redis-1

- 基本数据结构
  - 字符串
    - redis 没有是C语言的传统字符串(以空字符结尾的字符数组，以下简称 C 字符串) 自己构建了SDSC抽象模型
    - 使用SDS结构体的好处
      - 减少获取字符串长度的复杂度(C语言中要想获取一个字符串的长度需要挨个字符串遍历)
      - 杜绝缓冲区溢出(如果要进行修改操作时会提前判断空间是否需要扩容)
      - SDS通过预留空间减少内存重分配问题,SDS 实现了空间预分配和惰性空间释放两种优化策略
      - 二进制安全:所有 SDS API 都会以处理二进制的方式来处理 SDS 存放在 buf 数组里的数据 ,程序不会对其中的数据做任何限制、过滤、或者假设
      - 兼容部分C字符串函数
    - SDS vs C字符串
      - 获取字符串长度的复杂度为 O(N) 。 获取字符串长度的复杂度为 O(1) 。
      - API 是不安全的，可能会造成缓冲区溢出。 API 是安全的，不会造成缓冲区溢出。
      - 修改字符串长度 N 次必然需要执行 N 次内存重分配。 修改字符串长度 N 次最多需要执行 N 次内存重分配。
      - 只能保存文本数据。 可以保存文本或者二进制数据。
      - 可以使用所有 <string.h> 库中的函数。 可以使用一部分 <string.h> 库中的函数。
    - SDS API
      - sdsnew 创建一个包含给定 C 字符串的 SDS 。 O(N)
      - sdsempty 创建一个不包含任何内容的空 SDS 。 O(1)
      - sdsfree 释放给定的 SDS 。 O(1)
      - sdslen 返回 SDS 的已使用空间字节数。 这个值可以通过读取 SDS 的 len 属性来直接获得， 复杂度为 O(1) 。
      - sdsavail 返回 SDS 的未使用空间字节数。 这个值可以通过读取 SDS 的 free 属性来直接获得， 复杂度为 O(1) 。
      - sdsdup 创建一个给定 SDS 的副本（copy）。 O(N) ， N 为给定 SDS 的长度。
      - sdsclear 清空 SDS 保存的字符串内容。 因为惰性空间释放策略，复杂度为 O(1) 。
      - sdscat 将给定 C 字符串拼接到 SDS 字符串的末尾。 O(N) ， N 为被拼接 C 字符串的长度。
      - sdscatsds 将给定 SDS 字符串拼接到另一个 SDS 字符串的末尾。 O(N) ， N 为被拼接 SDS 字符串的长度。
      - sdscpy 将给定的 C 字符串复制到 SDS 里面， 覆盖 SDS 原有的字符串。 O(N) ， N 为被复制 C 字符串的长度。

