# 介绍：

redis是单线程所以保证了原子性。

Redis的网络请求，Redis会有一个EventLoop，里面有两个数组events,fired。events存放被注册的事件，fired用于存放EventLoop从多路复用器（epoll）中读取到的，将要执行的事件。异步和非阻塞就反映在这里，注册到多路复用器（epoll）后去做其他事，之后通过主动轮询多路复用器，来逐个取出将要执行的事件，放入fired，逐个执行，这个过程是单线程的，因此不会出现并发问题。

--------------------

shutdown

cd C:\Users\Administrator\Desktop\redis64-3.0.501

redis-server.exe redis.windows.conf

删除通用键：./redis-cli keys "ZOMBIE\_ScoreArena\*" | xargs ./redis-cli del

--------------------

# 通用键的操作

keys \*

keys my\*

keys my?

exists key

rename oldname newname

type key

object encoding key

setnx key value------------------键不存则设置值

expire key s--------------------设置有效时间秒

pexpire key ms--------------------设置有效时间毫秒

expireat key s--------------------设置有效时间至时间戳秒

pexpireat key ms--------------------设置有效时间至时间戳毫秒

persist key-------------删除有效时间

ttl key(pttl key) 所剩余过期时间（成功1，没有设置-1，不存在key-2）

SET key value [EX seconds] [PX milliseconds] [NX|XX]

**锁的实现方法**

# 数据库操作

默认的是0数据库

select index选择第几个数据库

move name index移动到第几个数据库

multi--------------------开启事务

exec--------------------提交

discard--------------------回滚

# 各个数据结构及其用途

**每个对象的结构：**

typedef struct redisObject {

unsigned type:4; /\* 对象类型 \*/

unsigned encoding:4; /\* 数据结构 \*/

unsigned lru:LRU\_BITS; /\* 上一次当前对象被访问的时间 \*/

int refcount; /\* 对象的引用计数，用于释放内存 \*/

void \*ptr; /\* 当前数据结构的首地址 \*/

} robj;

**type：对象类型**

#define OBJ\_STRING 0

#define OBJ\_LIST 1

#define OBJ\_SET 2

#define OBJ\_ZSET 3

#define OBJ\_HASH 4

**encoding：数据结构**

#define OBJ\_ENCODING\_RAW 0 /\* string-SDS 字符串 \*/

#define OBJ\_ENCODING\_INT 1 /\* string-整数 \*/

#define OBJ\_ENCODING\_HT 2 /\* 字典结构 \*/

*#define OBJ\_ENCODING\_ZIPMAP 3 /\* 压缩map，废弃 \*/*

*#define OBJ\_ENCODING\_LINKEDLIST 4 /\* LinkedList 双端链表，废弃 \*/*

#define OBJ\_ENCODING\_ZIPLIST 5 /\* hash-压缩列表 \*/

#define OBJ\_ENCODING\_INTSET 6 /\* 整数集合 \*/

#define OBJ\_ENCODING\_SKIPLIST 7 /\* 跳跃表 \*/

#define OBJ\_ENCODING\_EMBSTR 8 /\* string-短字符串 \*/

#define OBJ\_ENCODING\_QUICKLIST 9 /\* 压缩链表和双向链表组成的快速列表 \*/

## OBJ\_ENCODING\_RAW——字符串

## OBJ\_ENCODING\_INT——整数

## OBJ\_ENCODING\_HT——字典结构

## OBJ\_ENCODING\_ZIPMAP——压缩map，废弃

## OBJ\_ENCODING\_LINKEDLIST——LinkedList 双端链表，废弃

## OBJ\_ENCODING\_ZIPLIST——hash-压缩列表

## OBJ\_ENCODING\_INTSET——整数集合

## OBJ\_ENCODING\_SKIPLIST——跳跃表

## OBJ\_ENCODING\_EMBSTR——短字符串

## OBJ\_ENCODING\_QUICKLIST——压缩链表和双向链表组成的快速列表

## string：时效性的数据（token）和简单的key value数据。

字符串（raw）：普通的字符串，append，大于44的字符串

整数（int）：long 类型的整数值

短字符串（embstr ）：短字符串，set小于等于44的字符串

数据结构（simple dynamic string）：

len buf中已经占有的长度(表示此字符串的实际长度)

