redis个人总结

redis对象

- 1.字符串
- 2.列表
- 3.哈希
- 4.集合
- 5.有序集合

简单动态字符串(SDS)

- 1.redis中C字符串只作为字符串字面量用在一些无须对字符串值进行修改的地方。例如打印日志。
- 2.redis使用SDS(Simple Dynamic String)来表示字符串。
    - 1.O(1)获得字符串长度。
        - 结构体记录SDS长度，无须遍历整个SDS来获得长度。
    - 2.杜绝缓冲区溢出。
        - 自动扩展SDS的长度。
    - 3.减少内存重分配次数。
        - 1.空间预分配
            - 1.如果修改之后的SDS长度小于1MB，new_size=old_size+old_size+1byte(1字节是用来保存'\0')。
            - 1.如果修改之后的SDS长度大于等于1MB，new_size=old+1MB+1byte。
        - 2.惰性空间释放
            - 当需要缩短SDS的长度，程序使用free属性来标明该空间已经被释放，并不是真正的释放。
                - SDS也提供了API，可以真正的释放SDS空间。
    - 4.二进制安全。
        - 因为SDS使用len在判断字符串是否结束，而不是'\0'，所以字符串中间有'\0'，也不会提前结束字符串。
    - 5.兼容部分C字符串函数。

链表(linkedlist)

- 1.redis链表特性
    - 1.双端
    - 2.无环
    - 3.表头指针和表尾指针
    - 4.记录链表长度
    - 5.多态
        - 链表使用void指针来保存节点值，可以通过dup,free,match为节点值设定相应类型，所以链表可以保存各种不同类型的值。

字典(hashtable)

- 1.redis的字典底层用哈希表来实现，每个字典带有两个哈希表，一个保存元素，另一个进行rehash时使用。
- 2.字典的key不允许重复。
- 3.程序根据字典的key先计算哈希值，然后再计算索引值，插入到指定索引上。
- 4.哈希冲突
    - 使用开链法解决key冲突，新节点插入到表头位置(头插法)。
- 5.rehash
    - 扩展
        - 1.当服务器没有执行BGSAVE或BGREWRITEAOF操作，且哈希表的负载因子>=1时，执行拓展操作.
        - 2.当服务器执行BGSAVE或BGREWRITEAOF操作，且哈希表的负载因子>=5时，执行拓展操作.
    - 收缩
        - 当哈希表的负载因子小于0.1时，程序自动对哈希表进行收缩操作。
    - rehash流程
        - 为hashtable[1]分配空间
            - 扩展操作
                - hashtable[1]的大小与2^n中第一个>=hashtable[0].used*2的值.(例如hashtable[0].used=3,2^3>=3*2,所以hashtable[1]的长度为8)
            - 收缩操作
                - hashtable[1]的大小与2^n中第一个>=hashtable[0].used的值.(例如hashtable[0].used=3,2^2>=3,所以hashtable[1]的长度为4)
            - 将hashtable[0]的值保存到hashtable[1]上,并重新计算哈希值和索引值.
            - 将hashtable[0]置为空,释放hashtable[0],hashtable[1]设置为hashtable[0],并新创建一个空白哈希表作为hashtable[1]
    - 渐近rehash
        - 在渐进rehash期间中,字典的删除,查找,更新等操作都会在两个哈希表上进行.

跳跃表(skiplist)

- 1.跳跃表是一种有序数据结构,通过每个节点中维持多个指向其他节点的指针,来达到快速访问节点的目的.
- 2.跳跃表支持平均O(logn),最坏O(n)的时间复杂度查找节点,还可以通过顺序性操作来批量处理节点.
- 3.跳跃表实现
    - 每个跳跃节点从层高都是1到32之间的随机数.
    - 每个跳跃节点的成员对象必须是唯一的.
    - 跳跃节点按照分值大小进行排序,当分值相同时,节点按照队员对象大小进行排序.

整数集合(intset)

- 1.当一个集合中只包含整数数值元素,并且元素数量不多时,redis使用整数集合作为集合键的底层实现.
- 2.整数集合元素保存在contents数组中,升序,无重复项.
- 3.contents数组的类型取决与encoding属性的值.
- 4.升级
    - 如果新加入到元素比数组现有元素的类型都要长,进行升级操作.
        - 1.拓展contents数组的空间大小.
        - 2.将所有的元素都转换为与新元素相同的类型.
        - 3.在数组末尾插入新元素.

- 5.降级
    - 整数集合不支持降级操作,一旦对数组进行升级,编码就会一直保持升级后的状态.

