# 快速安装配置redis持久化服务

快速安装配置Redis持久化服务

## 1.1 redis简介

### 1.1.3 redis简介

RemoteDictionary Server是一个基于Key-value键值对的持久化数据库数据库存储系统，redis和memcached缓存服务器很像，但是redis支持的数据存储类型更丰富，包括String，list，set和zset等

这些数据类型都支持push/pop，add/remove及取交集，并集和差集及更丰富的操作，而这些操作都是原子性的，在此基础上，redis支持各种不同方式的排序，雨memcached缓存服务一样，为了保证销量，数据都是缓存再内存中提供服务的，和memcached不同的是redis持久化缓存服务器还会周期行的把更新的数据写入到磁盘以及把修改的操作记录追加到文件里记录下来，比memcached更有优势的是redis还支持master-slave主从同步，这点累死关系型数据库mysql

redis的出现在一定程度上弥补了memcached这类key-value内存缓存服务的不足，在部分场合可以对关系数据库起到很好的补充作用，redis提供了python，ruby，erlang，php客户端，使用很方便，redis官方文档如下：

<http://www.redis.io/documentation>

### 1. 1.2 redis的优点

性能很高：redis支持超过100K每秒的读取频率

丰富数据类型，redis支持二进制的String，Lisst，Hashes，Sets及Ordered Sets等数据类型操作

原子：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

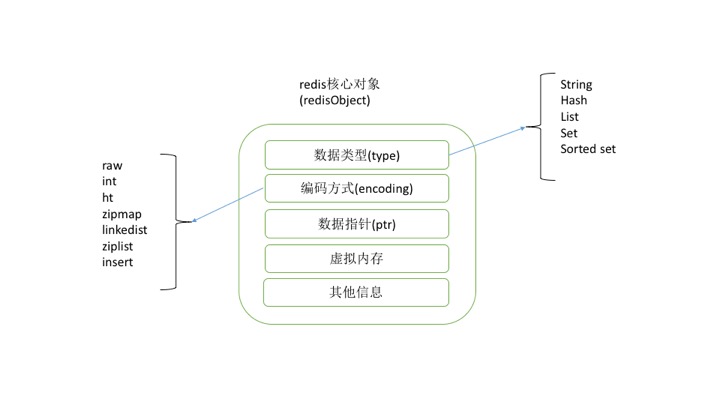
丰富的特性：redis还支持publiss/subscribe，通知，key过期等等特性

redis支持异机主从复制

### 1. 1.3 redis的数据类型

redis最为常用的数据类型主要有以下五种：

* + String
  + Hash
  + List
  + Set
  + Sorted set



### 1. 1.4 redis的应用场景

传统的Mysql+Memcached的网站架构遇到的问题

Mysql数据实际上是适合进行海量数据存储的，加上通过Memcached将热点数据存放到内存cache里，达到加速数据访问的目的，绝大部份公司都曾经使用过这样的架构，但是随着业务数据量的不断增加，和访问量的持续增长，很多问题就会暴露出来。

* + 需要不断的对Mysql进行拆库拆表，Memcached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
  + memcached与Mysql数据库数据导致一致性问题是个老大难
  + memcached数据命中率低或者宕机，会导致大量访问直接穿透到数据库，导致mysql无法支撑访问。

跨机房cache同步一致性问题

redis的最佳场景：

* + Rdis最佳使用场景的全部是数据in-memory
  + redis更多场景是作为Memcached的替代品来使用
  + 当需要除key/value之外的更多数据类型支持时，使用Redis更合适
  + 当存储的数据不能被提出时，使用Redis更合适

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

企业场景：

1、使用Redis bitmap进行活跃用户统计

http://blog.nosqlfan.com/html/3501.html

这里对Redis数据库做了如下小结：

* + 提高了DB的可拓展性，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
  + 小的数据库存的查询压力小，查询更快，性能更好

