Redis学习记录手册

# 第1章 Redis概述

## Redis简介

Redis（Remote Dictionary Server）是NoSQL数据库的一种，Redis是一个开源的、使用C语言编写的、支持网络交互的、可基于内存也可持久化的Key-Value数据库。，通常叫数据结构服务器（Data Structure Server）。

键（key）可以包含字符串strings、哈希hashes、list链表、集合sets、有序集合sorted sets/zsets,这些数据类型支持push、pop、add、move及集合等操作,支持各种不同方式排序，为了提高性能，数据都存放在内存中，为了数据的可靠性需要周期性的将数据更新到磁盘或日志文件中

## Redis特点

Redis作为NoSQL的一种，自然继承了NoSQL的高扩展、海量数据存储和访问需求、灵活的数据模型等特性。除此之外Redis本身的具体特性如下：

* key-value键值类型存储系统
* 支持数据可靠存储和落地
* 单进程单线程高性能服务器
* crash safe及recovery slow
* 单机QPS(每秒查询率)可达到10w
* 适合小数据量高速读写访问

## Redis优点

* + 性能很高：redis能支持超过100K+每秒的读写频率（10万+）
  + 丰富的数据类型：redis支持二进制的Strings，Lists，Hashes，Sets及Orderd Sets等数据类型操作
  + 原子性：redis的所有操作都是原子性的，同时redis还支持对几个操作全并后的原子性执行
  + 丰富的特性：redis还支持publish/subscribe、事务、通知key过期等等特性
  + Redis支持异步机制主从复制，支持高可用故障切换
  + Redis还提供了丰富的客户端，以便支持现阶段流行的大多数编程语言。

## Redis缺陷

* 系统运行有毛刺;
* 不同命令延迟差别极大;
* 内存管理开销大;
* buffer io造成系统OOM<内存溢出>。

## 1.5 Redis应用场景

要清楚redis应用场景我们先来了解下传统的MySQL + Memcached的网站架构遇到的问题。

MySQL数据库实际上是适合进行海量数据存储的，加上通过MC将热点数据存放到内存cache里，达到加速数据访问的目的；绝大部分公司都曾经使用过这样的架构，但随着业务数据量的不断增加，和访问量的持续增长，很多问题就会暴漏出来：

* 需要不断的对MySQL进行拆库拆表，MC也需要不断跟着扩容，扩容和维护工作占据大量开发运维时间；
* MC与MySQL数据库数据一致性问题是个老大难题；
* MC数据命中率低或宕机，会导致大量访问直接穿透到数据库，导致MySQL无法支持访问；
* 跨机房cache同步一致性的问题

Redis的最佳应用场景：

* redis最佳适用场景是全部数据in-memory
* redis更多场景是作为MC的替代品来使用
* 支持持久化，每秒10w读写频率
* 当需要除key/value之外的更多数据类型支持时，使用redis更合适
* 需要负载均衡的场景（redis主从同步）

更多Redis作者谈Redis应用场景见：<http://blog.nosqlfan.com/html/2235.html>

## 1.6 Redis使用的案例

1）appliaction====>redis 应用程序直接访问redis，类似一个独立的数据库



2）应用程序首先访问redis,只有当redis没有数据或访问失败时访问mysql



3) 二次开发实现，mysql与redis互相同步

## 1.7 Redis数据库小结

* 提高了DB的可扩展性，只需要将新加的数据放到新加的服务器上就可以了；
* 提高了DB的可用性，只影响到需要访问的shard服务器上的数据的用户；
* 提高了DB的可维护性，对系统的升级和配置可以按shard一个个来搞，对服务产生的影响较小；
* 当Redis物理内存使用超过内存总容量的3/5时，就会开始比较危险了，就开始做swap，内存碎片大；
* 当达到最大内存时，会清空带有过期时间的key，即使key未到过期时间；
* redis与DB同步写的问题，先写DB，后写redis，因为写内存基本上没有问题；

## 1.8 生产环境经验教训

* 要进行master-slave主从同步配置，在出现服务故障时可以切换
* 在master禁用数据持久化，只需在slave上配置数据持久化
* 物理内存+虚拟内内存不足，这时dump一直死着，时间久了服务器会挂掉，64G-128G ssd
* 当redis物理内存使用超过内存总量的3/5就比较危险了，不使用swap
* 当达到最大内存时，会清空带有过期时间的key,即使key未到过期时间
* redis与db同步写的问题，先写db,后写redis,因为写内存上基本没问题

# 第2章 Redis部署与管理

## 2.1 源码编译安装Redis

### 2.1.1 下载安装Redis

* 下载

[root@node1 ~]# wget <http://download.redis.io/releases/redis-3.0.6.tar.gz>

* 解压编译安装

[root@node1 ~]# tar -xf redis-3.0.6.tar.gz

[root@node1 ~]# cd redis-3.0.6

[root@node1 redis-3.0.6]# less README

[root@node1 redis-3.0.6]# make MALLOC=jemalloc

##Redis 2.4版本之后，默认使用jemalloc来做内存管理；tengine也整合jemalloc，我们直接make

[root@node1 redis-3.0.6]# make PREFIX=/usr/local/redis-3.0.6 install

[root@node1 ~]# ln -s /usr/local/redis-3.0.6/ /usr/local/redis

[root@node1 ~]# tree /usr/local/redis/

/usr/local/redis/

└── bin

├── redis-benchmark #Redis性能测试工具，测试redis在当前系统下的配置的读写性能

├── redis-check-aof #更新日志检查

├── redis-check-dump #用于本地数据库rdb文件的检查

├── redis-cli #Redis命令行操作工具，当然也可以使用telnet根据存文本协议来操作

├── redis-sentinel -> redis-server #redis-server的软连接

└── redis-server #Redis服务的daemo启动程序

### 2.1.2 配置启动Redis服务

* 配置环境变量

[root@node1 ~]# echo 'PATH=/usr/local/redis/bin/:$PATH' >> /etc/profile

[root@node1 ~]# . /etc/profile

[root@node1 ~]# which redis-server

/usr/local/redis/bin/redis-server

* 创建配置文件

[root@node1 ~]# mkdir /usr/local/redis/conf

[root@node1 ~]# cp redis-3.0.6/redis.conf /usr/local/redis/conf/

* 启动Redis服务

[root@node1 ~]# redis-server –help 查看帮助

Usage: ./redis-server [/path/to/redis.conf] [options]

./redis-server - (read config from stdin)

./redis-server -v or --version

./redis-server -h or --help

./redis-server --test-memory <megabytes>

[root@node1 ~]# redis-server /usr/local/redis/conf/redis.conf &

5651:M 15 Feb 17:16:55.530 # WARNING overcommit\_memory is set to 0! Background save may fail under low memory condition. To fix this issue add 'vm.overcommit\_memory = 1' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.overcommit\_memory=1' for this to take effect.

5651:M 15 Feb 17:16:55.530 # WARNING you have Transparent Huge Pages (THP) support enabled in your kernel. This will create latency and memory usage issues with Redis. To fix this issue run the command 'echo never > /sys/kernel/mm/transparent\_hugepage/enabled' as root, and add it to your /etc/rc.local in order to retain the setting after a reboot. Redis must be restarted after THP is disabled.

出现警告信息，里面也告诉了我们解决方法

[root@node1 ~]# echo 'vm.overcommit\_memory=1' >> /etc/sysctl.conf

[root@node1 ~]# sysctl -p

[root@node1 ~]# echo never > /sys/kernel/mm/transparent\_hugepage/enabled

[root@node1 ~]# echo 'echo never > /sys/kernel/mm/transparent\_hugepage/enabled' >>/etc/rc.local

参数解释vm.overcommit\_memory=1

默认为0

0：当用户空间请求更多的内存时，内核尝试估算出剩余可用的内存

1：当设置这个参数为1时，内核允许超量使用内存直到用完为止，主要用于科学计算

2：设为2时，内核会使用一个决不过量使用内存的算法，即系统整个内存地址空间不能超过swap+50%的RAM值，50%参数的设定是在overcommit\_ration中设定

* 检查是否启动

[root@node1 ~]# lsof -i :6379

COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME

redis-ser 5911 root 4u IPv6 16696 0t0 TCP \*:6379 (LISTEN)

redis-ser 5911 root 5u IPv4 16698 0t0 TCP \*:6379 (LISTEN)

* 关闭Redis服务

[root@node1 ~]# redis-cli shutdown 关闭的也会保存数据的

[root@node1 ~]# redis-cli shutdown save

## 2.2 以脚本自动生成配置文件的方式安装Redis

### 2.2.1下载安装

[root@node2 ~]# wget <http://download.redis.io/releases/redis-3.0.6.tar.gz>

[root@node2 ~]# tar -xf redis-3.0.6.tar.gz

[root@node2 ~]# cd redis-3.0.6

[root@node2 redis-3.0.6]# make

[root@node2 redis-3.0.6]# make install

[root@node2 redis-3.0.6]# cd utils/

[root@node2 utils]# ./install\_server.sh 全部默认就行

Welcome to the redis service installer

This script will help you easily set up a running redis server

Please select the redis port for this instance: [6379]

Selecting default: 6379

Please select the redis config file name [/etc/redis/6379.conf]

Selected default - /etc/redis/6379.conf

Please select the redis log file name [/var/log/redis\_6379.log]

Selected default - /var/log/redis\_6379.log

Please select the data directory for this instance [/var/lib/redis/6379]

Selected default - /var/lib/redis/6379

Please select the redis executable path [/usr/local/bin/redis-server]

Selected config:

Port : 6379

Config file : /etc/redis/6379.conf

Log file : /var/log/redis\_6379.log

Data dir : /var/lib/redis/6379

Executable : /usr/local/bin/redis-server

Cli Executable : /usr/local/bin/redis-cli

Is this ok? Then press ENTER to go on or Ctrl-C to abort.

Copied /tmp/6379.conf => /etc/init.d/redis\_6379

Installing service...

Successfully added to chkconfig!

Successfully added to runlevels 345!

Starting Redis server...

Installation successful!

### 2.2.2 修改服务启动脚本

[root@node2 ~]# vim /etc/init.d/redis\_6379

#!/bin/sh

# SysV Init Information

# chkconfig: - 58 74

# description: redis\_6379 is the redis daemon.

#Configurations injected by install\_server below....

EXEC=/usr/local/bin/redis-server

CLIEXEC=/usr/local/bin/redis-cli

PIDFILE=/var/run/redis\_6379.pid

CONF="/etc/redis/6379.conf"

REDISPORT="6379"

status) 这段代码官方的有点小小问题，手动改下

if [ ! -f $PIDFILE ]

then

echo 'Redis is not running'

else

PID=$(cat $PIDFILE)

echo "Redis is running ($PID)"

fi

;;

### 2.2.3添加开机启动，并启动服务

[root@node2 ~]# chkconfig redis\_6379 on --level 2345

[root@node2 ~]# chkconfig redis\_6379 --list

redis\_6379 0:off 1:off 2:on 3:on 4:on 5:on 6:off

[root@node2 ~]# /etc/init.d/redis\_6379 status

Redis is running (3313)

[root@node2 ~]# /etc/init.d/redis\_6379 stop

Stopping ...

