



AtomicInteger类保证线程安全的用法

J2SE 5.0提供了一组atomic class来帮助我们简化同步处理。基本工作原理是使用了同步synchronized的方法实现了对一个long, integer, 对象的增、减、赋值(更新)操作。比如对于++运算符AtomicInteger可以将它持有的integer 能够atomic 地递增。在需要访问两个或两个以上 atomic变量的程序代码(或者是对单一的atomic变量执行两个或两个以上的操作) 通常都需要被synchronize以使两者的操作能够被当作是一个atomic的单元。

Java多线程用法-使用AtomicInteger

下面通过简单的两个例子的对比来看一下 AtomicInteger 的强大功能

```
1 class Counter {
2     private volatile int count = 0;
3
4     public synchronized void increment() {
5         count++; //若要线程安全执行count++, 需要加锁
6     }
7
8     public int getCount() {
9         return count;
10    }
11 }
12
13 class Counter {
14     private AtomicInteger count = new AtomicInteger();
15
16     public void increment() {
17         count.incrementAndGet();
18     }
19
20     //使用AtomicInteger之后, 不需要加锁, 也可以实现线程安全
21     public int getCount() {
22         return count.get();
23     }
24 }
```

从上面的例子中我们可以看出: 使用AtomicInteger是非常的安全的
那么为什么不使用计数器自加呢, 例如count++这样的, 因为这种计数是线程不安全的, 高并发访问时统计会有误, 而AtomicInteger为什么能够达到多而不乱, 处理高并发应付自如呢?
这是由硬件提供原子操作指令实现的。在非激烈竞争的情况下, 开销更小, 速度更快。Java.util.concurrent中实现的原子操作类包括: AtomicBoolean, AtomicInteger, AtomicLong, AtomicReference。

另外其底层就是volatile和CAS 共同作用的结果:

- 1.首先使用了volatile 保证了内存可见性。
- 2.然后使用了CAS (compare-and-swap) 算法 保证了原子性。

其中CAS算法的原理就是里面包含三个值: 内存值A 预估值V 更新值B 当且仅当 V == A 时, V = B; 否则, 不会执行任何操作。

分类: Java源码解读

好文要读 关注 收藏该文

 Jason Bai

关注 - 6

粉丝 - 172

+加关注

0 推荐

0 反对

« 上一篇: 在项目启动时 使用监听器加载所有字典表数据
» 下一篇: springmvc导出excel并弹出下载框

posted @ 2016-07-12 09:25 Jason.Bai 阅读(10869) 评论(0) 编辑 收藏

昵称: Jason.Bai

园龄: 4年7个月

粉丝: 172

关注: 6

+加关注

2019年11月						
日	一	二	三	四	五	六
27	28	29	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
1	2	3	4	5	6	7

找我看

谷歌搜索

积分 - 278419

排名 - 1260

elasticsearch

1. Re:docker 安装MySQL远程连接

@ rarelyrm 在这儿是docker run的参
数, 在Linux指令中才是删除吧! ...

--Jason.Bai

2. Re:docker 安装MySQL远程连接

\$ docker run -it --rm --name mysql
-e MYSQL_ROOT_PASSWORD=1234
56 -p 3306:3306 -d mysql 这里为什
么要--rm呢? ~...

--rarely

3. Re:redis.conf配置详细翻译解析

感觉很不错, 虽说不是很全但是还是帮到
我了

--企鹅不会飞

4. Re:Java四种线程池newCachedThre
adPool,newFixedThreadPool,newSch
eduledThreadPool,newSingleThread
Executor

@ MicroCat致敬! 复制产生的错误。 ...

--Jason.Bai

5. Re:Java四种线程池newCachedThre
adPool,newFixedThreadPool,newSch
eduledThreadPool,newSingleThread
Executor

public static ScheduledExecutorServ
ice newScheduledThreadPool(int cor
ePoolSize) { return new Schedul...

--MicroCat

- 1. Lock和Synchronized的区别和使用(26)
- 2. HTTP与TCP的区别和联系(17)
- 3. ReentrantLock可重入锁 (和Synchronized的区别) 总结(9)

4. 一个简单的死锁例子(7)

5. Java四种线程池newCachedThreadPool,newFixedThreadPool,newScheduledThreadPool,newSingleThreadExecutor(7)