压缩列表**(ziplist)**

- 压缩列表是一种节约内存的顺序数据结构.
- 1.当一个列表键只包含少量元素,且每个元素要么是小整数值,要么是比较短的字符串,redis就使用压缩列表作为列表键的底层实现.
- 2.当一个列哈希键只包含少量键值对,且每个键值对的键和值要么是小整数值,要么是比较短的字符串,redis就使用压缩列表作为哈希键的底层实现.
- 3.压缩列表构成
    - 1.zibytes(uint32_t,4bytes),记录压缩列表占用的内存字节数.
    - 2.zlzail(uint32_t,4bytes),记录表尾节点距压缩列表起始地址字节数.
    - 3.zllen(unint16_t,2bytes),记录压缩列表所含节点数,小于2^16-1(65535)则直接记录节点数,如果zllen==2^16-1(65535),节点的真实数量需要遍历整个压缩列表才能得出.
    - 4.entryX(列表节点,不定),长度由节点保存的内容决定.
    - 5.zlend(uint8_8,1byte),特殊值0xFF,标记压缩列表的末端.
- 4.entryX(压缩列表节点)构成
    - 1.previous_entry_length.记录前一个节点的长度
        - 1字节.如果前一节点长度<254字节.
        - 5字节.如果前一节点长度>=254字节.其中第一字节被设置为0xFE,之后4个字节记录前一节点的长度.
    - 2.encoding.记录节点所保存数据的类型以及长度.
        - 1字节,2字节或5字节长,值最高位是00,01或10时,表示content保存的是数组,而数组长度由编码剩下的bits去保存.
        - 1字节长,且最高为是11,说明content保存的是整数,整数的类型和长度由编码剩下的bits去保存.
    - 3.content.保存节点的值.
- 5.连锁更新
    - 添加新节点或者删除节点都有可能依法连锁更新操作.

对象

- 1.redis中每个键值对的键和值都是一个对象.
- 2.每种类型的对象都至少由两种或以上的编码方式,不同的编码方式可以在不同的场景上优化对象的使用效率.
    - 1.字符串对象
        - 1.int.如果字符串对象保存的是一个可以用long表示整数值,那么该字符串的编码方式是int.
        - 2.raw.如果字符串对象保存的是一个长度>39字节的字符串,那么编码方式为raw.
            - raw编码会调用两次内存分配(一次分配redisObject结构,一次分配sdshdr结构).
        - 3.embstr.如果字符串对象保存的是一个长度<=39字节的字符串,那么编码方式为embstr.
            - embstr只调用一次内存分配空间,空间一次包含redisObject结构和sdshdr结构,放在一块连续的内存中.
            - embstr是只读的,如果修改embstr编码的字符串,则字符串对象会先变成raw编码的,再修改字符串对象.
    - 2.列表对象

        - 1.ziplist(压缩列表).当列表对象所保存的所有字符串长度都小于64字节且列表对象保存的字符串数量小于512个,使用ziplist编码.
        - 2.linkedlist(双端链表)
    - 3.哈希对象
        - 1.ziplist.当哈希对象所保存的键值对和键和值长度都小于64字节且列表对象保存的键值对数量小于512个,使用ziplist编码.
            - 同一键值对的键值的两个节点总是紧挨在一起,键的节点在前,值的节点在后.
            - 新的键值对插入到压缩列表的表尾.
        - 2.hashtable.哈希对象中每个键值对都使用一个字典键值对来保存.
            - 字典的键是字符串对象,对象中保存哈希对象的键值对的键
            - 字典的值是字符串对象,对象中保存哈希对象的键值对的值
    - 4.集合对象
        - 1.intset.集合对象中保存的元素都是整数值,且元素数量<=512个,使用intset编码.
        - 2.hashtable
            - 字典中的每个键都保存着一个集合元素,字典的值全部被设置为NULL.
    - 5.有序集合对象.有序集合对象保存元素成员长度都<64字节,且元素数量<128.
        - 1.ziplist.有序集合对象保存元素成员长度都<64字节,且元素数量<128,使用ziplist编码.
            - 每个集合元素使用两个紧挨着的节点来保存,第一个节点保存元素的成员,第二个节点保存元素的分值.
            - ziplist的元素按照升序来排序,分值小的在表头,分值大的在表尾.
        - 2.skiplist.
            - 1.skiplist编码同时使用一个字典和一个跳跃表来保存元素.
                - 跳跃表节点的object保存了元素的成员,score属性保存元素的分值.
                - 字典的键保存元素,字典的值保存元素的分值.
            - skiplist适合快速的范围操作,字典适合快速的单个元素查找O(1),所以同时使用字典和skiplist来保存元素.
- 3.服务器在执行命令前,会先检查给定键的类型是否执行指定的命令.
- 4.redis对象使用引用计数实现内存回收机制.
- 5.redis会共享值为0至9999这一万个字符串对象.
- 6.对象会记录自己最后一次被访问的时间,保存到lru属性中.

