---

link: null
title: 珠峰架构师成长计划
description: 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。
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=102 sentences=182, words=809

---

1. Redis简介

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。

2.Redis 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支持二进制的字符串、列表、哈希值、集合和有序集合等数据类型操作。
- 原子性 – 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 发布/订阅, 通知, key 过期等等特性。
- **3. 安装#****3.1 Window 下安装#**
- redis-windows (https://github.com/ServiceStack/redis-windows)
- redis服务端下载 (https://github.com/ServiceStack/redis-windows/raw/master/downloads/redis-latest.zip)
- redis客户端下载 (https://redisdesktop.com/download)
- redis官网 (https://redis.io/)
- redis中文网 (http://www.redis.cn/documentation.html)

```
redis-server.exe redis.windows.conf
 redis-cli.exe -h 127.0.0.1 -p 6379 。
```

** 3.2 Mac下安装 #**

```
brew install redis
brew services start redis
redis-server /usr/local/etc/redis.conf
```

** 3.3 配置 #**

```
CONFIG GET CONFIG_SETTING_NAME
CONFIG GET port
```

** 4. 数据类型 #**

- 字符串
- 哈希值
- 链表
- 集合
- 有序列表

** 4.1 字符串 #**

字符串是最基本的类型,一个key对应一个value

4.1.1 SET 设置值

```
SET name zfpx
```

4.1.2 GET 获取值

```
GET name
```

4.1.3 GETRANGE 获取子串

```
GETRANGE key start end
getrange name 1 2
"fp"
```

4.1.4 INCR INCRBY 递增

```
SET age 1
INCR age
INCRBY age 6
DECR age
DECRBY age  9
```

4.1.5 键

```
DEL key 删除 key
DEL user
EXISTS key 判断一个key是否存在
EXISTS user
EXPIRE key seconds  设置过期时间
EXPIRE user 10
TTL key 以秒为单位返回给定key的剩余生存时间
TTL user
TYPE key 返回key所存储的值的类型
TYPE user
```

** 4.2 哈希值 #**

哈希值是一个字符串类型的Key和值的映射表,特别适合用于存储对象。

4.2.1 HSET HMSET 设置值

```
HSET person name 设置单个值
HMSET user name zfpx age 9  设置多个值
```

4.2.2 HGET HGETALL 获取值

```
HGET user name 获取单个值
HMGET user name age 获取多个值
HGETALL user 获取多有值
```

4.2.3 HDEL key field 删除键

```
HDEL key field
HDEL user name
HGETALL user
```

4.2.3 HKEYS 获取所有的KEYS

```
 HKEYS user
```

** 4.3 列表 #**

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）。

4.3.1 LPUSH RPUSH 添加元素

返回列表的长度

```
LPUSH ids 2
LPUSH ids 1
RPUSH ids 3
RPUSH ids 4
RPUSH ids  5 6
```

4.3.2 LRANGE 查看元素

```
LRANGE ids 0 2
LRANGE ids 0 -1
```

4.3.3 LPOP RPOP 弹出元素

查看并删除

```
LPOP ids
RPOP ids
```

4.3.4 LINDEX ids 1

通过索引获取列表中的元素

```
LINDEX ids 0
```

4.3.5 LLEN key

获取列表长度

```
LLEN ids
```

4.3.6 LREM key count value

移除列表元素

- 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。

```
LREM ids count value
127.0.0.1:6379> lpush my 1
127.0.0.1:6379> lpush my 2
127.0.0.1:6379> lpush my 2
127.0.0.1:6379> lpush my 2
127.0.0.1:6379> lpush my 3
127.0.0.1:6379> lrange my 0 -1
1) "3"
2) "2"
3) "2"
4) "2"
5) "1"
127.0.0.1:6379> LREM my 2 2
(integer) 2
127.0.0.1:6379> lrange my 0 -1
1) "3"
2) "2"
3) "1"
```

** 4.4 集合 #**

集合是字符串类型的无序集合

4.4.1 SADD 添加

如果集合中已经存在指定的元素则返回0,如果不存在则添加成功返回1

```
SADD tags 1
SADD tags 2
SADD tags 2
SADD tags 3
SADD tags  4 5 6
SMEMBERS tags
```

4.4.2 SMEMBERS 查看集合

```
SMEMBERS tags
```

4.4.3 SCARD 获取集合元素个数

```
SCARD tags
```

4.4.4 SREM 删除元素

```
SREM tags member
SREM tags 4
SMEMBERS tags
```

4.4.5 集合运算

```
SADD A 1 2 3
SADD B 2 3 4
SINTER A B 交集
SDIFF A B 差集
SUNION A B 并集
```

** 4.5 有序集合 #**

有序集合和集合一样也是字符串的集合，而且不能重复 不同之外是每个集合都会关联一个double类型的分数，redis可以通过这个分类来为集合中的元素进行从小到大排序,元素不能重复，但分数可以重复