Redis stopped

[root@node2 ~]# /etc/init.d/redis\_6379 status

Redis is not running

## 2.3 Redis安全认证

### Redis密码认证

Redis中没有用户的概念，Redis速度非常快，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内进行上万次的密码尝试，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

[root@node1 ~]# cd /usr/local/redis/conf/

[root@node1 conf]# grep requirepass redis.conf

# If the master is password protected (using the "requirepass" configuration

# requirepass foobared

[root@node1 conf]# sed -i 's@# requirepass foobared@requirepass admin123@g' redis.conf 修改密码为admin123

修改后需要重启才能生效

[root@node1 ~]# redis-cli shutdown

[root@node1 ~]# redis-server /usr/local/redis/conf/redis.conf &

使用密码登录

* 方法一：不交互登录

[root@node1 ~]# redis-cli -a admin123

127.0.0.1:6379> set name admin

OK

127.0.0.1:6379> get name

"admin"

* 方法二：交互式登录

[root@node1 ~]# redis-cli

127.0.0.1:6379> get name

(error) NOAUTH Authentication required.

127.0.0.1:6379> auth admin123

OK

127.0.0.1:6379> get name

"admin"

### Redis权限控制

Redis没有mysql grant之类的设置权限，只能屏蔽或改名某些敏感的操作命令

* 屏蔽某个命令

[root@node1 ~]# vim /usr/local/redis/conf/redis.conf

rename-command set "" 屏蔽set命令

[root@node1 ~]# redis-cli -a admin123 shutdown

[root@node1 ~]# redis-server /usr/local/redis/conf/redis.conf &

[root@node1 ~]# redis-cli -a admin123

127.0.0.1:6379> set id 001

(error) ERR unknown command 'set' 已经找不到这个命令了

* 修改某个命令

[root@node1 ~]# vim /usr/local/redis/conf/redis.conf

rename-command get gets

[root@node1 ~]# redis-cli -a admin123 shutdown

[root@node1 ~]# redis-server /usr/local/redis/conf/redis.conf &

[root@node1 ~]# redis-cli -a admin123

127.0.0.1:6379> get name

(error) ERR unknown command 'get'

127.0.0.1:6379> gets name

"admin"

## Redis常用操作

* 客户端登录：

[root@node1 ~]# redis-cli -a admin123 加密码登录

127.0.0.1:6379>

* 插入数据：设置一个key-value对

127.0.0.1:6379> set id 001

OK

* 查询数据：取出key对应的value

127.0.0.1:6379> get id

"001"

* 删除键值：删除这个key及对应的value

127.0.0.1:6379> del id

(integer) 1

127.0.0.1:6379> get id

(nil)

* 验证键是否存在

127.0.0.1:6379> exists id

(integer) 0 #0表示不存在，1表示存在

* 远程连接

方法1：

[root@node1 ~]# redis-cli -h 172.10.25.26 -a admin123

172.10.25.26:6379>

方法2：

[root@node1 ~]# telnet 172.10.25.26 6379

Trying 172.10.25.26...

Connected to 172.10.25.26.

Escape character is '^]'.

auth admin123

+OK

get name

$5

admin

* 查看帮助

[root@node1 ~]# redis-cli –h

127.0.0.1:6379> help select

127.0.0.1:6379> help keys

127.0.0.1:6379> help @connection 按table可以查找@代表一个大的功能组

127.0.0.1:6379> help BITCOUNT

* 查看数据库

redis里面默认16个库0-15，是不可见的，使用select切换库，库的数量也可以到配置文件中修改

127.0.0.1:6379> select 1 切换到库1

OK

127.0.0.1:6379[1]>

* 刷新当前库和所有库

127.0.0.1:6379[1]> FLUSHDB 刷新当前

OK

127.0.0.1:6379[1]> FLUSHALL 刷新所有库

OK

* 查看redis所有状态信息

127.0.0.1:6379> info 查看所有

127.0.0.1:6379> info REPLICATION 查看某个功能组信息

# Replication

role:master

connected\_slaves:0

master\_repl\_offset:0

repl\_backlog\_active:0

repl\_backlog\_size:1048576

repl\_backlog\_first\_byte\_offset:0

repl\_backlog\_histlen:0

非交互的方式

[root@node1 ~]# redis-cli -a admin123 info CPU

# CPU

used\_cpu\_sys:26.12

used\_cpu\_user:8.98

used\_cpu\_sys\_children:0.00

used\_cpu\_user\_children:0.00

## 使用python操作redis

### 安装pip

[root@node1 ~]# yum install epel\* 安装epel源

[root@node1 ~]# yum install python-pip –y

[root@node1 ~]# pip install redis

[root@node1 ~]# pip install --upgrade pip

[root@node1 ~]# pip install redis –upgrade

### python操作redis

>>> import redis

>>> r = redis.Redis(host='192.168.1.25',port=6379,password='admin123',db=0)

>>> r.set('name','user1')

True

>>> r.get('name')

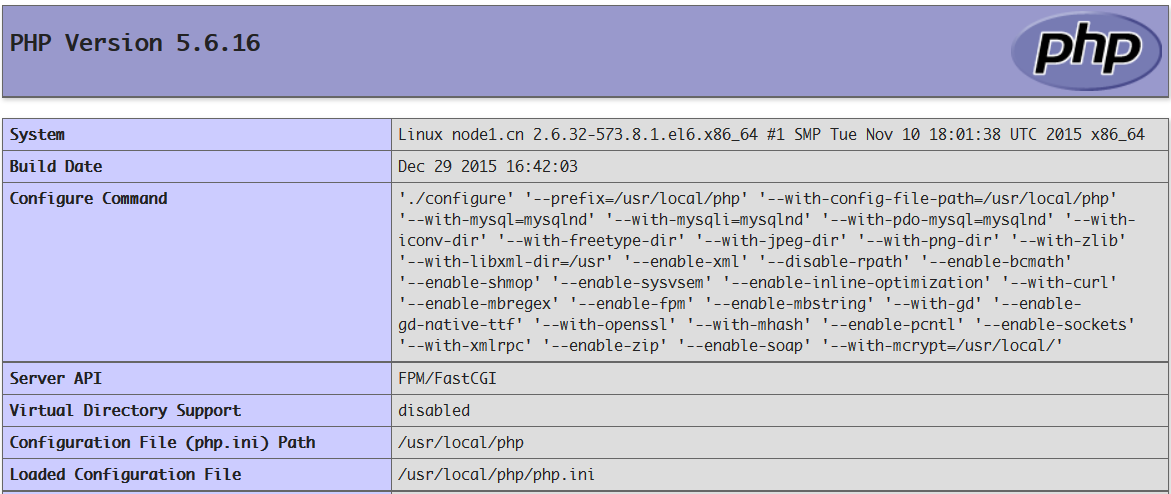
'user1'

>>> r.dbsize()

1L

## php操作redis

### 安装好lnmp或lamp环境



### 下载安装php-redis

* 下载

[root@node1 ~]# git clone <https://github.com/phpredis/phpredis.git>

* 安装

[root@node1 ~]# cd phpredis/

[root@node1 phpredis]# /usr/local/php/bin/phpize #动态扩展模块

[root@node1 phpredis]# ./configure --with-php-config=/usr/local/php/bin/php-config

[root@node1 phpredis]# make && make install

[root@node1 phpredis]# ls /usr/local/php/lib/php/extensions/no-debug-non-zts-20131226/

opcache.a opcache.so redis.so

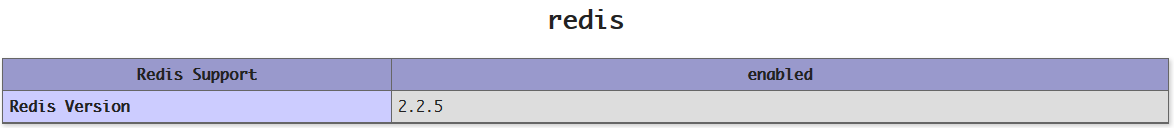
* 配置

[root@node1 ~]# vim /usr/local/php/php.ini

extension=redis.so 添加到902行，添加模块

* 查看是否ok

[root@node1 ~]# /etc/init.d/php-fpm restart



### 使用php操作redis

[root@node1 ~]# vim /website/redis.php

<?php

$redis = new Redis();

$redis ->connect('192.168.1.25',6379);

$redis ->auth('admin123');

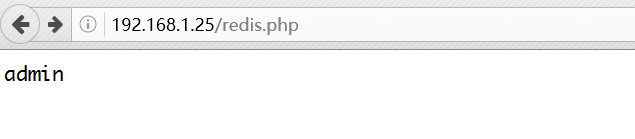
$redis ->flushall();

$redis ->set('name','admin');

$var = $redis ->get('name');

echo $var;

?>



## 2.7 Redis配置文件详解

42 daemonize no 缺省是否后台运行

46 pidfile /var/run/redis.pid

50 port 6379

59 tcp-backlog 511 tcp监听队列

69 # bind 127.0.0.1 监听本地ip地址

79 timeout 0 超时，0表示不超时

103 loglevel notice 日志级别，production生产级别

108 logfile "" 日志文件路径，不加默认以配置文件为准

123 databases 16 设置数据库的数，默认从0开始

145 # save "" 去掉注释就是不进行save操作，防止阻塞客户端请求

164 stop-writes-on-bgsave-error yes 出错的时候停止写

170 rdbcompression yes dump要不要压缩

182 dbfilename dump.rdb dump后的文件名

192 dir ./ dump后保存文件路径，默认当前路径

396 requirepass admin123 密码

428 # maxclients 10000 客户端并发

453 # maxmemory <bytes> 内存限制，超过内存就删除过期的key

509 appendonly no 是否启用aof

513 appendfilename "appendonly.aof" aof文件名

539 appendfsync everysec aof同步方式

580 auto-aof-rewrite-percentage 100 自动aof,数量和大小

581 auto-aof-rewrite-min-size 64mb

## 2.8 Redis管理

127.0.0.1:6379> CONFIG GET appendonly 获取当前参数

1) "appendonly"

2) "no"

127.0.0.1:6379> CONFIG SET appendonly yes 在线设置参数

127.0.0.1:6379> CONFIG RESETSTAT 重置 INFO 命令中的某些统计数据

OK

127.0.0.1:6379> CONFIG GET \* 查看所有参数

127.0.0.1:6379> DBSIZE 查看当前库的key数量

## 2.9 redis管理工具之phpredisadmin

<https://github.com/ErikDubbelboer/phpRedisAdmin>

* **安装apache,php,php-redis**

[root@node1 ~]# yum install httpd php php-redis php-common php-mbstring php-gd –y

[root@node1 ~]# vim /etc/httpd/conf/httpd.conf

AddType application/x-httpd-php .php

PHPIniDir "/etc/php.ini"