数据库

- redis默认创建16个数据库.
- 通过expired(以秒为单位)或pexpired(以毫秒为单位)命令设置键的生存时间.
- 通过TTL(以秒为单位)或PTTL(以毫秒为单位)返回键的剩余生存时间.
- 过期删除策略
    - 1.定时删除.键过期后立即删除.
    - 2.惰性删除.放任键过期不管,当获取键的时候,检测键是否过期,过期则删除.
        - 所有读取数据库的命令之前都先调用expireIfNeeded函数对输入键进行检查,如果过期,则删除该键,且返回NULL.
    - 3.定期删除.每隔一段时间,redis对数据库进行一次检查,删除里面的过期键.
        - 在规定的时间内,分多次遍历服务器中的各个数据库,从数据库中的expired字典中随机检查一部分键的过期时间,并删除其中的过期键.

过期键对持久化的影响

- 1.使用SAVE/BGSAVE创建一个新的RDB文件时,redis会对数据库中的键进行检查,所以已过期的键不会被保存到RDB文件中.
- 2.载入RDB文件
    - 1.如果服务器以主服务器模式运行时,载入RDB文件,redis会对RDB文件中的键进行检查,过期键会被忽略,所以过期键不会对其造成影响.
    - 2.如果服务器以藏服务器模式运行时,载入RDB文件,redis不会对RDB文件中的键进行检查,都载入到从服务器中.不过因为主从服务器在进行数据同步时,从服务器的数据库就会清空,所以过期键也不会对其造成影响.
- 3.当服务器以AOF持久化模式运行时,如果数据库中的键过期,但它还没有被惰性删除或定期删除,那么AOF文件不会因为过期键而产生任何影响.
    - 当过期键被惰性删除或定期删除,redis会向AOF文件追写一条DEL命令,以显式记录该键已被删除.

过期键对复制的影响

- 1.主服务器在删除一个过期键后,会显式向所有从服务器发送一个DEL命令,通知从服务器删除这个过期键.
- 2.从服务器在读取过期键时,遇到过期键也不会将其删除,而是当成未过期的键来处理.
- 3.从服务器只有接收到主服务器发送的DEL命令,才会删除过期的键.

持久化

- 1.RDB持久化
    - 1.使用SAVE/BGSAVE命令,向磁盘创建RDB文件,并同步数据库.
        - 1.SAVE命令会组塞redis服务器进程,直到RDB文件创建完毕.
        - 2.BGSAVE命令会派生出一个子进程,然后子进程扶着创建RDB文件,不会阻塞服务器进程.
    - 2.载入RDB文件
        - 1.RDB文件是在服务器启动时自动执行的,当检测到RDB文件的存在,服务器就会自动载入RDB文件.
        - 2.如果服务器开始了AOF持久化功能,那么服务器会优先使用AOF文件来还原数据库状态.只有服务器关闭了AOF持久化功能,服务器才使用RDB文件来还原数据库状态.
    - 3.在BGSAVE执行过程中,客户端发送的SAVE/BGSAVE命令会被服务器拒绝.
    - 4.在BGSAVE执行过程中,客户端发送的BGREWRITEAOF命令会被延迟到BGSAVE命令执行结束后执行.
    - 5.在BGREWRITEAOF执行期间,客户端发送的BGSAVE会被服务器拒绝.
    - 6.BGSAVE和BGREWRITEAOF写不同的文件,不会产生冲突.不能同时执行的原因是处于性能的考虑.
    - 7.RDB文件是一个经过压缩的二进制文件,由多个部分组成.
        - 1.REDIS.5bytes,保存'REDIS'这五个字符.快速检查是否是RDB文件.
        - 2.db_version.4bytes,记录RDB的版本号.
        - 3.databases.包含零个或多个数据库.
            - 1.SELECTDB.1byte,标志下面将读入数据库号
            - 2.db_number.数据库号,1byte,2bytes或5bytes
            - 3.key_value_pair.保存的键值对.
        - 4.EOF.1byte,标志RDB文件正文内容结束.
        - 5.check_sum.8bytes,校验和.
    - 8.对于不同类型的键值对,RDB文件会使用不同的方式来保存.
    - 9.服务器状态会保存所有用save选项设置的保存条件,当任意一个保存条件被满足时,服务器自动执行BGSAVE命令.
- 2.AOF持久化

