

Redis

Redis是一个开源的、支持分布式、键值存储的数据库，提供类似于Memcached的数据结构，但功能更加强大。

特点：

- 简单
- 易于部署和使用
- 支持持久化
- 支持丰富的数据结构（字符串、哈希、列表、集合、有序集合）
- 支持事务

Redis配置

```
## Redis配置文件示例
bind 0.0.0.0
## 启用守护进程
daemonize yes
## Redis版本
requirepass 123456
## 端口
port 6379
## 目录
dir .
## 数据库数量
databases 1
## Redis内存限制
maxmemory 512mb
## 日志文件
logfile "redis.log"
```

命令行工具

```
redis-cli [options] [commands]
```

命令行选项

- -h Redis服务器IP地址
- -p Redis端口号
- -a Redis密码

命令行命令

- ping 测试连接 pong
- help 帮助命令 commands 命令列表

命令

- KEYS 检查所有key
- DEL 删除所有key所有key key
- EXISTS 检查key是否存在

- EXPIRE 旣key旳時間範圍內key旳存在性
 - TTL 旣key旳存活時間
 - 旳-1時間範圍
 - 旳-2時間key旳存活

1

1

redis> SET key-value key String "value"

- String
 - Hash
 - List
 - Set
 - SortedSet
 - GEO
 - BitMap
 - HyperLog

Redis key

String

512MB

3

- string□□□□□
 - int□□□□
 - float□□□□□□

1

- SET ค่าของ key เป็น String
 - GET ค่าของ key คือ String ของ value
 - MSET ค่าของ key เป็น String
 - MGET ค่าของ key คือ String ของ value
 - INCR ค่าของ key คือ 1
 - INCRBY ค่าของ key คือ จำนวน INCRBY num 2
 - INCRBYFLOAT ค่าของ key คือ float
 - SETNX ค่าของ key คือ 1
 - SETEX ค่าของ key คือ จำนวน second

Hash

value java HashMap



redis

- HSET key field value
- HGET key field
- HMSET hash key field
- HMGET hash key field
- HGETALL hash key field value
- HKEYS hash key
- HVALS hash key value
- HINCRBY hash key increment
- HSETNX hash key field

List

java LinkedList

list

- LRANGE
- LPOP
- RPOP
- LPUSH

redis

- LPUSH key element
- LPOP key
- RPUSH key element
- RPOP key
- LRANGE key start end
- BLPOP | BRPOP | LPOP | RPOP

◦ null

Set

java HashSet value null HashMap

set

- SADD
- SREM
- SCARD
- SISMEMBER

redis

- SADD key member
- SREM key member
- SCARD key
- SISMEMBER key set
- SMEMBERS set

- SINTER key1 key2 ☐key1☐key2☐
- SDIFF key1 key2 ☐key1☐key2☐
- SUNION key1 key2 ☐key1☐key2☐

SortedSet

score score score +

...

-
-
-

...

- ZADD key score member sortedset score score score
- ZREM key member sortedset
- ZSCORE key member sortedset score
- ZRANK key member sortedset
- ZCARD key sortedset
- ZCOUNT key min max score
- ZINCRBY key increment member sortedset score increment
- ZRANGE key min max score score min max
- ZRANGEBYSCORE key min max score score min max
- ZDIFF ZINTER ZUNION

...

Jedis

Java Redis

...

1. 須要

```
<dependency>
    <groupId>redis.clients</groupId>
    <artifactId>jedis</artifactId>
    <version>3.7.0</version>
</dependency>
```

2. 須要

```
private Jedis jedis;

void setup(){
    // ...
    jedis = new Jedis("ip", 6379);
    // ...
    jedis.auth("");
```

```
// 选择键槽
jedis.select(0);
}
```

3. 释放连接

4. 关闭连接

```
void tearDown(){
    if(jedis != null){
        // 释放连接
        jedis.close();
    }
}
```

JedisPool

JedisPool是Jedis的一个线程安全的连接池，通过JedisPool可以方便地管理Jedis连接。

```
public class JedisConnectionFactory{
    private static final JedisPool jedisPool;

    static{
        // 释放连接
        JedisPoolConfig poolConfig = new JedisPoolConfig();
        // 最大连接数
        poolConfig.setMaxTotal(8);
        // 最大空闲数
        poolConfig.setMaxIdle(8);
        // 最小空闲数
        poolConfig.setMinIdle(0);
        // 最大等待时间
        poolConfig.setMaxWaitMillis(1000);
        // 构造连接池
        jedisPool = new JedisPool(poolConfig, "127.0.0.1", 6379, 1000);
    }
    public static Jedis getJedis(){
        return jedisPool.getResource();
    }
}
```

Spring Data Redis

Spring Data Redis是一个基于Spring的Redis客户端，提供了对Redis操作的简化。

- 提供了类似于Jedis的接口
- 提供了对Redis操作的简化
- 提供了对Redis操作的简化

API

- redisTemplate.opsForValue() 提供 String 操作

- `redisTemplate.opsForHash()` 用于 Hash 操作
- `redisTemplate.opsForList()` 用于 List 操作
- `redisTemplate.opsForSet()` 用于 Set 操作
- `redisTemplate.opsForZSet()` 用于 SortedSet 操作
- `redisTemplate` 其他方法
 - 通过 `TimeUnit` 定时操作
 - `.expire` 定时操作 `TimeUnit`