DirectoryIndex index.html index.html.var index.php

[root@node1 ~]# service httpd restart

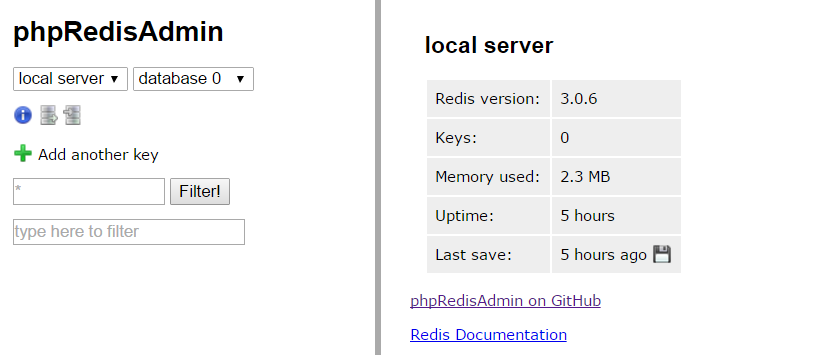
* **下载phpredisadmin**

[root@node1 ~]# cd /var/www/html/

[root@node1 html]# git clone <https://github.com/erikdubbelboer/phpRedisAdmin.git>

[root@node1 html]# cd phpRedisAdmin/

[root@node1 phpRedisAdmin]# git clone https://github.com/nrk/predis.git vendor



可以配置管理多台机器

[root@node1 admin]# cd includes/

[root@node1 includes]# cp config.sample.inc.php config.inc.php

[root@node1 includes]# vim config.inc.php

array( 按这种方式添加就行

'name' => '6380', // Optional name.

'host' => '192.168.1.24',

'port' => 6380,

'filter' => '\*',

),

## 2.10 redis管理工具之rdbtools

[root@node1 ~]# yum install python-pip –y

[root@node1 ~]# pip install rdbtools

[root@node1 ~]# rdb -c memory dump.rdb > memory.csv

[root@node1 ~]# sz memory.csv

<https://github.com/sripathikrishnan/redis-rdb-tool>

# 第3章 Redis数据类型

## 3.1 key/value介绍

redis key值是二进制安全的，这意味着可以用任何二进制序列作为key值，从形如“123，admin”的简单数字字符串到一个jpg文件的内容都可以，空字符串也是有效的key值

建议：

key--->-object-type:id:field 长度10-20

value--->string 不要超过2k,set sortedset元素不要超过5000，根据内容长度规划不同的实例端口

## 3.2 string字符串数据类型

这是redis最简单的数据类型之一，如果只使用这种数据类型，那么redis就是一个持久化的memcached服务器（注：memcached的数据仅保存在内存中，服务器重启后，数据将丢失）当然redis对string类型的功能比memcached还是多很多的

* **常规的string字符串类型**

127.0.0.1:6379> set name sky

OK

127.0.0.1:6379> get name

"sky"

value值可以是任何类型的字符串（包括二进制数据），比如你可以在一个键下保存一张jpg图片，但值的长度不能超过1GB

127.0.0.1:6379> APPEND name sun 追加

(integer) 6

127.0.0.1:6379> get name

"skysun"

* **string类型也可以用来存储数字，并支持对数字的加减操作**

127.0.0.1:6379> set counter 1

OK

127.0.0.1:6379> INCR counter

(integer) 2

127.0.0.1:6379> INCR counter

(integer) 3

127.0.0.1:6379> INCR counter

(integer) 4

127.0.0.1:6379> INCRBY counter 2

(integer) 6

127.0.0.1:6379> INCRBY counter 3

(integer) 9

127.0.0.1:6379> DECR counter

(integer) 8

127.0.0.1:6379> DECR counter

(integer) 7

127.0.0.1:6379> DECRBY counter 3

(integer) 4

127.0.0.1:6379> GET counter

"4"

127.0.0.1:6379> INCRBYFLOAT key 0.1 浮点型

"0.1"

127.0.0.1:6379> INCRBYFLOAT key 0.1

"0.2"

127.0.0.1:6379> INCRBYFLOAT key 0.1

"0.3"

* **为key设置新值并且返回原值**

对字符串，使用getset命令为key设置新值并且返回原值，

127.0.0.1:6379> set name admin

OK

127.0.0.1:6379> get name

"admin"

127.0.0.1:6379> getset name root

"admin"

127.0.0.1:6379> get name

"root"

* **string类型还支持批量读写操作**

127.0.0.1:6379> mset name admin age 25 sex male

OK

127.0.0.1:6379> mget name age sex

1) "admin"

2) "25"

3) "male"

127.0.0.1:6379> append name "test" 追加内容

(integer) 9

127.0.0.1:6379> get name

"admintest"

127.0.0.1:6379> strlen name 查看字符数

(integer) 9

127.0.0.1:6379> substr name 0 4 截取字符

"admin"

* **字符串类型查看帮助help**

127.0.0.1:6379> help @string

## 3.3 list 列表类型

实际上就是数组，一般意义上讲，列表就是有序元素的序列，但用数组实现list和用linked list实现的list,在属性方面大不相同

redis list基于linked list实现的list,这意味着即使在一个list中有数百万个元素，在头部或尾部添加一个元素的操作，其时间复杂度也是常数级别的，用lpush命令在十个元素的list头部添加新元素，和在千万元素list头部添加新元素的速度相同。

在数组实现的list中利用索引访问元素的速度极快，而同样的操作在linked list实现的list上没有那么快

redis list用linked list实现的原因是：对于数据库系统来说，至关重要的特性是能非常快的在很大的列表上添加元素，另一个重要因素是redis list 能在常数时间取得常数长度

lpush命令可向list的左边（头部）添加一个新元素

rpush命令可向list右边（尾部）添加一个新元素

lrange命令可从list中取出一定范围的元素

127.0.0.1:6379> help @list 帮助命令

127.0.0.1:6379> lpush students zhangs 创建

(integer) 1

127.0.0.1:6379> lpush students lis bob

(integer) 3

127.0.0.1:6379> llen students

(integer) 3

127.0.0.1:6379> lrange students 0 2 显示

1) "bob"

2) "lis"

3) "zhangs"

127.0.0.1:6379> lpop students 从第一个开始删

"bob"

127.0.0.1:6379> lrange students 0 2

1) "lis"

2) "zhangs"

127.0.0.1:6379> lrem students 2 zhangs 指定value移除

(integer) 1

127.0.0.1:6379> lrange students 0 1

1) "lis"

127.0.0.1:6379> lpush students a b c

(integer) 4

127.0.0.1:6379> linsert students before lis bob 指定位置添加

(integer) 5

127.0.0.1:6379> lrange students 0 4

1) "c"

2) "b"

3) "a"

4) "bob"

5) "lis"

127.0.0.1:6379> rpush students d

(integer) 6

127.0.0.1:6379> lrange students 0 5

1) "c"

2) "b"

3) "a"

4) "bob"

5) "lis"

6) "d"

## 3.4 set集合类型

redis的set是string类型的无序集合。set元素最大可以包含(2的32次方-1)个元素。set的是通过hash table实现的，所以添加，删除，查找的复杂度都是O(1)。hash table会随着添加或者删除自动的调整大小。需要注意的是调整hash table大小时候需要同步（获取写锁）会阻塞其他读写操作。关于set集合类型除了基本的添加删除操作，其他有用的操作还包含集合的取并集(union)，交集(intersection)，差集(difference)。通过这些操作可以很容易的实现

127.0.0.1:6379> help @set

127.0.0.1:6379> sadd users s1 添加无序不重复的

(integer) 1

127.0.0.1:6379> sadd users s2 s3

(integer) 2

127.0.0.1:6379> smembers users

1) "s3"

2) "s1"

3) "s2"

127.0.0.1:6379> sismember users s1 检查某个元素是否存在

(integer) 1

127.0.0.1:6379> sismember users s4

(integer) 0

|  |  |
| --- | --- |
| sadd key member | 添加一个string元素到,key对应的set集合中，成功返回1,如果元素以及在集合中返回0,key对应的set不存在返回错误 |
| srem key member | 从key对应set中移除给定元素，成功返回1，如果member在集合中不存在或者key不存在返回0，如果key对应的不是set类型的值返回错误 |
| spop key | 删除并返回key对应set中随机的一个元素,如果set是空或者key不存在返回nil |
| srandmember key | 同spop，随机取set中的一个元素，但是不删除元素 |
| smove srckey dstkey member | 从srckey对应set中移除member并添加到dstkey对应set中，整个操作是原子的。成功返回1,如果member在srckey中不存在返回0，如果key不是set类型返回错误 |
| scard key | 返回set的元素个数，如果set是空或者key不存在返回0 |
| sismember key member | 判断member是否在set中，存在返回1，0表示不存在或者key不存在 |
| sinter key1 key2...keyN | 返回所有给定key的交集 |
| sinterstore dstkey key1...keyN | 同sinter，但是会同时将交集存到dstkey下 |
| sunion key1 key2...keyN | 返回所有给定key的并集 |
| sunionstore dstkey key1...keyN | 同sunion，并同时保存并集到dstkey下 |
| sdiff key1 key2...keyN | 返回所有给定key的差集 |
| sdiffstore dstkey key1...keyN | 同sdiff，并同时保存差集到dstkey下 |
| smembers key | 返回key对应set的所有元素，结果是无序的 |

## 3.5 sorted set

sorted set和set一样也是string类型元素的集合，不同的是每个元素都会关联一个double类型的score。sorted set的实现是skip list（跳表）和hash table的混合体当元素被添加到集合中时，一个元素到score的映射被添加到hash table中，所以给定一个元素获取score的开销是O(1)，另一个score到元素的映射被添加到skip list并按照score排序，所以就可以有序的获取集合中的元素。添加，删除操作开销都是O(log(N))和skip list的开销一致，redis的skip list实现用的是双向链表，这样就可以逆序从尾部取元素。sorted set最经常的使用方式应该是作为索引来使用。我们可以把要排序的字段作为score存储，对象的id当元素存储。

127.0.0.1:6379> help @sorted\_set

|  |  |
| --- | --- |
| zadd key score member | 添加元素到集合，元素在集合中存在则更新对应score |
| zrem key member | 删除指定元素，1表示成功，如果元素不存在返回0 |
| zincrby key incr member | 增加对应member的score值，然后移动元素并保持skip list保持有序。返回更新后的score值 |
| zrank key member | 返回指定元素在集合中的排名（下标）,集合中元素是按score从小到大排序的 |
| zrevrank key member | 同上,但是集合中元素是按score从大到小排序 |
| zrange key start end | 类似lrange操作从集合中去指定区间的元素。返回的是有序结果 |
| zrevrange key start end | 同上，返回结果是按score逆序的 |
| zrangebyscore key min max | 返回集合中score在给定区间的元素 |
| zcount key min max | 返回集合中score在给定区间的数量 |
| zcard key | 返回集合中元素个数 |
| zscore key element | 返回给定元素对应的score |
| zremrangebyrank key min max | 删除集合中排名在给定区间的元素 |
| zremrangebyscore key min max | 删除集合中score在给定区间的元素 |

