Redis 数据结构

SDS简单动态字符串

```
struct sdshdr {

    int len;//记录已使用的长度

    int free;//记录未使用的长度

    char buf[]//保存字符串

}
```

优点:

1. 可以直接获取字符串的长度,O(1)复杂度
2. 可以杜绝缓冲区溢出（可能会把别的字符串覆盖掉）,先扩容,在修改
3. 扩容会申请额外的内存空间,如果SDS的长度小于1M,那么扩容为以前的2倍,如果大于1M,额外添加1M的额外内存
4. 如果Free空间够,则先使用Free空间,否则就申请额外空间
5. 惰性释放,如果SDS缩短,那么不会立刻释放内存,而是增长Free空间
6. SDS通过Len判断字符串是否结束

List

```
typedef struct listNode {

    listNode *prev;

    listNode *next;

    void *value;

}listNode;

typedef struct list{

    listNode *head;

    listNode *tail;

    unsigned  long len;

}list;
```

1. List 是一个双向链表
2. 无环, head.prev tail.next 都为null
3. 存储长度
4. 列表key,发布与订阅,慢查询, 监视器

字典dictht

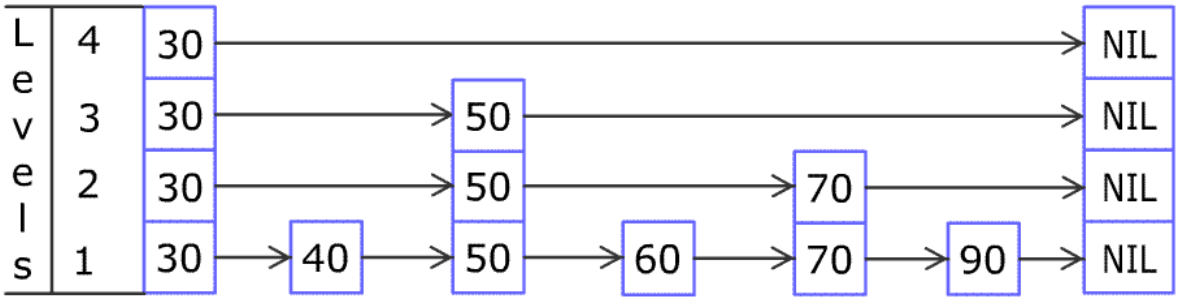
```
typedef struct dictht {

    dictEntry ** table; //hash数组

    unsigned long size;//hash表大小

    unsigned long sizemask; //用于计算索引值

    unsigned long used ;//已经使用的数量

}dictht
```

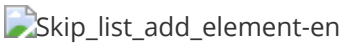
```
typedef struct dictEntry {

    void *key;//键

    union{//值

        void *val;

        uint64_t u64;

        int64_t s64;

    } v;

    struct dictEntry * next;

}dictEntry
```

1. skiplist 跳跃表 最好OlogN最坏 On的查询速度
2. intset整型集合
3. 压缩列表，当list或hash元素为较短的string或int时绘彩通这个数据结构
4. 拉链法解决hash冲突
5. 渐进式rehash
6. 用于数据库和哈希key

跳跃表

1. 用于Zset有序集合和在集群结点中做内部结构
2. 平均O(logN),最差O(N)的查询速度

3. 



整数集合

```
typedef struct intset {

    uint32_t encoding;//编码格式

    uint32_t length;//长度

    int8_t contents[];//数组

}intset;
```

1. 如果添加的int大于当前的数组的长度,会进行数组升级,也就是改变编码格式.
2. 如果发生升级,那么最大的数就在数组尾.
3. 通过升级更省内存,也更灵活

压缩列表ZipList

1. 如果列表Key只包含少量的列表项,并且每个元素都是小整数或者短的字符串,就会采用压缩表,同样的,当Key和Value 也符合上述条件时,也会用压缩列表实现Hash
2. 数据结构
    1. zlbytes 压缩列表的字节长度
    2. zltail 尾结点的头 的偏移量
    3. zllen 记录数组的长度
    4. entry
3. 结点的数据结构
    1. previous_entry_length 前一数组的字节长度
    2. encoding 前几位表示 是数组还是整数后几位代表长度
    3. content 可以是一个字节数组或者整数
    4. 可能会出现需要连锁更新的情况

