> 你知道的越多，你不知道的越多

> 点赞再看，养成习惯

> 本文 **GitHub** https://github.com/JavaFamily 上已经收录，有一线大厂面试点思维导图，也整理了很多我的文档，欢迎Star和完善，大家面试可以参照考点复习，希望我们一起有点东西。

前言

作为一个在互联网公司面一次拿一次Offer的面霸，打败了无数竞争对手，每次都只能看到无数落寞的身影失望的离开，略感愧疚（**请允许我使用一下夸张的修辞手法**）。

于是在一个寂寞难耐的夜晚，我痛定思痛，决定开始写互联网技术栈面试相关的文章，希望能帮助各位读者以后面试势如破竹，对面试官进行360°的反击，吊打问你的面试官，让一同面试的同僚瞠目结舌，疯狂收割大厂Offer！

所有文章的名字只是我的噱头，我们应该有一颗谦逊的心，所以希望大家怀着空杯心态好好学，一起进步。

絮叨

写这期其实比较纠结，我之前的写的比较通俗易懂，一是我都知道这些点，二是之前我在所在的电商公司对雪崩，击穿啥的还算有场景去接触。但是线上的Redis集群我实际操作经验很少，总不能在公司线上环境实践那些操作吧，所以最后看了下官网，还有一些资料（文章后面我都会贴出来），强行怼了这么篇出来。

最近双十一小忙，周末双十一值班目测没时间写，那我是暖男呀，我不能鸽啊，就有了这一篇，下一篇迟到你们不要喷我哈，而且下一篇还是**Redis**的终章还是得构思下，不熟悉的知识点我怕漏洞多，特意让以前的大牛同事看了下，所以有啥不对的地方大家及时留言**Diss**我，写这篇是真的难，诺下面就是我本人某天凌晨两点的拍的视频，多动症的仔。

之前说过系列第二篇到300赞我就发第三篇

咋样没骗你们吧，就很枯竭，不BB了，开搞。

不点个赞对不起我，这次不要白嫖我！

正文

上几期**《吊打面试官》**还没看的小伙伴可以回顾一下（明明就写了两期说的好像很多一样）！

- 《吊打面试官》系列-Redis基础
- 《吊打面试官》系列-缓存雪崩、击穿、穿透

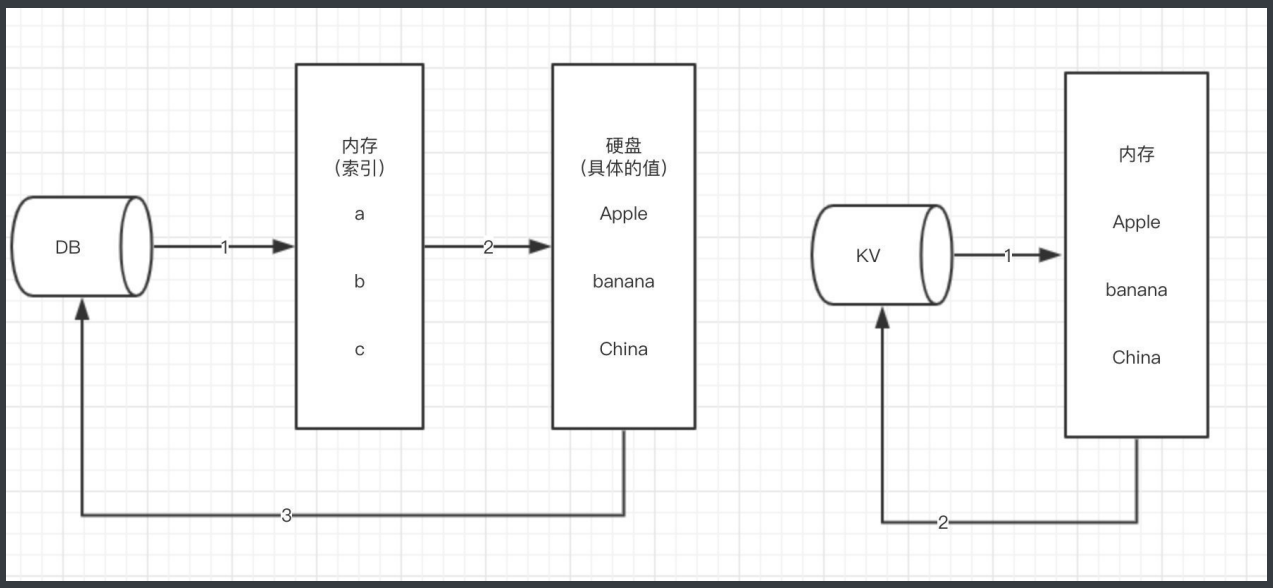
大家都知道一个技术的引入方便了开发，解决了各种问题，但是也会带来对应的问题，**技术是把双刃剑**嘛，集群的引入也会带来很多问题，如：集群的高可用怎么保证，数据怎么同步等等，我们话不多说，有请下一位受害者为我们展示。

