

# Redis

## 目录

1.安装部署 Redis.....	2
2.数据类型及操作.....	4
2.1 String 类型及操作.....	4
2.2 Hashes 类型及其操作.....	8
2.3 list 类型及其操作 .....	11
2.4 set 类型及操作.....	15
2.5 sorted sets 类型及操作 .....	20
3.Redis 常用命令.....	25
3.1.键值相关命令 .....	25
3.2.服务器相关操作 .....	28
4.Redis 高级实用特性 .....	31
4.1.安全性.....	31
4.2 主从复制 .....	32
4.3 事务处理 .....	33
4.4 持久化机制 .....	34
4.5 发布及订阅消息 .....	37
4.6 虚拟内存的使用【2.6后就取消了】 .....	38
5.Linux 下安装 PHPRedis 扩展 .....	39
6.Windows 下安装 PHPRedis 扩展 .....	40

# 1.安装部署 Redis

## 1.下载安装包

```
cd /var/lamp/  
wget http://download.redis.io/releases/redis-3.2.8.tar.gz
```

## 2.编译源程序

```
tar xzf redis-3.2.8.tar.gz  
cd redis-3.2.8  
make  
cd src && make install
```

## 3.移动文件，便于管理// bin 放命令， etc 放配置文件

```
mkdir -p /usr/local/redis/bin  
mkdir -p /usr/local/redis/etc  
mv /var/lamp/redis-3.2.8/redis.conf /usr/local/redis/etc  
cd /var/lamp/redis-3.2.8/src  
mv mkreleashdr.sh redis-benchmark redis-check-aof redis-check-rdb redis-cli redis-server /usr/local/redis/bin
```

【注意】新版本 redis-check-dump 改成 redis-check-rdb

## 4.启动 Redis 服务：

```
/usr/local/redis/bin/redis-server /usr/local/redis/etc/redis.conf
```

【注意】Redis 服务端默认的端口是 6379，MySQL 是 3306，monogDB 是 27017,28017  
要让 Redis 在后台启动，需要：修改下配置文件 redis.conf

```
vim /usr/local/redis/etc/redis.conf
```

进入编辑模式，修改 **daemonize** 为 **yes** ,保存退出。

启动 Redis 并且指定配置文件

```
redis-server /usr/local/redis/etc/redis.conf
```

## 查看系统进程情况

```
ps -ef |grep redis
```

```
root@sdb1:~# /usr/local/redis/bin/redis-server /usr/local/redis/etc/redis.conf  
root@sdb1:~# ps -ef |grep redis  
root    29407      1   0 14:13 ?        00:00:00 /usr/local/redis/bin/redis-server 127.0.0.1:6379  
root    29424  29269   0 14:13 pts/1    00:00:00 grep --color=auto redis  
root@sdb1:~#
```

参数：

-A : 所有的进程均显示出来，与 -e 具有同样的效用；

-a : 显示现行终端机下的所有进程，包括其他用户的进程；

-u : 以用户为主的进程状态；

x : 通常与 a 这个参数一起使用，可列出较完整信息

输出格式规划：

l : 较长、较详细的将该 PID 的信息列出；

j : 工作的格式 (jobs format)

-f : 做一个更为完整的输出。

## 查看网络状况

```
netstat -tunpl | grep 6379
```


---

```
root@iZbp15m012qi4y3tp6rhkpZ: /usr/local/redis/bin# ps -ef | grep redis
root      10203      1  0 13:32 ?        00:00:00 redis-server 127.0.0.1:6379
root      10238 10124  0 13:41 pts/5    00:00:00 grep  --color=auto redis
root@iZbp15m012qi4y3tp6rhkpZ: /usr/local/redis/bin# netstat -tunpl | grep 6379
tcp        0      0 127.0.0.1:6379      0.0.0.0:*           LISTEN      10203/redis-server
root@iZbp15m012qi4y3tp6rhkpZ: /usr/local/redis/bin#
```

### 5.客户端连接

/usr/local/redis/bin/redis-cli

```
root@iZbp15m012qi4y3tp6rhkpZ: /usr/local/redis/bin# ./redis-cli
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
```



打开客户端后

### 6.停止 Redis 实例

我们可以使用 /usr/local/redis/bin/redis-cli shutdown  
也可以使用 `kill redis-server` 或者 `kill all` 或者 `kill -9`

---

## 2.数据类型及操作

### 2.1 String 类型及操作

String 类型是最简单的类型，一个 key 对应一个 value 值，string 类型是二进制安全的，Redis 数据库的 string 可以包含任何数据，比如 jpg 图片，或者序列化的对象。

【命令】赋值与取值

#### set

SET key value -----如， set name zsong

GET key -----如， get name

```
127.0.0.1:6379>  
127.0.0.1:6379> set name zsong  
OK  
127.0.0.1:6379> get name  
"zsong"  
127.0.0.1:6379>
```

一个 key 对应一个 value，同一个键只能有一个。

```
127.0.0.1:6379> set name zsong  
OK  
127.0.0.1:6379> get name  
"zsong"  
127.0.0.1:6379> set name xiaoming  
OK  
127.0.0.1:6379> get name  
"xiaoming"  
127.0.0.1:6379>
```

会覆盖

#### setnx

设置 key 对应的 Sting 类型的 value，如果 key 已经存在，返回 0，nx 表示 not exist

```
127.0.0.1:6379> set name zhangsong  
OK  
127.0.0.1:6379> setnx name lijie  
(integer) 0  
127.0.0.1:6379> get name  
"zhangsong"  
127.0.0.1:6379>
```

#### setex


设置 key 对应的 Sting 类型的 value,并指定此键值对应的有效期。

例如，添加一个 HairColor=red 的键值对，设置有效时间为 10 秒。

```
127.0.0.1:6379> setex haircolor 10 red
OK
127.0.0.1:6379> get haircolor
"red"
127.0.0.1:6379> get haircolor
(nil)
127.0.0.1:6379>  有效期间为10秒
```


## setrange

设置 key 对应的 value 值得子字符串，  
例如，将小明的 126 邮箱修改为 gmail

```
127.0.0.1:6379> set name zsong16zj@126.com
OK
127.0.0.1:6379> get name
"zsong16zj@126.com"
127.0.0.1:6379> setrange name 10 gmail.com
(integer) 19
127.0.0.1:6379> get name
"zsong16zj@gmail.com"
127.0.0.1:6379> 
```


*表示从第几个下标开始替换*

注意一点，当替换的字符串长度小于被替换的字符串长度时，会有下面的现象出现。

```
127.0.0.1:6379> set email zsong@cjlhangzhou.com
OK
127.0.0.1:6379> get email
"zsong@cjlhangzhou.com"
127.0.0.1:6379> setrange email 6 163.com
(integer) 22
127.0.0.1:6379> get email
"zsong@163.comgzhou.com"
127.0.0.1:6379> 
```

## mset

如果一次性设置多个 key 值，成功返回 ok 表示所有的值都被设置了，失败返回 0 表示没有任何值被设置。

```
127.0.0.1:6379> mset key1 zsong key2 renlei key3 xiaoqiang
OK
127.0.0.1:6379> get key1
"zsong"
127.0.0.1:6379> get key2
"renlei"
127.0.0.1:6379> get key3
"xiaoqiang"
127.0.0.1:6379> get key4
(nil)
127.0.0.1:6379> 
```

## msetnx

如果一次性设置多个 key 值，成功返回 ok 表示所有的值都被设置了，失败返回 0 表示没有任何值被设置，但是不会覆盖掉之前已经存在的 key 值。

```

127.0.0.1:6379> mset key1 zsong key2 renlei key3 xiaoqiang
OK
127.0.0.1:6379> get key1
"zsong"
127.0.0.1:6379> get key2
"renlei"
127.0.0.1:6379> get key3
"xiaoqiang"
127.0.0.1:6379> get key4
(nil)
127.0.0.1:6379> msetnx key4 taian.net key5 chengxianghui key3 29
(integer) 0
127.0.0.1:6379> get key4
(nil)
127.0.0.1:6379> get key5
(nil)
127.0.0.1:6379>

```

key3已经存在了

## getset

简言之获取旧值，设置新值  
设置 key 的值，并返回 key 的旧值

```

127.0.0.1:6379> get key6
"zsong3"
127.0.0.1:6379> getset key6 xiaoming
"zsong3"
127.0.0.1:6379> get key6
"xiaoming"
127.0.0.1:6379>

```

key6的旧值

## getrange

获取 key 的 value 值得子字符串，

```

127.0.0.1:6379> getrange email 0 4
"zsong"
127.0.0.1:6379>

```

获取从0-4的值

## mget

一次获取多个 key 的 value 值，如果对应的 key 值不存在，则返回 nil

```

127.0.0.1:6379> mget key1 key2 key3 key4 key5 key6 key7
1) "zsong"
2) "renlei"
3) "xiaoqiang"
4) "zsong1"
5) "zsong2"
6) "xiaoming"
7) (nil)
127.0.0.1:6379>

