

【优雅代码】15-guavaCache本地缓存使用及源码解析



六边形战士夏宁 Lv1

2022年01月14日 21:53 · 阅读 275

关注



【优雅代码】15-guavaCache本地缓存使用及源码解析

欢迎关注b站账号/公众号【六边形战士夏宁】，一个要把各项指标拉满的男人。该文章已在[github目录](#)收录。

屏幕前的**大帅比**和**大漂亮**如果有帮助到你的话请顺手点个赞、加个收藏这对我真的很重要。别下次一定了，都不关注上哪下次一定。

- [视频讲解](#)
- [可直接运行的完整代码](#)
- [上一篇](#)guava精选方法及eventBus观察者模式源码解析
- [下一篇](#)guava布隆过滤与限流算法源码解析

1.背景

承接前一篇章的guava精选方法

2.cache

这一块的功能设计真的很精巧，特别是队列的设计

2.1使用

java 复制代码

@SneakyThrows

```
public static void cache() {  
    // 注意两个如果一起用有时候会有bug  
    Cache<Integer, Integer> accessBuild = CacheBuilder.newBuilder().expireAfterAccess(1, TimeUnit.SECONDS).build()  
    Cache<Integer, Integer> writeBuild = CacheBuilder.newBuilder().expireAfterWrite(1, TimeUnit.SECONDS).build();  
  
    accessBuild.put(1, 1);  
    accessBuild.put(2, 2);  
    writeBuild.put(1, 1);  
    writeBuild.put(2, 2);  
    // 输出1  
    System.out.println(accessBuild.getIfPresent(1));  
    // 输出1  
    System.out.println(writeBuild.getIfPresent(1));  
    Thread.sleep(500);  
    // 输出2  
    System.out.println(accessBuild.getIfPresent(2));  
    Thread.sleep(600);  
    // 输出null  
    System.out.println(accessBuild.getIfPresent(1));  
    // 输出2  
    System.out.println(accessBuild.getIfPresent(2));  
    // 输出null  
    System.out.println(writeBuild.getIfPresent(1));  
}
```

输出如下

text 复制代码

1
1

2.2核心源码详解

1. 构造方法

java 复制代码

```
// 整体构造链相对简单
// build方法
public <K1 extends K, V1 extends V> Cache<K1, V1> build() {
    checkWeightWithWeigher();
    checkNonLoadingCache();
    return new LocalCache.LocalManualCache<>(this);
}
// 给expireAfterAccessNanos赋值失效时间
public CacheBuilder<K, V> expireAfterAccess(long duration, TimeUnit unit) {
    checkState(
        expireAfterAccessNanos == UNSET_INT,
        "expireAfterAccess was already set to %s ns",
        expireAfterAccessNanos);
    checkArgument(duration >= 0, "duration cannot be negative: %s %s", duration, unit);
    this.expireAfterAccessNanos = unit.toNanos(duration);
    return this;
}
// 给expireAfterWriteNanos赋值失效时间
public CacheBuilder<K, V> expireAfterWrite(long duration, TimeUnit unit) {
    checkState(
        expireAfterWriteNanos == UNSET_INT,
        "expireAfterWrite was already set to %s ns",
        expireAfterWriteNanos);
    checkArgument(duration >= 0, "duration cannot be negative: %s %s", duration, unit);
    this.expireAfterWriteNanos = unit.toNanos(duration);
    return this;
}
```

