个人资料

王炎林

访问： 105916次
积分： 1670
等级：
排名： 第17680名

原创： 54篇 转载： 1篇
译文： 0篇 评论： 26条

synchronized锁住了什么

标签： sychronized java

2015-03-17 11:12 4524人阅读 评论(0) 收藏 举报

分类： Java（29）

先看一个简单示例，下面这段代码能够正常通过编译。

```
public class SyncTest {

    public SyncTest syncVar ,

    public static SyncTest syncStaticVar;

    public static synchronized void testStaticSync() {

关闭
    }

    blic synchronized void testNonStaticSync() {



    blic void testSyncThis() {
        synchronized (this ) {
            try {
                System. out.println("test sync this start" );
                Thread. sleep(5000);
                System. out.println("test sync this end" );
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    public void testSyncVar() {
        synchronized (syncVar ) {
            try {
                System. out.println("test sync var start" );
                Thread. sleep(3000);
                System. out.println("test sync var end" );
```

```
            } catch (InterruptedException e) {

            }
        }
    }

    public void testStaticSyncVar() {
        synchronized (syncStaticVar ) {

        }
    }

    public static void main(String[] args) {
        final SyncTest testSync = new SyncTest();
        testSync. syncVar = new SyncTest();
        testSync. syncVar = testSync;
        Thread threadOne = new Thread(new Runnable() {
            @Override
            public void run() {
                testSync.testSyncThis();
            }
        });
        Thread threadTwo = new Thread(new Runnable() {
            @Override
            public void run() {
                testSync.testSyncVar();
            }
        });
        threadOne.start();
        threadTwo.start();
    }
}
```

从上面的代码来看，synchronized使用的场景很广，既能够锁住类方法（static），又能够锁住对象方法（非static）。既能够对某个属性加锁，又能够对this加锁。那么，sychronized到底锁住了什么？另外，synchronized也是可重入的，那么，它与单独的ReentrantLock的区别是什么？

sychronized在使用的时候也是区分目标对象的，在这一点上其实非常像前面提到过的ReentrantLock的使用。上面的代码中，如果把testSync. syncVar = testSync;这句话注释掉，那么最后运行所产生的结果就会不一样。如果要和ReentrantLock做对比的话，可以把synchronized所针对的目标对象理解成一个锁，针对的目标对象不一样，那么就是不同的锁。

sychronized的实现原理需要深入到jvm中才能找到答案。。可以参考《深入理解Java虚拟机》

在这里主要讨论一个使用上的问题，当我们使用sychronized锁住某个对象时，我们锁住的是这个引用本身，还是内存（堆）中的这个对象本身。对这个问题的一个延伸是，当我们在sychronized作用区域内，为这个引用附一个新值的时候，sychronized是否还有效？

先给出结论，sychronized锁住的是内存（堆）中的对象，当引用被附上新值的时候，则相当于旧对象的锁被释放。这里不做理论讨论，只是用程序进行验证。

用三个例子进行说明，

示例1、

```
public class TestSyncAndModify implements Runnable {

    private A syncA;

    @Override
    public void run() {
        synchronized (syncA ) {
            System. out .println(Thread.currentThread().getName());
                syncA = new A();
            try {
                Thread. sleep(3000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System. out .println(Thread.currentThread().getName());
        }
    }

    static class A {
    };

    public static void main(String[] args) {
        TestSyncAndModify sync = new TestSyncAndModify();
        A testA = new A();
        sync. syncA = testA;
        Thread one = new Thread(sync);
        Thread two = new Thread(sync);
        one.start();
        try {
            Thread. sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        two.start();
    }
}
```

在sychronized作用域内对syncA做了修改，使它指向了一个新的对象，所以当这句话执行完之后，第二个线程就可以运行，因此输出的结果如下

```
Thread-0
Thread-1
Thread-0
Thread-1
```

示例2、就是最上面的那段代码SyncTest。在main函数内，有这样一句赋值testSync. syncVar = testSync;使得syncVar成员变量，指向了和this相同的区域。因此，在sychronized(this)和synchronized(syncVar)就形成了竞争，使得后者被阻塞，因此输出结果如下

```
test sync this start
test sync this end
test sync var start
```

```
test sync var end
```

示例3

```
public class ThreadUseBase extends Thread {

    private Base baseObject ;

    public ThreadUseBase(Base boj) {
        baseObject = boj;
    }

    public void testSyncBase() {
        synchronized (baseObject ) {
            System. out .println("enter thread use base" );
            try {
                Thread. sleep(2000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System. out .println("leave thread use base" );
        }
    }

    @Override
    public void run() {
        testSyncBase();
    }

    static class ThreadUseChild extends Thread {
        private SyncObject childObj ;

        public ThreadUseChild(SyncObject sobj) {
            childObj = sobj;
        }

        public void testSyncChild() {
            synchronized (childObj ) {
                System. out .println("enter thread use child" );
                try {
                    Thread. sleep(2000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                System. out .println("leave thread use child" );
            }
        }

        @Override
        public void run() {
            testSyncChild();
        }
    }

    static class Base {
```

```
    }

    static class SyncObject extends Base {

    }

    public static void main(String[] args) {
        SyncObject childObj = new SyncObject();
        Base baseObj = childObj;
        //Base baseObj = new Base();
        ThreadUseBase threadBase = new ThreadUseBase(baseObj);
        ThreadUseChild threadChild = new ThreadUseChild(childObj);
        threadBase.start();
        threadChild.start();
    }

}
```

虽然是两个线程中，sychronized锁住的是不同的引用，一个是Base一个是SyncObject，但由于存在继承关系，在main函数中，我们让Base对象也指向了SyncObject内存区域。这样，就形成了两个不同线程之间的竞争，因此后者被阻塞，输出结果如下

```
enter thread use base
leave thread use base
enter thread use child
leave thread use child
```

参考论文如下

JVM的锁结构设计原理 --- https://www.usenix.org/legacy/event/jvm01/full_papers/dice/dice.pdf

上一篇 Executor-ThreadPoolExecutor实现
下一篇 ZooKeeper简要笔记

我的同类文章

Java（29）

- Hadoop RPC分析（二）-... 2016-07-25 阅读 475
- Hadoop RPC分析（一）--... 2016-07-25 阅读 114
- ThreadLocal的原理 2016-03-05 阅读 359
- 一个ClassCastException... 2015-11-07 阅读 394
- Session和Cookie 2015-10-27 阅读 384
- 求解！关于Http和Socket 2015-09-02 阅读 295
- tomcat服务器解析（七）-- ... 2015-06-17 阅读 1139
- Java Reference 2015-06-11 阅读 547
- tomcat服务器解析（六）-- ... 2015-06-07 阅读 1265
- tomcat服务器解析（五）-- ... 2015-06-05 阅读 1723
- tomcat服务启动解析 2015-05-14 阅读 1294

更多文章

参考知识库