

# BLS能力中心技术分享

---

朱文魁

# 分享内容

---

1. 内容回顾及补充
2. guava 简介
3. guava 常用功能实例说明
4. guava cache 原理
5. LocalDate 实例及原理



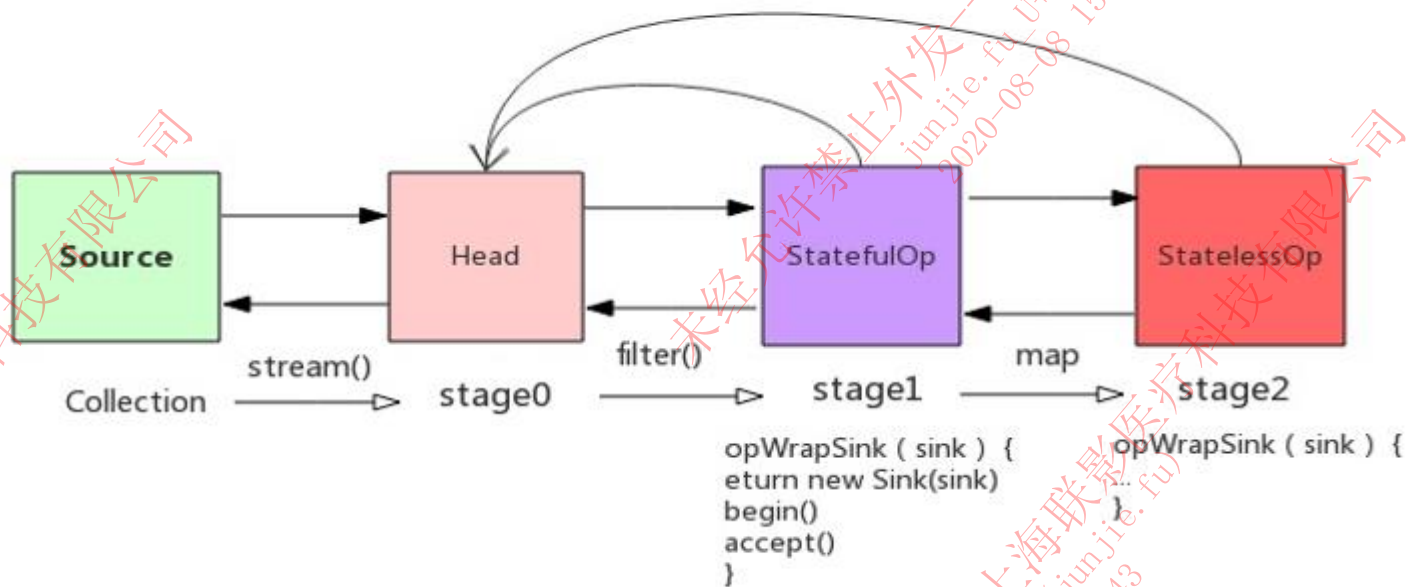
# 内容回顾

## 1. lambda表达式

语法/实例/函数式接口/streamApi及原理/原理的证明

## 2. 遗留作业

parallelStream 线程数 及证明方式？ ControlHelper tools& CatchUtils tools



# parallelStream并发原理

---

## 1. 实例效果展示

在10个数据 / 1w个数据 / 10w个数据下耗时演示

## 2. forkJoinTask

fork分配任务，join 聚合结果，工作窃取模式，适合cpu密集型任务

## 3. forkJoinPool 线程数大小

AbstractTask 中的 commonPool 初始化

# CatchUtil tools

---

## 1. 代码演示

## 2. 能带来什么

- 1). 可以和业务具体返回一起使用 简化 异常处理流程
- 2). 可以处理 check 异常，改完 手动 throw
- 3). 自动记录日志

## 3. U课堂使用实例

# Controller tools

---

## 1. 代码演示

## 2. 能带来什么

- 1). 可以自动使用Validator 进行参数验证
- 2). 可以扩展返回内容，如接口执行时间/监控/日志
- 3). 自动根据异常类型配置返回code等信息

## 3. U课堂使用实例

# Google guava

---

## 1. Google guava 是什么

Guava是一种基于开源的Java库，其中包含谷歌正在由他们很多项目使用的很多核心库。这个库是为了方便编码，并减少编码错误。这个库提供用于集合，缓存，支持原语，并发性，常见注解，字符串处理，I/O和验证的实用方法。

## 2. 能带来什么

- 1) 可以让你快乐编程，写出优雅的 Java 代码
- 2) 可靠，快速，有效的扩展JAVA标准库，让编码更加省心
- 3) guava是由google维护，代码经过高度优化