```

## incr

对 key 的值作加加操作，并返回新的值，key 不存在的时候会设置 key,并认为原来的 value 是 0

```
127.0.0.1:6379> set key7 20
OK
127.0.0.1:6379> incr key7
(integer) 21
127.0.0.1:6379> incr key7
(integer) 22
127.0.0.1:6379> incr key7
(integer) 23
127.0.0.1:6379> get key7
"23"
127.0.0.1:6379> █
```

## incrby

类似于 incr,加指定值, key 不存在的时候会设置 key,并认为原来的 value 是 0

```
127.0.0.1:6379> incrby key7 7
(integer) 30
127.0.0.1:6379> incrby key7 -7
(integer) 23
127.0.0.1:6379> █
```

## decr

对 key 的值作减减操作

## decrby

类似于 decr, 对 key 的值作减减操作

```
127.0.0.1:6379> decr key7
(integer) 22
127.0.0.1:6379> decr key7
(integer) 21
127.0.0.1:6379> decr key7
(integer) 20
127.0.0.1:6379> decr key7
(integer) 19
127.0.0.1:6379> decr key7
(integer) 18
127.0.0.1:6379> decrby key7 -12
(integer) 30
127.0.0.1:6379> █
```

## append

给指定 key 的字符串追加 value, 返回新字符串的值的长度

append name @zsong.com

```
127.0.0.1:6379> get name
"zsong16zj@gmail.com"
127.0.0.1:6379> append name .net
(integer) 23
127.0.0.1:6379> get name
"zsong16zj@gmail.com.net"
127.0.0.1:6379> █
```

## strlen

取指定 key 的 value 值长度

```
127.0.0.1:6379> append name .net
(integer) 23
127.0.0.1:6379> get name
"zsong16zj@gmail.com.net"
127.0.0.1:6379> strlen name
(integer) 23
127.0.0.1:6379>
```

## 2.2 Hashes 类型及其操作

Redis hash 是一个键名对集合。

Redis hash 是一个 string 类型的 field 和 value 的映射表，hash 特别适合于存储对象。他的删除和添加操作都是 O(1) (平均)。相较于将对象的每个字段存成单个 string 类型，将一个对象储存在 hash 类型中会占用更少的内存，并且可以更方便的存储整个对象。

### hset

设置 hash field 为指定值，如果 key 不存在，则先创建。

```
127.0.0.1:6379> hset myhash feild1 hello
(integer) 1
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
```

hash 表名称      表中的字段      字段内容

下面的表示，创建 user 表，并设置，ID 为 001 的用户 name 为 zsong。

```
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> hset user:001 name zsong
(integer) 1
127.0.0.1:6379> hget user:001 name
"zsong"
127.0.0.1:6379>
```

Hget user:001 name 获取 ID 为 001 的用户的 name 值

### hsetnx

设置 hash field 为指定值，如果 key 不存在，则先创建。如果存在，返回 0。

```
127.0.0.1:6379> hsetnx user:002 name lamp
(integer) 1
127.0.0.1:6379> hget user:002 name
"lamp"
127.0.0.1:6379> hsetnx user:002 name lampbrother
(integer) 0
127.0.0.1:6379> hget user:002 name
"lamp"
127.0.0.1:6379>
```



## hmset

同时设置多个 hash 的 field，如 hmset user:003 name zsong age 26 sex 1

```
127.0.0.1:6379> hmset user:003 name zsong age 26 sex 1
OK
127.0.0.1:6379> hget user:003 age
"26"
127.0.0.1:6379> hget user:003 name
"zsong"
127.0.0.1:6379> hget user:003 sex
"1"
127.0.0.1:6379> █
```

## Hget

获取指定的 hash field

## Hmget

获取全部指定的 hash field

```
127.0.0.1:6379> hmget user:003 name age sex
1) "zsong"
2) "26"
3) "1"
█
```

## hincrby

给指定的 hash field 加上给定值

```
127.0.0.1:6379> hmget user:003 name age sex
1) "zsong"
2) "26"
3) "1"
127.0.0.1:6379> hincrby user:003 age 4
(integer) 30
127.0.0.1:6379> hmget user:003 name age sex
1) "zsong"
2) "30"
3) "1"
127.0.0.1:6379> █
```

## hexists

测试指定的 field 是否存在。

```
127.0.0.1:6379> hexists user:003 name
(integer) 1
127.0.0.1:6379> hexists user:003 age
(integer) 1
127.0.0.1:6379> hexists user:003 sex
(integer) 1
127.0.0.1:6379> hexists user:003 province
(integer) 0
127.0.0.1:6379> █
```

## Hlen

返回指定 hash 的 field 类型个数。如，hlen user:001

```
127.0.0.1:6379> hlen user:001
(integer) 1
127.0.0.1:6379> hlen user:003
(integer) 3
127.0.0.1:6379> █
```

## hdel

删除指定的 hash 的 field

```
127.0.0.1:6379> hdel user:003 name
(integer) 1
127.0.0.1:6379> hget user:003 name
(nil)
127.0.0.1:6379> █
```

## hkeys

返回 hash 的所有 field，类似于 PHP 中 `array_keys` — 返回数组中所有的键名

```
127.0.0.1:6379> hkeys user:003
1) "age"
2) "sex"
127.0.0.1:6379> hkeys user:001
1) "name"
127.0.0.1:6379> hkeys user:002
1) "name"
127.0.0.1:6379> █
```

## hvals

返回 hash 的所有 values，类似于 PHP 中 `array_values` — 返回数组的所有值（非键名）

```
127.0.0.1:6379> hvals user:003
1) "30"
2) "1"
127.0.0.1:6379> hvals user:001
1) "zsong"
127.0.0.1:6379> hvals user:002
1) "lamp"
127.0.0.1:6379> █
```

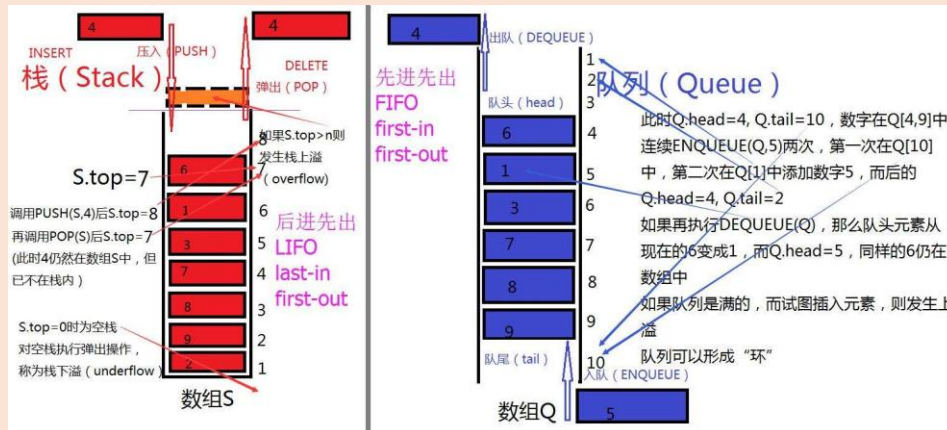
## hgetall

获取 hash 中全部的 field 中的所有 keys 和 values

```
127.0.0.1:6379> hgetall user:003
1) "age"
2) "30"
3) "sex"
4) "1"
127.0.0.1:6379> hgetall user:001
1) "name"
2) "zsong"
127.0.0.1:6379> hgetall user:002
1) "name"
2) "lamp"
127.0.0.1:6379> █
```

## 2.3 list 类型及其操作

list 是一个链表结构，主要功能是 push, pop, 获取一个范围的所有值等，在操作过程中 key 理解为链表的名称。Redis 的 list 类型其实就是一个每个子元素都是 string 类型的**双向**链表。我们可以通过 push、pop 操作从链表的头部或者尾部添加或者删除元素，这样 list 既可以作为栈，又可以作为队列。



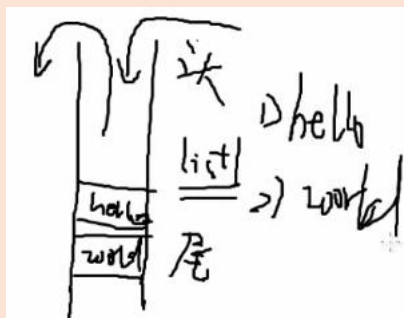
### lpush 操作

在 key 对应的 list 头部添加字符串元素。如下操作：

```
lpush list1 "world"
```

```
lpush list1 "hello"
```

```
lrange list1 0 -1
```



```
127.0.0.1:6379> lpush list1 "world"
(integer) 1
127.0.0.1:6379> lpush list1 "hello"
(integer) 2
127.0.0.1:6379> lrange list1 0 -1
1) "hello"
2) "world"
127.0.0.1:6379>
```

lrange 0 表示从头的 0 开始, -1 表示尾部第一个数据

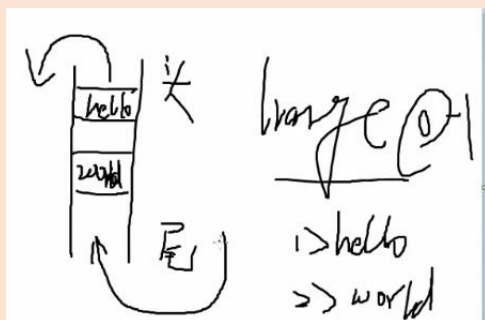
### rpush 操作, 相当于队列操作

与 lpush 相反, rpush 表示从 list 的尾部加入一个字符串元素。

```
rpush mylist "hello"
```

```
rpush mylist "world"
```

