synchronized与lock

8锁理解

List不安全

举例说明list线程不安全

CopyOnWriteArrayList浅析

CopyOnWriteArrayList的几个要点

CopyOnWrite的缺点

CopyOnWriteArrayList为什么并发安全且性能比Vector好

Set 不安全

Map 不安全

## synchronized与lock

1. 多线程交互中,必须防止多线程的虚假唤醒,也即(判断只用while,不能用if)。

什么是虚假唤醒:

当一个条件满足时,很多线程都被唤醒了,但是只有其中部分是有用的唤醒,其它的唤醒都是无用功。

为什么 if会出现虚假唤醒:

因为if只会执行一次,执行完会接着向下执行if()外边的,

而while不会,直到条件满足才会向下执行while()外边的。

2. synchronized的新写法为lock

Java 同 复制代码

- private Lock lock = new ReentrantLock();
- private Condition condition = lock.newCondition();
- 3 其中 wait()被condition.await()代替, notifyAll()被condition.signalAll()代替.

lock的使用方式:

ijang.liu

3. 为什么要用lock代替synchronized?

word lock可以定义多个condition(可以理解为一把锁有多个钥匙),精准通知,精准唤醒。 通过设置多个condition精确通知需要唤醒的线程,可以指定线程顺序调用。

## 8锁理解

① 非静态方法的默认锁是this, 静态方法的默认锁是class类。

ijang liu ②某一时刻内,只能有一个线程有锁,无论几个方法。

a liu

iiang.liu

<sub>liang liu</sub>

```
Java
                                                                   □ 复制代码
1
    /*
2
    1、标准的访问情况下,先执行 sendEmail 还是 sendSMS
 3
4
       答案: sendEmail
5
       被 synchronized 修饰的方法, 锁的对象是方法的调用者也就是实际new的对象, 所以说这里两个
    方法调用的对象是同一个
       先调用的先执行!
6
7
     */
8
    public class LockDemo01 {
        public static void main(String[] args) throws InterruptedException {
            Phone phone = new Phone();
10
11
12
            new Thread(()->{phone.sendEmail();},"A").start();
           TimeUnit.SECONDS.sleep(2);
13
            new Thread(()->{phone.sendSMS();},"B").start();
14
15
        }
16
    }
    class Phone{
17
        public synchronized void sendEmail(){
18
19
            System.out.println("sendEmail");
20
        }
        public synchronized void sendSMS(){
21
            System.out.println("sendSMS");
22
23
        }
    }
24
```

jiang.liu

jiang.liu jiang.liu

```
Java  复制代码
 1
    /*
 2
    2、sendEmail休眠4秒后 ,先执行 sendEmail 还是 sendSMS
 3
 4
       答案: sendEmail
 5
      被 synchronized 修饰的方法,锁的对象是方法的调用者,所以说这里两个方法调用的对象是同一
    个。 先调用的先执行! 执行sleep()方法的线程并不会释放锁。
 6
 7
      阳哥笔记:
 8
       一个对象里面如果有多个synchronized方法,某一个时刻内,只要一个线程去调用其中的一个
    synchronized方法了,其他线程都只能等待,换句话说,某一个时刻内,只能有唯一一个线程去访问这
    些synchronized方法,锁的是当前对象this,被锁定后,其他线程都不能进入到当前对象的其他的
    synchronized方法。
    */
9
    public class LockDemo02 {
10
11
        public static void main(String[] args) throws InterruptedException {
           Phone2 phone = new Phone2();
12
13
           new Thread(()->{
14
15
               try {
16
                  phone.sendEmail();
17
               } catch (InterruptedException e) {
                  e.printStackTrace();
18
19
           },"A").start();
20
21
22
           //Thread.sleep(200);
23
           TimeUnit.SECONDS.sleep(2);
24
           new Thread(()->{
25
26
               phone sendSMS():
27
           },"B").start();
28
       }
29
    }
30
31
    class Phone2{
32
       public synchronized void sendEmail() throws InterruptedException {
33
           TimeUnit.SECONDS.sleep(4);
           System.out.println("sendEmail");
34
       }
37
       public synchronized void sendSMS(){
           System.out.println("sendSMS");
39
       }
    }
40
```

