工具篇：介绍几个好用的guava工具类

潜行前行 Lv3
2021年06月16日 09:47 · 阅读 2025

关注

前言

平时我们都会封装一些处理缓存或其他的小工具。但每个人都封装一次，重复造轮子，有点费时间。有没有一些好的工具库推荐-guava。guava是谷歌基于java封装好的开源库，它的性能、实用性，比我们自己造的轮子更好，毕竟谷歌出品，下面介绍下几个常用的guava工具类

- LoadingCache（本地缓存）
- Multimap 和 Multiset
- BiMap
- Table（表）
- Sets和Maps（交并差）
- EventBus（事件）
- StopWatch（秒表）
- Files（文件操作）
- RateLimiter（限流器）

「guava的maven配置引入」

mvn 复制代码

```
<dependency>
    <groupId>com.google.guava</groupId>
    <artifactId>guava</artifactId>
    <version>27.0-jre</version>
</dependency>
```

「LoadingCache」

- LoadingCache 在实际场景中有着非常广泛的使用，通常情况下如果遇到需要大量时间计算或者缓存值的场景，就应当将值保存到缓存中。LoadingCache 和 ConcurrentMap 类似，但又不尽相同。最大的不同是 ConcurrentMap 会永久的存储所有的元素值直到他们被显示的移除，但是 LoadingCache 会为了保持内存使用合理会根据配置自动将过期值移除
- 通常情况下，Guava caching 适用于以下场景：
  - 花费一些内存来换取速度
  - 一些 key 会被不止一次被调用
  - 缓存内容有限，不会超过内存空间的值，Guava caches 不会存储内容到文件或者到服务器外部，如果有此类需求考虑使用 Memcached, Redis
- LoadingCache 不能缓存 null key
- CacheBuilder 构造 LoadingCache 参数介绍

initialCapacity(int initialCapacity)	缓存池的初始大小
concurrencyLevel(int concurrencyLevel)	设置并发数
maximumSize(long maximumSize)	缓存池大小，在缓存项接近该大小时， Guava开始回收旧的缓存项
weakValues()	设置value的存储引用是虚引用
softValues()	设置value的存储引用是软引用
expireAfterWrite(long duration, TimeUnit unit)	设置时间对象没有被写则对象从内存中删除(在另外的线程里面不定期维护)
expireAfterAccess(long duration, TimeUnit unit)	设置时间对象没有被读/写访问则对象从内存中删除(在另外的线程里面不定期维护)
refreshAfterWrite(long duration, TimeUnit unit)	和expireAfterWrite类似，不过不立马移除key，而是在下次更新时刷新，这段时间可能会返回旧值
removalListener(RemovalListener<? super K1, ? super V1> listener)	监听器，缓存项被移除时会触发
build(CacheLoader<? super K1, V1> loader)	当数据不存在时，则使用loader加载数据

- LoadingCache V get(K key)，获取缓存值，如果键不存在值，将调用CacheLoader的load方法加载新值到该键中
- 示例

java 复制代码

```
LoadingCache<Integer,Long> cacheMap = CacheBuilder.newBuilder().initialCapacity(10)
    .concurrencyLevel(10)
    .expireAfterAccess(Duration.ofSeconds(10))
    .weakValues()
    .recordStats()
    .removalListener(new RemovalListener<Integer,Long>(){
        @Override
        public void onRemoval(RemovalNotification<Integer, Long> notification) {
            System.out.println(notification.getValue());
        }
    })
    .build(new CacheLoader<Integer,Long>(){
```

```
});
cacheMap.get(1);
```

「Multimap 和 MultiSet」

- Multimap的特点其实就是可以包含有几个重复Key的value，可以put进入多个不同value但是相同的key，但是又不会覆盖前面的内容
- 示例

java 复制代码

```
//Multimap: key-value  key可以重复，value也可重复
Multimap<String, String> multimap = ArrayListMultimap.create();
multimap.put("csc","1");
multimap.put("lwl","1");
multimap.put("csc","1");
multimap.put("lwl","one");
System.out.println(multimap.get("csc"));
System.out.println(multimap.get("lwl"));
--------------------------
[1, 1]
[1, one]
```