使用过程中的一些经验和教训，如下时小结

* + 要进行Master-slave的配置，出现服务故障时可以支持切换
  + 在master侧禁用数据持久化，只需在slave上配置数据持久化
  + 物理内存+虚拟内存不足，这个dump一致死着，时间久了机器挂掉，这个情况下就是灾难
  + 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
  + 当达到最大内存时，回清空带有过期时间的key，即使key未过期时间
  + redis与DB同步写的问题，先写DB，后写redis，因为内存基本上没问题

## 1.2 redis部署环境搭建

Master:10.39.1.7 Centos 6.4

Master:10.39.1.8 Centos 6.4

## 1.3 开始安装redis

在redis的官方网站(http://www.redis.io)下载最新的稳定版本redis

操作命令如下：

1、获取redis安装包，并解压

wget http://download.redis.io/releases/redis-3.0.7.tar.gz

tar xzf redis-3.0.7.tar.gz

2、进入目录，并查看说明文件README

cd redis-3.0.7

less README

3、配置redis安装包

make MANIFESTO=jemalloc && \

make PREFIX=/application/redis-3.0.7 install

4、创建软连接

ln -s /application/redis-3.0.7/ /application/redis

5、配置环境语言

LANG=en\_US.UTF-8

6、查看redis命令的目录

tree /application/redis/bin/

执行结果如下：

/application/redis/bin/

├── redis-benchmark

├── redis-check-aof

├── redis-check-dump

├── redis-cli

├── redis-sentinel -> redis-server

└── redis-server

命令执行完成后，会在/applicaion/redis/bin/目录下生成5个可执行文件，分别是：

redis-benchmark redis-check-aof redis-check-dump redis-cli redis-sentinel redis-server

它们的作用如下：

|  |  |
| --- | --- |
| 命令 | 说明 |
| redis-server | Redis服务器的daemon启动程序 |
| redis-cli | Redis命令行操作工具，当然也可以用telnet跟其纯文本协议来操作 |
| redis-benchmark | Redis性能测试工具，测试Redis在你的系统及你的配置下的读写 |
| redis-check-aof | 更新日志检查 |
| redis-check-dump | 用语本地数据库的检查 |

## 1.3 配置并启动redis服务

a. 配置环境变量，命令如下

echo ' PATH=/application/redis/bin/:$PATH' >> /etc/profile

source /etc/profile

查看执行结果如下：

tail -1 /etc/profile

提示：PATH=/application/redis/bin/:$PATH

查看是否被导入到全局路径下

which redis-server

执行结果如下

/application/redis/bin/redis-server

b. 查看命令帮助

[root@mysql01 ~]# redis-server --help

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>

Examples:

./redis-server (run the server with default conf)

./redis-server /etc/redis/6379.conf

./redis-server --port 7777

./redis-server --port 7777 --slaveof 127.0.0.1 8888

./redis-server /etc/myredis.conf --loglevel verbose

Sentinel mode:

./redis-server /etc/sentinel.conf --sentinel

c. 启动redis服务

操作命令：

cp ~/redis-3.0.7/redis.conf /application/redis/conf #安装包内含有redis的默认配置文件

sysctl vm.overcommit\_memory=1

redis-server /application/redis/conf &

提示：启动后会出现夯住的命令行，但是其实只要回车就好，因为此处回车后台执行

提示:查看端口是否存在确定redis是否启动成功

[root@mysql01 ~]# netstat -lntup|grep -w 6379

tcp 0 0 0.0.0.0:6379 0.0.0.0:\* LISTEN 5896/redis-server \*

tcp 0 0 :::6379 :::\* LISTEN 5896/redis-server \*

d. 关闭redis

redis-cli shutdown

方式二

killall redis-server

## 1.5 通过客户端测试redis服务

redis-cli客户端帮助

[root@mysql01 ~]# redis-cli --help

redis-cli 3.0.7

Usage: redis-cli [OPTIONS] [cmd [arg [arg ...]]]

-h <hostname> Server hostname (default: 127.0.0.1).

-p <port> Server port (default: 6379).

-s <socket> Server socket (overrides hostname and port).

