Redis

# Redis简介

## 1.1Redis简介

Redis 是一个开源（BSD许可）的，内存中的数据结构存储系统，它可以用作数据库、缓存和消息中间件。 它支持多种类型的数据结构，如 字符串（strings）， 散列（hashes）， 列表（lists）， 集合（sets）， 有序集合（sorted sets） 与范围查询， bitmaps， hyperloglogs 和 地理空间（geospatial） 索引半径查询。 Redis 内置了 复制（replication），LUA脚本（Lua scripting）， LRU驱动事件（LRU eviction），事务（transactions） 和不同级别的 磁盘持久化（persistence）， 并通过 Redis哨兵（Sentinel）和自动 分区（Cluster）提供高可用性（high availability）。

REmote DIctionary Server(Redis) 是一个由Salvatore Sanfilippo写的key-value存储系统。

Redis是一个开源的使用ANSI C语言编写、遵守BSD协议、支持网络、可基于内存亦可持久化的日志型、Key-Value数据库，并提供多种语言的API。

它通常被称为数据结构服务器，因为值（value）可以是 字符串(String), 哈希(Map), 列表(list), 集合(sets) 和 有序集合(sorted sets)等类型。

**三)Redis的介绍三**

Redis 是完全开源免费的，遵守BSD协议，是一个高性能的key-value数据库。

Redis 与其他 key - value 缓存产品有以下三个特点：

Redis支持数据的持久化，可以将内存中的数据保存在磁盘中，重启的时候可以再次加载进行使用。

Redis不仅仅支持简单的key-value类型的数据，同时还提供list，set，zset，hash等数据结构的存储。

Redis支持数据的备份，即master-slave模式的数据备份

**四）Redis的优势**

**性能极高** – Redis能读的速度是110000次/s,写的速度是81000次/s 。

丰富的数据类型 – Redis支持二进制案例的 Strings, Lists, Hashes, Sets 及 Ordered Sets 数据类型操作。

**原子** – Redis的所有操作都是原子性的，意思就是要么成功执行要么失败完全不执行。单个操作是原子性的。多个操作也支持事务，即原子性，通过MULTI和EXEC指令包起来。

**丰富的特性** – Redis还支持 publish/subscribe, 通知, key 过期等等特性。

## Redis安装

### Windos下的安装

解压即可

### Linux下的安装

可以通过网络下载进行安装

[root@J2001 redis-stable]# **wget http://download.redis.io/releases/redis-stable.tar.gz**

[root@J2001 redis-stable]# **tar -zxvf redis-stable.tar.gz -C /usr/local/sofeware/**

[root@J2001 redis-stable]# **cd redis-stable/**

[root@J2001 redis-stable]# **make**

[root@J2001 redis-stable]# **make install**

### Redis的启动和停止

1. **不使用任何配置文件直接启动(使用比较少)**

[root@J2001 src]#

[root@J2001 src]# **./redis-server 启动redis服务 不使用配置文件**

[root@J2001 ~]# cd /usr/local/sofeware/redis-stable/

[root@J2001 redis-stable]#

[root@J2001 redis-stable]# cd src

[root@J2001 src**]# ./redis-cli 启动Redis客户端**

127.0.0.1:6379> **set myName guigu**

OK

127.0.0.1:6379> get myName

"guigu"

127.0.0.1:6379>

1. **停止Redis服务**

**查看Redis进程**

**[root@J2001 ~]# ps -ef | grep 6379**

**在.redis-cli 客户端下 输入shutdown 关闭 redis服务**

1. **使用任何配置文件直接启动**

**[root@J2001 redis-stable]# src/redis-server redis.conf**

**如果此时需要redis在后台运行**

**修改配置文件 redis.conf**

**daemonize yes**

**然后再次启动即可**

**[root@J2001 redis-stable]# src/redis-server redis.conf**

## 命令返回值

Redis命令返回值类型主要包含

1. 状态回复

[root@J2001 redis-stable]# src/redis-cli

127.0.0.1:6379> ping

PONG

状态的回复 最简单的回复 表示连接成功

1. 错误回复

当命令不存在的时候

127.0.0.1:6379> hello guigu

(error) ERR unknown command 'hello'

1. 整数回复

Redis虽然没有提供整数类型 但是提供了一些用于整数操作的命令

127.0.0.1:6379> incr foo

(integer) 1

127.0.0.1:6379> incr foo

(integer) 2

1. 字符串回复

最常见的一种回复方式 是得到某个结果

127.0.0.1:6379> get youname

"zhangsan"

127.0.0.1:6379> get foo

"2"

1. 多行字符串回复

127.0.0.1:6379> keys \* 查看当前可用的所有的key值

1) "youname"

2) "liuhuan"

3) "foo"

# Redis的数据类型

Redis支持五种数据类型 string（字符串） hash（哈希） list（列表） set（集合） zset(有序集合)

## 基础入门命令

**1）设置值 和得到所有的值**

127.0.0.1:6379> **set bar 1**

OK

127.0.0.1:6379> **keys \***

1) "bar"

2) "youname"

3) "liuhuan"

4) "foo"

**2）判断一个键是否存在**

127.0.0.1:6379> **exists myName**

(integer) 0

127.0.0.1:6379> **exists youname**

(integer) 1

127.0.0.1:6379> **exists foo**

(integer) 1

**3）删除键**

127.0.0.1:6379> **del foo**

(integer) 1

127.0.0.1:6379> **exists foo**

(integer) 0

**4）获取键值的数据类型**

127.0.0.1:6379> **type bar**

string

127.0.0.1:6379> **type liuhuan**

string

127.0.0.1:6379> lpush guigu 1

(integer) 1

127.0.0.1:6379> **type guigu**

list

**5）设置键的有效期**

127.0.0.1:6379> **expire school 10 设置键的有效期是10秒**

(integer) 1

127.0.0.1:6379> **keys \***

1) "liuhuan"

2) "school"

3) "bar"

4) "youname"

5) "guigu"

127.0.0.1:6379> **get school**

"hzguigu"

127.0.0.1:6379> **keys \***

1) "liuhuan"

2) "bar"

3) "youname"

4) "guigu"

127.0.0.1:6379>

**6）修改键的名称**

127.0.0.1:6379> **set msg "helloredis"**

OK

127.0.0.1:6379> get msg

"helloredis"

127.0.0.1:6379> **rename msg greeting 重命名一个值**

OK

127.0.0.1:6379> get msg

如果重命名的值已经存在 则会覆盖掉原来的值

127.0.0.1:6379> **keys \***

1) "liuhuan"

2) "greeting"

3) "bar"

4) "youname"

5) "guigu"

127.0.0.1:6379> **get bar**

"1"

127.0.0.1:6379> **get greeting**

"helloredis"

127.0.0.1:6379>  **rename greeting bar**

OK

127.0.0.1:6379> **get bar**

"helloredis"

## 字符串类型(string)

string是redis最基本的类型，你可以理解成与Memcached一模一样的类型，一个key对应一个value。

string类型是二进制安全的。意思是redis的string可以包含任何数据。比如jpg图片或者序列化的对象 。

string类型是Redis最基本的数据类型，一个键最大能存储512MB

### 基础命令

**1）赋值与取值**

Redis SET 命令用于设置给定 key 的值。如果 key 已经存储其他值， SET 就覆写旧值，且无视类型。

语法

redis SET 命令基本语法如下：

redis 127.0.0.1:6379> **SET KEY\_NAME VALUE**

Redis Get 命令用于获取指定 key 的值。如果 key 不存在，返回 nil 。如果key 储存的值不是字符串类型，返回一个错误。

语法

redis Get 命令基本语法如下：

redis 127.0.0.1:6379> **GET KEY\_NAME**

127.0.0.1:6379> set mykey "redis"

OK

127.0.0.1:6379> get mykey

"redis"

**可以使用Java连接Redis 完成简单的取值和赋值**

1. 导入redis相关的jar包

使用maven工程自动引入

<dependency>

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

<version>2.9.0</version>

</dependency>

1. 编写java代码

**public** **class** RedisConnect {

**public** **static** **void** main(String[] args) {

// 连接指定位置的redis

Jedis jedis = **new** Jedis("192.168.118.6", 6379);

jedis.auth("guigu");

System.***out***.println("服务器正在运行:"+jedis.ping());

}

}

**public** **class** RedisStringTest {

**public** **static** **void** main(String[] args) {

Jedis jedis=**new** Jedis("192.168.118.6",6379);

jedis.auth("guigu");

jedis.set("username", "zhangsanfeng");

System.***out***.println("获取到的值是:"+jedis.get("liuhuan"));