        - sdsgrowzero 用空字符将 SDS 扩展至给定长度。 O(N) ， N 为扩展新增的字节数。
        - sdsrange 保留 SDS 给定区间内的数据， 不在区间内的数据会被覆盖或清除。 O(N) ， N 为被保留数据的字节数。
        - sdstrim 接受一个 SDS 和一个 C 字符串作为参数， 从 SDS 左右两端分别移除所有在 C 字符串中出现过的字符。 O(M*N) ， M 为 SDS 的长度， N 为给定 C 字符串的长度。
        - sdscmp 对比两个 SDS 字符串是否相同。 O(N) ， N 为两个 SDS 中较短的那个 SDS 的长度。
- 链表
    - Redis 的链表实现的特性可以总结如下：
        - 双端： 链表节点带有 prev 和 next 指针， 获取某个节点的前置节点和后置节点的复杂度都是 O(1) 。
        - 无环： 表头节点的 prev 指针和表尾节点的 next 指针都指向 NULL ， 对链表的访问以 NULL 为终点。
        - 带表头指针和表尾指针： 通过 list 结构的 head 指针和 tail 指针， 程序获取链表的表头节点和表尾节点的复杂度为 O(1) 。
        - 带链表长度计数器： 程序使用 list 结构的 len 属性来对 list 持有的链表节点进行计数， 程序获取链表中节点数量的复杂度为 O(1)。
        - 多态： 链表节点使用 void* 指针来保存节点值， 并且可以通过 list 结构的 dup 、 free 、 match 三个属性为节点值设置类型特定函数， 所以链表可以用于保存各种不同类型的值。
    - 链表api
        - listSetDupMethod 将给定的函数设置为链表的节点值复制函数。 O(1) 。
        - listGetDupMethod 返回链表当前正在使用的节点值复制函数。 复制函数可以通过链表的 dup 属性直接获得，O(1)
        - listSetFreeMethod 将给定的函数设置为链表的节点值释放函数。 O(1) 。
        - listGetFree 返回链表当前正在使用的节点值释放函数。 释放函数可以通过链表的 free 属性直接获得，O(1)
        - listSetMatchMethod 将给定的函数设置为链表的节点值对比函数。 O(1)
        - listGetMatchMethod 返回链表当前正在使用的节点值对比函数。 对比函数可以通过链表的 match 属性直接获得，O(1)
        - listLength 返回链表的长度（包含了多少个节点）。 链表长度可以通过链表的 len 属性直接获得，O(1) 。
        - listFirst 返回链表的表头节点。 表头节点可以通过链表的 head 属性直接获得，O(1) 。
        - listLast 返回链表的表尾节点。 表尾节点可以通过链表的 tail 属性直接获得，O(1) 。
        - listPrevNode 返回给定节点的前置节点。 前置节点可以通过节点的 prev 属性直接获得，O(1) 。
        - listNextNode 返回给定节点的后置节点。 后置节点可以通过节点的 next 属性直接获得，O(1) 。

        - listNodeValue 返回给定节点目前正在保存的值。 节点值可以通过节点的 value 属性直接获得，O(1) 。
        - listCreate 创建一个不包含任何节点的新链表。 O(1)
        - listAddNodeHead 将一个包含给定值的新节点添加到给定链表的表头。 O(1)
        - listAddNodeTail 将一个包含给定值的新节点添加到给定链表的表尾。 O(1)
        - listInsertNode 将一个包含给定值的新节点添加到给定节点的之前或者之后。 O(1)
        - listSearchKey 查找并返回链表中包含给定值的节点。 O(N) ， N 为链表长度。
        - listIndex 返回链表在给定索引上的节点。 O(N) ， N 为链表长度。
        - listDelNode 从链表中删除给定节点。 O(1) 。
        - listRotate 将链表的表尾节点弹出，然后将被弹出的节点插入到链表的表头， 成为新的表头节点。 O(1)
        - listDup 复制一个给定链表的副本。 O(N) ， N 为链表长度。
        - listRelease 释放给定链表，以及链表中的所有节点。 O(N) ， N 为链表长度。
- 字典
    - 计算hash值使用 MurmurHash2算法
    - redis通链地址法解决hash冲突 如果有多个键分配到同一个索引上 通过next指针形成单链表
    - 渐进式rehash: 逐批的把旧的hash表上的值,移动到新表上
    - 字典API
        - dictCreate 创建一个新的字典。 O(1)
        - dictAdd 将给定的键值对添加到字典里面。 O(1)
        - dictReplace 将给定的键值对添加到字典里面， 如果键已经存在于字典，那么用新值取代原有的值。 O(1)
        - dictFetchValue 返回给定键的值。 O(1)
        - dictGetRandomKey 从字典中随机返回一个键值对。 O(1)
        - dictDelete 从字典中删除给定键所对应的键值对。 O(1)
        - dictRelease 释放给定字典，以及字典中包含的所有键值对。 O(N) ， N 为字典包含的键值对数量。
- 跳跃表
    - 跳跃表时有序集合的主要实现之一
- 整数集合
    - 整数集合是 集合的底层实现之一
    - 升级操作为整数集合带来了操作上的灵活性， 并且尽可能地节约了内存。
    - 整数集合支持升级不支持降级
    - 整数集合API
        - intsetNew 创建一个新的整数集合。 O(1)
        - intsetAdd 将给定元素添加到整数集合里面。 O(N)
        - intsetRemove 从整数集合中移除给定元素。 O(N)
        - intsetFind 检查给定值是否存在于集合。 因为底层数组有序，查找可以通过二分查找法来进行， 所以复杂度为 O(\log N) 。