-a <password> Password to use when connecting to the server.

执行redis-cli连接到数据库

[root@mysql01 ~]# redis-cli

127.0.0.1:6379>

运用help查看帮助

127.0.0.1:6379> help get

GET key

summary: Get the value of a key

since: 1.0.0

group: string

127.0.0.1:6379>

help查看set

127.0.0.1:6379> help set

SET key value [EX seconds] [PX milliseconds] [NX|XX]

summary: Set the string value of a key

since: 1.0.0

group: string

127.0.0.1:6379>

如果配置了密码和绑定IP，更换端口，则要用一下连接的方式

redis-cli -h 10.39.1.51 -p 22 -a '123456'

基本语法：

redis-cli set key value

redis-cli get key

连接方法1:(此方法可以远程使用)

[root@mysql01 ~]# redis-cli set key1 value1

OK

[root@mysql01 ~]# redis-cli get key1

"value1"

[root@mysql01 ~]# redis-cli del key1

(integer) 1

[root@mysql01 ~]# redis-cli get key1

(nil)

[root@mysql01 ~]#

连接方法二：

[root@mysql01 ~]# redis-cli

127.0.0.1:6379> set key1 value1

OK

127.0.0.1:6379> get key1

"value1"

127.0.0.1:6379> del key1

(integer) 1

127.0.0.1:6379> get key1

(nil)

127.0.0.1:6379>

连接方式3:

[root@mysql01 ~]# telnet 127.0.0.1 6379

Trying 127.0.0.1...

Connected to 127.0.0.1.

Escape character is '^]'.

set key1 values1

+OK

get key1

$7

values1

del key1

:1

get key1

$-1

quit

+OK

Connection closed by foreign host.

[root@mysql01 ~]#

redis支持多种数据类型

值可以是任何种类的字符串( 包括二进制数据)，例如可以在一个键下保存一副jpeg图片，值长度不能超过1G

[root@mysql01 ~]# redis-cli set mykey "hello world"

OK

[root@mysql01 ~]# redis-cli get mykey

"hello world"

[root@mysql01 ~]# redis-cli set mykey1 1

OK

[root@mysql01 ~]# redis-cli incr mykey1

(integer) 2

[root@mysql01 ~]# redis-cli incr mykey1

(integer) 3

[root@mysql01 ~]#

### 1.5.1 字符串类型

这是最贱的Redis类型，如果只使用此类型，Redis就是一个可以持久化memcached服务器(注：memcache的数据仅保存在内存中，服务器重启后，数据将丢失)

[root@mysql01 ~]# redis-cli set mykey "hello world"

OK

[root@mysql01 ~]# redis-cli get mykey

"hello world"

"my binary safe value"正如所见，通常set和get来设置获取字符串值

值可以是任何种类的字符串(包括二进制数据)，例如可以在一个键先保存一副图片，但是值的长度不能超过1GB

虽然字符串是Redis的基本类型，但仍然可以通过他完成一些有趣的操作，例如原子递增：

[root@mysql01 ~]# redis-cli set mykey1 1

OK

[root@mysql01 ~]# redis-cli incr mykey1

(integer) 2

[root@mysql01 ~]# redis-cli incr mykey1

(integer) 3

INCR命令将字符串值解析成整数，将其加1，最后将结果保存为新的字符串值，类似的命令有INCRBY，DECR 和DECRBY实际上它们在内部就是同一个命令，只是看着不同而已

INCR是原子操作意味着即使多个客户端对同一个key进行INCR命令，也绝不会导致竞争的情况，例如如下情况永远不可能发生同时独处"10"。他们俩都对其加到11，然后将新值设置为111，最重的值已定为12，read-increment-set操作完成时，其他客户端不糊自爱同一时间执行任何命令

对字符串，另一个的凌人感兴趣的操作时GETSET命令，行如其名，他的key设置新值并返回原值，当系统每当有新的用户访问的时候就用INCR  
命令操作一个Redis key，如果每个小时对这个信息收集一次，就可以GETSET这个key并给其赋值0并读取其原值