Java 复制代码 1 /\* 2 3、增加一个普通方法,请问先打印那个 sendEmail 还是 hello 3 4 答案: hello 5 新增加的这个方法没有 synchronized 修饰, 不是同步方法, 不受锁的影响! 6 \*/ public class LockDemo03 { 7 public static void main(String[] args) throws InterruptedException { 8 9 Phone3 phone = new Phone3(); 10 new Thread(()->{ 11 12 try { phone.sendEmail(); 13 } catch (InterruptedException e) { 14 15 e.printStackTrace(); 16 } 17 },"A").start(); TimeUnit.SECONDS.sleep(1); 18 19 20 new Thread(()->{ phone.hello(); 21 },"B").start(); 22 23 } 24 } 25 26 class Phone3{ 27 public synchronized void sendEmail() throws InterruptedException { TimeUnit.SECONDS.sleep(4); 28 29 System.out.println("sendEmail"); 30 } 31 // 没有 synchronized 没有 static 就是普通方法 32 public void hello(){ 33 System.out.println("hello"); 34

ijang.liu ijang.liu

35

}

}

```
Java  复制代码
1
    /*
 2
    4、两个手机,请问先执行sendEmail 还是 sendSMS
 3
        答案: sendSMS
 4
        被 synchronized 修饰的方式,锁的对象是调用者;我们这里有两个调用者,两个方法在这里是
    两个锁
 5
     */
    public class LockDemo04 {
 6
7
        public static void main(String[] args) throws InterruptedException {
8
            Phone4 phone1 = new Phone4();
            Phone4 phone2 = new Phone4();
10
            new Thread(()->{
11
12
                try {
                    phone1.sendEmail();
13
14
                } catch (InterruptedException e) {
15
                    e.printStackTrace();
                }
16
17
            },"A").start();
18
19
            //Thread.sleep(200);
20
            TimeUnit.SECONDS.sleep(1);
21
22
            new Thread(()->{
23
                phone2.sendSMS();
            },"B").start();
24
25
        }
26
    }
27
    class Phone4{
28
29
        public synchronized void sendEmail() throws InterruptedException {
30
            TimeUnit.SECONDS.sleep(3);
31
            System.out.println("sendEmail");
32
        }
34
        public synchronized void sendSMS(){
            System.out.println("sendSMS");
        }
    }
37
```

```
□ 复制代码
                                                                 Java
 1
    /*
 2
    5、两个静态同步方法,同一个手机请问先执行sendEmail 还是 sendSMS
 3
 4
        答案: sendEmail
 5
        只要方法被 static 修饰,锁的对象就是 Class模板对象,这个则全局唯一! 所以说这里是同一个
    锁
        并不是因为synchronized
 6
 7
     */
8
    public class LockDemo05 {
 9
        public static void main(String[] args) throws InterruptedException {
10
            Phone 5 phone = new Phone 5();
11
12
            new Thread(()->{
13
14
                try {
15
                    phone.sendEmail();
                } catch (InterruptedException e) {
16
                    e.printStackTrace();
17
18
                }
19
            },"A").start();
20
21
            //Thread.sleep(200);
22
            TimeUnit.SECONDS.sleep(1);
23
            new Thread(()->{
24
25
                phone.sendSMS();
26
            },"B").start();
        }
27
28
    }
29
30
    class Phone5{
31
32
        public static synchronized void sendEmail() throws InterruptedException {
            TimeUnit.SECONDS.sleep(3);
            System.out.println("sendEmail");
34
        }
37
        public static synchronized void sendSMS(){
            System.out.println("sendSMS");
        }
39
40
41
    }
```

