

- [首页](#)
- [所有文章](#)
- [资讯](#)
- [Web](#)
- [架构](#)
- [基础技术](#)
- [书籍](#)
- [教程](#)
- [Java小组](#)
- [工具资源](#)

- 导航条 - ▾

Java 8新特性探究（十）StampedLock将是解决同步问题的新宠

2015/02/05 | 分类: [基础技术](#) | [1 条评论](#) | 标签: [StampedLock](#)

分享到:

6 原文出处: [成熟的毛虫的博客](#)

Java8就像一个宝藏，一个小的API改进，也足与写一篇文章，比如同步，一直是多线程并发编程的一个老话题，相信没有人喜欢同步的代码，这会降低应用的吞吐量等性能指标，最坏的时候会挂起死机，但是即使这样你也没得选择，因为要保证信息的正确性。所以本文决定将从synchronized、Lock到Java8新增的StampedLock进行对比分析，相信StampedLock不会让大家失望。

synchronized

在java5之前，实现同步主要是使用synchronized。它是Java语言的关键字，当它用来修饰一个方法或者一个代码块的时候，能够保证在同一时刻最多只有一个线程执行该段代码。

有四种不同的同步块：

1. 实例方法
2. 静态方法
3. 实例方法中的同步块
4. 静态方法中的同步块

大家对此应该不陌生，所以不多讲了，以下是代码示例

```
1 synchronized(this)
2 // do operation
3 }
```

小结：在多线程并发编程中Synchronized一直是元老级角色，很多人都会称呼它为重量级锁，但是随着Java SE1.6对Synchronized进行了各种优化之后，性能上也有所提升。

Lock

```
1 rwlock.writeLock().lock();
2 try {
3 // do operation
4 } finally {
```

```

5 |  rwlock.writeLock().unlock();
6 |  }

```

它是Java 5在java.util.concurrent.locks新增的一个API。

Lock是一个接口，核心方法是lock(), unlock(), tryLock(), 实现类有ReentrantLock, ReentrantReadWriteLock.ReadLock, ReentrantReadWriteLock.WriteLock;

ReentrantReadWriteLock, ReentrantLock 和synchronized锁都有相同的内存语义。

与synchronized不同的是，Lock完全用Java写成，在java这个层面是无关JVM实现的。Lock提供更灵活的锁机制，很多synchronized 没有提供的许多特性，比如锁投票，定时锁等候和中断锁等候，但因为lock是通过代码实现的，要保证锁定一定会被释放，就必须将unlock()放到finally{}中

下面是Lock的一个代码示例

```

1 | class Point {
2 |     private double x, y;
3 |     private final StampedLock sl = new StampedLock();
4 |     void move(double deltaX, double deltaY) { // an exclusively locked method
5 |         long stamp = sl.writeLock();
6 |         try {
7 |             x += deltaX;
8 |             y += deltaY;
9 |         } finally {
10 |             sl.unlockWrite(stamp);
11 |         }
12 |     }
13 |     //下面看看乐观读锁案例
14 |     double distanceFromOrigin() { // A read-only method
15 |         long stamp = sl.tryOptimisticRead(); //获得一个乐观读锁
16 |         double currentX = x, currentY = y; //将两个字段读入本地局部变量
17 |         if (!sl.validate(stamp)) { //检查发出乐观读锁后同时是否有其他写锁发生?
18 |             stamp = sl.readLock(); //如果没有，我们再次获得一个读悲观锁
19 |             try {
20 |                 currentX = x; // 将两个字段读入本地局部变量
21 |                 currentY = y; // 将两个字段读入本地局部变量
22 |             } finally {
23 |                 sl.unlockRead(stamp);
24 |             }
25 |         }
26 |         return Math.sqrt(currentX * currentX + currentY * currentY);
27 |     }
28 |     //下面是悲观读锁案例
29 |     void moveIfAtOrigin(double newX, double newY) { // upgrade
30 |         // Could instead start with optimistic, not read mode
31 |         long stamp = sl.readLock();
32 |         try {
33 |             while (x == 0.0 && y == 0.0) { //循环，检查当前状态是否符合
34 |                 long ws = sl.tryConvertToWriteLock(stamp); //将读锁转为写锁
35 |                 if (ws != 0L) { //这是确认转为写锁是否成功
36 |                     stamp = ws; //如果成功 替换票据
37 |                     x = newX; //进行状态改变
38 |                     y = newY; //进行状态改变
39 |                     break;
40 |                 }
41 |                 else { //如果不能成功转换为写锁
42 |                     sl.unlockRead(stamp); //我们显式释放读锁
43 |                     stamp = sl.writeLock(); //显式直接进行写锁 然后再通过循环再试
44 |                 }
45 |             }
46 |         } finally {
47 |             sl.unlock(stamp); //释放读锁或写锁
48 |         }
49 |     }
50 | }