### 1.5.2 列表数据类型List

List这个词有点不准确，但是实际上是数组(同样的数据类型在Ruby中叫做数组)

一般意义上讲，列表就是有序元素的序列10,20,1,2,3这就是一个列表，但是数组是心啊的List和Linked实现的List，在属性方面大不相同。

Redis基于Linked实现，意味着即使在一个list中有数百万个元素，在头部或者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10+个元素的头部添加新的元素，和在千万元素list头部添加新元素的速度相同

而劣势在于，在数组实现的list中利用索引访问元素的速度极快，而同样在操作linked list实现的list上没有那么快

Redis Lists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：能分畅快的在很大的列表上添加元素，另一个重要的原因是Redis能在常数时间内取得常数长度

Redis list入门

LPUSH命令像list的左边(头部)添加一个新的元素，而RPUSH命令可以向右边(尾部)添加一个新元素，最后LRANGE命令可以从list中取得一定范围的元素

[root@mysql01 ~]# redis-cli rpush messages "Hello how are you"

(integer) 1

[root@mysql01 ~]# redis-cli rpush messages "Fine thanks ,I'm having a fun with Reids"

(integer) 2

[root@mysql01 ~]# redis-cli rpush messages "I should look into this NOSQL thing ASAP"

(integer) 3

[root@mysql01 ~]# redis-cli lrange messages 0 2

1) "Hello how are you"

2) "Fine thanks ,I'm having a fun with Reids"

3) "I should look into this NOSQL thing ASAP"

[root@mysql01 ~]# redis-cli lrange messages -2 -1

1) "Fine thanks ,I'm having a fun with Reids"

2) "I should look into this NOSQL thing ASAP"

[root@mysql01 ~]# redis-cli lrange messages 0 -2

1) "Hello how are you"

2) "Fine thanks ,I'm having a fun with Reids"

注意lrange带有两个索引，一定范围的第一个和最后一个元素，这两个索引都可以为负来告知Redis从尾部开始计数，因此-1表示最后一个元素，-2表示list中的倒数第二个元素，依次类推，重要的是下标位置要从左到右

list可以被用来实现聊天系统，还可以作为不同进程之间传递消息的队列，关键是可以每次都以原先添加的顺序访问数据，不需要任何sql order by操作，会很快，而且很容易扩展到到百万级别元素的规模

例如在评级系统中，比如社会化新闻网站reddit.com，可以每个新提交的链接添加到一个list，用lrange可简单的对结果分页

在博客引擎实现中，可以为每篇日志设置一个list，在该list总推入进博客评论等

向redis list压入ID而不是实际的数据

在上面的例子中，直接将“对象”（此例中是简单消息）直接压入Reids list，但是通常不应该这么做，由于对象可能被很多次饮用，例如在一个list中维护其他时间顺序，在一个集合中保存它的类别，只要有必要，他还会出现在其他list中，等等

回到reddit.com的例子，将用户提交的链接(新闻)添加到list中，有更可靠的方法如下所示

[root@mysql01 ~]# redis-cli incr next.news.id

(integer) 1

[root@mysql01 ~]# redis-cli set news:1:title "Redis is simple"

OK

[root@mysql01 ~]# redis-cli set news:1:url "http://code.google.com"

OK

[root@mysql01 ~]# redis-cli lpush submitted.news 1

(integer) 1

[root@mysql01 ~]#

我们自增一个key，很容易得到独一无二的自增ID，然后通过ID创建对象- 为对象的每个字断设置一个key，最后将新的对象ID压入submitted.news list

当然还可以删除元素，旋转元素，根据索引获取和设置元素，还可以采用llen得到list的长度

### 1.5.3 Redis集合

Redis集合是未排序的集合，其元素是二进制安全的字符串，SADD命令可以向集合添加一个新的元素，和sets相关的操作也有很多，比如检测某个元素是否存在，以及实现交集，并集，差集等操作

