Redis 源码解析

- 主要特点 (29)
- 持久化 (51)
- 事务 (57)
- 分布式 (54)
- 使用场景 (6)
- 数据结构与对象
  - 数据结构
    - 默认的字符串类型 SDS
      - Simple Dynamic String　带长度信息的字节数组
      - 代码位置：sds.h / struct sdshdr
      - SDS 与 C字符串的区别
        - C字符串获取长度操作的复杂度为 O(n)，SDS 通过 len 属性记录了长度，为 O(1)
        - C字符串可能会造成缓冲区溢出
SDS 会在修改时先检查空间是否足够
        - SDS 有空间预分配策略，可以减少连续分配内存操作的次数
        - SDS 有惰性空间释放策略，字符串缩小时，将多余的字节空间不释放，而是放到 free 空间
        - C字符串因为必须复合某种编码，所以只能存储文本数据
        - SDS 是二进制安全的，可以储存二进制数据
        - SDS 遵循了C字符串以空字符结尾的规则，可以重用部分C字符串功能
    - 链表 LinkedList　代码位置：adlist.h / listNode、list
    - 跳跃表 SkipList
      - 平均复杂度：O(logn)；最坏：O(n)
      - 代码位置：redis.h / zskiplistNode、zskiplist
    - 字典 Dict
      - 底层使用哈希表实现
      - 哈希表实现：dict.h / structure dictht、structure dictEntry
      - 字典实现：dict.h / structure dict
      - 渐进式哈希表扩容(rehash)
        - 给哈希表2分配更大的空间, 例如是当前哈希表1大小的两倍
        - 把哈希表1中的数据冲更新映射并拷贝到哈希表2
          - 拷贝数据时, redis仍然正常处理客户端请求
          - 每处理一个请求, 顺带从哈希表1中的第一个索引位置开始, 把这个位置上的所有的entry复制到哈希表2,下个请求就复制位置2
          - 直至全部复制完成
          - 具体实现
            - 在字典中维持一个索引计数器变量rehashidx, 并设置为0, 表示 rehash 开始
            - 在 rehash 期间，客户端每次对字典进行 CRUD 操作时，会将 ht [0] 中 rehashidx 索引上的值 rehash 到 ht [1]，操作完成后 rehashidx+1
            - 字典操作不断执行，最终在某个时间点，所有的键值对完成 rehash，这时将 rehashidx 设置为 - 1，表示 rehash 完成
        - 释放哈希表1的空间
        - 渐进式 rehash 执行时，除了根据针对字典的 CRUD 操作来进行数据迁移，Redis 本身还会有一个定时任务在执行 rehash，如果没有针对字典的请求时，这个定时任务会周期性地（例如每 100ms 一次）搬移一些数据到新的哈希表。
        - 在渐进式 rehash 过程中，字典会同时使用两个哈希表 ht [0] 和 ht [1]，所有的 CRUD 操作也会在两个哈希表进行。
        - 比如要查找一个键时，服务器会优先查找 ht [0]，如果不存在，再查找 ht [1]。当执行新增操作时，新的键值对一律保存到 ht [1]，不再对 ht [0] 进行任何操作，以保证 ht [0] 的键值对数量只减不增，最后变为空表。
    - 压缩列表 Ziplist
      - Redis 为了节约内存而开发，专门存储小整数值或短字符串
      - 是由一系列特殊编码的连续内存块组成的顺序型数据结构
    - 整数集合 IntSet　代码位置：intset.h / structure intset
  - 对象系统
    - 基于上面的数据结构创造了对象系统，根据场景不同，为对象设置不同的数据结构实现
    - 对象结构：structure redisObject
      - type 对象类型
        - 键总是字符串对象；而值可以是下面的类型
        - 字符串对象 REDIS_STRING
        - 列表对象 REDIS_LIST
        - 哈希对象 REDIS_HASH
        - 集合对象 REDIS_SET
        - 有序集合对象 REDIS_ZSET
      - encoding 对象编码
        - 也就是这个对象使用了什么底层数据结构实现
        - 使用 object encoding key 可以查看该值对象的编码
        - 字符串对象的编码
          - int　保存整数类型
          - raw　保存 > 32字节的字符串SDS
          - embstr
            - 保存 <= 32 字节的字符串SDS
            - 专门用于保存短字符串的一种优化编码方式
        - 列表对象的编码
          - ziplist
          - linkedlist
        - 哈希对象的编码
          - ziplist
          - hashtable
        - 集合对象的编码
          - intset
            - 所有元素都是整数值
            - 元素数量 < 512 个
          - hashtable
        - 有序集合对象的编码
          - ziplist
            - 元素数量 < 128 个
            - 所有元素成员长度 < 64 字节
          - skiplist
            - 此编码表示使用 ZSET 结构来实现该有序集合对象
            - ZSET结构包含 一个字典和一个跳跃表
              - 跳跃表按分值从小到大保存了所有集合元素，每个跳跃表节点都保存了一个集合元素：跳跃表节点的object属性保存了元素的成员，而跳跃表节点的score属性则保存了元素的分值。通过这个跳跃表，程序可以对有序集合进行范围型操作，比如ZRANK、ZRANGE等命令就是基于跳跃表API来实现的。
              - 字典为有序集合创建了一个从成员到分值的映射，字典中的每个键值对都保存了一个集合元素：字典的键保存了元素的成员，而字典的值则保存了元素的分值。通过这个字典，程序可以用O（1）复杂度查找给定成员的分值，ZSCORE命令就是根据这一特性实现的，而很多其他有序集合命令都在实现的内部用到了这一特性。
              - 两种数据结构都会通过指针来共享相同元素的成员和分值
      - ptr 指向底层实现数据结构的指针
      - refCount 引用计数
        - 因为C语言并不具备自动内存回收功能，所以Redis 通过 RedisObject 结构的 refCount 属性来记录对象引用信息
        - 基于引用计数技术，当程序不再使用某个对象，其所占内存会被自动释放
          - 考虑到验证共享对象与目标是否完全一致的操作开销，Redis 只共享整数值类型的字符串对象
        - 基于引用计数技术，实现了对象共享机制，多个键可以共享同一个对象来节约内存
      - lru 最近一次被访问的时间
        - object idletime key 可以查看该键的空转时长，通过当前时间减去 lru时间所得
        - object idletime 命令做了特殊处理，此命令不会改变值对象的 lru 属性
- 单机数据库实现 (89)
- 事件模型 (30)