

## 舞羊

当你的才华还撑不起你的野心时，你就应该静下心来学习。

博客园

首页

新随笔

联系

订阅

管理

随笔 - 19 文章 - 228 评论 - 5 阅读 - 51万

GitHub: [Ucky](#)

Email: wynjauu@163.com

昵称: 舞羊

园龄: 7年8个月

粉丝: 14

关注: 3

+加关注

## 搜索

找找看

谷歌搜索

## 我的标签

Java并发编程(6)

JVM性能(6)

Devops(5)

tools(2)

Date Handle(1)

overview(1)

## 积分与排名

积分 - 228616

排名 - 3892

## 文章分类

Antlr4(2)

CAS(14)

clickhouse(3)

Druid(7)

Elasticsearch(11)

Flink(11)

Flume(1)

Freemarker(2)

Front(5)

## Guava Cache 数据变化实现回调的监听器RemovalListener

当我们需要在缓存被移除的时候，得到通知产生回调，并做一些额外处理工作。这个时候RemovalListener就派上用场了。



```
public class Main {

    // 创建一个监听器
    private static class MyRemovalListener implements RemovalListener<Integer, Integer> {
        @Override
        public void onRemoval(RemovalNotification<Integer, Integer> notification) {
            String tips = String.format("key=%s,value=%s,reason=%s", notification.getKey(), notification.getValue(), notification.getReason());
            System.out.println(tips);
        }
    }

    public static void main(String[] args) {

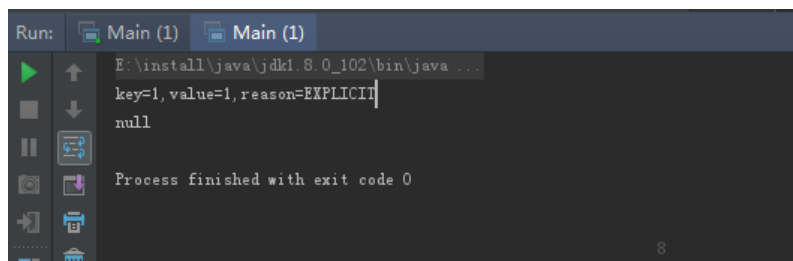
        // 创建一个带有RemovalListener监听的缓存
        Cache<Integer, Integer> cache = CacheBuilder.newBuilder().removalListener(new MyRemovalListener()).build();

        cache.put(1, 1);

        // 手动清除
        cache.invalidate(1);

        System.out.println(cache.getIfPresent(1)); // null
    }
}
```

使用invalidate()清除缓存数据之后，注册的回调被触发了



下面是只有主动删除数据使用的回调



```
public class CacheConnection {

    public static RemovalListener<String, Connection> myRemovalListener = new RemovalListener<String, Connection>() {
        @Override
        public void onRemoval(RemovalNotification<String, Connection> notification) {
            String tips = String.format("key=%s,value=%s,reason=%s in myRemovalListener", notification.getKey(), notification.getValue(), notification.getReason());
            System.out.println(tips);
            //when expireAfterAccess to do
        }
    }
}
```

Git(15)
Guava(3)
Guice(4)
Hessian(3)
HTTP(6)
Idea(18)
更多

## 最新评论

1. Re:jsonpath - 使用 JSONPath 解析 JS  
ON 完整内容详解

您好, 想请问一下, 怎么用C++引入  
jsonpath, 一直没找到, 觉得jsonp  
ath处理json很方便

--junluocode

2. Re:mybatis怎么实现对象参数和注解参  
数同时传入

哪个傻狗反对的, 明明很有效

--眼熟法拉瑟

3. Re:idea中修改git提交代码的用户名

111111

--yuetoon

4. Re:druid.io实践---实现近似直方图和分  
位数

```json{ "error": "Unknown excepti  
on", "errorMessage": "Incompatible  
type for metric[mem\_usage], exp...

--川川籽

5. Re:ANTLR4在windows上的安装(java  
版)

写的很好, 很有用

--石恩升

```
if (notification.getCause().equals("EXPIRED") && notification.getValue() != null) {
    try {
        notification.getValue().close();
    } catch (SQLException e) {
        System.out.printf("Exception in myRemovalListener:\n");
        e.printStackTrace();
    }

    System.out.printf("Remove %s in cacheConnection", notification.getKey());
}

};
```

```
public static Cache<String, Connection> cacheConnection = CacheBuilder.newBuilder()
    //设置cache的初始大小为20000, 要合理设置该值
    .initialCapacity(20000)
    //设置并发数为5, 即同一时间最多只能有5个线程往cache执行写入操作
    .concurrencyLevel(100)
    //设置cache中的数据在600秒没有被读写将自动删除
    .expireAfterAccess(600, TimeUnit.SECONDS)
    //设置监听, 当出现自动删除时的回调
    .removalListener(myRemovalListener)
    //构建cache实例
    .build();
```