配置

1. 配置 SpringBoot 项目中 SpringDataRedis 相关的属性

```
<!-- 配置 -->
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-pool2</artifactId>
</dependency>
```

2. 配置

```
spring:
  redis:
    host:
    port:
    password:
    letture:
    pool:
      max-active:
      max-idle:
      min-idle:
      max-wait:
```

3. 使用 RedisTemplate

```
@Autowired
private RedisTemplate redisTemplate;
```

4. 测试

测试

RedisTemplate 对象的 `Object` 属性是 **RedisTemplate** Object，`Object` 属性的值是 `DK`，表示不知道。



测试

- Redis
- RedisTemplate

RedisTemplate 介绍

- key 为 StringRedisSerializer 实现的 String 类型
- value 为 GenericJackson2JsonRedisSerializer 实现的 Json 类型
 - 通过 RedisTemplate 的 RedisSerializer 属性实现对值的序列化和反序列化
 - 通过 RedisTemplate 的 Json 属性将 value 转换为 String 类型，再将 String 类型转换为 Java 对象

```
// 代码
@Autowired
private StringRedisTemplate stringRedisTemplate;
// Json
private String final ObjectMapper mapper = new ObjectMapper();
void testStringTemplate(){
    // 代码
    User user = new User("张三",18);
    // 代码
    String json = mapper.writeValueAsString(user);
    // 通过redis
    stringRedisTemplate.opsForValue().set("user:200",json);
    // 代码
    String val = stringRedisTemplate.opsForValue().get("user:200");
    // 代码
    User user1 = mapper.readValue(val,User.class);
}
```

- Spring 提供的 StringRedisTemplate 可以将 key 和 value 都设置为 String 类型

Hash API

- .put() 方法
- .entries() 方法

二、

通过 RedisTemplate 提供的 Hash API 可以对 Hash 结构进行操作

Hash API

- RedisTemplate 提供的 Hash API 包括
 - .put() 方法
 - .entries() 方法
 - .get() 方法
 - .getHashKey() 方法



- Redis
- Redis
- Redis CRUD

Redis

Redis



Redis

- Redis
- Redis Null TTL Redis Null



- Redis
- Redis
- Redis



Redis

Redis



Redis

- Redis key TTL
- Redis
- Redis
- Redis

Redis

Redis

- Redis
- Redis key redis null



Redis

- Redis
- Redis
- Redis

- Redis
 - Redis缓存的过期时间TTL
 - Redis缓存的过期策略
 - Redis缓存的过期操作



二

Redis缓存的过期策略

- Redis缓存的过期策略
- Redis的id过期策略
- Redis缓存的过期策略 Function 二
 - Function<ID,R> 二 Redis ID 返回值 R 二
 - .apply(二) 二
 - Redis缓存的过期策略Function和lambda
- Redis缓存的过期策略 Class<R> 二

```
public <R, ID> R queryWithPassThrough(String keyPrefix, ID id, Class<R> type,
Function<ID, R> dbFallback, Long time, TimeUnit timeUnit) {
    String key = keyPrefix + id;

    // 二redis
    String json =
stringRedisTemplate.opsForValue().get(RedisConstants.CACHE_SHOP_KEY + id);
    // 二redis缓存的过期策略Null
    if (StrUtil.isNotEmpty(json)) {
        // 二
        return JSONUtil.toBean(json, type);
    }
    // 二
    if (json != null) {
        return null;
    }
    // 二
    R r = dbFallback.apply(id);
    // 二
    if (r == null) {
        stringRedisTemplate.opsForValue().set(RedisConstants.CACHE_SHOP_KEY +
id, "", RedisConstants.CACHE_NULL_TTL, TimeUnit.MINUTES);
        return null;
    }
    // 二redis
    this.set(key, r, time, timeUnit);
    return r;
}
```

二二ID二

redis ID 生成 方 法

- 亂序
- 递增
- 递减
- 日期
- 线程

redis ID 生成 方 法 Redis 生成 方 法

- 乱序 1bit
- 递增 31bit
- 递减 32bit \$2^{32}\$ ID



乱序

线程 **synchronized** 保证线程安全的 VM

线程安全，但效率低，不适合高并发场景



递增



Redis 生成 方 法

线程安全，但效率低，不适合高并发场景

- 乱序
 - 通过锁实现线程安全
 - SETNX lock thread1
 - 通过过期实现线程安全
 - EXPIRE lock 10
 - 通过事务实现线程安全
 - SET lock thread1 EX 10 NX
- 递增
 - 通过 DEL key 实现线程安全
 - 通过双线程实现线程安全

递减

- 线程1 从大到小遍历，线程2 从大到小遍历
- 线程2 从大到小遍历，线程1 从大到小遍历

- 1 2

1



2

- **UUID** (Universally Unique Identifier)
 - **UUID** は **UUID** の **ID** である
 - **JVM** が **UUID** の **ID** を生成する
 - **UUID** の **生成方法**
 - **UUID** の **生成方法**

1

- 1
 - **JVM GC**
 - 2
 - 1 2

A decorative horizontal bar consisting of a series of small, evenly spaced rectangles.