面试开始

> 三个大腹便便，穿着格子衬衣的中年男子，拿着三个满是划痕的mac向你走来，看着快秃顶的头发，心想着肯定是尼玛顶级架构师吧！而且还是三个，但是还好我看过敖丙写的《吊打面试官》系列，腹有诗书气自华，根本虚都不虚好伐。

小伙子你好，之前问过了你基础知识以及一些缓存的常见几个大问题了，那你能跟我聊聊为啥Redis那么快么？

哦，帅气迷人的面试官您好，我们可以先看一下关系型数据库跟Redis本质上的区别。



Redis采用的是基于内存的采用的是单进程单线程模型的 KV 数据库，由C语言编写，官方提供的数据是可以达到100000+的**QPS（每秒内查询次数）**。

- 完全基于内存，绝大部分请求是纯粹的内存操作，非常快速。它的，数据存在内存中，类似于**HashMap**，**HashMap**的优势就是查找和操作的时间复杂度都是O(1)；
- 数据结构简单，对数据操作也简单，**Redis**中的数据结构是专门进行设计的；
- 采用单线程，避免了不必要的上下文切换和竞争条件，也不存在多进程或者多线程导致的切换而消耗 **CPU**，不用去考虑各种锁的问题，不存在加锁释放锁操作，没有因为可能出现死锁而导致的性能消耗；
- 使用多路I/O复用模型，非阻塞IO；
- 使用底层模型不同，它们之间底层实现方式以及与客户端之间通信的应用协议不一样，**Redis**直接自己构建了VM 机制 ，因为一般的系统调用系统函数的话，会浪费一定的时间去移动和请求；

我可以问一下啥是上下文切换么?

我可以打个比方么：我记得有过一个小伙伴微信问过我上下文切换是啥，为啥可能会线程不安全，我是这么说的，就好比你看一本英文书，你看到第十页发现有个单词不会读，你加了个书签，然后去查字典，过了一会你又回来继续从书签那里读，ok到目前为止没啥问题。

如果是你一个人读肯定没啥问题，但是你去查的时候，别的小伙伴好奇你在看啥他就翻了一下你的书，然后溜了，哦豁，你再看的时候就发现书不是你看的那一页了。不知道到这里为止我有没有解释清楚，以及为啥会线程不安全，就是因为你一个人怎么看都没事，但是人多了换来换去的操作一本书数据就乱了。可能我的解释很粗糙，但是道理应该是一样的。

那他是单线程的，我们现在服务器都是多核的，那不是很浪费?

是的他是单线程的，但是，我们可以通过在单机开多个**Redis实例**嘛。

既然提到了单机会有瓶颈，那你们是怎么解决这个瓶颈的?

