Java 中的线程：对象锁 - I

切坦·库达尔卡

2021 年 6 月 14 日 警察

帮助读者了解 Java 中锁概念的系列文章中的第一篇

本系列文章试图帮助读者了解 Java 中的锁概念。假设读者对 Java 和 Java 中的线程创建有一些先验知识。

介绍

对于大多数探索和使用 Java 的开发人员来说，Java 中的线程是最具挑战性的主题之一。已经有很多关于这个主题的书，但顺便说一句，这些都是用非常沉重的技术语言写成的，以至于许多参赛者在阅读了几页充满沉重技术术语的页面后就失去了兴趣。这是尝试使用 Java 中的线程使事情变得简单易懂。本系列文章试图帮助了解 Java 中的锁概念。

背景

让我们探索一下 Java 中锁的概念。有一个典型的问题“你知道Java中的对象锁吗？你知道Java中的类级锁吗？”接下来的文章将演示 Java 中对象锁和类级锁的概念。

考虑以下程序：

爪哇 复制代码

```
locks.java
==========

class processor implements Runnable{
    public void run(){
        display();
    }
    public void display(){
        for (int i = 0 ; i<= 5; i++) {
            System.out.println("i = " + i + " In thread" + Thread.currentThread());
        }
    }
}

class locks {
    public static void main(String[] args)
    {
        processor p1 = new processor();
        Thread t1 = new Thread(p1, "t1");
        Thread t2 = new Thread(p1, "t2");
        t1.start();
```

```
        t2.start();
    }
}
```

上面的程序尝试创建两个线程，每个线程尝试显示从 1 到 10 的数字。它有 2 个类，即“ processor”和“ locks”。

类“ processor”实现Runnable接口，因为它的主要工作是表示一个打印数字 1 到 10 的线程，它通过通过run表示线程的方法调用 display 方法来完成。

在类“ locks”中，我们有“ main()”方法。在其中，我们创建了一个名为“ p1”的处理器类对象。接下来，我们基于同一个对象“ ”创建线程“ t1”和t2“ p1”。需要记住的一点是，这里的两个线程都基于同一个对象“ p1”，并且“ p1”是从实现Runnable表示要执行的线程代码的类中实例化的。

上面的程序编译并成功运行以产生以下输出：

复制代码

```
C:\javaproj\locks>javac locks.java

C:\javaproj\locks>java locks
i = 0 In threadThread[t1,5,main]
i = 0 In threadThread[t2,5,main]
i = 1 In threadThread[t1,5,main]
i = 2 In threadThread[t1,5,main]
i = 1 In threadThread[t2,5,main]
i = 3 In threadThread[t1,5,main]
i = 2 In threadThread[t2,5,main]
i = 4 In threadThread[t1,5,main]
i = 3 In threadThread[t2,5,main]
i = 5 In threadThread[t1,5,main]
i = 4 In threadThread[t2,5,main]
i = 5 In threadThread[t2,5,main]
```

如果仔细观察输出，它是一个交错输出。当 时i=0，线程“ t1”i在display()函数的 forloop 中打印 的值。完成此操作后，将时间片处理到线程“ t2”，该线程i在display()函数的 forloop 中打印 的值。在此之后，“ t2”被换出，“ t1”被换为动作。它增加ito的值，1然后i在display()函数的 forloop 中打印 的值。完成此操作后，将时间片处理到线程“ t2”，该线程在函数的 forloop 中打印ias的值。这以交错方式继续，直到到达。1display()i5

我们需要一个有序的输出，这样第一个线程“ t1”拥有for循环并打印“ i”的所有值，然后“ t2”拥有for循环并打印“ i”的所有值。有帮助的是对象锁。检查以下程序：

爪哇 缩小▲ 复制代码

```
class processor implements Runnable{
    public void run() {
        display();
    }
    public void display() {
        //===== Object Lock Used Here ======
        synchronized(this) {
            for (int i = 0 ; i<= 5; i++) {
                System.out.println("i = " + i + " In thread" + Thread.currentThread());
            }
        }
    }
}

class locks {
    public static void main(String[] args) {
        processor p1 = new processor();
        Thread t1 = new Thread(p1, "t1");
        Thread t2 = new Thread(p1, "t2");
        t1.start();
        t2.start();
    }
}
```

```
Output
=======

i = 0 In threadThread[t1,5,main]
i = 1 In threadThread[t1,5,main]
i = 2 In threadThread[t1,5,main]
i = 3 In threadThread[t1,5,main]
i = 4 In threadThread[t1,5,main]
i = 5 In threadThread[t1,5,main]
i = 0 In threadThread[t2,5,main]
i = 1 In threadThread[t2,5,main]
i = 2 In threadThread[t2,5,main]
i = 3 In threadThread[t2,5,main]
i = 4 In threadThread[t2,5,main]
i = 5 In threadThread[t2,5,main]
```

上述程序编译并成功运行以抛出如上所示的有序输出。首先，“ t1”在 中执行完整的for循环display()。接下来，“ t2”在 中执行一个完整的for循环display()。发生这种情况是因为我们使用了对象锁。该for循环包含在使用对象锁的块中。关键字“ this”指的是当前对象，在我们的例子中是“ p1”。线程 " t1" 和 " t2" 都是从 " p1"创建的，如下所示：

爪哇 复制代码

```
Thread t1 = new Thread(p1, "t1");
Thread t2 = new Thread(p1, "t2");
```

因此，当“ t1”for首先访问循环时，它会用对象“ this”，即“ p1”锁定块，并将钥匙放在口袋里。与此同时，即使“ t1”被换掉，“ t2”也会发现门是关着的，因为物品锁的钥匙被“ t1”小心地放在口袋里。" t1" 执行完整的 forloop，然后才解锁对象的对象锁，p1 并将钥匙留在门口供任何人使用。“ t2”被换入的那一刻，它拿起钥匙，进入同步块并用对象“ this”锁定块，即“ p1”for

因此“对象锁”确保所有依赖于同一对象的线程都是同步的。

兴趣点

因此“对象锁”确保所有依赖于同一对象的线程都是同步的。

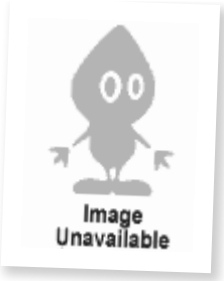
历史

- 2021 年 6 月13日：初始版本

执照

分享

关于作者



切坦·库达尔卡
软件开发人员（高级）
印度

手表
该会员

我是一名软件工程师，拥有大约 7 年以上的经验。我的大部分经验是在存储技术方面。

评论和讨论

添加评论或问题 | 电子邮件提醒 | Search Comments

第一 页上一页 下一页

一个问题
TX6430 14-Jun-21 6:09

回复：一个问题
Chetan Kudalkar 14-Jun-21 14:25

刷新 1

一般 新闻 建议 问题 错误 答案 笑话 赞美 咆哮 管理员

使用Ctrl+Left/Right 切换消息，Ctrl+Up/Down 切换主题，Ctrl+Shift+Left/Right 切换页面。