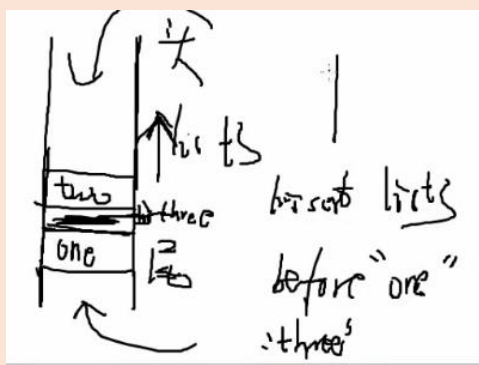
```
lrange mylist 0 -1
```



```
127.0.0.1:6379> rpush list2 "hello"
(integer) 1
127.0.0.1:6379> rpush list2 "world"
(integer) 2
127.0.0.1:6379> lrange list2 0 -1
1) "hello"
2) "world"
127.0.0.1:6379>
```

## linsert 操作

在 key 对应的 list 的特定位置前或者后添加字符串



```
127.0.0.1:6379> rpush list3 one
(integer) 1
127.0.0.1:6379> rpush list3 two
(integer) 2
127.0.0.1:6379> linsert list3 before two three
(integer) 3
127.0.0.1:6379> lrange list3 0 -1
1) "one"
2) "three"
3) "two"
127.0.0.1:6379>
```

## lset

将 list 中指定元素的下标替换掉。

```

127.0.0.1:6379> rpush list4 one
(integer) 1
127.0.0.1:6379> rpush list4 two
(integer) 2
127.0.0.1:6379> rpush list4 three
(integer) 3
127.0.0.1:6379> lrange list4 0 -1
1) "one"
2) "two"
3) "three"
127.0.0.1:6379> lset list4 1 four
OK
127.0.0.1:6379> lrange list4 0 -1
1) "one"
2) "four"
3) "three"
127.0.0.1:6379>

```

## lrem

从 key 对应的 list 中删除 n 个和 value 值相同的元素。(n<0,从尾部删除, n=0 全部删除)

```

127.0.0.1:6379> rpush list4 one
(integer) 4
127.0.0.1:6379> rpush list4 one
(integer) 5
127.0.0.1:6379> rpush list4 one
(integer) 6
127.0.0.1:6379> lrange list4 0 -1
1) "one"
2) "four"
3) "three"
4) "one"
5) "one"
6) "one"
127.0.0.1:6379> lrem list4 2 one
(integer) 2
127.0.0.1:6379> lrange list4 0 -1
1) "four"
2) "three"
3) "one"
4) "one"
127.0.0.1:6379>

```

删除的个数大于0, 从头开始删除

删除的个数

## ltrim

保留指定 key 的值范围内的数据。

```

127.0.0.1:6379> rpush list5 one
(integer) 1
127.0.0.1:6379> rpush list5 two
(integer) 2
127.0.0.1:6379> rpush list5 three
(integer) 3
127.0.0.1:6379> rpush list5 four
(integer) 4
127.0.0.1:6379> lrange list5 0 -1
1) "one"
2) "two"
3) "three"
4) "four"
127.0.0.1:6379> ltrim list5 1 2
OK
127.0.0.1:6379> lrange list5 0 -1
1) "two"
2) "three"
127.0.0.1:6379>

```

指范围从下标1到下标2保留, 其余的删除

## lpop, rpop

lpop 与 rpop 正好相反, lpop 是指从 list 的头部删除元素, 并且返回删除的元素, rpop 是指从 list 的尾部删除元素, 并返回删除的元素。

```

127.0.0.1:6379> rpush list6 three
(integer) 3
127.0.0.1:6379> rpush list6 four
(integer) 4
127.0.0.1:6379> rpush list6 five
(integer) 5
127.0.0.1:6379> rpush list6 six
(integer) 6
127.0.0.1:6379> lrange list6 0 -1
1) "one"
2) "two"
3) "three"
4) "four"
5) "five"
6) "six"
127.0.0.1:6379> lpop list6
"one"
127.0.0.1:6379> lrange list6 0 -1
1) "two"
2) "three"
3) "four"
4) "five"
5) "six"
127.0.0.1:6379> rpop list6
"six"
127.0.0.1:6379> lrange list6 0 -1
1) "two"
2) "three"
3) "four"
4) "five"
127.0.0.1:6379>

```

头部删除

尾部删除，返回删除元素

## rpoplpush

从第一个 list 的尾部删除一个元素，并把这个元素添加到下一个 list 的头部。

```

127.0.0.1:6379> rpush list7 hello
(integer) 1
127.0.0.1:6379> rpush list7 world
(integer) 2
127.0.0.1:6379> lrange list7 0 -1
1) "hello"
2) "world"
127.0.0.1:6379> rpush list8 nihao
(integer) 1
127.0.0.1:6379> rpush list8 zsong
(integer) 2
127.0.0.1:6379> lrange list 0 -1
(empty list or set)
127.0.0.1:6379> lrange list8 0 -1
1) "nihao"
2) "zsong"
127.0.0.1:6379> rpoplpush list7 list8
"world"
127.0.0.1:6379> lrange list7 0 -1
1) "hello"
127.0.0.1:6379> lrange list8 0 -1
1) "world"
2) "nihao"
3) "zsong"
127.0.0.1:6379>

```

source

destination

新加入的

## lindex

返回名称为 key 的 list 中的 index 位置的元素。

```
127.0.0.1:6379> lrange list8 0 -1
1) "world"
2) "nihao"
3) "zsong"
127.0.0.1:6379> lindex list8 1 相当于数组
"nihao" 的索引
127.0.0.1:6379> lindex list8 0
"world"
127.0.0.1:6379> lindex list8 2
"zsong"
127.0.0.1:6379> █
```

## llen

返回 key 对应的 list 的长度（元素个数）

```
127.0.0.1:6379> lrange list8 0 -1
1) "world"
2) "nihao"
3) "zsong"
127.0.0.1:6379> llen list8
(integer) 3 返回长度
127.0.0.1:6379> █
```

## 2.4 set 类型及操作

set 是集合，他是 string 类型的无序集合。Set 是通过 hash table 实现的，添加、删除和查找的复杂度都是  $O(1)$ 。对于集合，我们可以采取并集、交集、差集。通过这些操作我们可以实现 sns 中好友推荐的和 blog 的 tag 功能。

## sadd

向名称为 key 的 set 中添加元素。

```
127.0.0.1:6379> sadd myset1 one
(integer) 1
127.0.0.1:6379> sadd myset1 two 重复元素显
(integer) 1 示0，无法添
127.0.0.1:6379> sadd myset1 two 加
(integer) 0
127.0.0.1:6379> smembers myset1 集合元素查看
1) "one"
2) "two"
127.0.0.1:6379> █
```

## Srem

删除名称为 key 的 set 中的元素。

```

127.0.0.1:6379> sadd myset2 one
(integer) 1
127.0.0.1:6379> sadd myset2 two
(integer) 1
127.0.0.1:6379> sadd myset2 three
(integer) 1
127.0.0.1:6379> smembers myset2
(error) ERR unknown command 'smembers'
127.0.0.1:6379> smembers myset2
1) "three"
2) "one"
3) "two"
127.0.0.1:6379> srem myset2 two
(integer) 1
127.0.0.1:6379> srem myset2 two
(integer) 0
127.0.0.1:6379> smembers myset2
1) "three"
2) "one"
127.0.0.1:6379>

```

重复删除显示0

## Spop

从 set 集合中随机删除名称为 key 的元素。

```

127.0.0.1:6379> smembers myset3
1) "three"
2) "one"
3) "four"
4) "two"
5) "five"
127.0.0.1:6379> spop myset3
"three"
127.0.0.1:6379> spop myset3
"five"
127.0.0.1:6379> spop myset3
"two"
127.0.0.1:6379> smembers myset3
1) "one"
2) "four"
127.0.0.1:6379>

```

随机弹出一个元素

## Sdiff

返回所有给定 key 与第一个 key 的差集。

```

127.0.0.1:6379> smembers myset2
1) "three"
2) "one"
127.0.0.1:6379> smembers myset3
1) "one"
2) "four"
127.0.0.1:6379> sdiff myset2 myset3
1) "three"
127.0.0.1:6379>

```

以myset2为标准返回与myset3不一样的



```
127.0.0.1:6379> sdiff myset2 myset3
1) "three"
127.0.0.1:6379> sdiff myset3 myset2
1) "four"
127.0.0.1:6379> █
```