iiang.liu

```
□ 复制代码
                                                                 Java
 1
    /*
 2
    6、两个静态同步方法,两个手机,请问先执行sendEmail 还是 sendSMS
 3
4
        答案: sendEmail
 5
        只要方法被 static 修饰,锁的对象就是 Class模板对象,这个则全局唯一! 所以说这里是同一个
    锁
        并不是因为synchronized
 6
 7
     */
8
    public class LockDemo06 {
9
        public static void main(String[] args) throws InterruptedException {
10
            Phone6 phone = new Phone6();
11
            Phone6 phone2 = new Phone6();
12
            new Thread(()->{
13
14
                try {
15
                    phone.sendEmail();
                } catch (InterruptedException e) {
16
17
                    e.printStackTrace();
18
                }
19
            },"A").start();
20
            //Thread.sleep(200);
21
22
            TimeUnit.SECONDS.sleep(1);
23
            new Thread(()->{
24
25
                phone2.sendSMS();
26
            },"B").start();
        }
27
28
    }
29
30
    class Phone6{
31
32
        public static synchronized void sendEmail() throws InterruptedException {
            TimeUnit.SECONDS.sleep(3);
            System.out.println("sendEmail");
34
        }
37
        public static synchronized void sendSMS(){
            System.out.println("sendSMS");
        }
39
40
    }
41
```

```
□ 复制代码
                                                                 Java
1
    /*
 2
    7、一个普通同步方法,一个静态同步方法,只有一个手机,请问先执行sendEmail 还是 sendSMS
 3
 4
        答案: sendSMS
 5
        synchronized 锁的是这个调用的对象
        static 锁的是这个类的Class模板
 6
 7
        这里是两个锁! 互不影响
8
     */
9
    public class LockDemo07 {
10
        public static void main(String[] args) throws InterruptedException {
            Phone7 phone = new Phone7();
11
12
13
            new Thread(()->{
14
                try {
15
                    phone.sendEmail();
16
                } catch (InterruptedException e) {
17
                    e.printStackTrace();
18
                }
            },"A").start();
19
20
21
            //Thread.sleep(200);
            TimeUnit.SECONDS.sleep(1);
22
23
24
            new Thread(()->{
25
                phone.sendSMS();
26
            },"B").start();
27
        }
28
    }
29
30
    class Phone7{
31
32
        public static synchronized void sendEmail() throws InterruptedException {
33
            TimeUnit.SECONDS.sleep(3);
            System.out.println("sendEmail");
34
        }
35
37
        public synchronized void sendSMS(){
            System.out.println("sendSMS");
```

iiang.liu

39 40 41

}

}

```
□ 复制代码
                                                                  Java
1
    1 /*
 2
     2 8、一个普通同步方法,一个静态同步方法,两个手机,请问先执行sendEmail 还是 sendSMS
 3
4
           答案: sendSMS
     4
5
     5
           synchronized 锁的是这个调用的对象
           static 锁的是这个类的Class模板
6
     7
7
           这里是两个锁!
     8
       */
9
     9 public class LockDemo08 {
           public static void main(String[] args) throws InterruptedException {
10
    10
               Phone8 phone = new Phone8();
11
    11
               Phone8 phone2 = new Phone8();
12
    12
13
    13
14
    14
               new Thread(()->{
15
    15
                   try {
                       phone.sendEmail();
16
    16
    17
                   } catch (InterruptedException e) {
17
18
    18
                       e.printStackTrace();
19
    19
20
    20
               },"A").start();
21
    21
22
    22
               //Thread.sleep(200);
               TimeUnit.SECONDS.sleep(1);
23
    23
24
    24
25
    25
               new Thread(()->{
    26
                   phone2.sendSMS();
26
    27
               },"B").start();
27
28
    28
           }
29
    29 }
30
    30
    31 class Phone8{
31
32
    32
33
           public static synchronized void sendEmail() throws
    InterruptedException {
34
    34
               TimeUnit.SECONDS.sleep(3);
    35
               System.out.println("sendEmail");
           }
    36
37
    37
    38
           public synchronized void sendSMS(){
39
    39
               System.out.println("sendSMS");
40
    40
           }
41
    41
42
    42 }
```

#### 8锁小结

- 1、new this 调用的这个对象,是一个具体的对象!
- 2、static class 唯一的一个模板!

在我们编写多线程程序得时候,只需要搞明白这个到底锁的是什么就不会出错了!

synchronized(Demo.class){

| \*\*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
| \*\*|
|

synchronized(this){
}//等同于synchronized普通方法

## List不安全

### 举例说明list线程不安全

注意: ArrayList不是为并发情况而设计的集合类

举例: 多个线程操作ArraList, 边写边读, 会出现并发修改异常.