127.0.0.1:6379> zadd days 0 mon

(integer) 1

127.0.0.1:6379> zadd days 1 tue

(integer) 1

127.0.0.1:6379> zadd days 2 wed

(integer) 1

127.0.0.1:6379> zadd days 3 thu

(integer) 1

127.0.0.1:6379> zadd days 4 fri

(integer) 1

127.0.0.1:6379> zadd days 5 sat

(integer) 1

127.0.0.1:6379> zadd days 6 sun

(integer) 1

127.0.0.1:6379> zrange days 0 6 显示

1) "mon"

2) "tue"

3) "wed"

4) "thu"

5) "fri"

6) "sat"

7) "sun"

127.0.0.1:6379> zscore days wed 查看排序

"2"

127.0.0.1:6379> zrangebyscore days 3 6 指定排序

1) "thu"

2) "fri"

3) "sat"

4) "sun"

127.0.0.1:6379> zrevrange days 0 6 方向排序

1) "sun"

2) "sat"

3) "fri"

4) "thu"

5) "wed"

6) "tue"

7) "mon"

## 3.6 hash

redis hash是一个string类型的field和value的映射表。它的添加，删除操作都是O(1)（平均）。hash特别适合用于存储对象。相较于将对象的每个字段存成单个string类型。将一个对象存储在hash类型中会占用更少的内存，并且可以更方便的存取整个对象。内存的原因是因为它有存储优化机制

|  |  |
| --- | --- |
| set key field value | 设置hash field为指定值，如果key不存在，则先创建 |
| hget key field | 获取指定的hash field |
| hmget key filed1....fieldN | 获取全部指定的hash filed |
| hmset key filed1 value1 ... filedN valueN | 同时设置hash的多个field |
| hincrby key field integer | 将指定的hash filed 加上给定值 |
| hexists key field | 测试指定field是否存在 |
| hdel key field | 删除指定的hash field |
| hlen key | 返回指定hash的field数量 |
| hkeys key | 返回hash的所有field |
| hvals key | 返回hash的所有value |
| hgetall | 返回hash的所有filed和value |

# 第4章 Redis多实例

**背景：**

**我们redis一般保存的数据库不能太大，大概在20-30G之间，数据量太大的会严重影响效率，一般slave分开**

## 4.1 在redis中保存数据库

* **创建存放库的目录**

[root@node01 ~]# mkdir -p /data/redis/6379

* **修改redis配置文件，主要是设置dump路径**

[root@node01 ~]# vim /usr/local/redis/conf/redis.conf

dir /data/redis/6379

* **重启redis，并添加数据保存**

[root@node01 ~]# redis-cli -a admin123 shutdown save 关闭redis并保存数据

[root@node01 ~]# redis-server /usr/local/redis/conf/redis.conf &

[root@node01 ~]# redis-cli -a admin123

127.0.0.1:6379> set name admin

OK

127.0.0.1:6379> get name

"admin"

127.0.0.1:6379> save 保存数据

3682:M 05 Mar 13:48:24.687 \* DB saved on disk

OK

[root@node01 ~]# ll /data/redis/6379 已经有保存的数据了

total 4

-rw-r--r-- 1 root root 32 Mar 5 13:48 dump.rdb

## 4.2 部署redis多实例

**其实这个非常简单，只要不同的配置文件，不同的端口，不同的数据存放目录就ok**

**下面我们来添加两个redis多实例**

* **创建多实例的目录**

[root@node01 ~]# mkdir -p /data/6380/data 创建数据存放目录

[root@node01 ~]# mkdir -p /data/6381/data

[root@node01 ~]# cp /usr/local/redis/conf/redis.conf /data/6380/ 复制配置文件

[root@node01 ~]# cp /usr/local/redis/conf/redis.conf /data/6381/

* **修改配置文件，设置保存目录和开启aof日志，修改端口**

[root@node01 ~]# vim /data/6380/redis.conf

pidfile /var/run/redis\_6380.pid

logfile /var/log/redis\_6380.log

port 6380

dir /data/6380/data

appendonly yes

[root@node01 ~]# vim /data/6381/redis.conf

pidfile /var/run/redis\_6381.pid

logfile /var/log/redis\_6381.log

port 6381

dir /data/6381/data

appendonly yes

* **启动多实例**

[root@node01 ~]# redis-server /data/6380/redis.conf &

[root@node01 ~]# redis-server /data/6381/redis.conf &

[root@node01 ~]# lsof -i :6380

COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME

redis-ser 3964 root 4u IPv6 43174 0t0 TCP \*:6380 (LISTEN)

redis-ser 3964 root 5u IPv4 43176 0t0 TCP \*:6380 (LISTEN)

[root@node01 ~]# lsof -i :6381

COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME

redis-ser 3967 root 4u IPv6 43184 0t0 TCP \*:6381 (LISTEN)

redis-ser 3967 root 5u IPv4 43186 0t0 TCP \*:6381 (LISTEN)

* **登录多实例测试**

[root@node01 ~]# redis-cli -p 6380 -a admin123

127.0.0.1:6380> set name 6380

OK

127.0.0.1:6380> get name

"6380"

[root@node01 ~]# redis-cli -p 6381 -a admin123

127.0.0.1:6381> set name 6381

OK

127.0.0.1:6381> get name

"6381"

# 第5章 Redis主从

## 5.1 redis主从同步原理

Redis的主从复制功能非常强大，一个master可以拥有多个slave，而一个slave又可以拥有多个slave，如此下去，形成了强大的多级服务器集群架构。

下面是关于redis主从复制的一些特点：

1、master可以有多个slave

2、除了多个slave连到相同的master外，slave也可以连接其他slave形成图状结构

3、主从复制不会阻塞master。也就是说当一个或多个slave与master进行初次同步数据时，master可以继续处理client发来的请求。相反slave在初次同步数据时则会阻塞不能处理client的请求。

4、主从复制可以用来提高系统的可伸缩性,我们可以用多个slave 专门用于client的读请求，比如sort操作可以使用slave 来处理。也可以用来做简单的数据冗余

5、可以在master禁用数据持久化，只需要注释掉master 配置文件中的所有save配置，然后只在slave上配置数据持久化。

## 5.2 redis主从同步的过程

1、配置好slave服务器连接master后，slave会建立和master的链接，然后发送sync命令

2、无论是第一次同步建立的连接还是连接断开后的重新连接，master都会启动一个后台进程，将数据库快照保存到磁盘文件中，同时master主进程会开始收集新的写命令并缓存起来

3、当后台进程完成写磁盘文件后，master就将快照文件发送给slave,slave将文件保存到磁盘上，然后加载到内存将数据库快照恢复到slave上

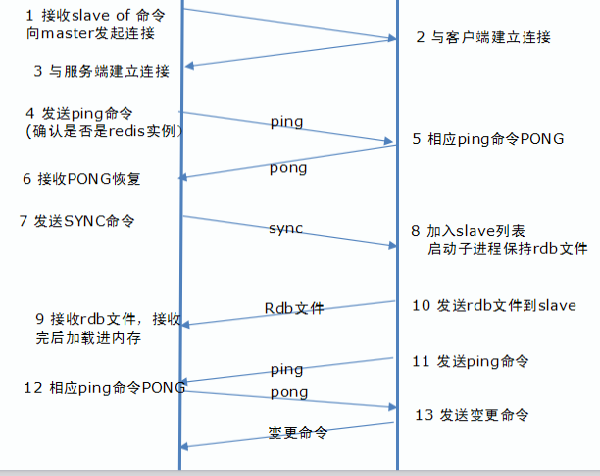
4、slave完成快照文件恢复后，master就会把缓存的命令都转发给slave,slave更新内存数据库

5、后续master收到的写命令都会通过开始建立的连接发送给slave,从master到slave的同步数据的命令和从client到master发送的命令使用相同的协议格式

6、当master和slave的连接断开时slave可以自动重新建立连接。如果master同时收到多个 slave发来的同步连接命令，只会使用启动一个进程来写数据库镜像，然后发送给所有slave



slave服务连接到master服务器---->slave服务器发送sync命令---->master服务器备份数据库到.rdb文件----->master服务器把.rdb文件传输给slave服务器---->slave服务器把.rdb文件数据导入到数据库中



## 5.3 redis主从部署

主从架构可以本机多实例数据库实现，也可以异机多实例之间实现

只需要在slave服务器上进行，

指定slave需要连接的redis服务器（可能是master，也可能是slave）

通过配置redis.conf实现

    只需要配置slave就可以

* **实验环境**

本机多实例

redis-master 172.10.25.26 6379

redis-slave 172.10.25.26 6380

* **修改salve配置文件**

[root@node01 ~]# vim /data/6380/redis.conf

slaveof 172.10.25.26 6379 指定master，也可以使用域名的方式

masterauth admin123 安全认证

其他参数说明：

参数说明：

slave-serve-stale-data yes 当slave丢失master或者同步正在进行时，如果发生对slave的服务请求

slave-serve-stale-data 设置为yes则slave依然正常提供服务，设置为no则slave返回client错误：“SYNC with master in progress”

repl-ping-slave-period 10 salve发送pings到master的时间间隔

repl-timeout 60 IO超时时间

repl-backlog-size 1mb backlog大小，增量同步

repl-backlog-ttl 3600 backlog生存期，当主库和从库断开时，backlog在内存中多长时间释放

slave-priority 100 slave的优先级，master挂掉，根据优先级将其中一台slave提升为master

* **重启slave，测试主从**

[root@node01 ~]# redis-cli -a admin123 -p 6380 shutdown

[root@node01 ~]# redis-server /data/6380/redis.conf & 输出一些信息

4285:S 05 Mar 17:57:35.198 # Server started, Redis version 3.0.6

4285:S 05 Mar 17:57:35.198 \* DB loaded from append only file: 0.000 seconds

4285:S 05 Mar 17:57:35.198 \* The server is now ready to accept connections on port 6380

4285:S 05 Mar 17:57:35.198 \* Connecting to MASTER 172.10.25.26:6379

4285:S 05 Mar 17:57:35.198 \* MASTER <-> SLAVE sync started

分别登录master和slave

[root@node01 ~]# redis-cli -a admin123 -p 6379

[root@node01 ~]# redis-cli -a admin123 -p 6380

127.0.0.1:6380> monitor 监听从状态，每隔10s ping一下

OK

1457172013.507909 [0 172.10.25.26:6379] "PING"

1457172023.629103 [0 172.10.25.26:6379] "PING"

master插入数据，查看slave

127.0.0.1:6379> set name admin

OK

127.0.0.1:6380> monitor 监控状态

OK

127.0.0.1:6379> set id 0001

OK

1457172173.449099 [0 172.10.25.26:6379] "SELECT" "0"

1457172173.449109 [0 172.10.25.26:6379] "set" "name" "admin"