Redis|Lua||

Redis
Lua
Redis

□□□□□Lua□□□□

Redis | Lua

- EVAL script numkeys key[key...] arg [arg ...]
 - 例如 EVAL "return redis.call('set','name','jack')" 0
 - 通过key\value参数调用script key[KEYS] arg[ARGV] KEY[KEYS]
 - 例如 EVAL "return redis.call('set',KEYS[1],ARGV[1])" 1 name Rose

IDEA Lua

Redisson

setnx

- 二〇一〇年九月三十日
 - 二〇一〇年十月三十日
 - 二〇一〇年十一月三十日

Redisson Redis Java Java

1. Maven

```
<dependency>
    <groupId>org.redisson</groupId>
    <artifactId>redisson</artifactId>
    <version>3.13.6</version>
</dependency>
```

2. Redisson

- @Configuration
- @Bean

```
@Configuration
public class RedissonConfig {
    @Bean
    public RedissonClient redissonClient() {
        // config
        Config config = new Config();

        config.useSingleServer().setAddress("redis://127.0.0.1:6379").setPassword("123456");
        // RedissonClient
        return Redisson.create(config);
    }
}
```

3. Lock

- redissonClient.getLock("Lock")
- "Lock"
- redissonClient.tryLock(1,10,TimeUnit.SECONDS)
- 1
- 10
- TimeUnit.SECONDS
- redissonClient.unLock()

Redisson

Redisson Hash



- redissonHash
- redissonHash.get("key")
- redissonHash.set("key", value)
- redissonHash.delete("key")

- Redisson WatchDog



Redisson WatchDog

Redis



Redis

- Redis PubSub

Redis

- Redis Stream
- Redis Lock

Redis MultiLock

Redisson WatchDog Redis

- Redisson WatchDog Redis
- Redisson WatchDog Redis



Redis

- @PostConstruct

Redis

Redisson WatchDog 3

- Redisson WatchDog
- Redisson WatchDog
- Redisson WatchDog



Redis

- list List
- PubSub
- Stream

Redis

- Redis

List

BRPOP BLPOP

redis

- 亂數
- 亂序

redis PubSub

redis 亂數 channel channel 亂數

- SUBSCRIBE channel[channel] 亂數
- PUBLISH channel msg 亂數
- PSUBSCRIBE pattern[pattern] 亂數 pattern 亂數



redis

- 亂數
- 亂序
- 亂序

redis stream

redis 亂數 [] 亂數

- XADD 亂數
 - key 亂數
 - [NOMKSTREAM] 亂數
 - [MAXLEN] 亂數
 - [*|ID] 亂數 id 亂數 ' 亂數 - 亂數 '
 - 亂數 redis 亂數
 - field value 亂數 Entry 亂數 key-value 亂數
 - 亂數 XADD users * name jack age 21
- XREAD 亂數
 - [COUNT] 亂數
 - [BLOCK milliseconds] 亂數
 - STREAMS key [key ...] 亂數 key 亂數
 - ID [ID ...] 亂數 id 亂數 id 亂數
 - 0 亂數
 - \$ 亂數
- XLEN 亂數

redis

redis 亂數

redis 亂數

- 亂數
- 亂數
- 亂數



redis命令

```
XGROUP CREATE key groupName ID [MKSTREAM]
```

- key 任意字符串
- groupName 任意字符串
- ID 任意ID
 - 0 任意字符串
 - \$ 任意字符串
- MKSTREAM 任意字符串

redis命令

```
XGROUP DESTORY key groupName
```

redis命令

```
XGROUP CREATECONSUMER key groupName consumername
```

redis命令

```
XGROUP DELCONSUMER key groupName consumername
```

redis命令

```
XREADGROUP GROUP group consumer [COUNT count] [BLOCK milliseconds] [NOACK] STREAMS key  
[key ...] ID [ID ...]
```

- group 任意字符串
- consumer 任意字符串
- count 任意整数
- BLOCK milliseconds 任意整数
- NOACK 任意字符串
 - pending-list 任意字符串 pending-list
- STREAMS key 任意字符串
- ID 任意字符串ID
 - > 任意字符串
 - id pending-list 任意字符串 pending-list 0 pending-list 任意字符串

redis命令

```
XACK key group ID[ID ...]
```

- key 任意字符串
- group 任意字符串
- ID 任意字符串ID

pending-list

```
XPENDING key group [[IDLE min-dile-time]] start end count [[consumer]]
```

- key 任意字符串
- group 任意字符串
- IDLE min-dile-time 任意字符串

- start end egin{ID}end{ID}
 - - egin{ID}
 - + egin{ID}
 - count egin{ID}end{ID}
 - consumer egin{ID}end{ID} pending-list

Feed

ANSWER

- Timeline
 -

Timeline □□□□□□□

- 





- **Логистика**
 - **Логистика в промышленности**
 - **Логистика в производстве**
 - **Логистика в сбыте**
 - **Логистика в сельском хозяйстве**