- AOF持久化是通过保存redis服务器执行的写命令来记录数据库的状态.
- AOF持久化的实现
  - 1.命令追加
    - 当AOF持久化功能处于打开状态时,服务器在执行完一个写命令后,会以协议格式将写命令追加到服务器状态的aof_buf缓冲区的末尾.
  - 2.文件写入
    - 服务器每次结束一次事件循环之前,调用flushAppendOnlyFile函数,考虑是否将aof_buf缓冲区的内容写入和保存到AOF文件中.
      - 1.always.总是将aof_buf缓冲区的内容写入并同步到AOF文件中
      - 2.everysec.将aof_buf缓冲区的内容写入到AOF文件中,如果距离上次同步AOF文件的时间超过1秒钟,则再次同步AOF文件.
      - 3.no.将aof_buf缓冲区的内容写入到AOF文件中,由操作系统决定何时同步AOF文件.
  - 3.文件同步
- AOF文件载入和数据还原
  - 1.AOF文件载入流程
    - 1.创建一个不带网络连接的伪客户端
      - 因为redis的命令只能在客户端上下文执行,而载入AOF文件时所使用的命令直接来自AOF文件而不是网络,所以创建一个不带网络连接的伪客户端.
      - 2.从AOF文件中分析并读出一条写命令.
      - 3.使用伪客户端执行被读出的写命令.
      - 4.重复执行2和3,知道AOF所有的命令被读出.
- AOF重写
  - 1.为解决AOF文件体积膨胀文件,redis使用AOF文件重写功能.
    - redis服务器创建一个新的AOF文件来替代现有的AOF文件,新的AOF文件不会包含任何浪费空间的冗余命令.
  - 2.redis将AOF重写程序放在子进程里执行
    - 1.子进程进行AOF重写期间,服务器进程可以继续处理命令请求.
    - 2.子进程带有服务器进程的数据副本,使用**子进程而不是线程**,是因为可以避免使用锁的情况下,保证数据的安全性.
  - 3.执行AOF重写期间,服务器执行了写命令,导致数据不一致.
    - 1.redis服务器设置了一个AOF重写缓冲区,当redis服务器执行一个写命令后,会同时将AOF缓冲区和AOF重写缓冲区.
  - 4.当子进程完成AOF重写之后,会向服务器进程发送信号,服务器接受到该信号,会调用一个信号处理函数,并执行一下工作
    - 1.将AOF重写缓冲区的所有内部同步到新的AOF文件中,此时新AOF文件所保存的状态与服务器一致.
    - 2.将新的AOF文件**原子地**覆盖旧AOF文件.

事件

- 1.文件事件.redis服务器通过套接字与客户端进行连接,文件事件是服务器对套接字操作的抽象.
  - 1.文件事件处理器使用I/O多路复用程序来同时监听多个套接字,并根据套接字目前执行的任务来为套接字关联到不同的事件处理器.
  - 2.文件事件处理器构成
    - 1.套接字
    - 2.I/O多路复用程序

        - 3.文件事件分派器
        - 4.事件处理器
            - 1.连接应答器.对连接redis服务器的客户端进行应答.
            - 2.命令请求处理器.接收客户端发送来的命令请求.
            - 3.命令回复处理器.向客户端返回命令请求的执行结果.
            - 4.复制处理器.执行主服务器和从服务器的复制操作.
    - 3.当上一个套接字产生的事件被处理完毕后,I/O多路复用程序才会继续向文件事件分派器传送下一个套接字.
    - 4.事件的类型
        - 1.当套接字变得可读或者新的可应答的套接字出现,套接字产生AE_READABLE事件.
        - 2.当套接字变得可写,套接字产生AE_WRITEABLE事件.
        - 如果一个套接字同时产生AE_READABLE事件和AE_WRITEABLE事件,文件事件分派器会优先处理AE_WRITEABLE事件,然后处理AE_READABLE事件.
- 2.时间事件.时间事件是服务器定时操作的抽象.
    - 1.定时事件.指定时间执行一次.
    - 2.周期事件.每隔指定时间就执行一次.
    - 实现
        - 服务器将所有的时间事件放在一个无序链表中,每当时间事件执行器运行时,遍历整个无序链表,查到已经到达时间的事件,并调用相应的事件处理器.
            - 无序链表并不会影响时间事件处理器的性能,无序链表几乎退化成一个指针来使用
                - 在正常模式下,redis服务器只使用serverCron一个时间事件.
                - 在benchmark模式下,redis服务器只使用两个时间事件.
        - 新的时间事件总是插入到表头.
- 对文件事件和时间事件的处理都是同步,有序,原子地执行.
- 文件事件处理后,检查是否由时间事件需要处理.所以时间事件实际处理世家通常要比设定的到达事件稍晚一些.

