



Lock与synchronized 的区别

1、ReentrantLock 拥有Synchronized相同的并发性和内存语义，此外还多了 锁投票，定时锁等候和中断锁等候
线程A和B都要获取对象O的锁定，假设A获取了对象O锁，B将等待A释放对O的锁定，
如果使用 synchronized ，如果A不释放，B将一直等下去，不能被中断
如果 使用ReentrantLock，如果A不释放，可以使B在等待了足够长的时间以后，中断等待，而干别的事情

ReentrantLock获取锁定与三种方式：

- a) lock(), 如果获取了锁立即返回，如果别的线程持有锁，当前线程则一直处于休眠状态，直到获取锁
- b) tryLock(), 如果获取了锁立即返回true，如果别的线程正持有锁，立即返回false;
- c)tryLock(long timeout,TimeUnit unit)， 如果获取了锁定立即返回true，如果别的线程正持有锁，会等待参数给定的时间，在等待的过程中，如果获取了锁定，就返回true，如果等待超时，返回false;
- d) lockInterruptibly:如果获取了锁定立即返回，如果没有获取锁定，当前线程处于休眠状态，直到或者锁定，或者当前线程被别的线程中断

2、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，但是使用Lock则不行，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

3、在资源竞争不是很激烈的情况下，Synchronized的性能要优于ReentrantLock，但是在资源竞争很激烈的情况下，Synchronized的性能会下降几十倍，但是ReentrantLock的性能能维持常态；

5.0的多线程任务包对于同步的性能方面有了很大的改进，在原有synchronized关键字的基础上，又增加了ReentrantLock，以及各种Atomic类。了解其性能的优劣程度，有助与我们在特定的情形下做出正确的选择。

总体的结论先摆出来：

synchronized：

在资源竞争不是很激烈的情况下，偶尔会有同步的情形下，synchronized是很合适的。原因在于，编译程序通常会尽可能的进行优化synchronize，另外可读性非常好，不管用没用过5.0多线程包的程序员都能理解。

ReentrantLock：

ReentrantLock提供了多样化的同步，比如有时间限制的同步，可以被Interrupt的同步（synchronized的同步是不能Interrupt的）等。在资源竞争不激烈的情形下，性能稍微比synchronized差点点。但是当同步非常激烈的时候，synchronized的性能一下子能下降好几十倍。而ReentrantLock确还能维持常态。

Atomic：

和上面的类似，不激烈情况下，性能比synchronized略逊，而激烈的时候，也能维持常态。激烈的时候，Atomic的性能会优于ReentrantLock一倍左右。但是其有一个缺点，就是只能同步一个值，一段代码中只能出现一个Atomic的变量，多于一个同步无效。因为他不能在多个Atomic之间同步。

所以，我们写同步的时候，优先考虑synchronized，如果有特殊需要，再进一步优化。ReentrantLock和Atomic如果用的不好，不仅不能提高性能，还可能带来灾难。

先贴测试结果：再贴代码（Atomic测试代码不准确，一个同步中只能有1个Actomic，这里用了2个，但是这里的测试只看速度）

```
=====
round:100000 thread:5
Sync = 35301694
Lock = 56255753
Atom = 43467535
=====
round:200000 thread:10
Sync = 110514604
Lock = 204235455
Atom = 170535361
=====
round:300000 thread:15
Sync = 253123791
Lock = 448577123
Atom = 362797227
=====
```

昵称：[一万年以前](#)
园龄：[11个月](#)
粉丝：[3](#)
关注：[1](#)
[+加关注](#)

< 2017年8月 >						
日	一	二	三	四	五	六
30	31	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31	1	2
3	4	5	6	7	8	9

搜索

找找看

谷歌搜索

常用链接

[我的随笔](#)
[我的评论](#)
[我的参与](#)
[最新评论](#)
[我的标签](#)
[更多链接](#)

随笔档案

[2017年7月 \(1\)](#)
[2017年4月 \(2\)](#)
[2017年3月 \(10\)](#)
[2017年2月 \(13\)](#)
[2017年1月 \(1\)](#)
[2016年12月 \(2\)](#)
[2016年11月 \(2\)](#)
[2016年10月 \(5\)](#)
[2016年9月 \(5\)](#)
[2016年8月 \(6\)](#)

