

redis命令学习

./redis-server [conf] /usr/local/redis/etc/redis.conf
./redis-cli

关闭：pkill redis[cli/ser]

string 类型

set [key] [val]
get [key]

getrange val 7 5 10 范围获取

setnx not exist 不重复设置

mset [key1] [val1] [key2] [val2] · 批量设置

mget 批量获取

incr [key] 设置递增 整数有效 incrby [key] [num] 以num自增 num可以为负数

decr decrby 同理

strlen 获取长度

hashes

hset [hashname] [fields] [val]

hkeys 返回所有key

hvals 返回所有val

hgetall 返回所有key val

```
127.0.0.1:6379> hset user:001 val lau
(integer) 1
127.0.0.1:6379> hget val
(error) ERR wrong number of arguments for 'hget' command
127.0.0.1:6379> hget user:001 val
"lau"
127.0.0.1:6379>
```

```
127.0.0.1:6379> hmset user:003 name lau age 20 sex 1
OK
127.0.0.1:6379> hmget user:003 sex name
1) "1"
2) "lau"
```

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

lpush [listname] "value" 头部压入

0 -1表示从头到尾 (0和 -1中间有空格)

linsert list1 before one three
rpush

Lset

设置list中指定下标的元素值。

```
redis 127.0.0.1:6379> rpush mylist4 "hello"
(integer) 1
redis 127.0.0.1:6379> lset mylist4 0 "world"
OK
redis 127.0.0.1:6379> lrange mylist4 0 -1
1) "world"
```

Lrem

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

```
redis 127.0.0.1:6379> rpush mylist5 "hello"
(integer) 1
redis 127.0.0.1:6379> rpush mylist5 "hello"
(integer) 2
redis 127.0.0.1:6379> lrem mylist5 1 "hello"
(integer) 1
```

Lpop

从list的头部删除元素, 并返回删除元素

```
redis 127.0.0.1:6379> lrange mylist 0 -1
1) "hello"
2) "world"
redis 127.0.0.1:6379> lpop mylist
"hello"
redis 127.0.0.1:6379> lrange mylist 0 -1
1) "world"
redis 127.0.0.1:6379>
```

rpoplpush

从第一个list的尾部移除元素并添加到第二个list的头部。

Llen

返回key对应list的长度

```
redis 127.0.0.1:6379> llen mylist5
(integer) 2
redis 127.0.0.1:6379>
```

Redis适用场合

1. 取最新N个数据的操作
2. 排行榜应用, 取TOP N 操作 *key 过期*
3. 需要精确设定过期时间的应用
4. 计数器应用
5. Uniq操作, 获取某段时间所有数据排重值
6. 实时系统, 反垃圾系统

默认 6379

Redis的配置

daemonize 如果需要在后台运行, 把该项改为yes
pidfile 配置多个pid的地址 默认在/var/run/redis.pid
bind 绑定ip, 设置后只接受来自该ip的请求
port 监听端口, 默认为6379
timeout 设置客户端连接时的超时时间, 单位为秒
loglevel 分为4级, debug、verbose、notice、warning
logfile 配置log文件地址
databases 设置数据库的个数, 默认使用的数据库为0
save 设置redis进行数据库镜像的频率

Strings类型

```
redis 127.0.0.1:6379> setex haircolor 10 red
OK
redis 127.0.0.1:6379> get haircolor
"red"
redis 127.0.0.1:6379> get haircolor
(nil)
```

有效期设置

```
127.0.0.1:6379> set name 12345@126.com
OK
127.0.0.1:6379> get name
"12345@126.com"
127.0.0.1:6379> setrange name 6 hah.me
(integer) 13
127.0.0.1:6379> get name
"12345@hah.mem"
127.0.0.1:6379>
```