[root@mysql01 ~]# redis-cli sadd myset a

(integer) 1

[root@mysql01 ~]# redis-cli sadd myset b

(integer) 1

[root@mysql01 ~]# redis-cli sadd myset c

(integer) 1

[root@mysql01 ~]# redis-cli smembers myset

1) "c"

2) "a"

3) "b"

[root@mysql01 ~]#

向集合中添加了三个元素，并让每个redis返回所有元素，它们是无序的

检测某个元素是否存在

[root@mysql01 ~]# redis-cli sismember myset a

(integer) 1

[root@mysql01 ~]# redis-cli sismember myset d

(integer) 0

[root@mysql01 ~]#

其中：a是这个集合的成员，而'd'不是，集合特别适合表现对象之间的关系，例如用Redis集合很容易实现标签功能

对每个想加标签的对象，用一个标签ID集合与之关联，并且对每个已有的标签，一组对象ID预知关联

例如，新闻ID1000被加了三个标签tar 1,2,5和77，就可以设置下面两个集和

[root@mysql01 ~]# redis-cli sadd news:1000:tags 1

(integer) 1

[root@mysql01 ~]# redis-cli sadd news:1000:tags 2

(integer) 1

[root@mysql01 ~]# redis-cli sadd news:1000:tags 5

(integer) 1

[root@mysql01 ~]# redis-cli sadd tag:1:objects 1000

(integer) 1

[root@mysql01 ~]# redis-cli sadd tag:2:objects 1000

(integer) 1

[root@mysql01 ~]# redis-cli sadd tag:5:objects 1000

要获取一个对象的所有标签

[root@mysql01 ~]# redis-cli smembers news:1000:tags

1) "1"

2) "2"

3) "5"

[root@mysql01 ~]# redis-cli smembers tag:5:objects

1) "1000"

而有些看上去并不简单的操作仍然可以使用相应的Redis轻松实现，例如想获得一份同时拥有变迁1,2,5和27的对象列表，这里可以使用sinter命令实现，他能在不同集合之间取出交集

[root@mysql01 ~]# redis-cli sinter tag:1:objects tag:2:objects tag:5:objects

1) "1000"

如何为字符串获取唯一标识

在标签的例子里，我们用到了标签ID，却没有提到ID从何而来，基本上每个加入系统的标签分配一个唯一标识，多个客户端之间同时添加同样的标签时不要出现竞争情况，此外如果标签已经存在，希望返回他的ID，否则创建一个新的唯一标识并将其与此标签关联

redis增加了Hash类型，有了他字符串和唯一ID关联的事很容易

### 1.5.4 有序集合

集合是实用频率很高的数据类型，但是它们不讲顺序，因此引入了有序集合，同集合类似，也是二进制安全的字符串集合，但是这次带有关联的score，以及类似lrange的操作可以返回有序元素在有序集合中是zranges命令

127.0.0.1:6379> zadd hackers 1940 "Alan kay"

(integer) 1

127.0.0.1:6379> zadd hackers 1953 "Richard"

(integer) 1

127.0.0.1:6379> zadd hackers 1965 "Yukihiro"

(integer) 1

127.0.0.1:6379> zadd hackers 1916 "Claude"

(integer) 1

对于有序集合来说，按生日排序返回这些很容易，因为已经有序了，有序集合通过一个dual-ported数据结构实现的，他包含一个精简的有序列表和一个hash table因此添加一个元素的时间复杂度为O(log(N)),但是需要访问有序的元素时，redis不必做任何事情，因为已经有序了

127.0.0.1:6379> ZRANGE hackers 0 -1

1) "Claude"

2) "Alan kay"

3) "Richard"

4) "Yukihiro"

如果反对这些元素排序，可以使用zrevrange代替zrange

127.0.0.1:6379> ZREVRANGE hackers 0 -1

1) "Yukihiro"

2) "Richard"

3) "Alan kay"

4) "Claude"

