个类似于锁的东东， Synchronzied关键字的作用一个词概括就是：线程同步。它可以...

 zhuangyalei    2015-09-23 20:22:57    阅读数：732

## 线程进阶：多任务处理（17）——Java中的锁（Unsafe基础）

本专题在之前的文章中详细介绍了Java中最常使用的一种锁机制——同步锁。但是同步锁肯定是不适合在所有应用场景中使用的。所以从本文开始，笔者将试图通过两到三篇  
文章的篇幅向读者介绍Java中锁的分类、原理...

 yinwenjie    2017-06-14 18:41:06    阅读数：14512


## 类锁 synchronized对象锁 和 Lock对象锁

在并发环境下，解决共享资源冲突问题时，可以考虑使用锁机制。1. 对象锁所有对象都自动含有单一的锁。JVM负责跟踪对象被加锁的次数。如果一个对象被解锁，其计数变  
为0。在任务（线程）第一次给对象加锁的时候...

 qq\_33530388    2017-03-18 18:45:49    阅读数：1720


## Java中Synchronized和Lock的使用和区别

转载自：http://blog.csdn.net/imzoer/article/details/9457639 Lock的锁定是通过代码实现的，而 synchronized 是在 JVM 层面...

 zgjkflmkyc    2016-03-02 11:43:42    阅读数：4188


## java同步机制对象锁使用方式比较

```
class Sync { private byte[] lock = new byte[0]; public void sync() throws InterruptedExcep...
```

 Kingson\_Wu    2015-04-09 14:16:59    阅读数：1339

## 《Java源码分析》：ReentrantLock.lock 锁机制

《Java源码分析》：ReentrantLock.lock 锁机制AbstractQueuedSynchronizer这个类真的很难也很复杂，是构建锁以及实现其他相关同步类的基础框架。本来是没有打算...

 u010412719    2016-08-01 10:47:05    阅读数：2242

Lock完全用Java写成，在java这个层面是无关JVM实现的。 在java.util.concurrent.locks包中有很多Lock的实现类，常用的有ReentrantLock、ReadWr...

 wl6965307    2016-04-26 11:35:06    阅读数：11873

## Java的LockSupport.park()实现分析

LockSupport类是Java6(JSR166-JUC)引入的一个类，提供了基本的线程同步原语。LockSupport实际上是调用了Unsafe类里的函数，归结到Unsafe里，只有两个函数： ...

 hengyunabc    2014-06-03 02:52:12    阅读数：22406

## java并发包系列---LockSupport

长久以来对线程阻塞与唤醒经常我们会使用object的wait和notify,除了这种方式，java并发包还提供了另外一种方式对线程进行挂起和恢复，它就是并发包子包locks提供的LockSupport...

 opensure    2016-11-26 11:35:02    阅读数：2321

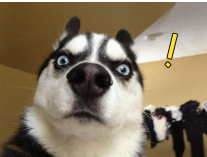
## Java中Lock的使用

Lock和Synchronized的功能类似，都可以实现线程的同步，

 lzm1340458776    2014-05-31 17:08:21    阅读数：906


## 程序猿不会英语怎么行？英语文档都看不懂！

老司机教你用数学公式读懂天下英文→



## Java中synchronized和Lock实现并发锁

前言 总结

 Alone\_Rojer    2017-03-06 21:07:15    阅读数：332


## java中线程同步Synchronized，监视器monitor和锁lock的关系是什

既然有关监视器monitor的概念比较难，大家怎么解释的都有。首先我给出一下java的官方文档，也是最权威的解释： Synchronizationis built around an interna...

 mark\_to\_win    2017-04-12 12:56:13    阅读数：1386

## java并发编程---synchronized和lock两种锁的比较

性能比较 在JDK1.5中，synchronized是性能低效的。因为这是一个重量级操作，它对性能最大的影响是阻塞的是实现，挂起线程和恢复线程的操作都需要转入内核态中完成，这些操作给系统的并...

 u012470138    2016-12-06 11:27:42    阅读数：3116


## java同步锁中synchronized和Lock接口类的区别

Lock提供了和synchronized类似的同步功能，只是在使用时需要显示地获取和释放锁。虽然Lock缺少了synchronized隐式获取释放锁的便捷性，但是却拥有了锁获取与是释放的可操作性、可中断的...

 xuqiaobo    2016-05-23 10:54:10    阅读数：2001

## lock和synchronized的同步区别与选择

区别如下： 1. lock是一个接口，而synchronized是java的一个关键字，synchronized是内置的语言实现；（具体实现上的区别在《Java虚拟机》中有讲解底层的CAS不同，以...

 qq\_24486393    2017-02-04 15:14:07    阅读数：4453

## java并发之Lock与synchronized的区别

1 ) Lock是一个接口，而synchronized是Java中的关键字，synchronized是内置的语言实现；        2 ) synchronized在发生异常时，会自动释放线程占有的锁，因此不会导致...

 gongpulin    2016-04-21 19:22:06    阅读数：3611

没有更多推荐了，[返回首页](#)