替换

集合中是无重复

set

string 的无序集合

hash实现

sets类型

sadd

向名称为key的set中添加元素

```
redis 127.0.0.1:6379> sadd myset "hello"
(integer) 1
redis 127.0.0.1:6379> sadd myset "world"
(integer) 1
redis 127.0.0.1:6379> sadd myset "world"
(integer) 0
redis 127.0.0.1:6379>
```

sets类型

srem

删除名称为key的set中的元素

```
redis 127.0.0.1:6379> sadd myset2 "one"
(integer) 1
redis 127.0.0.1:6379> sadd myset2 "two"
(integer) 1
redis 127.0.0.1:6379> srem myset2 "one"
(integer) 1
redis 127.0.0.1:6379>
```

sets类型

spop

随机返回并删除名称为key的set中一个元素

```
redis 127.0.0.1:6379> sadd myset3 "one"
(integer) 1
redis 127.0.0.1:6379> sadd myset3 "two"
(integer) 1
redis 127.0.0.1:6379> spop myset3
"two"
redis 127.0.0.1:6379>
```

sets类型

sdiff

返回所有给定key与第一个key的差集

```
redis 127.0.0.1:6379> smembers myset2
1) "three"
2) "two"
redis 127.0.0.1:6379> smembers myset3
1) "two"
2) "one"
redis 127.0.0.1:6379> sdiff myset2 myset3
1) "three"
```

sdiffstore

返回所有给定key与第一个key的差集，并将结果存为另一个key

```
redis 127.0.0.1:6379> smembers myset2
1) "three"
2) "two"
redis 127.0.0.1:6379> smembers myset3
1) "two"
2) "one"
redis 127.0.0.1:6379> sdiffstore myset4 myset2 myset3
(integer) 1
```

将 set2 和 set3的差集存到set4里

sinter

返回所有给定key的交集

```
redis 127.0.0.1:6379> smembers myset2
1) "three"
2) "two"
redis 127.0.0.1:6379> smembers myset3
1) "two"
2) "one"
redis 127.0.0.1:6379> sinter myset2 myset3
1) "two"
```

sunion

返回所有给定key的并集

```
redis 127.0.0.1:6379> smembers myset2
1) "three"
2) "two"
redis 127.0.0.1:6379> smembers myset3
1) "two"
2) "one"
redis 127.0.0.1:6379> sunion myset2 myset3
1) "three"
2) "one"
3) "two"
```

sunionstore

返回所有给定key的并集

```
redis 127.0.0.1:6379> smembers myset2
1) "three"
2) "two"
redis 127.0.0.1:6379> smembers myset3
1) "two"
2) "one"
redis 127.0.0.1:6379> sunionstore myset6 myset2 myset3
(integer) 3
```

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

```
redis 127.0.0.1:6379> smembers myset2
1) "three"
2) "two"
redis 127.0.0.1:6379> smembers myset3
1) "two"
2) "one"
redis 127.0.0.1:6379> smove myset2 myset7 three
(integer) 1
```

将 set2中 的 three 放入 set7中

scard

返回名称为key的set的元素个数

```
redis 127.0.0.1:6379> scard myset2
(integer) 1
redis 127.0.0.1:6379>
```

sismember

测试member是否是名称为key的set的元素

```
redis 127.0.0.1:6379> smembers myset2
1) "two"
redis 127.0.0.1:6379> sismember myset2 two
(integer) 1
redis 127.0.0.1:6379> sismember myset2 one
(integer) 0
redis 127.0.0.1:6379>
```

srandmember

随机返回名称为key的set的一个元素，但不删除元素

```
redis 127.0.0.1:6379> smembers myset3
1) "two"
2) "one"
redis 127.0.0.1:6379> srandmember myset3
"two"
redis 127.0.0.1:6379> srandmember myset3
"one"
redis 127.0.0.1:6379>
```

sorted sets类型

sorted sets类型及操作

sorted set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。操作中key理解为zset的名字。

有序集合

zadd

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

zadd

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

zrem

删除名称为key的zset中的元素member

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

Sorted sets类型

zincrby

```
redis 127.0.0.1:6379> zadd myzset2 1 "one"
(integer) 1
redis 127.0.0.1:6379> zadd myzset2 2 "two"
(integer) 1
redis 127.0.0.1:6379> zincrby myzset2 2 "one"
"3"
redis 127.0.0.1:6379> zrange myzset2 0 -1 withscores
1) "two"
2) "2"
3) "one"
4) "3"
```

Sorted sets类型

zrank

```
redis 127.0.0.1:6379> zrange myzset3 0 -1 withscores
1) "one"
2) "1"
3) "two"
4) "2"
5) "three"
6) "3"
7) "five"
8) "5"
redis 127.0.0.1:6379> zrank myzset3 two
(integer) 1
redis 127.0.0.1:6379>
```