}

}

修改配置文件redis.conf

1. 关闭防火墙

Systemctl stop firewalld

Systemctl disable firewalld

2)修改配置文件 将默认端口号 127.0.0.1 注释掉

#bind 127.0.0.1

3)取消保护模式 设置权限

protected-mode no

4）设置权限 设置密码

requirepass guigu

### 其他命令的补充

**1.INCR 和 INCRBY**

增加1 增加指定的整数

127.0.0.1:6379> incr school

(integer) 2

127.0.0.1:6379>

127.0.0.1:6379> get school

"2"

127.0.0.1:6379> incrby school 5

(integer) 7

127.0.0.1:6379> get school

"7"

**2.DECR**

127.0.0.1:6379> **decr school**

(integer) 6

127.0.0.1:6379> **get school**

"6"

1. **同时设置多个值 同时获取多个值**

127.0.0.1:6379> **mset age 14 address hangzhou sex female**

OK

127.0.0.1:6379> **keys \***

1) "school"

2) "liuhuan"

3) "bar"

4) "age"

5) "address"

6) "guigu"

7) "sex"

8) "mykey"

9) "username"

10) "youname"

127.0.0.1:6379> **mget username age address sex**

1) "zhangsanfeng"

2) "14"

3) "hangzhou"

4) "female"

1. **同时设置追加和得到长度**

127.0.0.1:6379> **APPEND address "xiasha"**

(integer) 14

127.0.0.1:6379> **get address**

"hangzhouxiasha"

127.0.0.1:6379> **strlen address**

(integer) 14

## 散列类型（hash）

Redis hash 是一个键值(key=>value)对集合。

Redis hash是一个string类型的field和value的映射表，hash特别适合用于存储对象。

Redis hash 是一个string类型的field和value的映射表，hash特别适合用于存储对象。

Redis 中每个 hash 可以存储 232 - 1 键值对（40多亿）。

### 基础常用命令

**1）赋值与取值**

Redis Hget 命令用于返回哈希表中指定字段的值。

语法

redis Hget 命令基本语法如下：

redis 127.0.0.1:6379> **HGET KEY\_NAME FIELD\_NAME**

Redis Hset 命令用于为哈希表中的字段赋值 。

如果哈希表不存在，一个新的哈希表被创建并进行 HSET 操作。

如果字段已经存在于哈希表中，旧值将被覆盖。

语法

redis Hset 命令基本语法如下：

redis 127.0.0.1:6379> **HSET KEY\_NAME FIELD VALUE**

**1）设置单个值和获取单个值**

127.0.0.1:6379> **hset car price 500000**

(integer) 1

127.0.0.1:6379> hset car name Audi

(integer) 1

127.0.0.1:6379> **hset car color white**

(integer) 1

127.0.0.1:6379> **hget car name**

"Audi"

127.0.0.1:6379> **hget car price**

"500000"

127.0.0.1:6379> type car

hash

**2）同时设置多个值**

127.0.0.1:6379> **hmset person username xiaoming age 23 sex male**

OK

127.0.0.1:6379> **hmget person username age sex**

1) "xiaoming"

2) "23"

3) "male"

127.0.0.1:6379> **hgetall person**

1) "username"

2) "xiaoming"

3) "age"

4) "23"

5) "sex"

6) "male"

**public** **class** RedisStringTest {

**public** **static** **void** main(String[] args) {

Jedis jedis=**new** Jedis("192.168.118.6",6379);

jedis.auth("guigu");

Map<String,String> perosnMap=jedis.hgetAll("person");

perosnMap.forEach((key,value)->System.***out***.println(key+"---"+value));

}

}

**3）判断字段是否存在**

127.0.0.1:6379> **HEXISTS car model**

(integer) 0

127.0.0.1:6379> **HEXISTS car name**

(integer) 1

**4）增加数字**

Redis Hincrby 命令用于为哈希表中的字段值加上指定增量值。

增量也可以为负数，相当于对指定字段进行减法操作。

如果哈希表的 key 不存在，一个新的哈希表被创建并执行 HINCRBY 命令。

如果指定的字段不存在，那么在执行命令前，字段的值被初始化为 0 。

对一个储存字符串值的字段执行 HINCRBY 命令将造成一个错误。

本操作的值被限制在 64 位(bit)有符号数字表示之内。

语法

redis Hincrby 命令基本语法如下：

redis 127.0.0.1:6379> **HINCRBY KEY\_NAME FIELD\_NAME INCR\_BY\_NUMBER**

127.0.0.1:6379> HINCRBY student score 80

(integer) 80

127.0.0.1:6379> hget student score

"80"

**5）删除字段**

127.0.0.1:6379> hdel car price

(integer) 1

127.0.0.1:6379> hdel car price

(integer) 0

127.0.0.1:6379> HGETALL car

1) "name"

2) "Audi"

3) "color"

4) "white"

### 命令补充

1） 获取哈希表所有的值

Redis Hvals 命令返回哈希表所有域(field)的值。

语法

redis Hvals 命令基本语法如下：

redis 127.0.0.1:6379> HVALS KEY\_NAME FIELD VALUE

127.0.0.1:6379> **hvals car**

1) "Audi"

2) "white"

127.0.0.1:6379> **hvals person**

1) "xiaoming"

2) "23"

3) "male"

1. 获取所有的key值

Redis Hkeys 命令用于获取哈希表中的所有域（field）。

语法

redis Hkeys 命令基本语法如下：

redis 127.0.0.1:6379> HKEYS key

127.0.0.1:6379> **hkeys person**

1) "username"

2) "age"

3) "sex"

127.0.0.1:6379> **hkeys car**

1) "name"

2) "color"

3）获取字段的数量

Redis Hlen 命令用于获取哈希表中字段的数量。

redis Hlen 命令基本语法如下：

redis 127.0.0.1:6379> HLEN KEY\_NAME

127.0.0.1:6379> **hlen car**

(integer) 2

127.0.0.1:6379> **hlen perosn**

(integer) 0

127.0.0.1:6379> **hlen person**

(integer) 3

## 列表类型List

Redis 列表是简单的字符串列表，按照插入顺序排序。你可以添加一个元素到列表的头部（左边）或者尾部（右边）。

Redis列表是简单的字符串列表，按照插入顺序排序。你可以添加一个元素到列表的头部（左边）或者尾部（右边）

一个列表最多可以包含 232 - 1 个元素 (4294967295, 每个列表超过40亿个元素)。

### 基础常用命令

**1）取值和赋值**

127.0.0.1:6379> lpush number 1

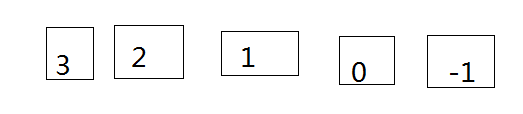
(integer) 1

127.0.0.1:6379> lpush number 2 3

(integer) 3

127.0.0.1:6379> rpush number 0 -1

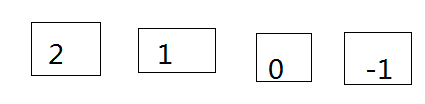
(integer) 5



1. **从列表两端弹出元素**

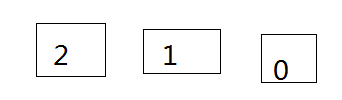
**127.0.0.1:6379> lpop number**

**"3"**



127.0.0.1:6379> rpop number

"-1"



1. **获取列表的长度**

127.0.0.1:6379> llen number

(integer) 3

1. **获取列表的数据**

**Lrange 的命令返回索引之间的所有元素 如果索引是整数就是 下标**

127.0.0.1:6379> lrange number 0 2

1) "2"

2) "1"

3) "0"

-2代表最右边第二个元素 -1 最右边第一个元素 也就是最后一个元素

127.0.0.1:6379> lrange number -2 -1

1) "1"

2) "0"

如果实际索引大于最大索引 返回最大的数据即可

127.0.0.1:6379> lrange number 1 999

1) "1"

2) "0"

**5)删除列表中指定的值**

Redis Lrem 根据参数 COUNT 的值，移除列表中与参数 VALUE 相等的元素。

COUNT 的值可以是以下几种：

count > 0 : 从表头开始向表尾搜索，移除与 VALUE 相等的元素，数量为 COUNT 。

count < 0 : 从表尾开始向表头搜索，移除与 VALUE 相等的元素，数量为 COUNT 的绝对值。

count = 0 : 移除表中所有与 VALUE 相等的值。

redis 127.0.0.1:6379> **LREM KEY\_NAME COUNT VALUE**

127.0.0.1:6379> lrange number 0 -1

1) "2"

2) "1"

3) "0"

4) "2"

#从右边删除第一个值为2的元素

127.0.0.1:6379> **lrem number -1 2**

(integer) 1

127.0.0.1:6379> lrange number 0 -1

1) "2"

2) "1"

3) "0"

127.0.0.1:6379> **rpush number 1 0 5**

(integer) 6

127.0.0.1:6379**> lrange number 0 -1**

1) "2"

2) "1"

3) "0"

4) "1"

5) "0"

6) "5"

#从右侧删除数据 删除2 个1

127.0.0.1:6379> lrem number -2 1

(integer) 2

127.0.0.1:6379> lrange number 0 -1

1) "2"