提示：zsets只是又一个默认的顺序，但是仍可以使用sort命令对有序集合做不同的排序,要想得到多种排序，一种可选方案是同时将每个元素加入多个有序集合

区间操作

有序集合还能在区间上操作，例如获取所有1950年之前出生的人，在此用ZRANGEBYSCORE命令实现：

127.0.0.1:6379> ZRANGEBYSCORE hackers -inf 1950

1) "Claude"

2) "Alan kay"

我们请求Reis返回score介于负无穷到1950之间的元素(两个值也包含了)

也可以删除区间内的元素，例如从有序集合中删除生日介于1940到1960年之间的黑客

127.0.0.1:6379> ZREMRANGEBYSCORE hackers 1940 1940

ZREMRANGEBYSCORE这个名字虽然不太好，但是非常有用，会返回已经删除的元素数量

最后Redis的例子，现在我们有一个基于有序集合的像样方案来生成首页，用一个有序集合来包涵最近几天的新闻(用ZREMRANGEBYSCORE不时的删除旧新闻)用一个后台任务从有序集合中获取所有元素，根据用户投票和新闻时间计算score，然后用新闻IDs和score，然后用新闻IDs和scores关联成reddit.home。page有序集合，要显示首页，我们只需闪电般的调用ZRANGE

不时的葱reddis.home.page有序集合中删除过旧的新闻也是为了让系统总是工作在有限的新闻集合之上

更新有序集合的scores

注意，有序集合scores可以在任何时候更新，制药用ZADD对有序集合内的元素操作就会更新他的score(和位置)，时间复杂度是O(log(N)),因此即使大量更新，有序集合也是合适的

## 1.6 为php安装redis客户端扩展

获取源码安装包

wget https://github.com/nicolasff/phpredis/archive/master.zip

安装

unzip master.zip && \

cd phpredis-master

/application/php/bin/phpize && \

./configure --with-php-config=/application/php/bin/php-config && \

make && make install

提示必须在插件目录下，否则会报Cannot find config.m4

修改php.ini设置，重启php

echo "extension = redis.so" >> /application/php/lib/php.ini

## 1.7 解读redis默认配置文件

[root@manager redis-3.0.7]# grep -vE "#|^$" /application/redis/conf redis.conf

daemonize no #是否后台进程运行，默认为no

pidfile /var/run/redis.pid #如果后台执行，需要指定pid路径

port 6379 #端口号

tcp-backlog 511 #tcp的backlog

timeout 0 #

tcp-keepalive 0 #tcp链接的一个保持

loglevel notice #日志级别，尽量不要notice，可以用warning

logfile ""

databases 16 #设置数据库的数量

save 900 1 #900秒内有1个变化磁盘保存

save 300 10 #300秒内有10个变化磁盘保存

save 60 10000 #60秒内有10000个变化保存磁盘

stop-writes-on-bgsave-error yes #如果出现错误是否要停止保存

# slaveof <masterip> <masterport> #如果主从同步从库指定主库的IP及端口

rdbcompression yes

rdbchecksum yes

dbfilename dump.rdb

dir ./

slave-serve-stale-data yes

slave-read-only yes

repl-diskless-sync no

repl-diskless-sync-delay 5

repl-disable-tcp-nodelay no

slave-priority 100

appendonly no

appendfilename "appendonly.aof"

appendfsync everysec

no-appendfsync-on-rewrite no

auto-aof-rewrite-percentage 100

auto-aof-rewrite-min-size 64mb

aof-load-truncated yes

lua-time-limit 5000

slowlog-log-slower-than 10000

slowlog-max-len 128

latency-monitor-threshold 0

notify-keyspace-events ""