我们用到了集群的部署方式也就是**Redis cluster**，并且是主从同步读写分离，类似**Mysql**的主从同步，**Redis cluster** 支撑 N 个 **Redis master node**，每个**master node**都可以挂载多个 **slave node**。

这样整个 **Redis** 就可以横向扩容了。如果你要支撑更大数据量的缓存，那就横向扩容更多的 **master** 节点，每个 **master** 节点就能存放更多的数据了。

哦? 那问题就来了，他们之间是怎么进行数据交互的? 以及Redis是怎么进行持久化的? Redis数据都在内存中，一断电或者重启不就木有了嘛?

是的，持久化的话是**Redis**高可用中比较重要的一个环节，因为**Redis**数据在内存的特性，持久化必须得有，我了解到的持久化是有两种方式的。

- RDB：**RDB** 持久化机制，是对 **Redis** 中的数据执行**周期性**的持久化。
- AOF：**AOF** 机制对每条写入命令作为日志，以 **append-only** 的模式写入一个日志文件中，因为这个模式是只追加的方式，所以没有任何磁盘寻址的开销，所以很快，有点像Mysql中的**binlog**。

两种方式都可以把**Redis**内存中的数据持久化到磁盘上，然后再将这些数据备份到别的地方去，**RDB**更适合做**冷备**，**AOF**更适合做**热备**，比如我杭州的某电商公司有这两个数据，我备份一份到我杭州的节点，再备份一个到上海的，就算发生无法避免的自然灾害，也不会两个地方都一起挂吧，这**灾备**也就是**异地容灾**，地球毁灭他没办法。

tip：两种机制全部开启的时候，Redis在重启的时候会默认使用AOF去重新构建数据，因为AOF的数据是比RDB更完整的。

那这两种机制各自优缺点是啥?

我先说**RDB**吧

优点：

他会生成多个数据文件，每个数据文件分别都代表了某一时刻**Redis**里面的数据，这种方式，有没有觉得很适合做**冷备**，完整的数据运维设置定时任务，定时同步到远端的服务器，比如阿里的云服务，这样一旦线上挂了，你想恢复多少分钟之前的数据，就去远端拷贝一份之前的数据就好了。

RDB对**Redis**的性能影响非常小，是因为在同步数据的时候他只是**fork**了一个子进程去做持久化的，而且他在数据恢复的时候速度比**AOF**来的快。

缺点：

RDB都是快照文件，都是默认五分钟甚至更久的时间才会生成一次，这意味着你这次同步到下次同步这中间五分钟的数据都很可能全部丢失掉。**AOF**则最多丢一秒的数据，**数据完整性**上高下立判。

还有就是**RDB**在生成数据快照的时候，如果文件很大，客户端可能会暂停几毫秒甚至几秒，你公司在做秒杀的时候他刚好在这个时候**fork**了一个子进程去生成一个大快照，哦豁，出大问题。

我们再来说说**AOF**

优点：

上面提到了，**RDB**五分钟一次生成快照，但是**AOF**是一秒一次去通过一个后台的线程 `fsync` 操作，那最多丢这一秒的数据。

AOF在对日志文件进行操作的时候是以 `append-only` 的方式去写的，他只是追加的方式写数据，自然就少了很多磁盘寻址的开销了，写入性能惊人，文件也不容易破损。

AOF的日志是通过一个叫**非常可读**的方式记录的，这样的特性就适合做**灾难性数据误删除**的紧急恢复了，比如公司的实习生通过**flushall**清空了所有的数据，只要这个时候后台重写还没发生，你马上拷贝一份**AOF**日志文件，把最后一条**flushall**命令删了就完事了。

tip：我说的命令你们别真去线上系统操作啊，想试去自己买的服务器上装个Redis试，别到时候来说，敖丙真是个渣男，害我把服务器搞崩了，Redis官网上的命令都去看看，不要乱试！！！

缺点：

一样的数据，**AOF**文件比**RDB**还要大。

AOF开启后，**Redis**支持写的**QPS**会比**RDB**支持写的要低，他不是每秒都要去异步刷新一次日志嘛**fsync**，当然即使这样性能还是很高，我记得**ElasticSearch**也是这样的，异步刷新缓存区的数据去持久化，为啥这么做呢，不直接来一条怼一条呢，那我会告诉你这样性能可能低到没办法用的，大家可以思考下为啥哟。

那两者怎么选择?