2) "0"

3) "0"

4) "5"数

### 命令补充

**1）获取和设置指定索引的元素的值**

Redis Lindex 命令用于通过索引获取列表中的元素。你也可以使用负数下标，以 -1 表示列表的最后一个元素， -2 表示列表的倒数第二个元素，以此类推。

语法

redis Lindex 命令基本语法如下：

redis 127.0.0.1:6379> LINDEX KEY\_NAME INDEX\_POSITION

127.0.0.1:6379> **lrange number 0 -1**

1) "2"

2) "1"

3) "0"

127.0.0.1:6379>

127.0.0.1:6379> **lindex number 1**

"1"

127.0.0.1:6379**> lindex number 0**

"2"

127.0.0.1:6379> **lset number 1 9**

OK

127.0.0.1:6379> **lrange number 0 -1**

1) "2"

2) "9"

3) "0"

2.） 只保留列表指定片段

Redis Ltrim 对一个列表进行修剪(trim)，就是说，让列表只保留指定区间内的元素，不在指定区间之内的元素都将被删除。

下标 0 表示列表的第一个元素，以 1 表示列表的第二个元素，以此类推。 你也可以使用负数下标，以 -1 表示列表的最后一个元素， -2 表示列表的倒数第二个元素，以此类推。

语法

redis Ltrim 命令基本语法如下：

redis 127.0.0.1:6379> LTRIM KEY\_NAME START STOP

127.0.0.1:6379> **rpush number 9 3 5 6 12 56 78 123 56**

(integer) 12

127.0.0.1:6379> **lrange number 0 -1**

1) "2"

2) "9"

3) "0"

4) "9"

5) "3"

6) "5"

7) "6"

8) "12"

9) "56"

10) "78"

11) "123"

12) "56"

127.0.0.1:6379> **ltrim number 1 6**

OK

127.0.0.1:6379> **lrange number 0 -1**

1) "9"

2) "0"

3) "9"

4) "3"

5) "5"

6) "6"

3）向列表中插入指定元素

Redis Linsert 命令用于在列表的元素前或者后插入元素。 当指定元素不存在于列表中时，不执行任何操作。 当列表不存在时，被视为空列表，不执行任何操作。 如果 key 不是列表类型，返回一个错误。

语法

redis Linsert 命令基本语法如下：

redis 127.0.0.1:6379> LINSERT KEY\_NAME BEFORE EXISTING\_VALUE NEW\_VALUE

127.0.0.1:6379> **lrange number 0 -1**

1) "9"

2) "0"

3) "33"

4) "9"

5) "3"

6) "5"

7) "6"

127.0.0.1:6379> **linsert number after 9 22**

(integer) 8

127.0.0.1:6379> **lrange number 0 -1**

1) "9"

2) "22"

3) "0"

4) "33"

5) "9"

6) "3"

7) "5"

8) "6"

127.0.0.1:6379> **linsert number before 22 11**

(integer) 9

127.0.0.1:6379> **lrange number 0 -1**

1) "9"

2) "11"

3) "22"

4) "0"

5) "33"

6) "9"

7) "3"

8) "5"

9) "6"

4） 将元素从一个列表转移到另外一个列表

Redis Rpoplpush 命令用于移除列表的最后一个元素，并将该元素添加到另一个列表并返回。

语法

redis Rpoplpush 命令基本语法如下：

redis 127.0.0.1:6379> RPOPLPUSH SOURCE\_KEY\_NAME DESTINATION\_KEY\_NAME

127.0.0.1:6379**> rpush mylist "hello"**

(integer) 1

127.0.0.1:6379> rpush mylist "java"

(integer) 2

127.0.0.1:6379> rpush mylist "abcd"

(integer) 3

127.0.0.1:6379> rpush mylist "qwer"

(integer) 4

127.0.0.1:6379> rpush mylist "guigu"

(integer) 5

127.0.0.1:6379> rpush mylist "hangzhou"

(integer) 6

127.0.0.1:6379> **lrange mylist 0 -1**

1) "hello"

2) "java"

3) "abcd"

4) "qwer"

5) "guigu"

6) "hangzhou"

127.0.0.1:6379> **keys \***

1) "1"

2) "school"

3) "student"

4) "car"

5) "liuhuan"

6) "bar"

7) "number"

8) "person"

9) "age"

10) "address"

11) "guigu"

12) "mylist"

13) "sex"

14) "mykey"

15) "username"

16) "youname"

127.0.0.1:6379> **rpoplpush mylist myotherlist**

"hangzhou"

127.0.0.1:6379> **lrange myotherlist 0 -1**

1) "hangzhou"

127.0.0.1:6379> **rpoplpush mylist myotherlist\**

"guigu"

127.0.0.1:6379> **rpoplpush mylist myotherlist**

"qwer"

127.0.0.1:6379> **lrange myotherlist 0 -1**

1) "qwer"

2) "hangzhou"

127.0.0.1:6379> **keys \***

1) "school"

2) "1"

3) "car"

4) "student"

5) "myotherlist"

6) "myotherlist\\"

7) "person"

8) "number"

9) "sex"

10) "mykey"

11) "username"

12) "liuhuan"

13) "bar"

14) "age"

15) "address"

16) "guigu"

17) "mylist"

18) "youname"

## 集合类型(set)

Redis的Set是string类型的无序集合。

集合是通过哈希表实现的，所以添加，删除，查找的复杂度都是O(1)。

Redis 的 Set 是 String 类型的无序集合。集合成员是唯一的，这就意味着集合中不能出现重复的数据。

Redis 中集合是通过哈希表实现的，所以添加，删除，查找的复杂度都是 O(1)。

集合中最大的成员数为 2^32 - 1 (4294967295, 每个集合可存储40多亿个成员)。

### 常用命令

**1.增加/删除元素**

Redis Sadd 命令将一个或多个成员元素加入到集合中，已经存在于集合的成员元素将被忽略。

假如集合 key 不存在，则创建一个只包含添加的元素作成员的集合。

当集合 key 不是集合类型时，返回一个错误。

注意：在Redis2.4版本以前， SADD 只接受单个成员值。

语法

redis Sadd 命令基本语法如下：

redis 127.0.0.1:6379> **SADD KEY\_NAME VALUE1..VALUEN**

Redis Srem 命令用于移除集合中的一个或多个成员元素，不存在的成员元素会被忽略。

当 key 不是集合类型，返回一个错误。

在 Redis 2.4 版本以前， SREM 只接受单个成员值。

语法

redis Srem 命令基本语法如下：

redis 127.0.0.1:6379> **SREM KEY MEMBER1..MEMBERN**

127.0.0.1:6379> **sadd letters a**

(integer) 1

127.0.0.1:6379> **srem letters a d**

(integer) 2

127.0.0.1:6379> SMEMBERS letters

1) "b"

2) "c"

2）获取集合中的所有元素

Redis Smembers 命令返回集合中的所有的成员。 不存在的集合 key 被视为空集合。

语法

redis Smembers 命令基本语法如下：

redis 127.0.0.1:6379> SMEMBERS key

127.0.0.1:6379> **SMEMBERS letters**

1) "b"

2) "a"

3) "c"

127.0.0.1:6379> **sadd letters a a d b c**

(integer) 1

127.0.0.1:6379> **SMEMBERS letters**

1) "c"

2) "b"

3) "d"

4) "a"

3）判断元素是否在集合中存在

sismember

Redis Sismember 命令判断成员元素是否是集合的成员。

语法

redis Sismember 命令基本语法如下：

redis 127.0.0.1:6379> SISMEMBER KEY VALUE

127.0.0.1:6379> **SMEMBERS letters**

1) "b"

2) "c"

127.0.0.1:6379> **sismember letters a**

(integer) 0

127.0.0.1:6379> **sismember letters b**

(integer) 1

4)集合之间的运算

4.1）Redis Sinter 命令返回给定所有给定集合的**交集。** 不存在的集合 key 被视为空集。 当给定集合当中有一个空集时，结果也为空集(根据集合运算定律)。

语法

redis Sinter 命令基本语法如下：

redis 127.0.0.1:6379> **SINTER** KEY KEY1..KEYN

4.2）Redis Sdiff 命令返回给定集合之间的**差集**。不存在的集合 key 将视为空集。

差集的结果来自前面的 FIRST\_KEY ,而不是后面的 OTHER\_KEY1，也不是整个 FIRST\_KEY OTHER\_KEY1..OTHER\_KEYN 的差集。

实例:

key1 = {a,b,c,d}

key2 = {c}

key3 = {a,c,e}

SDIFF key1 key2 key3 = {b,d}

4.3）Redis Sunion 命令返回给定集合的**并集**。不存在的集合 key 被视为空集。

语法

redis Sunion 命令基本语法如下：

redis 127.0.0.1:6379> **SUNION K**EY KEY1..KEYN

127.0.0.1:6379> **sadd setA 1 2 3 4 5**

(integer) 5

127.0.0.1:6379> **smembers setA**

1) "1"

