黑客派

Redis 笔记

作者：changming

原文链接：https://hacpai.com/article/1471428097316

来源网站：黑客派

<blockquote>
<h2>数据结构</h2>
</blockquote>

SortedSet zunionstore 方法可以对一个或者多个有序集合(sortedset)进行并集计算，如果目集合是set，而不是sorted set，也可以进行合并
SortedSet zunionstore 方法，当目标存储集合已经存在时，会进行值的覆盖
Hash

这种结构，其内容实现其实是一个HashMap，即key对应一个字符串，而value是使用HashMa实现的。如图

<img title="hashredis" src="https://static.hacpai.com/images/ig-loading.svg" alt="hashredis" width="383" height="192" data-src="http://obu3fc5lw.bkt.clouddn.com/9c06c1168d594abba030e23a2f3073ce.png">

只需要通过key+field既可以获取到value值。redis是单线程模型，如果map中的数据过大，遍需要很长时间的话，就会阻塞客户端其他对redis的操作，这点需要注意。
另外，当hash成员数量较少时，其内部实现是使用类似一维数组的方式(ziplist)来紧凑存储，以约内存空间，而当成员数量较大时，才会使用hashmap的方式存储。

key

必须是一个字符串类型，不能包含边界字符，比如包括空格和换行\n等是被认为非法的，也不包括redis协议中认定的特殊字符，如\r\n
推荐key的写法是object-type:id:field，同时key不能太长，明显占内存

List 内部实现是使用一个双向链表，方便正向和反向进行查找
Set

内部实现也是一个HashMap，只是HashMap的value值永远为null
set是不会有重复元素的，就是根据计算value的hash值进行快速排重
set是无序的，不过可以通过sort命令进行二次排序，而且可以参照第三方的进行排序

SortedSet

内部使用HashMap和SkipList实现，skiplist用来存储value，其特点是查找快，节约存储空间HashMap中存储的是value和score的映射关系
自动排序，即放入其中后，根据score值自动排序，当你需要一个有序且不重复的集合的时候考使用sortedset

<blockquote>
 <h2>性能</h2>
</blockquote>

 SortedSet 性能取决于其中存储的元素个数

<img title="性能" src="https://static.hacpai.com/images/img-loading.vg" alt="性能" width="1076" height="60" data-src="http://obu3fc5lw.bkt.clouddn.com/ebca76ca7f344d7aa2d511da9acc2df.png">
<blockquote>
 <h2>持久化</h2>
</blockquote>

 定时快照方式(snapshot)

  内部使用一个定时器，定期检查数据发生的改变次数和时间是否满足配置的持久化触发条件，果满足就由操作系统fork一个子进程来执行持久化操作，即子进程就可以遍历整个内存空间进行存储而此时父进程仍然提供正常的服务
  该方案的缺点是，存储的定时快照只是一段时间内的内存映像，重启后，上次快照存储后到重这段时间内的内存数据会丢失
  每次做快照方式存储数据时，都会将整个内存数据重新保存一次，而不是增量存储

 基于语句追加文件的方式(aof)

  即append of file 方式，将每次使redis数据发生变化的语句都会被追加到一个log文件中，这个lg文件即是持久化的数据
  缺点是数据量大的时侯，log文件体积过大，从而重启时，加载数据进入内存会非常缓慢，几十的数据需要几个小时，因为每个命令都要执行一遍，另外，每个命令都要写log，redis读写性能也会降

 虚拟内存(vm) — 放弃
 disk store — 实践中
 在持久化方案中，磁盘IO带来的问题

  当使用的物理内存将要达到上限时，会发生redis崩溃的现象，这种情况原因是在读写持久化文时会将该文件加载到内存中，从而内存中就会有2份重复的数据，导致操作系统让你的进程进行swap作，此时开始不稳定甚至崩溃(现在已经有maxmemory-policy选项了)
  经验是 redis使用的物理内存达到内存总容量的3/5时就比较危险了

<script async src="https://pagead2.googlesyndication.com/pagead/js/adsbygoogle.js"></script>
<!-- 黑客派PC帖子内嵌-展示 -->
<ins class="adsbygoogle" style="display:block" data-ad-client="ca-pub-5357405790190342" data-ad-slot="8316640078" data-ad-format="auto" data-full-width-responsive="true"></in>
<script>
    (adsbygoogle = window.adsbygoogle || []).push({});
</script>
<blockquote>
 <h2>事务</h2>

</blockquote>

 提供事务支持，可以保证一串命令操作的原子性，中间不会被其他命令打断，其实质是开启事务(multi)，所有操作命令会放在一个队列中，然后一起执行，保证执行过程中不会被其他命令打断
 当这一串命令中某个命令发生错误时，事务终止，同时已经执行的命令不能进行回滚
 watch命令可以监视某个key，在调用watch命令后到exec执行，这段时间内如果这个key的值生了变化，那么整个事务就会失败(乐观锁)
 在事务中，2个操作存在依赖关系，就要注意了，不能让一个写操作依赖一个读操作，因为事务的操作并没有立即执行，在事务将操作命令加入队列过程中，有可能其他的客户端会操作这个key，就是这个key并没有做任何的同步，所以我们需要使用watch来监视此key
 关于事务的应用请参考这篇<a href="https://link.hacpai.com/forward?goto=http%3A%2F%Fyangchangming.com%2Fblog%2Fshow%2F40" target="_blank" rel="nofollow ugc">文章</a