小孩子才做选择，**我全都要**，你单独用**RDB**你会丢失很多数据，你单独用**AOF**，你数据恢复没**RDB**来的快，真出什么时候第一时间用**RDB**恢复，然后**AOF**做数据补全，真香！冷备热备一起上，才是互联网时代一个高健壮性系统的王道。

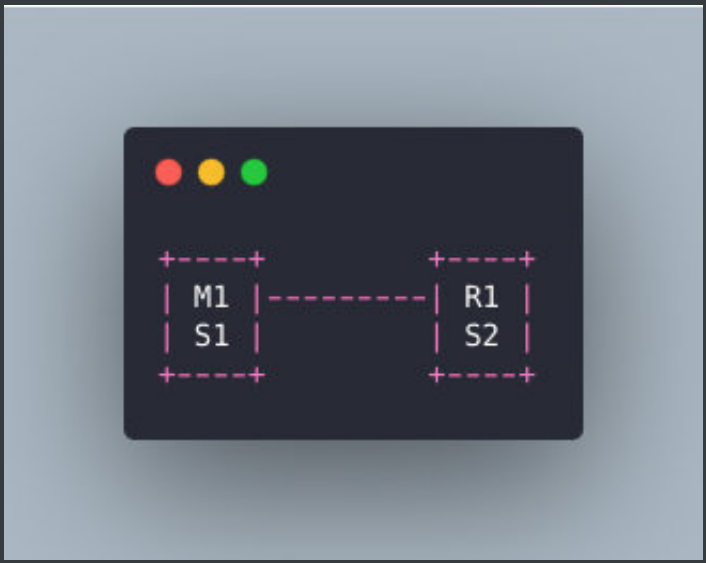
看不出来年纪轻轻有点东西的呀，对了我听你提到了高可用，Redis还有其他保证集群高可用的方式么?