2. put

java 复制代码

```
// put方法, 和老版本的ConcurrentHashMap一样的设计模式, 用segment桶
@Override
public V put(K key, V value) {
    checkNotNull(key);
    checkNotNull(value);
    int hash = hash(key);
    return segmentFor(hash).put(key, hash, value, false);
}
```

```

V put(K key, int hash, V value, boolean onlyIfAbsent) {
    lock();
    try {
        long now = map.ticker.read();
        // *****重点方法:清除过期内容*****
        preWriteCleanup(now);

        int newCount = this.count + 1;
        if (newCount > this.threshold) { // ensure capacity
            expand();
            newCount = this.count + 1;
        }

        AtomicReferenceArray<ReferenceEntry<K, V>> table = this.table;
        int index = hash & (table.length() - 1);
        ReferenceEntry<K, V> first = table.get(index);

        // Look for an existing entry.
        // 这里判断一下是不是已经有同样的key了
        for (ReferenceEntry<K, V> e = first; e != null; e = e.getNext()) {
            K entryKey = e.getKey();
            if (e.getHash() == hash
                && entryKey != null
                && map.keyEquivalence.equivalent(key, entryKey)) {
                // We found an existing entry.

                ValueReference<K, V> valueReference = e.getValueReference();
                V entryValue = valueReference.get();

                if (entryValue == null) {
                    ++modCount;
                    if (valueReference.isActive()) {
                        enqueueNotification(
                            key, hash, entryValue, valueReference.getWeight(), RemovalCause.COLLECTED);
                        setValue(e, key, value, now);
                        newCount = this.count; // count remains unchanged
                    } else {
                        setValue(e, key, value, now);
                        newCount = this.count + 1;
                    }
                }
                this.count = newCount; // write-volatile
                evictEntries(e);
                return null;
            } else if (onlyIfAbsent) {
                // Mimic
                // "if (!map.containsKey(key)) ..."
                // else return map.get(key);
            }
        }
    }
}

```

```

    } else {
        // clobber existing entry, count remains unchanged
        ++modCount;
        enqueueNotification(
            key, hash, entryValue, valueReference.getWeight(), RemovalCause.REPLACED);
        setValue(e, key, value, now);
        evictEntries(e);
        return entryValue;
    }
}

// Create a new entry.
++modCount;
ReferenceEntry<K, V> newEntry = newEntry(key, hash, first);
// *****重点方法:赋值*****
setValue(newEntry, key, value, now);
table.set(index, newEntry);
newCount = this.count + 1;
this.count = newCount; // write-volatile
evictEntries(newEntry);
return null;
} finally {
    unlock();
    // *****重点方法:调用监听者*****
    postWriteCleanup();
}
}

```

3. setValue

java 复制代码

```

@GuardedBy("this")
void setValue(ReferenceEntry<K, V> entry, K key, V value, long now) {
    // 获取这个包装entry原先的值, 如果原先这个key不存在, 则获取不到东西
    ValueReference<K, V> previous = entry.getValueReference();
    int weight = map.weigher.weigh(key, value);
    checkState(weight >= 0, "Weights must be non-negative");

    ValueReference<K, V> valueReference =
        map.valueStrength.referenceValue(this, entry, value, weight);
    // 将value写入到entry包装对象中
    entry.setValueReference(valueReference);
    // 核心方法
    recordWrite(entry, weight, now);
    previous.notifyNewValue(value);
}

```

```
// we are already under lock, so drain the recency queue immediately
drainRecencyQueue();
totalWeight += weight;

if (map.recordsAccess()) {
    // 设置访问时间
    entry.setAccessTime(now);
}
if (map.recordsWrite()) {
    // 设置写时间
    entry.setWriteTime(now);
}
// *****重点方法*****这个地方将指针存了两份队列到末尾，因为缓存时间是一致的，所以只要判断队列
accessQueue.add(entry);
writeQueue.add(entry);
}
```

4. WriteQueue与accessQueue

这两个用的实现类基本一致，这里重写了add方法，重写的目的是如果key一样的entry就进行重排而不是插入

java 复制代码

@Override

```
public boolean offer(ReferenceEntry<K, V> entry) {
    // unlink
    // 将entry的前一个和后一个进行互指
    connectWriteOrder(entry.getPreviousInWriteQueue(), entry.getNextInWriteQueue());

    // add to tail
    // entry和对头互指，和队尾互指，即添加到队尾
    connectWriteOrder(head.getPreviousInWriteQueue(), entry);
    connectWriteOrder(entry, head);

    return true;
}
```

// 两个对象进行互指

```
static <K, V> void connectWriteOrder(ReferenceEntry<K, V> previous, ReferenceEntry<K, V> next) {
    previous.setNextInWriteQueue(next);
    next.setPreviousInWriteQueue(previous);
}
```

