Google Guava LoadingCache

缓存在很多场景下相当有用，例如计算或检索一个值的代价很高，并且对同样的输入需要不止一次获取值的时候，就应当考虑使用缓存。Guava中有cache包，提供本地内存缓存功能，还考虑很多其他的功能，包括：

并发问题

缓存失效问题

内存不够时释放缓存

缓存命中率

缓存的移除等。

其与Java HashMap最大的不同点在于get(key)时遇到缓存中没有value的处理方式上，HashMap直接返回null，但是Guava Cache会触发load方法（从定义的目的存储）获取值，流入进行热点数据查询时，如果缓存中没有，则去数据库中查询，并把查询到的结果保存到缓存中，如果数据库也没有，则抛出异常。

Guava Cache适用于消耗一些内存空间提升速度，某些键值会被查询一次以上，缓存中存放的数据总量不会超出内存容量等。LoadingCache是Guava中提供的一种缓存实现，在Hadoop的MR-HistoryServer中的使用如下：



当通过REST获取某个JobId的JobInfo时，从LoadingCache中获取，如果没有则通过HistoryFileManager对象中获取JobInfo信息（对应/mr-histroy/tmp及/done中的Job信息），源码如下：

|  |  |
| --- | --- |
| *CacheLoader<JobId, Job> loader;*  *loader = new CacheLoader<JobId, Job>() {*  *@Override*  *public Job load(JobId key) throws Exception {*  *return loadJob(key);}*  *};*  *if (!useLoadedTasksCache) {*  *loadedJobCache = CacheBuilder.newBuilder()*  *.maximumSize(loadedJobCacheSize)*  *.initialCapacity(loadedJobCacheSize)*  *.concurrencyLevel(1)*  *.build(loader);*  *}* | 其中loadJob，源码如下：  p*rivate Job loadJob(JobId jobId) throws RuntimeException, IOException {*  *HistoryFileInfo fileInfo;*  *fileInfo = hsManager.getFileInfo(jobId);*  *if (fileInfo == null) {*  *throw new HSFileRuntimeException*  *} else if (fileInfo.isDeleted()) {*  *throw new HSFileRuntimeException*  *} else {*  *return fileInfo.loadJob();*  *}*  *}* |

LoadingCache是通过CacheBuilder对象构建，下面是一些重要的参数介绍：

1. maximumSize(long size)： 配置缓存数量上限，达到上限时，处理时间最长没被访问的对象或者根据配置释放对象
2. [expireAfterWrite](https://google.github.io/guava/releases/19.0/api/docs/src-html/com/google/common/cache/CacheBuilder.html#line.603)(long duration,[TimeUnit](http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/TimeUnit.html?is-external=true) unit): 缓存项在给定时间内没有被写访问，则回收，如果认为缓存数据在固定时候就不可用，则这种回收方式可取
3. expireAfterAccess(long, TimeUnit): 缓存项在给定时间内没有被读写访问，则回收，如果认为缓存数据在固定时候就不可用，则这种回收方式可取
4. refreshAfterWrite(long duration, TimeUnit unit):定时刷新，可以为缓存增加自动定时刷新功能，通过定时刷新可以让缓存项保持可用。
5. removalListener ([RemovalListener](https://google.github.io/guava/releases/19.0/api/docs/com/google/common/cache/RemovalListener.html)<? super K1,? super V1> listener): 定义Listener事件，当触发后，将缓存项从缓存中移除。
6. [weigher](https://google.github.io/guava/releases/19.0/api/docs/src-html/com/google/common/cache/CacheBuilder.html#line.473)([Weigher](https://google.github.io/guava/releases/19.0/api/docs/com/google/common/cache/Weigher.html)<? super K1,? super V1> weigher)：定义缓存项的权重，根据权重回收缓存项。例如JobHistoryServer中的LoadingCache中定义的Weigher:

*Weigher<JobId, Job> weightByTasks;*

*weightByTasks = new Weigher<JobId, Job>() {*

*//权重为task的数目，Maps数目+Reduces数目*

*@Override*

*public int weigh(JobId key, Job value) {*

*int taskCount = Math.min(loadedTasksCacheSize,*

*value.getTotalMaps() + value.getTotalReduces());*

*return taskCount;*

*}*

*};*

*loadedJobCache = CacheBuilder.newBuilder()*

*.maximumWeight(loadedTasksCacheSize)*

*.weigher(weightByTasks)*

*.concurrencyLevel(1)*

*.build(loader);*

7）recordStats，开启缓存统计，可以通过hitRate, verageLoadPenalty,evicionCount等方法获取缓存的使用情况，默认不开启

备注：在Java8中，可以实现Loader的功能，其使用方式的对比如下：

|  |  |
| --- | --- |
| *LoadingCache<Key, Graph> graphs = CacheBuilder.newBuilder()*  *.build(*  *new CacheLoader<Key, Graph>() {*  *public Graph load(Key key) throws AnyException {*  *return createExpensiveGraph(key);*  *}*  *});* | *JDK >1.8*  *ConcurrentMap<Key, Graph> map = new ConcurrentHashMap<>();*  *map.computeIfAbsent(aKey, key -> createExpensiveGraph(key));* |