**1、Redis数据类型**

五大基本类型

String，Hash，List，Set，ZSet

三种特殊类型

地理位置 Geospatial

基数统计 Hyperloglog

位图 Bitmap

**2、zset数据结构**

zset底层的存储结构包括ziplist或skiplist，在同时满足以下两个条件的时候使用ziplist，其他时候使用skiplist，两个条件如下：

---有序集合保存的元素数量小于128个

---有序集合保存的所有元素的长度小于64字节

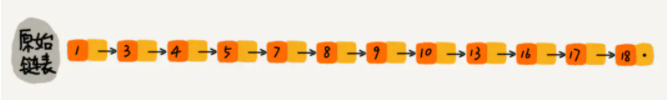
1、当ziplist作为zset的底层存储结构时候，每个集合元素使用两个紧挨在一起的压缩列表节点来保存，第一个节点保存元素的成员，第二个元素保存元素的分值。



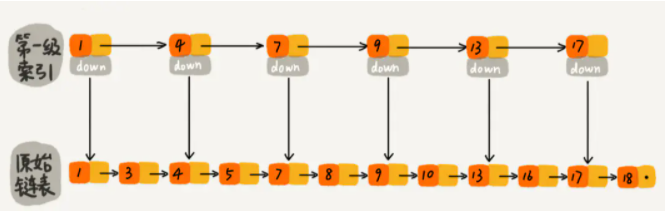
2、当skiplist作为zset的底层存储结构的时候，使用skiplist按序保存元素及分值，使用dict来保存元素和分值的映射关系。

**跳表（Skip List）**

跳表，是基于链表实现的一种类似“二分”的算法。它可以快速的实现增，删，改，查操作。

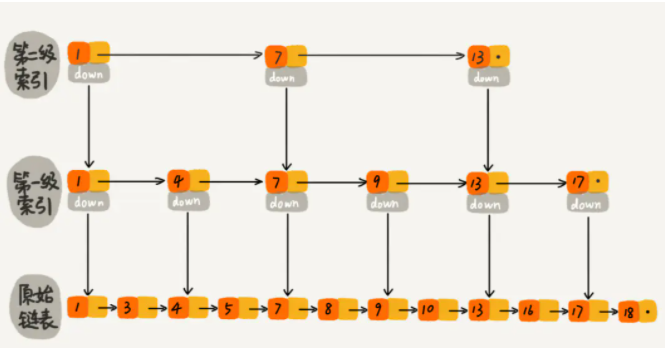


如果我们给该单链表加一级索引，将会改善查询效率。



其中的down指针指向原始链表。

如果我们再加多几级索引的话，效率将会进一步提升。**这种链表加多级索引的结构，就叫做跳表**。



定位到要插入或者删除数据的位置需要的时间复杂度为**O(logn)**.  
在插入的时候，我们需要考虑将要插入的数据也插入到索引中去。在这里使用的策略是通过随机函数生成一个随机数K,然后将要插入的数据同时插入到k级以下的每级索引中。

**\*\*3、Redis的过期策略**

redis 过期策略是：**定期删除+惰性删除**

**定期删除：**

每隔一定的时间，抽取一定数量的key，并清除其中已过期的key。

**惰性删除：**

只有当访问一个key时，才会判断该key是否已过期，过期则清除。该策略可以最大化地节省CPU资源，却对内存非常不友好。极端情况可能出现大量的过期key没有再次被访问，从而不会被清除，占用大量内存。