客户端

- 1.redis服务器状态结构的client属性是一个链表,该链表保存了所有与该服务器连接的客户端的状态结构.
- 2.套接字描述符(fd)
    - fd==-1,说明该客户端是伪客户端.
    - fd>-1,说明该客户端是普通客户端.
- 3.客户端的输入缓冲区保存客户端发送的命令请求,大小会根据输入内容动态增长或缩小,但是最大大小不能超过1GB,否则服务器将会关闭这个客户端.
- 4.客户端的输出缓冲区保存服务器返回的执行结果,每个客户端有两个输出缓冲区
    - 1.固定大小的输出缓冲区用于保存长度比较小的回复.
    - 2.可变大小输出缓冲区用于保存长度比较大的回复.
- 5.关闭客户端
    - 1.客户端进程退出或被杀死
    - 2.客户端向服务器发送带有不符合协议格式的命令请求
    - 3.客户端成为了CLIENT KILL命令的目标
    - 4.客户端发送的命令请求的大小超过了输入缓冲区的限制大小
    - 5.服务器发送给客户端的命令回复超过了输出缓冲区的限制大小
    - 6.服务器设置了timeout配置选项,当客户端的空转时间超过timeout设定的值.
- 6.服务器限制客户端输出缓冲区的大小的两种模式
    - 1.硬性限制.如果输出缓冲区的大小超过了硬性限制的大小,那么服务器立即关闭客户端.

    - 2.软性限制..如果输出缓冲区的大小超过了软性限制的大小,但是没有超过硬性限制的大小,那么服务器会继续监视客户端,并记录下客户端到达软性限制的起始时间.如果之后输出缓冲区一直超过软性限制,且超过服务器设定的时长,则关闭客户端.
- 7.处理lua脚本的伪客户端会一直存在,直到服务器关闭.
- 6.载入AOF文件的伪客户端在载入工作完成后自动关闭.

服务器

- 命令请求从发送到完成的处理步骤
    - 1.客户端将命令请求发送给服务器.
    - 2.服务器读取命令请求,并分析命令参数.
    - 3.命令执行器根据参数查找命令的实现函数,然后执行实现函数并得到命令回复.
    - 4.服务器将命令回复发送给客户端.
- serverCron函数默认每隔100ms执行一次,主要功能包括更新服务器状态,处理服务器接收的SIGTERM信号,管理客户端资源和数据库状态,检查并执行持久化操作等.
- 服务器启动步骤
    - 1.初始化服务器状态
    - 2.载入服务器配置
    - 3.初始化服务器数据结构
    - 4.还原数据库状态
    - 5.执行事件循环

复制

- 用户可以执行SLAVEOF命令或者设置slaveof选项，让一个服务器(从服务器)去复制另一个服务器(主服务器)。
- 旧版复制功能
    - 1.同步。同步操作时将从服务器的数据库状态更新至主服务器当前所处的数据库状态。
        - 1.从服务器向主服务器发送SYNC命令。
        - 2.收到SYNC命令的主服务器执行BGSAVE命令，在后台生成一个RDB文件，并使用一个缓冲区记录从下面睡觉哦老司机哦执行的所有写命令。
        - 3.主服务器的BGSAVE命令执行完毕时，主服务器会将BGSAVE命令生成的RDB文件发送给从服务器，从服务器接收并载入这个RDB文件，将从服务器的更新至主服务器执行BGSAVE命令时的数据库状态。
        - 4.主服务器将记录在缓冲区里面的所有的写命令都发送到从服务器，从服务器执行这些写命令，将自己的数据库状态更新至主服务器的数据库的当前所处的状态，从而得到数据库的状态一致性。
    - 2.命令传播。命令传播操作用于在主服务器的数据库状态被修改，导致主从服务器的数据库状态出现不一致时，让主从服务器的数据库重新回到一直状态。
        - 主服务器将执行的写命令发送给从服务器，从服务器执行相同的写命令，主从服务器再次回到一致位置。
- 新版复制功能的实现
    - 为解决旧版复制功能处理断线重时主从服务器复制的低效问题，剔除了部分重同步策略。使用PSYNC命令代替SYNC命令。
    - 1.完整重同步，与旧版同步功能一致。
    - 2.部分重同步