free buf中未使用的缓冲区长度

buf[] 实际保存字符串数据的地方

1、空间预分配：扩展比实际的多，减少连续增长操作的内存分配次数（小于1M翻倍，否则1M）。

2、惰性空间释放：缩短操作时，程序不立即回收缩短后多余的字节，而是使用 free 属性将这些字节的数量记录下来。

## hash：更直观和容易维护信息，如用户信息。

一种是ziplist，同时满足以下两个条件时才会使用这种结构：

当键的个数小于hash-max-ziplist-entries（默认512）

当所有值都小于hash-max-ziplist-value（默认64）

另一种就是hashtable。这种结构的时间复杂度为O(1)，但是会消耗比较多的内存空间。

## list：队列

ziplist，字面意是压缩列表，另一个是quicklist，字面意是快速列表

## set：去重

## zset：排行榜

# string

set key value

get key

del key

incr key--------------------存在值加1，否则设置0

decr key--------------------存在值减1，否则设置0

incrby key increment--------------------存在值加increment，否则设置0+increment

decrby key increment--------------------存在值减increment，否则设置0-increment

append key value--------------------字符串往后加字符串

strlen key

setbit key offset value--------------------设置偏移量位的值0，1

getbit key offset--------------------得到偏移量位的值

# hash

hset key field value

hmset key field value1 age value2

hget key field

hdel key field

del key

hincrby key field value--------------------增加数值

hgetall key

hlen key

hkeys key

# list(arraylist,linkedlist)

lpush key value--------------------头部添加

rpush key value--------------------尾部添加

lrange key value1 value2--------------------索引的范围

lpop key--------------------头部元素移出

rpop key--------------------尾部元素移出

llen key--------------------集合大小

lpushx key value--------------------列表存在再插入

del key

lrem key count value--------------------删除num个value ,num>0从前删除，<0从后删除，=0删除所有

lset key index value--------------------改变指定位置元素

linsert key after value1 value2 在2后面插入1--------------------向指定元素后插入数据

rpoplpush key1 key2--------------------1的尾部取出放入2的头部，适用于消息队列

# set

sadd key member

srem key member

smembers key

sismember name member--------------------0不存在，1存在

sdiff key key1--------------------求差集

sinter key key1--------------------求交集

sunion key key1--------------------求并集

scard key--------------------数量

sdiffstore key key1 key2--------------------求差集赋值给key

sinterstore key key1 key2 求交集赋值给key

sunionstore key key1 key2 求并集赋值给key

# zset

zadd key score1 member1 score2 member2

zscore key member

zcard key--------------------数量

zrange key rank1 rank2 (withscores)--------------------index范围输出

zrangebyscore key score1 score2 (limit offset count)--------------------分数范围

zincrby key increment member//此处先写分数再写键值

zrem key member1 member2--------------------删除

zremrangebyrank key rank1 rank2--------------------排名范围

zremrangebyscore key score1 score2--------------------分数排名

# RDB（默认开启）

## redis.conf——SNAPSHOTTING标签下

save ""#关闭RDB，默认语句被注释

save <seconds> <changes>#900 1,300 10,60 10000,指定时间操作数进行持久化

rdbcompression yes#默认开始LZF算法数据压缩（会占用cpu时间）

## 触发条件

在指定的时间间隔内，执行指定次数的写操作  
执行save（阻塞， 只管保存快照，其他的等待） 或者是bgsave （异步）命令  
执行flushall 命令，清空数据库所有数据，意义不大。  
执行shutdown 命令，保证服务器正常关闭且不丢失任何数据，意义...也不大。