返回 two的下标索引值是 1

Sorted sets类型

zrevrange

```
redis 127.0.0.1:6379> zrevrange myzset3 0 -1 withscores
1) "five"
2) "5"
3) "three"
4) "3"
5) "two"
6) "2"
7) "one"
8) "1"
redis 127.0.0.1:6379>
```

降序

```
Sorted sets类型
zrangebyscore
redis 127.0.0.1:6379> zrange myzset3 0 -1 withscores
1) "one"
2) "1"
3) "two"
4) "2"
5) "three"
6) "3"
redis 127.0.0.1:6379> zrangebyscore myzset3 2 3 withscores
1) "two"
2) "2"
3) "three"
4) "3"
```

找到 2~3范围的元素

```
zcount
redis 127.0.0.1:6379> zrange myzset3 0 -1 withscores
1) "one"
2) "1"
3) "two"
4) "2"
5) "three"
6) "3"
7) "five"
8) "5"
redis 127.0.0.1:6379> zcount myzset3 2 3
(integer) 2
redis 127.0.0.1:6379>
```

```
Sorted sets类型
zcard
redis 127.0.0.1:6379> zrange myzset3 0 -1 withscores
1) "one"
2) "1"
3) "two"
4) "2"
5) "three"
6) "3"
7) "five"
8) "5"
redis 127.0.0.1:6379> zcard myzset3
(integer) 4
redis 127.0.0.1:6379>
```

返回所有的个数

```
Sorted sets类型
zremrangebyrank
redis 127.0.0.1:6379> zrange myzset3 0 -1 withscores
1) "one"
2) "1"
3) "two"
4) "2"
redis 127.0.0.1:6379> zremrangebyrank myzset3 1 1
(integer) 1
redis 127.0.0.1:6379> zrange myzset3 0 -1 withscores
1) "one"
2) "1"
redis 127.0.0.1:6379>
```

删除 指定区间的元素

```
Sorted sets类型
zremrangebyscore
redis 127.0.0.1:6379> zrange myzset3 0 -1 withscores
1) "one"
2) "1"
3) "two"
4) "2"
5) "three"
6) "3"
redis 127.0.0.1:6379> zremrangebyscore myzset3 1 2
(integer) 2
redis 127.0.0.1:6379> zrange myzset3 0 -1 withscores
1) "three"
2) "3"
```


键值相关命令

Keys

返回满足给定pattern的所有key

```
redis 127.0.0.1:6379> keys *
```

```
1) "myzset2"
2) "myzset3"
3) "mylist"
4) "myset2"
5) "myset3"
6) "myset4"
7) "k_zs_1"
```

用表达式*, 代表取出所有的key

```
127.0.0.1:6379> keys my*
```

```
1) "myset2"
2) "myzset"
3) "myset4"
4) "myset3"
5) "myzset2"
5) "myset1"
7) "myset"
```

以 my 开头的所有内容 模糊匹配

键值相关命令

exists

确认一个key是否存在

```
redis 127.0.0.1:6379> exists name
(integer) 0
redis 127.0.0.1:6379> exists age
(integer) 1
redis 127.0.0.1:6379>
```

键值相关命令

del

删除一个key

```
redis 127.0.0.1:6379> del age
(integer) 1
redis 127.0.0.1:6379> exists age
(integer) 0
redis 127.0.0.1:6379>
```

键值相关命令

expire

设置一个key的过期时间

```
redis 127.0.0.1:6379> expire addr 10
(integer) 1
```

在本例中, 我们设置addr 这个key 的过期时间是10 秒, 然后我们不断的用ttl 来获取这个key 的有效时长, 直至为-1 说明此值已过期

```
(integer) -1
redis 127.0.0.1:6379>
```

expire

设置一个key的过期时间

```
redis 127.0.0.1:6379> expire addr 10
(integer) 1
redis 127.0.0.1:6379> ttl addr
(integer) 8
redis 127.0.0.1:6379> ttl addr
(integer) 1
redis 127.0.0.1:6379> ttl addr
(integer) -1
redis 127.0.0.1:6379>
```

服务器相关命令

select

选择数据库。Redis数据库编号从0~15，我们可以选择任意一个数据库来进行数据的存取

```
redis 127.0.0.1:6379> select 1
OK
redis 127.0.0.1:6379[1]> select 16
(error) ERR invalid DB index
redis 127.0.0.1:6379[16]>
```