        - intsetRandom 从整数集合中随机返回一个元素。 O(1)
        - intsetGet 取出底层数组在给定索引上的元素。 O(1)
        - intsetLen 返回整数集合包含的元素个数。 O(1)
        - intsetBlobLen 返回整数集合占用的内存字节数。 O(1)
- 压缩列表
    - 压缩链表API
        - ziplistNew 创建一个新的压缩列表。 O(1)
        - ziplistPush 创建一个包含给定值的新节点， 并将这个新节点添加到压缩列表的表头或者表尾。 平均 O(N) ，最坏 O(N^2) 。
        - ziplistInsert 将包含给定值的新节点插入到给定节点之后。 平均 O(N) ，最坏 O(N^2) 。
        - ziplistIndex 返回压缩列表给定索引上的节点。 O(N)
        - ziplistFind 在压缩列表中查找并返回包含了给定值的节点。 因为节点的值可能是一个字节数组， 所以检查节点值和给定值是否相同的复杂度为 O(N) ， 而查找整个列表的复杂度则为 O(N^2) 。
        - ziplistNext 返回给定节点的下一个节点。 O(1)
        - ziplistPrev 返回给定节点的前一个节点。 O(1)
        - ziplistGet 获取给定节点所保存的值。 O(1)
        - ziplistDelete 从压缩列表中删除给定的节点。 平均 O(N) ，最坏 O(N^2) 。
        - ziplistDeleteRange 删除压缩列表在给定索引上的连续多个节点。 平均 O(N) ，最坏 O(N^2) 。
        - ziplistBlobLen 返回压缩列表目前占用的内存字节数。 O(1)
        - ziplistLen 返回压缩列表目前包含的节点数量。 节点数量小于 65535 时 O(1) ，大于 65535 时 O(N)
- 对象
    - redis使用对象表示键和值, 每次新增的时候都会生成两个对象 键对象 和 值对象
    - 键对象中的type记录着建对象的类型(string字符串,list列表,hash,set集合,zset有序集合) , 通过 TYPE key 来查询键的对象类型
    - 键对象中的encoding 记录着存储的数据结构
        - REDIS_ENCODING_INT long 类型的整数
        - REDIS_ENCODING_EMBSTR embstr 编码的简单动态字符串
        - REDIS_ENCODING_RAW 简单动态字符串
        - REDIS_ENCODING_HT 字典
        - REDIS_ENCODING_LINKEDLIS 双端链表
        - REDIS_ENCODING_ZIPLIST 压缩列表
        - REDIS_ENCODING_INTSET 整数集合
        - REDIS_ENCODING_SKIPLIST 跳跃表和字典
    - 每种类型所对应的实现结构
        - REDIS_STRING REDIS_ENCODING_INT 使用整数值实现的字符串对象。
        - REDIS_STRING REDIS_ENCODING_EMBSTR 使用 embstr 编码的简单动态字符串实现的字符串对象。