```

小结：比synchronized更灵活、更具可伸缩性的锁定机制，但不管怎么说还是synchronized代码要更容易书写些

StampedLock

它是java8在java.util.concurrent.locks新增的一个API。

ReentrantReadWriteLock 在没有任何读写锁时，才可以取得写入锁，这可用于实现了悲观读取（Pessimistic Reading），即如果执行中进行读取时，经常可能有另一执行要写入的需求，为了保持同步，ReentrantReadWriteLock 的读取锁定就可派上用场。

然而，如果读取执行情况很多，写入很少的情况下，使用 ReentrantReadWriteLock 可能会使写入线程遭遇饥饿（Starvation）问题，也就是写入线程吃无法竞争到锁定而一直处于等待状态。

StampedLock控制锁有三种模式（写，读，乐观读），一个StampedLock状态是由版本和模式两个部分组成，锁获取方法返回一个数字作为票据stamp，它用相应的锁状态表示并控制访问，数字0表示没有写锁被授权访问。在读锁上分为悲观锁和乐观锁。

所谓的乐观读模式，也就是若读的操作很多，写的操作很少的情况下，你可以乐观地认为，写入与读取同时发生几率很少，因此不悲观地使用完全的读取锁定，程序可以查看读取资料之后，是否遭到写入执行的变更，再采取后续的措施（重新读取变更信息，或者抛出异常），这一个小小改进，可大幅度提高程序的吞吐量！！