当选择16时，报错，说明没有编号为16的这个数据库

键值相关命令

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

select选择第几个数据库 0~15

move 从一个数据库 移动到另一个输出库

键值相关命令

persist

移除给定key的过期时间

```
redis 127.0.0.1:6379[1]> expire age 300
(integer) 1
redis 127.0.0.1:6379[1]> ttl age
(integer) 294
redis 127.0.0.1:6379[1]> persist age
(integer) 1
redis 127.0.0.1:6379[1]> ttl age
(integer) -1
redis 127.0.0.1:6379[1]>
```

randomkey

随机返回key空间的一个key

```
redis 127.0.0.1:6379> randomkey
"mylist7"
redis 127.0.0.1:6379> randomkey
"mylist5"
redis 127.0.0.1:6379>
```

rename

重命名key

```
redis 127.0.0.1:6379[1]> keys *
1) "age"
redis 127.0.0.1:6379[1]> rename age age_new
OK
redis 127.0.0.1:6379[1]> keys *
1) "age_new"
redis 127.0.0.1:6379[1]>
```

键值对大命令

type

返回值的类型

```
redis 127.0.0.1:6379> type addr
string
redis 127.0.0.1:6379> type myzset2
zset
redis 127.0.0.1:6379> type mylist
list
redis 127.0.0.1:6379>
```

```
redis 127.0.0.1:6379> ping
PONG
//执行下面命令之前，我们停止redis 服务器
redis 127.0.0.1:6379> ping
Could not connect to Redis at 127.0.0.1:6379: Connection
refused
//执行下面命令之前，我们启动redis 服务器
not connected> ping
PONG
redis 127.0.0.1:6379>
```

pong即正常

echo

在命令行打印一些内容

```
redis 127.0.0.1:6379> echo lijie
"lijie"
redis 127.0.0.1:6379>
```

服务器大命令

dbsize

返回当前数据库中key的数目。

```
redis 127.0.0.1:6379> dbsize
(integer) 18
redis 127.0.0.1:6379>
```

结果说明此库中有18 个key

服务器相关命令

info

获取服务器的信息和统计。

服务器相关命令

config get

实时传输收到的请求。

```
redis 127.0.0.1:6379> config get dir
1) "dir"
2) "/root/4setup/redis-2.2.12"
redis 127.0.0.1:6379>
```

本例中我们获取了dir 这个参数配置的值，如果想获取全部参数的配置值也很简单，只需要执行“`config get *`”即可将全部的值都显示出来。

查找某个配置选项的内容 `config get *`

```
127.0.0.1:6379> config get timeout
1) "timeout"
2) "0"
```

flushdb

删除当前选择数据库中的所有key。

```
redis 127.0.0.1:6379> dbsize
(integer) 18
redis 127.0.0.1:6379> flushdb
OK
redis 127.0.0.1:6379> dbsize
(integer) 0
redis 127.0.0.1:6379>
```

在本例中我们将0号数据库中的key 都清除了。

flushall

删除所有数据库中的所有key。

```
redis 127.0.0.1:6379[1]> dbsize
(integer) 1
redis 127.0.0.1:6379[1]> select 0
OK
redis 127.0.0.1:6379> flushall
OK
redis 127.0.0.1:6379> select 1
OK
redis 127.0.0.1:6379[1]> dbsize
(integer) 0
redis 127.0.0.1:6379[1]>
```

redis问题解决 (MISCONF Redis is configured to save RDB snapshots)

(error) MISCONF Redis is configured to save RDB snapshots, but is currently not able to persist on disk. Commands that may modify the data set are disabled. Please check Redis logs for details about the error.

config set stop-writes-on-bgsave-error no

redis 高级用法



1.安全性

安全性

设置客户端连接后进行任何其他指定前需要使用的密码。

警告：因为redis 速度相当快，所以在一台比较好的服务器下，一个外部的用户可以在一秒钟进行150K 次的密码尝试，这意味着你需要指定非常非常强大的密码来防止暴力破解。

配置文件中设置

安全性

设置客户端连接后进行任何其他指定前需要使用的密码。

```
# requirepass foobared
requirepass beijing
```

```
5 #
6 # requirepass foobared
7 requirepass ireanlau
8 # Command renaming.
9 #
```

