

Java中wait()方法为什么要放在同步块中?



Myabe
我是myabe, 一个风一样帅气的男子!

关注他

1 人赞同了该文章

关于问题

我在工作的时候, 有一位组员问题一个问题:如果wait()方法不放在同步代码块会怎样?

我马上要开会忙得不可开交, 只是回答了一句话: “ 规定 ”。

等到有时间了, 我仔细回顾下, 如果wait()方法不在同步块中, 代码的确会抛出IllegalMonitorStateException:

```
@Test
public void test() {
    try {
        new Object().wait();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}

java.lang.IllegalMonitorStateException
    at java.lang.Object.wait(Native Method)
    at java.lang.Object.wait(Object.java:502)
    at DemoTest.test(DemoTest.java:10) <25 intern@117539609>
```

但是, 为毛呢? ? 为毛呢? 我也不知道啊, 经过一番查阅, 我找到了答案。

事情得从一个多线程编程里面臭名昭著的问题"Lost wake-up problem"说起。



这个问题并不是说只在Java语言中会出现，而是会在所有的多线程环境下出现。

假如我们有两个线程，一个消费者线程，一个生产者线程。生产者线程的任务可以简化成先将count加一，而后唤醒消费者；消费者则是先将count减一，而后在减到0的时候陷入睡眠：

生产者伪代码：

```
count+1;
notify();
```

消费者伪代码：

```
while(count<=0)
    wait();

count--
```

这里面有问题。什么问题呢？

生产者是两个步骤：

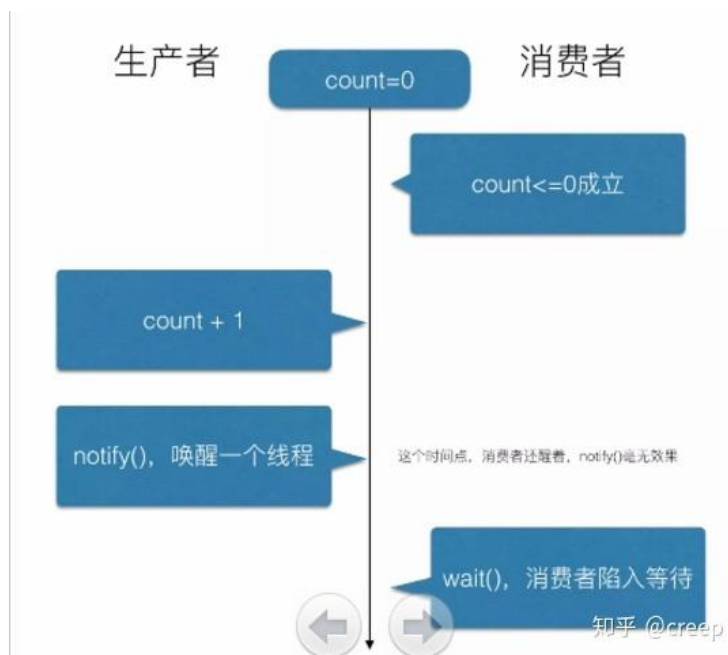
1. count+1;
2. notify();

消费者也是两个步骤：

1. 检查count值;
2. 睡眠或者减一;

如果这些步骤混在一起会怎样呢？

比如说，初始的时候count等于0，这个时候消费者检查count的值，发现count小于等于0的条件成立；就在这个时候，发生了上下文切换，生产者进来了，噼里啪啦一顿操作，把两个步骤都执行完了，也就是发出了通知，准备唤醒一个线程。这个时候消费者刚决定睡觉，还没睡呢，所以这个通知就会被丢掉。紧接着，消费者就睡过去了，消费者成睡美人了。



这就是所谓的lost wake up问题。



尝试解决

问题的根源在于，消费者在检查count到调用wait()之间，count就可能被改掉了。

常见的解决方式是加锁，让消费者和生产者竞争一把锁，竞争到了的，才能够修改count的值。

我这里将两者的两个操作都放进去了同步块中，于是生产者的代码是：

```
tryLock()

count+1

notify()

releaseLock()
```

消费者的代码是：

```
tryLock()

while(count <= 0)
    wait()

count-1

releaseLock()
```

但是这样改后依旧会出现lost wake up问题，而且和无锁的表现是一样的。

最终解决

为了避免出现这种lost wake up问题，在这种模型之下，应该将我们的代码放进去的同步块中。

Java强制我们的wait()/notify()调用必须要在一个同步块中，就是不想让我们在不经意间出现这种lost wake up问题。

不仅仅是这两个方法，包括java.util.concurrent.locks.Condition的await()/signal()也必须要在同步块中：

```
private ReentrantLock lock = new ReentrantLock();
private Condition condition = lock.newCondition();
@Test
public void test2() {
    try {
        condition.signal();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

```
"C:\Program Files\Java\jdk1.8.0_191\bin\java.exe" ...
java.lang.IllegalMonitorStateException <1 internal call>
    at DemoTest.test2(DemoTest.java:26) <25 internal calls>
```

准确的来说，即便是我们自己在实现自己的锁机制的时候，也应该要确保类似于wait()和notify()这种调用，要在同步块内，防止使用者出现lost wake up问题。

Java的这种检测是很严格的。它要求的是，一定要处于锁对象的同步块中。举例来说：

```
private Object obj = new Object();
private Object anotherObj = new Object();
@Test
public void test3() {
    synchronized (obj) {
        try {
            anotherObj.notify();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
"C:\Program Files\Java\jdk1.8.0_191\bin\java.exe" ...
java.lang.IllegalMonitorStateException
    at java.lang.Object.notify(Native Method)
    at DemoTest.test3(DemoTest.java:40) <25 internal calls>
```

这样也是没什么用的。一样出现IllegalMonitorStateException。

提高面试技巧

假如面试官问你这个问题了，你不要一五一十的全部说出来。只需要轻描淡写地说：“这是Java设计者为了避免使用者出现lost wake up问题而搞出来的”，其中的逼格大家自己去体会。

发布于 2020-03-26 17:04

同步 数据同步 Java

推荐阅读



Java多线程同步的5种方法

我在代码里遨游

【从入门到放弃-Java】并发编程-锁-synchronized

简介上篇 【从入门到放弃-Java】并发编程-线程安全中，我们了解到，可以通过加锁机制来保护共享对象，来实现线程安全。synchronized是java提供了一种内置的锁机制。通过synchronized...

阿里云云栖... 发表于我是程序员



Rclone Browser 让你在Linux 中以图形化的方式与云...

Linux... 发表于Linux...


那些年，我们见过的J端“问题”

导读明代著名的心学集大明先生在《传习录》中有精粗，人之所见有精粗。...房，人初进来，只见一个此。处久，便柱壁之类，...明白。再久，如柱上有些...阿里云云栖... 发表于...

4 条评论

切换为时间排序

写下你的评论...



啃冰块的熊猫2020-12-11

规定“把wait方法放在同步代码块中”的原因，是因为调用wait方法时，会将线程放入管程的条件变量的等待队列中，Java的同步代码块中才会有管程存在，线程才能放入条件变量的等待队列中，而同步代码块之外，不存在管程，更没有“把线程放入条件变量的等待队列”这一说。这和“Lost wake-up problem”没有任何关系