2) "2"

3) "3"

4) "4"

5) "5"

127.0.0.1:6379> **sadd setB 3 4 5**

(integer) 3

127.0.0.1:6379> **smembers setB**

1) "3"

2) "4"

3) "5"

127.0.0.1:6379> **sdiff setA setB**

1) "1"

2) "2"

127.0.0.1:6379>

127.0.0.1:6379> **sadd setC 4 5**

(integer) 2

127.0.0.1:6379> **sdiff setA setB setC**

1) "1"

2) "2"

127.0.0.1:6379> **sinter setA setB**

1) "3"

2) "4"

3) "5"

127.0.0.1:6379> **sinter setA setB setC**

1) "4"

2) "5"

127.0.0.1:6379> **sadd setC 6 7**

(integer) 2

127.0.0.1:6379> **smembers setC**

1) "4"

2) "5"

3) "6"

4) "7"

127.0.0.1:6379> **sunion setA setC**

1) "1"

2) "2"

3) "3"

4) "4"

5) "5"

6) "6"

7) "7"

### 命令补充

1. 获取集合中元素的个数

Redis Scard 命令返回集合中元素的数量。

语法

redis Scard 命令基本语法如下：

redis 127.0.0.1:6379> SCARD KEY\_NAME

127.0.0.1:6379> **smembers setC**

1) "4"

2) "5"

3) "6"

4) "7"

127.0.0.1:6379> **scard letters**

(integer) 2

127.0.0.1:6379> **scard setC**

(integer) 4

2)随机获取集合中的元素

Redis Srandmember 命令用于返回集合中的一个随机元素。

从 Redis 2.6 版本开始， Srandmember 命令接受可选的 count 参数：

如果 count 为正数，且小于集合基数，那么命令返回一个包含 count 个元素的数组，数组中的元素各不相同。如果 count 大于等于集合基数，那么返回整个集合。

如果 count 为负数，那么命令返回一个数组，数组中的元素可能会重复出现多次，而数组的长度为 count 的绝对值。

该操作和 SPOP 相似，但 SPOP 将随机元素从集合中移除并返回，而 Srandmember 则仅仅返回随机元素，而不对集合进行任何改动。

语法

redis Srandmember 命令基本语法如下：

redis 127.0.0.1:6379> SRANDMEMBER KEY [count]

127.0.0.1:6379> **srandmember setC**

"7"

127.0.0.1:6379> **srandmember setC**

"6"

127.0.0.1:6379> s**randmember setC 2**

1) "7"

2) "5"

127.0.0.1:6379> **srandmember setC 3**

1) "5"

2) "6"

3) "4"

127.0.0.1:6379> **srandmember setC 5**

1) "4"

2) "5"

3) "6"

4) "7"

127.0.0.1:6379> **srandmember setC -3**

1) "7"

2) "6"

3) "4"

127.0.0.1:6379> **srandmember setC -3**

1) "7"

2) "7"

3) "7"

3)从集合中弹出一个元素

Redis Spop 命令用于移除并返回集合中的一个随机元素。

语法

redis Spop 命令基本语法如下：

redis 127.0.0.1:6379> SPOP KEY

127.0.0.1:6379> **smembers setC**

1) "4"

2) "5"

3) "6"

4) "7"

127.0.0.1:6379> **spop setC**

"5"

127.0.0.1:6379> **smembers setC**

1) "4"

2) "6"

3) "7"

## 有序集合(zset)

Redis zset 和 set 一样也是string类型元素的集合,且不允许重复的成员。

不同的是每个元素都会关联一个double类型的分数。redis正是通过分数来为集合中的成员进行从小到大的排序。

zset的成员是唯一的,但分数(score)却可以重复。

s

集合是通过哈希表实现的，所以添加，删除，查找的复杂度都是O(1)。 集合中最大的成员数为 2^32 - 1 (4294967295, 每个集合可存储40多亿个成员)。

### 常用基础命令

1. **增加元素**

Redis Zadd 命令用于将一个或多个成员元素及其分数值加入到有序集当中。

如果某个成员已经是有序集的成员，那么更新这个成员的分数值，并通过重新插入这个成员元素，来保证该成员在正确的位置上。

分数值可以是整数值或双精度浮点数。

如果有序集合 key 不存在，则创建一个空的有序集并执行 ZADD 操作。

当 key 存在但不是有序集类型时，返回一个错误。

注意： 在 Redis 2.4 版本以前， ZADD 每次只能添加一个元素。

语法

redis Zadd 命令基本语法如下：

redis 127.0.0.1:6379> **ZADD KEY\_NAME SCORE1 VALUE1.. SCOREN VALUEN**

127.0.0.1:6379> **zadd scoreboard 88 tom 86 peter 78 smith 新增**

(integer) 3

127.0.0.1:6379> **zadd scoreboard 99 smith 修改**

(integer) 0

127.0.0.1:6379> **zscore scoreboard smith 查看结果**

"99"

**2)获取元素的分数**

127.0.0.1:6379> **zscore scoreboard peter**

"86"

1. 获得排名的某个范围

Redis Zrange 返回有序集中，指定区间内的成员。

其中成员的位置按分数值递增(从小到大)来排序。

具有相同分数值的成员按字典序(lexicographical order )来排列。

如果你需要成员按

值递减(从大到小)来排列，请使用 ZREVRANGE 命令。

下标参数 start 和 stop 都以 0 为底，也就是说，以 0 表示有序集第一个成员，以 1 表示有序集第二个成员，以此类推。

你也可以使用负数下标，以 -1 表示最后一个成员， -2 表示倒数第二个成员，以此类推。

语法

redis Zrange 命令基本语法如下：

redis 127.0.0.1:6379> ZRANGE key start stop [WITHSCORES]

127.0.0.1:6379> **zadd scoreboard 66 xiaoming 85 xiaohong 34 zhangsan**

(integer) 3

127.0.0.1:6379> **zrange scoreboard 0 2**

1) "zhangsan"

2) "xiaoming"

3) "xiaohong"

127.0.0.1:6379> **zrange scoreboard -2 -1**

1) "tom"

2) "smith"

127.0.0.1:6379> **zrange scoreboard 0 -1**

1) "zhangsan"

2) "xiaoming"

3) "xiaohong"

4) "peter"

5) "tom"

6) "smith"

127.0.0.1:6379> **zrange scoreboard 0 -1 WITHSCORES 同时显示对应的分数**

1) "zhangsan"

2) "34"

3) "xiaoming"

4) "66"

5) "xiaohong"

6) "85"

7) "peter"

8) "86"

9) "tom"

10) "88"

11) "smith"

12) "99"0

127.0.0.1:6379**> zadd scoreboard 99 abc 85 qqq**

(integer) 2

127.0.0.1:6379> **zrange scoreboard 0 -1 WITHSCORES 具有相同分数值的成员按字典序(lexicographical order )来排列。**

1) "zhangsan"

2) "34"

3) "xiaoming"

4) "66"

5) "qqq"

6) "85"

7) "xiaohong"

8) "85"

9) "peter"

10) "86"

11) "tom"

12) "88"

13) "abc"

14) "99"

15) "smith"

16) "99"

4）获取指定分数范围的元素 80-100之间的

Redis Zrangebyscore 返回有序集合中指定分数区间的成员列表。有序集成员按分数值递增(从小到大)次序排列。

具有相同分数值的成员按字典序来排列(该属性是有序集提供的，不需要额外的计算)。

默认情况下，区间的取值使用闭区间 (小于等于或大于等于)，你也可以通过给参数前增加 ( 符号来使用可选的开区间 (小于或大于)。

举个例子：

ZRANGEBYSCORE zset (1 5

语法

redis Zrangebyscore 命令基本语法如下：

redis 127.0.0.1:6379> **ZRANGEBYSCORE key min max [WITHSCORES] [LIMIT offset count]**

**127.0.0.1:6379> zrangebyscore scoreboard 85 100 获取指定范围的数**

**1) "qqq"**

**2) "xiaohong"**

**3) "peter"**

**4) "tom"**

**5) "abc"**

**6) "smith"**

**127.0.0.1:6379> zrangebyscore scoreboard (85 100 不包含左侧的数据**

**1) "peter"**

**2) "tom"**

**3) "abc"**

**4) "smith"**

**#-inf 负无穷 +inf正无穷**

127.0.0.1:6379> **zrangebyscore scoreboard (85 +inf**

1) "peter"

2) "tom"

3) "abc"

4) "smith"

#获取大于85的所有人的姓名和成绩

127.0.0.1:6379> **zrangebyscore scoreboard (85 +inf withscores**

1) "peter"

2) "86"

3) "tom"

4) "88"

5) "abc"

6) "99"

7) "smith"