**\*\*4、Redis的内存淘汰策略**

（1）volatile-lru：从已设置过期时间的数据集中挑选最近最少使用的数据淘汰。

（2）volatile-ttl：从已设置过期时间的数据集中挑选将要过期的数据淘汰。

（3）volatile-random：从已设置过期时间的数据集中任意选择数据淘汰。

（4）volatile-lfu：从已设置过期时间的数据集挑选使用频率最低的数据淘汰。

（5）allkeys-lru：从数据集中挑选最近最少使用的数据淘汰

（6）allkeys-lfu：从数据集中挑选使用频率最低的数据淘汰。

（7）allkeys-random：从数据集（server.db[i].dict）中任意选择数据淘汰

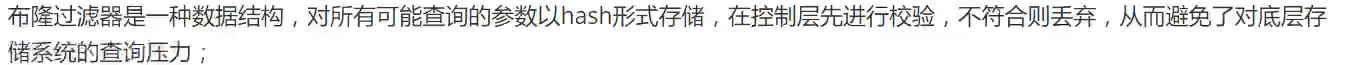
（8） no-enviction（驱逐）：禁止驱逐数据，这也是默认策略。意思是当内存不足以容纳新入数据时，新写入操作就会报错，请求可以继续进行，线上任务也不能持续进行，采用no-enviction策略可以保证数据不被丢失。

**5、解决Redis缓存穿透、缓存击穿、缓存雪崩**

**缓存穿透**

查询一个一定不存在的数据，导致这个不存在的数据每次请求都要到存储层去查询

解决方案：布隆过滤器/缓存空对象



直观的说，bloom算法类似一个hash set，用来判断某个元素（key）是否在某个集合中。  
和一般的hash set不同的是，这个算法无需存储key的值，对于每个key，只需要k个比特位，每个存储一个标志，用来判断key是否在集合中。

算法：  
1. 首先需要k个hash函数，每个函数可以把key散列成为1个整数  
2. 初始化时，需要一个长度为n比特的数组，每个比特位初始化为0  
3. 某个key加入集合时，用k个hash函数计算出k个散列值，并把数组中对应的比特位置为1  
4. 判断某个key是否在集合时，用k个hash函数计算出k个散列值，并查询数组中对应的比特位，如果所有的比特位都是1，认为在集合中。