1457172236.446206 [0 172.10.25.26:6379] "PING"

1457172237.598070 [0 172.10.25.26:6379] "set" "id" "0001"

127.0.0.1:6379> get id get操作不显示

"0001"

1457172237.598070 [0 172.10.25.26:6379] "set" "id" "0001"

1457172246.575216 [0 172.10.25.26:6379] "PING"

1457172256.672216 [0 172.10.25.26:6379] "PING"

查看slave是否有数据

127.0.0.1:6380> get name 已经有数据

"admin"

127.0.0.1:6380> get id

"0001"

127.0.0.1:6380> set sex 1 slave不能写数据，只读的

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

* **查看主从同步状态**

127.0.0.1:6380> info replication

# Replication

role:slave

master\_host:172.10.25.26

master\_port:6379

master\_link\_status:up

master\_last\_io\_seconds\_ago:8

master\_sync\_in\_progress:0

slave\_repl\_offset:1041

slave\_priority:100

slave\_read\_only:1

connected\_slaves:0

master\_repl\_offset:0

repl\_backlog\_active:0

repl\_backlog\_size:1048576

repl\_backlog\_first\_byte\_offset:0

repl\_backlog\_histlen:0

127.0.0.1:6379> info replication

# Replication

role:master

connected\_slaves:1

slave0:ip=172.10.25.26,port=6380,state=online,offset=1083,lag=1

master\_repl\_offset:1083

repl\_backlog\_active:1

repl\_backlog\_size:1048576

repl\_backlog\_first\_byte\_offset:2

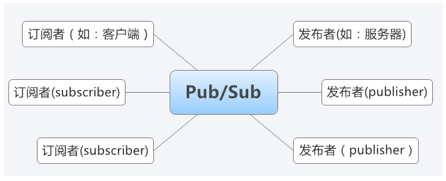
repl\_backlog\_histlen:1082

# 第6章 Redis高级特性

## 消息发布订阅

### 6.1.1 Pub/Sub功能

Pub/Sub功能（means Publish, Subscribe）即发布及订阅功能。基于事件的系统中，Pub/Sub是目前广泛使用的通信模型，它采用事件作为基本的通信机制，提供大规模系统所要求的松散耦合的交互模式：订阅者(如客户端)以事件订阅的方式表达出它有兴趣接收的一个事件或一类事件；发布者(如服务器)可将订阅者感兴趣的事件随时通知相关订阅者。

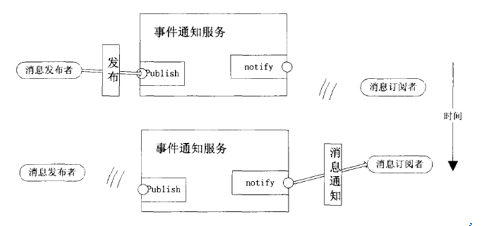


### 6.1.2 Pub/Sub机制



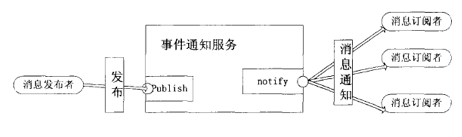
**1、时间非耦合：**

发布者和订阅者不必同时在线，它们不必同时参与交互



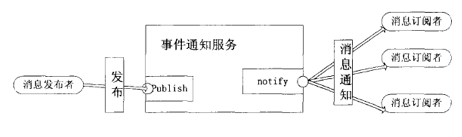
**2、空间非耦合：**

发布者和订阅者不必相互知道对方所在的位置。发布者通过事件服务发布事件，订阅者通过事件服务间接获得事件。发布者和订阅者不需要拥有直接到对方的引用，也不必知道有多少个订阅者或者是发布者参与交互。



**3、同步非耦合：**

发布者/订阅者是异步模式。发布者可不断地生产事件，而订阅者(通过一个回调)则可异步地得到产生事件的通知



**订阅**

[root@node1 ~]# redis-cli -a admin123

127.0.0.1:6379> subscribe first

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

1) "subscribe"

2) "first"

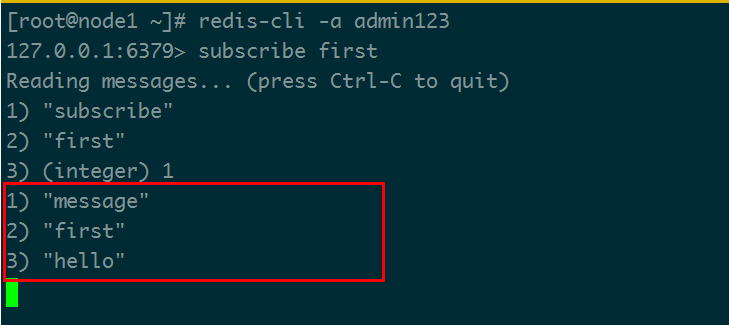
3) (integer) 1

**发布**

127.0.0.1:6379> publish first hello

(integer) 1

**订阅的终端查看**



**取消订阅：**

127.0.0.1:6379> UNSUBSCRIBE first

1) "unsubscribe"

2) "first"

3) (integer) 0

**批量订阅：**

127.0.0.1:6379> psubscribe channel\*

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

1) "psubscribe"

2) "channel\*"

3) (integer) 1

**发布：**

127.0.0.1:6379> publish channelone hello

(integer) 1

127.0.0.1:6379> publish channelone world

(integer) 1



**取消批量订阅：**

127.0.0.1:6379> PUNSUBSCRIBE channel\*

1) "punsubscribe"

2) "channel\*"

3) (integer) 0

## 6.2 redis数据过期设置及过期机制

R，edis支持按key设置过期时间，过期后值将被删除（在客户端看来是不删除的）在Redis中可以使用 **EXPIRE** 命令设置一个键的存活时间(ttl: time to live)，过了这段时间，该键就会自动被删除

用TTL命令可以获取某个key值的过期时间（-1表示永不过期）

**设置key过期：**

127.0.0.1:6379> FLUSHDB清空当前数据库

OK

127.0.0.1:6379> set name admin

127.0.0.1:6379> TTL name 查看TTL值

(integer) -1

127.0.0.1:6379> help expire 命令帮助

EXPIRE key seconds

summary: Set a key's time to live in seconds

since: 1.0.0

group: generic

127.0.0.1:6379> EXPIRE name 5 设置5s过期

(integer) 1

127.0.0.1:6379> TTL name查看

(integer) -2

127.0.0.1:6379> get name 没有了

(nil)

**指定时间戳过期：**

[root@node1 ~]# date +%s 查看当前时间戳

1460875072

[root@node1 ~]# date -d@1460876898 "+%Y/%m/%d %H:%M:%S" 时间戳转换

2016/04/17 15:08:18

127.0.0.1:6379> set name admin

OK

127.0.0.1:6379> get name

"admin"

127.0.0.1:6379> expireat name 1460875557 指定过期时间，以时间戳表示

(integer) 1

127.0.0.1:6379> TTL name

(integer) 72

127.0.0.1:6379> TTL name 过期

(integer) -2

127.0.0.1:6379> get name

(nil)

如果想取消某个键的过期时间，可以使用 **PERSIST** 命令，用法如下：

PERSIST key

清除成功返回1，失败返回0.

**过期机制：**

redis对过期key采用了lazy expiration,在访问key的时候判定key是否过期，如果过期则进行过期处理

过期的key没有被访问可能不会被删除

其次，每秒对volatile keys进行抽样测试，如果有过期key，那么对所有过期key进行处理

## Redis事务性

redis本身支持一些简单的组合型命令，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：

1、事务是一个单独的隔离操作：事务中的所有命令都会序列化、按顺序地执行、事务在执行的过程中，不会被其他客户端发送 来的命令请求所打断。

2、事务是一个原子操作：事务中的命令要么全部被执行，要么全部都不执行。

**一个事务从开始到执行会经历以下三个阶段：**

1：开始事务。

2：命令入队。

3：执行事务。

**redis事务的命令：**

MULTI 标记一个事务块的开始。

EXEC 执行所有事务块内的命令。

DISCARD 取消事务，放弃执行事务块内的所有命令

UNWATCH 取消 WATCH 命令对所有 key 的监视。

WATCH key [key ...] 监视一个(或多个) key ，如果在事务执行之前这个(或这些) key 被其他命令所改动，那么事务将被打断。

127.0.0.1:6379> help @transactions 事务的所有命令帮助

http://ju.outofmemory.cn/entry/81786

127.0.0.1:6379> MULTI

OK

127.0.0.1:6379> set name admin

QUEUED

127.0.0.1:6379> get name

QUEUED

127.0.0.1:6379> sadd names tom alice

QUEUED

127.0.0.1:6379> SMEMBERS names

QUEUED

127.0.0.1:6379> EXEC 提交事务

1) OK

2) "admin"

3) (integer) 2

4) 1) "alice"

2) "tom"

127.0.0.1:6379> MULTI

OK

127.0.0.1:6379> PING

QUEUED

127.0.0.1:6379> SET say Hello

QUEUED

127.0.0.1:6379> DISCARD 取消事务

OK

## 持久化

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磁盘来保证持久化。

redis支持四种持久化方式：

1、Snapshotting（快照）也是默认方式

2、Append-only file（缩写aof）的方式

3、虚拟内存方式

4、diskstore方式

### （一）Snapshotting

快照是默认的持久化方式。这种方式是就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

下面是默认的快照保存配置：

vim /usr/local/redis/conf/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

快照保存过程：

1. redis调用fork,现在有了子进程和父进程

2. 父进程继续处理client请求，子进程负责将内存内容写入到临时文件。由于os的写时复制机制（copy on write)父子进 程会共享相同的物理页面，当父进程处理写请求时os会为父进程 要修改的页面创建副本，而不是写共享的页面，所以 子进程的地址空间内的数据是fork时刻整个数据库的一个快照

3. 当子进程将快照写入临时文件完毕后，用临时文件替换原来的快照文件，然后子进程退出（fork一个进程入内在也被复制了 ，即内存会是原来的两倍）

client 也可以使用save或者bgsave命令通知redis做一次快照持久化。save操作是在主线程中保存快照的，由于redis是用一个主线程来处理所有 client的请求，这种方式会阻塞所有client请求。所以不推荐使用。另一点需要注意的是，每次快照持久化都是将内存数据完整写入到磁盘一次，并不是增量的只同步脏数据。如果数据量大的话，而且写操作比较多，必然会引起大量的磁盘io操作，可能会严重影响性能。

127.0.0.1:6379> keys \*

1) "names"

2) "name"

3) "name1"

127.0.0.1:6379> flushall 清空所有库

OK

127.0.0.1:6379> keys \*

(empty list or set)

