线程同步--synchronized

笔记本：	java-多线程		
创建时间：	2019/5/3 17:49	**更新时间：**	2019/5/3 19:26
作者：	ANNER		
URL：	https://www.jianshu.com/p/ea9a482ece5f		

由于同一进程的多个线程共享同一片存储空间，在带来方便的同时，也带来了访问冲突这个严重的问题。Java语言提供了专门机制以解决这种冲突，有效避免了同一个数据对象被多个线程同时访问。

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码块的对象；
2. **修饰一个方法**，被修饰的方法称为同步方法，其作用的范围是整个方法，作用的对象是调用这个方法的对象；
3. 修饰一个静态的方法，其作用的范围是整个静态方法，作用的对象是**这个类的所有对象**；
4. 修饰一个类，其作用的范围是synchronized后面括号括起来的部分，作用的对象是这个类的所有对象

总结：

- 每个对象只有一个锁（lock）与之相关联
- 无论synchronized关键字加在方法上还是对象上，它取得的锁都是对象，而不是把一段代码或函数当作锁一而且**同步方法很可能还会被其他线程的对象访问**

修饰类中的普通方法

下面的示例程序获取的修饰普通方法中的锁，且使用的是同一个对象。

```
public class NormalThread implements Runnable {
    private int num;
    private Trans trans;
    public  NormalThread(int num,Trans trans){
        this.num=num;
        this.trans=trans;
    }
    public void run() {
        while (true){
            this.trans.printNum(num);
            try {
                Thread.sleep( millis: 500);
            }catch (InterruptedException e){
                e.printStackTrace();
            }
        }
    }

    public static void main(String[] args) {
        Trans trans=new Trans();
        Thread t1=new Thread(new NormalThread( num: 1,trans));
        Thread t2=new Thread(new NormalThread( num: 2,trans));
        t1.start();
        t2.start();
```

synchronized关键字会获得锁，所以在这里synchronized关键字获得的锁就是传递给线程的那个Trans对象的锁，两个线程用的是同一个锁，所以当一个线程执行到synchronized修饰的方法时，这个线程就会获得Trans对象的锁，并且这个锁是互斥的，所以其他试图想要获得锁的线程都必须等待，直到这个线程释放了锁，从而起到了互斥的作用。

```
    public static void main(String[] args) {
        Trans trans=new Trans();
        Thread t1=new Thread(new NormalThread(1,trans));
        Thread t2=new Thread(new NormalThread(2,trans));
        t1.start();
        t2.start();
      //测试不同的对象
      Trans trans1=new Trans();
      Trans trans2=new Trans();
      Thread t1=new Thread(new NormalThread( num: 1,trans1));
      Thread t2=new Thread(new NormalThread( num: 2,trans2));
      t1.start();
      t2.start();

    }
```

既然每个对象都只有一把互斥锁，那么同一个对象的多个synchronized关键字修饰的方法之间也是无法同时访问的。例如下面的例子：

```java
public class DoubleTrans {
    //同步方法，无限循环，理论上其他的同步方法永远不会被执行
    public synchronized void printNum1(int num){
        while (true){
            System.out.print("同步方法1..");
            for(int i=0;i<25;i++){
                System.out.print(i+" ");
            }
            System.out.println();
            try{
                Thread.sleep( millis: 500);
            }catch (InterruptedException e){
                e.printStackTrace();
            }
        }
    }
    //同步方法
    public synchronized void printNum2(int num){
        System.out.print("同步方法2..");
        for(int i=0;i<25;i++){
            System.out.print(i+" ");
        }
    }
    //普通方法
    public void printNum3(int num){
        System.out.print("普通方法...");
        for(int i=0;i<25;i++)System.out.print(i+" ");
    }
}
```

方法一和方法三交替执行，方法二不会执行

类中静态方法添加线程锁

每个类都有一个互斥锁，每个类可能会有多个不同的对象，每个对象都有一个它自己的对象锁。自然，类的锁的管辖范围比对象的锁的管辖范围大，不同的对象之间的锁互不影响，但是他们都受类的锁的控制，**如果一个类的锁被一个线程获得，那么这个类的所有的对象都不能访问需要获得类的锁的方法，但是可以访问不需要锁的方法和需要某个对象锁的方法**

下面这个例子即为同一个类中全部的对象均无法访问静态加锁的方法

```java
public class StaticTrans {
    //加锁的静态方法，类中所有的对象均无法同时访问
    public static synchronized void printNum1(int num){
        System.out.println("静态加锁方法:");
        for(int i=0;i<25;i++)System.out.print(i+" ");
        System.out.println();
    }
    //普通方法加锁方法
    public synchronized void printNum2(int num){
        System.out.println("普通加锁方法...");
        for(int i=0;i<25;i++)System.out.print(i+" ");
        System.out.println();
    }
}
```

修饰代码块

- synchronized（this）这里的this指的是执行这段代码的对象，synchronized得到的锁就是this这个对象的锁
- synchronized（A.class）这里A.class得到的是A这类，所以synchronized关键字得到的锁是类的锁
- synchronized（object）这里synchronized关键字拿到的锁是对象object的锁，所有需要这个对象的锁的方法都不能同时执行。

最后例子，获取一个object对象的锁

```
public class BlockTrans {
    private Object lock=new Object();
    public void printNum(int num){
        synchronized (lock){
            System.out.println(lock.toString());
            for(int i=0;i<25;i++)System.out.print(i+" ");
            System.out.println();
        }
    }
}
```

最后总结：

synchronized锁住的是一个对象或者类（其实也是对象），而不是方法或者代码段。