下面是java doc提供的StampedLock一个例子

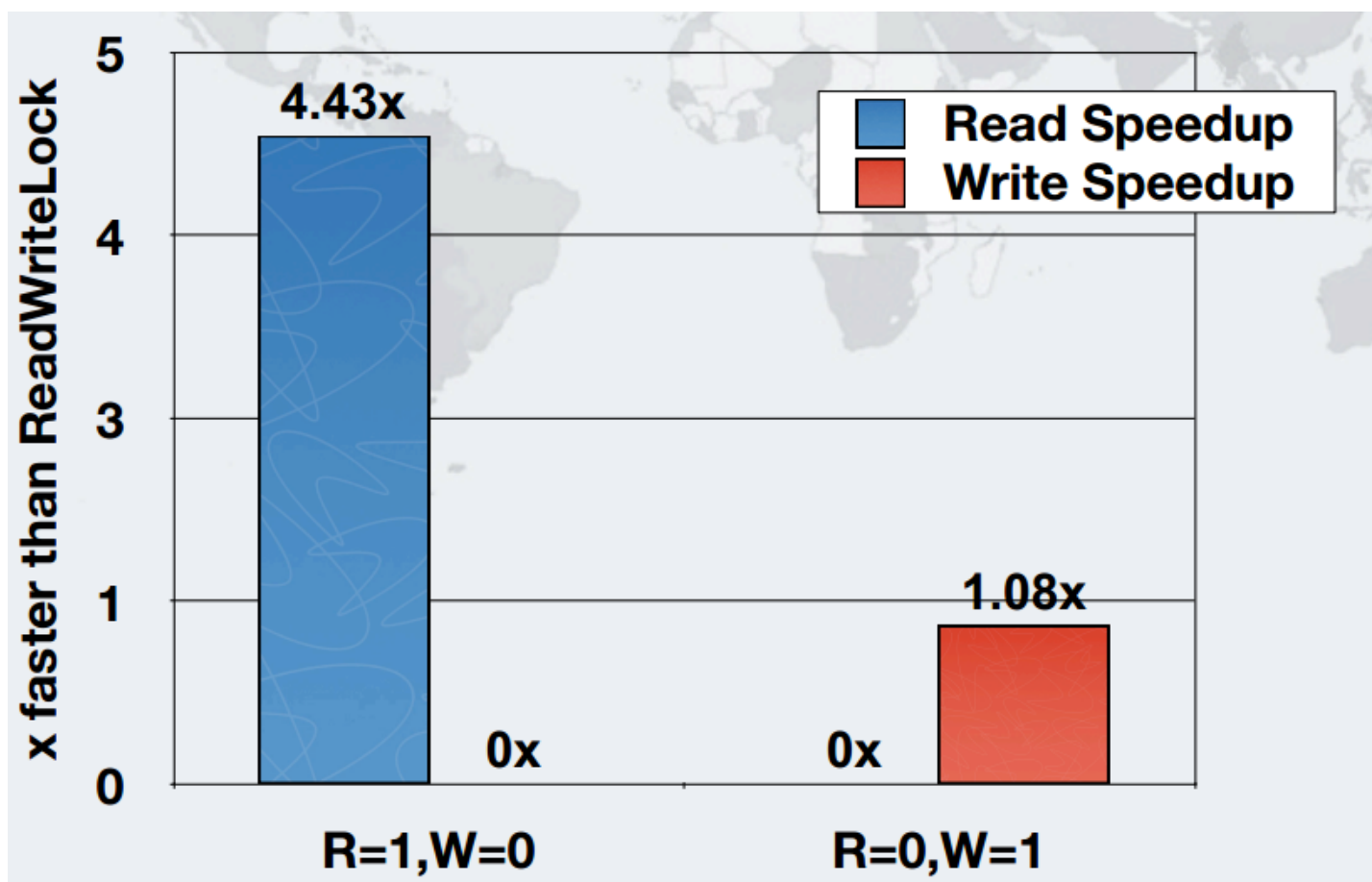
```
1  class Point {
2      private double x, y;
3      private final StampedLock sl = new StampedLock();
4      void move(double deltaX, double deltaY) { // an exclusively locked method
5          long stamp = sl.writeLock();
6          try {
7              x += deltaX;
8              y += deltaY;
9          } finally {
10             sl.unlockWrite(stamp);
11         }
12     }
13     //下面看看乐观读锁案例
14     double distanceFromOrigin() { // A read-only method
15         long stamp = sl.tryOptimisticRead(); //获得一个乐观读锁
16         double currentX = x, currentY = y; //将两个字段读入本地局部变量
17         if (!sl.validate(stamp)) { //检查发出乐观读锁后同时是否有其他写锁发生?
18             stamp = sl.readLock(); //如果没有，我们再次获得一个读悲观锁
19             try {
20                 currentX = x; // 将两个字段读入本地局部变量
21                 currentY = y; // 将两个字段读入本地局部变量
22             } finally {
23                 sl.unlockRead(stamp);
24             }
25         }
26         return Math.sqrt(currentX * currentX + currentY * currentY);
27     }
28     //下面是悲观读锁案例
29     void moveIfAtOrigin(double newX, double newY) { // upgrade
30         // Could instead start with optimistic, not read mode
31         long stamp = sl.readLock();
32         try {
33             while (x == 0.0 && y == 0.0) { //循环，检查当前状态是否符合
34                 long ws = sl.tryConvertToWriteLock(stamp); //将读锁转为写锁
35                 if (ws != 0L) { //这是确认转为写锁是否成功
36                     stamp = ws; //如果成功 替换票据
37                     x = newX; //进行状态改变
38                     y = newY; //进行状态改变
39                     break;
40                 }
41                 else { //如果不能成功转换为写锁
42                     sl.unlockRead(stamp); //我们显式释放读锁
43                     stamp = sl.writeLock(); //显式直接进行写锁 然后再通过循环再试
44                 }
45             }
46         } finally {
47             sl.unlock(stamp); //释放读锁或写锁
48         }
49     }
50 }
```