Feed Redis ZSet

GEO

GEO Geolocation Redis

- **GEOADD**longitude latitude member
 - **GEODIST**
 - **GEOHASH**member hash
 - **GEOPOS**member
 - **GEORADIUS**member
 - **GEOSEARCH**member
 - **GEOSEARCHSTORE**GEOSEARCHkey

GEOZSet

- score

- value



BitMap

Redis String BitMap

- 1 0
- bit 0 1

操作

- **SETBIT**: (offset) 0 1
- **GETBIT**: (offset) bit
- **BITCOUNT**: BitMap 1 bit
- **BITFIELD**: (type) BitMap bit (offset)
 - type
 - offset
 - u
- **BITFIELD_RO**: BitMap bit
- **BITOP**: BitMap (操作)
- **BITPOS**: bit 0 1

HyperLogLog

- UV
- PV

Redis HyperLogLog String HLL 16kb

操作

- **PFADD** HLL
- **PFCOUNT** HLL
- **PFMERGE** HLL

Redis

- Redis
- Redis
- Redis
- Redis

Redis

RDB

RDB Redis

- save RDB
 - Redis RDB
- bgsave RDB

- Redis[¶] RDB[¶]

- Redis[¶] RDB[¶]

- 亂子进程

```
# 900秒未被访问的key进行bgsave
save 900 1

# 停止RDB
save ""

# 压缩
rdbccompression yes

# RDB文件名
dbfilename dump.rdb

# 压缩目录
dir ./
```

bgsave

- bgsave 亂子进程fork[¶] RDB[¶] fork[¶] RDB[¶]
 - 乱子进程在父进程中运行
 - 乱子进程在父进程中运行
 - 乱子进程在父进程中运行
- fork[¶] copy-on-write[¶]
 - 乱子进程在父进程中运行
 - 乱子进程在父进程中运行



AOF[¶]

Redis[¶] AOF[¶] redis.conf[¶] AOF

AOF[¶] redis.conf[¶] AOF

```
## 亂子进程AOF
appendonly yes

## AOF文件名
appendfilename "appendonly.aof"
```

AOF[¶] redis.conf[¶]

- always 亂子进程
- everysec 亂子进程
- no 亂子进程

```
## 旣存檔案寫入後立即回寫到磁碟AOF檔  
appendfsync always

## 旣存檔案寫入後每秒回寫到磁碟AOF檔  
appendfsync everysec

## 旣存檔案寫入後不回寫到磁碟AOF檔  
appendfsync no
```

	RDB	AOF
数据持久化	将内存中的数据持久化到磁盘	将内存中的数据持久化到磁盘
持久化命令	SAVE	BGSAVE
持久化间隔	无	无
持久化策略	将所有修改过的数据持久化到磁盘	将所有修改过的数据持久化到磁盘
持久化方法	将所有修改过的数据持久化到磁盘	将所有修改过的数据持久化到磁盘
持久化影响	CPU消耗大	IO消耗大

Redis

□□□□□□□□□□□□



4

□□□□□□□□□□

1. ပုံစံ
 - ပုံစံပေါ်လောက်ခြင်း
 - ပုံစံ Replication Id ပေါ်လောက်ခြင်း
 - ပုံစံပေါ်လောက်ခြင်း Replication Id & offset
 2. ပုံစံ
 - ပုံစံ bgsave ပုံစံRDBပေါ်လောက်RDBပုံ
 - ပုံစံRDBပေါ်လောက် repl_backlog ပုံစံပေါ်လောက်ခြင်း
 3. ပုံစံ
 - ပုံစံ repl_backlog ပုံစံပေါ်လောက်ခြင်း



6

- slave offset slave offset master offset **slave** **master**

···

slave···

1. ···
 - ···
 - ··· **replid** ···
 - ··· continue
2. ···
 - ··· **offset**···



repl_baklog···

-
- ··· slave master···



- offset···
- slave master···



··· repl_baklog ··· **slave**···log···



Redis···

Redis···

···

- Sentinel···master slave···
- master···Sentinel···**slave**···**master**···master···
- Sentinel···**Redis**···Redis···



Sentinel···1 ping ···

-
- quorum···
 - quorum···

···**master**

A horizontal row of ten empty square boxes, intended for handwritten responses or drawing.

1

5

- မြန်မာဘာသာ slaveof no one မြန်မာဘာ master
 - မြန်မာဘာ slave မှု slvae ip:port မြန်မာ slave မြန်မာ master မြန်မာ master မြန်မာ
 - မြန်မာဘာသာ slave မြန်မာဘာသာ မြန်မာ master မြန်မာ slave မြန်မာ



Redis

Redis

1

- master master
 - master slave
 - master ping
 - master

1

- redis-cli --cluster

1

Redis master 0-16383 16384

key redis key

- key \in { } \cup { } \cup 1 \cup { } \cup 0000000000
 - key \in { } \cup key \in 00000000

██████████CBC16████████hash████████16384████████████████slot█

1

```
redis-cli --cluster add-node 192.168.1.11 7001
```

- new_host:new_port 重新启动
 - existing_host:existing_port 重新启动 ip 重新映射
 - --cluster-save 重新启动

```
redis-cli --cluster reshard [options]
```

