

# Redis入门记

# 1.1 什么是Redis



远程字典服务器，**R**emote **d**ictionary **s**erver

一个开源的基于内存的数据库，常用作键值存储、缓存和消息队列等

\*什么是数据库见《一万小时计划11》

\*按db-engines的排名，redis是连续几年最受欢迎的键值存储数据库

Redis通常将全部数据存储在内存中，也可以不时的将数据写入硬盘实现持久化，但仅用于重新启动后将数据加载回内存

\*内存的速度比硬盘快一个数量级

## 1.1.1 发展简史



Salvatore Sanfilippo，意大利程序员

他最早使用传统数据库做了一个实时的 Web 日志分析器

因为对其性能不够满意，开发了 Redis

2020 年 6 月，Salvatore Sanfilippo 辞去了 Redis 维护者的职务

## 1.2 安装

### 1.2.1 Windows 10 16237+

开启WSL功能

\*WSL, **W**indows **S**ubsystem for **L**inux, WSL 是一个用于在 Windows 上运行 Linux 可执行文件的兼容层; 但也存在基于 Hyper-V 的 WSL 虚拟机

安装Ubuntu WSL版

\*Ubuntu是基于Debian的Linux发行版, 有机会详讲

<https://www.microsoft.com/p/ubuntu/9nblggh4msv6>

使用apt安装Redis

`sudo apt update`

`sudo apt install redis-server`

\*apt是Debian的软件包管理器

启用Redis

`sudo service redis-server start`

### 1.2.2 老版本Windows

不推荐! 微软官方提供的windows版redis, 只有 Redis 3, 且上次更新是2016

<https://github.com/microsoftarchive/redis>

### 1.2.3 MacOS

安装Homebrew

<https://brew.sh>

\*Homebrew是一款开源的MacOS的软件包管理器

使用Homebrew安装Redis

`brew install redis`

启用Redis

`brew services start redis`

### 1.2.4 Linux

Ubuntu/Debian使用apt安装Redis

`sudo apt update`

```
sudo apt install redis-server
```

CentOS/Red Hat Enterprise Linux使用yum安装Redis

```
sudo yum update
```

\*yum是也是一个软件包管理器

```
sudo yum install redis
```

启用Redis

```
sudo systemctl start redis
```

配置Redis自启动(可选)

```
sudo systemctl enable redis
```

喜欢动手的同学也可以在 `/etc/redis` 或 `/etc` 中找到 `redis.conf`  
编辑 `redis.conf`，将 `supervised` 的值改为 `systemd` \*默认为no

### 1.2.5 通用操作

前台启动redis服务器: `redis-server`

手动启动redis客户端: `redis-cli`

## 2.1 基本操作

官方文档: <https://redis.io/documentation>

### 2.1.1 数据库操作

\*redis默认有16个数据库, 编号为0~15, 且默认访问0号数据库

|                           |              |
|---------------------------|--------------|
| <code>select</code> 数据库编号 | 选择指定数据库      |
| <code>dbsize</code>       | 获取当前数据库键值对数量 |
| <code>flushdb</code>      | 清空当前数据库      |
| <code>flushall</code>     | 清空所有数据库      |

|                       |   |
|-----------------------|---|
| <code>save</code>     | 将数据保存至磁盘                                    |
| <code>bgsave</code>   | 将数据异步保存至磁盘 (后台: Background)<br>*默认每两秒自动执行一次 |
| <code>lastsave</code> | 获取最后一次成功保存的unix秒                            |

### 2.1.2 通用数据操作

\*操作范围为当前数据库

|                                     |                     |
|-------------------------------------|---------------------|
| <code>keys</code> 格式                | 查看符合指定格式的key, *为通配符 |
| <code>exists</code> key1 [key2 ...] | 查看是否存在一至多个指定的key    |
| <code>type</code> key               | 按key查看value的数据类型    |
| <code>del</code> key1 [key2 ...]    | 按key删除一至多个键值对       |

|                                 |                             |
|---------------------------------|-----------------------------|
| <code>rename</code> key1 key2   | 重命名key1, 如果key2已经存在, 其值会被覆盖 |
| <code>renamenx</code> key1 key2 | key2不存在时重命名key1             |

|                             |                    |
|-----------------------------|--------------------|
| <code>move</code> key 数据库编号 | 按key将一个键值对移动到指定数据库 |
| <code>copy</code> key1 key2 | 将key1的值拷贝给key2     |

