链滴

redis【删除策略 & 高级数据类型】

作者：haxLook

原文链接：https://ld246.com/article/1617266399532

来源网站：链滴

<h2 id="删除策略">删除策略</h2>
<h3 id="1-数据删除策略">1、数据删除策略</h3>

定时删除
惰性删除
定期删除

<h4 id="时效性数据的存储结构">时效性数据的存储结构</h4>

Redis 中的数据，在 expire 中以哈希的方式保存在其中。其 value 是数据在内存中的地址，filed 是对应的生命周期

<h4 id="数据删除策略的目标">数据删除策略的目标</h4>
在内存占用与 CPU 占用之间寻找一种平衡，顾此失彼都会造成整体 redis 能的下降，甚至引发服务器宕机或内存泄露
<h3 id="2-三种删除策略">2、三种删除策略</h3>
<h4 id="定时删除">定时删除</h4>

创建一个定时器，当 key 设置有过期时间，且过期时间到达时，由定时器任务立即执对键的删除操作
优点：节约内存，到时就删除，快速释放掉不必要的内存占用
缺点：CPU 压力很大，无论 CPU 此时负载量多高，均占用 CPU，会影响 redis 服务器响应时间和指令吞吐量
总结：用处理器性能换取存储空间 （拿时间换空间）

<h4 id="惰性删除">惰性删除</h4>

数据到达过期时间，不做处理。等下次访问该数据时

如果未过期，返回数据
发现已过期，删除，返回不存在

优点：节约 CPU 性能，发现必须删除的时候才删除
缺点：内存压力很大，出现长期占用内存的数据
总结：用存储空间换取处理器性能 （拿空间换时间）

<h4 id="定期删除">定期删除</h4>

周期性轮询 redis 库中的时效性数据，采用随机抽取的策略，利用过期数占比的方式控制删除频度
特点 1：CPU 性能占用设置有峰值，检测频度可自定义设置
特点 2：内存压力不是很大，长期占用内存的冷数据会被持续清理
总结：周期性抽查存储空间 （随机抽查，重点抽查）

<h3 id="3-逐出算法">3、逐出算法</h3>
**当新数据进入 redis 时，如果内存不足怎么办？ **

Redis 使用内存存储数据，在执行每一个命令前，会调用 freeMemoryIfNeeded()<strng>检测内存是否充足。如果内存不满足新加入数据的最低存储要求，redis 要临时删除一些数据为前指令清理存储空间。清理数据的策略称为逐出算法
注意：逐出数据的过程不是 100% 能够清理出足够的可使用的内存空间，果不成功则反复执行。当对所有数据尝试完毕后，如果不能达到内存清理的要求，将出现错误信息。

<h4 id="影响数据逐出的相关配置">影响数据逐出的相关配置</h4>

最大可使用内存
<pre><code class="highlight-chroma">maxmemoryCopy
</code></pre>
占用物理内存的比例，默认值为 0，表示不限制。生产环境中根据需求设定，通常设置在 50% 上。

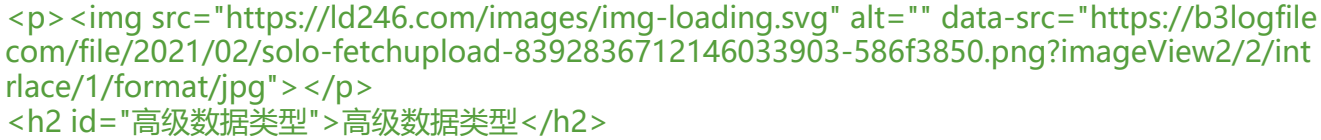
每次选取待删除数据的个数
<pre><code class="highlight-chroma">maxmemory-samplesCopy
</code></pre>
选取数据时并不会全库扫描，导致严重的性能消耗，降低读写性能。因此采用随机获取数据的方作为待检测删除数据

删除策略
<pre><code class="highlight-chroma">maxmemory-policyCopy
</code></pre>
达到最大内存后的，对被挑选出来的数据进行删除的策略

<h4 id="影响数据逐出的相关配置-">影响数据逐出的相关配置</h4>

LRU：最长时间没被使用的数据
LFU：一段时间内使用次数最少的数据
<h4 id="数据逐出策略配置依据">数据逐出策略配置依据</h4>

使用 INFO 命令输出监控信息，查询缓存 hit 和 miss的次数，根据业务需求调优 Redis 配置


<h2 id="高级数据类型">高级数据类型</h2>
<h3 id="1-Bitmaps">1、Bitmaps</h3>
<h4 id="基础操作">基础操作</h4>

获取指定 key 对应偏移量上的 bit 值
<pre><code class="highlight-chroma">getbit key offsetCopy

</code></pre>

设置指定 key 对应偏移量上的 bit 值，value 只能是 1 或 0
<pre><code class="highlight-chroma">setbit key offset valueCopy
</code></pre>

<h4 id="扩展操作">扩展操作</h4>

对指定 key 按位进行交、并、非、异或操作，并将结果保存到 destKey 中
/p>
<pre><code class="highlight-chroma">bitop op destKey key1 [key2...]Copy
</code></pre>

and：交
or：并
not：非
xor：异或

统计指定 key 中 1 的数量
<pre><code class="highlight-chroma">bitcount key [start end]Copy
</code></pre>

<h3 id="2-HyperLogLog">2、HyperLogLog</h3>
<h4 id="基数">基数</h4>

基数是数据集去重后元素个数
HyperLogLog 是用来做基数统计的，运用了 LogLog 的算法

<img sr ="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile.com/file/202 /02/solo-fetchupload-194665734374029783 1-62142178.png?imageView2/2/interlace/1/form t/jpg">
<h4 id="基本操作">基本操作</h4>

添加数据
<pre><code class="highlight-chroma">pfadd key element1, element2...Copy
</code></pre>

统计数据
<pre><code class="highlight-chroma">pfcount key1 key2....Copy
</code></pre>

合并数据

<pre><code class="highlight-chroma">pfmerge destkey sourcekey [sourcekey...]Copy
</code></pre>

<h4 id="相关说明">相关说明</h4>

用于进行基数统计，不是集合，不保存数据，只记录数量而不是具体数据</l
>
核心是基数估算算法，最终数值存在一定误差
误差范围：基数估计的结果是一个带有 0.81% 标准错误的近似值
耗空间极小，每个 hyperloglog key 占用了 12K 的内存用于标记基数
pfadd 命令不是一次性分配 12K 内存使用，会随着基数的增加内存逐渐增大</strong

Pfmerge 命令合并后占用的存储空间为 12K，无论
并之前数据量多少

<h3 id="3-GEO">3、GEO</h3>
<h4 id="基本操作-">基本操作</h4>

添加坐标点
<pre><code class="highlight-chroma">geoadd key longitude latitude member [longitude latitude member ...]
georadius key lon
itude latitude radius m|km|ft|mi [withcoord] [withdist] [withhash] [count count]Copy
</code></pre>

获取坐标点
<pre><code class="highlight-chroma">geopos key member [member ...]
georadiusbymem
er key member radius m|km|ft|mi [withcoord] [withdist] [withhash] [count count]Copy
</code></pre>

计算坐标点距离
<pre><code class="highlight-chroma">geodist key member1 member2 [unit]
geohash key me
ber [member ...]
</code></pre>