！！！晕 自己给自己埋个坑（其实是明早就准备好了，故意抛出这个词等他问，就怕他不问）。

假装思考一会（**不要太久，免得以为你真的不会**），哦我想起来了，还有哨兵集群**sentinel**。

哨兵必须用三个实例去保证自己的健壮性的，哨兵+主从并**不能保证数据不丢失**，但是可以保证集群的**高可用**。

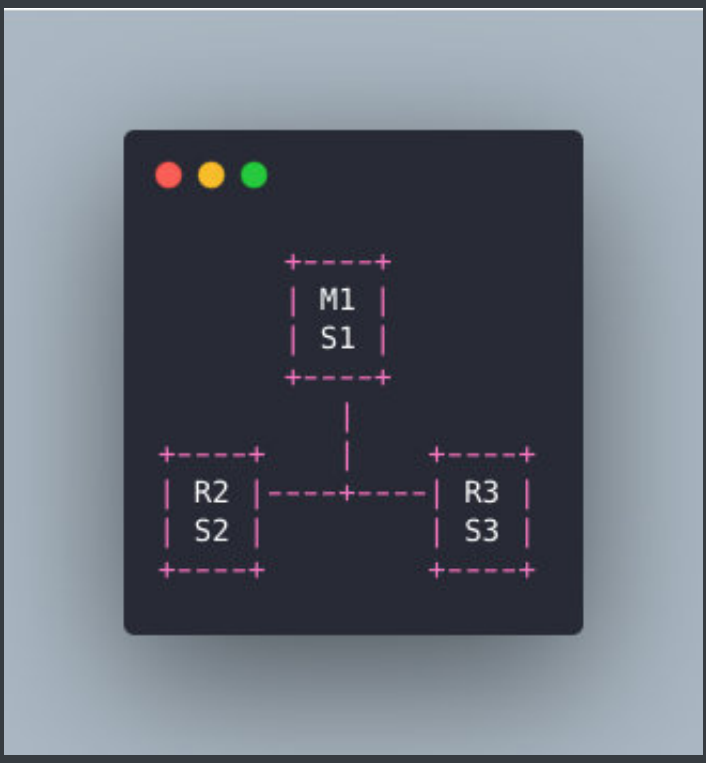
为啥必须要三个实例呢？我们先看看两个哨兵会咋样。



master宕机了 s1和s2两个哨兵只要有一个认为你宕机了就切换了，并且会选举出一个哨兵去执行故障，但是这个时候也需要大多数哨兵都是运行的。

那这样有啥问题呢？M1宕机了，S1没挂那其实是OK的，但是整个机器都挂了呢？哨兵就只剩下S2个裸屌了，没有哨兵去允许故障转移了，虽然另外一个机器上还有R1，但是故障转移就是不执行。

经典的哨兵集群是这样的：



M1所在的机器挂了，哨兵还有两个，两个人一看他不是挂了嘛，那我们就选举一个出来执行故障转移不就好了。

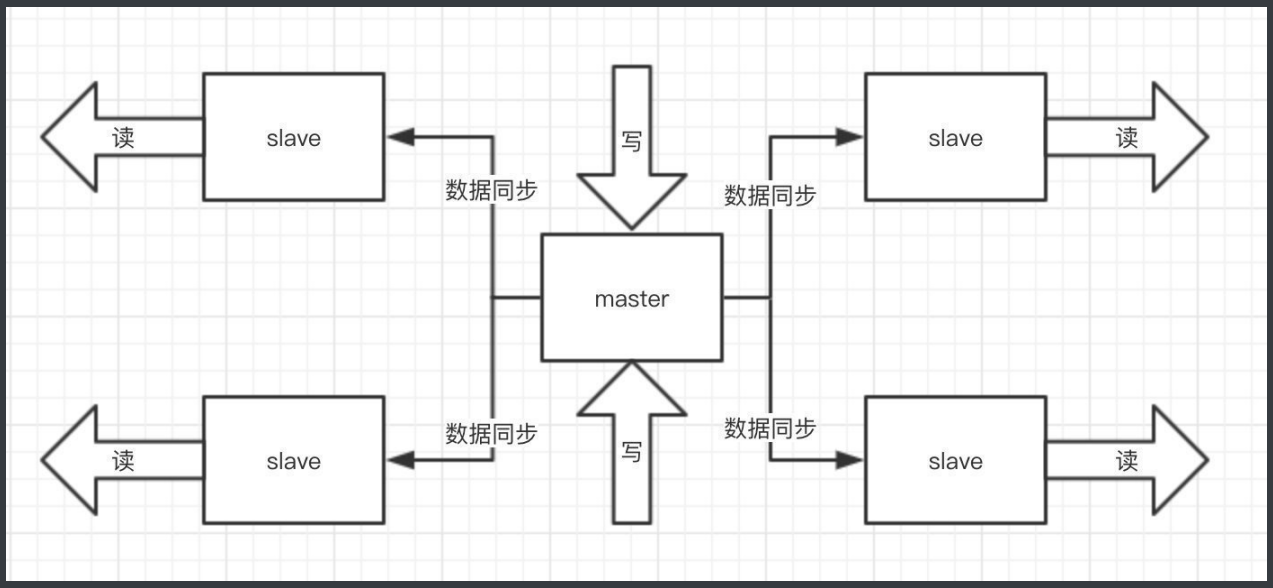
暖男我，小的总结下哨兵组件的主要功能：

- 集群监控：负责监控 Redis master 和 slave 进程是否正常工作。
- 消息通知：如果某个 **Redis** 实例有故障，那么哨兵负责发送消息作为报警通知给管理员。
- 故障转移：如果 master node 挂掉了，会自动转移到 slave node 上。
- 配置中心：如果故障转移发生了，通知 client 客户端新的 master 地址。

我记得你还提到了主从同步，能说一下主从之间的数据怎么同步的么?