优点：

不需要存储key，节省空间

缺点：

1. 算法判断key在集合中时，有一定的概率key其实不在集合中  
2. 无法删除

**缓存击穿**

热频Key持续接受高并发访问，当key失效瞬间就会穿透缓存直接请求数据库

解决方案：设置热点Key永不过期/互斥锁

**缓存雪崩**

缓存集中过期/Redis宕机

解决方案：

宕机：Redis集群（异地多活）

互斥锁

数据预热：手动加载不同的key，设置不同的过期时间，尽量均匀失效

**6、Redis持久化**

**RDB（Redis DataBase）**

子进程定期（配置文件）将Redis快照写入RDB文件

**优势**

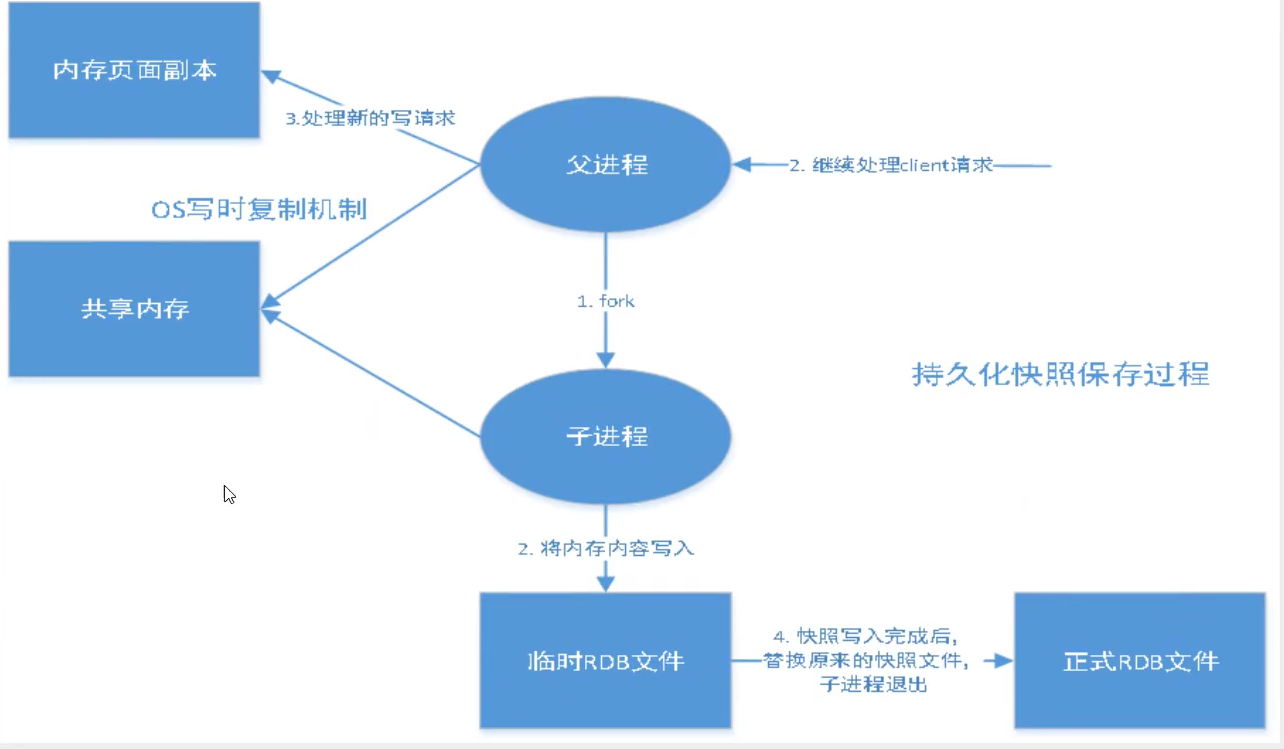
（1）RDB文件紧凑，全量备份，非常适合用于进行备份和灾难恢复。

（2）生成RDB文件的时候，redis主进程会fork()一个子进程来处理所有保存工作，主进程不需要进行任何磁盘IO操作。

（3）RDB 在恢复大数据集时的速度比 AOF 的恢复速度要快。

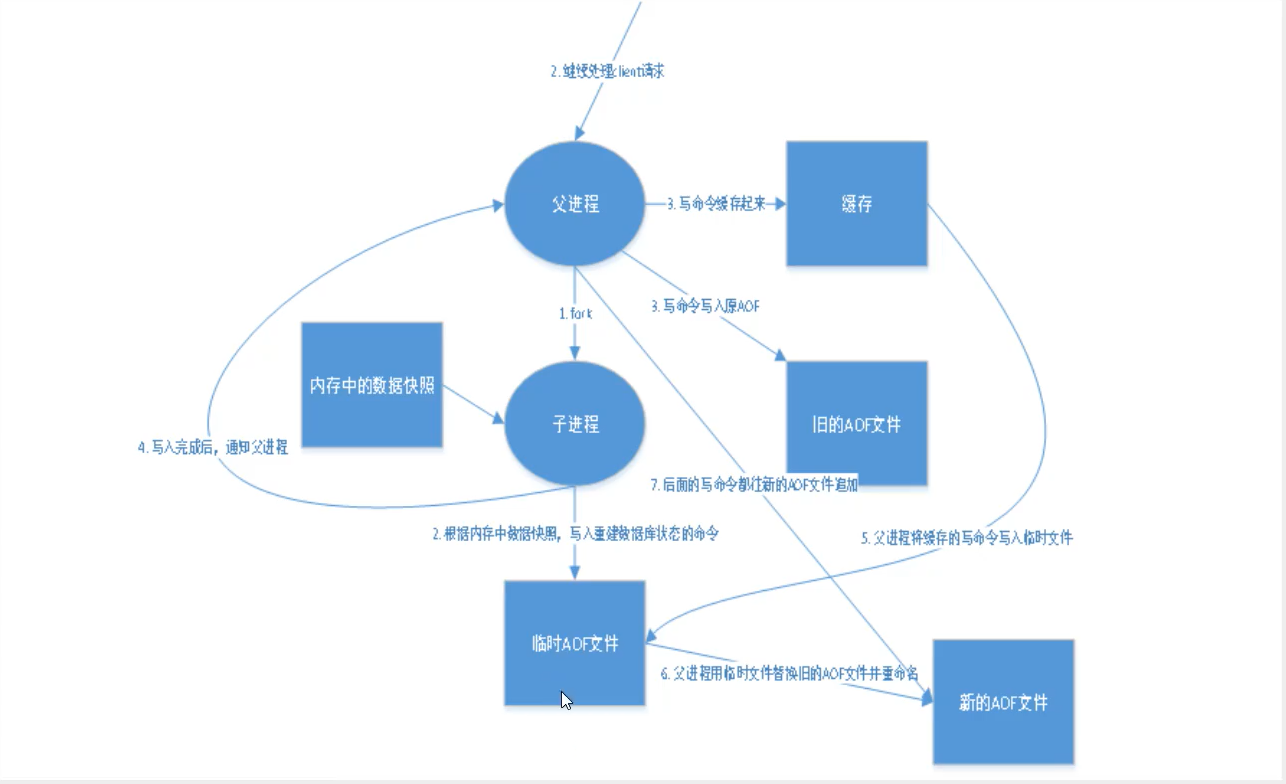
**劣势**

当进行快照持久化时，会开启一个子进程专门负责快照持久化，子进程会拥有父进程的内存数据，父进程修改内存子进程不会反应出来，所以在**快照持久化期间修改的数据不会被保存**，可能丢失数据。



**AOF（Append Only File）**

以日志的形式将所有写操作增量记录在一个.aof文件中，无限追加



**优点**

（1）AOF可以更好的保护数据不丢失，一般AOF会每隔1秒，通过一个后台线程执行一次fsync操作，最多丢失1秒钟的数据。