故障现象:

java.util.ConcurrentModificationException 并发修改异常.

#### 导致原因:

#### 解决方法:

- 1) List<String> list = new Vector<>(); // vector 线程安全, add方法有synchronized, 读写性能低,同一时间段只能一个人读/写.
- 2) List<String> list = Collections.synchronizedList(new ArrayList<>()); // Collections工具类提供,将ArrayList转换为线程安全.
- <sub>kang</sub>™3)List<String> list = new CopyOnWriteArrayList(); // JUC包提供的 写时复制(读写分离思想).

## CopyOnWriteArrayList浅析

和ArrayList一样,其底层数据结构也是数组,加上transient不让其被序列化,加上volatile修饰来保证多线程下的其可见性和有序性。

其修改操作是基于fail-safe机制,像我们的String一样,不在原来的对象上直接进行操作,而是复制一份对其进行修改,另外此处的修改操作是利用Lock锁进行上锁的,所以保证了线程安全问题。

### CopyOnWriteArrayList的几个要点

- 实现了List接口
- 内部持有一个ReentrantLock lock = new ReentrantLock();
- 底层是用volatile transient声明的数组 array

• 读写分离,写时复制出一个新的数组,完成插入、修改或者移除操作后将新数组赋值给array volatile (挥发物、易变的): 变量修饰符,只能用来修饰变量。volatile修饰的成员变量在每次被线程访问时,都强迫从共享内存中重读该成员变量的值。而且,当成员变量发生变化时,强迫线程将变化值回写到共享内存。这样在任何时刻,两个不同的线程总是看到某个成员变量的同一个值。

**transient** (暂短的、临时的): 修饰符,只能用来修饰字段。在对象序列化的过程中,标记为transient的变量不会被序列化。

### CopyOnWrite的缺点

CopyOnWrite容器有很多优点,但是同时也存在两个问题,即内存占用问题和数据一致性问题。所以在 开发的时候需要注意一下。

内存占用问题。因为CopyOnWrite的写时复制机制,所以在进行写操作的时候,内存里会同时驻扎两个对象的内存,旧的对象和新写入的对象(注意:在复制的时候只是复制容器里的引用,只是在写的时候会创建新对象添加到新容器里,而旧容器的对象还在使用,所以有两份对象内存)。如果这些对象占用的内存比较大,比如说200M左右,那么再写入100M数据进去,内存就会占用300M,那么这个时候很有可能造成频繁的Yong GC和Full GC。之前我们系统中使用了一个服务由于每晚使用CopyOnWrite机制更新大对象,造成了每晚15秒的Full GC,应用响应时间也随之变长。

针对内存占用问题,可以通过压缩容器中的元素的方法来减少大对象的内存消耗,比如,如果元素全是10进制的数字,可以考虑把它压缩成36进制或64进制。或者不使用CopyOnWrite容器,而使用其他的并发容器,如ConcurrentHashMap。

**数据一致性问题**。CopyOnWrite容器只能保证数据的最终一致性,不能保证数据的实时一致性。所以如果你希望写入的的数据,马上能读到,请不要使用CopyOnWrite容器。

### CopyOnWriteArrayList为什么并发安全且性能比Vector好

我们知道Vector是增删改查方法都加了synchronized,保证同步,但是每个方法执行的时候都要去获得锁,性能就会大大下降,而CopyOnWriteArrayList 只是在增删改上加锁,但是读不加锁,在读方面的性能就好于Vector,CopyOnWriteArrayList支持读多写少的并发情况。

## Set 不安全

HashSet底层数据结构是HashMap。 hashSet的add方法底层调用的就是hashMap的put方法,为什么hashMap的put是2个参数,而hashSet的add是一个参数呢,因为add进去的一个值就是put的key,value永远是一个Object类型的常量: PRSENT,固定写死的。

Set<String> set = new CopyOnWriteArraySet(); // new HashSet<>();

Collections.synchronizedArraySet(new HashSet());

# Map 不安全

Map<String,Object> map = new ConCurrentHashMap<>(); // new HashMap<>();
Collections.synchronizedMap();

<sub>jiang liu</sub>