```
public static Connection getCache(String key) {
    try {
        Connection var = cacheConnection.getIfPresent(key);
        return var;
    } catch (Exception e) {
        // TODO: handle exception
        System.out.println("the value of cacheConnection is null");
        e.printStackTrace();
        return null;
    }
}
```

```
public static void putCache(String key, Connection value) {
    cacheConnection.put(key, value);
}
```



RemovalNotification中包含了缓存的key、value以及被移除的原因RemovalCause。通过源码可以看出, 移除原因与容量管理方式是相对应的。下面是具体的消息

```
public enum RemovalCause {
    /**
     * The entry was manually removed by the user. This can result from the user invoking
     * {@link Cache#invalidate}, {@link Cache#invalidateAll(Iterable)}, {@link Cache#invalidateAll()},
     * {@link Map#remove}, {@link ConcurrentMap#remove}, or {@link Iterator#remove}.
     */
    EXPLICIT {
        @Override
        boolean wasEvicted() {
            return false;
        }
    },

    /**
     * The entry itself was not actually removed, but its value was replaced by the user. This can
     * result from the user invoking {@link Cache#put}, {@link LoadingCache#refresh}, {@link Map#put},
     * {@link Map#putAll}, {@link ConcurrentMap#replace(Object, Object)}, or
     * {@link ConcurrentMap#replace(Object, Object, Object)}.
     */
    REPLACED {
        @Override
        boolean wasEvicted() {
            return false;
        }
    },
}
```

```

/**
 * The entry was removed automatically because its key or value was garbage-collected. This
 * can occur when using {@link CacheBuilder#weakKeys}, {@link CacheBuilder#weakValues}, or
 * {@link CacheBuilder#softValues}.
 */
COLLECTED {
    @Override
    boolean wasEvicted() {
        return true;
    }
},

/**
 * The entry's expiration timestamp has passed. This can occur when using
 * {@link CacheBuilder#expireAfterWrite} or {@link CacheBuilder#expireAfterAccess}.
 */
EXPIRED {
    @Override
    boolean wasEvicted() {
        return true;
    }
},

/**
 * The entry was evicted due to size constraints. This can occur when using
 * {@link CacheBuilder#maximumSize} or {@link CacheBuilder#maximumWeight}.
 */
SIZE {
    @Override
    boolean wasEvicted() {
        return true;
    }
};

/**
 * Returns {@code true} if there was an automatic removal due to eviction (the cause is neither
 * {@link #EXPLICIT} nor {@link #REPLACED}).
 */
abstract boolean wasEvicted();
}

```

监听器使用很简单，有几个特点需要注意下：

1、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）。如果监听器方法比较耗时，会导致调用者线程阻塞时间变长。下面这段代码，由于监听器执行需要2s，所以main线程调用invalidate()要2s后才能返回。

```

public class Main {

    // 创建一个监听器
    private static class MyRemovalListener implements RemovalListener<Integer, Integer> {
        @Override
        public void onRemoval(RemovalNotification<Integer, Integer> notification) {
            String tips = String.format("key=%s,value=%s,reason=%s", notification.getKey(), notification.getValue(), notification.getReason());
            System.out.println(tips);

            try {
                // 模拟耗时
                Thread.sleep(2000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    public static void main(String[] args) {

        // 创建一个带有RemovalListener监听的缓存
        final Cache<Integer, Integer> cache = CacheBuilder.newBuilder().removalListener(new MyRemovalListener()).build();
        cache.put(1, 1);
    }
}

```

```

cache.put(2, 2);

System.out.println("main...begin.");
cache.invalidate(1);// 耗时2s
System.out.println("main...over.");
}
}

```

解决这个问题的方法是：使用异步监听`RemovalListeners.asynchronous(RemovalListener, Executor)`。

```

public class Main {

    // 创建一个监听器
    private static class MyRemovalListener implements RemovalListener<Integer, Integer> {
        @Override
        public void onRemoval(RemovalNotification<Integer, Integer> notification) {
            String tips = String.format("key=%s,value=%s,reason=%s", notification.getKey(), notification.getVal
            System.out.println(tips);

            try {
                // 模拟耗时
                Thread.sleep(2000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    public static void main(String[] args) {

        RemovalListener<Integer, Integer> async = RemovalListeners.asynchronous(new MyRemovalListener(
        // 创建一个带有RemovalListener监听的缓存
        final Cache<Integer, Integer> cache = CacheBuilder.newBuilder().removalListener(async).build();
        cache.put(1, 1);
        cache.put(2, 2);

        System.out.println("main...begin.");
        cache.invalidate(1);// main线程立刻返回
        System.out.println("main...over.");
    }
}

```

2、创建cache的时候只能添加1个监听器，这个监听器对象会被多个线程共享，所以如果监听器需要操作共享资源，那么一定要做好同步控制。下面这段代码可以看出：2个线程会交替执行监听器的发方法。

