缓存篇：Guava cache 之全面剖析

愤怒的小...
我爱大风与烈酒，也爱孤独和自由

关注

32 人赞同了该文章

最近在项目中接触到redis分布式缓存，然后在我师父的指导下，本地缓存guava cache了解一下，不看不知道，一看吓一跳， Guava Cache是本地缓存的不二之选呵，当然，我们也可以写一个map当作缓存，但无疑在多线程环境下，其**线程安全，容量溢出，垃圾回收**等均得仔细考究，下面来看看 Guava Cache的操作与原理。

请双手扶好键盘，一顿操作猛如虎：

1 创建（加载）cache

两种方法 CacheLoader和Callable ， 直接上代码：

```
@Test
public void testCacheLoader() throws ExecutionException{
    LoadingCache userCache = CacheBuilder.newBuilder()
                .maximumSize(100) // maximum 100 records can be cached
                .expireAfterAccess( duration: 30, TimeUnit.MINUTES) // 过期时间
                .build(new CacheLoader() {
                    @Override
                    public Object load(Object name) throws Exception {   //在cache找不到就采用load
                        //make the expensive call
                        User user = new User();
                        user.setUsername(name.toString());
                        user.setEmail("123456@qq.com");
                        return user;
                    }
                });
    System.out.println(userCache.get("hetao"));
}
```

知乎 @愤怒的小吹球

```
@Test
public void testCallableCache()throws Exception{
    Cache<String, String> cache = CacheBuilder.newBuilder().maximumSize(1000).build();
    String resultVal = cache.get( k: "hetao", new Callable<String>() {
        public String call() {
            String strProValue="hello "+"hetao"+"!";
            return strProValue;
        }
    });
    System.out.println("hetao value : " + resultVal);
}
```

知乎 @愤怒的小吹球

get的时候指定key，这就告诉我们一个道理：

> *在使用缓存前，拍拍自己的四两胸肌，问自己一个问题：有没有【默认方法】来加载或计算与键关联的值？如果有的话，你应当使用CacheLoader。如果没有，或者想要覆盖默认的加载运算。你应该在调用get时传入一个Callable实例。----沃*自己硕德*

2 添加 ,插入key

get : 要么返回已经缓存的值，要么使用CacheLoader向缓存原子地加载新值；

getUnchecked：CacheLoader 会抛异常，定义的CacheLoader没有声明任何检查型异常，则可以 getUnchecked 查找缓存；**反之不能；**

getAll ： 方法用来执行批量查询；

put : 向缓存显式插入值，Cache.asMap()也能修改值，但不具原子性；

getIfPresent ： 该方法只是简单的把Guava Cache当作Map的替代品，不执行load方法；

3 清除 key

guava cache 自带 清除机制，但仍旧可以手动清除：

个别清除：Cache.invalidate(key)

批量清除：Cache.invalidateAll(keys)

清除所有缓存项：Cache.invalidateAll()

4 refresh和expire刷新机制

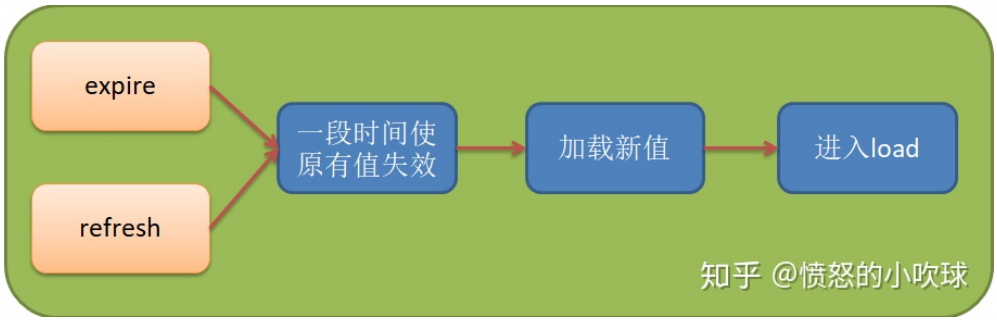
expireAfterAccess(30, TimeUnit.*MINUTES*) // 30min内没有被读或写就会被回收；