阅读排行榜

1. Lock与synchronized 的区别(20689)
2. mySQL： 两表更新（用一个表更新另一个表）的SQL语句(4340)
3. 谷歌Chrome浏览器提示adobe flash player已过期完美解决办法(2865)
4. 下载MySQL历史版本(1259)
5. Android 6.0 双向通话自动录音(832)

推荐排行榜

1. Lock与synchronized 的区别(2)

round:400000 thread:20

Sync = 16562148262

Lock = 846454786

Atom = 667947183

=====

round:500000 thread:25

Sync = 26932301731

Lock = 1273354016

Atom = 982564544

```
1  package test.thread;
2
3  import static java.lang.System.out;
4
5  import java.util.Random;
6  import java.util.concurrent.BrokenBarrierException;
7  import java.util.concurrent.CyclicBarrier;
8  import java.util.concurrent.ExecutorService;
9  import java.util.concurrent.Executors;
10 import java.util.concurrent.atomic.AtomicInteger;
11 import java.util.concurrent.atomic.AtomicLong;
12 import java.util.concurrent.locks.ReentrantLock;
13
14 public class TestSyncMethods {
15
16     public static void test(int round,int threadNum,CyclicBarrier cyclicBarrier){
17         new SyncTest("Sync",round,threadNum,cyclicBarrier).testTime();
18         new LockTest("Lock",round,threadNum,cyclicBarrier).testTime();
19         new AtomicTest("Atom",round,threadNum,cyclicBarrier).testTime();
20     }
21
22     public static void main(String args[]){
23
24         for(int i=0;i<5;i++){
25             int round=100000*(i+1);
26             int threadNum=5*(i+1);
27             CyclicBarrier cb=new CyclicBarrier(threadNum*2+1);
28             out.println("=====");
29             out.println("round:"+round+" thread:"+threadNum);
30             test(round,threadNum,cb);
31
32         }
33     }
34 }
35
36 class SyncTest extends TestTemplate{
37     public SyncTest(String _id,int _round,int _threadNum,CyclicBarrier _cb){
38         super(_id, _round, _threadNum, _cb);
39     }
40     @Override
41     /**
42      * synchronized关键字不在方法签名里面，所以不涉及重载问题
43      */
44     synchronized long  getValue() {
45         return super.countValue;
46     }
47     @Override
48     synchronized void  sumValue() {
49         super.countValue+=preInit[index++%round];
50     }
51 }
52
53
54 class LockTest extends TestTemplate{
55     ReentrantLock lock=new ReentrantLock();
56     public LockTest(String _id,int _round,int _threadNum,CyclicBarrier _cb){
57         super(_id, _round, _threadNum, _cb);
58     }
59 }
```



```

59  /**
60   * synchronized关键字不在方法签名里面，所以不涉及重载问题
61   */
62  @Override
63  long getValue() {
64      try{
65          lock.lock();
66          return super.countValue;
67      }finally{
68          lock.unlock();
69      }
70  }
71  @Override
72  void sumValue() {
73      try{
74          lock.lock();
75          super.countValue+=preInit[index++%round];
76      }finally{
77          lock.unlock();
78      }
79  }
80  }
81
82
83  class AtomicTest extends TestTemplate{
84      public AtomicTest(String _id,int _round,int _threadNum,CyclicBarrier _cb){
85          super( _id, _round, _threadNum, _cb);
86      }
87      @Override
88      /**
89       * synchronized关键字不在方法签名里面，所以不涉及重载问题
90       */
91      long  getValue() {
92          return super.countValueAtmoic.get();
93      }
94      @Override
95      void  sumValue() {
96          super.countValueAtmoic.addAndGet(super.preInit[indexAtomic.get()%round]);
97      }
98  }
99  abstract class TestTemplate{
100      private String id;
101      protected int round;
102      private int threadNum;
103      protected long countValue;
104      protected AtomicLong countValueAtmoic=new AtomicLong(0);
105      protected int[] preInit;
106      protected int index;
107      protected AtomicInteger indexAtomic=new AtomicInteger(0);
108      Random r=new Random(47);
109      //任务栅栏，同批任务，先到达wait的任务挂起，一直等到全部任务到达制定的wait地点后，才能全部唤醒，
110      private CyclicBarrier cb;
111      public TestTemplate(String _id,int _round,int _threadNum,CyclicBarrier _cb){
112          this.id=_id;
113          this.round=_round;
114          this.threadNum=_threadNum;
115          cb=_cb;
116          preInit=new int[round];
117          for(int i=0;i<preInit.length;i++){
118              preInit[i]=r.nextInt(100);
119          }
120      }
121
122      abstract void sumValue();
123      /*
124       * 对long的操作是非原子的，原子操作只针对32位
125       * long是64位，底层操作的时候分2个32位读写，因此不是线程安全