小结：

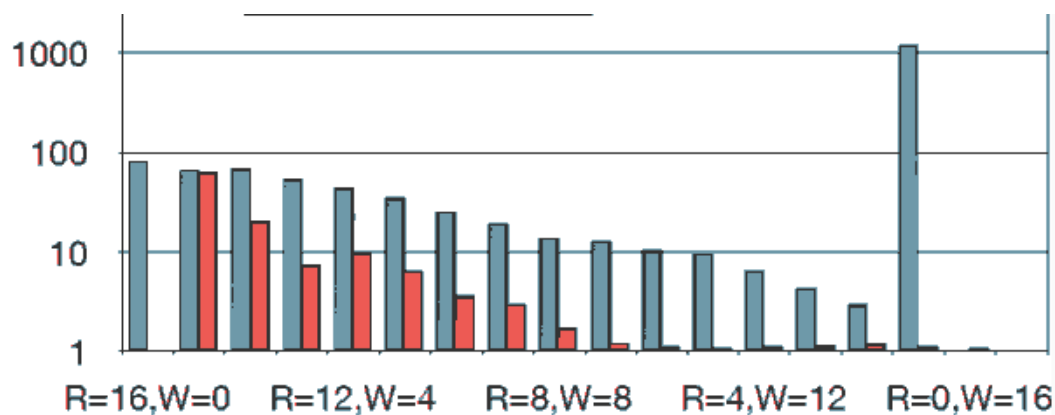
StampedLock要比ReentrantReadWriteLock更加廉价，也就是消耗比较小。