## 2.2 字符串String

key-value，键-值

|      |        |
|------|--------|
| key1 | value1 |
| key2 | value2 |
| ...  |        |

### 2.2.1 基本操作

|  |   |
|--|---|
| <code>set key value [EX seconds PX milliseconds EXAT unix-time-seconds PXAT unix-time-milliseconds KEEPTTL] [NX XX] [GET]</code> | 添加/修改一个键值对<br>EX(expire): 设置过期秒数<br>PX: 设置过期毫秒数<br>EXAT: 设置过期的unix秒<br>PXAT: 设置过期的unix毫秒<br>NX: 仅当成员不存在时添加成员<br>KEEPTTL: set时不重置ttl(见2.2.3)<br>XX: 仅当成员存在时修改成员<br>GET: 修改一个键值对并返回原值，原值不存在则返回nil |
| <code>get key</code>   | 按key获取value   |
| <code>mset key1 value1 [key2 value2 ...]</code>  | 添加/修改一至多个键值对  |
| <code>mget key1 [key2 ...]</code>  | 按key获取一至多个value   |

|   |  |
|---|--|
| <code>append key value</code>                 | 在原有value后追加内容  |
| <code>strlen key</code>                       | 查看字符串长度  |
| <code>getrange key startindex endindex</code> | 获取范围时[startindex, endindex]的子串<br>*index从0开始，-n表示倒数第n个字符 |

|   |                         |
|---|-------------------------|
| <code>*setnx key value</code>                     | 不再推荐，同set key value nx  |
| <code>*getset key value</code>                    | 不再推荐，同set key value get |
| <code>msetnx key1 value1 [key2 value2 ...]</code> | 批量版setnx                |

### 2.2.2 如果字符串的内容是数值

|                       |                                      |
|-----------------------|--------------------------------------|
| <code>incr key</code> | 按key创建值为1的value，或使value增长(increase)1 |
|-----------------------|--------------------------------------|

|                     |                            |
|---------------------|----------------------------|
| incrby key 整数值      | 按key使value增长(increase)给定数值 |
| incrbyfloat key 小数值 | 按key使value增长(increase)给定数值 |
| decr key            | 按key使value减小(decrease)1    |
| decrby key 数值       | 按key使value减小(decrease)给定数值 |

### 2.2.3 临时键值对

生存时间time to live, 缩写为ttl, 指键值对距离被删除的剩余秒数

\*如果重新set, 生存时间(time to live, ttl)将被重置

\*以下操作支持各种数据类型

|                 |   |
|-----------------|---|
| expire key 秒数   | 设定一个生存时间                                |
| ttl key         | 查看生存时间的剩余秒数<br>*key不存在则返回-2, 永久键值对则返回-1 |
| pexpire key 毫秒数 | 毫秒版expire                               |
| pttl key        | 毫秒版ttl                                  |
| persist key     | 持久化(取消生存时间)                             |

|                       |                             |
|-----------------------|-----------------------------|
| *setex key 秒数 value   | 不再推荐, 同set key value ex 秒数  |
| *psetex key 毫秒数 value | 不再推荐, 同set key value px 毫秒数 |

## 2.3 散列表Hash

key-field-value，键-字段-值

|      |        |        |
|------|--------|--------|
| key1 | field1 | value1 |
|      | field2 | value2 |
|      | ...    |        |
| key2 | field1 | value1 |
|      | field2 | value2 |
|      | ...    |        |
| ...  |        |        |