设置 密码 ireanlau

```
[lau@localhost bin]$ ./redis-cli
127.0.0.1:6379> keys *
(error) NOAUTH Authentication required.
```

设置密码成功后，进入执行命令 显示权限不允许

```
127.0.0.1:6379> auth ireanlau
OK
```

键入 auth + [密码] 登陆

```
redis-check-dump redis-check-dump redis-check-dump  
[la@localhost bin]$ ./redis-cli -a ireanlau
```

或者 在登陆的时候 -a + [密码]

2 主从复制

主从复制

Redis主从复制配置和使用都非常简单。通过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

Redis主从复制过程：

- 1.Slave与master建立连接，发送sync同步命令
- 2.Master会启动一个后台进程，将数据库快照保存到文件中，同时master主进程会开始收集新的写命令并缓存。
- 3.后台完成保存后，就将此文件发送给slave
- 4.Slave将此文件保存到硬盘上

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
```

info 可以查看 谁是主机 谁是从机

```
role:slave  
master_host:192.168.88.89  
master_port:6379  
master_link_status:up
```

```
role:master  
slave8:192.168.88.98,6379,online
```

实例：

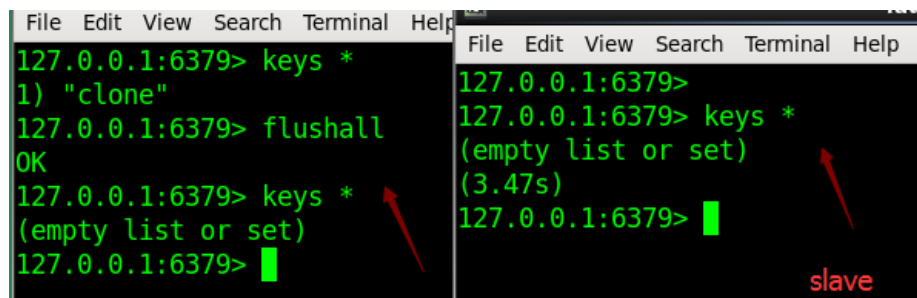
克隆一个主机



更改指定配置

```
#
slaveof <masterip> <masterport>
slaveof 192.168.253.128 6379

# If the master is password protected
# directive below) it is possible to
# starting the replication synchroniz
# refuse the slave request.
#
masterauth <master-password>
masterauth ireanlau
```



从机查询，没有结果

主机设置后从机可以查询

事物处理

Redis对事务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，而中间不会插入其他client的命令。当一个client在一个连接中发出multi命令时，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。

```
redis 127.0.0.1:6379> getage
"33"
redis 127.0.0.1:6379> multi
OK
redis 127.0.0.1:6379> setage 10
QUEUED
redis 127.0.0.1:6379> setage 20
QUEUED
redis 127.0.0.1:6379> exec
1) OK
2) OK
redis 127.0.0.1:6379> getage
"20"
redis 127.0.0.1:6379>
```

如何取消一个事务

```
redis 127.0.0.1:6379> get age
"20"
redis 127.0.0.1:6379> multi
OK
redis 127.0.0.1:6379> set age 30
QUEUED
redis 127.0.0.1:6379> set age 40
QUEUED
redis 127.0.0.1:6379> discard
OK
redis 127.0.0.1:6379> get age
"20"
redis 127.0.0.1:6379>
```

discard 取消事物队列，退出事物上下文（事物回滚）

乐观锁 =》（redis中的版本控制器）

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”字段来实现读取出数据时，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。

乐观锁复杂事务控制

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，我们来看一下结果如何。

```
(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>
```

watch age：监控age键 是否被修改，如果被修改过，事物不被执行

watch 命令会监视给定的key,当exec时候如果监视的key从调用watch后发生过变化,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的所有监视。

(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>
```

事务回滚

```
redis 127.0.0.1:6379> get age
"30"
redis 127.0.0.1:6379> get name
"lijie"
redis 127.0.0.1:6379> multi
OK
redis 127.0.0.1:6379> incr age
QUEUED
redis 127.0.0.1:6379> incr name
QUEUED
redis 127.0.0.1:6379> exec
1) (integer) 31
2) (error) ERR value is not an integer or out of range
redis 127.0.0.1:6379> get age
"31"
redis 127.0.0.1:6379> get name
"lijie"
redis 127.0.0.1:6379>
```