5. preWriteCleanup

java 复制代码

```

    }
    void runLockedCleanup(long now) {
        if (tryLock()) {
            try {
                drainReferenceQueues();
                // 核心方法继续进入
                expireEntries(now); // calls drainRecencyQueue
                readCount.set(0);
            } finally {
                unlock();
            }
        }
    }
    void expireEntries(long now) {
        drainRecencyQueue();

        ReferenceEntry<K, V> e;
        // 就两个队列不停的判断头节点是不是失效
        while ((e = writeQueue.peek()) != null && map.isExpired(e, now)) {
            if (!removeEntry(e, e.getHash(), RemovalCause.EXPIRED)) {
                throw new AssertionError();
            }
        }
        while ((e = accessQueue.peek()) != null && map.isExpired(e, now)) {
            if (!removeEntry(e, e.getHash(), RemovalCause.EXPIRED)) {
                throw new AssertionError();
            }
        }
    }
}

```

6. 监听者模式

java 复制代码

```

void postWriteCleanup() {
    // 核心方法继续进入
    runUnlockedCleanup();
}

void runUnlockedCleanup() {
    // locked cleanup may generate notifications we can send unlocked
    if (!isHeldByCurrentThread()) {
        // 核心方法继续进入
        map.processPendingNotifications();
    }
}

void processPendingNotifications() {
    RemovalNotification<K, V> notification;

```

```
try {  
    // 核心流程，只要实现removalListener，通过构造方法传进来，然后这里就会同步调用实现的回调方法  
    // PS第一遍看源码一度卡在这个位置，不知道这玩意儿就一个通知机制怎么就移除元素了  
    removalListener.onRemoval(notification);  
} catch (Throwable e) {  
    logger.log(Level.WARNING, "Exception thrown by removal listener", e);  
}  
}  
}
```

分类： 后端 标签： 后端

评论

输入评论（Enter换行，Ctrl + Enter发送）

相关推荐

前端要努力 6天前 前端 后端 面试

接了很多私活的感触

3.4w 329 200

程序员阿牛 9月前 架构 后端

领导：谁再用定时任务实现关闭订单，立马滚蛋！

4.3w 347 134

沉默王二 2月前 后端 GitHub

再见收费的Navicat！操作所有数据库就靠它了！

4.1w 257 113

NanBox 2年前 Android

Android 后台运行白名单，优雅实现保活

MacroZheng 1月前 后端 Java Redis

颜值爆表！Redis官方可视化工具来啦，功能真心强大！

4.5w 282 41

暮色妖娆丶 1月前 后端 Java

优秀的后端应该有哪些开发习惯？

3.9w 481 187

拉不拉米 4月前 后端 程序员

『2021年终总结』10年深飘，3辆车，3套房

3.6w 138 237

后端架构进阶 6月前 后端

Caffeine Cache进阶本地缓存之王

1344 4 2

我有一只喵喵 6月前 后端 Java

Java开发利器Guava Cache之使用篇

779 5 评论

why技术 3月前 前端 后端 Java

请问各位程序员，是我的思维方式有错误吗？

4.9w 253 234

快跑啊小卢_ 29天前 前端 JavaScript 面试

几个一看就会的实用JavaScript优雅小技巧🌟

3.1w 412 61

一灰灰 9月前 后端

Spring系列缓存注解@Cacheable @CacheEvict @CachePut 使用姿势介绍

1485 10 评论

沉默王二 1年前 Java 后端

捡田螺的小男孩 11月前 后端 Java

美团二面：Redis与MySQL双写一致性如何保证？

5.0w 599 125

Ethan01 2月前 前端 后端

面试题 -- 跨域请求如何携带cookie？

4.8w 633 76

JiandanDream 11月前 iOS

源码浅析-iOS缓存NSCache

1111 6 10

烂猪皮 1年前 Java

缓存之王Caffeine Cache，性能比Guava更强

1492 4 评论

Sunshine_Lin 5月前 前端 Vue.js 面试

后端一次给你10万条数据，如何优雅展示，到底考察我什么？

9.5w 1145 201

MacroZheng 3月前 Java 后端

再见 Typora！这款开源的 Markdown 神器界面更炫酷，逼格更高！

5.4w 236 98



1



评论



收藏