Redis中的对象

```
typedef struct redisObject {

    usigned type; //数据类型

    unsigned encoding; //具体的编码格式

    void *ptr; //指向 存储的内存地址

};
```

不同类型和编码的对象

类型	编码	对象
redis_string	redis_encoding_int	整数字符串
redis_string	redis_encoding_embstr	embstr编码的简单动态字符串
redis_string	redis_encoding_raw	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	使用双向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	使用哈希表实现的哈希对象
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	使用跳跃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

1. StringObject
    1. 整数类型的用int编码方式
    2. 字符传类型,如果长度小于39字节,使用embstr方式编码,否者用raw
    3. embstr只能读,所以任何对embstr的增删都会变成raw编码
2. ListObject

   每一个元素的长度小于64字节,并且元素个数小于512,使用ziplist,否则使用linkedlist
3. 通过引用计数器来判断是否需要回收
4. ZsetObject是综合了跳跃表和字典实现的
5. Redis是有对象共享机制的,但是只共享包含整数的字符串对象.

   系统会缓存0到9999的字符串对象
6. 对象有一个lur属性,记录了对象最后一次被命令程序访问的时间,通过这个属性来实现淘汰key
7. Redis通过一个过期字典,保存数据库中所有键的过期时间
8. appendfsync 决定了aof的 效率和安全性

9. aof 重写是从数据库中读取键现在的值,然后用一条命令去记录键值对
10. 服务器进程需要
    1. 执行客户端发来的命令
    2. 将执行后的写命令追加到AOF缓冲区
    3. 将执行后的写命令追加到AOF重写缓冲区
11. redis是IO多路复用, 请求的事件堵塞在一个队列中,单线程执行
12. redis serverCron函数,每隔100毫秒执行一次, 主要的任务就是更新服务器状态信息,处理服务器接受的Sigterm信号,管理客户端资源和数据库状态,并进行持久化操作等等
13. Sync是非常消耗资源的, 使用Psync来避免断线重复制
14. Redis 每个从服务器都有复制偏移量,通过复制偏移量来判断是否需要把复制积压缓冲区的写命令传播给从服务器, 如果复制偏移量大于复制缓存区,则进行主从复制
15. 从服务器 每隔一秒 会向主服务器发送命令 replconf ack 当前从服务器的偏移量, 这样 主服务器如果发现 偏移量不一致,会重新发送丢失的命令
16. redis-server 配置文件 --sentinel 开启哨兵模式

哨兵的选举

1. 每次进行哨兵选举,所有哨兵的纪元都会+1;
2. 局部领头一旦确立,他的纪元就不能在变
3. 一个哨兵A向另一个哨兵B发送命令,要求B把A作为B的局部领头
4. 一旦一个哨兵的局部领头确定,那么它的局部领头就不能变了
5. 当一个哨兵是局部领头的数量超过一半,那么他就被选为领头哨兵
6. 如果未有超过半数的,重新进行选举
7. 哨兵会进行三个操作
    1. 选举一个从服务器作为主服务器
    2. 将其他从服务器同步主服务器
    3. 将下线的主服务器上线后,设为从服务器

Cluster集群

1. 在配置文件中配置cluster-enabled=yes来开启
2. 最好使用docker来配置和使用集群,配置很麻烦
3. 每一个Cluster都会存储着包括自己在内的其他节点的信息
4. 数据库被分为16384个槽, 每个槽都有结点处理,集群才处于上线状态
5. 接收命令的结点会判断这个命令的槽位是不是自己能处理,如果不是,发送Moved错误,把这条命令发送给对的结点处理
6. redis通过 CRC16(key)&16383来计算槽位, CRC(16)是key的校验和;
7. 集群结点只能使用0号数据库

发布订阅模式

1. redisServer 里面有一个 pubsub_channels字典表