StampedLock与ReadWriteLock性能对比

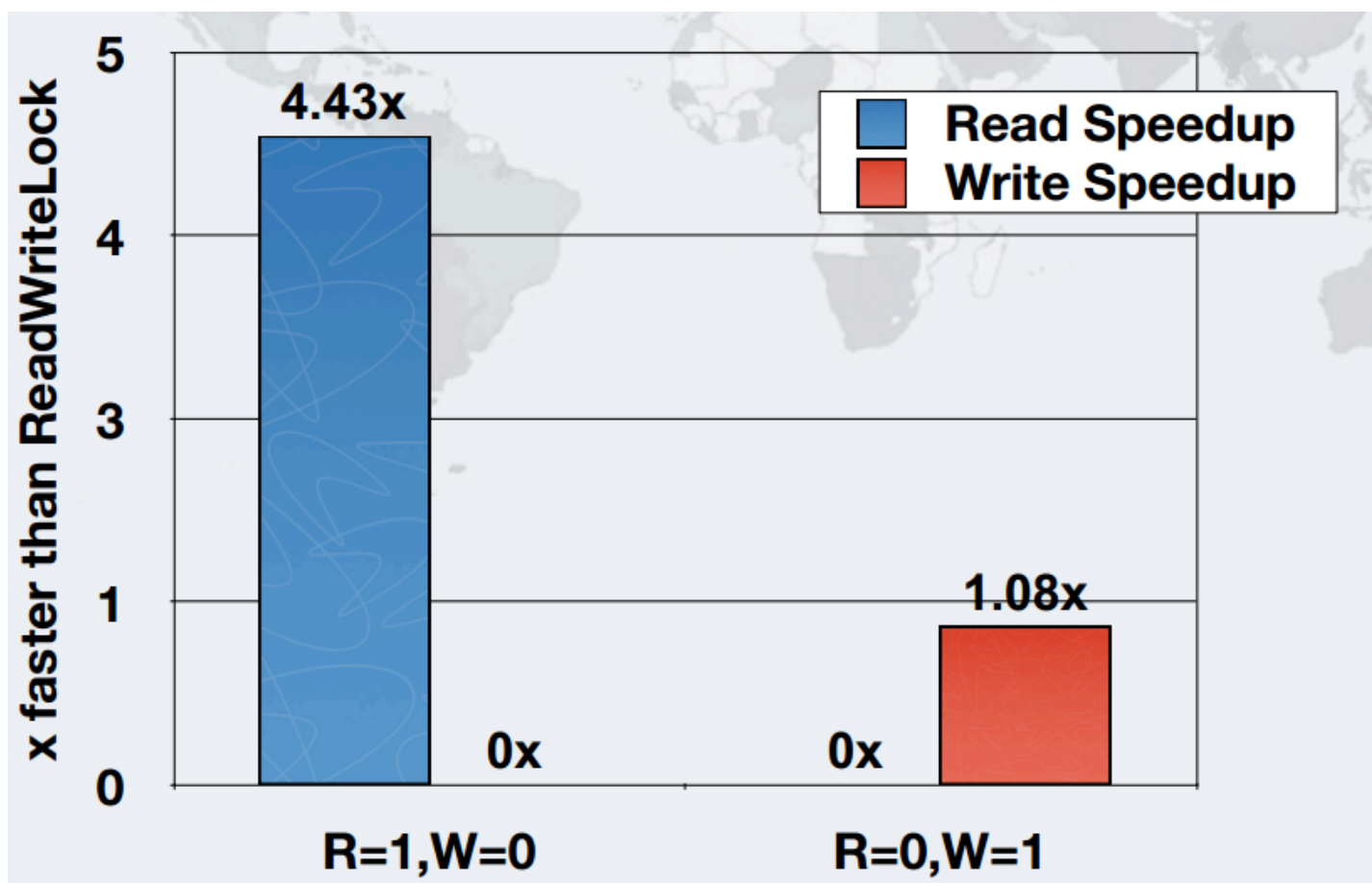
下图是和ReadWritLock相比，在一个线程情况下，是读速度其4倍左右，写是1倍。



下图是六个线程情况下，读性能是其几十倍，写性能也是几十倍左右：



下图是吞吐量提高：



总结



1. synchronized是在JVM层面上实现的，不但可以通过一些监控工具监控synchronized的锁定，而且在代码执行时出现异常，JVM会自动释放锁定；
2. ReentrantLock、ReentrantReadWriteLock、StampedLock都是对象层面的锁定，要保证锁定一定会被释放，就必须将unlock()放到finally{}中；
3. StampedLock 对吞吐量有巨大的改进，特别是在读线程越来越多的场景下；
4. StampedLock有一个复杂的API，对于加锁操作，很容易误用其他方法；
5. 当只有少量竞争者的时候，synchronized是一个很好的通用的锁实现；
6. 当线程增长能够预估，ReentrantLock是一个很好的通用的锁实现；

StampedLock 可以说是Lock的一个很好的补充，吞吐量以及性能上的提升足以打动很多人了，但并不是说要替代之前Lock的东西，毕竟他还是有些应用场景的，起码API比StampedLock容易入手，下篇博文争取更新快一点，可能会是Nashorn的内容，这里允许我先卖个关子。。。



相关文章

- [Java 8: StampedLock、ReadWriteLock以及synchronized的比较](#)
- [Java 8 新特性](#)
- [Facebook推出Android构建工具——Buck](#)
- [可以重写静态方法吗？](#)

- [Java垃圾回收精粹 — Part2](#)
- [Eclipse Mars 正式版发布, 列数 10 大特点](#)
- [Spring+SpringMVC+Maven+Mybatis+MySQL项目搭建](#)
- [Spring MVC REST异常处理最佳实践 \(下\)](#)
- [Java Selenium \(十四\) 处理 Iframe 中的元素](#)
- [Jetty 基本介绍](#)