- MultiSet 有一个相对有用的场景，就是跟踪每种对象的数量，所以可以用来进行数量统计
- 示例

java 复制代码

```
//MultiSet: 无序+可重复   count()方法获取单词的次数  增强了可读性+操作简单
Multiset<String> set = HashMultiset.create();
set.add("csc");
set.add("lwl");
set.add("csc");
System.out.println(set.size());
System.out.println(set.count("csc"));
--------------------------
3
2
```

「BiMap」

- BiMap的键必须唯一，值也必须唯一，可以实现value和key互转
- 示例

```
biMap.put(2,"csc");
BiMap<String, Integer> map = biMap.inverse(); // value和key互转
map.forEach((v, k) -> System.out.println(v + "-" + k));
```

「Table」

- Table<R,C,V> table = HashBasedTable.create();，由泛型可以看出，table由双主键R（行）,C（列）共同决定，V是存储值
- 新增数据： table.put(R,C,V)
- 获取数据： V v = table.get(R,C)
- 遍历数据: Set<R> set = table.rowKeySet(); Set<C> set = table.columnKeySet();
- 示例

java 复制代码

```
// 双键的Map Map--> Table-->rowKey+columnKey+value
Table<String, String, Integer> tables = HashBasedTable.create();
tables.put("csc", "lwl", 1);
//row+column对应的value
System.out.println(tables.get("csc","lwl"));
```

「Sets和Maps」

java 复制代码

```
// 不可变集合的创建
ImmutableList<String> iList = ImmutableList.of("csc", "lwl");
ImmutableSet<String> iSet = ImmutableSet.of("csc", "lwl");
ImmutableMap<String, String> iMap = ImmutableMap.of("csc", "hello", "lwl", "world");
```

» set的交集, 并集, 差集

java 复制代码

```
HashSet setA = newHashSet(1, 2, 3, 4, 5);
HashSet setB = newHashSet(4, 5, 6, 7, 8);
//并集
SetView union = Sets.union(setA, setB);
//差集 setA-setB
SetView difference = Sets.difference(setA, setB);
//交集
SetView intersection = Sets.intersection(setA, setB);
```

java 复制代码

```
HashMap<String, Integer> mapA = Maps.newHashMap();
mapA.put("a", 1);mapA.put("b", 2);mapA.put("c", 3);
HashMap<String, Integer> mapB = Maps.newHashMap();
mapB.put("b", 20);mapB.put("c", 3);mapB.put("d", 4);
MapDifference<String, Integer> mapDifference = Maps.difference(mapA, mapB);
//mapA 和 mapB 相同的 entry
System.out.println(mapDifference.entriesInCommon());
//mapA 和 mapB key相同的value不同的 entry
System.out.println(mapDifference.entriesDiffering());
//只存在 mapA 的 entry
System.out.println(mapDifference.entriesOnlyOnLeft());
//只存在 mapB 的 entry
System.out.println(mapDifference.entriesOnlyOnRight());;
----------结果----------
{c=3}
{b=(2, 20)}
{a=1}
{d=4}
```

「EventBus」

- EventBus是Guava的事件处理机制，是设计模式中的观察者模式（生产/消费者编程模型）的优雅实现。对于事件监听和发布订阅模式
- EventBus内部实现原理不复杂，EventBus内部会维护一个Multimap<Class<?>, Subscriber> map，key就代表消息对应的类(不同消息不同类，区分不同的消息)、value是一个Subscriber，Subscriber其实就是对应消息处理者。如果有消息发布就去这个map里面找到这个消息对应的Subscriber去执行
- 使用示例

java 复制代码

```
@Data
@AllArgsConstructor
public class OrderMessage {
    String message;
}
//使用 @Subscribe 注解,表明使用dealWithEvent 方法处理 OrderMessage类型对应的消息
//可以注解多个方法,不同的方法 处理不同的对象消息
public class OrderEventListener {
    @Subscribe
    public void dealWithEvent(OrderMessage event) {
        System.out.println("内容：" + event.getMessage());
    }
}
```

```
// 发布消息
eventBus.post(new OrderMessage("csc"));
```

「StopWatch」

java 复制代码

```
Stopwatch stopwatch = Stopwatch.createStarted();
for(int i=0; i<100000; i++){
    // do some thing
}
long nanos = stopwatch.elapsed(TimeUnit.MILLISECONDS);
System.out.println("逻辑代码运行耗时："+nanos);
```

「Files文件操作」

- 数据写入

java 复制代码

```
File newFile = new File("D:/text.txt");
Files.write("this is a test".getBytes(), newFile);
//再次写入会把之前的内容冲掉
Files.write("csc".getBytes(), newFile);
//追加写
Files.append("lwl", newFile, Charset.defaultCharset());
```