4.5.1 ZADD 添加元素

```
ZADD key score1 member1 [score2 member2]

ZADD levels 1 one
ZADD levels 2 two
ZADD levels 3 three
ZADD levels 4 four
```

4.5.2 ZCARD 获取有序集合的成员数

```
ZCARD key
ZCARD levels
```

4.5.3 ZRANGE 查看有序集合

```
ZRANGE levels  0 -1 按范围查看
ZRANGE levels  0 2 WITHSCORES 按范围查看，并显示分数
```

4.5.4 ZREM 移除有序集合中的一个或多个成员

```
ZREM key member [member ...]
ZADD levels 1 one
ZADD levels 2 two
ZREM levels one
 ZRANGE levels  0 -1
```

** 5. Node.js中的使用 #**

- redis (https://www.npmjs.com/package/redis)

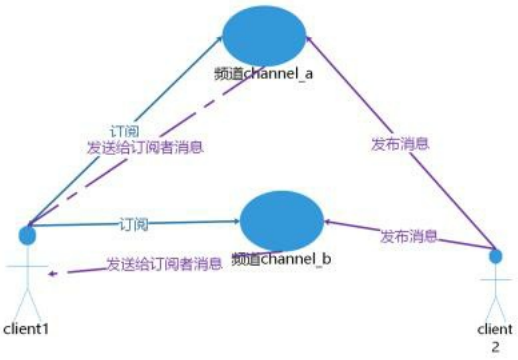
```
const redis = require('redis');
let client = redis.createClient(6379, '127.0.0.1');
client.on('error', function (error) {
    console.error(error);
});

client.set('name', 'zfpx', redis.print);
client.get('name', redis.print);

client.hset('user', 'name', 'zfpx', redis.print);
client.hset('user', 'age', '8', redis.print);
client.hget('user', 'age', redis.print);
client.hkeys('user', function (err, replies) {
    replies.forEach(function (item, index, items) {
        client.hget('user', item, redis.print);
    });
});
```

** 6. Redis发布订阅 #**

Redis 发布订阅是一种消息通信模式：发送者发送消息，订阅者接收消息，客户端可以订阅任意数量的频道。




```
SUBSCRIBE chat
PUBLISH chat zfpx
```

```
let client1 = redis.createClient(6379, '127.0.0.1');
let client2 = redis.createClient(6379, '127.0.0.1');
let count = 0;
client1.subscribe('food');
client1.subscribe('drink');
client1.on('message', function (channel, message) {
    console.log(channel, message);
    client1.unsubscribe('food');
});

client2.publish('food', '面包');
client2.publish('drink', '桔汁');
setTimeout(() => {
    client2.publish('food', '面包2');
    client2.publish('drink', '桔汁2');
}, 2000);
```

** 7. Redis事务 #**

Redis 事务可以一次执行多个命令

- 多个命令可以在执行EXEC命令之前放入缓存队列
- 收到EXEC命令后会将缓存队列执行
- 在执行事务的过程中，新提交的并不能被插入到事务执行序列中
- DISCARD 可以取消事务，放弃执行事务块内的所有命令

```
127.0.0.1:6379> MULTI
127.0.0.1:6379> SET account1 1
QUEUED
127.0.0.1:6379> SET account2 3
QUEUED
127.0.0.1:6379> EXEC
1) OK
2) OK
127.0.0.1:6379> GET account1
"1"
127.0.0.1:6379> GET account2
"3"
```

- 单个 Redis 命令的执行是原子性的，但 Redis 没有在事务上增加任何维持原子性的机制，所以 Redis 事务的执行并不是原子性的
- 事务可以理解为一个打包的批量执行脚本，但批量指令并非原子化的操作，中间某条指令的失败不会导致前面已做指令的回滚，也不会造成后续的指令不做

```
let redis = require('redis');
let client = redis.createClient(6379, '127.0.0.1');
client.multi().hset('user2', 'name2', 'zfpx2').hset('user2', 'age2', '92').exec(redis.print);
```

** 8. 备份与恢复 #**** 8.1 备份 #**

```
127.0.0.1:6379> SAVE
OK
```

> 该命令将在 redis 安装目录中创建dump.rdb文件。
> ** 8.2 恢复 #**

将备份文件 (dump.rdb) 移动到 redis 安装目录并启动服务

```
 CONFIG GET dir
 BGSAVE
```

** 8. 安全 #**

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。

```
127.0.0.1:6379> CONFIG get requirepass
1) "requirepass"
2) ""
127.0.0.1:6379> CONFIG set requirepass 'zfpx'
OK
127.0.0.1:6379> CONFIG get requirepass
(error) NOAUTH Authentication required.

127.0.0.1:6379> AUTH zfpx
OK
127.0.0.1:6379> CONFIG get requirepass
```