# 概述

内存数据库

数据类型

String list set sortedSet hashes

一个字符串类型的值存储最大容量是512M

Redis集群方案

1.twemproxy，大概概念是，它类似于一个代理方式，使用方法和普通redis无任何区别，设置好它下属的多个redis实例后，使用时在本需要连接redis的地方改为连接twemproxy，它会以一个代理的身份接收请求并使用一致性hash算法，将请求转接到具体redis，将结果再返回twemproxy。使用方式简便(相对redis只需修改连接端口)，对旧项目扩展的首选。 问题：twemproxy自身单端口实例的压力，使用一致性hash后，对redis节点数量改变时候的计算值的改变，数据无法自动移动到新的节点。

2.codis，目前用的最多的集群方案，基本和twemproxy一致的效果，但它支持在 节点数量改变情况下，旧节点数据可恢复到新hash节点。

3.redis cluster3.0自带的集群，特点在于他的分布式算法**不是一致性hash**，而是hash槽的概念，以及自身支持节点设置从节点。具体看官方文档介绍。

Redis集群之间异步复制

Redis并不能保证数据的强一致性

集群最大节点数 16384

Redis过期:expire

redis回收使用的算法:LRU

Redis大数据量插入:pip mode

# String

SDS,内部维护一个字节数组,并且预分配了一定的空间. 当字符串长度小于`1M,每次扩容加倍现有空间,当字符串长度大于1M,每次扩容会扩展1M空间

常用操作

Set key value,get ,del,setex,setnx,incr,exists,mget,mset

struct SDS{

T capacity; //数组容量

T len; //实际长度

byte flages; //标志位,低三位表示类型

byte[] content; //数组内容

}

# List

当数据量较少的时候他的底层存储结构为一块连续内存,称为ziplist,当数据量较多的时候会变成quicklist

struct ziplist<T>{

int32 zlbytes; //压缩列表占用字节数

int32 zltail\_offset; //最后一个元素距离起始位置的偏移量,用于快速定位到最后一个节点

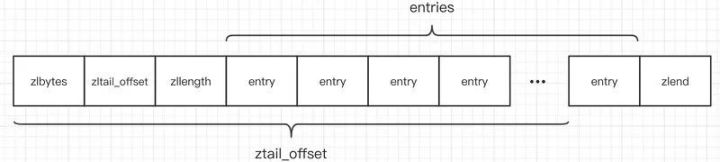
int16 zllength; //元素个数

T[] entries; //元素内容

int8 zlend; //结束位 0xFF

}

Zltail\_offset可以实现双向遍历



应用:消息队列,点赞,评论

Lpush,lpop,lrange,lrem,llen

# Hash

以链表方式解决hash碰撞,value只能是字符串.

hset [key] [field] [value] 新建字段信息

hget [key] [field] 获取字段信息

hdel [key] [field] 删除字段

hlen [key] 保存的字段个数

hgetall [key] 获取指定key 字典里的所有字段和值 （字段信息过多,会导致慢查询 慎用：亲身经历 曾经用过这个这个指令导致线上服务故障）

hmset [key] [field1] [value1] [field2] [value2] ...... 批量创建

hincr [key] [field] 对字段值自增

hincrby [key] [field] [number] 对字段值增加number

rehash是怎么搞得

# set

内部的键值对是无序的,唯一的

Sad,smembers,sismember,scard,sinter(交集)

# Zset

以跳跃表的数据结构实现.

zadd [key] [score] [value] 向指定key的集合中增加元素

zrange [key] [start\_index] [end\_index] 获取下标范围内的元素列表，按score 排序输出

zrevrange [key] [start\_index] [end\_index] 获取范围内的元素列表 ，按score排序 逆序输出

zcard [key] 获取集合列表的元素个数

zrank [key] [value] 获取元素再集合中的排名

zrangebyscore [key] [score1] [score2] 输出score范围内的元素列表

zrem [key] [value] 删除元素

zscore [key] [value] 获取元素的score

跳跃表和avl之类的有什么不同