expireAfterWrite(30, TimeUnit.*MINUTES*) // 30min内没有没有更新就会被回收：

refreshAfterAccess(30, TimeUnit.*MINUTES*) //上一次更新操作30min后再刷新；

注意这里面精准的措辞，**“30min内没有被读或写就会被回收” 不等于 “30min内会被回收”，因为真正的过期/刷新操作是在key被读或写时发生的。**

先用一张图说明expire 与refresh的大体过程：



从图上容易看出，每次只有找不到值都会进入load方法，换句话说，只有发生“取值”操作，才会**个线程操作执行**

expire 在load 阶段——同步机制：当前线程load未完成，其他线程呈阻塞状态，待当前线程load完成，其他线程均需进行”获得锁--获得值--释放锁“的过程。这种方法会让性能有一定的损耗。

refresh 在load阶段——异步机制：当前线程load未完成，其他线程仍可以取原来的值，等当前线程load完成后，下次某线程再取值时，会判断系统时间间隔是否时超过设定refresh时间，来决定是否设定新值。所以，refresh机制的特点是，**设定30分钟刷新，30min后并不一定就是立马就能保证取到新值。**

那么问题来了，expire与refresh 都能控制key得回收，究竟如何选择?

答案是，**两个一起来!**

能够想象，只要refresh得时间小于expire时间，就能保证多线程在load取值时不阻塞，也能保证refresh时间到期后，取旧值向新值得平滑过渡，当然，**仍旧不能解决取到旧值得问题。**

5 监听

在guava cache中移除key可以设置相应得监听操作，以便key被移除时做一些额外操作。缓存项被移除时，RemovalListener会获取移除通知[RemovalNotification]，其中包含移除原因[RemovalCause]、键和值。监听有同步监听和异步监听两种：

同步监听

默认情况下，监听器方法是被同步调用的（在移除缓存的那个线程中执行）,执行清理key的操作与执行监听是单线程模式，当然监听器中抛出的任何异常都不会影响其正常使用，顶多把异常写到日记了。同步监听示例：

```
@Test
public void testCacheRemovedNotification() {
    CacheLoader<String, String> loader = CacheLoader.from(String::toUpperCase);
    RemovalListener<String, String> listener = notification ->
    {
        if (notification.wasEvicted()) {
            RemovalCause cause = notification.getCause();
            System.out.println("remove cacase is :" + cause.toString());
            System.out.println("key:" + notification.getKey() + "value:" + notification.getValue());
        }
    };
    LoadingCache<String, String> cache = CacheBuilder.newBuilder()
            .maximumSize(4)
            .removalListener(listener)// 添加删除监听
            .build(loader);
    cache.getUnchecked( k: "wangji");
    cache.getUnchecked( k: "wangwang");
    cache.getUnchecked( k: "guava");
    cache.getUnchecked( k: "test");
    cache.getUnchecked( k: "test1");
}
```

此处cache容量是4，超过容量，自动清除，监听程序跟清除是单线程模式

异步监听

试想这么一种情景，假如在同步监听模式下，**监听方法中的逻辑特别复杂，执行效率慢，那此时如果有大量的key进行清理，会使整个缓存性能变得很低下**，所以此时适合用异步监听，移除key与监听key的移除分属2个线程。异步监听示例：

6 统计

guava cache还有一些其他特性，比如weight 按权重回收资源，统计等，这里列出统计。**CacheBuilder.recordStats()用来开启Guava Cache的统计功能。**统计打开后Cache.stats()方法返回如下统计信息：

- hitRate()：缓存命中率；
- hitMiss(): 缓存失误率；
- loadcount() ; 加载次数；
- averageLoadPenalty()：加载新值的平均时间，单位为纳秒；
- evictionCount()：缓存项被回收的总数，不包括显式清除。

唯一值得注意的一点是：**当通过asmap（）方法查询key时，stat项是不作任何变化的**，修改值时会有影响。此外，还有其他很多统计信息。这些统计信息对于调整缓存设置是至关重要的，在性能监控时可以依据的重要指标。

发布于 2018-09-13 11:02

缓存 Guava Redis

文章被以下专栏收录

风中饮酒的程序员
有趣的互联网应用，全新的技术实践与总结

Guava Cache 原理分析与最佳实践

阿里巴巴大... 发表于淘系后端：...

万字详解本地缓存之王 Caffeine 的高性能设计之道！

里奥ii 发表于Java学...

一、 什么是缓存雪崩?如何避免和解决？当缓存服务器重启或者大量缓存集中在某一个时间段失效，这样在失效的时候，会给后端系统带来很大压力。导致系统崩溃。 避免方法： 1.在缓存失效后，通...

程序猿

1 条评论 切换为时间排序

写下你的评论...

52Hz 2019-07-16

可以可以 收藏了

赞