（2）AOF日志文件的命令通过非常可读的方式进行记录，这个特性非常适合做灾难性的误删除的紧急恢复。比如某人不小心用flushall命令清空了所有数据，只要这个时候后台rewrite还没有发生，那么就可以立即拷贝AOF文件，将最后一条flushall命令给删了，然后再将该AOF文件放回去，就可以通过恢复机制，自动恢复所有数据

**缺点**

（1）对于同一份数据来说，AOF日志文件通常比RDB数据快照文件更大

（2）AOF开启后，支持的写QPS会比RDB支持的写QPS低，因为AOF一般会配置成每秒fsync一次日志文件，当然，每秒一次fsync，性能也还是很高的

**\*\*7、Redis为什么这么快**

1、Redis是完全基于**内存**的数据库

2、**数据结构**简单，操作节省时间

3、处理网络请求使用的是**单线程**，避免了不必要的上下文切换和锁的竞争维护。

4、使用了I/O**多路复用**模型。（I/O指的是网络I/O，多路指的是多个网络连接，复用指的是复用一个线程）

1. 在Redis中的I/O多路复用程序会监听多个客户端连接的Socket

2. 每当有客户端通过Socket流向Redis发送请求进行操作时，I/O多路复用程序会将其放入一个队列中。

3. 同时I/O多路复用程序会同步、有序、每次传送一个任务给处理器处理。

4. I/O多路复用程序会在上一个请求处理完毕后再继续分派下一个任务。（同步）

**8、Redis实现分布式锁**

**SET命令**

SET key value [EX seconds] [PX milliseconds] [NX|XX]

