

[博客专区](#) > [mononite的博客](#) > [博客详情](#)

一站式构建百万流量网站

3大通用架构
12大应用场景深度剖析 阿里云
230万用户信赖

原 ConcurrentHashMap使用示例

[mononite](#) 发表于 4年前 阅读 18372 收藏 36 点赞 7 评论 4[收藏](#)**【粉丝福利】 - 《web 前端基础到实战系列课程》免费在线直播教学>>>** 

ConcurrentHashMap通常只被看做并发效率更高的Map，用来替换其他线程安全的Map容器，比如Hashtable和Collections.synchronizedMap。实际上，线程安全的容器，特别是Map，应用场景没有想象中的多，很多情况下一个业务会涉及容器的多个操作，即复合操作，并发执行时，线程安全的容器只能保证自身的数据不被破坏，但无法保证业务的行为是否正确。

举个例子：统计文本中单词出现的次数，把单词出现的次数记录到一个Map中，代码如下：

```
private final Map<String, Long> wordCounts = new ConcurrentHashMap<>();

public long increase(String word) {
    Long oldValue = wordCounts.get(word);
    Long newValue = (oldValue == null) ? 1L : oldValue + 1;
    wordCounts.put(word, newValue);
    return newValue;
}
```

如果多个线程并发调用这个increase()方法，increase()的实现就是错误的，因为多个线程用相同的word调用时，很可能会覆盖相互的结果，造成记录的次数比实际出现的次数少。

除了用锁解决这个问题，另外一个选择是使用ConcurrentMap接口定义的方法：

```
public interface ConcurrentMap<K, V> extends Map<K, V> {
    V putIfAbsent(K key, V value);
    boolean remove(Object key, Object value);
    boolean replace(K key, V oldValue, V newValue);
    V replace(K key, V value);
}
```

这是个被很多人忽略的接口，也经常有人错误地使用这个接口。ConcurrentMap接口定义了几个基于CAS (Compare and Set) 操作，很简单，但非常有用，下面的代码用ConcurrentMap解决上面问题：

```
private final ConcurrentMap<String, Long> wordCounts = new ConcurrentHashMap<>();

public long increase(String word) {
    Long oldValue, newValue;
    while (true) {
        oldValue = wordCounts.get(word);
        if (oldValue == null) {
            newValue = 1L;
        } else {
            newValue = oldValue + 1;
        }
        wordCounts.put(word, newValue);
    }
}
```

```

// Add the word firstly, initial the value as 1
newValue = 1L;
if (wordCounts.putIfAbsent(word, newValue) == null) {
    break;
}
} else {
    newValue = oldValue + 1;
    if (wordCounts.replace(word, oldValue, newValue)) {
        break;
    }
}
}
return newValue;
}

```

代码有点复杂，主要因为ConcurrentMap中不能保存value为null的值，所以得同时处理word不存在和已存在两种情况。

上面的实现每次调用都会涉及Long对象的拆箱和装箱操作，很明显，更好的实现方式是采用AtomicLong，下面是采用AtomicLong后的代码：

```

private final ConcurrentMap<String, AtomicLong> wordCounts = new ConcurrentHashMap<>();

public long increase(String word) {
    AtomicLong number = wordCounts.get(word);
    if (number == null) {
        AtomicLong newNumber = new AtomicLong(0);
        number = wordCounts.putIfAbsent(word, newNumber);
        if (number == null) {
            number = newNumber;
        }
    }
    return number.incrementAndGet();
}

```

这个实现仍然有一处需要说明的地方，如果多个线程同时增加一个目前还不存在的词，那么很可能会产生多个newNumber对象，但最终只有一个newNumber有用，其他的都会被扔掉。对于这个应用，这不算问题，创建AtomicLong的成本不高，而且只在添加不存在词是出现。但换个场景，比如缓存，那么这很可能就是问题了，因为缓存中的对象获取成本一般都比较低，而且通常缓存都会经常失效，那么避免重复创建对象就有价值了。下面的代码演示了怎么处理这种情况：

```

private final ConcurrentMap<String, Future<ExpensiveObj>> cache = new ConcurrentHashMap<>();

public ExpensiveObj get(final String key) {
    Future<ExpensiveObj> future = cache.get(key);
    if (future == null) {
        Callable<ExpensiveObj> callable = new Callable<ExpensiveObj>() {
            @Override
            public ExpensiveObj call() throws Exception {
                return new ExpensiveObj(key);
            }
        };
        FutureTask<ExpensiveObj> task = new FutureTask<>(callable);

        future = cache.putIfAbsent(key, task);
        if (future == null) {
            future = task;
        }
    }
}

```

```
        task.run();
    }
}

try {
    return future.get();
} catch (Exception e) {
    cache.remove(key);
    throw new RuntimeException(e);
}
}
```

解决方法其实就是用了一个Proxy对象来包装真正的对象，跟常见的lazy load原理类似；使用FutureTask主要是为了保证同步，避免一个Proxy创建多个对象。注意，上面代码里的异常处理是不准确的。

最后再补充一下，如果真要实现前面说的统计单词次数功能，最合适的方法是Guava包中AtomicLongMap；一般使用ConcurrentHashMap，也尽量使用Guava中的MapMaker或cache实现。

© 著作权归作者所有

分类：工作日志 字数：932

标签：Java

打赏

点赞

收藏

分享



mononite

程序员 朝阳

+ 关注

粉丝 11 | 博文 5 | 码字总数 6640



相关博客

JAVA的问题



崔钢

4154

0

java



it-world

475

1

JAVA



做个坏蛋去社会

144

0

评论 (4)

Ctrl+Enter 发表评论



WinWill2012

1楼 2015/09/10 23:31

这篇文章深度剖析了ConcurrentHashMap的内部实现，推荐大家看看：

[http://qifuguang.me/2015/09/10/\[Java并发包学习八\]深度剖析ConcurrentHashMap/](http://qifuguang.me/2015/09/10/[Java并发包学习八]深度剖析ConcurrentHashMap/)

一只小桃子

2楼 2015/11/10 10:06

写的很好



xtaetg2

3楼 2016/08/05 18:14

```
private final ConcurrentMap<String, AtomicLong> wordCounts = new ConcurrentHashMap<>();
```

```
public long increase(String word) {  
    AtomicLong number = wordCounts.get(word);  
    if (number == null) {  
        AtomicLong newNumber = new AtomicLong(0);  
        number = wordCounts.putIfAbsent(word, newNumber);  
        if (number == null) {  
            number = newNumber;  
        }  
    }  
    return number.incrementAndGet();  
}
```

第一次访问的并发情况下，数字统计会漏掉，比如两个线程同时访问，会new 两个AtomicLong，第一个被覆盖，number.incrementAndGet()返回的是1，但是实际上应该是2次了。



xtaetg2

4楼 2016/08/05 18:35

不好意思，😓我理解错了。。。

社区

开源项目
技术问答
动弹
博客

众包

项目大厅
软件与服务
接活赚钱
招聘

码云

Git代码托管
Team
PaaS
在线工具

活动

线下活动
发起活动
源创会

关注微信公众号



下载手机客户端

