Java 中的线程：看同步

切坦·库达尔卡

评价我： 5.00/5 (1 票)

2021 年 6 月 20 日 警察

更深入地了解同步关键字的含义

本文试图解释关键字synchronized wrt object以及类级锁的含义。本文中完成的练习的目的是了解同步函数在内部使用对象级锁，而静态同步函数在其整个代码内部使用类级锁。

介绍

一些智者说过“永远不要以颜值来评判一个人。很多东西都隐藏在他们的潜意识里”。这适用于同步函数吗？让我们来看看。采取以下程序：

爪哇 缩小▲ 复制代码

```
locks.java
-----------
class processor implements Runnable{
    public void run() {
        display();
    }
    public void display() {
            synchronized (processor.lock){
            for (int i = 0 ; i<= 5; i++) {
                System.out.println("i = " + i + " In thread2" + Thread.currentThread() );
            }
        }
    }
}

class locks {
    public static void main(String[] args) {
        processor p1 = new processor();
        processor p2 = new processor();
        Thread t1 = new Thread(p1, "t1");
        Thread t2 = new Thread(p2, "t2");

        t1.start();
        t2.start();
    }
}

Output
=======
i = 0 In thread2Thread[t1,5,main]
```

```
i = 1 In thread2Thread[t1,5,main]
i = 2 In thread2Thread[t1,5,main]
i = 3 In thread2Thread[t1,5,main]
i = 4 In thread2Thread[t1,5,main]
i = 5 In thread2Thread[t1,5,main]
i = 0 In thread2Thread[t2,5,main]
i = 1 In thread2Thread[t2,5,main]
i = 2 In thread2Thread[t2,5,main]
i = 3 In thread2Thread[t2,5,main]
i = 4 In thread2Thread[t2,5,main]
i = 5 In thread2Thread[t2,5,main]
```

上述程序编译并成功运行以产生所需的同步输出。这不需要解释。线程t1和t2使用类级别锁“ processor.lock”进行同步for，以便display()函数中的循环由从处理器类的任何对象创建的一个且仅一个线程执行。

现在来看一个外观略有不同的类似程序。

爪哇 缩小▲ 复制代码

```
locks.java
----------
class processor implements Runnable{
    public void run() {
        display();
    }
    // Observe the synchronized statement below
    public static synchronized void display() {
        for (int i = 0 ; i<= 5; i++) {
            System.out.println("i = " + i + " In thread2" + Thread.currentThread() );
        }
    }
}

class locks {
    public static void main(String[] args) {
        processor p1 = new processor();
        processor p2 = new processor();
        Thread t1 = new Thread(p1, "t1");
        Thread t2 = new Thread(p2, "t2");

        t1.start();
        t2.start();
    }
}

Output
=======
i = 0 In thread2Thread[t1,5,main]
i = 1 In thread2Thread[t1,5,main]
i = 2 In thread2Thread[t1,5,main]
i = 3 In thread2Thread[t1,5,main]
i = 4 In thread2Thread[t1,5,main]
i = 5 In thread2Thread[t1,5,main]
i = 0 In thread2Thread[t2,5,main]
i = 1 In thread2Thread[t2,5,main]
i = 2 In thread2Thread[t2,5,main]
i = 3 In thread2Thread[t2,5,main]
i = 4 In thread2Thread[t2,5,main]
i = 5 In thread2Thread[t2,5,main]
```

上面的程序编译并成功运行以产生与早期程序完全相似的所需输出。好吧，这里唯一的变化是我们将 display 声明为静态同步函数()。我们得到了与之前程序完全相似的输出。

爪哇 缩小▲ 复制代码

```
public static synchronized void display() {
    ....
    ....
```

```
    ....
}

EXACTLY EQUALS

public void display() {
    synchronized(process.lock) {
        ....
        ....
        ....
        ....
    }
}

And NOT EQUAL TO

public void display() {
    ......
    ......
    ......
    synchronized(process.lock){
        ....
        ....
        ....
    }
}
```