```

public class Main {

    // 创建一个监听器
    private static class MyRemovalListener implements RemovalListener<Integer, Integer> {
        @Override
        public void onRemoval(RemovalNotification<Integer, Integer> notification) {
            String tips = String.format("key=%s,value=%s,reason=%s", notification.getKey(), notification.getVal
            System.out.println(tips);

            try {
                // 模拟耗时
                Thread.sleep(2000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            System.out.println("process over.");
        }
    }
}

```

```

    }

    public static void main(String[] args) {

        // 创建一个带有RemovalListener监听的缓存
        final Cache<Integer, Integer> cache = CacheBuilder.newBuilder().removalListener(new MyRemovalList
        cache.put(1, 1);
        cache.put(2, 2);

        new Thread(new Runnable() {
            @Override
            public void run() {
                System.out.println("thread1...trigger RemovalListener begin.");
                cache.invalidate(1);
                System.out.println("thread1...trigger RemovalListener over.");
            }
        }).start();

        new Thread(new Runnable() {
            @Override
            public void run() {
                System.out.println("thread2...trigger RemovalListener begin.");
                cache.invalidate(2);
                System.out.println("thread2...trigger RemovalListener over.");
            }
        }).start();
    }
}

```

```

Run Main (1)
E:\install\java\jdk1.8.0_102\bin\java ...
thread1... trigger RemovalListener begin.
thread2... trigger RemovalListener begin.
key=2, value=2, reason=EXPLICIT
key=1, value=1, reason=EXPLICIT
process over.
thread1... trigger RemovalListener over.
process over.
thread2... trigger RemovalListener over.

Process finished with exit code 0

```

3、监听器中抛出的任何异常，在被记录到日志后，会被guava丢弃，不会导致监听器不可用。下面这段代码可以看到：监听器中抛出的异常只是被记录了(打印到了控制台)，并没有导致JVM退出，之后缓存被移除一样可以再次触发。

```

public class Main {

    // 创建一个监听器
    private static class MyRemovalListener implements RemovalListener<Integer, Integer> {
        @Override
        public void onRemoval(RemovalNotification<Integer, Integer> notification) {
            String tips = String.format("key=%s,value=%s,reason=%s", notification.getKey(), notification.getVal
            System.out.println(tips);

            throw new RuntimeException();
        }
    }

    public static void main(String[] args) {

        // 创建一个带有RemovalListener监听的缓存
        final Cache<Integer, Integer> cache = CacheBuilder.newBuilder().removalListener(new MyRemovalList
        cache.put(1, 1);
        cache.put(2, 2);

        cache.invalidate(1);
        cache.invalidate(2);
    }
}

```

```
}  
十一月 11, 2016 12:49:23 下午 com.google.common.cache.LocalCache processPendingNotifications  
key=1,value=1,reason=EXPLICIT  
警告: Exception thrown by removal listener  
key=2,value=2,reason=EXPLICIT  
java.lang.RuntimeException  
    at net.aty.guava.Main$MyRemovalListener.onRemoval(Main.java:27)  
    at com.google.common.cache.LocalCache.processPendingNotifications(LocalCache.java:1960)  
    at com.google.common.cache.LocalCache$Segment.runUnlockedCleanup(LocalCache.java:3475)  
    at com.google.common.cache.LocalCache$Segment.postWriteCleanup(LocalCache.java:3451)  
    at com.google.common.cache.LocalCache$Segment.remove(LocalCache.java:3122)  
    at com.google.common.cache.LocalCache.remove(LocalCache.java:4188)  
    at com.google.common.cache.LocalCache$LocalManualCache.invalidate(LocalCache.java:4816)  
    at net.aty.guava.Main.main(Main.java:38) <5 internal calls> 8
```

学习: <https://blog.csdn.net/zhangjikian/article/details/76408578>

分类: [Guava](#)

好文要顶 关注我 收藏该文 微博 微信



舞羊  
关注 - 3  
粉丝 - 14  
[+加关注](#)

0 0

posted @ 2018-07-12 20:45 舞羊 阅读(1075) 评论(0) 编辑 收藏 举报

[刷新评论](#) [刷新页面](#) [返回顶部](#)

登录后才能查看或发表评论, 立即 [登录](#) 或者 [逛逛](#) 博客园首页

编辑推荐:

- 从技术体系到商业洞察, 中小研发团队架构实践之收尾篇
- 为什么 HttpContextAccessor 要这么设计?
- 【大话云原生】负载均衡篇-小饭馆客流量变大了
- 编程为什么那么难
- 【大话云原生】煮饺子与docker、kubernetes之间的关系

最新新闻:

- 马斯克收购推特, 7000名员工怀揣问号
  - 视频会员费还要涨多高?
  - 重磅! 曝马斯克正在招聘手机专家 或直接叫板苹果
  - 港股破发, 知乎商业化究竟取悦了谁?
  - 裁员、改制、电商梦, 估值200亿的小红书挺不起“腰杆”
- » 更多新闻...