参数说明：

**EX second** ：设置键的过期时间为 second 秒。 SET key value EX second 效果等同于 SETEX key second value 。

**PX millisecond** ：设置键的过期时间为 millisecond 毫秒。 SET key value PX millisecond 效果等同于 PSETEX key millisecond value 。

**NX** ：只在键不存在时，才对键进行设置操作。 SET key value NX 效果等同于 SETNX key value 。

**XX** ：只在键已经存在时，才对键进行设置操作。

**9、Redis如何做内存优化**

1、缩减键值对象

　　缩减键（key）和值（value）的长度，

key长度：如在设计键时，在完整描述业务情况下，键值越短越好。

value长度：首先应该在业务上精简业务对象，去掉不必要的属性避免存储无效数据。其次在序列化工具选择上，应该选择更高效的序列化工具来降低字节数组大小。

2、共享对象池

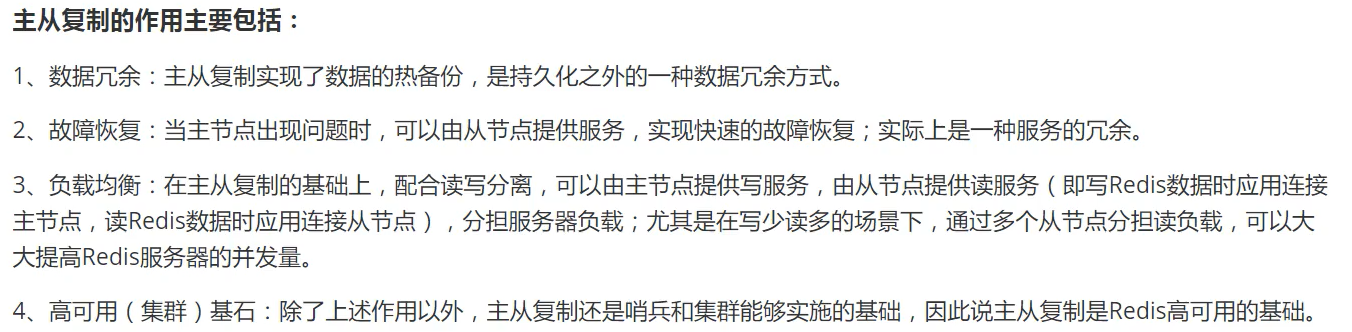
对象共享池指Redis内部维护[0-9999]的整数对象池。创建大量的整数类型redisObject存在内存开销，每个redisObject内部结构至少占16字节，甚至超过了整数自身空间消耗。所以Redis内存维护一个[0-9999]的整数对象池，用于节约内存。 除了整数值对象，其他类型如list,hash,set,zset内部元素也可以使用整数对象池。因此开发中在满足需求的前提下，尽量使用整数对象以节省内存。