- host:port

1

master slave master

1

cluster failover master cluster failover slave

5



1

5



JVM

Nginx

Lua

-

- nil 空值
 - boolean 布尔值
 - number 数字
 - string 字符串
 - function 函数
 - table 表
 - 表 {} 包含

- II

- local
 - type

```
-- 问候语  
local str = 'hello'  
  
-- 问候语  
local num = 21  
  
-- 问候语  
local flag = true  
  
-- 问候语  
local arr = {'java','python','lua'}
```

```
-- 表
local map = {name = 'jack', age = 21}
```

- 表

```
-- 表
print(arr[1])
-- 表
print(map['name'])
print(map.name)
```

- 表

- 表

```
-- 表 key值表 table
local arr = {'java','python','lua'}
-- 表
for index,value in ipairs(arr) do
    print(index,value)
end
```

- 表

```
-- 表
local map = {name = 'jack',age = 21}
-- 表
for key,value in pairs(map) do
    print(key,value)
end
```

- 表

- 表操作

```
function 表(argument 1,argument2 ... ,argumentn)
    return 表
end
```

- 表操作

- and 表
 - or 表
 - not 表

```
if(表)
then
    --[ 表 true 表 --]
else
```

```
--[ false --]
end
```

□□

Redis[®]c[®]SDS

□□□

□□□SDS

□□Redis[®]c[®]SDS

Redis[®]c[®]SDS

- SDS[®]c[®]SDS

```
struct __attribute__((__packed__)) sdshdr8{
    uint8_t len; /* buf®len 8bit 0~255 */
    uint8_t alloc; /* buf®alloc 8bit 0~255 */
    unsigned char flags; /* SDS®flags 8bit */
    char buf[];
}
```

- len



- alloc

- SDS[®]alloc
- 1M+1
- 1M+1+1M+1
- +1\0

IntSet

□Redis[®]set[®]c[®]IntSet

□□□

```
typedef struct intset{
    uint32_t encoding; /* 16bit32bit64bit */
    uint32_t length; /* count */
    int8_t contents[]; /* encoding */
}intset;
```

encoding

```
##define INTSET_ENC_INT16 (sizeof(int16_t)) /* 2字节, java short */
##define INTSET_ENC_INT32 (sizeof(int32_t)) /* 4字节, java int */
##define INTSET_ENC_INT64 (sizeof(int64_t)) /* 8字节, java long */
```

Redis中 intset 结构体的内存布局 contents 字段



int16_t 结构体的内存布局 INTSET_ENC_INT16 内存布局

- encoding 4字节
- length 4字节
- contents 2字节*3=6字节

startPtr + (sizeof(int16) * index)

- index 4字节
- startPtr 8字节
- sizeof() 4字节 encoding 4字节

inset()

- 该函数实现 inset 结构体的插入操作
 1. 遍历链表找到插入位置
 2. 插入新节点
 3. 重新计算长度
 4. 将 intset 的 encoding 和 length 赋值

Dict

字典结构

- 一个哈希表
- 表头
- 哈希槽

字典

```
typedef struct dictht{
    // entry
    // table<dictEntry>*dictEntry*
    // 表头
    dictEntry **table;
    // 表头大小
    unsigned long size;
    // 表头大小-1
    unsigned long sizemask;
    // entry
    unsigned long used;
}dictht;
```

redis

```
typedef struct dictEntry{
    void *key; // key
    union {
        void *val;
        uint64_t u64;
        int64_t s64;
        double d;
    }v; // value
    // dictEntry*next;
} dictEntry;
```

Dict Redis key hash h & sizemask

- size <= n & sizemask
- size > n & sizemask

4

0000 0000 0000 0100

7

0000 0000 0000 0111

11

0000 0000 0011

3

00

```
typedef struct dict{
    dictType *type; // dict type
    void *privdata; // private data
    dichtht ht[2]; // dict hash table
    long rehashidx; // rehash index -1
    int16_t pauserehash; // rehash pause 1 or 0
} dict;
```

Dict

Redis key hash h & sizemask

Dict(LoadFactor = used / size)

- LoadFactor >= 1 BGSAVE or BGREWRITEAOF
- CPU usage
- BGSAVE Redis RDB
- RDB
- BGREWRITEAOF AOF
- AOF

- LoadFactor > 5

LoadFactor < 0.1 时不会进行重hash

DictRehash

当key的大小大于size或小于sizemask时会进行重hash

当key的大小等于sizemask时会进行重hash

1. 计算 realSize
 - size > sizemask 时 `dich.ht[0].used + 1`
 - size < sizemask 时 `dich.ht[0].used + 1` 且 `size % 4 == 0`
2. 将 realSize 赋值给 dictht 的 dict.ht[1]
3. 将 dict.rehashidx = 0 赋值给 rehash
4. 将 dict.ht[0] 赋值给 dictEntry 的 rehash 为 dict.ht[1]
5. 将 dict.ht[1] 赋值给 dict.ht[0] 且 dict.ht[1] 赋值给 dictht 的 dict.ht[0]

DictRehash

当key的大小等于sizemask时会进行重hash

DictRehash流程

- 将 dict.rehashidx 赋值为 -1
- 将 dict.ht[0].table.[rehashindex] 赋值给 entry 的 rehash 为 dict.ht[1] 且将 rehashidx++
- 将 dict.ht[0] 赋值给 dict.ht[1]

循环

- 将 rehash 赋值为 dict.ht[1] 且将 dict.ht[0] 赋值为 dict.ht[1] 且将 dictht 赋值为 dict

ZipList

链表的实现



- zlend 赋值为 0xff
- zlbytes 赋值为 0
- zltail 赋值为 0
- zllen 赋值为 entry 节点数
 - entry 节点数为 16的倍数
- entry 节点数
 - entry 节点数为 1到5之间



- previous_entry_length 节点数为 1到5之间
 - 节点数为 254的倍数



- 11. content encoding 11.1.
 - encoding 4.1.



