# 海 子

不经历风雨，怎能见彩虹？做一个快乐的程序员。





联系方式：
邮箱：dolphin0520@163.com
微信：scu_dolphin0520

昵称：海 子
园龄：7年5个月
粉丝：5815
关注：6
+加关注



< 2018年9月 >

| 日 | 一 | 二 | 三 | 四 | 五 | 六 |
|---|---|---|---|---|---|---|
| 26 | 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 |

搜索

找找看

谷歌搜索


## Java并发编程：并发容器之CopyOnWriteArrayList（转载）

Java并发编程：并发容器之CopyOnWriteArrayList（转载）

原文链接：

http://ifeve.com/java-copy-on-write/

Copy-On-Write简称COW，是一种用于程序设计中的优化策略。其基本思路是，从一开始大家都在共享同一个内容，当某个人想要修改这个内容的时候，才会真正把内容Copy出去形成一个新的内容然后再改，这是一种延时懒惰策略。从JDK1.5开始Java并发包里提供了两个使用CopyOnWrite机制实现的并发容器,它们是CopyOnWriteArrayList和CopyOnWriteArraySet。CopyOnWrite容器非常有用，可以在非常多的并发场景中使用到。

### 什么是CopyOnWrite容器

CopyOnWrite容器即写时复制的容器。通俗的理解是当我们往一个容器添加元素的时候，不直接往当前容器添加，而是先将当前容器进行Copy，复制出一个新的容器，然后新的容器里添加元素，添加完元素之后，再将原容器的引用指向新的容器。这样做的好处是我们可以对CopyOnWrite容器进行并发的读，而不需要加锁，因为当前容器不会添加任何元素。所以CopyOnWrite容器也是一种读写分离的思想，读和写不同的容器。

## 最新随笔

1. 【置顶】博客恢复更新公告
2. 金山快盘+TortoiseSVN构建版本控制仓库
3. 在windows下安装配置Ulipad
4. Java并发编程：线程间协作的两种方式：wait、notify、notifyAll和Condition
5. JAVA多线程和并发基础面试问答（转载）
6. Java线程面试题 Top 50 （转载）
7. Java并发编程：Timer和TimerTask（转载）
8. Java并发编程：Callable、Future和FutureTask
9. Java并发编程：CountDownLatch、CyclicBarrier和Semaphore
10. Java并发编程：线程池的使用

## 随笔分类(194)

Android(14)

C/C++(30)

Java NIO(2)

Java并发编程(19)


## CopyOnWriteArrayList的实现原理

在使用CopyOnWriteArrayList之前，我们先阅读其源码了解下它是如何实现的。以下代码是向CopyOnWriteArrayList中add方法的实现（向CopyOnWriteArrayList里添加元素），可以发现在添加的时候是需要加锁的，否则多线程写的时候会Copy出N个副本出来。

```
/**
     * Appends the specified element to the end of this list.
     *
     * @param e element to be appended to this list
     * @return <tt>true</tt> (as specified by {@link Collection#add})
     */
    public boolean add(E e) {
    final ReentrantLock lock = this.lock;
    lock.lock();
    try {
        Object[] elements = getArray();
        int len = elements.length;
        Object[] newElements = Arrays.copyOf(elements, len + 1);
        newElements[len] = e;
        setArray(newElements);
        return true;
    } finally {
        lock.unlock();
    }
    }
```

读的时候不需要加锁，如果读的时候有多个线程正在向CopyOnWriteArrayList添加数据，读还是会读到旧的数据，因为写的时候不会锁住旧的CopyOnWriteArrayList。

```
public E get(int index) {
    return get(getArray(), index);
}
```

JDK中并没有提供CopyOnWriteMap，我们可以参考CopyOnWriteArrayList来实现一个，基本代码如下：

```
import java.util.Collection;
import java.util.Map;
import java.util.Set;

public class CopyOnWriteMap<K, V> implements Map<K, V>, Cloneable {
    private volatile Map<K, V> internalMap;

    public CopyOnWriteMap() {
        internalMap = new HashMap<K, V>();
    }

    public V put(K key, V value) {

        synchronized (this) {
            Map<K, V> newMap = new HashMap<K, V>(internalMap);
            V val = newMap.put(key, value);
            internalMap = newMap;
            return val;
        }
    }

    public V get(Object key) {
        return internalMap.get(key);
    }

    public void putAll(Map<? extends K, ? extends V> newData) {
        synchronized (this) {
            Map<K, V> newMap = new HashMap<K, V>(internalMap);
            newMap.putAll(newData);
            internalMap = newMap;
        }
    }
```