8) "99"

# 获取成绩大于60的三个人 高于60的第二个人开始

127.0.0.1:6379> **zadd scoreboard 58 wer 59 piou 60 tyui 61 guigu123 65 kingqwe**

(integer) 5

127.0.0.1:6379> **zrange scoreboard 0 -1 WITHSCORES**

1) "zhangsan"

2) "34"

3) "wer"

4) "58"

5) "piou"

6) "59"

7) "tyui"

8) "60"

9) "guigu123"

10) "61"

11) "kingqwe"

12) "65"

13) "xiaoming"

14) "66"

15) "qqq"

16) "85"

17) "xiaohong"

18) "85"

19) "peter"

20) "86"

21) "tom"

22) "88"

23) "abc"

24) "99"

25) "smith"

26) "99"

127.0.0.1:6379> **zrangebyscore scoreboard 60 +inf limit 1 3**

1) "guigu123"

2) "kingqwe"

3) "xiaoming"

**5）Zrevrangebyscore**

Redis Zrevrangebyscore 返回有序集中指定分数区间内的所有的成员。有序集成员按分数值递减(从大到小)的次序排列。

具有相同分数值的成员按字典序的逆序(reverse lexicographical order )排列。

除了成员按分数值递减的次序排列这一点外， ZREVRANGEBYSCORE 命令的其他方面和 ZRANGEBYSCORE 命令一样。

语法

redis Zrevrangebyscore 命令基本语法如下：

redis 127.0.0.1:6379> ZREVRANGEBYSCORE key max min [WITHSCORES] [LIMIT offset count]

#获取分数小于等于100 的前三个人

127.0.0.1:6379> **zrevrangebyscore scoreboard 100 0 limit 0 3**

1) "smith"

2) "abc"

3) "tom"

1. 增加某个元素的分数

Redis Zincrby 命令对有序集合中指定成员的分数加上增量 increment

可以通过传递一个负数值 increment ，让分数减去相应的值，比如 ZINCRBY key -5 member ，就是让 member 的 score 值减去 5 。

当 key 不存在，或分数不是 key 的成员时， ZINCRBY key increment member 等同于 ZADD key increment member 。

当 key 不是有序集类型时，返回一个错误。

分数值可以是整数值或双精度浮点数。

语法

redis Zincrby 命令基本语法如下：

redis 127.0.0.1:6379> **ZINCRBY key increment member**

127.0.0.1:6379> **zincrby scoreboard 5 piou**

"64"

127.0.0.1:6379> **zincrby scoreboard -7 piou**

"57"

### 命令补充

1. **获取所有的数量**

Redis Zcard 命令用于计算集合中元素的数量。

语法

redis Zcard 命令基本语法如下：

redis 127.0.0.1:6379> ZCARD KEY\_NAME

127.0.0.1:6379> **zcard scoreboard**

(integer) 13

2）获取总数

Redis Zcount 命令用于计算有序集合中指定分数区间的成员数量。

语法

redis Zcount 命令基本语法如下：

redis 127.0.0.1:6379> **ZCOUNT key min max**

127.0.0.1:6379> **zcount scoreboard -inf +inf**

(integer) 13

127.0.0.1:6379> **zcount scoreboard 60 100**

(integer) 10

3）删除一个元素

Redis Zrem 命令用于移除有序集中的一个或多个成员，不存在的成员将被忽略。

当 key 存在但不是有序集类型时，返回一个错误。

注意： 在 Redis 2.4 版本以前， ZREM 每次只能删除一个元素。

语法

redis Zrem 命令基本语法如下：

redis 127.0.0.1:6379> ZREM key member [member ...]

127.0.0.1:6379> **zrem scoreboard smith**

(integer) 1

127.0.0.1:6379> **zrange scoreboard 0 -1 WITHSCORES**

1. 按照排名范围删除元素

Redis Zremrangebyrank 命令用于移除有序集中，指定排名(rank)区间内的所有成员。

语法

redis Zremrangebyrank 命令基本语法如下：

redis 127.0.0.1:6379> ZREMRANGEBYRANK key start stop

127.0.0.1:6379> **zadd testRem 1 a 2 b 3 c 4 d 5 e 6 f**

(integer) 6

127.0.0.1:6379**> zrange testRem 0 -1**

1) "a"

2) "b"

3) "c"

4) "d"

5) "e"

6) "f"

127.0.0.1:6379> **zremrangebyrank testRem 0 3**

(integer) 4

127.0.0.1:6379> **zrange testRem 0 -1**

1) "e"

2) "f"

Redis Zrevrangebyscore 返回有序集中指定分数区间内的所有的成员。有序集成员按分数值递减(从大到小)的次序排列。

具有相同分数值的成员按字典序的逆序(reverse lexicographical order )排列。

除了成员按分数值递减的次序排列这一点外， ZREVRANGEBYSCORE 命令的其他方面和 ZRANGEBYSCORE 命令一样。

语法

redis Zrevrangebyscore 命令基本语法如下：

redis 127.0.0.1:6379> **ZREVRANGEBYSCORE key max min [WITHSCORES] [LIMIT offset count]**

1. 获取元素排名

Redis Zrank 返回有序集中指定成员的排名。其中有序集成员按分数值递增(从小到大)顺序排列。

语法

redis Zrank 命令基本语法如下：

redis 127.0.0.1:6379> **ZRANK key member**

127.0.0.1:6379> **zrank scoreboard abc**

(integer) 11

127.0.0.1:6379> **zrange scoreboard 0 -1**

1) "zhangsan"

2) "piou"

3) "wer"

4) "tyui"

5) "guigu123"

6) "kingqwe"

7) "xiaoming"

8) "qqq"

9) "xiaohong"

10) "peter"

11) "tom"

12) "abc"

127.0.0.1:6379> **zrank scoreboard qqq**

(integer) 7

127.0.0.1:6379> **zrevrank scoreboard abc**

(integer) 0

127.0.0.1:6379> **zrevrank scoreboard qqq**

(integer) 4

127.0.0.1:6379>

# Redis事务

## Redis事务概述

Redis 事务可以一次执行多个命令， 并且带有以下两个重要的保证：

批量操作在发送 EXEC 命令前被放入队列缓存。

收到 EXEC 命令后进入事务执行，事务中任意命令执行失败，其余的命令依然被执行。

在事务执行过程，其他客户端提交的命令请求不会插入到事务执行命令序列中。

一个事务从开始到执行会经历以下三个阶段：

开始事务。

命令入队。

执行事务。

## Redis事务的执行

127.0.0.1:6379> **multi**

OK

127.0.0.1:6379> set book-name "redis入门高级到"

QUEUED

127.0.0.1:6379> get book-name

QUEUED

127.0.0.1:6379> sadd tag "java" "java-web" "ssh" "abc"

QUEUED

127.0.0.1:6379> smembers tag

QUEUED

127.0.0.1:6379> **exec**

1) OK

2) "redis\xe5\x85\xa5\xe9\x97\xa8\xe9\xab\x98\xe7\xba\xa7\xe5\x88\xb0"

3) (integer) 4

4) 1) "java-web"

2) "ssh"

3) "java"

4) "abc"

## Redis错误处理

127.0.0.1:6379> **multi**

OK

127.0.0.1:6379> **set color red**

QUEUED

127.0.0.1:6379> **abcds**

(error) ERR unknown command 'abcds'

127.0.0.1:6379> **exec**

(error) EXECABORT Transaction discarded because of previous errors.

127.0.0.1:6379> **get color**

(nil)

127.0.0.1:6379> **multi**

OK

127.0.0.1:6379> **set keyy 1**

QUEUED

127.0.0.1:6379> **sadd keyy 2**

QUEUED

127.0.0.1:6379> **set keyy 3**

QUEUED

127.0.0.1:6379> **exec**

1) OK

2) (error) WRONGTYPE Operation against a key holding the wrong kind of value

3) OK

127.0.0.1:6379> get keyy

"3"

## watch命令

Watch命令是可以监控一个或者多个键 ，一旦其中一个键被修改或者删除，之后的事务就不会执行，监控会一直持续到exec命令 。

127.0.0.1:6379> s**et keya 1**

OK

127.0.0.1:6379> **watch keya 监控值keya**

OK

127.0.0.1:6379> **set keya 1234**

OK

127.0.0.1:6379> **multi 开启事务**

OK

127.0.0.1:6379> **set keya 3456 修改值**

QUEUED

127.0.0.1:6379> **exec**

(nil)

127.0.0.1:6379> **get keya 值并未变化**

"1234"

127.0.0.1:6379> **unwatch 取消监控**

OK

127.0.0.1:6379> multi 开启事务

OK

127.0.0.1:6379> set keya 3456

QUEUED

127.0.0.1:6379> exec

1) OK

127.0.0.1:6379> get keya

"3456"