### 2.3.1 基本操作

|  |                             |
|--|-----------------------------|
| <code>hset key field1 value1 [field2 value2 ...]</code><br>*老版本为hset为单个字段和值，多个为hmset | 添加/修改一个键与一至多对字段和值           |
| <code>hget key field</code>  | 按key和field获取一对value         |
| <code>hget key field1 [field2 ...]</code>  | 按key和field获取一至多对value       |
| <code>hgetall key</code>   | 按key获取field获取全部的field-value |
| <code>hdel key field1 [field2 ...]</code>  | 删除一至多对field-value           |

|                                     |                            |
|-------------------------------------|----------------------------|
| <code>hsetnx key field value</code> | 仅在field不存在时添加一对field-value |
|-------------------------------------|----------------------------|

|                                |                         |
|--------------------------------|-------------------------|
| <code>hkeys key</code>         | 查看一个散列表中所有的field        |
| <code>hvals key</code>         | 查看一个散列表中所有的value        |
| <code>hlen key</code>          | 统计一个散列表中有多少对field-value |
| <code>hexists key field</code> | 查看一个field是否存在           |

|                                |                      |
|--------------------------------|----------------------|
| <code>hstrlen key field</code> | 按key和field查看value的长度 |
|--------------------------------|----------------------|

### 2.3.2 如果value字符串的内容是数值



|   |                                  |
|---|----------------------------------|
| <code>hincrby</code> <code>key</code> <code>field</code> 整数值      | 按key和field使value增长(increase)给定数值 |
| <code>hincrbyfloat</code> <code>key</code> <code>field</code> 小数值 | 按key和field使value增长(increase)给定数值 |

## 2.4 列表List

key-value0-value1-..., 键-有序的值列队

|      |        |        |     |
|------|--------|--------|-----|
| key1 | value0 | value1 | ... |
| key2 | value0 | value1 | ... |
| ...  |        |        |     |

### 2.4.1 基本操作

|  |                                     |
|--|-------------------------------------|
| <code>rpush key value0 [value1 ...]</code> | 在列表右侧推入1至多个值                        |
| <code>lpush key [... value1] value0</code> | 在列表左侧推入1至多个值                        |
| <code>rpop key [数量]</code>                 | 从列表右侧弹出(指定数量的)值<br>*全部弹出后, key也会被删除 |
| <code>lpop key [数量]</code>                 | 从列表左侧弹出(指定数量的)值<br>*全部弹出后, key也会被删除 |

|   |                       |
|---|-----------------------|
| <code>rpushx key value0 [value1 ...]</code> | 仅当列表存在时, 在列表右侧推入1至多个值 |
| <code>lpushx key [... value1] value0</code> | 仅当列表存在时, 在列表左侧推入1至多个值 |

|   |  |
|---|--|
| <code>lset key *index value</code>                  | 修改指定位置的值                               |
| <code>linsert key before/after 定位value value</code> | 在定位value前/后插入一个值                       |
| <code>lindex key *index</code>                      | 按索引查看值                                 |
| <code>lrange key *startindex *endindex</code>       | 查看给定范围的值                               |
| <code>llen key</code>                               | 查看队列长度                                 |
| <code>lrem key 数量 value</code>                      | 删除指定值<br>*数量为正代表从左侧开始删除, 数量为负代表从右侧开始删除 |
| <code>ltrim key *startindex *endindex</code>        | 将列表修剪到给定范围                             |

\*索引从0开始, -n表示倒数第n个

## 2.5 集合Set

key-stringX, stringY..., 键-无序的不重复的成员

|      |         |         |     |
|------|---------|---------|-----|
| key1 | stringX | stringY | ... |
| key2 | stringY | stringZ | ... |
| ...  |         |         |     |