## 3. 现状

目前项目中还在重复造轮子，各个项目使用不统一，工具包不是最优解

# Guava 字符串处理

## 1. Guava 中的字符串处理 充分考虑了 null/空格等处理

工具类

Joiner

Spilter

CharMatcher

CaseFormat

程序名称和说明

实用加入对象，字符串等。

实用程序用来分割字符串。

实用的字符操作。

实用程序，用于改变字符串格式。

## 2. GuavaTest1 演示



# Guava 集合

## Guava 集合 基于开发者的应用开发经验，定制了许多先进的集合

- 1) 集合创建习惯性写法
- 2) 扩展集合

### 扩展集合

Multiset

Multimap

BiMap

Table

### 集合名称和说明

一个扩展来设置界面，允许重复的元素。

一个扩展来映射接口，以便其键可一次被映射到多个值

一个扩展来映射接口，支持反向操作。

表代表一个特殊的图，其中两个键可以在组合的方式被指定为单个值。

## 2. GuavaTest1 演示



# Guava cache

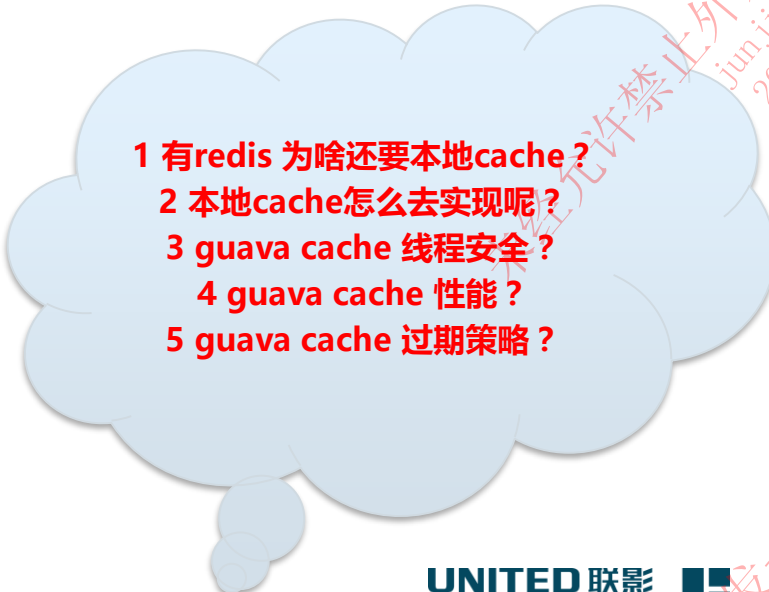
## 1 缓存

使用Cache时，我们优先读取缓存，当缓存不存在时，则从实际的数据存储获取，如DB、磁盘、网络等，即get-if-absent-compute。guava提供了CacheLoader机制，允许我们通过设置Loader来自动完成这一过程

## 2 接口说明

```
LoadingCache<String, Optional<Document>> documentCache =  
    CacheBuilder.newBuilder().maximumSize(100).expireAfterAccess(30, TimeUnit.MINUTES).expire  
AfterWrite(30, TimeUnit.MINUTES)  
    .build(new CacheLoader<String, Optional<Document>>() {  
        @Override  
        public Optional<Document> load(String key) {  
            return Optional.of(getFromDatabase(key));  
        }  
    });  
maximumSize // 缓存最大值  
expireAfterAccess 访问后多久过期  
expireAfterWrite 写入后多久过期
```

LoadingCache 不支持null 值，如果需要使用 Optional 处理

- 
- 1 有redis 为啥还要本地cache ?
  - 2 本地cache怎么去实现呢 ?
  - 3 guava cache 线程安全 ?
  - 4 guava cache 性能 ?
  - 5 guava cache 过期策略 ?

