登录 | 注册

Lynn 生活与技术,缺一不可

:■ 目录视图 늘 摘要视图 RSS 订阅 个人资料 深度学习代码专栏 攒课--我的学习我做主 开启你的知识管理,知识库个人图谱上线 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) 🐷 版权声明:本文为博主原创文章,未经博主允许不得转载。 ▣ 访问: 105916次 积分: 1670 先看一个简单示例,下面这段代码能够正常通过编译。 等级: BLOG > 4 排名: 第17680名 public class SyncTest { 原创: 54篇 转载: 1篇 译文: 0篇 评论: 26条 public SyncTest syncrum , 文章搜索 public static SyncTest syncStaticVar; public static synchronized void testStaticSync() { 关闭 blic synchronized void testNonStaticSync() { blic void testSyncThis() { synchronized (this) { try { Him 32 00 System. out.println("test sync this start"); Thread. sleep(5000); 文章存档 System. out.println("test sync this end"); 2016年07月 (2) } catch (InterruptedException e) { 2016年04月 (1) e.printStackTrace(); 2016年03月 (1) 2015年11月 (1) } 2015年10月 (1) } 展开 public void testSyncVar() { 阅读排行 synchronized (syncVar) { ReentrantLock解析 (9982)Netty的ChannelPipeline (6997) System. out.println("test sync var start"); Netty的ChannelFuture Thread. sleep(3000); synchronized锁住了什么 (4523) System. out.println("test sync var end"); Netty详解 (4196)

```
      Struts2缓存解析
      (4145)

      Linux信号系统详解
      (4063)

      Struts2的loC解析
      (3653)

      http请求的参数和属性
      (3152)

      共享锁和排它锁(Reent (3067)
```

评论排行	
Linux高速缓存概述	(4)
Linux中断详解	(3)
C++虚继承小结	(3)
i节点	(2)
tomcat服务器解析(一)	(2)
Executor实现Abstract	(2)
Linux高速缓存详解(二)	(2)
Struts2的Builder模式解析	(2)
ReentrantLock解析	(1)
Struts2缓存解析	(1)

推荐文章

- * 2016 年最受欢迎的编程语言是什么?
- * Chromium扩展(Extension)的页面(Page)加载过程分析
- * Android Studio 2.2 来啦
- * 手把手教你做音乐播放器(二)技术原理与框架设计
- * JVM 性能调优实战之:使用阿里开源工具 TProfiler 在海量业务代码中精确定位性能代码

最新评论

C++虚继承小结

yujie1993: @liujiayu2:最初的问题就是,如果FinalClass不是虚继承自MakeFinal,那么T...

Executor实现----AbstractExecuto 王炎林: @qq_35681566:对。不 县昌新的

Executor实现----AbstractExecuto qq_35681566: 这是1.6的源码分 长四?

C++虚继承小结

王炎林: @liujiayu2:不好意思,已 经很长时间没有接触C++了。这 些内容我现在也不熟悉了。

C++虚继承小结

H-KING: 完整看完你的博客,还是 没有理解你最初那个问题该怎么 解释呀?

tomcat服务器解析 (一)

王炎林: @renzhengzhi:对apr这部分不太了解。。力有不逮,不好意思啊

Linux高速缓存概述

王炎林: @lipeng_bigdata:高手不敢当。毕业后就没接触linux 了,当时只是兴趣所致,去了解

ReentrantLock解析

辰辰爸的技术博客: 很好, 赞!

Linux高速缓存概述

辰辰爸的技术博客: 高手啊,多谢 分享,希望能持续推出,辛苦!

共享锁和排它锁(ReentrantRea 辰辰爸的技术博客: 不错,顶!

```
} 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)

・ nadoop κρር が何( □)・... 2010-07-25 阅读 4/5 ・ nadoop κρር が何( □)・... 2010-07-25 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
```

参考知识库



Java EE知识库

6601 关注 | 687 收录



Java SE知识库

13891 关注 | 457 收录



Java 知识库

15542 关注 | 1288 收录

猜你在找

javaScript-高级面向对象视频教程

Java基础核心技术:面向对象编程(day05-day07)

Java日志实战及解析

Android开发精品课程【Java核心知识】 java语言从入门到精通2016+项目实训

线程同步

线程处理的基本问题

A Look Inside JBoss Cache

通过DBCC Page查看在SQL Server中哪行数据被锁住了

找出表被谁锁住了

查看评论

暂无评论

您还没有登录,请[登录]或[注册]

以上用户言论只代表其个人观点,不代表CSDN网站的观点或立场

核心技术类目

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker OpenStack VPN Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC WAP jQuery BI HTML5 Spring Apache .NET API HTML SDK IIS Fedora XML LBS Unity Splashtop UML components Windows Mobile Rails QEMU KDE Cassandra CloudStack FTC coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App SpringSide Maemo Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP HBase Pure Solr Angular Cloud Foundry Redis Scala Django Bootstrap

公司简介 | 招贤纳士 | 广告服务 | 银行汇款帐号 | 联系方式 | 版权声明 | 法律顾问 | 问题报告 | 合作伙伴 | 论坛反馈

400-600-2320 | 北京创新乐知信息技术有限公司 版权所有 | 江苏乐知网络技术有限公司 提供商务支持 网站客服 杂志客服 微博客服 webmaster@csdn.net

京 ICP 证 09002463 号 | Copyright © 1999-2016, CSDN.NET, All Rights Reserved