        - 当从服务器锻炼后重新连接主服务器时，主服务器将连接断开期间所执行的写命令发送给从服务器，从服务器执行这些写命令后，就可以将数据库状态与主服务器的数据库状态保持一致。
        - 实现
            - 1.主服务器的复制偏移量和从服务器的复制偏移量。
                - 主服务器和从服务器分别维护自身的复制偏移量。
                    - 1.主服务器每向从服务器传播N个字节的数据时，就将自身的复制偏移量的值加上N。
                    - 2.从服务器每次收到主服务器传播来的N个字节的数据时，就将自身的复制偏移量的值加上N。
                - 如果主从服务器的复制偏移量相同，说明当前主从服务器处于一致状态。
                - 如果主从服务器的复制偏移量不同，说明当前主从服务器处于非一致状态。
            - 2.主服务器的复制积压缓冲区。
                - 复制积压缓冲区时由主服务器维护的一个固定长度的先进先出队列，默认大小为1MB。当主服务器执行命令传播时，同时将命令传播给从服务器和写到复制积压缓冲区中。
                - 如果从服务器复制偏移量及其之后的数据仍然存在于复制积压缓冲区，则主服务器执行部分重同步操作。。否则执行完整重同步操作。
            - 3.服务器的运行ID。
                - 主从服务器保存自己的运行ID，从服务器也保存当前连接的主服务器的运行ID。
                - 如果从服务器保存的主服务器运行ID和当前连接的主服务器的运行ID相同，说明从服务器短线钱连接的主服务器正是当前连接的主服务器，主服务器可以尝试执行部分重同步操作。
                - 如果从服务器保存的主服务器运行ID和当前连接的主服务器的运行ID不同，说明从服务器短线钱连接的主服务器不是当前连接的主服务器，主服务器执行完整重同步操作。
- 心跳检测
    - 在命令传播阶段，从服务器默认以每秒一次的频率，向主服务器发送命令。
        - 检测主从服务器的网络连接状态。
            - 主从服务器通过发送REPLCONF_ACK命令来检测网络连接是否正常。
            - 如果主服务器超过1秒没有收到REPLCONF_ACK命令，那么主服务器就知道网络连接状态出现了问题。
        - 辅助实现min-slave选项。
            - redis的min-slaves-to-write和min-slaves-max-lag来防止主服务器在不安全的情况下执行写命令。（从服务器小于min-slaves-to-write或从服务器的延迟都>=min-slaves-max-lag，主服务器将拒绝写）。
        - 检测命令丢失。
            - 当主服务器命令传播从服务器的命令丢失了，当从服务器向主服务器发送REPLCONF_ACK命令时，主服务器会发觉从服务器的复制偏移量小于自身的复制偏移量，然后主服务器根据从服务器的复制偏移量在复制积压缓冲区找到丢失的命令，并将这些命令重新发送给从服务器。

哨兵(Sentinel)

- 由一个或多个哨兵施例组成的哨兵系统可以监视任意多个主从服务器，并且当被监视的主服务器进入下线状态，自动将其从属的某个从服务器提升为新的主服务器。

- 如果下线的主服务器重新上线，哨兵系统会将其降为从服务器。
- 哨兵会创建两个连向主服务器的异步网络连接
    - 命令连接。专门用于向主服务器发送命令，并接收命令回复。
        - 哨兵必须向主服务器发送命令，用来和主服务器通信，所以哨兵必须向主服务器创建一个命令连接。
    - 订阅连接。专门用于订阅主服务器的__sentinel__:hello频道。
        - 为了不丢失__sentinel__:hello频道的任何信息，哨兵必须专门用一个订阅连接来接收该频道的信息。
- 哨兵默认以10秒一次的频率，通过命令连接向被监视的主服务器发送INFO命令，并分析INFO命令的回复来获取主服务器的当前信息。
    - INFO消息回复包括主服务器自身的信息和其属下的所有从服务器的信息。
    - 主服务器实例结构的flags属性值为SRI_MASTER,而从服务器的为SRI_SLAVE。
    - 主服务器实例结构的name属性的值是用户使用哨兵配置文件设置的，从服务器则是哨兵根据从服务器的IP地址和端口号自动配置的。
- 当哨兵发现被监视的主服务器有新的从服务器出现时，哨兵除了会为这个新的从服务器创建相应的实例结构之外，哨兵同时创建连接从服务器的命令连接和订阅连接。
- 哨兵之间不创建订阅连接，只创建命令连接。
    - 因为哨兵需要通过订阅主从服务器发来的频道信息来发现新的哨兵，所以需要创建订阅连接，而相互已知的哨兵只需要命令连接进行通信就足够了。
- 哨兵在默认情况下每秒一次的频率向所有与它相创建了命令连接的实例(包括主从服务器和其他哨兵)发送PING命令，并通过返回的PING命令回复来判断实例是否在线。
    - 当该哨兵认为该主服务器已经主观下线了，就向监视该主服务器的其他哨兵进行询问，询问他们的判断状态。
    - 当认为主服务器已经下线状态的哨兵的数量>quorum参数的值，则该哨兵就会认为该主服务器已经进入客观下线状态。