数据挖掘(1)
算法(27)
无线传感器网络(1)
信息检索
业余娱乐(7)
转载(14)
自然语言处理

常用链接
C++ Reference
MSDN 主页
SOJ
北大OJ
并发编程网
杭电OJ

积分与排名
积分 - 439715
排名 - 370

最新评论

```
33  }
```

实现很简单，只要了解了CopyOnWrite机制，我们可以实现各种CopyOnWrite容器，并且在不同的应用场景中使用。

## CopyOnWrite的应用场景

CopyOnWrite并发容器用于读多写少的并发场景。比如白名单，黑名单，商品类目的访问和更新场景，假如我们有一个搜索网站，用户在这个网站的搜索框中，输入关键字搜索内容，但是某些关键字不允许被搜索。这些不能被搜索的关键字会被放在一个黑名单当中，黑名单每天晚上更新一次。当用户搜索时，会检查当前关键字在不在黑名单当中，如果在，则提示不能搜索。实现代码如下：

```
1  package com.ifeve.book;
2
3  import java.util.Map;
4
5  import com.ifeve.book.forkjoin.CopyOnWriteMap;
6
7  /**
8   * 黑名单服务
9   *
10  * @author fangtengfei
11  *
12  */
13 public class BlackListServiceImpl {
14
15     private static CopyOnWriteMap<String, Boolean> blackListMap = new CopyOnWriteMap<String, Boolean>(
16             1000);
17
18     public static boolean isBlackList(String id) {
19         return blackListMap.get(id) == null ? false : true;
20     }
```

```
21
22      public static void addBlackList(String id) {
23          blackListMap.put(id, Boolean.TRUE);
24      }
25
26      /**
27       * 批量添加黑名单
28       *
29       * @param ids
30       */
31      public static void addBlackList(Map<String,Boolean> ids) {
32          blackListMap.putAll(ids);
33      }
34
35  }
```

代码很简单，但是使用CopyOnWriteMap需要注意两件事情：

1. 减少扩容开销。根据实际需要，初始化CopyOnWriteMap的大小，避免写时CopyOnWriteMap扩容的开销。

2. 使用批量添加。因为每次添加，容器每次都会进行复制，所以减少添加次数，可以减少容器的复制次数。如使用上面代码里的addBlackList方法。

## CopyOnWrite的缺点

CopyOnWrite容器有很多优点，但是同时也存在两个问题，即内存占用问题和数据一致性问题。所以在开发的时候需要注意一下。

**内存占用问题**。因为CopyOnWrite的写时复制机制，所以在进行写操作的时候，内存里会同时驻扎两个对象的内存，旧的对象和新写入的对象（注意:在复制的时候只是复制容器里的引用，只是在写的时候会创建新对象添加到

新容器里，而旧容器的对象还在使用，所以有两份对象内存）。如果这些对象占用的内存比较大，比如说200M左右，那么再写入100M数据进去，内存就会占用300M，那么这个时候很有可能造成频繁的Yong GC和Full GC。之前我们系统中使用了一个服务由于每晚使用CopyOnWrite机制更新大对象，造成了每晚15秒的Full GC，应用响应时间也随之变长。

针对内存占用问题，可以通过压缩容器中的元素的方法来减少大对象的内存消耗，比如，如果元素全是10进制的数字，可以考虑把它压缩成36进制或64进制。或者不使用CopyOnWrite容器，而使用其他的并发容器，如ConcurrentHashMap。

**数据一致性问题**。CopyOnWrite容器只能保证数据的最终一致性，不能保证数据的实时一致性。所以如果你希望写入的的数据，马上能读到，请不要使用CopyOnWrite容器。

下面这篇文章验证了CopyOnWriteArrayList和同步容器的性能：

http://blog.csdn.net/wind5shy/article/details/5396887

下面这篇文章简单描述了CopyOnWriteArrayList的使用：

http://blog.csdn.net/imzoer/article/details/9751591



分类：Java并发编程

标签： java并发编程

海 子
关注 - 6
粉丝 - 5815
+加关注

18 0

« 上一篇：Java并发编程：并发容器之ConcurrentHashMap（转载）
» 下一篇：Java并发编程：阻塞队列

posted @ 2014-08-27 10:47 海 子 阅读(85573) 评论(10) 编辑 收藏

**评论列表**

#1楼 2016-04-05 12:13 陈树义

学习了。

支持(0) 反对(0)

#2楼 2016-04-27 14:24 AudienL

学习了

支持(0) 反对(0)

#3楼 2016-05-20 18:32 大鹏520

关于“ 读的时候不需要加锁，如果读的时候有多个线程正在向CopyOnWriteArrayList添加数据，读还是会读到旧的数据，因为写的时候不会锁住旧的CopyOnWriteArrayList。”

JDK文档上的说明：内存一致性效果：当存在其他并发 collection 时，将对象放入 CopyOnWriteArrayList 之前的线程中的操作 happen-before 随后通过另一线程从 CopyOnWriteArrayList 中访问或移除该元素的操作。

按JDK说明，应该是不会读到脏数据的吧?