## 生存时间

### 概述和命令介绍

比如需要对某个键在指定的日期后失效 ， 系统redis会自动检查。到期自动删除。

127.0.0.1:6379> **set session:ug34234asd uid1314 设置key value**

OK

127.0.0.1:6379> **get session:ug34234asd 得到值**

"uid1314"

127.0.0.1:6379> **expire session:ug34234asd 5 设置过期时间**

(integer) 1

127.0.0.1:6379> **get session:ug34234asd**

(nil)

127.0.0.1:6379> **set session:ug34234asd uid1314**

OK

127.0.0.1:6379> **del session:ug34234asd**

(integer) 1

127.0.0.1:6379> **get session:ug34234asd**

(nil)

127.0.0.1:6379> set session:ug34234asd uid1314

OK

127.0.0.1:6379> expire session:ug34234asd 20

(integer) 1

127.0.0.1:6379> **ttl session:ug34234asd 查看指定的key值过期剩余时间**

(integer) 13

-2代表不存在 过期

-1 代表永不过期

127.0.0.1:6379> set session:qwe321 helloredis

OK

127.0.0.1:6379> ttl session:qwe321

(integer) -1

可以重新指定key值的过期时间

127.0.0.1:6379> **set session:56tyer qwertt**

OK

127.0.0.1:6379> **expire session:56tyer 30**

(integer) 1

127.0.0.1:6379> ttl session:56tyer

(integer) 26

127.0.0.1:6379> **ttl session:56tyer**

(integer) 23

127.0.0.1:6379> **expire session:56tyer 50**

(integer) 1

127.0.0.1:6379> **ttl session:56tyer**

(integer) 48

127.0.0.1:6379> **ttl session:56tyer**

(integer) 46

# Redis脚本和管理

## Redis服务器相关的命令

**Redis 默认 16个数据库**

1）查看数据库key值的数量

**127.0.0.1:6379> dbsize**

**(integer) 31**

1. 查看所有的服务器配置信息

127.0.0.1:6379> **config get \***

127.0.0.1:6379> **config get dir**

1) "dir"

2) "/usr/local/sofeware/redis-stable"

3）详细查看服务器的信息

127.0.0.1:6379**> info**

# Server

redis\_version:4.0.8

redis\_git\_sha1:00000000

redis\_git\_dirty:0

redis\_build\_id:81c1f5cd200b54d3

redis\_mode:standalone

os:Linux 3.10.0-693.el7.x86\_64 x86\_64

arch\_bits:64

multiplexing\_api:epoll

atomicvar\_api:atomic-builtin

gcc\_version:4.8.5

process\_id:40368

run\_id:a3afe706198f4bd6d1b25a5a26c9f7aadd1bd902

tcp\_port:6379

uptime\_in\_seconds:85572

uptime\_in\_days:0

hz:10

lru\_clock:10280163

executable:/usr/local/sofeware/redis-stable/src/redis-server

config\_file:/usr/local/sofeware/redis-stable/redis.conf

# Clients

connected\_clients:1

client\_longest\_output\_list:0

client\_biggest\_input\_buf:0

blocked\_clients:0

# Memory

used\_memory:833040

used\_memory\_human:813.52K

used\_memory\_rss:2555904

used\_memory\_rss\_human:2.44M

used\_memory\_peak:848960

used\_memory\_peak\_human:829.06K

used\_memory\_peak\_perc:98.12%

used\_memory\_overhead:816918

used\_memory\_startup:765760

used\_memory\_dataset:16122

used\_memory\_dataset\_perc:23.96%

total\_system\_memory:1911857152

total\_system\_memory\_human:1.78G

used\_memory\_lua:37888

used\_memory\_lua\_human:37.00K

maxmemory:0

maxmemory\_human:0B

maxmemory\_policy:noeviction

mem\_fragmentation\_ratio:3.07

mem\_allocator:jemalloc-4.0.3

active\_defrag\_running:0

lazyfree\_pending\_objects:0

# Persistence

loading:0

rdb\_changes\_since\_last\_save:0

rdb\_bgsave\_in\_progress:0

rdb\_last\_save\_time:1520229220

rdb\_last\_bgsave\_status:ok

rdb\_last\_bgsave\_time\_sec:0

rdb\_current\_bgsave\_time\_sec:-1

rdb\_last\_cow\_size:389120

aof\_enabled:0

aof\_rewrite\_in\_progress:0

aof\_rewrite\_scheduled:0

aof\_last\_rewrite\_time\_sec:-1

aof\_current\_rewrite\_time\_sec:-1

aof\_last\_bgrewrite\_status:ok

aof\_last\_write\_status:ok

aof\_last\_cow\_size:0

# Stats

total\_connections\_received:9

total\_commands\_processed:264

instantaneous\_ops\_per\_sec:0

total\_net\_input\_bytes:11051

total\_net\_output\_bytes:11159

instantaneous\_input\_kbps:0.00

instantaneous\_output\_kbps:0.00

rejected\_connections:0

sync\_full:0

sync\_partial\_ok:0

sync\_partial\_err:0

expired\_keys:3

evicted\_keys:0

keyspace\_hits:146

keyspace\_misses:9

pubsub\_channels:0

pubsub\_patterns:0

latest\_fork\_usec:584

migrate\_cached\_sockets:0

slave\_expires\_tracked\_keys:0

active\_defrag\_hits:0

active\_defrag\_misses:0

active\_defrag\_key\_hits:0

active\_defrag\_key\_misses:0

# Replication

role:master

connected\_slaves:0

master\_replid:7590c914883c89b532326c56af8ee78eb777f2c6

master\_replid2:0000000000000000000000000000000000000000

master\_repl\_offset:0

second\_repl\_offset:-1

repl\_backlog\_active:0

repl\_backlog\_size:1048576

repl\_backlog\_first\_byte\_offset:0

repl\_backlog\_histlen:0

# CPU

used\_cpu\_sys:42.76

used\_cpu\_user:22.27

used\_cpu\_sys\_children:0.18

used\_cpu\_user\_children:0.05

# Cluster

cluster\_enabled:0

# Keyspace

db0:keys=31,expires=0,avg\_ttl=0

4）情况当前数据库所有数据

Redis Flushdb 命令用于清空当前数据库中的所有 key。

语法

redis Flushdb 命令基本语法如下：

redis 127.0.0.1:6379> FLUSHDB

127.0.0.1:6379> **flushdb**

OK

127.0.0.1:6379> keys \*

(empty list or set)

5）清空所有服务器信息

Redis Flushall 命令用于清空整个 Redis 服务器的数据(删除所有数据库的所有 key )。

语法

redis Flushall 命令基本语法如下：

redis 127.0.0.1:6379> FLUSHALL

## redis.conf配置文件详解

redis.conf 配置项说明如下：

1. Redis默认不是以守护进程的方式运行，可以通过该配置项修改，使用yes启用守护进程

**daemonize no**

2. 当Redis以守护进程方式运行时，Redis默认会把pid写入/var/run/redis.pid文件，可以通过pidfile指定

**pidfile /var/run/redis.pid**

3. 指定Redis监听端口，默认端口为6379，作者在自己的一篇博文中解释了为什么选用6379作为默认端口，因为6379在手机按键上MERZ对应的号码，而MERZ取自意大利歌女Alessia Merz的名字

**port 6379**

4. 绑定的主机地址

**bind 127.0.0.1**

5.当 客户端闲置多长时间后关闭连接，如果指定为0，表示关闭该功能

**timeout 300**

6. 指定日志记录级别，Redis总共支持四个级别：debug、verbose、notice、warning，默认为verbose

**loglevel verbose**

7. 日志记录方式，默认为标准输出，如果配置Redis为守护进程方式运行，而这里又配置为日志记录方式为标准输出，则日志将会发送给/dev/null

**logfile stdout**

8. 设置数据库的数量，默认数据库为0，可以使用SELECT <dbid>命令在连接上指定数据库id

**databases 16**

9. 指定在多长时间内，有多少次更新操作，就将数据同步到数据文件，可以多个条件配合

**save <seconds> <changes>**

    Redis默认配置文件中提供了三个条件：

**save 900 1**

**save 300 10**

**save 60 10000**

    分别表示900秒（15分钟）内有1个更改，300秒（5分钟）内有10个更改以及60秒内有10000个更改。

10. 指定存储至本地数据库时是否压缩数据，默认为yes，Redis采用LZF压缩，如果为了节省CPU时间，可以关闭该选项，但会导致数据库文件变的巨大

**rdbcompression yes**

11. 指定本地数据库文件名，默认值为dump.rdb

**dbfilename dump.rdb**

12. 指定本地数据库存放目录

**dir ./**

**主从复制使用的配置**

13. 设置当本机为slav服务时，设置master服务的IP地址及端口，在Redis启动时，它会自动从master进行数据同步

**slaveof <masterip> <masterport>**

14. 当master服务设置了密码保护时，slav服务连接master的密码

**masterauth <master-password>**

15. 设置Redis连接密码，如果配置了连接密码，客户端在连接Redis时需要通过AUTH <password>命令提供密码，默认关闭

**requirepass foobared**

16. 设置同一时间最大客户端连接数，默认无限制，Redis可以同时打开的客户端连接数为Redis进程可以打开的最大文件描述符数，如果设置 **maxclients 0，表示不作限制**。当客户端连接数到达限制时，Redis会关闭新的连接并向客户端返回max number of clients reached错误信息

**maxclients 128**

17. 指定Redis最大内存限制，Redis在启动时会把数据加载到内存中，达到最大内存后，Redis会先尝试清除已到期或即将到期的Key，当此方法处理 后，仍然到达最大内存设置，将无法再进行写入操作，但仍然可以进行读取操作。Redis新的vm机制，会把Key存放内存，Value会存放在swap区

**maxmemory <bytes>**

18. 指定是否在每次更新操作后进行日志记录，Redis在默认情况下是异步的把数据写入磁盘，如果不开启，可能会在断电时导致一段时间内的数据丢失。因为 redis本身同步数据文件是按上面save条件来同步的，所以有的数据会在一段时间内只存在于内存中。默认为no

**appendonly no**

19. 指定更新日志文件名，默认为appendonly.aof

**appendfilename appendonly.aof**

20. 指定更新日志条件，共有3个可选值：   
    **no**：表示等操作系统进行数据缓存同步到磁盘（快）   
    **always**：表示每次更新操作后手动调用fsync()将数据写到磁盘（慢，安全）   
    **everysec**：表示每秒同步一次（折衷，默认值）

**appendfsync everysec**

21. 指定是否启用虚拟内存机制，默认值为no，简单的介绍一下，VM机制将数据分页存放，由Redis将访问量较少的页即冷数据swap到磁盘上，访问多的页面由磁盘自动换出到内存中（在后面的文章我会仔细分析Redis的VM机制）

**vm-enabled no**

22. 虚拟内存文件路径，默认值为/tmp/redis.swap，不可多个Redis实例共享

**vm-swap-file /tmp/redis.swap**

23. 将所有大于vm-max-memory的数据存入虚拟内存,无论vm-max-memory设置多小,所有索引数据都是内存存储的(Redis的索引数据 就是keys),也就是说,当vm-max-memory设置为0的时候,其实是所有value都存在于磁盘。默认值为0

**vm-max-memory 0**

24. Redis swap文件分成了很多的page，一个对象可以保存在多个page上面，但一个page上不能被多个对象共享，vm-page-size是要根据存储的 数据大小来设定的，作者建议如果存储很多小对象，page大小最好设置为32或者64bytes；如果存储很大大对象，则可以使用更大的page，如果不 确定，就使用默认值

**vm-page-size 32**

25. 设置swap文件中的page数量，由于页表（一种表示页面空闲或使用的bitmap）是在放在内存中的，，在磁盘上每8个pages将消耗1byte的内存。

**vm-pages 134217728**

26. 设置访问swap文件的线程数,最好不要超过机器的核数,如果设置为0,那么所有对swap文件的操作都是串行的，可能会造成比较长时间的延迟。默认值为4

**vm-max-threads 4**

27. 设置在向客户端应答时，是否把较小的包合并为一个包发送，默认为开启

**glueoutputbuf yes**

28. 指定在超过一定的数量或者最大的元素超过某一临界值时，采用一种特殊的哈希算法

**hash-max-zipmap-entries 64**

**hash-max-zipmap-value 512**

29. 指定是否激活重置哈希，默认为开启（后面在介绍Redis的哈希算法时具体介绍）

**activerehashing yes**

30. 指定包含其它的配置文件，可以在同一主机上多个Redis实例之间使用同一份配置文件，而同时各个实例又拥有自己的特定配置文件

**include /path/to/local.conf**

## Redis安全

我们可以通过 redis 的配置文件设置密码参数，这样客户端连接到 redis 服务就需要密码验证，这样可以让你的 redis 服务更安全。

实例

我们可以通过以下命令查看是否设置了密码验证：

127.0.0.1:6379> CONFIG get requirepass

1) "requirepass"

2) ""

要求密码必须安全 不能是简单的数字或者字母 放置暴力破解

## Redis主从复制

一般来说，要将Redis运用于工程项目中，只使用一台Redis是万万不能的，原因如下：

从结构上，单个Redis服务器会发生单点故障，并且一台服务器需要处理所有的请求负载，压力较大；

从容量上，单个Redis服务器内存容量有限，就算一台Redis服务器内容容量为256G，也不能将所有内容用作Redis存储内存，一般来说，单台Redis最大使用内存不应该超过20G。

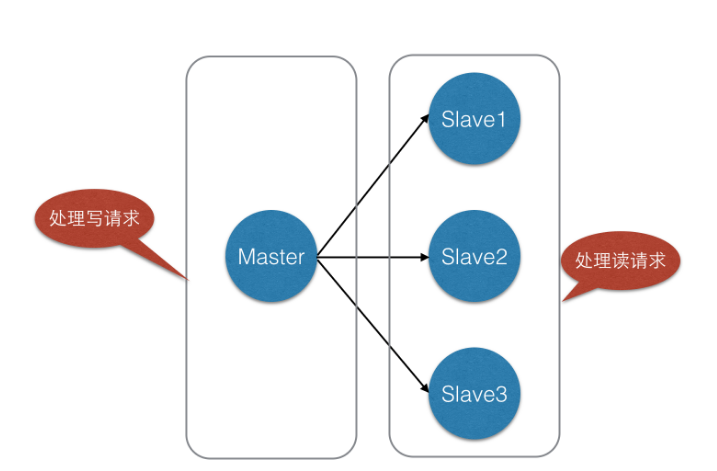
本文先讨论第一点的解决方案：Redis主从复制，第二点可以使用Redis集群解决，下一篇文章将介绍Redis集群。

主从复制

考虑如下一种场景：

电子商务网站上的商品，一般都是一次上传，无数次浏览的，说专业点也就是”多读少写”。

对于这种场景，我们可以使如下这种架构：



如图中所示，我们将一台Redis服务器作主库(Matser)，其他三台作为从库(Slave)，主库只负责写数据，每次有数据更新都将更新的数据同步到它所有的从库，而从库只负责读数据。这样一来，就有了两个好处：

读写分离，不仅可以提高服务器的负载能力，并且可以根据读请求的规模自由增加或者减少从库的数量，棒极了；

数据被复制成了了好几份，就算有一台机器出现故障，也可以使用其他机器的数据快速恢复。

需要注意的是：在Redis主从模式中，一台主库可以拥有多个从库，但是一个从库只能隶属于一个主库。

### 配置

保证三台服务都有redis服务

使用一台master 192.168.118.6

两台从服务器slave

192.168.118.9

192.168..118.10

1. **检查环境**

检查是否安装gcc

关闭防火墙

[root@J2001 sofeware]# **yum –y install gcc**

[root@J2001 sofeware]# **systemctl stop firewalld**

[root@J2001 sofeware]# **systemctl disable firewalld**

**2)每个服务器都需要安装redis**

可以单独每个服务再次安装redis

把主服务器上的redis直接远程拷贝到其他服务器上。

[root@J2001 sofeware]# **scp -r redis-stable/ 192.168.118.10:/usr/local/sofeware**

[root@J2001 sofeware]# **scp -r redis-stable/ 192.168.118.9:/usr/local/sofeware**

1. **主从配置**

在从服务器192.168.118.9上修改如下内容

# salveof 主服务器ip地址 主服务器端口号

**slaveof 192.168.118.6 6379**

#配置权限 主服务器的连接密码

**masterauth guigu**

在从服务器192.168.118.10上修改如下内容

# salveof 主服务器ip地址 主服务器端口号

**slaveof 192.168.118.6 6379**

#配置权限 主服务器的连接密码

**masterauth guigu**

在主服务器上配置一个密码即可

#配置权限 主服务器的连接密码

**masterauth guigu**

4）启动服务器并且查看是否配置正确

[root@J2001 redis-stable]# **src/redis-server redis.conf**

6741:C 03 Mar 14:41:12.701 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo

6741:C 03 Mar 14:41:12.701 # Redis version=4.0.8, bits=64, commit=00000000, modified=0, pid=6741, just started

6741:C 03 Mar 14:41:12.701 # Configuration loaded

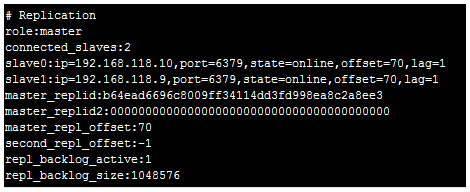
[root@J2001 redis-stable]# clear

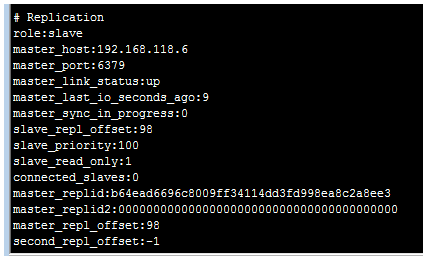
[root@J2001 redis-stable]# **ps -ef | grep 6379**

root 6742 1 0 14:41 ? 00:00:00 src/redis-server \*:6379

root 6756 6446 0 14:41 pts/1 00:00:00 grep --color=auto 6379

[root@J2001 redis-stable]# src/redis-cli





192.168.118.6:6379> keys \*

(empty list or set)

192.168.118.6:6379> set username guigu

OK

127.0.0.1:6379> set age 14

OK

主数据库自动把数据同步到从数据库

192.168.118.9:6379> keys \*

(empty list or set)

192.168.118.9:6379> get username

"guigu"

从服务器只能读不能写

127.0.0.1:6379> set msg hello

(error) READONLY You can't write against a read only slave.