集群

- 集群通过分片来进行数据共享，并提供复制和故障转移功能。
- redis服务器会在启动时根据cluster-enabled配置选项来决定是否开启服务器的集群模式。
- redisClient结构中的套接字和缓冲区是用于连接客户端，而clusterLink结构中的套接字和缓冲区是用于连接节点。
- 节点会将新加入的节点的信息通过Gossip协议传播给集群中其他节点，让其他节点也与新加入的节点握手，经过一段时间之后最终新加入的节点会被集群中所有节点认识。
- 槽指派
    - 集群的整个数据库被分为16384个槽，数据库中的每个键都属于这16384个槽中一个，集群中的每个节点可以处理0至16384个槽。
    - 当数据库的16384个槽都有节点在处理，那么集群就处于上线状态。否则集群处于下线状态。
        - slots(二进制数组)数组在索引i上的二进制的值为1，就说明该节点负责处理槽i。
        - 集群中每个节点都将自己的slots数组通过消息发送给集群中其他节点，这样每个节点都知道16384个槽被分派给了哪些节点。
    - 当集群处于上线状态，客户端向集群中的节点发送命令请求，接收命令的节点计算出命令要处理的的数据库键属于哪个槽
        - 如果键所对应槽正好就是指派给了当前节点，那么当前节点直接执行命令。
        - 如果键所对应槽没有指派给当前节点，则节点向客户端返回一个MOVE错误，指引客户端转向至正确的节点，并再次发送之前的命令。
- 集群节点只能使用0号数据库，而单机服务器没有这一限制。

- 重新分片操作可以在线进行，集群不需要下线，且源节点和目标节点都可以继续处理命令请求。
    - 在重新分片过程中，可能会出现迁移槽的一部分键值对在源节点中，另一部分在目标节点中。
        - 源节点会先在自己的数据库查找指定的键，如果找到就直接执行命令；
        - 否则说明该键对应的槽已经迁移到目标节点中，源节点向客户端返回一个ASK错误，指引客户端转向目标节点，并发送之前的命令。
        - ASK错误和MOVED错误的区别
            - MOVED错误代表槽的负责权已经从一个节点转移到另外一个节点，客户端以后关于槽i的命令请求都发送到MOVED错误所指向的节点。
            - ASK错误是两个节点在迁移槽的过程中使用的临时措施，转向不会对客户端今后发送的关于槽i的命令请求产生影响，仍然发送至目前节点。
- 集群中的从节点用于复制主节点，主节点用于处理槽。主节点下线后，从节点代替主节点继续处理命令请求。
- 集群中的节点通过发送和接收消息来进行通信。
    - 1.MEET
        - 邀请接收者加入到发送者当前所处的集群中。
    - 2.PING
        - 集群中每个节点默认每隔1s就会从已知节点列表中随机选择5个节点，然后对这5个节点中最长时间没有发送过PING消息的节点发送PING消息，以检测该节点是否在线。
    - 3.PONG
        - 对MEET和PING的确认回复。
    - 4.PUBLISH
        - 不直接向节点广播PUBLISH命令是因为不符合redis集群中"各个节点通过发送和接收消息来进行通信"这一规则。
    - 5.FAIL
        - 当主节点判断另外主节点处于FAIL状态，则向集群广播关于这个节点的FAIL消息，所有收到这个消息的节点都会立即将这个节点标记为下线。

发布和订阅

- 1.客户端通过SUBSRIBE命令可以订阅一个或多个频道,服务器状态在pubsub_channels字典保存了所有频道的订阅关系.
- 2.客户端通过PSUBSRIBE命令可以订阅一个或多个模式,,服务器状态在pubsub_patterns链表保存了所有频道的订阅关系.
- 3.PUBLISH命令通过访问pubsub_channels字典来向所有的订阅者发布消息,通过访问pubsub_patterns链表来向所有匹配频道的模式的订阅者发布消息.