面试官您的记性可真是一级棒呢，我都要忘了你还记得，我特么谢谢你，提到这个，就跟我前面提到的数据持久化的**RDB**和**AOF**有着比密切的关系了。

我先说下为啥要用主从这样的架构模式，前面提到了单机**QPS**是有上限的，而且**Redis**的特性就是必须支撑读高并发的，那你一台机器又读又写，**这谁顶得住啊**，不当人啊！但是你让这个master机器去写，数据同步给别的slave机器，他们都拿去读，分发掉大量的请求那是不是好很多，而且扩容的时候还可以轻松实现水平扩容。



回归正题，他们数据怎么同步的呢?

你启动一台slave 的时候，他会发送一个**psync**命令给master ，如果是这个slave第一次连接到master，他会触发一个全量复制。master就会启动一个线程，生成**RDB**快照，还会把新的写请求都缓存在内存中，**RDB**文件生成后，master会将这个**RDB**发送给slave的，slave拿到之后做的第一件事情就是写进本地的磁盘，然后加载进内存，然后master会把内存里面缓存的那些新命名都发给slave。

数据传输的时候断网了或者服务器挂了怎么办啊?

传输过程中有什么网络问题啥的，会自动重连的，并且连接之后会把缺少的数据补上的。

大家需要记得的就是，RDB快照的数据生成的时候，缓存区也必须同时开始接受新请求，不然你旧的数据过去了，你在同步期间的增量数据咋办？是吧?

那说了这么多你能说一下他的内存淘汰机制么，来手写一下LRU代码?

手写LRU？你是不是想直接跳起来说一句：Are U F**k Kidding me?

这个问题是我在蚂蚁金服三面的时候亲身被问过的问题，不知道大家有没有被怼到过这个问题。

Redis的过期策略，是有**定期删除+惰性删除**两种。

定期好理解，默认100s就随机抽一些设置了过期时间的key，去检查是否过期，过期了就删了。

为啥不扫描全部设置了过期时间的key呢?

假如Redis里面所有的key都有过期时间，都扫描一遍？那太恐怖了，而且我们线上基本上也都是会设置一定的过期时间的。全扫描跟你去查数据库不带where条件不走索引全表扫描一样，100s一次，Redis累都累死了。

如果一直没随机到很多key，里面不就存在大量的无效key了?

好问题，**惰性删除**，见名知意，惰性嘛，我不主动删，我懒，我等你来查询了我看看你过期没，过期就删了还不给你返回，没过期该怎么样就怎么样。

最后就是如果的如果，定期没删，我也没查询，那可咋整?

内存淘汰机制!

官网上给到的内存淘汰机制是以下几个:

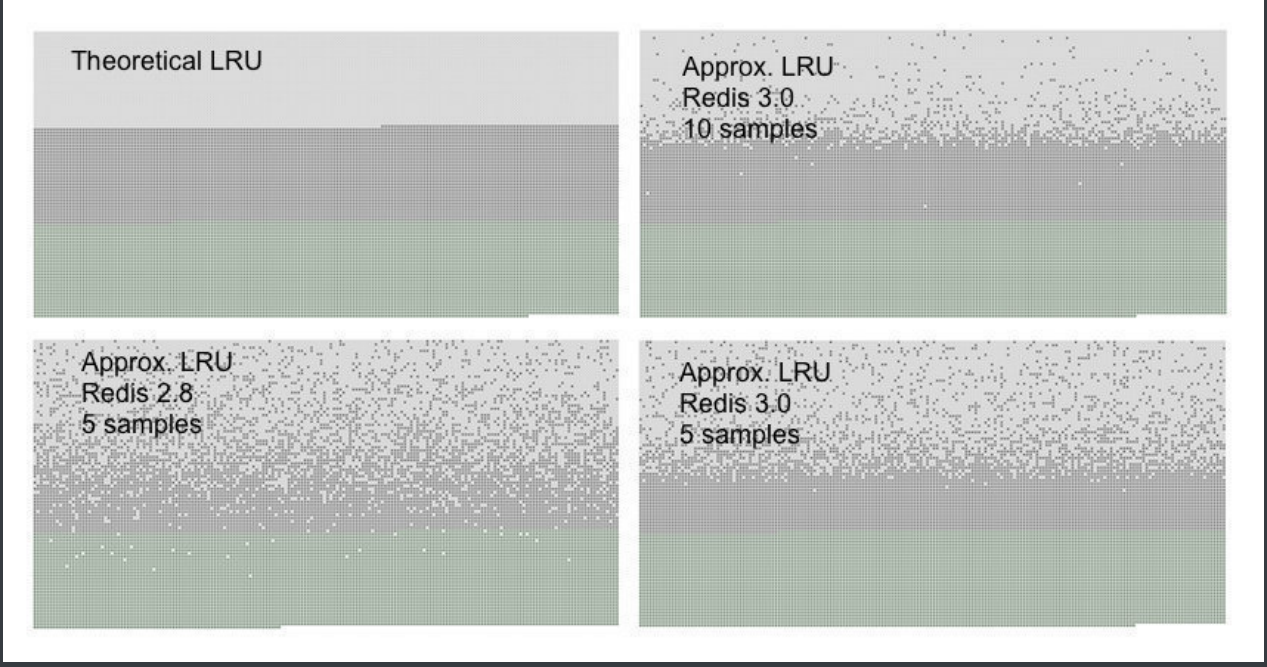
- **noeviction**:返回错误当内存限制达到并且客户端尝试执行会让更多内存被使用的命令（大部分的写入指令，但DEL和几个例外）
- **allkeys-lru**: 尝试回收最少使用的键（LRU），使得新添加的数据有空间存放。
- **volatile-lru**: 尝试回收最少使用的键（LRU），但仅限于在过期集合的键,使得新添加的数据有空间存放。

- **allkeys-random**: 回收随机的键使得新添加的数据有空间存放。
- **volatile-random**: 回收随机的键使得新添加的数据有空间存放，但仅限于在过期集合的键。
- **volatile-ttl**: 回收在过期集合的键，并且优先回收存活时间（TTL）较短的键,使得新添加的数据有空间存放。

  如果没有键满足回收的前提条件的话，策略**volatile-lru**, **volatile-random**以及**volatile-ttl**就和 noeviction 差不多了。

至于**LRU**我也简单提一下，手写实在是太长了，大家可以去**Redis官网**看看，我把**近视LUR**效果给大家看看

tip：Redis为什么不使用真实的LRU实现是因为这需要太多的内存。不过近似的LRU算法对于应用而言应该是等价的。使用真实的LRU算法与近似的算法可以通过下面的图像对比。



你可以看到三种点在图片中, 形成了三种带.

- 浅灰色带是已经被回收的对象。
- 灰色带是没有被回收的对象。
- 绿色带是被添加的对象。
- 在**LRU**实现的理论中，我们希望的是，在旧键中的第一半将会过期。**Redis**的**LRU**算法则是概率的过期旧的键。

你可以看到，在都是五个采样的时候Redis 3.0比Redis 2.8要好，Redis2.8中在最后一次访问之间的大多数的对象依然保留着。使用10个采样大小的Redis 3.0的近似值已经非常接近理论的性能。