## Sdiffstore

返回所有给定 key 与第一个 key 的差集,并将结果保存到另一个 key。

```
127.0.0.1:6379> sdiff myset2 myset3
1) "three"
127.0.0.1:6379> sdiff myset3 myset2
1) "four"
127.0.0.1:6379> sdiffstore myset4 myset3 myset2
(integer) 1
127.0.0.1:6379> smembers myset4 将差集保存到 myset4
1) "four"
127.0.0.1:6379> █
```

## Sinter

返回所有给定 key 的交集。

```
127.0.0.1:6379> smembers myset2
1) "three"
2) "one"
127.0.0.1:6379> smembers myset3
1) "one"
2) "four"
127.0.0.1:6379> sinter myset2 myset3
1) "one"
127.0.0.1:6379> █
```

## Sinterstore

返回所有给定 key 与第一个 key 的交集,并将结果保存到另一个 key。

```
127.0.0.1:6379> smembers zsong1
1) "three"
2) "one"
3) "two"
127.0.0.1:6379> smembers zsong2 将zsong1和zsong2
1) "one" 的交集保存到
2) "four" zsong3
3) "two"
127.0.0.1:6379> sinterstore zsong3 zsong1 zsong2
(integer) 2
127.0.0.1:6379> smembers zsong3
1) "one"
2) "two"
127.0.0.1:6379> █
```

## sunion

返回所有给定的 key 的并集。

```
127.0.0.1:6379> smembers zsong1
1) "three"
2) "one"
3) "two"
127.0.0.1:6379> smembers zsong2
1) "one"
2) "four"
3) "two"
127.0.0.1:6379> sunion zsong1 zsong2
1) "three"
2) "one"
3) "four"
4) "two"
127.0.0.1:6379> █
```

## sunionstore

返回所有给定的 key 的并集，并保存。

```
127.0.0.1:6379> smembers zsong1
1) "three"
2) "one"
3) "two"
127.0.0.1:6379> smembers zsong2
1) "one"
2) "four"
3) "two"
127.0.0.1:6379> sunionstore zsong4 zsong1 zsong2
(integer) 4
127.0.0.1:6379> smembers zsong4
1) "three"
2) "one"
3) "four"
4) "two"
```

## smove

从第一个 key 对应的 set 中移除 member 并添加到第二个对应的 set 中。

```

127.0.0.1:6379> smembers zsong3
1) "one"
2) "two"
127.0.0.1:6379> smembers zsong4
1) "three"
2) "one"
3) "four"
4) "two"
127.0.0.1:6379> smembers zsong5
(empty list or set)
127.0.0.1:6379> smove zsong3 zsong4 one
(integer) 1
127.0.0.1:6379> smember zsong3
(error) ERR unknown command 'smember'
127.0.0.1:6379> smembers zsong3
1) "two"
127.0.0.1:6379> smembers zsong4
1) "one"
2) "four"
3) "two"
4) "three"
127.0.0.1:6379>

```

把zsong3中的one移除到zsong4

## scard

返回名称为 key 的 set 元素个数。

```

127.0.0.1:6379> smembers zsong4
1) "one"
2) "four"
3) "two"
4) "three"
127.0.0.1:6379> scard zsong4
(integer) 4
127.0.0.1:6379>

```

4个

## sismember

测试 member 是否是名称为 key 的 set 的元素。(类似于 PHP 的 in\_array())

```

127.0.0.1:6379> smembers zsong4
1) "one"
2) "four"
3) "two"
4) "three"
127.0.0.1:6379> scard zsong4
(integer) 4
127.0.0.1:6379> sismember zsong4 one
(integer) 1
127.0.0.1:6379> sismember zsong4 five
(integer) 0
127.0.0.1:6379>

```

判断one元素是否在zsong4中，是-->1 否-->0

## srandmember

随机返回名称为 key 的 set 中的一个元素，但不删除元素。

```

127.0.0.1:6379> smembers zsong4
1) "one"
2) "four"
3) "two"
4) "three"
127.0.0.1:6379> scard zsong4
(integer) 4
127.0.0.1:6379> sismember zsong4 one
(integer) 1
127.0.0.1:6379> sismember zsong4 five
(integer) 0
127.0.0.1:6379> srandmember zsong4
"four"
127.0.0.1:6379> srandmember zsong4 2
1) "one"
2) "four"
127.0.0.1:6379>

```

随机返回zsong4的元素

返回个数

## srandmember

随机返回名称为 key 的 set 中的一个元素，但不删除元素。

## 2.5 sorted sets 类型及操作

sorted sets 是 set 的一个升级版本，它在 set 的基础上增加了一个顺序属性，这一属性在添加、修改元素的时候可以指定，每次指定后，zset 会自动重新按照新的值调整顺序。可以理解为有两列的 MySQL 表，一列存 value，一列存顺序。在操作中 key 理解为有序集合 zset 的名称。

## zadd

向名称为 key 的 zset 中添加 member、score 用于排序，如果该元素存在，则更新其顺序。

```

127.0.0.1:6379> zadd myzset 1 one
(integer) 1
127.0.0.1:6379> zadd myzset 2 two
(integer) 1
127.0.0.1:6379> zadd myzset 3 two
(integer) 0
127.0.0.1:6379> zrange myzset 0 -1 withscores
1) "one"
2) "1"
3) "two"
4) "3"
127.0.0.1:6379>

```

顺序号

two元素已存在，只更新顺序

起始点

倒数第1个

## zrem

删除名称为 key 的 zset 中元素的 member.

```

127.0.0.1:6379> zrange myzset 0 -1 withscores
1) "one"
2) "1"
3) "two"
4) "3"
127.0.0.1:6379> zrem myzset one
(integer) 1
127.0.0.1:6379> zrange myzset 0 -1 withscores
1) "two"
2) "3"
127.0.0.1:6379>

```

删除one

## zincrby

如果存在名称为 key 的 zset 中已经存在元素 member，则该元素的 score 增加 increment，否则向该元素中添加该元素，其中 score 的值为 increment。【换言之，就是对顺序号进行增加】

```

127.0.0.1:6379> zrange myzset 0 -1 withscores
1) "two"
2) "3"
127.0.0.1:6379> zincrby myzset 2 two
(error) ERR unknown command 'zincrby'
127.0.0.1:6379> zincrby myzset 2 two
"5"
127.0.0.1:6379> zrange myzset 0 -1 withscores
1) "two"
2) "5"
127.0.0.1:6379>

```

对two的顺序号增加2

```

127.0.0.1:6379> zrange myzset 0 -1 withscores
1) "two"
2) "5"
127.0.0.1:6379> zincrby myzset -2 two
"3"
127.0.0.1:6379> zrange myzset 0 -1 withscores
1) "two"
2) "3"
127.0.0.1:6379>

```

同样可以减

## zrank

返回名称为 key 的 zset 中 member 元素的排名，按照 score 从小到大 开始排序即下标。

```

127.0.0.1:6379> zrange myzset 0 -1 withscores
1) "one"
2) "1"
3) "two"
4) "2"
5) "four"
6) "3"
7) "five"
8) "4"
127.0.0.1:6379> zrank myzset two
(integer) 1
127.0.0.1:6379>

```

索引为0  
索引为1  
索引为2  
要与顺序号区别开

## zrevrank

返回名称为 key 的 zset 中 member 元素的排名，按照 score 从大到小 开始排序，即索引下

标。

```
127.0.0.1:6379> zrange myzset 0 -1 withscores
1) "one"
2) "1"
3) "two"
4) "2"
5) "four"
6) "3"
7) "five"
8) "4"
127.0.0.1:6379> zrank myzset two
(integer) 1
127.0.0.1:6379> zrevrank myzset two
(integer) 2
127.0.0.1:6379>
```

索引为3  
索引为2  
索引为1  
索引为0

区别于zrank,一个是从前面排,一个从后面

## zrevrange

返回名称为 key 的 zset 中(按照 **score 从大到小**排序)的 index 从 start 到 end 的所有元素。

```
127.0.0.1:6379> zrange myzset 0 -1 withscores
1) "one"
2) "1"
3) "two"
4) "2"
5) "four"
6) "3"
7) "five"
8) "4"
```

```
127.0.0.1:6379> zrevrange myzset 0 -1 withscores
1) "five"
2) "4"
3) "four"
4) "3"
5) "two"
6) "2"
7) "one"
8) "1"
127.0.0.1:6379>
```

## zrangebyscore

返回集合中 score 在给定区间的元素。

```
127.0.0.1:6379> zrange myzset 0 -1 withscores
1) "one"
2) "1"
3) "two"
4) "2"
5) "four"
6) "3"
7) "five"
8) "4"
127.0.0.1:6379> zrangebyscore myzset 2 3 withscores
1) "two"
2) "2"
3) "four"
4) "3"
127.0.0.1:6379>
```

默认按照索引返回

按照顺序返回

```

4) "three"
127.0.0.1:6379> zrangebyscore myzset 2 4 withscores
1) "two"
2) "2"
3) "four"
4) "3"
5) "five"
6) "4"
127.0.0.1:6379>