从这个例子中可以看到, age 由于是个数字,那么它可以有自增运算,但是name 是个字符串,无法对其进行自增运算,所以会报错,如果按传统关系型数据库的思路来讲,整个事务都会回滚,但是我们看到redis 却是将可以执行的命令提交了所以这个现象对于习惯于关系型数据库操作的朋友来说是很别扭的,这一点也是redis 今天需要改进的地方。

持久化

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支持两种持久化方式：

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

- 1.将数据存到文件
- 2.将（写，更改之类的操作方式）存到文件

Snapshotting方式

快照是默认的持久化方式。这种方式是将内存中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 #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
```

```
147 save 900 1
148 save 300 10
149 save 60 10000
```

aof方式

由于快照方式是在一定间隔时间做一次的，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性，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。

例如 快照方式 设置 900s，发起一次快照，如果在这个时间间隙中，宕机，则无法保存数据

当然由于os会在内核中缓存write做的修改，所以可能不是立即写到磁盘上。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。

```
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
```

fsync 同步函数

先设置 appendonly选项

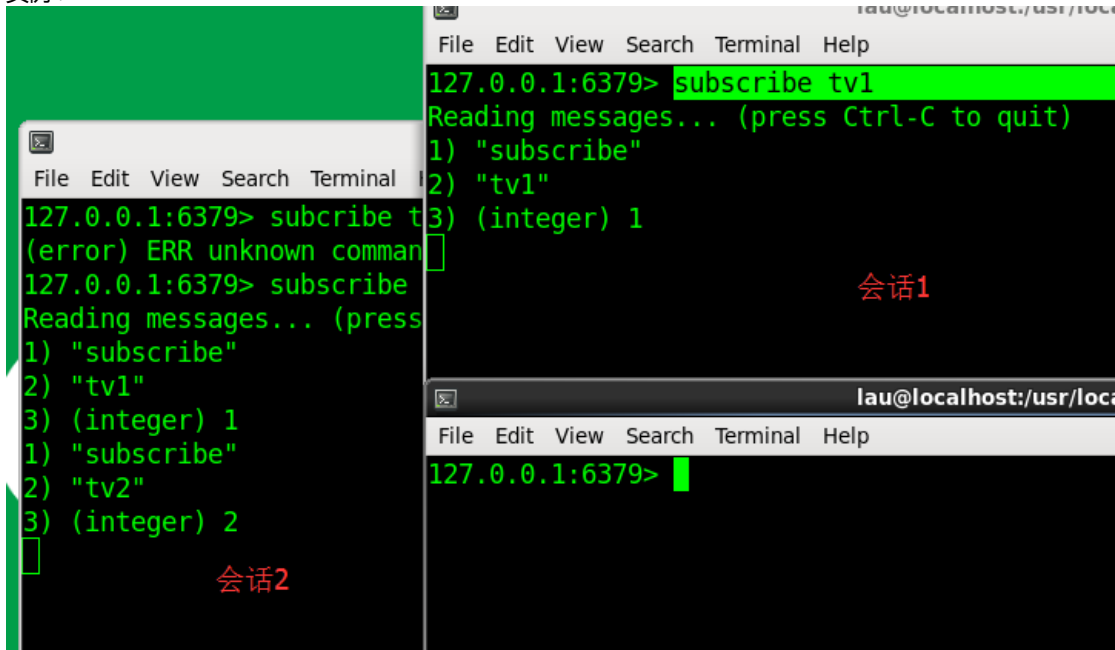
```
#appendonly no
appendonly yes
```

```
541 # appendfsync always
542 appendfsync everysec
543 # appendfsync no
544
```

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发送特定类型的信息时，订阅该信息类型的全部client都会收到此消息。

实例：



The image shows two terminal windows. The left window, labeled '会话2' (Session 2), shows a Redis client at 127.0.0.1:6379. It enters 'subscribe tv1' and receives a response: '1) "subscribe" 2) "tv1" 3) (integer) 1'. Then it enters 'subscribe tv2' and receives: '1) "subscribe" 2) "tv2" 3) (integer) 2'. The right window, labeled '会话1' (Session 1), shows a Redis client at 127.0.0.1:6379. It enters 'subscribe tv1' and receives the same response: '1) "subscribe" 2) "tv1" 3) (integer) 1'.