发表评论

Comment form

Name*

姓名

邮箱*


请填写邮箱

网站 (请以 http://开头)

请填写网站地址

评论内容*

请填写评论内容



(*) 表示必填项

[提交评论](#)

1 条评论

1. *qalong* 说道:

[2015/02/05 下午 2:39](#)

ReentrantLock的示例代码贴错误了

 1  0

[回复](#)

[« Java 8新特性探究 \(九\) 跟OOM: Permgen说再见吧
一个Java对象到底占多大内存? »](#)

Search for:

Search

Search



- [本周热门文章](#)
- [本月热门](#)
- [热门标签](#)

0 [并发一枝花之 ConcurrentLinkedQue...](#)

1 [Java 多线程知识小抄集（一）](#)

2 [Java 多线程知识小抄集（二）](#)

3 [Java 多线程知识小抄集（三）](#)

4 [Spring Boot & Spring MVC...](#)

5 [千亿 KV 数据存储和查询方案](#)

6 [Java 消息队列任务的平滑关闭](#)

7 [就是让你懂 Spring 中 Mybatis...](#)

8 [电子凭证：Java 生成 Pdf](#)

9 [面试中单例模式有几种写法](#)



最新评论

- 
Re: [Java 高并发综合](#)
写的真好，点赞写的真好，点赞 dsafds
- 
Re: [Java 高并发综合](#)
Hi，请到伯乐在线的小组发帖提问，支持微信登录。链接是： <http://group.jobbole....> 唐尤华
- 
Re: [ArrayList 初始化 — J...](#)
讲的很透彻，尤其是内存空间分配情况 洋
- 
Re: [Java 高并发综合](#)
老看到资料说ArrayList 不是线程安全的，请问能举个例子吗~ 翼枪
- 
Re: [我的编码习惯 — 异常处理](#)

同意，这对于严谨的数据处理系统来说极为必要，除了方法内部能解决的错误其他异常都尽量抛出。如果一个方法... kppom



Re: [java中文乱码解决之道 \(2\) : 字符...](#)

这句话说的有点不合适吧。。 $x = f(y)$ $y = f(x)$ ，这个是自反性啊 戏子



Re: [40个Java集合面试问题和答案](#)

31题：PriorityQueue是在java.util包下。I hipilee



Re: [Java8 lambda表达式10个示例](#)

具体类型不对，Object 和 String 的问题，，还有main函数中，需要自己写传入类型 [zhang](#)

关于ImportNew

ImportNew 专注于 Java 技术分享。于2012年11月11日 11:11正式上线。是的，这是一个很特别的时刻 :)

ImportNew 由两个 Java 关键字 import 和 new 组成，意指：Java 开发者学习新知识的网站。import 可认为是学习和吸收，new 则可认为是新知识、新技术圈子和新朋友.....



联系我们

Email: ImportNew.com@gmail.com

新浪微博: [@ImportNew](#)

推荐微信号



ImportNew



安卓应用频道



Linux爱好者

反馈建议: ImportNew.com@gmail.com

广告与商务合作QQ: 2302462408

推荐关注

[小组](#) — 好的话题、有启发的回复、值得信赖的圈子

[头条](#) — 写了文章？看干货？去头条！

[相亲](#) — 为IT单身男女服务的征婚传播平台

[资源](#) — 优秀的工具资源导航

[翻译](#) — 活跃 & 专业的翻译小组

[博客](#) — 国内外的精选博客文章

[设计](#) — UI,网页，交互和用户体验

[前端](#) — JavaScript, HTML5, CSS

[安卓](#) — 专注Android技术分享

[iOS](#) — 专注iOS技术分享

[Java](#) — 专注Java技术分享

[Python](#) — 专注Python技术分享