```

表示顺序的范围

## zcount

返回集合中 score 在给定区间的数量。

```

0) "four"
127.0.0.1:6379> zcount myzset 2 4
(integer) 3
127.0.0.1:6379>

```

按照顺序

## zcard

返回集合中元素个数。

```

127.0.0.1:6379> zrange myzset 0 -1 withscores
1) "one"
2) "1"
3) "two"
4) "2"
5) "four"
6) "3"
7) "five"
8) "4"
127.0.0.1:6379> zcard myzset
(integer) 4
127.0.0.1:6379>

```

## zremrangebyrank

删除集合中排名(索引)在给定区间的元素。

```

127.0.0.1:6379> zrange myzset 0 -1 withscores
1) "one"
2) "1"
3) "two"
4) "2"
5) "four"
6) "3"
7) "five"
8) "4"
127.0.0.1:6379> zcard myzset
(integer) 4
127.0.0.1:6379> zremrangebyrank myzset 1 1
(integer) 1
127.0.0.1:6379> zrange myzset 0 -1 withscores
1) "one"
2) "1"
3) "four"
4) "3"
5) "five"
6) "4"
127.0.0.1:6379>

```

索引为1

删除索引1-1

## zremrangebyscore

删除集合中 score 在给定区间的元素。

```
127.0.0.1:6379> zrange myzset 0 -1 withscores
1) "one"
2) "1"
3) "four"
4) "3"
5) "five"
6) "4"
127.0.0.1:6379> zremrangebyscore myzset 1 2
(integer) 1
127.0.0.1:6379> zrange myzset 0 -1 withscores
1) "four"
2) "3"
3) "five"
4) "4"
127.0.0.1:6379>
```

按照顺序删除

chansonpro all right reserved



## 3.Redis 常用命令

Redis 提供了丰富的命令对数据库和各种数据类型进行操作，这些命令可以在 Linux 终端使用。

### 3.1.键值相关命令

#### Keys

返回满足给定的所有 pattern 的 key。

```
127.0.0.1:6379> keys *
1) "key6"
2) "zsong3"
3) "myset1"
4) "list2"
5) "myhash"
6) "user:002"
7) "list6"
8) "zsong4"
9) "zsong1"
10) "myset2"
11) "myset"
12) "key5"
13) "key4"
14) "key3"
15) "list3"
16) "myzset"
17) "key1"
18) "mylist"
19) "list7"
20) "key2"
21) "myset3"
22) "email"
23) "key7"
24) "list1"
25) "user:003"
26) "myset4"
27) "list5"
28) "list8"
```

通配符

```
127.0.0.1:6379> keys my*
1) "myset1"
2) "myhash"
3) "myset2"
4) "myset"
5) "myzset"
6) "mylist"
7) "myset3"
8) "myset4"
```

#### exists

确认一个 key 是否存在。

```
7) myset3
8) "myset4" 存在返回1
127.0.0.1:6379> exists name
(integer) 1
127.0.0.1:6379> exists age
(integer) 0
127.0.0.1:6379> █
```

## del

删除一个 key。

```
127.0.0.1:6379> del name
(integer) 1 删除成功
127.0.0.1:6379> del name
(integer) 0
127.0.0.1:6379> █
```

## expire

设置一个 key 的过期时间。为给定 key 设置生存时间，当 key 过期时(生存时间为 0)，它会被自动删除。

```
127.0.0.1:6379> expire age 10 生存时间10s
(integer) 1
127.0.0.1:6379> ttl age
(integer) 5
127.0.0.1:6379> ttl age
(integer) 4
127.0.0.1:6379> ttl age
(integer) 3
127.0.0.1:6379> ttl age 查看剩余时间
(integer) 1
127.0.0.1:6379> ttl age
(integer) 0
127.0.0.1:6379> ttl age
(integer) -2
127.0.0.1:6379> ttl age
(integer) -2
127.0.0.1:6379> ttl age
(integer) -2
127.0.0.1:6379> exists age
(integer) 0
127.0.0.1:6379> █
```

## Move

将当前数据库中的 key 转移到其他数据库中。

```

127.0.0.1:6379> select 0
OK
127.0.0.1:6379> set age 30
OK
127.0.0.1:6379> get age
"30"
127.0.0.1:6379> move age 1
(integer) 1
127.0.0.1:6379> get age
(nil)
127.0.0.1:6379> select 1
OK
127.0.0.1:6379[1]> get age
"30"
127.0.0.1:6379[1]>

```

在数据库0中

将key移动到1数据库

数据库有0-14个，都是数字表示的，客户端默认进的是0数据库

## Persist

移除给定 key 的过期时间。

```

127.0.0.1:6379[1]> expire age 300
(integer) 1
127.0.0.1:6379[1]> ttl age
(integer) 296
127.0.0.1:6379[1]> ttl age
(integer) 295
127.0.0.1:6379[1]> persist age
(integer) 1
127.0.0.1:6379[1]> ttl age
(integer) -1
127.0.0.1:6379[1]> get age
"30"
127.0.0.1:6379[1]>

```

## Randomkey

随机返回 key 空间的一个 key。

```

127.0.0.1:6379[1]> select 0
OK
127.0.0.1:6379> randomkey
"list7"
127.0.0.1:6379> randomkey
"user:003"
127.0.0.1:6379> randomkey
"list2"
127.0.0.1:6379>

```

## Rename

重新命名 key。

```
127.0.0.1:6379> select 1
OK
127.0.0.1:6379[1]> keys *
1) "age"
127.0.0.1:6379[1]> rename age newage
OK
127.0.0.1:6379[1]> keys *
1) "newage"
127.0.0.1:6379[1]>
```

## Type

返回键的**类型**。

```
127.0.0.1:6379> type zsong3
set
127.0.0.1:6379> type list1
list
127.0.0.1:6379>
```

## 3.2.服务器相关操作

### ping

测试连接是否存活。

第一个ping 时，说明此连接正常  
第二个ping 之前，我们将redis 服务器停止，那么  
ping 是失败的  
第三个ping 之前，我们将redis 服务器启动，那么  
ping 是成功的

```
root@sdb1:~# pkill redis-server
root@sdb1:~# netstat -tunpl |grep 6379
root@sdb1:~#
```

```
(error) ERR unknown command 'ping'
127.0.0.1:6379> ping
Could not connect to Redis at 127.0.0.1:6379: Connection refused
not connected>
```

### echo

在命令行打印一些内容。

```
127.0.0.1:6379> echo hello
"hello"
127.0.0.1:6379> echo world
"world"
127.0.0.1:6379>
```

### select

选择数据库，Redis 数据库，编号为 0-15，可以选择任意一个数据库来选取。

### quit

退出当前连接。

## dbsize

返回当前数据库中 key 的个数。

## info

获取服务器的信息和统计。

```
127.0.0.1:6379> info
# Server
redis_version:3.2.8
redis_git_sha1:00000000
redis_git_dirty:0
redis_build_id:51c0e574fb41fdbf
redis_mode:standalone
os:Linux 4.4.0-53-generic i686
arch_bits:32
multiplexing_api:epoll
gcc_version:4.8.4
process_id:6202
run_id:ee52b9e2030732495a3a7bc2c2dfb2541096a7a1
tcp_port:6379
uptime_in_seconds:884
uptime_in_days:0
hz:10
lru_clock:1247081
executable:/usr/local/redis/bin/redis-server
config_file:/usr/local/redis/etc/redis.conf

# Clients
```

## config get

实时转储收到的请求。

```
127.0.0.1:6379> config get *
1) "dbfilename"
2) "dump.rdb"
3) "requirepass"
4) ""
5) "masterauth"
6) ""
7) "unixsocket"
8) ""
9) "logfile"
10) ""
11) "pidfile"
12) "/var/run/redis_6379.pid"
13) "slave-announce-ip"
14) ""
15) "maxmemory"
16) "3221225472"
```

```
127.0.0.1:6379> config get maxclients
1) "maxclients"
2) "10000"
127.0.0.1:6379>
```

## flushdb

删除当前选择的数据库中的所有 key。

```
OK
127.0.0.1:6379[1]> keys *
1) "myzset"
2) "email"
3) "name"
127.0.0.1:6379[1]> flushdb
OK
127.0.0.1:6379[1]> keys *
(empty list or set)
127.0.0.1:6379[1]>
```

将当前数据库的 key 清除。

## flushall

删除所有数据库中的所有 key。

```
(empty list or set)
127.0.0.1:6379[1]> flushall
OK
127.0.0.1:6379[1]> keys*
(error) ERR unknown command 'keys*'
127.0.0.1:6379[1]> keys *
(empty list or set)
127.0.0.1:6379[1]> select 0
OK
127.0.0.1:6379> keys *
(empty list or set)
127.0.0.1:6379>
```