- contents

1

- ZipList

ZipList

- 250-253entryentry previous_entry_length 1
 - 254entryentry previous_entry_length 5
 -

QuickList

 ZipList

- `list-max-ziplist-size` QuickList ZipList entry
 - `list-compress-depth` ZipList

QuickList

```
typedef struct quicklist{
    // 亂數
    quicklistNode *head;
    // 亂數
    quicklistNode *tail;
    // ziplist\entry
    unsigned long count;
    // ziplists\len
    unsigned long len;
    // ziplist\entry\len\len-2
    int fill : QL_FILL_BITS;
    // 亂數
    unsigned int compress : QL_COMP_BITS;
    // 亂數
    unsigned int bookmark count : QL_BM_BITS;
```

```
    quicklistBookmark bookmarks[];
}quicklist;
```

QuickListNode

```
typedef struct quicklistNode{
    // ...
    struct quicklistNode *prev;
    // ...
    struct quicklistNode *next;
    // ...
    unsigned char *zl;
    // ...
    unsigned int sz;
    // ...
    unsigned int entry;
    unsigned int count : 16;
    // ...
    unsigned int encoding : 2 ;
    // ...
    unsigned int container : 2;
    // ...
    unsigned int recompress : 1;
    unsigned int attempted_compress : 1;
    unsigned int extra : 10;
} quicklistNode;
```

SkipList

redis的SkipList实现

- 简单的SkipList实现
- 复杂的SkipList实现



...

```
typedef struct zskiplist{
    // ...
    struct zskiplistNode *head,*tail;
    // ...
    unsigned long length;
    // ...
    int level;
}zskiplist;
```

...

```
typedef struct zskiplistNode{
    sds ele; // ...
}
```

```

double score; // 分数
struct zskiplistNode *backward; // 前向指针
struct zskiplistLevel{
    struct zskiplistNode *forward;// 后向指针
    unsigned long span; // 跨度
}level []; // 层级
}zskiplistNode;

```

等等

- 分数score指向ele
- 分数score指向score指向ele指向的
- 分数指向的范围是1~32位
- 分数指向的范围是1~32位

RedisObject

Redis对象RedisObjectRedisObjectRedis

```

typedef struct redisObject{
    unsigned type:4; // 对象类型string,hash,list,set,zset
    unsigned encoding:4; // 编码方式11
    unsigned lru:LRU_BITS;// Redis对象LRU
    int refcount; // 参照计数器0表示对象未被引用
    void *str; // 对象字符串表示形式
} robj;

```

等等等等等等

- STRING INT RAW EMBSTR
- LIST LinkedList ZipList 3.2 QuickList 3.2
- SET intset HT Hash Table Redis Dict
- ZSET ZipList HT SkipList
- HASH ZipList HT

String

- 原生字符串RAW SDS 512MB
- SDS SDS 44 EMBSTR RedisObject head SDS SDS



- 长整型LONG_MAX 原生字符串INT RedisObject * ptr 80 SDS

List

等等等等等等

- 3.2 Redis ZipList LinkedList 512 64 ZipList LinkedList
- 3.2 Redis QuickList List



Set

三

- 
 - 
 - 

5

- Set \sqsubseteq HT \sqsubseteq (Dict) \sqsubseteq Dict \sqsubseteq key \sqsubseteq string \sqsubseteq value \sqsubseteq null
 - \sqsubseteq \sqsubseteq \sqsubseteq \sqsubseteq \sqsubseteq \sqsubseteq set-max-insert-entries \sqsubseteq Set \sqsubseteq IntSet



ZSet

SortedSet score member

- score
 - member
 - member score

1


```
typedef struct zset{  
    // Dict[]  
    dict *dict;  
    // SkipList[]  
    zskiplist *zsl;  
} zset;
```



- `zset_max_ziplist_entries` `zset_max_ziplist_value` `ZipList` `元素`
 - `ZipList` `元素``score``element``entry`
 - `score``元素``score``元素``score``元素`



Hash

Zset

- Hash 结构中 ZipList 表示的键值对 entry 包含 field 和 value



- Redis Hash 实现 HT 包含 Dict 表示的键值对
 - ZipList 表示键 hash-max-ziplist-entries
 - ZipList 表示值 hash-max-ziplist-value



Redis 命令

命令行工具

32 位系统 \$2^{32}\$ \$4GB

命令行工具通过命令行输入命令，直接执行命令。

- Redis 提供了丰富的命令行工具
- Redis 提供了 Redis 客户端 Ring 3 用于连接 Redis 服务器
- Redis 提供了 Redis 客户端 Ring 0 用于连接 Redis 服务器