127.0.0.1:6379> set name admin

OK

127.0.0.1:6379> save 生成.rdb

OK

ll --full-time /data/redis/ 对比两次的时间 ll --full-time 显示详细时间

total 4

-rw-r--r-- 1 root root 32 2015-12-13 03:58:49.640909646 +0800 dump.rdb

ll --full-time /data/redis/

total 4

-rw-r--r-- 1 root root 32 2015-12-13 04:00:12.359909390 +0800 dump.rdb

如果没有save，shutdown redis后是否会保存

127.0.0.1:6379> set ke v2

OK

127.0.0.1:6379> BGSAVE 保存

ll --full-time /data/redis/ 没保存之前的时间

-rw-r--r-- 1 root root 39 2015-12-19 11:54:36.521987617 +0800 dump.rdb

ll --full-time /data/redis/ 保存之后的时间

-rw-r--r-- 1 root root 46 2015-12-19 11:55:52.185989029 +0800 dump.rdb

127.0.0.1:6379> set k3 v3

OK

127.0.0.1:6379> exit

redis-cli -a admin123 shutdown 关闭服务

ll --full-time /data/redis/ 查看是否保存

total 4

-rw-r--r-- 1 root root 55 2015-12-19 12:00:24.003988648 +0800 dump.rdb

redis-server /usr/local/redis/conf/redis.conf & 再启动服务

127.0.0.1:6379> get k3 查看

"v3"

**问题：**

另外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。如果应用要求不能丢失任何修改的话，可以采用aof持久化方式

### （二）Append-only file<追加式的操作日志记录>

aof 比快照方式有更好的持久化性，是由于在使用aof持久化方式时，redis会将每一个收到的写命令都通过write函数追加到文件中(默认是appendonly.aof)。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。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。不过我们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。有三种方式如下（默认是：每秒fsync一次）：

vim /usr/local/redis/conf/redis.conf

appendonly yes #启用aof持久化方式

# appendfsync always #每次收到写命令就立即强制写入磁盘，最慢的，但是保证完全的持久化，不推荐使用

appendfsync everysec #每秒钟强制写入磁盘一次，在性能和持久化方面做了很好的折中，推荐

# appendfsync no #完全依赖os，性能最好,持久化没保证

redis-cli -a admin123 shutdown

redis-server /usr/local/redis/conf/redis.conf & 修改启用aof，重启redis

ll /data/redis/

total 4

-rw-r--r-- 1 root root 0 Dec 19 13:44 appendonly.aof 生成了aof文件

-rw-r--r-- 1 root root 55 Dec 19 13:43 dump.rdb

redis-cli -a admin123

127.0.0.1:6379> set name admin 插入数据测试

OK

127.0.0.1:6379> get name

"admin"

127.0.0.1:6379> quit

ll /data/redis/

total 8

-rw-r--r-- 1 root root 57 Dec 19 13:45 appendonly.aof 已经有数据了

-rw-r--r-- 1 root root 55 Dec 19 13:43 dump.rdb

**工作流程**

aof 的方式也同时带来了另一个问题。持久化文件会变的越来越大。

例如我们调用incr test命令100次，文件中必须保存全部的100条命令，其实有99条都是多余的。因为要恢复数据库的状态其实文件中保存一条set test 100就够了。为了压缩aof的持久化文件。redis提供了bgrewriteaof命令。收到此命令redis将使用与快照类似的方式将内存中的数据以命令的方式保存到临时文件中，最后替换原来的文件。具体过程如下：

1. redis调用fork ，现在有父子两个进程

2. 子进程根据内存中的数据库快照，往临时文件中写入重建数据库状态的命令

3. 父进程继续处理client请求，除了把写命令写入到原来的aof文件中。同时把收到的写命令缓存起来。这样就能保证如果子 进程重写失败的话并不会出问题。

4. 当子进程把快照内容写入已命令方式写到临时文件中后，子进程发信号通知父进程。然后父进程把缓存的写命令也写入到临 时文件。

5. 现在父进程可以使用临时文件替换老的aof文件，并重命名，后面收到的写命令也开始往新的aof文件中追加。

需要注意的是重写aof文件的操作，并没有读取旧的aof文件，而是将整个内存中的数据库内容用命令的方式重写了一个新的aof文件，这点和快照有点类似。

**通过aof记录恢复数据库：**

127.0.0.1:6379> get name 查看当前

"admin"

127.0.0.1:6379> del name 删除

(integer) 1

127.0.0.1:6379> get name

(nil)

127.0.0.1:6379> keys \* 已经没有数据了

(empty list or set)

恢复，修改

vim /data/redis/appendonly.aof 把关于del的记录删除

\*2

$3

del

$4

name

删除后重启redis

redis-cli -a admin123 shutdown

redis-server /usr/local/redis/conf/redis.conf &

redis-cli -a admin123

127.0.0.1:6379> keys \* 数据又回来了

1) "name"

127.0.0.1:6379> get name

"admin"

**智能优化压缩aof文件**

127.0.0.1:6379> BGREWRITEAOF 将重复日志刷新掉

对应的参数

vim /usr/local/redis/conf/redis.conf

auto-aof-rewrite-percentage 100 日志的百分比，设置0的话就禁用自动重写功能

auto-aof-rewrite-min-size 64mb aof大小 ，达到64m的时候就自动重写，智能整理

其他两个持久化用的比较少，甚至新版本已经取消了这块的功能

# 第7章 redis优化

## 7.1内存管理优化

Redis Hash是value内部为一个HashMap，如果该Map的成员数比较少，则会采用类似一维线性的紧凑格式来存储该Map, 即省去了大量指针的内存开销，这个参数控制对应在redis.conf配置文件中下面2项：

hash-max-zipmap-entries 64

hash-max-zipmap-value 512

当value这个Map内部不超过多少个成员时会采用线性紧凑格式存储，默认是64,即value内部有64个以下的成员就是使用线性紧凑存储，超过该值自动转成真正的HashMap。

hash-max-zipmap-value 含义是当 value这个Map内部的每个成员值长度不超过多少字节就会采用线性紧凑存储来节省空间。

以上2个条件任意一个条件超过设置值都会转换成真正的HashMap，也就不会再节省内存了，那么这个值是不是设置的越大越好呢，答案当然是否定的，HashMap的优势就是查找和操作的时间复杂度都是O(1)的，而放弃Hash采用一维存储则是O(n)的时间复杂度，如果成员数量很少，则影响不大，否则会严重影响性能，所以要权衡好这个值的设置，总体上还是最根本的时间成本和空间成本上的权衡。

list-max-ziplist-value 64

list-max-ziplist-entries 512

list数据类型节点值大小小于多少字节会采用紧凑存储格式、list数据类型多少节点以下会采用去指针的紧凑存储格式。

## 7.2内存预分配

Redis内部实现没有对内存分配方面做过多的优化（对比Memcache），在一定程度上会存在内存碎片，不过大多数情况下这个不会成为Redis的性能瓶颈，不过如果在Redis内部存储的大部分数据是数值型的话，Redis内部采用了一个shared integer的方式来省去分配内存的开销，即在系统启动时先分配一个从1~n 那么多个数值对象放在一个池子中，如果存储的数据恰好是这个数值范围内的数据，则直接从池子里取出该对象，并且通过引用计数的方式来共享，这样在系统存储了大量数值下，也能一定程度上节省内存并且提高性能，这个参数值n的设置需要修改源代码中的一行宏定义REDIS\_SHARED\_INTEGERS，该值默认是10000，可以根据自己的需要进行修改，修改后重新编译就可以了。

## **7.3持久化机制**

**定时快照方式(snapshot)：**

该持久化方式实际是在Redis内部一个定时器事件，每隔固定时间去检查当前数据发生的改变次数与时间是否满足配置的持久化触发的条件，如果满足则通过操作系统fork调用来创建出一个子进程，这个子进程默认会与父进程共享相同的地址空间，这时就可以通过子进程来遍历整个内存来进行存储操作，而主进程则仍然可以提供服务，当有写入时由操作系统按照内存页(page)为单位来进行copy-on-write保证父子进程之间不会互相影响。

该持久化的主要缺点是定时快照只是代表一段时间内的内存映像，所以系统重启会丢失上次快照与重启之间所有的数据。

**基于语句追加方式(aof)：**

aof方式实际类似mysql的基于语句的binlog方式，即每条会使Redis内存数据发生改变的命令都会追加到一个log文件中，也就是说这个log文件就是**Redis的持久化数据。**

aof的方式的主要缺点是追加log文件可能导致体积过大，当系统重启恢复数据时如果是aof的方式则加载数据会非常慢，几十G的数据可能需要几小时才能加载完，当然这个耗时并不是因为磁盘文件读取速度慢，而是由于读取的所有命令都要在内存中执行一遍。另外由于每条命令都要写log,所以使用aof的方式，Redis的读写性能也会有所下降。

可以考虑将数据保存到不同的Redis实例中，每个实例的内存大小在2G左右，避免将鸡蛋放到一个篮子里，既可以减少缓存失效给系统带来的影响，又可以加快数据恢复的速度，不过同时也给系统设计带来了一定的复杂性。

**Redis持久化崩溃问题：**

有Redis线上运维经验的人会发现Redis在物理内存使用比较多，但还没有超过实际物理内存总容量时就会发生不稳定甚至崩溃的问题，有人认为是基于快照方式持久化的fork系统调用造成内存占用加倍而导致的，这种观点是不准确的，因为fork 调用的copy-on-write机制是基于操作系统页这个单位的，也就是只有有写入的脏页会被复制，但是一般你的系统不会在短时间内所有的页都发生了写入而导致复制，那么是什么原因导致Redis崩溃的呢？

答案是Redis的持久化使用了Buffer IO造成的，所谓Buffer IO是指Redis对持久化文件的写入和读取操作都会使用物理内存的Page Cache,而大多数数据库系统会使用Direct IO来绕过这层Page Cache并自行维护一个数据的Cache，而当Redis的持久化文件过大(尤其是快照文件)，并对其进行读写时，磁盘文件中的数据都会被加载到物理内存中作为操作系统对该文件的一层Cache,而这层Cache的数据与Redis内存中管理的数据实际是重复存储的，虽然内核在物理内存紧张时会做Page Cache的剔除工作，但内核很可能认为某块Page Cache更重要，而让你的进程开始Swap ,这时你的系统就会开始出现不稳定或者崩溃了。我们的经验是当你的Redis物理内存使用超过内存总容量的3/5时就会开始比较危险了。

**总结：**

1、根据业务需要选择合适的数据类型，并为不同的应用场景设置相应的紧凑存储参数。

2、当业务场景不需要数据持久化时，关闭所有的持久化方式可以获得最佳的性能以及最大的内存使用量。

3、如果需要使用持久化，根据是否可以容忍重启丢失部分数据在快照方式与语句追加方式之间选择其一，不要使用虚拟内存以及diskstore方式。

4、不要让你的Redis所在机器物理内存使用超过实际内存总量的3/5。

5、redis.conf中的maxmemory选项，该选项是告诉Redis当使用了多少物理内存后就开始拒绝后续的写入请求，该参数能很好的保护好你的Redis不会因为使用了过多的物理内存而导致swap,最终严重影响性能甚至崩溃。