### 2.5.1 基本操作

|                                      |                            |
|--------------------------------------|----------------------------|
| sadd key stringX [stringY ...]       | 添加1至多个成员                   |
| srem key stringX [stringY ...]       | 删除1至多个成员                   |
| scard key                            | 返回成员数量<br>*集合基数cardinality |
| sismember key string                 | 查看是否存在指定成员                 |
| smismember key stringX [stringY ...] | 批量查看是否存在指定成员               |
| smembers key                         | 查看集合中所有成员                  |
| randmember key [数量]                  | 随机查看指定数量的成员                |
| spop key [数量]                        | 随机取出指定数量的成员                |

|                                    |                |
|------------------------------------|----------------|
| smove key1 key2 string             | 将指定成员从集合1移至集合2 |
| sinter key1 [key2 ...]             | 查看给定集合的交集      |
| sinterstore newkey key1 [key2 ...] | 存储给定集合的交集      |
| sunion key1 [key2 ...]             | 查看给定集合的并集      |
| sunionstore newkey key1 [key2 ...] | 存储给定集合的并集      |
| sdiff key1 [key2 ...]              | 查看给定集合的差集      |
| sdiffstore newkey key1 [key2 ...]  | 存储给定集合的差集      |

## 2.6 有序集合ZSet

key-score1:stringX, score2:stringY...

键-按分数排序的不重复的成员

|      |         |         |     |
|------|---------|---------|-----|
| key1 | score1  | score2  | ... |
|      | stringX | stringY | ... |
| key2 | score1  | score2  | ... |
|      | stringY | stringZ | ... |
| ...  |         |         |     |

\*分数为(范围为float64)的浮点数，且可以重复

\*什么是float64/int53见《一万小时计划02》

### 2.6.1 基本操作

|  |  |
|--|--|
| <code>zadd key [nx/xx] [gt/lt] [ch] [incr] score1 stringX [score2 stringY...]</code> | 添加1至多个成员<br>NX: 仅当成员不存在时添加成员<br>XX: 仅当成员存在时修改成员<br>LT(less than): 仅当分数低于原有分数时才更新分数<br>GT(greater than): 仅当分数高于原有分数时才更新分数<br>*NX不可以与LT/GT同时使用<br>CH(changed): 返回变更的成员数量<br>*默认返回新增的成员数量<br>incr: 累加分数<br>*选择incr时, 只能操作一个分数-成员对 |
| <code>zrem key stringX [stringY ...]</code>  | 删除1至多个成员   |
| <code>zcount key minScore maxScore</code>  | 返回指定分数区间内的成员数量<br>支持开区间: 分数前加 "("<br>支持无穷大: "-inf" 负无穷大, "+inf" 正无穷大   |
| <code>zscore key string</code>   | 查看成员分数<br>*成员不存在时返回nil   |
| <code>zmscore key stringX [stringY ...]</code>                                       | 批量查看成员分数<br>*成员不存在时返回nil   |
| <code>zcard key</code>   | 查看成员数量   |

|                       |                |
|-----------------------|----------------|
| zincrby key 数值 string | 将指定成员的分数增加给定数值 |
|-----------------------|----------------|

## 2.6.2 按区间操作

|   |  |
|---|--|
| zrange key start end [byscore/bylex] [rev]<br>[limit 偏移量 查看数量] [withscores] | 查看指定分数区间内的成员<br>BYScore: 按分数生序排序<br>*BYScore支持开区间: 分数前加 "("<br>*BYScore支持无穷大: "-inf"负无穷大, "+inf"正无穷大<br>BYLex(lexicographical): (分数相同时)按成员字符排序<br>*BYLex指定的字符串区间, 且需要指定开闭: "[string"表示闭区间, "(string"表示开区间<br>*BYLex支持选取到开始或结束: "-"表示开始, "+"表示结束<br>REV(reverse): 反转<br>*LIMIT用于指定查看范围, 仅在开启BYScore/BYLex时可用<br>WITHSCORES: 带分数 |
| *zrevrange key start end [withscores]                                       | 不再推荐, 等效于zrange key start end rev [withscores]   |
| zrangestore newkey key start end [byscore/bylex] [rev] [limit 偏移量 查看数量]     | 存储指定分数区间内的成员<br>参考zrange key start end [byscore/bylex] [rev] [limit 偏移量 查看数量]  |