# Guava 缓存回收机制

## 1. 缓存回收，惰性回收机制

### 1) 基于容量回收

`CacheBuilder.maxmumSize(long)`

### 2) 基于时间回收

`CacheBuilder.expireAfterAccess()`

`CacheBuilder.expireAfterWrite()`

### 3) 基于引用回收

`CacheBuilder.weakKeys()`

### 4) 显示清除

个别清除: `Cache.invalidate(key)`

批量清除: `Cache.invalidateAll(keys)`

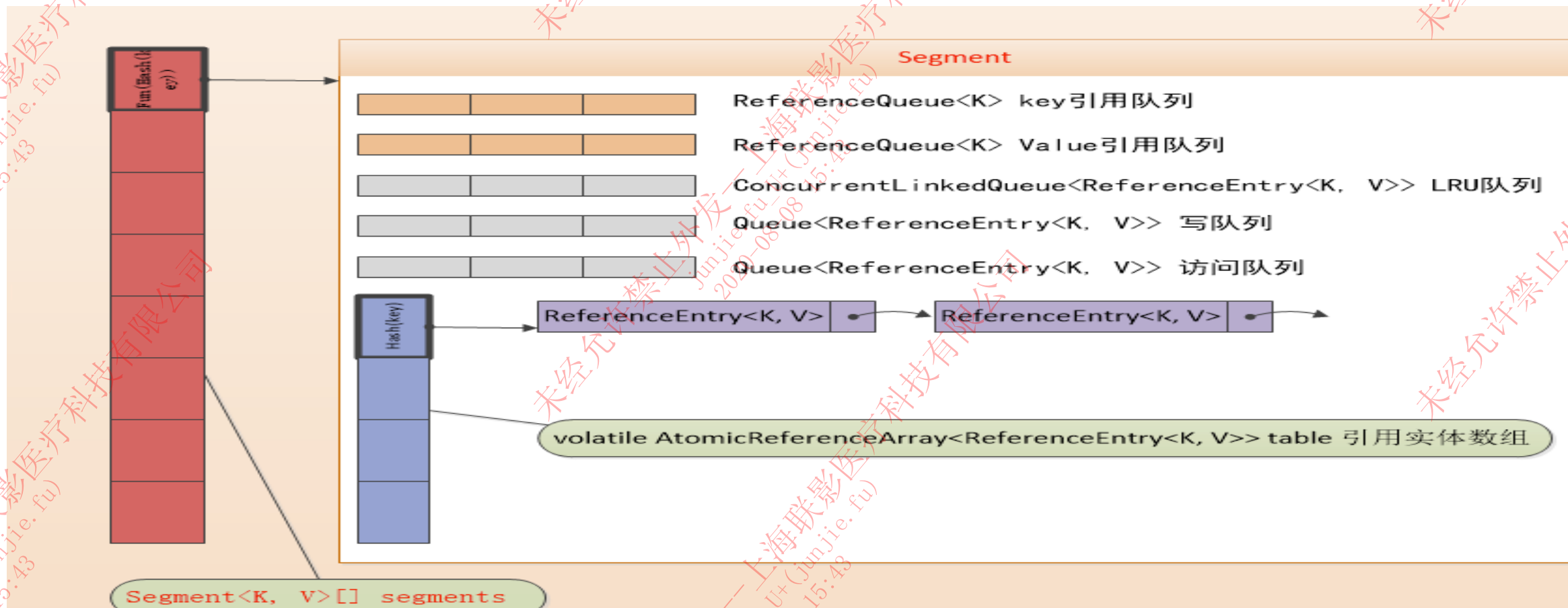
清除所有缓存项: `Cache.invalidateAll()`

## 2. 代码演示

# Guava cache 原理

## 1. 是ConcurrentHashMap一种类似实现

LocalCache<K, V> extends AbstractMap<K, V> implements ConcurrentMap 和 常用的 ConcurrentHashMap jdk1.7一致



# Guava cache 原理

## 1. 属性解析

名称	类型	作用
segments	Segment<k,V>[]	实现ReentrantLock锁，减少锁的粒度，提高并发度。好处：分段锁很好保证并发读写的效率，因此支持非阻塞的读和不同段之间的并发写
table	AtomicReferenceArray<ReferenceEntry<K,V>	存放键值对的地方
keyReferenceQueue	ReferenceQueue<K>	已经被GC，需要内部清理key引用队列
valueReferenceQueue	ReferenceQueue<V>	已经被GC，需要内部清理value引用队列
recencyQueue	Queue<ReferenceEntry<K,V>>	记录当entries被访问时，去更新accessQueue中顺序。在segment中当segment上限值或是写操作发生会去更新accessQueue顺序，同时清空recencyQueue。
writeQueue	Queue<ReferenceEntry<K,V>>	按照写入时间进行排序的元素队列，写入元素时会把它加入队列的队尾
accessQueue	Queue<ReferenceEntry<K,V>>	按照访问时间进行排序的元素队列，访问或是写入元素时会把它加入到队列的队尾。



# Guava cache 原理

## 1. Java 引用说明

`Object obj = new Object();`

`ReferenceQueue queue = new ReferenceQueue();`

强引用 (StrongReference) `list.add(obj)`

软引用 (SoftReference) `list.add(new SoftReference<>(obj, queue))`

弱引用 (WeakReference) `list.add(new WeakReference<>(obj, queue))`

虚引用 (PhantomReference) `list.add(new PhantomReference<Object>(obj, queue))`

在gc时候 obj 被回收, 然后Reference 放入 queue, 可自定义清除

# LocalDate

---

## 1. LocalDate 是什么

LocalDate、LocalTime、LocalDateTime 是java8中的新特性

## 2. 为什么要用LocalDate

- 1) 老版Date 格式化 SimpleDateFormat使用Calendar实例非线程安全，性能低
- 2) Date有的我都有，Date没有的我也有

## 3. LocalDate实例演示

## 4. 实现原理简介

LocalDateTime.now()解析

参考：

<https://www.cnblogs.com/lixinjie/p/java-8-time-system.html>

# 结论

---

## Q&A