```
lau@localhost: /usr/local
File Edit View Search Terminal Help
127.0.0.1:6379> subscribe tv1
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "tv1"
3) (integer) 1
会话1
```

```
lau@localhost: /usr/local
File Edit View Search Terminal Help
127.0.0.1:6379> subscribe tv1
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "tv1"
3) (integer) 1
127.0.0.1:6379> subscribe tv2
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "tv2"
3) (integer) 2
会话2
```

会话1 订阅 tv1频道， 会话2订阅tv1 tv2频道


```
File Edit View Search Terminal Help
127.0.0.1:6379> subscribe tv1
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) "ireanlau"
█
```

```
lau@localhost:/usr/local/
File Edit View Search Terminal Help
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) "ireanlau"
█
```

```
lau@localhost:/usr/local/
File Edit View Search Terminal Help
127.0.0.1:6379> publish tv1 ireanlau
(integer) 2
127.0.0.1:6379> █
```

1. 发布者 通过tv1发布消息

2 订阅者 会收到订阅信息 (订阅该频道)

```
File Edit View Search Terminal Help
127.0.0.1:6379> subscribe tv1
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) "ireanlau"
1) "message"
2) "tv2"
3) "ermao"
█
```

```
File Edit View Search Terminal Help
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) "ireanlau"
█
```

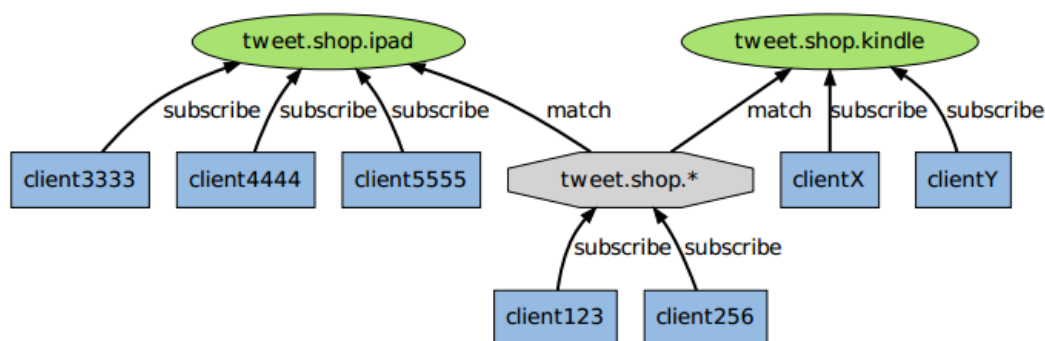
```
lau@localhost:/usr/local/
File Edit View Search Terminal Help
127.0.0.1:6379> publish tv1 ireanlau
(integer) 2
127.0.0.1:6379> publish tv2 ermao
(integer) 1
127.0.0.1:6379> █
```

1.通过tv2频道发布的ermao只有 会话2收到

使用 UNSUBSCRIBE 命令可以退订指定的频道

当使用 `PUBLISH` 命令发送信息到某个频道时，不仅所有订阅该频道的客户端会收到信息，如果有某个/某些模式和这个频道匹配的话，那么所有订阅这个/这些频道的客户端也同样会收到信息。

下图展示了一个带有频道和模式的例子，其中 `tweet.shop.*` 模式匹配了 `tweet.shop.kindle` 频道和 `tweet.shop.ipad` 频道，并且有不同的客户端分别订阅它们三个：



虚拟内存的使用

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 and 目的都是相同的。就是暂时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**。

```
948 #VM-config by ireanlau
949
950 # 开启 vm
951 #vm-enabled yes
952
953 #交换 value保存的路径
954 #vm-swap-file /tmp/redis.swap
955
956 #redis 使用的最大内存上限
957 #vm-max-memory 100000
958
959 #每个页面的大小32字节
960 #vm-page-size 32
961
962 #最多使用多少个页面
963 #pages 134217728
964
965 #用于执行 value 对象 换入的工作线程数目
966 #vm-max-threads 4
```

友情提示：闻说 2.4 version 之后的redis 取消了 vm的功能，此处仅作为了解配置。