|  |  |
|--|--|
| *zrangebylex key startString endString [limit 偏移量 查看数量]    | 不再推荐, 等效于zrange key startString endString bylex [limit 偏移量 查看数量]     |
| *zrevrangebylex key startString endString [limit 偏移量 查看数量] | 不再推荐, 等效于zrange key startString endString bylex rev [limit 偏移量 查看数量] |
| zlexcount key startString endString                        | 查看指定字符串区间内的成员数量<br>参考zrange key startString endString bylex          |
| zremrangebylex key startString endString                   | 删除指定字符串区间内的成员<br>参考zrange key startString endString bylex            |

|   |   |
|---|---|
| *zrangebyscore key minScore maxScore [withscores] [limit 偏移量 查看数量]    | 不再推荐, 等效于zrange key minScore maxScore bylex [limit 偏移量 查看数量] [withscores]     |
| *zrevrangebyscore key minScore maxScore [withscores] [limit 偏移量 查看数量] | 不再推荐, 等效于zrange key minScore maxScore bylex rev [limit 偏移量 查看数量] [withscores] |

|   |  |
|---|--|
| <code>zremrangebyscore key minScore maxScore</code> | 删除指定分数区间内的成员<br>参考 <code>zrange key minScore maxScore byscore</code> |
| <code>zrank key string</code>                       | 查看成员升序排名<br>*从第0名开始  |
| <code>zrevrank key string</code>                    | 查看成员降序排名<br>*从第0名开始  |
| <code>zremrangebyrank key startRank endRank</code>  | 删除指定排名区间内的成员<br>*从第0名开始, -n表示倒数第n名                                   |

### 2.6.3 交集，并集，差集

|   |  |
|---|--|
| <code>zinter key的数量 key1 [key2 ...] [weights 权重1 [权重2 ...]] [aggregate sum min max] [withscores]</code>             | 查看给定集合的交集<br>WEIGHTS: 依次为每个集合的分数设置权重<br>*权重默认为1<br>AGGREGATE: 新分数的计算方法<br>*默认为sum<br>WITHSCORES: 带分数<br><br>新分数=AGGREGATE(集合1中该成员的分数*权重1, 集合2中该成员的分数*权重2, ...) |
| <code>zinterstore newkey key的数量 key1 [key2 ...] [weights 权重1 [权重2 ...]] [aggregate sum min max]</code>              | 存储给定集合的交集  |
| <code>zunion key的数量 key1 [key2 ...] [weights 权重1 [权重2 ...]] [aggregate sum min max] [withscores]</code>             | 查看给定集合的并集  |
| <code>zunionstore newkey key的数量 key1 [key2 ...] [weights 权重1 [权重2 ...]] [aggregate sum min max] [withscores]</code> | 存储给定集合的并集  |
| <code>zdiff key的数量 key1 [key2 ...] [withscores]</code>  | 查看给定集合的差集  |
| <code>zdiffstore newkey key的数量 key1 [key2 ...]</code>   | 存储给定集合的差集  |

## 2.7 遍历

### 2.7.1 遍历key

|  |  |
|--|--|
| <code>scan</code> 游标 [match 格式] [count 数量] [type 指定类型] | 遍历一定数量的key<br>游标：开始遍历的位置，从0开始，scan命令将返回遍历结束位置+1，如果返回0，说明已经遍历到了最后一个<br>match：指定key的格式，*为通配符<br>count：指定遍历数量，默认为10<br>type：指定遍历的类型 |
|--|--|

### 2.7.2 特定类型的遍历

|   |                 |
|---|-----------------|
| <code>hscan</code> key 游标 [match 格式] [count 数量] | 遍历指定散列表中的字段与值   |
| <code>sscan</code> key 游标 [match 格式] [count 数量] | 遍历指定集合中的成员      |
| <code>zscan</code> key 游标 [match 格式] [count 数量] | 遍历指定有序集合中的成员与分数 |

# 完

预告：接下来将在《Golang学习记》中更新如何在Go语言中使用Redis