- 文本数据读取

java 复制代码

```
File newFile = new File("E:/text.txt");
List<String> lines = Files.readLines(newFile, Charset.defaultCharset());
```

- 其他操作

Files.copy(File from, File to)	复制文件
Files.deleteDirectoryContents(File directory)	删除文件夹下的内容(包括文件与子文件夹)
Files.deleteRecursively(File file)	删除文件或者文件夹
Files.move(File from, File to)	移动文件
Files.touch(File file)	创建或者更新文件的时间戳
Files.getFileExtension(String file)	获得文件的扩展名
Files.getNameWithoutExtension(String file)	获得不带扩展名的文件名
Files.map(File file, MapMode mode)	获取内存映射buffer

「RateLimiter」

java 复制代码

```
//RateLimiter 构造方法，每秒限流permitsPerSecond
public static RateLimiter create(double permitsPerSecond)
//每秒限流 permitsPerSecond，warmupPeriod 则是数据初始预热时间，从第一次acquire 或 tryAcquire 执行开时计算
public static RateLimiter create(double permitsPerSecond, Duration warmupPeriod)
//获取一个令牌，阻塞，返回阻塞时间
public double acquire()
//获取 permits 个令牌，阻塞，返回阻塞时间
public double acquire(int permits)
//获取一个令牌，超时返回
public boolean tryAcquire(Duration timeout)
////获取 permits 个令牌，超时返回
public boolean tryAcquire(int permits, Duration timeout)
```

- 使用示例

java 复制代码

```
RateLimiter limiter = RateLimiter.create(2, 3, TimeUnit.SECONDS);
System.out.println("get one permit cost time: " + limiter.acquire(1) + "s");
System.out.println("get one permit cost time: " + limiter.acquire(1) + "s");
System.out.println("get one permit cost time: " + limiter.acquire(1) + "s");
System.out.println("get one permit cost time: " + limiter.acquire(1) + "s");
```

```
get one permit cost time: 0.0s
get one permit cost time: 1.331672s
get one permit cost time: 0.998392s
get one permit cost time: 0.666014s
get one permit cost time: 0.498514s
get one permit cost time: 0.498918s
get one permit cost time: 0.499151s
get one permit cost time: 0.488548s
```

- 因为RateLimiter滞后处理的，所以第一次无论取多少都是零秒
- 可以看到前四次的acquire，花了三秒时间去预热数据，在第五次到第八次的acquire耗时趋于平滑

「Guava Retry」

- maven引入

mvn 复制代码

```
<dependency>
  <groupId>com.github.rholder</groupId>
  <artifactId>guava-retrying</artifactId>
  <version>2.0.0</version>
</dependency>
```

- RetryerBuilder 构造方法

withRetryListener	重试监听器
withWaitStrategy	失败后重试间隔时间
withStopStrategy	停止策略
withBlockStrategy	阻塞策略BlockStrategy
withAttemptTimeLimiter	执行时间限制策略
retryIfException	发生异常，则重试
retryIfRuntimeException	发生RuntimeException异常，则重试
retryIfExceptionOfType(Class<? extends Throwable> ex)	发生ex异常，则重试
retryIfException(Predicate<Throwable> exceptionPredicate)	对异常判断，是否重试
retryIfResult(Predicate<V> resultPredicate)	对返回结果判断，是否重试

java 复制代码

```
Retryer<Boolean> retryer = RetryerBuilder.<Boolean>newBuilder()
    .retryIfException()
    .retryIfResult(Predicates.equalTo(false))
    .withAttemptTimeLimiter(AttemptTimeLimiters.fixedTimeLimit(1, TimeUnit.SECONDS))
    .withStopStrategy(StopStrategies.stopAfterAttempt(5))
    .build();
//Retryer调用
retryer.call(() -> true);
```

- spring也有对应的重试机制，相关文章可以看看重试框架Guava-Retry和spring-Retry

「欢迎指正文中错误（故事纯属虚构，如有雷同纯属巧合）」

参考文章

- Google guava工具类的介绍和使用

文章被收录于专栏：

随手笔记
寻找过往

关注专栏

评论

输入评论（Enter换行，Ctrl + Enter发送）

全部评论 3

最新 最热

chores 10月前
榨油
点赞 回复

轩晨 10月前
我思故我在
点赞 回复

士兵王 10月前
666
点赞 回复

相关推荐

cjinhuo 6月前 前端 Mac

优秀的后端应该有哪些开发习惯?