    - REDIS_STRING REDIS_ENCODING_RAW 使用简单动态字符串实现的字符串对象。
    - REDIS_LIST REDIS_ENCODING_ZIPLIST 使用压缩列表实现的列表对象。
    - REDIS_LIST REDIS_ENCODING_LINKEDLIST 使用双端链表实现的列表对象。
    - REDIS_HASH REDIS_ENCODING_ZIPLIST 使用压缩列表实现的哈希对象。
    - REDIS_HASH REDIS_ENCODING_HT 使用字典实现的哈希对象。
    - REDIS_SET REDIS_ENCODING_INTSET 使用整数集合实现的集合对象。
    - REDIS_SET REDIS_ENCODING_HT 使用字典实现的集合对象。
    - REDIS_ZSET REDIS_ENCODING_ZIPLIST 使用压缩列表实现的有序集合对象。
    - REDIS_ZSET REDIS_ENCODING_SKIPLIST 使用跳跃表和字典实现的有序集合对象。
- 使用 OBJECT ENCODING 查询存储对象使用的数据结构
    - 整数 REDIS_ENCODING_INT "int"
    - embstr 编码的简单动态字符串（SDS） REDIS_ENCODING_EMBSTR "embstr"
    - 简单动态字符串 REDIS_ENCODING_RAW "raw"
    - 字典 REDIS_ENCODING_HT "hashtable"
    - 双端链表 REDIS_ENCODING_LINKEDLIST "linkedlist"
    - 压缩列表 REDIS_ENCODING_ZIPLIST "ziplist"
    - 整数集合 REDIS_ENCODING_INTSET "intset"
    - 跳跃表和字典 REDIS_ENCODING_SKIPLIST "skiplist"
- string命令
    - embstr对象不支持修改当发生修改时会变成raw对象
    - 测试阿里云使用的是raw
    - set 添加 get 获取 del删除
    - APPEND 将对象转换成 raw 编码 调用 sdscatlen 函数 将给定字符串追加到现有字符串的末尾
    - INCRBYFLOAT 取出整数值/字符串值并将其转换成 longdouble 类型的浮点数，对这个浮点数进行加法计算， 然后将得出的浮点数结果保存起来
    - INCRBY 对整数进行加运算
    - DECRBY 对整数进行减运算
    - STRLEN 获取字符串长度
    - SETRANGE 将字符串特定索引上的值设置为给定的字符。
    - GETRANGE 直接取出并返回字符串指定索引上的字符。
- list 命令
    - LPUSH 将新元素推入到链表的表头
    - RPUSH 将新元素推入到链表的表尾
    - LPOP 删除表头节点
    - RPOP 删除表尾节点
    - LINDEX 定位指定节点并返回
    - LLEN 返回表长度
    - LINSERT 插入到指定位置
    - LREM 删除指定元素

        - LTRIM 删除列表中所有不在指定索引范围内的节点
        - LSET 赋值操作更新节点的值
    - hash 命令
        - HSET 添加 HGET获取 HDEL删除
        - HEXISTS 判断键是否存在
        - HLEN 获取数量
        - HGETALL 返回所有的键值对
    - set 集合
        - SADD 添加元素
        - SCARD 返回包含的元素数量
        - SISMEMBER 查找元素是否存在
        - SMEMBERS 返回集合中所有的键
        - SRANDMEMBER 从集合中随机返回一个
        - SPOP 删除一个元素并返回
        - SREM 删除所有给定元素
    - zset 有序集合
        - ZADD 添加
        - ZREM 删除
        - ZSCORE 获取分值
        - ZREVRANK 从尾向头遍历获取排名
        - ZRANK 从头向尾遍历获取排名
        - ZRANGE 从表头向表尾遍历跳跃表， 返回给定索引范围内的所有元素。
        - ZREVRANGE 从表尾向表头遍历跳跃表， 返回给定索引范围内的所有元素。
        - ZCOUNT 遍历跳跃表， 统计分值在给定范围内的节点的数量
        - ZCARD 直接返回集合元素的数量。
    - 对象模型属性
        - type 类型决定 是string,list,hsah,set ,zset
        - encoding 决定着使用什么存储结构
        - ptr 指针
        - refcount 引用计数器
        - lru 最后一次访问时间
- 持久化
    - RDB
    - AOF
- 事件
    - Reactor 模式 IO多路复用
- 主从复制
    - 同步
        - 从服务器向主服务器发送 SYNC 命令。
        - 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现在开始执行的所有写命令。

    - 当主服务器的 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， 将自己的数据库状态更新至主服务器执行 BGSAVE 命令时的数据库状态。
    - 主服务器将记录在缓冲区里面的所有写命令发送给从服务器， 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数据库当前所处的状态
- 传播
    - 当完成 同步 后就把对主服务器的操作传播给从服务器
- redis2.8以上版本 PSYNC 可以只传播数据库断开重连接所丢失的数据,PSYNC一般应用于断线重连

以上内容整理于 幕布