注意LRU只是个预测键将如何被访问的模型。另外，如果你的数据访问模式非常接近幂定律，大部分的访问将集中在一个键的集合中，LRU的近似算法将处理得很好。

其实在大家熟悉的**LinkedHashMap**中也实现了Lru算法的，实现如下：

```
final Map<Long, TimeoutInfoHolder> timeoutInfoHandlers =
        Collections.synchronizedMap(new LinkedHashMap<Long, TimeoutInfoHolder>(100, .75F, true) {
            @Override
            protected boolean removeEldestEntry(Map.Entry eldest) {
                return size() > 100;
            }
        });
```

当容量超过100时，开始执行**LRU**策略：将最近最少未使用的 **TimeoutInfoHolder** 对象 **evict** 掉。

真实面试中会让你写LUR算法，你可别搞原始的那个，那真TM多，写不完的，你要么怼上面这个，要么怼下面这个，找一个数据结构实现下Java版本的LRU还是比较容易的，知道啥原理就好了。

```
class LRUCache<K, V> extends LinkedHashMap<K, V> {
    private final int CACHE_SIZE;

    /**
     * 传递进来最多能缓存多少数据
     *
     * @param cacheSize 缓存大小
     */
    public LRUCache(int cacheSize) {
        // true 表示让 linkedHashMap 按照访问顺序来进行排序，最近访问的放在头部，最老访问的放在尾部。
        super((int) Math.ceil(cacheSize / 0.75) + 1, 0.75f, true);
        CACHE_SIZE = cacheSize;
    }

    @Override
    protected boolean removeEldestEntry(Map.Entry<K, V> eldest) {
        // 当 map中的数据量大于指定的缓存个数的时候，就自动删除最老的数据。
        return size() > CACHE_SIZE;
    }
}
```

面试结束

小伙子，你确实有点东西，HRBP会联系你的，请务必保持你的手机畅通好么？

好的谢谢面试官，面试官真好，我还想再面几次，噗此。

能回答得这么全面这么细节还是忍不住点赞

（暗示点赞，每次都看了不点赞，你们想白嫖我么？你们好坏喲，不过我好喜欢）

总结

好了，我们玩归玩，闹归闹，别拿面试开玩笑，我这么写是为了节目效果，大家面试请认真对待。

这一期是这期没前面好理解了对吧，我就在自己的服务器上启动了，然后再去官网看看命令一顿瞎操作的，查阅了部分资料，这里给大家推荐几本经典的Redis入门的书籍和我参考的资料。

- Redis中文官网
- 《Redis入门指南(第2版)》
- 《Redis实战》
- 《Redis设计与实现》
- 《大型网站技术架构——李智慧》
- 《Redis 设计与实现——黄健宏》
- 《Redis 深度历险——钱文品》
- 《亿级流量网站架构核心技术——张开涛》
- 《中华石杉——石杉》

不出意外的话这是Redis的倒数第二期，最后一期不知道写啥还没想好，我得好好想想，加上最近不是双十一嘛得加加班，**你看看开头的我，多可怜，那还不点个赞？买个服务器**？不确定下一期多久出，想早点看到更新的小伙伴可以去公众号**催更**，公众号提前一到两天更新。

点关注，不迷路

好了各位，以上就是这篇文章的全部内容了，能看到这里的人呀，都是**人才**。

我后面会每周都更新几篇一线互联网大厂面试和常用技术栈相关的文章，非常感谢**人才**们能看到这里，如果这个文章写得还不错，觉得「敖丙」我**有点东西**的话 **求点赞**👍 **求关注**❤️ **求分享**👥 对暖男我来说真的 **非常有用**！！！

创作不易，各位的支持和认可，就是我创作的最大动力，我们下篇文章见！

敖丙 | 文 【原创】

如果本篇博客有任何错误，请批评指教，不胜感激！

> 文章每周持续更新，可以微信搜索「 **三太子敖丙** 」第一时间阅读和催更（比博客早一到两篇哟），本文 **GitHub** https://github.com/JavaFamily 已经收录，有一线大厂面试点思维导图，也整理了很多我的文档，欢迎Star和完善，大家面试可以参照考点复习，希望我们一起有点东西。