3.9w 481 187

Java小咖秀 2月前 开源 后端 Java

12 个适合做外包项目的开源后台管理系统

2.3w 178 22

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

前端鬼哥 1年前 JavaScript

推荐11款开发中超实用的工具

6011 185 10

前端阿飞 3月前 前端 JavaScript

10个常用的JS工具库，80%的项目都在用！

2.7w 1005 63

小学生study 3月前 Vue.js JavaScript

新一代状态管理工具，Pinia.js 上手指南

3.7w 699 170

沉默王二 1年前 Java 后端

Guava - 拯救垃圾代码，写出优雅高效，效率提升N倍

7816 75 7

MacroZheng 4月前 Java 后端

再见 Xshell！这款开源的终端工具逼格更高！

2.6w 195 46

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?

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

5.4w 236 98

树酱 1年前 前端

推荐几个数据大屏可视化开发工具

1.9w 311 42

苏世_ 5月前 后端 Java

新来的同事问我where 1=1 是什么意思

2.9w 83 55

楼下小黑哥 1年前 Java

求求你了，不要再自己实现这些逻辑了，开源工具类不香吗?

3.7w 526 67

JavaGuide 3月前 后端 Java

豆瓣 9.7！2022 值得一读的 15 本技术书籍！

3.5w 241 18

知否技术 4月前 后端 Java

在 IDEA 中使用 Debug，简直太爽了！

2.2w 252 56

程序新视界 4月前 Java 安全 后端

Log4j史诗级漏洞，我们这些小公司能做些什么?

3.4w 79 37

码匠笔记 7月前 后端 Java

自己工作中一直接触不到高并发、分布式怎么办?

2.9w 126 33

crossoverJie 5月前 Go

Go 日常开发常备第三方库和工具

装了这几个IDEA插件，基本上一站式开发了！

2.2w 302 22

CLSugger 3年前 测试

简单分享常用抓包工具：fillder

750 点赞 评论

捡田螺的小男孩 7月前 Java 后端

2W字！详解20道Redis经典面试题！（珍藏版）

2.3w 492 37

MacroZheng 4月前 Java 后端

为啥人家的命令行终端如此炫酷？原来用了这款137K+Star的神器！

2.9w 217 10

MacroZheng 28天前 后端 Java Spring Boot

神器 SpringDoc 横空出世！最适合 SpringBoot 的API文档工具来了！

2.2w 79 23

漫话编程 2年前 后端 Java CDN

漫话：如何给女朋友解释什么是CDN？

2.1w 293 26

小黑说Java 2月前 后端 Java

使用MyBatis拦截器后，摸鱼时间又长了。🐟

3.6w 247 48

沉默王二 1月前 后端 Java

两天两夜，1M图片优化到100kb！

2.3w 134 52

小圆脸儿 1年前 前端框架

前端页面可视化搭建工具业界的轮子

推荐 12 个学习前端必备的神仙级工具类项目与网站

2.6w 738 26

程序员依扬 2年前 面试 前端

【1 月最新】前端 100 问：能搞懂 80% 的请把简历给我

53.3w 9328 309

Java中文社群 3月前 后端 Spring Boot Java

Spring Boot Admin，贼好使！

2.2w 132 21

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

MacroZheng 4月前 后端 Java

取代 Postman + Swagger！这款神器功能更强，界面更炫酷！

2.4w 171 33

敖丙 3月前 后端 Redis Java

Redis为什么这么快?

2.8w 160 16

程序猿DD 4月前 后端 Java

今年你因为 YYYY-MM-dd 被锤了吗?

5.7w 264 48

拥抱心中的梦想 4年前 Spring 后端 单元测试

一起来谈谈 Spring AOP！

2.1w 214 10

sunshine小小倩 3年前 Chrome 前端 图片资源

Mac 提升开发效率的小工具

实战！聊聊工作中使用了哪些设计模式

2.4w 443 46

CodeFox 2月前 后端 Java

美团动态线程池实践思路，开源了

3.5w 470 23

iCourt 5年前 MySQL Redis 面试

[北京 年入25-45万] iCourt招聘：方向对了，速度才有意义

2.3w 96 61