支持(0) 反对(0)

---

#4楼 2016-11-23 14:07 杰-维斯布鲁克

@ 大鹏520

引用

关于“ 读的时候不需要加锁，如果读的时候有多个线程正在向CopyOnWriteArrayList添加数据，读还是会读到旧的数据，因为写的时候不会锁住旧的CopyOnWriteArrayList。”
JDK文档上的说明：内存一致性效果：当存在其他并发 collection 时，将对象放入 CopyOnWriteArrayList 之前的线程中的操作 happen-before 随后通过另一线程从 CopyOnWriteArrayList 中访问或移除该元素的操作。

按JDK说明，应该是不会读到脏数据的吧?

我认为JDK文档上的说明没有问题，博主的意思也没错。确实根据happen-before原则，对一个变量的写操作先行发生于后面对这个变量的读操作。但CopyOnWriteArrayList可以并发读，这个时候读到的确实是所谓的“旧数据”，并不是“脏数据”。其实我觉得博主这句话根本没必要写，因为add操作，是往里面加数据，并不是修改原来元素的值。add之后，博主所谓的旧数据，和新的array的对应数据的值是一样。说旧也只能是说，在add的时候读是由原先的array提供的，add之后相同的值是由新的array提供，物理地址不一样吧

支持(2) 反对(0)

---

#5楼 2017-10-20 21:17 岁月下的车辙

这边添加数据的时候添加的只是引用，为什么会造成内存问题?

支持(0) 反对(0)

---

#6楼 2018-01-01 14:29 甜甜咿呀咿呀哟

@ 岁月下的车辙
因为旧的数组可能没有及时被垃圾收集器回收掉

支持(0) 反对(0)

#7楼 2018-01-01 14:31 甜甜咿呀咿呀哟

volatile Object[] array，有volatile修饰不是立刻能被其他线程知道吗，如果你希望写入的的数据，马上能读到，请不要使用CopyOnWrite容器，感觉影响不会很大啊

支持(0) 反对(0)

#8楼 2018-01-11 15:02 龙须子

@ 大鹏520
JDK的注释原话：

```
/*
 * <p>Memory consistency effects: As with other concurrent
 * collections, actions in a thread prior to placing an object into a
 * {<a href="http://home.cnblogs.com/u/25114/" target="_blank">@code</a> CopyOnWriteArrayList}
 * <a href="package-summary.html#MemoryVisibility"><i>happen-before</i></a>
 * actions subsequent to the access or removal of that element from
 * the {<a href="http://home.cnblogs.com/u/25114/" target="_blank">@code</a> CopyOnWriteArrayList} in
 */
```

应该这么翻译：内存一致性的原理：与其他并发集合一样，由于happen-before原则，一个线程将一个对象放进CopyOnWriteArrayList的动作之前于另一个线程从CopyOnWriteArrayList访问或者删除这个对象。

所以说 CopyOnWriteArrayList 是ArrayList的一个线程安全的变体实现，在这个变体上进行的所有操作，例如add,set等是通过生成底层数组的一个新副本来实现的。

支持(0) 反对(0)

#9楼 2018-01-11 15:11 龙须子

@ 甜甜咿呀咿呀哟
存放元素的数组确实是

```
1 private transient volatile Object[] array;
```

volatile 修饰的，理论上被volatile 修饰的变量修改了以后，对于别的线程是可见的，但是CopyOnWrite在写操作的时候相当于复制了一份新的数组，新添加的元素是放在了新的数组里面，在没有修改原来数组的指针之前，别的线程读的还是原来的那个array，所以肯定是读取不到新添加的元素的，所以说“如果你希望写入的的数据，马上能读到，请不要使用CopyOnWrite容器”

支持(0) 反对(0)

---

#10楼 2018-07-19 17:31 吴小凯

@ 大鹏520
这个叫做幻读

支持(0) 反对(0)

---

刷新评论 刷新页面 返回顶部

**注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。**