3、字符串优化

4、编码优化

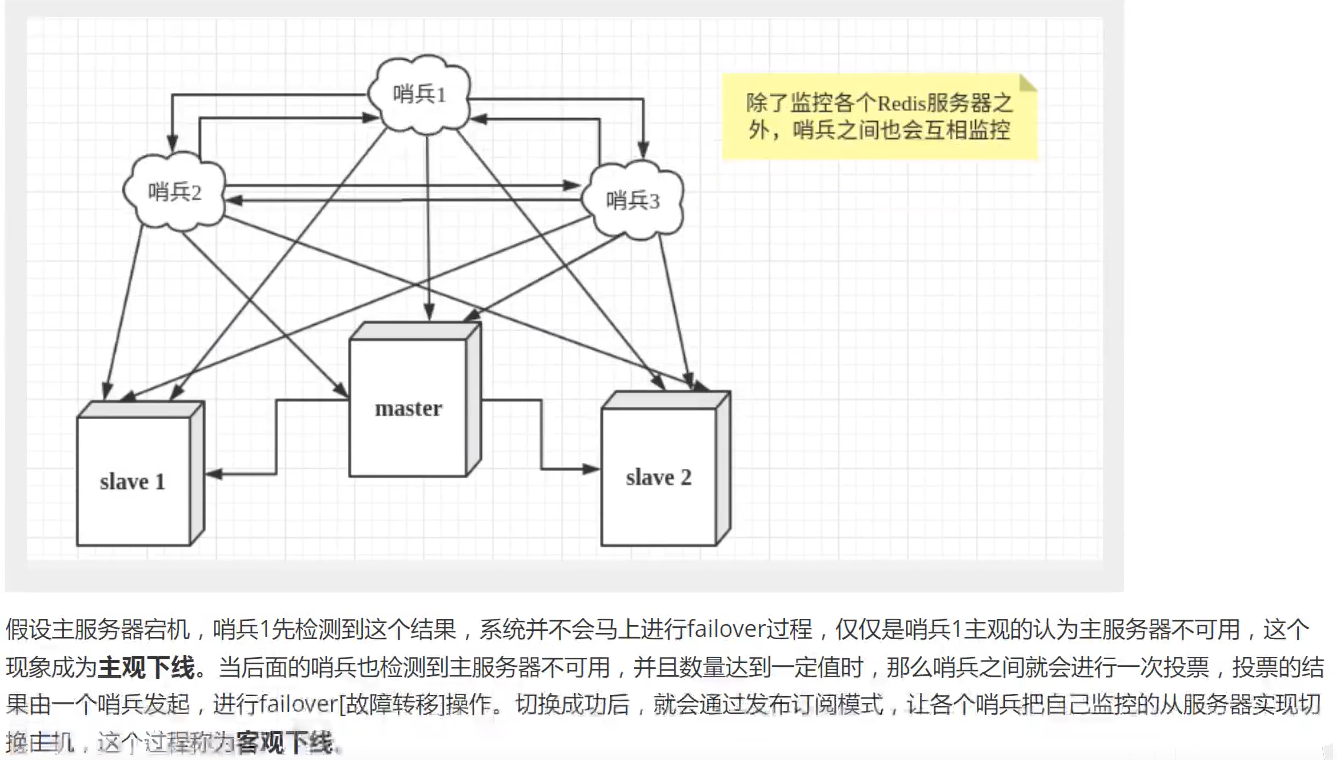
5、控制key的数量

**10、主从复制**



配置端口：一主二从

哨兵模式：哨兵通过发送命令，等待Redis服务器响应，从而监控运行的多个Redis实例



**\*\*11、MemCache和Redis的区别**

**MemCache**

MemCache的工作流程如下：先检查客户端的请求数据是否在memcached中，如有，直接把请求数据返回，不再对数据库进行任何操作；如果请求的数据不在memcached中，就去查数据库，把从数据库中获取的数据返回给客户端，同时把数据缓存一份到memcached中（memcached客户端不负责，需要程序明确实现）；每次更新数据库的同时更新memcached中的数据，保证一致性；当分配给memcached内存空间用完之后，会使用LRU（Least Recently Used，最近最少使用）策略加上到期失效策略，失效数据首先被替换，然后再替换掉最近未使用的数据。

Memcache是一个高性能的[分布式](https://baike.baidu.com/item/%E5%88%86%E5%B8%83%E5%BC%8F)的内存对象缓存系统，通过在[内存](https://baike.baidu.com/item/%E5%86%85%E5%AD%98)里维护一个统一的巨大的[hash表](https://baike.baidu.com/item/hash%E8%A1%A8)，它能够用来存储各种[格式](https://baike.baidu.com/item/%E6%A0%BC%E5%BC%8F)的数据，包括[图像](https://baike.baidu.com/item/%E5%9B%BE%E5%83%8F)、[视频](https://baike.baidu.com/item/%E8%A7%86%E9%A2%91)、[文件](https://baike.baidu.com/item/%E6%96%87%E4%BB%B6)以及[数据库](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93)检索的结果等。简单的说就是将数据调用到[内存](https://baike.baidu.com/item/%E5%86%85%E5%AD%98)中，然后从内存中读取，从而大大提高读取速度。

1、Redis和Memcache都是将数据存放在内存中。

2、Redis不仅仅支持简单的k/v类型的数据，同时还提供list，set，hash等数据结构的存储。

4、过期策略--memcache在set时就指定，例如set key1 0 0 8,即永不过期。Redis可以通过例如expire 设定，例如expire name 10

5、分布式--设定memcache集群（平行关系）;redis可以做一主多从。

6、存储数据安全--memcache挂掉后，数据没了；redis可以定期保存到磁盘（持久化）,支持持久化，服务器重启，缓存依然可以获取。