因此，当您看到一个声明为`static`同步的函数时，这意味着其中的代码受到类级锁的保护。

现在让我们继续讨论`object`锁。采取以下程序：

爪哇 缩小▲ 复制代码

```
locks.java
-----------
class processor implements Runnable{
    public void run() {
        display();
    }
    public void display() {
            synchronized (this){
            for (int i = 0 ; i<= 5; i++) {
                System.out.println("i = " + i + " In thread2" + Thread.currentThread() );
            }
        }
    }
}

class locks {
    public static void main(String[] args) {
        processor p1 = new processor();
        processor p2 = new processor();
        Thread t1 = new Thread(p1, "t1");
        Thread t2 = new Thread(p1, "t2");

        t1.start();
        t2.start();

    }
}

Output
=======
i = 0 In thread2Thread[t1,5,main]
i = 1 In thread2Thread[t1,5,main]
i = 2 In thread2Thread[t1,5,main]
i = 3 In thread2Thread[t1,5,main]
i = 4 In thread2Thread[t1,5,main]
i = 5 In thread2Thread[t1,5,main]
```

```
i = 0 In thread2Thread[t2,5,main]
i = 1 In thread2Thread[t2,5,main]
i = 2 In thread2Thread[t2,5,main]
i = 3 In thread2Thread[t2,5,main]
i = 4 In thread2Thread[t2,5,main]
i = 5 In thread2Thread[t2,5,main]
```

上述程序编译并成功运行以产生所需的同步输出。这不需要解释。线程`t1`和`t2`使用对象级锁“`this`”进行同步`for`，以便`display()`函数中的循环由从处理器类的一个且仅一个对象中创建的一个且仅一个线程执行。

现在来看一个外观略有不同的类似程序。

爪哇 缩小▲ 复制代码

```
locks.java
----------
class processor implements Runnable{
    public void run() {
        display();
    }
    // Observe the synchronized statement below
    public synchronized void display() {
        for (int i = 0 ; i<= 5; i++) {
            System.out.println("i = " + i + " In thread2" + Thread.currentThread() );
        }
    }
}

class locks {
    public static void main(String[] args) {
        processor p1 = new processor();
        processor p2 = new processor();
        Thread t1 = new Thread(p1, "t1");
        Thread t2 = new Thread(p1, "t2");

        t1.start();
        t2.start();
    }
}

Output
=======
i = 0 In thread2Thread[t1,5,main]
i = 1 In thread2Thread[t1,5,main]
i = 2 In thread2Thread[t1,5,main]
i = 3 In thread2Thread[t1,5,main]
i = 4 In thread2Thread[t1,5,main]
i = 5 In thread2Thread[t1,5,main]
i = 0 In thread2Thread[t2,5,main]
i = 1 In thread2Thread[t2,5,main]
i = 2 In thread2Thread[t2,5,main]
i = 3 In thread2Thread[t2,5,main]
i = 4 In thread2Thread[t2,5,main]
i = 5 In thread2Thread[t2,5,main]
```

上面的程序编译并成功运行以产生与早期程序完全相似的所需输出。好吧，这里唯一的变化是我们将 display 声明为一个同步函数（）。我们得到了与之前程序完全相似的输出。

爪哇 缩小▲ 复制代码

```
public synchronized void display() {
    ....
    ....
    ....
}

EXACTLY EQUALS
```

```
public void display() {
    synchronized(this) {
        ....
        ....
        ....
        ....
    }
}

And NOT EQUAL TO

public void display() {
    ......
    ......
    ......
    synchronized(this){
        ....
        ....
        ....
    }
}
```

因此，当您看到一个声明为同步的函数时，这意味着其中的代码受到对象级锁的保护。

本练习的目的是了解同步函数在内部使用对象级锁，而静态同步函数在其整个代码内部使用类级锁。好吧，Java 中的关键字 synchronized 的潜意识背后确实隐藏着很多东西。说啥？

兴趣点

应用于方法时的静态同步是指类级别的锁，而应用于方法时的同步是指对象级别的锁。

读者们，如果您觉得这篇文章对您有帮助，请随时发表评论。这让我明白是否需要添加更多内容作为解释的一部分。非常欢迎您的反馈。谢谢！

历史

- 2021 年 6 月19日：初步修订

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

-- 本论坛暂无消息 --