# 第8章 redis高可用集群

## 8.1 redis集群介绍

<http://redisdoc.com/topic/cluster-tutorial.html> 参考

集群技术是构建高性能网站架构的重要手段，试想在网站承受高并发访问压力的同时，还需要从海量数据中查询出满足条件的数据，并快速响应，我们必然想到的是将数据进行切片，把数据根据某种规则放入多个不同的服务器节点，来降低单节点服务器的压力

Redis 集群是一个可以**在多个 Redis 节点之间进行数据共享**的设施（installation）。

Redis 集群不支持那些需要同时处理多个键的 Redis 命令， 因为执行这些命令需要在多个 Redis 节点之间移动数据， 并且在高负载的情况下， 这些命令将降低 Redis 集群的性能， 并导致不可预测的行为。

Redis 集群**通过分区（partition）来提供一定程度的可用性**（availability）： 即使集群中有一部分节点失效或者无法进行通讯， 集群也可以继续处理命令请求。

Redis 集群提供了以下两个好处：

将数据自动切分（split）到多个节点的能力。

当集群中的一部分节点失效或者无法进行通讯时， 仍然可以继续处理命令请求的能力。

## 8.2 redis集群方案

redis集群方案主要有

* 1. 客户端分片，即在程序源码中做好配置
  2. 代理 如：Twemproxy
  3. redis cluster redis3.0以上支持 redis主从+ Sentinel也可以实现 3.0之前的版本推荐使用

4、codis 国内豌豆荚开源，推荐使用

## 8.3 部署redis-cluster集群

### 8.3.1 节点规划

**要让集群正常运作至少需要三个主节点**，使用六个节点： 其中三个为主节点， 而其余三个则是各个主节点的从节点。

现在服务器上已经有3个节点，6379、6380、6381,再创建3个节点，6382、6383、6384。

**创建目录：**

[root@node1 ~]# cd /data/

[root@node1 data]# mkdir `seq 6382 6384`

[root@node1 data]# ls

6380 6381 6382 6383 6384 redis

[root@node1 data]# mkdir 6383/data

[root@node1 data]# mkdir 6382/data

[root@node1 data]# mkdir 6384/data

**创建配置文件：**

[root@node1 ~]# sed 's/6379/6382/g' /usr/local/redis/conf/redis.conf >>/data/6382/redis.conf

[root@node1 ~]# sed 's/6379/6383/g' /usr/local/redis/conf/redis.conf >>/data/6383/redis.conf

[root@node1 ~]# sed 's/6379/6384/g' /usr/local/redis/conf/redis.conf >>/data/6384/redis.conf

### 8.3.2 部署redis-cluster

**修改配置文件**

[root@node1 ~]# vim /usr/local/redis/conf/redis.conf 这个配置最好也放/data/6379下面去

daemonize yes

pidfile /var/run/redis\_6379.pid

port 6379

cluster-enabled yes

cluster-config-file nodes.conf

cluster-node-timeout 5000

logfile /var/log/redis\_6379.log

dir /data/6379/data

appendonly yes

[root@node1 ~]# sed 's/6379/6380/g' /usr/local/redis/conf/redis.conf >>/data/6380/redis.conf

[root@node1 ~]# sed 's/6379/6381/g' /usr/local/redis/conf/redis.conf >>/data/6381/redis.conf

[root@node1 ~]# sed 's/6379/6382/g' /usr/local/redis/conf/redis.conf >>/data/6382/redis.conf

[root@node1 ~]# sed 's/6379/6383/g' /usr/local/redis/conf/redis.conf >>/data/6383/redis.conf

[root@node1 ~]# sed 's/6379/6384/g' /usr/local/redis/conf/redis.conf >>/data/6384/redis.conf

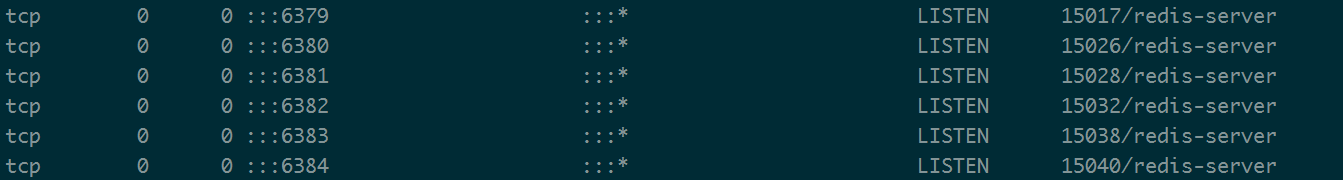
[root@node1 ~]# mv /usr/local/redis/conf/redis.conf /data/6379

### 8.3.3 启动redis节点

[root@node1 ~]# pkill redis

[root@node1 ~]# for i in `seq 6379 6384`;do cd /data/$i && /usr/local/redis/bin/redis-server redis.conf;done

[root@node1 6384]# netstat -antup | grep 63\*



[root@node1 ~]# redis-cli -h 192.168.1.24 -p 6379 -a admin123

### 8.3.4 创建redis-cluster集群

**安装ruby环境**

[root@node1 ~]# yum install ruby rubygems –y

**安装ruby的redis，客户端**

**redis-trib.rb 是官方出品的，是ruby开发的，其实他的原理也是调用reids的cluster指令**

[root@node1 ~]# gem install redis

[root@node1 ~]# cp redis-3.0.6/src/redis-trib.rb /usr/local/bin/redis-trib 从源码拷贝ruby脚本

[root@node1 ~]# redis-trib help 查看帮助

**创建集群：**

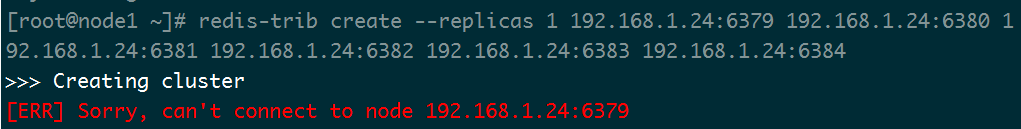
[root@node1 ~]# redis-trib create --replicas 1 192.168.1.24:6379 192.168.1.24:6380 192.168.1.24:6381 192.168.1.24:6382 192.168.1.24:6383 192.168.1.24:6384

命令的意义如下：

给定 redis-trib.rb 程序的命令是 create ， 这表示我们希望创建一个新的集群。

选项 --replicas 1 表示我们希望为集群中的每个主节点创建一个从节点。

之后跟着的其他参数则是实例的地址列表， 我们希望程序使用这些地址所指示的实例来创建新集群



报错：因为有密码验证，要取消密码验证

sed -i 's@requirepass admin123@#requirepass@g' redis.conf 取消密码

[root@node1 data]# sed -i 's@requirepass admin123@#requirepass@g' 6379/redis.conf

[root@node1 data]# sed -i 's@requirepass admin123@#requirepass@g' 6380/redis.conf

[root@node1 data]# sed -i 's@requirepass admin123@#requirepass@g' 6381/redis.conf

[root@node1 data]# sed -i 's@requirepass admin123@#requirepass@g' 6382/redis.conf

[root@node1 data]# sed -i 's@requirepass admin123@#requirepass@g' 6383/redis.conf

[root@node1 data]# sed -i 's@requirepass admin123@#requirepass@g' 6384/redis.conf

[root@node1 ~]# redis-trib create --replicas 1 192.168.1.24:6379 192.168.1.24:6380 192.168.1.24:6381 192.168.1.24:6382 192.168.1.24:6383 192.168.1.24:6384

>>> Creating cluster

>>> Performing hash slots allocation on 6 nodes...

Using 3 masters:

192.168.1.24:6379

192.168.1.24:6380

192.168.1.24:6381

Adding replica 192.168.1.24:6382 to 192.168.1.24:6379

Adding replica 192.168.1.24:6383 to 192.168.1.24:6380

Adding replica 192.168.1.24:6384 to 192.168.1.24:6381

M: fca0dd3f75506d379064a884c0b916bb1dfb2137 192.168.1.24:6379

slots:0-5460 (5461 slots) master

M: 5266396d4f817a75a34557ce5a26fa9a4db6b197 192.168.1.24:6380

slots:5461-10922 (5462 slots) master

M: 27e3720db919002e50f00f3b3e09eea32518fb00 192.168.1.24:6381

slots:10923-16383 (5461 slots) master

S: ffa95268b6bf63b5d85fc9e1b8066bb073dd8d0f 192.168.1.24:6382

replicates fca0dd3f75506d379064a884c0b916bb1dfb2137

S: 17852e7528009011b92335fc2bcaf9fe2c92ff75 192.168.1.24:6383

replicates 5266396d4f817a75a34557ce5a26fa9a4db6b197

S: bc6ded8f7b1af5ee7f3811397bba0565145c4455 192.168.1.24:6384

replicates 27e3720db919002e50f00f3b3e09eea32518fb00

Can I set the above configuration? (type 'yes' to accept): **yes 输入yes**

>>> Nodes configuration updated

>>> Assign a different config epoch to each node

>>> Sending CLUSTER MEET messages to join the cluster

Waiting for the cluster to join...

>>> Performing Cluster Check (using node 192.168.1.24:6379)

M: fca0dd3f75506d379064a884c0b916bb1dfb2137 192.168.1.24:6379

slots:0-5460 (5461 slots) master 槽位分配

M: 5266396d4f817a75a34557ce5a26fa9a4db6b197 192.168.1.24:6380

slots:5461-10922 (5462 slots) master

M: 27e3720db919002e50f00f3b3e09eea32518fb00 192.168.1.24:6381

slots:10923-16383 (5461 slots) master

M: ffa95268b6bf63b5d85fc9e1b8066bb073dd8d0f 192.168.1.24:6382

slots: (0 slots) master

replicates fca0dd3f75506d379064a884c0b916bb1dfb2137

M: 17852e7528009011b92335fc2bcaf9fe2c92ff75 192.168.1.24:6383

slots: (0 slots) master

replicates 5266396d4f817a75a34557ce5a26fa9a4db6b197

M: bc6ded8f7b1af5ee7f3811397bba0565145c4455 192.168.1.24:6384

slots: (0 slots) master

replicates 27e3720db919002e50f00f3b3e09eea32518fb00

[OK] All nodes agree about slots configuration.

>>> Check for open slots...

>>> Check slots coverage...

[OK] All 16384 slots covered.