事务

- 1.事务过程
    - 1.事务开始
        - 1.MULTI命令将客户端从非事务状态切换至事务状态,通过打开客户端状态的flags属性的REDIS_MULTI.
    - 2.命令入队
        - 1.当客户端处于非事务状态时,发送的命令会立即被服务器执行.
        - 2.当客户端处于事务状态时,
            - 如果客服端发送的命令是EXEC/DISCARD/WATCH/MULTI,则服务器会立即执行该命令.

        - 否则,服务器不会立即执行该命令,而是将该命令放入一个事务队列中,并向客户端返回QUEUED回复.
            - 1.事务队列以先进先出的方式保存入队的命令.
    - 3.事务执行
        - 当处于事务状态的客户端向服务器发送EXEC命令,事务立即被执行.
- WATCH命令
    - WATCH命令是一个乐观锁,他可以在EXEC命令执行之前,监视任意数量的数据库键,并在EXEC命令执行时,如果检查到被监视的键被修改过了,则服务器拒绝执行事务,并向客户端返回代表事务执行失败的空回复.
- 事务的ACID性质
    - 原子性
        - redis不支持事务回滚机制,如果某个命令执行出现了错误,整个事务也会继续执行下去.
        - 不支持事务回滚机制是因为这种复杂的功能与redis追求简单高效的设计主旨不符.
    - 一致性
        - 事务在入队命令过程中,出现了命令不存在或者格式不正确,redis将拒绝执行这个事务.
        - 执行错误(不能在入队时被服务器检测出,执行错误只会在命令实际执行时被触发),服务器不会终止事务的执行,继续执行事务中余下的其他命令.
            - 因为执行错误会被服务器识别出来,并进行相应的处理(出错命令不会对数据库左任何修改),所有不会对事务的一致性产生任何影响.
        - 服务器停机
            - 如果服务器在无持久化的内存模式下运行时,中途停机,重启后的数据库是空白的,因此数据总是一致的.
            - 如果服务器在RDB持久化模式下运行,中途停机,重启后的数据库根据RDB文件来恢复数据,把数据库还原到一个一致的状态,因此数据总是一致的.
            - 如果服务器在AOF持久化模式下运行,中途停机,重启后的数据库根据AOF文件来恢复数据,把数据库还原到一个一致的状态,因此数据总是一致的.
    - 隔离性
        - 隔离性指的是数据库中多个事务并发地执行,各个事务之间也不会相互影响,且并发状态下的执行事务和串行执行的事务产生的结果完全相同.
        - 因为redis是使用单线程来执行事务,事务总是以串行的方式运行,所以事务具有隔离性.
    - 持久性
        - 持久性指的是当一个事务执行完毕后,执行这个事务所得到的结果已经被保存到永久性存储介质中.
        - redis的持久性由redis所使用的持久化模式决定
            - 如果服务器在无持久化的内存模式下运行时,此时事务不具有持久性.
            - 如果服务器在RDB持久化模式下运行,不能保证事务数据被第一时间保存到RDB文件中,所以此时事务不具有持久性.
            - 当服务器在AOF持久化模式下运行,
                - 1.当appendfsync值为always,可以保证事务数据被第一时间保存到AOF文件中,此时事务具有持久性.
                - 2.当appendfsync值为everysec,不能保证事务数据被第一时间保存到AOF文件中,此时事务不具有持久性.
                - 3.当appendfsync值为no,不能保证事务数据被第一时间保存到AOF文件中,此时事务不具有持久性.

慢查询日志

- 1.redis的慢查询日志是用于记录执行事件超过给定时长的命令请求.
- 2.服务器使用先进先出的方式保存多条慢查询日志.
- 3.新的慢查询日志会被添加到表头(头插法),如果日志数量超过showlog_max_len的值,则从表尾删除日志.

监视器

- 1.通过执行**MONITOR**命令,客户端可以将自己变成一个监视器,实时地接收并打印出服务器当前处理的命令请求相关信息,REDIS_MONITO标志会被打开.
- 2.服务器在每次处理命令前,都会调用replicationFeedMonitors函数,将被处理的命令请求相关信息发送给各个监视器.
- 3.服务器将所有的监视器都记录在monitors链表中.

二进制数组

- redis使用SDS来保存位数组.
- BITCOUNT命令使用查表算法和variable-precision SWAR算法来优化命令的执行效率.
- BITOP命令的所有操作都是使用C语言内置的位操作来实现.