<blockquote>
 <h2>适用场景</h2>
</blockquote>

 朋友最近活动
 leader board 排行榜功能，在线游戏时，可以随时获取新的排行数据，通过ZRANG
 猜测可能认识的人，结合使用ZRANK和ZRANGE方法，可以查出与指定对象相似的对象出来</li

 关注数与被关注数
 计数：最近用户在页面间停顿时间不超过60秒的页面浏览量
 作为索引数据存储的结构，score存储为数值，value为对象id，使用ZRANGEBYSCORE检索一确切的范围

<blockquote>
 <h2>用作缓存</h2>
</blockquote>

 2种方式可以把redis当做缓存来使用

  给每个键设置一个固定的过期时间，但是这样内存也需要额外的空间来存储这些过期时间
  通过设置内存使用上限和指定内存策略算法

<pre class="brush: bash">maxmemory 2mb
maxmemory-policy allkeys-lru</pre>

 相比使用额外内存空间存储多个键的过期时间，使用缓存设置是一种更加有效利用内存的方式。且相比每个键固定的 过期时间，使用LRU也是一种更加推荐的方式，因为这样能使应用的热数(更频繁使用的键) 在内存中停留时间更久。
 基本上这么配置下的Redis可以当成memcached使用。当我们把Redis当成缓存来使用的时候，果应用程序同时也需要把Redis当成存储系统来使用，那么强烈建议 使用两个Redis实例。一个是缓，使用上述方法进行配置，另一个是存储，根据应用的持久化需求进行配置，并且 只存储那些不需要缓存的数据。

<blockquote>
 <h2>Master Slave复制</h2>
</blockquote>

 一个master可以有多个slave，而一个slave也可以连接其他的slave，形成图状结构
 master在与slave进行通信时，是可以继续处理客户端请求的；而slave在与master进行通信时

就会阻塞客户端的请求
 主从复制的场景，用于提高系统的伸缩性，即让slave负责读取操作，master负责写操作；或者可以仅仅让slave作为冗余存储
 主从复制主要是分为2个步骤，第一个是同步数据，从master中把snapshot发送给slave，第二是命令传播，master中的写操作都会传递给slave
 没有增量复制功能，只是用snapshot重建slave内存结构，新浪微薄版本的redis已经实现了增量制功能
 每次slave重连master，master都会把shapshot重新发送给slave，重建内存数据结构(使用2.8本+以上的psync命令，可以实现增量复制)
 可以通过主动复制来避免redis自身主从复制的缺点，既从客户端对存储的数据进行双写或者多，数据的多份复制机制，可以避免单点失效故障
 这种主动复制功能的缺陷就是如何保证多个节点数据的一致性？需要引入一致性算法，但是这样会降低写性能
 如果有多个slave发来同步命令，master后台只会启动一个进程来写snapshot，然后发送给所有slave
 关于主从复制实现的详细步骤，可以参考黄健宏的视频和ppt
 master最好不要写snapshot，数据量大时会阻塞主线程运行

<script async src="https://pagead2.googlesyndication.com/pagead/js/adsbygoogle.js"></script>
<!-- 黑客派PC帖子内嵌-展示 -->
<ins class="adsbygoogle" style="display:block" data-ad-client="ca-pub-5357405790190342" data-ad-slot="8316640078" data-ad-format="auto" data-full-width-responsive="true"></ins>
<script>
    (adsbygoogle = window.adsbygoogle || []).push({});
</script>
<blockquote>
 <h2>可扩展集群</h2>
</blockquote>

 100g的业务数据量，假设服务器内存是50g，那么根据磁盘io问题考虑，需要3-4台服务器存储
 当数据量达到这个级别时就需要考虑动态在线扩容的问题了
 拆分过程（presharding方案）

  在新机器上启动好对应端口的Redis实例
  配置新端口为待迁移端口的从库
  待复制完成，与主库完成同步后，切换所有客户端配置到新的从库的端口
  配置从库为新的主库
  移除老的端口实例
  重复上述过程迁移好所有的端口到指定服务器上

<blockquote>
 <h2>使用总结</h2>
</blockquote>

 根据业务需要选择合适的数据结构，为不同的使用场景设定不同的紧凑型存储参数
 当业务场景不需要进行数据持久化时，关闭所有的持久化参数，以获得最大的性能和内存使用量/li>

需要使用持久化时，根据业务特性，是否容忍数据丢失，从而选择aof或者snapshot方式中的一，不要使用vm和diskstore
不要使得redis实际内存使用量达到最大物理内存的3/5
基于实际项目的一些建议
持久化方案首选AOF方式

<blockquote>
<h2>改进思路</h2>
</blockquote>

可以做到按照AOF文件大小自动分割滚动
实际项目可以采取与Mysql结合的方式，即mysql作为主库，Redis作为高速查询从库的异构读分离的方案

<img title="redismysql" src="https://static.hacpai.com/imagesimg-loading.svg" alt="redismysql" width="500" height="351" data-src="http://obu3fc5lw.bktclouddn.com/27991cbf94e644fbb515d4dcb1bbfc69.png">
<blockquote>
<h2>客户端、二次开发相关</h2>
</blockquote>

jedis，官方推荐java客户端
客户端与spring集成 http://projects.spring.io/spring-data-redis/