## 优点

适合大规模的数据恢复。

如果业务对数据完整性和一致性要求不高，RDB是很好的选择。

## 缺点

数据的完整性和一致性不高，因为RDB可能在最后一次备份时宕机了。

备份时占用内存，备份时会独立创建一个子进程，将数据写入到一个临时文件（此时数据是原来的两倍），最后再将临时文件替换之前的备份文件。

所以Redis 的持久化和数据的恢复要选择在夜深人静的时候执行是比较合理的。

## 如何恢复数据

将dump.rdb 文件拷贝到redis的安装目录的bin目录下，重启redis服务即可。

在实际开发中，一般会考虑到物理机硬盘损坏情况，选择备份dump.rdb

## 注意项

flushall和shutdown也会触发RDB快照

# AOF（默认关闭）

## redis.conf——APPEND ONLY MODE标签下

appendonly yes#开启AOF

appendfilename "appendonly.aof#数据库名字

# appendfsync always#变化立刻写入到磁盘，虽然慢但是安全。

appendfsync everysec#出厂默认推荐，每秒异步记录一次(默认值)

# appendfsync no#不同步

auto-aof-rewrite-percentage 100#百分比

auto-aof-rewrite-min-size 64mb#重写文件的最小size

当AOF文件大小是上次rewrite后大小的一倍且文件大于64M时触发。一般都设置为3G。

## 触发条件

根据appendfsync设置方式。

## 优点

数据的完整性和一致性更高

## 缺点

因为AOF记录的内容多，文件会越来越大，数据恢复也会越来越慢。

## 如何恢复数据

redis-check-aof --fix appendonly.aof//恢复aof格式异常

## 重写机制

Redis 会fork出一条新进程，读取内存中的数据，并重新写到一个临时文件中。并没有读取旧文件。最后替换旧的aof文件。

## 注意项

每个操作写入日志

优点：安全

缺点：效率低

3、同时使用RDB和AOF

4、无持久化内存缓存机制

# 内存淘汰策略

maxmemory <bytes>#设置最大的内存大小

maxmemory-policy noeviction#内存淘汰策略设置

noeviction：所有申请内存的命令会报错。

allkeys-lru：在主键空间中，优先移除最近未使用的key。（推荐）

allkeys-random：在主键空间中，随机移除某个key。

volatile-lru：在设置了过期时间的键空间中，优先移除最近未使用的key。

volatile-random：在设置了过期时间的键空间中，随机移除某个key。

volatile-ttl：在设置了过期时间的键空间中，具有更早过期时间的key优先移除。

# 其他

Redis读的速度是11W次/s,写的速度是8.1次/s .

NoSql=not only sql:非关系型数据库

SNS网站：社交网站服务

非关系型数据库：

mongoDB(文档数据库，查询性能低，没有统一查询语句) redis（键值对）

高并发读写，海量数据的存储和访问，高可扩展性和高可用性，易扩展，灵活的数据模型

缓存，任务队列，应用排行榜，网站访问统计，数据过期处理，分布式集群架构中的session分离

key:不能太长也不能太短，要有统一的命名的规范，1024字节

value：512M

## scan

scan n match \* count 100

SCAN 命令返回，数据库键。

SSCAN 命令返回，集合成员。

HSCAN 命令返回，键值对成。

ZSCAN 命令返回，有序集合元素。

# 集群

## 主从模式

一个主数据库和众多从数据库。读写分离，异步的执行更新。sync命令请求，通过bgsave记录写命令生成rdb文件，分全量和增量更新。

缺点：不具备自动容错和恢复功能，宕机都会导致部分读写失败；从机不能同时启动；

## 哨兵模式

为了解决主从模式的高不可用性，Sentinel系统来监视主从模式，自身先选举，再来执行任务，

## 集群模式

分布式数据库方案，从数据库和主数据库，不同的key16384个槽。

# RedLock

解决主机的锁宕机后，从机还能获取锁的问题。

# 多路复用技术

多路：多个socket连接

复用：复用同一个线程