## 4.Redis 高级实用特性

### 4.1.安全性

设置客户端连接后，进行其他任何操作都需要输入密码。

**警告：**因为 Redis 的速度特别快，所以在一台比较好的服务器下，一个外部的用户可以在 1 秒钟进行 150k 次的密码尝试，这意味着需要指定非常强大的密码来防止暴力破解。

```
(empty list or set)
127.0.0.1:6379> exit
root@sdb1:~# vim /usr/local/redis/etc/redis.conf
```

进入配置文件，搜索 **requirepass**，vim 进入 vim /usr/local/redis/etc/redis.conf 文件里想查找单词 **requirepass** 切换到命令行模式，输入 **/requirepass**。之后按键盘上的“n”，继续向后查找，“n+shift”，表示向前继续查找。

```
# use a very strong password
#
# requirepass foobared
requirepass zsong
# Command renaming.
#
# It is possible to change the command renaming to another
# environment. For instance to ignore the command renaming,
# just add the following line to the configuration file:
```

之后重启服务。pkill redis-server

```
root@sdb1:~# vim /usr/local/redis/etc/redis.conf
root@sdb1:~# pkill redis-server
root@sdb1:~# /usr/local/redis/bin/redis-server /usr/local/redis/etc/redis.conf
root@sdb1:~#
root@sdb1:~# /usr/local/redis/bin/redis-cli 没有输入密码: zsong
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> keys *
(error) NOAUTH Authentication required.
127.0.0.1:6379>
```

```
127.0.0.1:6379> keys *
(error) NOAUTH Authentication required.
127.0.0.1:6379> auth zsong
OK
127.0.0.1:6379> keys *
(empty list or set)
127.0.0.1:6379>
```

auth表示授权，ok  
表示成功

为了避免每一次操作都需要输入密码，可以在登录的时候进行输入。

```
127.0.0.1:6379> exit
root@sdb1:~#
root@sdb1:~#
root@sdb1:~# /usr/local/redis/bin/redis-cli -a zsong
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> keys *
(empty list or set)
127.0.0.1:6379>
```

## 4.2 主从复制

Redis 主从复制可以允许多个 slave server 拥有和 master server 相同的数据库副本。

### Redis主从复制特点：

- 1.Master可以拥有多个slave
- 2.多个slave可以连接同一个master外，还可以连接到其它slave
- 3.主从复制不会阻塞master，在同步数据时，master可以继续处理client请求
- 4.提高系统的伸缩性

### 配置主从服务器：

配置slave服务器很简单，只需要在slave的配置文件中加入以下配置：

```
slaveof 192.168.1.1 6379 #指定master 的ip 和端口
```

```
masterauth lamp #这是主机的密码
```

陈服务器的操作如下：

slaveof 114.215.239.232 6379#指定主机的 IP 地址和端口号

masterauth zsong #设置的主机密码

### 设置成功后，我们来做一个实验：

我们在主数据库上设置一对键值对

```
redis 127.0.0.1:6379> set name master
OK
redis 127.0.0.1:6379>
```

在从数据库上取这个键

```
redis 127.0.0.1:6378> get name
"master"
redis 127.0.0.1:6378>
```

## 主从复制

那我们怎么判断哪个是主哪个是从呢？我们只需调用info就可以得到主从的信息，我们在从库中执行info

```
redis 127.0.0.1:6378> info
role:slave
master_host:localhost
master_port:6379
master_link_status:up
master_last_io_seconds_ago:10
master_sync_in_progress:0
db0:keys=1,expires=0
redis 127.0.0.1:6378>
```



## 4.3 事务处理

Redis 对事物的支持目前还比较简单, Redis 只能保证一个 client 发起的事务中的命令可以连续的执行, 中间不会插入其他的 client 命令。当一个 client 在一个连接中发出 **multi** 命令, 这个连接会进入一个事务上下文, 该连接后续的命令不会立即执行, 而是先放到一个队列中, 当执行 **exec** 命令时, Redis 会顺序的执行队列中的所有命令。

```
127.0.0.1:6379> get age
"33"
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> multi
OK
127.0.0.1:6379> set age 10
QUEUED
127.0.0.1:6379> set age 100
QUEUED
127.0.0.1:6379> exec
1) OK
2) OK
127.0.0.1:6379> get age
"100"
127.0.0.1:6379>
```

开启事务处理

表示将命令放入到序列

执行上面两个命令

取消一个事务。

**discard** 的命令其实就是清空事务的命令队列并退出事务上下文, 也就是我们说的事务回滚。

```
127.0.0.1:6379> get age
"100"
127.0.0.1:6379> multi
OK
127.0.0.1:6379> set age 10
QUEUED
127.0.0.1:6379> set age 100
QUEUED
127.0.0.1:6379> discard
OK
127.0.0.1:6379> get age
"100"
127.0.0.1:6379>
```

【注意】Redis 的事务回滚处理与 MySQL 不同, Redis 当执行两条命令时, 假设其中一条命令执行失败, 并不会影响到另一条记录的执行成功。

```
127.0.0.1:6379> keys *
1) "name"
2) "age"
127.0.0.1:6379> incr name
(error) ERR value is not an integer or out of range
127.0.0.1:6379> multi
OK
127.0.0.1:6379> incr age
QUEUED
127.0.0.1:6379> incr name
QUEUED
127.0.0.1:6379> exec
1) (integer) 101
2) (error) ERR value is not an integer or out of range
127.0.0.1:6379> get age
"101"
127.0.0.1:6379>
```

name是字符串, 无法自增

## 乐观锁复杂事务控制

乐观锁：大多数是基于数据版本(version)的记录机制实现的。即为数据增加一个版本标识，在基于数据库表的版本解决方案中，一般是通过为数据库表添加一个“version”字段来实现读取数据时，将此版本号一同读出，之后更新时，对此版本号加 1。此时，将提交数据的版本号与数据库表对应记录的当前版本号进行比对，如果提交的数据版本号大于数据库当前版本号，则予以更新，否则认为是过期数据。

### Redis 乐观锁实例：

假设有一个 age 的 key,我们开 2 个 session 来对 age 进行赋值操作，来看结果如何。

```
(1)第1步 session1
redis 127.0.0.1:6379> get age
"10"
redis 127.0.0.1:6379> watch age
OK
redis 127.0.0.1:6379> multi
OK
redis 127.0.0.1:6379>
```

```
(2)第2步 session2
redis 127.0.0.1:6379> set age 30
OK
redis 127.0.0.1:6379> get age
"30"
redis 127.0.0.1:6379>
```

```
(3)第3步 session1
redis 127.0.0.1:6379> set age 20
QUEUED
redis 127.0.0.1:6379> exec
(nil)
redis 127.0.0.1:6379> get age
"30"
redis 127.0.0.1:6379>
```

watch 命令会监视给定的 key，当 exec 时候如果监视的 key 从调用 watch 后发生变化，则整个事务会失败，也可以调用 watch 多次监视多个 key，这样就可以对指定的 key 加乐观锁了。注意 watch 的 key 是对整个连接有效的事务也一样。如果连接断开，监视和事务都会被自动清除。当然了，exec，discard，unwatch 命令都会清除连接中所有的监视。

## 4.4 持久化机制

Redis 是一个支持持久化的内存数据库，也就是说 Redis 需要经常将内存中的数据同步到硬盘来保证持久化。

Redis 支持两种持久化机制：

1. snapshotting（快照）也是默认方式
2. append-only file（缩写 aof）的方式

## snapshotting 方式

快照是默认的持久化方式。这种方式是将内存中的**数据**以快照的方式写入到二进制文件中，默认的文件名为 dump.rdb。可以通过配置文件设置自动做快照持久化的方式，我们可以配置 Redis 在 n 秒内如果超过 m 个 key 被修改就自动做快照。

```
save 900 1
save 300 10
save 60 10000
```

usr/local/redis/etc/redis.conf

save 900 1 #900秒内如果超过1个key被修改，则发起快照保存  
save 300 10 #300秒内容如超过10个key被修改，则发起快照保存  
save 60 10000

```
127.0.0.1:6379> exit
root@sdb1:~# cd /usr/local/redis/bin
root@sdb1:/usr/local/redis/bin# ll
total 12068
drwxr-xr-x 2 root root 4096 May 10 20:09 ./
drwxr-xr-x 4 root root 4096 Apr 27 11:48 ../
-rw-r--r-- 1 root root 881 May 10 20:09 dump.rdb
-rwxrwxr-x 1 root root 566 Feb 12 23:14 mkreleasehdr.sh*
-rwxr-xr-x 1 root root 1904142 Apr 27 11:46 redis-benchmark*
-rwxr-xr-x 1 root root 23994 Apr 27 11:46 redis-check-aof*
-rwxr-xr-x 1 root root 4177916 Apr 27 11:46 redis-check-rdb*
-rwxr-xr-x 1 root root 2053866 Apr 27 11:46 redis-cli*
-rwxr-xr-x 1 root root 4177916 Apr 27 11:46 redis-server*
root@sdb1:/usr/local/redis/bin# cat dump.rdb
REDIS0007 redis-ver3.2.8
ser:002name1ampÿlist6##twothreefourfiveÿzsong4threethwoonefourzsong1threoneetwomy
1zsongmylist
"hello"ÿlist7
ser:001namezsongÿb[]wage[]ÿ4wzroot@sdb1:/usr/local/redis/bin# XshellXshellXshellX
shellXshellXshellXshellXshellXshellXshellXshellXshellXshellXshellXshellXshell
XshellXshellXshellXshellXshellXshellXshellXshellXshellXshellXshellXshellXs
ellXshellXshellXshellXshellXshellXshellXshell: command not found
root@sdb1:/usr/local/redis/bin#
root@sdb1:/usr/local/redis/bin#
```