```




```

126     */
127     abstract long getValue();
128
129     public void testTime(){
130         ExecutorService se=Executors.newCachedThreadPool();
131         long start=System.nanoTime();
132         //同时开启2*ThreadNum个数的读写线程
133         for(int i=0;i<threadNum;i++){
134             se.execute(new Runnable(){
135                 public void run() {
136                     for(int i=0;i<round;i++){
137                         sumValue();
138                     }
139
140                     //每个线程执行完同步方法后就等待
141                     try {
142                         cb.await();
143                     } catch (InterruptedException e) {
144                         // TODO Auto-generated catch block
145                         e.printStackTrace();
146                     } catch (BrokenBarrierException e) {
147                         // TODO Auto-generated catch block
148                         e.printStackTrace();
149                     }
150
151
152                 }
153             });
154             se.execute(new Runnable(){
155                 public void run() {
156
157                     getValue();
158                     try {
159                         //每个线程执行完同步方法后就等待
160                         cb.await();
161                     } catch (InterruptedException e) {
162                         // TODO Auto-generated catch block
163                         e.printStackTrace();
164                     } catch (BrokenBarrierException e) {
165                         // TODO Auto-generated catch block
166                         e.printStackTrace();
167                     }
168
169                 }
170             });
171         }
172
173         try {
174             //当前统计线程也wait,所以CyclicBarrier的初始值是threadNum*2+1
175             cb.await();
176         } catch (InterruptedException e) {
177             // TODO Auto-generated catch block
178             e.printStackTrace();
179         } catch (BrokenBarrierException e) {
180             // TODO Auto-generated catch block
181             e.printStackTrace();
182         }
183         //所有线程执行完成之后,才会跑到这一步
184         long duration=System.nanoTime()-start;
185         out.println(id+" = "+duration);
186
187     }
188
189 }

```


摘自：
<http://houlinyan.iteye.com/blog/1112535>
<http://zzhonghe.iteye.com/blog/826162>

[好文要顶](#)[关注我](#)[收藏该文](#)



[一万年以前](#)
[关注 - 1](#)
[粉丝 - 3](#)
[+加关注](#)

2

推荐


0


反对

« 上一篇：[java遍历hashMap、hashSet、Hashtable](#)
» 下一篇：[Java多线程学习（吐血超详细总结）](#)


posted @ 2016-08-30 13:56 一万年以前 阅读(20688) 评论(0) 编辑 收藏

[刷新评论](#) [刷新页面](#) [返回顶部](#)

 发表评论

昵称： 克己、

评论内容：



[提交评论](#) [退出](#) [订阅评论](#)

[Ctrl+Enter快捷键提交]

- 最新**IT**新闻：
- [鸡走路时为什么脑袋一抖一抖的？总算明白了](#)
 - [马云说新制造要来了，结果中国连造伞都还没自动化](#)
 - [TensorFlow 1.3.0正式发布，包含诸多更新](#)
 - [开放Java EE？甲骨文考虑将Java EE移至开源社区](#)
 - [GitHub首席执行官正在寻找接替者 将辞去CEO职务](#)
- » [更多新闻...](#)

- 最新知识库文章：
- [做到这一点，你也可以成为优秀的程序员](#)
 - [写给立志做码农的大学生](#)
 - [架构腐化之谜](#)
 - [学会思考，而不只是编程](#)
 - [编写Shell脚本的最佳实践](#)
- » [更多知识库文章...](#)