- Redis 提供了丰富的命令行工具



IO 模型 Linux IO 模型

- Linux 提供了阻塞 IO 模型
- Linux 提供了非阻塞 IO 模型



IO 模型

- 阻塞 IO 模型
- 非阻塞 IO 模型

IO 模型

- 阻塞 IO 模型
- 非阻塞 IO 模型



IO Blocking IO

阻塞 IO 模型

- 阻塞 IO 模型



IO Nonblocking IO

IO recvfrom 非阻塞IO実装

- 直接IO
- 间接IO
 - CPU時間



IO Multiplexing

IO多路复用IO实现 recvfrom 实现

- 直接 recvfrom IO CPU时间CPU
- 间接 recvfrom IO时间

Socket Socket Socket Socket Socket Socket

IO

- 直接IO Socket Socket
- 间接FD Linux Linux Linux Socket
- 间接FD FD
 - select FD
 - FD readable
 - recvfrom



FD

- select
- poll
- epoll

IO

- select poll FD FD FD FD
- epoll FD FD FD FD

select

Linux I/O

```
// 定义 __fd_mask, long int(32bit)
typedef long int __fd_mask;
```

```
// fd_set 定义fd
typedef struct{
```

```

// fds_bits long int 1024/32=32
// 1024 bit 1024bit / 32 = 32
__fd_mask fds_bits[ __FD_SETSIZE / __NFDBITS ];
} fd_set;

// select FD
int select(
    int nfds, // fd_set fd+1
    fd_set *readfds, // fd
    fd_set *writefds, // fd
    fd_set *exceptfds, // fd
    // null-0-0-0
    struct timeval *timeout
)

```

步骤

1. fd_set fd_set fd 1 nfds fd fd 1
2. select fd_set fd
3. fd_set fd fd fd_set fd fd fd fd
4. fd fd fd fd
5. select fd fd fd_set fd_set fd_set fd
6. fd_set fd fd fd



步骤

- select
- select fd fd_set
- fd_set fd

poll

步骤

```

// pollfd
#define POLLIN //0
#define POLLOUT //0
#define POLLERR //0
#define POLLNVAL //fd0

// pollfd
struct pollfd {
    int fd; //fd */
    short int events; /* */
    short int revents; /* */
};

// poll
int poll(
    struct pollfd *fds, //pollfd

```

```
    nfds_t nfds, //FD数目  
    int timeout //超时  
);
```

返回值

1. 通过 pollfd 结构体的 fd 字段返回文件描述符
2. 通过 poll 结构体的 pollfd 数组返回所有文件描述符
3. 通过 fd 数组返回文件描述符
4. 通过 pollfd 结构体的 revents 字段返回事件0到n-1的 pollfd 结构体的 fd 字段
5. 通过 n 返回0
6. 通过0返回 pollfd 结构体的 fd

epoll

通过select和poll实现的 epoll

```
struct eventpoll{  
    struct rb_root rbr;           // 红黑树根节点FD(epitem)  
    struct list_head rdlist; // 读事件队列FD(epitem)  
}  
  
// 1. 创建eventpoll并返回对应的epfd  
int epoll_create(int size);  
  
// 2. 将FD添加到epoll并设置回调函数ep_poll_callback  
// callback函数参数为fd和rdlist  
int epoll_ctl(  
    int epfd, // epoll句柄  
    int op,      // 操作类型ADD|MOD|DEL  
    int fd, // 监听fd  
    struct epoll_event *event // 事件结构体  
);  
  
// 3. 从rdlist中取出事件并返回对应的FD  
int epoll_wait(  
    int epfd, // eventpoll句柄  
    struct epoll_event *events, // 事件数组FD  
    int maxevents, // events数量  
    int timeout // 超时-1毫秒到0毫秒  
);
```

返回值

1. 通过 epoll_create 返回 eventpoll 结构体并将其插入到全局的事件队列中
2. 通过将FD添加到epoll_ctl 返回 eventpoll 结构体的 ep_poll_callback 回调函数
将FD添加到事件队列RDlist的FD中 eventpoll 结构体
3. 通过 epoll_wait 从事件队列 RDlist 中取出 FD(epitem) 为 eventpoll 结构体的 FD(epitem) 为事件
FD(epitem) 为事件结构体的 epoll_event 为
4. 通过 epoll_events 为事件 FD(epitem) 为IO事件



IO

FD epoll_wait

- **LevelTriggered** FD epoll_wait
- **EdgeTriggered** FD epoll_wait

IO

- ET epoll_wait LT
- FD epoll_wait LT
- ET IO FD LT

IO WEB

- serverSocket fd ssfd
- serverSocket ssfd
- socket fd



IO Signal Driven IO

SIGIO FD SIGIO



IO

- IO SIGIO
- IO

IO ASynchronous IO

API IO ASynchronous IO



IO

- IO

IO

IO

Redis

Redis

- Redis
- Redis

Redis

- Redisのデータ構造
- データ操作命令
- データ構造操作命令

Redisのデータ



- データ構造
- データ操作
 - Redisのデータ構造
 - データ操作命令Redisのデータ構造
 - データ構造操作命令Redisのデータ構造
 - データ構造操作命令

