# 总纲

Guava

<https://github.com/google/guava>

http://outofmemory.cn/java/guava/cache/how-to-use-guava-cache

# 一、Caches

原理： 由segment构成（由并行度设置）；

segment底层是数组+链表的组成（数组长度由初始大小/并行度，但是是2的幂次方）

底层数组：AtomicReferenceArray<ReferenceEntry<K, V>>

先求hash，所有余数相同的在一个链表上。

## 1） Cache API

1 指定key，从缓存中查找

单个：get、getIfPresent

批量：getAll、getAllPresent

2 向缓存中存数据

单条：put

批量：putAll

3 缓存失效,即删除缓存

单条:invalidate(k k)

批量：invalidateAll(Keys)

全部：invalidateAll()

4 更新某个k的缓存

refresh(k k): 会重新调用load方法

## 2）CacheBuilder api

1 提供一个静态方法，newBuilder(),返回一个CacheBuilder对象,后续都是级联操作

2 设置缓存的过期时间：根据访问和写

expireAfterAccess(**long** duration, TimeUnit unit)

expireAfterWrite(**long** duration, TimeUnit unit)

最大条目的个数：接近这个值的时候就开始清除；当设置为0的时候，一放入就会被清理。

maximumSize(**long** size)

maximumWeight(**long** weight)

initialCapacity(**int** initialCapacity)： 初始容量,默认为16

concurrencyLevel(**int** concurrencyLevel)：设置并行度，同ConcurrentHashMap一样，会建立几个segment，默认是4.

[refreshAfterWrite(long, TimeUnit)](http://google.github.io/guava/releases/snapshot/api/docs/com/google/common/cache/CacheBuilder.html#refreshAfterWrite(long,%20java.util.concurrent.TimeUnit)): 定时刷新。

## 3） Population

插入k-v的几种方法：

### 1 From a CacheLoader

继承CacheLoader<k,v>，并在Build中设置这个对象。里面有两个方法load和loadAll

1 逐条

重写V load(K k)方法。

2 批量

重写批量方法Map<K, V> loadAll(Iterable<? **extends** K> keys) {}

### 2 From a Callable

[get(K, Callable<V>)](http://google.github.io/guava/releases/snapshot/api/docs/com/google/common/cache/Cache.html#get(java.lang.Object,java.util.concurrent.Callable))

### 3 直接插入

## 4）Eviction

### 1 Size-based Eviction

1 按照条数：

[CacheBuilder.maximumSize(long)](http://google.github.io/guava/releases/snapshot/api/docs/com/google/common/cache/CacheBuilder.html#maximumSize(long))： 超过后会按照LRU移除缓存项

2 按照“比重” ： 每条的比重由自己在函数中设计

[CacheBuilder.weigher(Weigher)](http://google.github.io/guava/releases/snapshot/api/docs/com/google/common/cache/CacheBuilder.html" \l "weigher(com.google.common.cache.Weigher))： 计算每条的重量

C[acheBuilder.maximumWeight(long)](http://google.github.io/guava/releases/snapshot/api/docs/com/google/common/cache/CacheBuilder.html#maximumWeight(long))：缓存的最大重量

### 2 Timed Eviction

根据写入或根据访问，设置过期时间

[expireAfterAccess(long, TimeUnit)](http://google.github.io/guava/releases/snapshot/api/docs/com/google/common/cache/CacheBuilder.html#expireAfterAccess(long,%20java.util.concurrent.TimeUnit))

[expireAfterWrite(long, TimeUnit)](http://google.github.io/guava/releases/snapshot/api/docs/com/google/common/cache/CacheBuilder.html#expireAfterWrite(long,%20java.util.concurrent.TimeUnit))

### 3 Reference-based Eviction

## 5）Explicit Removals

## 6）Romoval Listeners

可以设置删除监听器，监听器默认是同步的，会拉低cache的性能。可以设置为异步的。

## 7）When Does Cleanup Happen?

删除key是延迟的，并不是一过期就删除。并不像redis那样开一个专门的线程去检查过期时间并删除。原因是：删除key，线程需要竞争写锁，会带来性能影响。

Cleanup发生时机：当写时(已经竞争到写锁)，或读一个过期的key时

参考：<http://blog.csdn.net/guozebo/article/details/51590517>

可以自己手动通过[ScheduledExecutorService](http://docs.oracle.com/javase/1.5.0/docs/api/java/util/concurrent/ScheduledExecutorService.html)周期手动调用cleanup。

## 8）Refresh

1 refresh默认是同步的，CacheLoader中调用reload方法，再调用load方法。

2 在CacheLoader中，可以重写reload方法。支持同步和异步，参看官方文档。

3 C[acheBuilder.refreshAfterWrite(long, TimeUnit)](http://google.github.io/guava/releases/snapshot/api/docs/com/google/common/cache/CacheBuilder.html#refreshAfterWrite(long,%20java.util.concurrent.TimeUnit)).

## 9）Features

提供数据统计的api，如统计缓存命中率；和视图map

## 10）中断支持