二进制文件

## aof 方式

由于快照方式是在一定间隔时间做一次，所以如果 Redis 意外 down 掉的话，就会丢失最后一次快照后的所有修改。

aof 比快照方式有更好的持久化机制，是由于在使用 aof 时，Redis 会将每一个收到的**写命令**都通过 write 函数追加到文件中，当 Redis 重启时候，会通过重新执行文件中保存的写命令来在内存中重建整个数据库的内容。

当然由于 os 会在内核中**缓存 write 做的修改**，所以可能不是立刻写到磁盘上，这样 aof 方式的持久化也还是有可能会丢失部分修改。

可以通过配置文件告诉 Redis 我们想要通过 **fsync 函数**强制 os 写入到磁盘的时机。

【fsync 函数同步内存中所有已修改的文件数据到储存设备】

```

appendonly yes //启用aof持久化方式
# appendfsync always //收到写命令就立即写入磁盘，最慢，但是保证完全的持久化
appendfsync everysec //每秒钟写入磁盘一次，在性能和持久化方面做了很好的折中
# appendfsync no //完全依赖os，性能最好，持久化没保证

```

```

# If the AOF is enabled on startup Redis will load the AOF, that is the
# with the better durability guarantees.
#
# Please check http://redis.io/topics/persistence for more information.

appendonly yes

# The name of the append only file (default: "appendonly.aof")

appendfilename "appendonly.aof"

# The fsync() call tells the Operating System to actually write data on
# disk. There's no need to flush right away, as the OS will flush as
# appropriate. If you have a large number of writes, you might want
# to use the "always" policy, as it will flush every time you write.

```

```

# More details please check the following article:
# http://antirez.com/post/redis-persistence-demystified.html
#
# If unsure, use "everysec".

# appendfsync always
appendfsync everysec
# appendfsync no

# When the AOF fsync policy is set to always or everysec, the Redis
# saving process (a background save or AOF log background save) will
# performing a lot of I/O against the disk, in some cases Redis may
# block too long on the fsync() call. Note that this is only a problem
# this currently, as even performing fsync in a different way will

```

保存退出后，`pkill redis-server`

```

root@sdb1:/usr/local/redis/etc# pkill redis-server
root@sdb1:/usr/local/redis/etc#
root@sdb1:/usr/local/redis/etc# cd
root@sdb1:~# /usr/local/redis/bin/redis-server /usr/local/redis/etc/redis.conf
root@sdb1:~# /usr/local/redis/bin/redis-cli
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> set name xiaoming
(error) NOAUTH Authentication required.
127.0.0.1:6379> auth zsong
OK
127.0.0.1:6379> set name xiaoming
OK
127.0.0.1:6379> exit
root@sdb1:~# ll
total 356
drwx----- 5 root root 4096 May 11 22:24 ./
drwxr-xr-x 21 root root 4096 Apr 27 12:24 ../
-rw-r--r-- 1 root root 60 May 11 22:25 appendonly.aof
-rw----- 1 root root 6291 May 10 21:41 .bash_history
-rw-r--r-- 1 root root 3106 Feb 20 2014 .bashrc
drwx----- 2 root root 4096 Dec 14 19:10 .cache/
-rw-r--r-- 1 root root 100 May 11 22:23 dump.rdb
-rw----- 1 root root 303468 May 10 13:38 .mysql_history
drwxr-xr-x 2 root root 4096 Dec 14 19:25 .pip/
-rw-r--r-- 1 root root 140 Feb 20 2014 .profile
-rw-r--r-- 1 root root 64 Dec 14 19:25 pydistutils.cfg
-rw----- 1 root root 1032 May 11 22:26 .rediscli_history
drwxr-xr-x 2 root root 4096 Apr 25 19:54 .rpmdb/
-rw----- 1 root root 2784 May 11 22:21 .viminfo
root@sdb1:~# cat appendonly.aof

```

```

root@sdb1:~# cat appendonly.aof
*2
$6
SELECT
$1
0
*3
$3
set
$4
name
$8
xiaoming
root@sdb1:~#

```

## 4.5 发布及订阅消息

发布订阅 (pub/sub) 是一种消息通信模式，主要的目的是解除消息发布者和消息订阅者之间的耦合，Redis 作为一个 pub/sub 的 server，在订阅者和发布者之间起到了消息路由的功能。订阅者可以通过 **subscribe** 和 **psubscribe** 命令向 Redis server 订阅自己感兴趣的消息类型，Redis 将信息类型称为**通道** (channel)。当发布者通过 **publish** 命令向 Redis server 发送数据特定类型的消息时，订阅该信息类型的**全部 client** 都会收到此消息。

下面开始试验：

另外打开两个窗口。

窗口 1：订阅 TV1

窗口 2：订阅 TV1 TV2

Qrcoder:消息发布者

<pre> 127.0.0.1:6379&gt; 127.0.0.1:6379&gt; 127.0.0.1:6379&gt; 127.0.0.1:6379&gt; 127.0.0.1:6379&gt; subscribe tv1 Reading messages... (press Ctrl-C to quit) 1) "subscribe" 2) "tv1" 3) (integer) 1 </pre> <p style="text-align: right;">窗口1</p>	<pre> 127.0.0.1:6379&gt; 127.0.0.1:6379&gt; 127.0.0.1:6379&gt; 127.0.0.1:6379&gt; subscribe tv1 tv2 Reading messages... (press Ctrl-C to quit) 1) "subscribe" 2) "tv1" 3) (integer) 1 1) "subscribe" 2) "tv2" 3) (integer) 2 </pre> <p style="text-align: right;">窗口2</p>
---	---

```

127.0.0.1:6379> publish tv1 nihaobeijing
(integer) 2
127.0.0.1:6379>
127.0.0.1:6379>

```

向TV1发送消息

<pre> 127.0.0.1:6379&gt; subscribe tv1 Reading messages... (press Ctrl-C to quit) 1) "subscribe" 2) "tv1" 3) (integer) 1 1) "message" 2) "tv1" 3) "nihaobeijing" 3) (integer) 1 </pre>	<pre> Reading messages... (press Ctrl-C to quit) 1) "subscribe" 2) "tv1" 3) (integer) 1 1) "subscribe" 2) "tv2" 3) (integer) 2 1) "message" 2) "tv1" 3) "nihaobeijing" </pre>
--	---

```

127.0.0.1:6379>
127.0.0.1:6379> publish tv2 hangzhouxihu
(integer) 1
127.0.0.1:6379>

```

```
(error) ERR unknown command '/usr/local/redis/bin/redi
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> subscribe tv1
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "tv1"
3) (integer) 1
1) "message"
2) "tv1"
3) "nihaobeijing"

127.0.0.1:6379> subscribe tv1 tv2
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "tv1"
3) (integer) 1
1) "subscribe"
2) "tv2"
3) (integer) 2
1) "message"
2) "tv1"
3) "nihaobeijing"
1) "message"
2) "tv2"
3) "hangzhouxihu"
```

只有TV2订阅者才能收到

## 4.6 虚拟内存的使用【2.6 后就取消了】

Redis 的虚拟内存与操作系统的**虚拟内存**不是一回事，但是思路和目的都是相同的。就是暂时不要把不经常访问的数据从**内存交换**到磁盘中，从而腾出宝贵的内存空间用于其他需要访问的数据。尤其是对于 Redis 这样的内存数据库，内存总是不够用的。除了可以将数据分割到多个 Redis server 外，另外能够提高数据库容量的办法就是使用虚拟内存把那些**不经常访问的数据交换到磁盘上**。

### 虚拟内存配置

下面是vm相关配置：

vm-enabled yes	#开启vm功能
vm-swap-file /tmp/redis.swap	#交换出来的value保存的文件路径
vm-max-memory 1000000	#redis使用的最大内存上限
vm-page-size 32	#每个页面的大小32字节
vm-pages 134217728	#最多使用多少页面
vm-max-threads 4	#用于执行value对象换入的工作线程数量

#### 重点提示：

Redis 的虚拟内存(VM) 目前不被提倡使用，Redis 2.4将是有虚拟内存特性的最新版本（但它同样提示不鼓励使用虚拟内存）。我们发现使用虚拟内存会有一些不足和问题。对于Redis的未来，至少目前在不考虑支持比RAM更大的数据库时，我们希望能提供最好的内存数据库（持久化仍然在磁盘上）。我们随后的成果将关注提供脚本，集群以及更好的持久化方面。