RESPプロトコル

RedisのプロトコルCSプロトコル

1. リクエストを送信する
2. レスポンスを受信する

リクエストとレスポンスの構造

RedisのプロトコルRESP

RESPプロトコルの構造

- リンクフレーム '+ ' リンクフレームCRLF("\r\n")
 - リンクフレーム "\r\n" リンクフレーム
- エラーフレーム '- ' エラーフレーム
- フィールドフレーム ':' フィールドフレームCRLF
- バイナリーフレーム '\$' バイナリーフレーム512MB
 - フィールドフレーム
 - フィールド0フィールド
 - フィールド-1フィールド



- フィールド '* ' フィールド



リクエスト

リクエスト

リクエスト expire RedisのkeyのTTLを設定する

DB

Redis یه key-value store یهیه که key-value یهیه Dict یهیه

database یهیه Dict یهیه key-value یهیه key-TTL

```
typedef struct redisDb{
    dict *dict;                                // یهیه key-value
    dict *expires;                             // یهیه key-TTL
    TTL*key
    dict *blocking_keys;                      //
    dict *ready_keys;                          //
    dict *watched_keys;                        //
    int id;                                    // یهیه ID
    long long avg_ttl;                         // یهیه TTL
    unsigned long expires_cursor;             // expire یهیه dict
    list *defrag_later;                        // یهیه key
} redisDb;
```



یهیه

یهیه TTL یهیه key یهیه key یهیه

یهیه

یهیه key یهیه

- Redis یهیه serverCron() یهیه key یهیه **Slow**
 - server.hz یهیه 10 یهیه 10 یهیه **100ms**
 - 25% یهیه
 - db یهیه db یهیه bucket یهیه **20 key**
- Redis یهیه beforeSleep() یهیه key یهیه **Fast**
 - beforeSleep() یهیه **Fast** یهیه **2ms**
 - 1ms**
 - db یهیه db یهیه bucket یهیه **20 key**

یهیه

یهیه Redis یهیه Redis یهیه Redis یهیه key یهیه

Redis یهیه 8 یهیه key

- noeviction یهیه key یهیه
- volatile-ttl یهیه TTL یهیه key یهیه TTL یهیه TTL یهیه
- allkeys-random یهیه key یهیه
- volatile-random یهیه TTL یهیه key یهیه
- allkeys-lru یهیه key یهیه **LRU** یهیه
- volatile-lru یهیه TTL یهیه key یهیه **LRU** یهیه
- allkeys-lfu یهیه key یهیه **LFU** یهیه
- volatile-lfu یهیه TTL یهیه key یهیه **LFU** یهیه

Redis缓存淘汰策略

- **LRU** Least Recently Used
- **LFU** Least Frequently Used

Redis中 RedisObject 结构体如下所示

```
typedef struct redisObject{
    unsigned type:4; // string,hash,list,set,zset
    unsigned encoding:4; // raw,compressed,11
    unsigned lru:LRU_BITS;// LRU位数24bit
    // LFU位数16+LFU衰减时间8位数
    int refcount; // 0~1000000000
    void *str; // string,hash,list,set,zset
} robj;
```

LFU缓存淘汰策略的实现逻辑如下所示

1. $0 \sim 10000$ R
2. $(1 / (\text{R} * \text{lru_log_factor}) + 1) \leq P \leq \text{lru_log_factor}$ $\text{lru_log_factor} = 10$
3. $R < P \leq 255$
4. $\text{lru_decay_time} = \text{lru_decay_time} + 1$

二、

- $P \leq R$ $R < P \leq 255$
- $R < P \leq 255$

三、



BigKey

1. 定义

Redis中key的大小限制

- $[key]:[value]:[id]$
- $44\text{字节} \leq \text{String} \leq \text{Raw}$
- MaxSize

BigKey BigKey

- key:value **10kb**
- key数量 **1000**

2. 测量

- redis-cli --bigkeys
 - Key大小Top1 big key
- scan 测量key
 - Redis命令
 - keys * key Redis命令

- `unlink` 释放 big key

性能优化

Redis 通过以下方式提高性能：

命令行优化

- `mset` 和 `hmset` 命令
- `Pipeline` 命令
- `UNLINK`

内存优化

- 使用 Spring 的 `RedisTemplate` 和 `slot`



持久化

配置文件

- Redis 配置文件
- **AOF**
- **slave** 和 **RDB**
- `no-appendfsync-on-rewrite=yes` 表示 AOF rewrite 时 RDB fork，不使用 AOF

慢日志 Redis 慢日志配置

- `slowlog-log-slower-than` 慢日志

慢日志 Redis 慢日志配置

- `slowlog-max-len` 慢日志

慢日志 Redis 慢日志

- `slowlog len` 慢日志
- `slowlog get[n]` 慢日志
- `slowlog reset` 慢日志

Redis 安全性

- Redis 安全性 Redis 安全性
- Redis 安全性 redis 安全性 Redis 安全性
- Redis 安全性 Redis 安全性

集群优化

- 集群优化 集群优化
 - `cluster-require-full-coverage no`
- 集群优化 ping Redis 集群优化 Redis
- Redis 集群优化 Redis