链接redis

[root@node2 ~]# redis-cli -c -h 192.168.1.24 -p 6379 端口随便哪个

192.168.1.24:6379> set name admin

-> Redirected to slot [5798] located at 192.168.1.24:6380 自动转到6380

OK

192.168.1.24:6380> CLUSTER INFO 查看集群状态

cluster\_state:ok

cluster\_slots\_assigned:16384

cluster\_slots\_ok:16384

cluster\_slots\_pfail:0

cluster\_slots\_fail:0

cluster\_known\_nodes:6

cluster\_size:3

cluster\_current\_epoch:6

cluster\_my\_epoch:2

cluster\_stats\_messages\_sent:3738

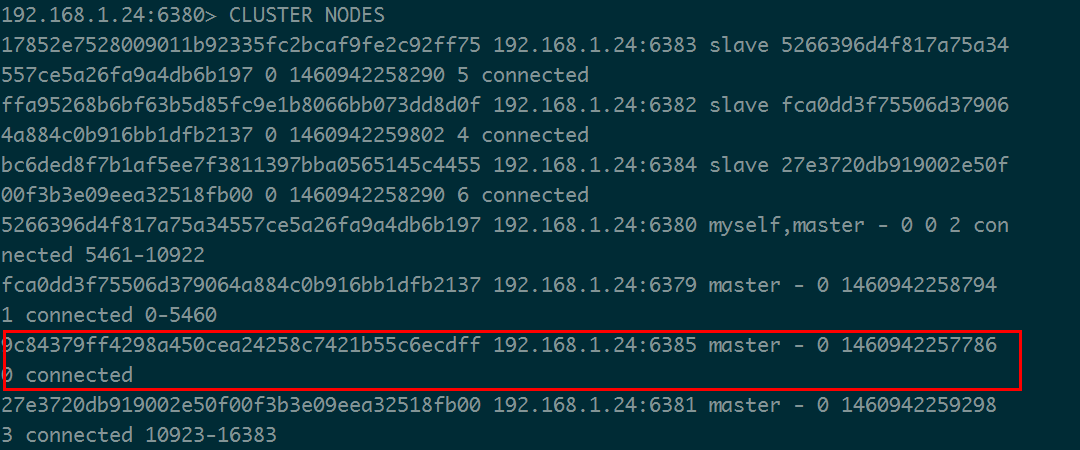
cluster\_stats\_messages\_received:3738

### 8.3.5 添加一个节点，完成一对主从

准备2个节点6385 6386，方法参照之前的创建就行

**添加节点**

[root@node1 ~]# redis-trib add-node 192.168.1.24:6385 192.168.1.24:6384 端口随便后面，添加的节点默认为master



一个槽位都没有，集群中写入不了数据，要分配槽位给它

**分配槽位给6385**

[root@node1 ~]# redis-trib reshard 192.168.1.24:6385

[OK] All 16384 slots covered.

How many slots do you want to move (from 1 to 16384)? 500 分配多少槽位

What is the receiving node ID? 9c84379ff4298a450cea24258c7421b55c6ecdff 分配到哪个节点

**查看**

192.168.1.24:6380> CLUSTER NODES

9c84379ff4298a450cea24258c7421b55c6ecdff 192.168.1.24:6385 master - 0 1460942774480 7 connected 0-165 5461-5627 10923-11088

* **再添加一个节点作为6385的从节点**

[root@node1 ~]# redis-trib add-node 192.168.1.24:6386 192.168.1.24:6384

**添加进来默认为master,没有槽位**

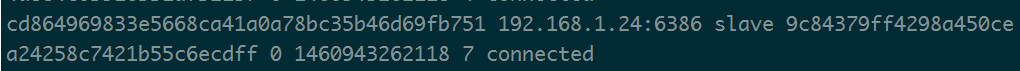


**连接6386进行操作**

[root@node1 ~]# redis-cli -c -h 192.168.1.24 -p 6386

192.168.1.24:6386> CLUSTER REPLICATE 9c84379ff4298a450cea24258c7421b55c6ecdff id为6385的id

OK



## 8.4 codis集群

<https://github.com/CodisLabs/codis>

codis介绍：<https://github.com/CodisLabs/codis/blob/master/doc/tutorial_zh.md>

### 8.4.1安装go环境：

[root@node1 ~]# yum install golang –y

[root@node1 ~]# mkdir /opt/gopath

[root@node1 ~]# vim /etc/profile

export GOPATH=/opt/gopath

PATH=/usr/local/redis/bin/:$GOPATH/bin:$PATH

[root@node1 ~]# source /etc/profile

### 8.4.2下载codis:

[root@node1 ~]# go get -u -d github.com/CodisLabs/codis 需要等待很长时间

### 8.4.3安装jdk:

[root@node1 ~]# rpm -ivh jdk-8u73-linux-x64.rpm

[root@node1 ~]# vim /etc/profile

export JAVA\_HOME=/usr/java/jdk1.8.0\_73

export JAVA\_BIN=/usr/java/jdk1.8.0\_73/bin

export PATH=${JAVA\_HOME}/bin:$PATH

export CLASSPATH=.:${JAVA\_HOME}/lib/dt.jar:${JAVA\_HOME}/lib/tools.jar

[root@node1 ~]# source !$

source /etc/profile

[root@node1 ~]# java -version

java version "1.8.0\_73"

Java(TM) SE Runtime Environment (build 1.8.0\_73-b02)

Java HotSpot(TM) 64-Bit Server VM (build 25.73-b02, mixed mode)

### 8.4.5安装配置zookeeper:

[root@node1 ~]# cd /opt/

[root@node1 opt]# mkdir zk1 zk2 zk3

[root@node1 opt]# echo "1" > zk1/myid

[root@node1 opt]# echo "2" > zk2/myid

[root@node1 opt]# echo "3" > zk3/myid

[root@node1 ~]# wget http://mirrors.hust.edu.cn/apache/zookeeper/zookeeper-3.4.8/zookeeper-3.4.8.tar.gz

[root@node1 ~]# tar -xf zookeeper-3.4.8.tar.gz

[root@node1 ~]# mv zookeeper-3.4.8 /usr/local/zookeeper

[root@node1 ~]# cd !$

cd /usr/local/zookeeper

[root@node1 zookeeper]# cd conf/

[root@node1 conf]# cp zoo\_sample.cfg /opt/zoo.cfg

[root@node1 conf]# cd /opt/

[root@node1 opt]# vim zoo.cfg

dataDir=/opt/zk1/2/3

clientPort=2181/2182/2183

server.1=192.168.1.24:2887:3887

server.2=192.168.1.24:2888:3888

server.3=192.168.1.24:2889:3889

[root@node1 opt]# cp zoo.cfg zk1/zk1.cfg

[root@node1 opt]# cp zoo.cfg zk2/zk2.cfg

[root@node1 opt]# cp zoo.cfg zk3/zk3.cfg

再把各配置dir改下dataDir=/opt/zk1/2/3

grep '^[a-z]' zk1/zk1.cfg查看下

### 8.4.6启动zookeeper：

[root@node1 ~]# /usr/local/zookeeper/bin/zkServer.sh start /opt/zk1/zk1.cfg

[root@node1 ~]# /usr/local/zookeeper/bin/zkServer.sh start /opt/zk2/zk2.cfg

[root@node1 ~]# /usr/local/zookeeper/bin/zkServer.sh start /opt/zk3/zk3.cfg

### 8.4.7查看状态：

[root@node1 ~]# /usr/local/zookeeper/bin/zkServer.sh status /opt/zk3/zk3.cfg

ZooKeeper JMX enabled by default

Using config: /opt/zk3/zk3.cfg

Mode: follower

[root@node1 ~]# /usr/local/zookeeper/bin/zkServer.sh status /opt/zk2/zk2.cfg

ZooKeeper JMX enabled by default

Using config: /opt/zk2/zk2.cfg

Mode: leader

[root@node1 ~]# /usr/local/zookeeper/bin/zkServer.sh status /opt/zk1/zk1.cfg

ZooKeeper JMX enabled by default

Using config: /opt/zk1/zk1.cfg

Mode: follower

### 8.4.8连接zookeeper：

[root@node1 ~]# /usr/local/zookeeper/bin/zkCli.sh -server 192.168.1.24:2181

### 8.4.9配置codis

[root@node1 ~]# cd /opt/gopath/src/github.com/CodisLabs/codis/

[root@node1 codis]# make

[root@node1 codis]# make gotest

[root@node1 ~]# vim config.ini

zk=192.168.1.24:2181,192.168.1.24:2182,192.168.1.24:2183

dashboard\_addr=192.168.1.24:18087

[root@node1 codis]# bin/codis-config dashboard & #启动 dashboard

[root@node1 codis]# bin/codis-config slot init 在zookeeper上创建slot相关信息

### 8.4.10添加 Redis Server Group:

每一个 Server Group 作为一个 Redis 服务器组存在, 只允许有一个 master, 可以有多个 slave, *group id 仅支持大于等于1的整数*

[root@node1 codis]# bin/codis-config server -h

usage:

codis-config server list

codis-config server add <group\_id> <redis\_addr> <role>

codis-config server remove <group\_id> <redis\_addr>

codis-config server promote <group\_id> <redis\_addr>

codis-config server add-group <group\_id>

codis-config server remove-group <group\_id>

添加一个group，group的id为1， 并添加一个redis master到该group

[root@node1 codis]# bin/codis-config server add 1 192.168.1.24:6379 master

添加一个redis slave到该group

[root@node1 codis]# bin/codis-config server add 1 192.168.1.24:6380 slave

类似的，再添加group，group的id为2

[root@node1 codis]# bin/codis-config server add 2 localhost:6381 master

[root@node1 codis]# bin/codis-config server add 2 localhost:6382 slave

### 8.4.11设置 server group 服务的 slot 范围:

Codis 采用 Pre-sharding 的技术来实现数据的分片, 默认分成 1024 个 slots (0-1023), 对于每个key来说, 通过以下公式确定所属的 Slot Id : SlotId = crc32(key) % 1024 每一个 slot 都会有一个且必须有一个特定的 server group id 来表示这个 slot 的数据由哪个 server group 来提供.

[root@node1 codis]# bin/codis-config slot -h

usage:

codis-config slot init [-f]

codis-config slot info <slot\_id>

codis-config slot set <slot\_id> <group\_id> <status>

codis-config slot range-set <slot\_from> <slot\_to> <group\_id> <status>

codis-config slot migrate <slot\_from> <slot\_to> <group\_id> [--delay=<delay\_time\_in\_ms>]

codis-config slot rebalance [--delay=<delay\_time\_in\_ms>]

设置编号为[0, 511]的 slot 由 server group 1 提供服务, 编号 [512, 1023] 的 slot 由 server group 2 提供服务

[root@node1 codis]# bin/codis-config slot range-set 0 511 1 online

[root@node1 codis]# bin/codis-config slot range-set 512 1023 2 online

### 8.4.12启动 codis-proxy:

[root@node1 codis]# bin/codis-proxy -c config.ini -L ./log/proxy.log --cpu=4 --ad=0.0.0.0:19000 --http-addr=0.0.0.0:11000 &

刚启动的 codis-proxy 默认是处于 offline状态的, 然后设置 proxy 为 online 状态, 只有处于 online 状态的 proxy 才会对外提供服务

[root@node1 codis]# bin/codis-config -c config.ini proxy online proxy\_1 &

### 8.4.13 使用web浏览器访问

<http://192.168.1.24:18087/admin/>

再添加一组redis主从

在web上操作就行了

参考：

<http://www.fwqtg.net/codis.html>

<http://blog.csdn.net/dc_726/article/details/47052607>