## 5.Linux 下安装 PHPRedis 扩展

phpredis 下载地址：<https://github.com/phpredis/phpredis/releases>

```
root@sdb1:/var/lamp# ls
phpredis-3.1.2.tar.gz  redis-3.2.8  redis-3.2.8.tar.gz
root@sdb1:/var/lamp# tar zxvf phpredis-3.1.2.tar.gz
```

### 【安装源代码包的三个步骤】

#### 【1】安装 redis 扩展

```
root@chanson:/usr/local/src# ls
phpredis-3.1.2.tar.gz  redis-3.2.8  redis-3.2.8.tar.gz
//解压文件
root@chanson:/usr/local/src# tar zxvf phpredis-3.1.2.tar.gz
//进入安装目录
root@chanson:/usr/local/src/phpredis-3.1.2# /usr/bin/phpize
// 用 phpize 生成 configure 配置文件，结果如下
root@chanson:/usr/local/src/phpredis-3.1.2# /usr/bin/phpize
Configuring for:
PHP Api Version:      20151012
Zend Module Api No:   20151012
Zend Extension Api No: 320151012
```

#### //配置

```
root@chanson:/usr/local/src/phpredis-3.1.2#
./configure --with-php-config=/usr/bin/php-config
```

#### //编译&&安装

```
make
make install
```

//安装完成之后，出现下面的安装路径，/usr/lib/php/20151012

```
root@chanson:/usr/local/src/phpredis-3.1.2# make install
Installing shared extensions: /usr/lib/php/20151012/
```

#### 【2】配置 php 支持，//编辑 php 配置文件，在最后一行添加以下内容

```
vi /etc/php/7.0/cli/php.ini
vi /etc/php/7.0/apache2/php.ini
extension="redis.so"
```

```
911 ;extension=php_xsl.dll
912 extension=redis.so
913 ;;;;;;;;;;;;;;;;;;
914 ; Module Settings ;
915 ;;;;;;;;;;;;;;;;;;
```

!wq! //保存退出

#### 【3】测试

/usr/bin/php -m //看下新安装的 redis 扩展是否存在，同样可以通过 phpinfo()查看

redis	
Redis Support	enabled
Redis Version	3.1.2
Available serializers	php

## 6.Windows 下安装 PHPRedis 扩展

Windows 下安装：

下载地址：<https://github.com/MSOpenTech/redis/releases>。

下载完成后解压到任意盘符如：D:\Program Files\redis

里面包括：如图所示。

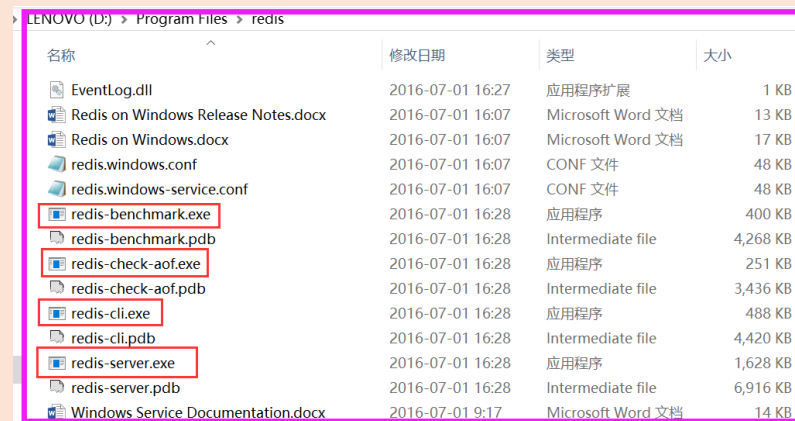
redis-server.exe：服务程序

redis-check-dump.exe：本地数据库检查

redis-check-aof.exe：更新日志检查

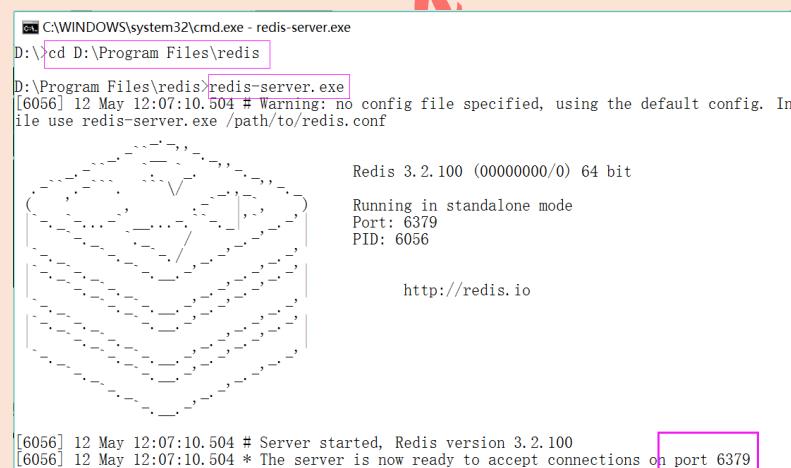
redis-benchmark.exe：性能测试，用以模拟同时由 N 个客户端发送 M 个 SETs/GETs 查询 (类似于 Apache 的 ab 工具)。

当然还需要一个：redis.conf



名称	修改日期	类型	大小
EventLog.dll	2016-07-01 16:27	应用程序扩展	1 KB
Redis on Windows Release Notes.docx	2016-07-01 16:07	Microsoft Word 文档	13 KB
Redis on Windows.docx	2016-07-01 16:07	Microsoft Word 文档	17 KB
redis.windows.conf	2016-07-01 16:07	CONF 文件	48 KB
redis.windows-service.conf	2016-07-01 16:07	CONF 文件	48 KB
redis-benchmark.exe	2016-07-01 16:28	应用程序	400 KB
redis-benchmark.pdb	2016-07-01 16:28	Intermediate file	4,268 KB
redis-check-aof.exe	2016-07-01 16:28	应用程序	251 KB
redis-check-aof.pdb	2016-07-01 16:28	Intermediate file	3,436 KB
redis-cli.exe	2016-07-01 16:28	应用程序	488 KB
redis-cli.pdb	2016-07-01 16:28	Intermediate file	4,420 KB
redis-server.exe	2016-07-01 16:28	应用程序	1,628 KB
redis-server.pdb	2016-07-01 16:28	Intermediate file	6,916 KB
Windows Service Documentation.docx	2016-07-01 9:17	Microsoft Word 文档	14 KB

运行 redis-server.exe 服务器开启，不能关闭此窗口，否则服务器会关闭。



```
C:\WINDOWS\system32\cmd.exe - redis-server.exe
D:\>cd D:\Program Files\redis
D:\Program Files\redis>redis-server.exe
[6056] 12 May 12:07:10.504 # Warning: no config file specified, using the default config. In
ile use redis-server.exe /path/to/redis.conf

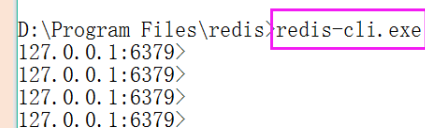
Redis 3.2.100 (00000000/0) 64 bit
Running in standalone mode
Port: 6379
PID: 6056

http://redis.io

[6056] 12 May 12:07:10.504 # Server started, Redis version 3.2.100
[6056] 12 May 12:07:10.504 * The server is now ready to accept connections on port 6379
```

重新打开一窗口：

运行 redis-cli,打开客户端。



```
D:\Program Files\redis>redis-cli.exe
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
```

Redis 安装好之后，开始配置 PHPRedis 扩展。注意根据自己的型号对应下载。



Build Date	Aug 18 2016 11:34:28
Compiler	MSVC11 (Visual C++ 2012)
Architecture	x64
Configure Command	escript /nologo configure.js "--enable-snapshot-build" "--disable-isapi" "--enable-without-mssql" "--without-pdo-mssql" "--without-p3web" "--with-pdo-oci=c:\php-sdk\oracle\x64\instantclient_12_1\sdk,shared" "--with-oci8-12c=c:\php-sdk\oracle\x64\instantclient_12_1\sdk,shared" "--enable-object-out-dir=../obj/" "dotnet=shared" "--with-mcrypt=static" "--without-analyzer" "--with-pgo"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\WINDOWS
Loaded Configuration File	C:\wamp64\bin\apache\apache2.4.23\bin\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20131106
PHP Extension	20131226
Zend Extension	220131226
Zend Extension Build	API220131226, TS, VC11
PHP Extension Build	API20131226, TS, VC11
Debug Build	no
Thread Safety	enabled
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring

下载 phpredis 扩展文件：



php\_redis扩展.rar

php\_redis.dll php\_igbinary.dll，将两个文件放到 C:\wamp64\bin\php\php5.6.25\ext  
修改 php.ini 文件，添加 扩展的时候一定要

extension=php\_igbinary.dll

extension=php\_redis.dll

```

918 ;extension=php_sybase_ct.dll
919 ;extension=php_tidy.dll
920 extension=php_xmlrpc.dll
921 extension=php_xsl.dll
922
923 extension=php_igbinary.dll
924 extension=php_redis.dll
925
926

```