### 哨兵监测

有了主从复制的实现 后，如果需要对主服务器进行监控，那么在从服务器上提供一个哨兵监测即可。

哨兵监测是指监控redis的运行状态，可以启动多个哨兵监测。

监控redis的运行状态 其主要有两点：

1.监控主数据库和从数据库是否正常运行

2.主数据库出现故障 可以自动将从数据库转换为主数据库 实现自动切换ss

内部选举机制

1. 配置哨兵监测

在任意一台从服务器上进行相关的配置

# master的ip地址 端口号 选举投票数

sentinel monitor mymaster 192.168.118.6 6379 1

sentinel auth-pass mymaster guigu

[root@J2001 redis-stable]# **src/redis-server sentinel.conf --sentinel & 启动哨兵监控**

[root@J2001 redis-stable]# ps -ef | grep 26379 查看哨兵是否启动

1. 测试

让主服务器挂掉 然后查看从服务器相关的信

## Redis持久化

Redis的强大程序在于数据的处理，由于所有的数据都是存储在内存中，当重启服务器内存中的数据如果没有固化将会丢失。 在进行重启或者其他操作需要把数据固化到硬盘中。这一过程就是redis的持久化。

Redis支持两种持久化的方式

* RDB的方式(snapshotting快照)方式 也是redis默认的支持方式
* AOF的方式

### RDB(snapshotting)

这种持久化的方式是redis默认的支持方式，将内存中的数据以快照的形式写入到二进制文件中。默认文件名是dump.rdb .

Rdb默认方式的配置文件如下:

save 900 1 # 900秒内有超过1个key被修改 则发起快照

save 300 10 #300秒内 超过10个key被修改则发起快照

save 60 10000 # 60秒内 超过10000个key被修改则发起快照

### AOF方式

Append-Olny\_File 将操作+数据以格式化指令进行追加

AOF能最大程度的保证数据的完整性

AOF相关的配置文件

默认AOF的关闭的 需要打开AOF文件

# 开启AOF持久化的方式

appendonly yes

#aof持久化数据的文件名

appendfilename "appendonly.aof"

#配置AOF文件的同步策略

# appendfsync **always # 收到任何命令立即写入到磁盘中 效率是最慢的 但是数据是最完整的**

appendfsync **everysec # 每秒写入磁盘一次 在性能和持久化方面做到平衡**

# appendfsync **no # 完全依赖操作系统 性能最好 但是持久化没有保证**

#在aof写期间 appendfsync 是否暂缓文件同步 no表示不暂缓 yes表示暂缓

no-appendfsync-on-rewrite no

# aof配置文件的大写 默认是64mb 建议512mb

auto-aof-rewrite-percentage 100

auto-aof-rewrite-min-size 64mb

AOF持久化方式开启那么RDB方式自动关闭

[root@J2001 redis-stable]# **vim appendonly.aof**

[root@J2001 redis-stable]# **rm -rf appendonly.aof**

**[root@J2001 redis-stable]# src/redis-server redis.conf**

[root@J2001 redis-stable]# src/redis-cli

[root@J2001 redis-stable]# **src/redis-cli**

127.0.0.1:6379> auth guigu

OK

127.0.0.1:6379> **keys \***

(empty list or set)

127.0.0.1:6379> set username guigu

OK

127.0.0.1:6379> **set age 14**

OK

127.0.0.1:6379> quit

[root@J2001 redis-stable]# **vim appendonly.aof**

实际生产环境中 一般是两种方式结合使用 。

Master通常使用AOF方式 保证数据的完整性

Slave使用snapshot方式

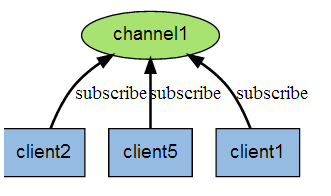
主要原因master保证数据完整性 它作为第一备份选择 salve仅仅提供只读服务 主要是快速响应客户端read请求

## Redis发布与订阅消息

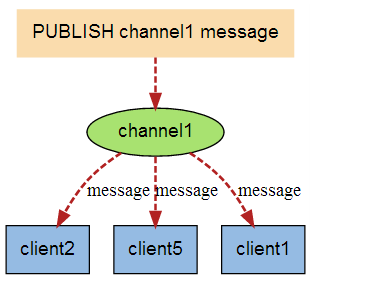
Redis 发布订阅(pub/sub)是一种消息通信模式：发送者(pub)发送消息，订阅者(sub)接收消息。

Redis 客户端可以订阅任意数量的频道。

下图展示了频道 channel1 ， 以及订阅这个频道的三个客户端 —— client2 、 client5 和 client1 之间的关系：



当有新消息通过 PUBLISH 命令发送给频道 channel1 时， 这个消息就会被发送给订阅它的三个客户端：



开启多个客户端 连接到同一个服务器

[root@J2001 redis-stable]# src/redis-cli

127.0.0.1:6379> auth guigu

OK

127.0.0.1:6379> SUBSCRIBE cctv

Reading messages... (press Ctrl-C to quit)

1) "subscribe"

2) "cctv"

[root@J2001 redis-stable]# src/redis-cli

127.0.0.1:6379> auth guigu

OK

127.0.0.1:6379> SUBSCRIBE cctv

Reading messages... (press Ctrl-C to quit)

1) "subscribe"

2) "cctv"

3) (integer) 1

1) "message"

2) "cctv"

3) "weidadezhuiguo"

[root@J2001 redis-stable]# src/redis-cli

127.0.0.1:6379> auth guigu

OK

127.0.0.1:6379> PUBLISH cctv "weidadezhuiguo"

(integer) 3

# Redis与java

## 简单操作数据类型

**工具类redis连接池**

**package** com.guigu.linux.utils;

/\*\*

\* <p>project\_name:JavaRedis</p>

\* <p>package\_name:com.guigu.linux.utils.RedisUtils</p>

\* <p>description：</p>

\* <p>@author：刘老师<p>

\* <p> date:2018年3月6日上午10:05:50 </p>

\* <p>comments： </p>

\* <p>@version jdk1.8</p>

\*

\* <p>Copyright (c) 2018, 980991634@qq.com All Rights Reserved. </p>

\*/

**import** redis.clients.jedis.Jedis;

**import** redis.clients.jedis.JedisPool;

**import** redis.clients.jedis.JedisPoolConfig;

**public** **class** RedisUtils {

//定义redis服务器ip 读取配置文件

**private** **static** String *ADDR\_IP* ="192.168.118.6";

//定义Redis端口号

**private** **static** **int** *PORT* =6379;

//访问redis的密码

**private** **static** String *AUTH*="guigu";

//控制一个pool最多有多个状态idle(空闲的)的jedis实例

**private** **static** **int** *MAX\_IDLE* =200;

**private** **static** **int** *TIMEOUT* =10000;

//在使用一个jedis实例之前是否进行相关的验证

**private** **static** **boolean** *TEST\_ON\_BORROW*=**true**;

//redis连接池

**private** **static** JedisPool *jedisPool*=**null**;

/\*\*

\* 初始化redis连接池

\*/

**static** {

**try** {

JedisPoolConfig config =**new** JedisPoolConfig();

config.setMaxIdle(*MAX\_IDLE*);

config.setTestOnBorrow(*TEST\_ON\_BORROW*);

*jedisPool*=**new** JedisPool(config, *ADDR\_IP*,*PORT*,*TIMEOUT*,*AUTH*);

} **catch** (Exception e) {

e.printStackTrace();

}

}

/\*\*

\* 获取redis实例

\*/

**public** **synchronized** **static** Jedis getJedis() {

**if**(*jedisPool*!=**null**) {

Jedis resouce =*jedisPool*.getResource();

**return** resouce;

}**else** {

**return** **null**;

}

}

/\*\*

\* 释放jedis实例

\*/

**public** **static** **void** returnResourece(Jedis jedis) {

**if**(jedis!=**null**) {

*jedisPool*.close();

}

}

}