hash-max-ziplist-entries 512

hash-max-ziplist-value 64

list-max-ziplist-entries 512

list-max-ziplist-value 64

set-max-intset-entries 512

zset-max-ziplist-entries 128

zset-max-ziplist-value 64

hll-sparse-max-bytes 3000

activerehashing yes

client-output-buffer-limit normal 0 0 0

client-output-buffer-limit slave 256mb 64mb 60

client-output-buffer-limit pubsub 32mb 8mb 60

hz 10

aof-rewrite-incremental-fsync yes

## 1.8 redis设置主从同步

从库配置reids主库的IP及Port，然后主库开启，主库增加数据，查看从库

主库查看redis功能统计

[root@mysql01 ~]# redis-cli -h localhost -p 6379 info

# Server

redis\_version:3.0.7

redis\_git\_sha1:00000000

redis\_git\_dirty:0

redis\_build\_id:49f470b525dbe837

redis\_mode:standalone

os:Linux 2.6.32-573.el6.x86\_64 x86\_64

arch\_bits:64

multiplexing\_api:epoll

gcc\_version:4.4.7

process\_id:1331

run\_id:58c1496d6e8592aa31a10cc3b032b0a091933ccd

tcp\_port:6379

uptime\_in\_seconds:483

uptime\_in\_days:0

hz:10

lru\_clock:11870445

config\_file:

# Clients

connected\_clients:1

client\_longest\_output\_list:0

client\_biggest\_input\_buf:0

blocked\_clients:0

# Memory

used\_memory:1887032

used\_memory\_human:1.80M

used\_memory\_rss:2465792

used\_memory\_peak:1887032

used\_memory\_peak\_human:1.80M

used\_memory\_lua:36864

mem\_fragmentation\_ratio:1.31

mem\_allocator:jemalloc-3.6.0

# Persistence

loading:0

rdb\_changes\_since\_last\_save:0

rdb\_bgsave\_in\_progress:0

rdb\_last\_save\_time:1454710773

rdb\_last\_bgsave\_status:ok

rdb\_last\_bgsave\_time\_sec:0

rdb\_current\_bgsave\_time\_sec:-1

aof\_enabled:0

aof\_rewrite\_in\_progress:0

aof\_rewrite\_scheduled:0

aof\_last\_rewrite\_time\_sec:-1

aof\_current\_rewrite\_time\_sec:-1

aof\_last\_bgrewrite\_status:ok

aof\_last\_write\_status:ok

# Stats

total\_connections\_received:4

total\_commands\_processed:322

instantaneous\_ops\_per\_sec:1

total\_net\_input\_bytes:11371

total\_net\_output\_bytes:1737

instantaneous\_input\_kbps:0.04

instantaneous\_output\_kbps:0.00

rejected\_connections:0

sync\_full:2

sync\_partial\_ok:0

sync\_partial\_err:0

expired\_keys:0

evicted\_keys:0

keyspace\_hits:0

keyspace\_misses:0

pubsub\_channels:0

pubsub\_patterns:0

latest\_fork\_usec:740

migrate\_cached\_sockets:0

# Replication

role:master

connected\_slaves:1

slave0:ip=10.39.1.50,port=6379,state=online,offset=672,lag=0

master\_repl\_offset:672

repl\_backlog\_active:1

repl\_backlog\_size:1048576

repl\_backlog\_first\_byte\_offset:2

repl\_backlog\_histlen:671

# CPU

used\_cpu\_sys:0.26

used\_cpu\_user:0.15

used\_cpu\_sys\_children:0.00

used\_cpu\_user\_children:0.00

# Cluster

cluster\_enabled:0

# Keyspace

db0:keys=17,expires=0,avg\_ttl=0

在监控中如果想统计redis的主从同步中某些便签块，则加上便签块的名字

[root@mysql01 ~]# redis-cli -h localhost -p 6379 info Server

# Server

redis\_version:3.0.7

redis\_git\_sha1:00000000

redis\_git\_dirty:0

redis\_build\_id:49f470b525dbe837

redis\_mode:standalone

os:Linux 2.6.32-573.el6.x86\_64 x86\_64

arch\_bits:64

multiplexing\_api:epoll

gcc\_version:4.4.7

process\_id:1331

run\_id:58c1496d6e8592aa31a10cc3b032b0a091933ccd

tcp\_port:6379

uptime\_in\_seconds:655

uptime\_in\_days:0

hz:10

lru\_clock:11870617

config\_file: