# Redis 使用

教学目标：

a) 了解 NoSQL 及其产品
b) 掌握 Redis 的安装和启动
c) 掌握 Redis 的数据类型
d) 掌握 Redis 的操作命令
e) 掌握 JRedis 的使用
f) 掌握 Redis 的持久化
g) 掌握 Redis 的 replication

## 1. redis 介绍

### 1.1 什么是 NoSQL

NoSQL 锁定

本词条由"科普中国"百科科学词条编写与应用工作项目 审核 。

NoSQL，泛指非关系型的数据库。随着互联网web2.0网站的兴起，传统的关系数据库在应付web2.0网站，特别是超大规模和高并发的SNS类型的web2.0纯动态网站已经显得力不从心，暴露了很多难以克服的问题，而非关系型的数据库则由于其本身的特点得到了非常迅速的发展。NoSQL数据库的产生就是为了解决大规模数据集合多重数据种类带来的挑战，尤其是大数据应用难题。

虽然NoSQL的流行与火起来才短短一年的时间，但是不可否认，现在已经开始了第二代运动。尽管早期的堆栈代码只能算是一种实验，然而现在的系统已经更加的成熟、稳定。不过现在也面临着一个严酷的事实：技术越来越成熟——以至于原来很好的NoSQL数据存储不得不进行重写，也有少数人认为这就是所谓的2.0版本。该工具可以为大数据建立快速、可扩展的存储库。

NoSQL 数据库的产生就是为了解决大规模数据集合多重数据种类带来的挑战，尤其是大数据应用难题。目前一些主流的 NOSQL 产品：

## 1.2 NoSQL 数据库的分类

### 1.2.1 键值(Key-Value)存储数据库

相关产品有 Tokyo Cabinet/Tyrant、Redis、Voldemort、Berkeley DB，主要应用于内容缓存，主要用于处理大量数据的高访问负载。

### 1.2.2 列存储数据库

相关产品有 Cassandra, HBase, Riak，典型应用于分布式的文件系统。

### 1.2.3 文档型数据库

相关产品有 CouchDB、MongoDB，典型应用于 Web 应用，可以突破传统关系型数据库的结构限制，存储数据更为灵活。

### 1.2.4 图形(Graph)数据库

相关数据库有 Neo4J、InfoGrid、Infinite Graph，典型应用于社交网络，存储数据采用图结构利用图结构相关算法。

## 1.3 什么是 redis


Redis 锁定

本词条由"科普中国"百科科学词条编写与应用工作项目 审核 。

Redis是一个开源的使用ANSI C语言编写、支持网络、可基于内存亦可持久化的日志型、Key-Value数据库，并提供多种语言的API。从2010年3月15日起，Redis的开发工作由VMware主持。从2013年5月开始，Redis的开发由Pivotal赞助。

## 1.4 redis 历史发展

2008 年 ,意大利的一家创业公司 Merzia 推出了一款基于 MySQL 的网站实时统计系统 LLOOGG，然而没过多久该公司的创始人 Salvatore Sanfilippo 便对 MySQL 的性能感到失望 ,于是他决定亲自为 LLOOGG 量身定做一个数据库，并于 2009 年开发完成，这个数据库就是 Redis。 不过 Salvatore Sanfilippo

并不满足只将 Redis 用于 LLOOGG 这一款产品，而是希望更多的人使用它，于是在同一年 Salvatore Sanfilippo 将 Redis 开源发布 ,并开始和 Redis 的另一名主要的代码贡献者 Pieter Noordhuis 一起继续着 Redis 的开发，直到今天。

Salvatore Sanfilippo 自己也没有想到，短短的几年时间，Redis 就拥有了庞大的用户群体。Hacker News 在 2012 年发布了一份数据库的使用情况调查，结果显示有近 12%的公司在使用 Redis。国内如新浪微博、街旁网、知乎网，国外如 GitHub、Stack Overflow、Flickr 等都是 Redis 的用户。

VMware 公司从 2010 年开始赞助 Redis 的开发， Salvatore Sanfilippo 和 Pieter Noordhuis 也分别在 3 月和 5 月加入 VMware，全职开发 Redis。完全免费

## 1.5 redis 的应用场景

Redis 广泛应用于缓存，布式集群架构中的 session 分离，聊天室的在线好友列表，任务队列，应用排行榜，网站访问统计，数据过期处理等场景。

# 2. redis 的安装

## 2.1 redis 安装环境

redis 是 C 语言开发，建议在 linux 上运行，本教程使用 Centos6.4 作为安装环境。

安装 redis 需要先将官网下载的源码进行编译，编译依赖 gcc 环境，如果没有 gcc 环境，需要安装 gcc：yum install gcc-c++。

```
Dependency Updated:
  glibc.i686 0:2.12-1.209.el6_9.2     glibc-common.i686 0:2.12-1.209.el6_9.2
  tzdata.noarch 0:2017c-1.el6

Complete!
[root@localhost bin]#
```

## 2.2 redis 安装

我们使用的 Redis 版本为 redis3.0 版本 ,首先可以从官网下载 Redis 压缩包 http://download.redis.io/releases/redis-3.0.0.tar.gz ,然后将下载的 redis 上传到虚拟机的/usr/local 目录下。

```
[root@localhost local]# rz
rz waiting to receive.
 zmodem trl+C d
  100%    1342 KB 1342 KB/s 00:00:01       0 Errors

[root@localhost local]# ls
apache-tomcat-7.0.57          lib
apache-tomcat-7.0.57.tar.gz   libexec
bin                           mysql
etc                           MySQL-5.6.22-1.el6.i686.rpm-bundle.tar
games                         redis-3.0.7.tar.gz
include                       sbin
jdk1.7.0_71                   share
jdk-7u71-linux-i586.tar.gz    src
[root@localhost local]#
```

接下来解压 Redis 压缩包，使用命令：tar -zxvf redis-3.0.7.tar.gz

```
[root@localhost local]# ls
apache-tomcat-7.0.57          libexec
apache-tomcat-7.0.57.tar.gz   mysql
bin                           MySQL-5.6.22-1.el6.i686.rpm-bundle.tar
etc                           redis-3.0.7
games                         redis-3.0.7.tar.gz
include                       sbin
jdk1.7.0_71                   share
jdk-7u71-linux-i586.tar.gz    src
lib
[root@localhost local]#
```

进入解压后的目录进行编译和安装，编译使用命令 make，安装使用命令 make install。安装成功后运行 Redis 服务，如果出现以下界面，则 Redis 安装成功，启动命令：redis-server。

```
[root@localhost redis-3.0.7]# redis-server
5482:C 03 Nov 08:16:22.940 # Warning: no config file specified, using the defaul
t config. In order to specify a config file use redis-server /path/to/redis.conf
5482:M 03 Nov 08:16:22.941 * Increased maximum number of open files to 10032 (it
 was originally set to 1024).
5482:M 03 Nov 08:16:22.942 # Warning: 32 bit instance detected but no memory lim
it set. Setting 3 GB maxmemory limit with 'noeviction' policy now.

                Redis 3.0.7 (00000000/0) 32 bit

                Running in standalone mode
                Port: 6379
                PID: 5482

                      http://redis.io

5482:M 03 Nov 08:16:22.962 # WARNING: The TCP backlog setting of 511 cannot be e
nforced because /proc/sys/net/core/somaxconn is set to the lower value of 128.
5482:M 03 Nov 08:16:22.962 # Server started, Redis version 3.0.7
5482:M 03 Nov 08:16:22.963 # WARNING overcommit_memory is set to 0! Background s
ave may fail under low memory condition. To fix this issue add 'vm.overcommit_me
mory = 1' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.over
commit_memory=1' for this to take effect.
5482:M 03 Nov 08:16:22.963 * The server is now ready to accept connections on po
rt 6379
```

Redis 安装之后，我们在/usr/local/bin 下可以发现编译之后的 Redis 执行文件。

```
[root@localhost local]# cd bin/
[root@localhost bin]# ls
redis-benchmark  redis-check-dump  redis-sentinel
redis-check-aof  redis-cli         redis-server
[root@localhost bin]#
```

## 2.3 redis 的启动

### 2.3.1 前端模式启动

直接运行 redis-server 将以前端模式启动，前端模式启动的缺点是占用终端，同时 ssh 命令窗口关闭则 redis-server 程序结束，所以不推荐使用此方法。

### 2.3.2 后端模式启动

修改 redis.conf 配置文件， 将 daemonize 设置 yes 以后即可以后端模式来启动 Redis。

```
# at the beginning of this file to avoid overwriting config change at runtime.
#
# If instead you are interested in using includes to override configuration
# options, it is better to use include as the last line.
#
# include /path/to/local.conf
# include /path/to/other.conf

################################ GENERAL  #####################################

# By default Redis does not run as a daemon. Use 'yes' if you need it.
# Note that Redis will write a pid file in /var/run/redis.pid when daemonized.
daemonize yes

# When running daemonized, Redis writes a pid file in /var/run/redis.pid by
# default. You can specify a custom pid file location here.
pidfile /var/run/redis.pid

-- INSERT --
```

修改之后用当前修改后的配置文件来启动 Redis 服务器。

```
[root@localhost redis-3.0.7]# redis-server ./redis.conf
[root@localhost redis-3.0.7]#
```

此时我们可以使用命令的方式查看服务是否已经启动。

```
[root@localhost redis-3.0.7]# redis-server ./redis.conf
[root@localhost redis-3.0.7]# ps -ef | grep redis
root      5524     1  0 08:28 ?        00:00:00 redis-server *:6379
root      5529  1717  0 08:28 pts/0    00:00:00 grep redis
[root@localhost redis-3.0.7]#
```

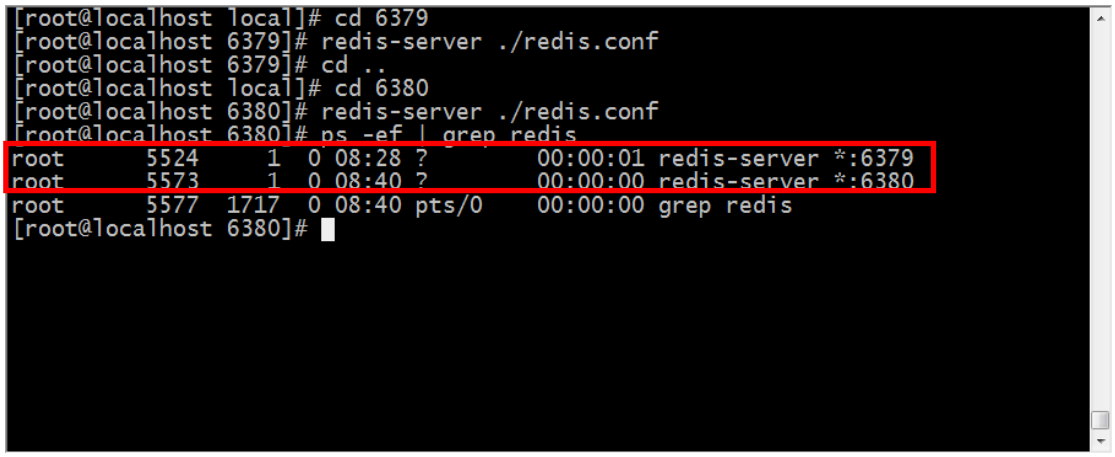
我们发现 Redis 服务已经启动，默认使用 6379 端口。

### 2.3.3 启动多个 redis 进程

启动多个 Redis 服务，一般采用定义多个对应的配置文件的方式来启动不同配置的 Redis 服务，同时需要分别修改每一个 Redis 服务器启动的端口号。这里我们启动两个 Redis 服务器，分别使用 6379 和 6380 端口。

首先我们创建两个目录。


在两个目录中定义 redis 的配置文件 ,这里我们将 redis 默认的配置文件拷贝过来，并修改对应的端口号，然后分别启动两个 Redis 服务。




## 2.4 redis 的停止

要停止正在运行的 Redis 服务，可以采用 Linux 命令的方式直接杀掉 Redis 服务的进程，也可以使用 Redis 客户端想服务器发送 shutdown 指令来关闭

Redis 服务器。

```
[root@localhost 6380]# redis-cli shutdown
[root@localhost 6380]# ps -ef | grep redis
root      5573     1  0 08:40 ?        00:00:00 redis-server *:6380
root      5583  1717  0 08:43 pts/0    00:00:00 grep redis
```

## 2.5 使用 redis 客户端

在 redis 的安装目录中有 redis 的客户端，即 redis-cli（Redis Command Line Interface），它是 Redis 自带的基于命令行的 Redis 客户端。

### 2.5.1 连接 Redis 服务器

使用 redis-cli 命令即可连接默认的本机 6379 端口的 Redis 服务器。如下图表示连接成功。

```
[root@localhost 6379]# redis-cli
127.0.0.1:6379>
```

如果希望远程连接其他主机或者其他端口的 Redis 服务器，可以加上以下的参数，使用-h 设置远程主机的 ip，使用-p 设置远程主机的端口号。

```
[root@localhost 6379]# redis-cli -h 192.168.5.128 -p 6380
192.168.5.128:6380>
```

### 2.5.2 向 Redis 服务器发送命令

redis-cli 连上 redis 服务后，可以在命令行向 Redis 服务器发送命令。

#### 1.5.1.1 ping 命令

Redis 提供了 PING 命令来测试客户端与 Redis 的连接是否正常，如果连接正常会收到回复 PONG。

```
[root@localhost 6379]# redis-cli -h 192.168.5.128 -p 6380
192.168.5.128:6380> ping
PONG
192.168.5.128:6380>
```

#### 1.5.1.2 set/get 命令

使用 set 和 get 可以向 redis 设置数据、获取数据。

```
192.168.5.128:6380> set name zhangsan
OK
192.168.5.128:6380> get name
"zhangsan"
192.168.5.128:6380>
```

#### 1.5.1.3 del 命令

del 命令用来删除指定 key 的内容。

```
192.168.5.128:6380> del name
(integer) 1
192.168.5.128:6380> get name
(nil)
192.168.5.128:6380>
```

#### 1.5.1.4 keys*命令

keys*命令用来查看当前 redis 数据库中存在的 key 的名称。

```
192.168.5.128:6380> set name zhangsan
OK
192.168.5.128:6380> set age 30
OK
192.168.5.128:6380> keys *
1) "age"
2) "name"
192.168.5.128:6380>
```

#### 1.5.1.5 flushdb 命令

flushdb 命令用来清空当前数据库中的所有数据。

```
192.168.5.128:6380> keys *
1) "age"
2) "name"
192.168.5.128:6380> flushdb
OK
192.168.5.128:6380> keys *
(empty list or set)
192.168.5.128:6380>
```

## 2.6 Redis 多数据库

一个 Redis 数据库可以包括多个数据库，客户端可以指定连接某个 redis 实例的哪个数据库，就好比一个 mysql 中创建多个数据库，客户端连接时指定连接哪个数据库。一个 redis 实例默认提供 16 个数据库，这由启动的配置文件中指定，下标从 0 到 15，客户端默认连接第 0 号数据库，也可以通过 select 选择连接哪个数据库，不同数据库中的数据不能共享。

```
192.168.5.128:6380> select 0
OK
192.168.5.128:6380> keys *
1) "age"
2) "name"
192.168.5.128:6380> select 1
OK
192.168.5.128:6380[1]> keys *
(empty list or set)
192.168.5.128:6380[1]>
```

需要注意的问题是，如果我们使用 flushall 命令清空某一个库中的数据时，其他库中的数据也会被清空，因此，在我们不同的业务访问 Redis 服务器时应该选择不同的 Redis 服务器，而不是同一个服务器内部的不同的数据库。

```
192.168.5.128:6380[1]> select 0
OK
192.168.5.128:6380> keys *
1) "age"
2) "name"
192.168.5.128:6380> select 1
OK
192.168.5.128:6380[1]> keys *
1) "k2"
2) "k1"
192.168.5.128:6380[1]> flushall
OK
192.168.5.128:6380[1]> keys *
(empty list or set)
192.168.5.128:6380[1]> select 0
OK
192.168.5.128:6380> keys *
(empty list or set)
192.168.5.128:6380>
```

# 3. Jedis 的使用

## 3.1 Jedis 介绍

Redis 不仅是使用命令来操作，现在基本上主流的语言都有客户端支持，比如 java、C、C#、C++、php、Node.js、Go 等。

在官方网站里列一些 Java 的客户端，有 Jedis、Redisson、Jredis、JDBC-Redis、等其中官方推荐使用 Jedis 和 Redisson。 在企业中用的最多的就是 Jedis，下面我们就重点学习下 Jedis。

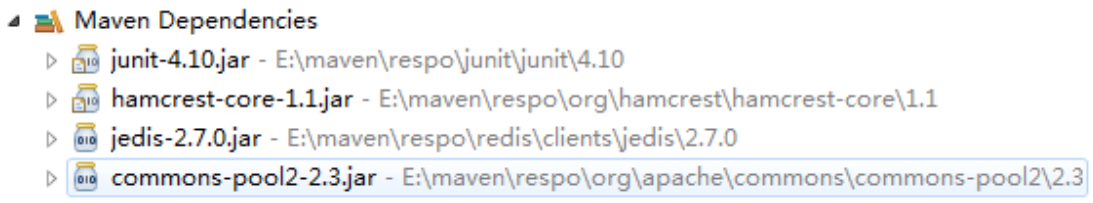
## 3.2 通过 jedis 连接 redis

### 3.2.1 添加 jar 包

要使用 Jedis 需要加入 Jedis 对应的 jar 包。我们使用 maven 来构建工程，因此直接使用 jedis 的 jar 包依赖坐标即可。

```
<dependency>
    <groupId>redis.clients</groupId>
    <artifactId>jedis</artifactId>
    <version>2.7.0</version>
</dependency>
```

Maven 会自动在工程中添加 jedis 需要依赖的 jar 包。




### 3.2.2 单实例连接 Redis

通过创建单实例 jedis 对象连接 redis 服务，直接创建 Jedis 对象即可，如下

代码：

```
package com.igeek.redis_jedis_01;

import org.junit.Test;

import redis.clients.jedis.Jedis;

public class AppTest{
    @Test
    public void testJedisSingle() {

        //创建Jedis对象，传递两个参数：第一个host为redis服务器的名称，
第二个为redis服务器的端口号
        Jedis jedis = new Jedis("192.168.5.128", 6379);

        //进行设值操作
        jedis.set("username","zhangsan");

        //从redis中取值
        String username = jedis.get("username");

        System.out.println("username:"+username);
    }
}
```

### 3.2.3 使用连接池连接 Redis

通过单实例连接 redis 不能对 redis 连接进行共享，可以使用连接池对 redis 连接进行共享，提高资源利用率，使用 jedisPool 连接 redis 服务，如下代码：

```
package com.igeek.redis_jedis_01;

import org.junit.Test;

import redis.clients.jedis.Jedis;
import redis.clients.jedis.JedisPool;
import redis.clients.jedis.JedisPoolConfig;

public class AppTest{

    @Test
```

```
    public void testJedisPool(){
        //创建JedisPool连接池配置
        JedisPoolConfig jedisPoolConfig = new JedisPoolConfig();

        //设置连接池的最大连接数
        jedisPoolConfig.setMaxTotal(20);

        //设置连接池的最大连接空闲数
        jedisPoolConfig.setMaxIdle(2);

        //设置redis服务器的主机
        String host = "192.168.5.128";
        //设置redis服务器的端口
        int port = 6379;
        //创建JedisPool连接池对象
        JedisPool jedisPool = new JedisPool(jedisPoolConfig, host,
port);

        //获取Jedis对象
        Jedis jedis = jedisPool.getResource();
        //存入数据
        jedis.set("info", "message");

        //获取数据
        String info = jedis.get("info");

        System.out.println("info:"+info);

        //关闭资源
        jedisPool.close();
    }
}
```

# 4. redis 的数据类型

## 4.1 String 类型

redis 中没有使用 C 语言的字符串表示，而是自定义一个数据结构叫 SDS（simple dynamic string )即简单动态字符串。redis 的字符串是二进制安全的，

存入什么数据取出的还是什么数据。

String 类型的的常用命令有：

### 4.1.1 赋值命令

redis 中赋值操作使用 set 命令，语法 set key value。

范例：向 redis 中插入数据，键为 name，值为 zhangsan。

```
127.0.0.1:6379> keys *
(empty list or set)
127.0.0.1:6379> set name zhangsan
OK
127.0.0.1:6379> keys *
1) "name"
127.0.0.1:6379>
```

如果希望一次性插入多条数据，可以使用 mset 命令，语法 mset key1 value1 key2 value2 …

范例：一次性向 redis 中插入以下数据，id 为 1，age 为 20。

```
127.0.0.1:6379> keys *
1) "name"
127.0.0.1:6379> mset id 1 age 20
OK
127.0.0.1:6379> keys *
1) "id"
2) "name"
3) "age"
127.0.0.1:6379>
```

### 4.1.2 取值命令

redis 中进行取值操作使用 get 命令，语法：get key。

范例：获取键为 name 的数据的值

```
127.0.0.1:6379> get name
"zhangsan"
127.0.0.1:6379>
```

需要注意的是，如果通过 get 命令获取数据时，给定的键不存在，则返回(nil)，表示查询结果为空。

如果希望一次性从 redis 中获取多个键对应的数据的值，可以使用 mget 命令，语法：mget key1 key2 …

范例：获取 redis 中 id，name，age 的数据的值。

```
127.0.0.1:6379> mget id name age
1) "1"
2) "zhangsan"
3) "20"
127.0.0.1:6379>
```

我们也可以使用 getset 命令，在获取对应键的数据的同时，为该键指定新的值。语法：getset key newValue。

范例：获取 name 对应的名称，并将名称的值改为 lisi。

```
127.0.0.1:6379> get name
"zhangsan"
127.0.0.1:6379> getset name lisi
"zhangsan"
127.0.0.1:6379> get name
"lisi"
127.0.0.1:6379>
```

### 4.1.3 删除命令

在 Redis 中删除数据使用 del 命令，语法：del key。

范例：删除 id 对应的数据的值。

```
127.0.0.1:6379> keys *
1) "id"
2) "name"
3) "age"
127.0.0.1:6379> del id
(integer) 1
127.0.0.1:6379> keys *
1) "name"
2) "age"
127.0.0.1:6379>
```

如果需要直接清空该库中所有数据，可以使用 flushdb 命令。

范例：清空当前数据库中的数据。

```
127.0.0.1:6379> keys *
1) "name"
2) "age"
127.0.0.1:6379> flushdb
OK
127.0.0.1:6379> keys *
(empty list or set)
127.0.0.1:6379>
```

如果要清空所有库中的所有数据使用 flushall 命令。

范例：清理 redis 所有库中的所有数据。

```
127.0.0.1:6379[1]> select 0
OK
127.0.0.1:6379> keys *
1) "k2"
2) "k1"
127.0.0.1:6379> select 1
OK
127.0.0.1:6379[1]> keys *
1) "k2"
2) "k1"
127.0.0.1:6379[1]> flushall
OK
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>
```

### 4.1.4 数值递增

当存储的字符串是整数时，Redis 提供了一个实用的命令 incr，其作用是让当前键值递增，并返回递增后的值。语法:incr key。

范例：向数据库中插入数据 num 初始值为 1，然后自增加 1。

```
127.0.0.1:6379> set num 1
OK
127.0.0.1:6379> get num
"1"
127.0.0.1:6379> incr num
(integer) 2
127.0.0.1:6379>
```

以上的自增表示每次加 1，如果需要按照指定的数据来进行增长，可以使用 incrby 命令，语法：incrby key increment。

范例：使用 incrby 命令将 num 的数据值加 10。

```
127.0.0.1:6379> get num
"2"
127.0.0.1:6379> incrby num 10
(integer) 12
127.0.0.1:6379>
```

### 4.1.5 数值递减

当需要对数据进行递减操作时，使用 decr 命令，语法：decr key。

范例：将 num 对应数据的值递减 1。

```
127.0.0.1:6379> get num
"12"
127.0.0.1:6379> decr num
(integer) 11
127.0.0.1:6379>
```

如果希望递减指定的数值 ,可以使用 decrby ,语法 :decrby key decrement。

范例：将 num 对应数据的值减 10.

```
127.0.0.1:6379> get num
"11"
127.0.0.1:6379> decrby num 10
(integer) 1
127.0.0.1:6379>
```

### 4.1.6 追加数据

在 Redis 中可以使用 append 命令向指定数据的末尾追加内容，语法：append key value。

范例：向 info 数据的末尾添加内容。

```
127.0.0.1:6379> set info "Hello"
OK
127.0.0.1:6379> append info " igeek"
(integer) 11
127.0.0.1:6379> get info
"Hello igeek"
127.0.0.1:6379>
```

注意：在插入数据时，字符串的双引号默认可以省略，但是如果字符串中包含空格的情况，需要加上双引号，否则空格会被忽略。

### 4.1.7 获取长度

使用 strlen 命令可以返回指定键对应的值的长度，如果键不存在则返回 0。

语法：strlen key。

范例：获取 info 对应的数据的长度。

```
127.0.0.1:6379> strlen info
(integer) 11
127.0.0.1:6379>
```

## 4.2 hash 类型

hash 叫散列类型，它提供了字段和字段值的映射。字段值只能是字符串类型，不支持散列类型、集合类型等其它类型。

hash 类型的的常用命令有：

### 4.2.1 赋值命令

向 hash 中插入获设置数据没使用 hset 命令，语法：hset key field value。

范例：向 redis 中插入 user 的 name 属性值为 songjiang。

```
127.0.0.1:6379> hset user name zhangsan
(integer) 1
127.0.0.1:6379>
```

如果希望从 redis 中一次性设置多个字段的值，可以使用 hmset，语法：hmset key field1 value1 field2 value2 ...

范例：向 redis 中插入 user 的 id 和 age 属性。

```
127.0.0.1:6379> hmset user id 1 age 20
OK
127.0.0.1:6379>
```

### 4.2.2 取值命令

从 hash 中获取数据使用 hget 命令，语法：hget key field。

范例：获取 user 的 name 属性的值。

```
127.0.0.1:6379> hget user name
"zhangsan"
127.0.0.1:6379>
```

使用 hmget 可以一次性获取多个字段的值。语法 :hmget key field1 field2 field3 ...

范例：获取 user 的 id，name，age 字段的值。

```
127.0.0.1:6379> hmget user id name age
1) "1"
2) "zhangsan"
3) "20"
127.0.0.1:6379>
```

可以使用 hgetall 命令，或者指定键的所有字段和字段的值。语法：hgetall key。

```
127.0.0.1:6379> hgetall user
1) "name"
2) "zhangsan"
3) "id"
4) "1"
5) "age"
6) "20"
127.0.0.1:6379>
```

使用 HKEYS 和 HVALS 命令可以分别用来获取指定键的所有键和所有值。

范例：获取 user 的所有键和所有值。

```
127.0.0.1:6379> hmset user id 1 name zhangsan age 20
OK
127.0.0.1:6379> hkeys user
1) "id"
2) "name"
3) "age"
127.0.0.1:6379> hvals user
1) "1"
2) "zhangsan"
3) "20"
127.0.0.1:6379>
```

### 4.2.3 判断字段是否存在

判断字段是否存在使用 HEXISTS 命令，如果存在则返回 1，如果不存在则返回 0 ，语法：HEXISTS key field

范例：判断 user 是否存在。

```
127.0.0.1:6379> hexists user name
(integer) 1
127.0.0.1:6379> flushdb
OK
127.0.0.1:6379> hexists user name
(integer) 0
127.0.0.1:6379>
```

### 4.2.4 获取字段数量

获取字段数量使用 HLEN 命令，如果键不存在则返回 0，语法 HLEN key。

范例：获取指定键的数据中键的数量。

```
127.0.0.1:6379> hmset user id 1 name zhangsan age 20
OK
127.0.0.1:6379> hlen user
(integer) 3
127.0.0.1:6379> hlen user1
(integer) 0
127.0.0.1:6379>
```

### 4.2.5 删除命令

要删除一个或多个字段，使用 hdel 命令，语法：HDEL key field [field ...]。

范例：删除 user 的 id 字段的值。

```
127.0.0.1:6379> hgetall user
1) "name"
2) "zhangsan"
3) "id"
4) "1"
5) "age"
6) "20"
127.0.0.1:6379> hdel user id
(integer) 1
127.0.0.1:6379> hgetall user
1) "name"
2) "zhangsan"
3) "age"
4) "20"
127.0.0.1:6379>
```

范例：删除 user 的 name 和 age 字段。

```
127.0.0.1:6379> hgetall user
1) "name"
2) "zhangsan"
3) "age"
4) "20"
127.0.0.1:6379> hdel user name age
(integer) 2
127.0.0.1:6379> hgetall user
(empty list or set)
127.0.0.1:6379>
```

### 4.2.6 增加数字

如果 hash 中的字段存储的值为数值类型，可以进行自增操作。语法：HINCRBY key field increment。

范例：将 user 的 age 字段的值增加 1.

```
127.0.0.1:6379> hmset user id 1 name zhangsan age 20
OK
127.0.0.1:6379> hincrby user age 1
(integer) 21
127.0.0.1:6379> hget user age
"21"
127.0.0.1:6379>
```

## 4.3 list 类型

列表类型（list）可以存储一个有序的字符串列表，常用的操作是向列表两端添加元素，或者获得列表的某一个片段。

列表类型内部是使用双向链表（double linked list）实现的，所以向列表两端添加元素的时间复杂度为 0(1)，获取越接近两端的元素速度就越快。这意味着即使是一个有几千万个元素的列表，获取头部或尾部的 10 条记录也是极快的。

list 类型的的常用命令有：

### 4.3.1 添加数据

Redis 列表采用双向链表实现，因此在插入数据时可以从两端分别操作，向列表左边增加元素使用 lpush 命令，向列表右边增加元素使用 rpush 命令。

范例：向列表中插入数据

```
127.0.0.1:6379> lpush nums 1
(integer) 1
127.0.0.1:6379> lpush nums 2
(integer) 2
127.0.0.1:6379> lpush nums 3
(integer) 3
127.0.0.1:6379> lpush nums 4
(integer) 4
127.0.0.1:6379> lpush nums 5
(integer) 5
127.0.0.1:6379> rpush nums2 a
(integer) 1
127.0.0.1:6379> rpush nums2 b
(integer) 2
127.0.0.1:6379> rpush nums2 c
(integer) 3
127.0.0.1:6379> rpush nums2 d
(integer) 4
127.0.0.1:6379> rpush nums2 e
(integer) 5
127.0.0.1:6379>
```

### 4.3.2 查看列表

LRANGE 命令是列表类型最常用的命令之一，获取列表中的某一片段，将返回 start、stop 之间的所有元素（包含两端的元素），索引从 0 开始。索引可以是负数，如：“-1”代表最后边的一个元素。

范例：获取列表 nums 和 nums2 中的数据

```
127.0.0.1:6379> lrange nums 0 4
1) "5"
2) "4"
3) "3"
4) "2"
5) "1"
127.0.0.1:6379> lrange nums2 0 4
1) "a"
2) "b"
3) "c"
4) "d"
5) "e"
127.0.0.1:6379>
```

从以上获取的数据结构来看，我们发现从列表的左侧插入数据，在列表中数据以倒叙存放，从列表的右侧插入数据则是正序存放。

### 4.3.3 获取数据

在列表中可以使用 lpop 和 rpop 分别列表的左侧和右侧弹出一个数据，该数据会首先返回，然后再从原列表中删除。

范例：弹出 nums 的左侧数据。

```
127.0.0.1:6379> lpop nums
"5"
127.0.0.1:6379> lrange nums 0 -1
1) "4"
2) "3"
3) "2"
4) "1"
127.0.0.1:6379>
```

范例：弹出 nums 的右侧数据。

```
127.0.0.1:6379> rpop nums
"1"
127.0.0.1:6379> lrange nums 0 -1
1) "4"
2) "3"
3) "2"
127.0.0.1:6379>
```

### 4.3.4 获取列表中元素的个数

使用 LLEN 命令可以获取列表中元素的个数。范例：LLEN key

范例：获取 nums2 列表中元素的个数。

```
127.0.0.1:6379> llen nums2
(integer) 5
127.0.0.1:6379>
```

### 4.3.5 删除列表中指定的值

LREM 命令会删除列表中前 count 个值为 value 的元素 ,返回实际删除的元素个数。根据 count 值的不同，该命令的执行方式会有所不同：

- 当 count>0 时， LREM 会从列表左边开始删除。

- 当 count<0 时， LREM 会从列表后边开始删除。
- 当 count=0 时， LREM 删除所有值为 value 的元素。

范例：删除 nums 中的 4 元素

```
127.0.0.1:6379> lrange nums 0 -1
1) "4"
2) "3"
3) "2"
127.0.0.1:6379> lrem nums 0 4
(integer) 1
127.0.0.1:6379> lrange nums 0 -1
1) "3"
2) "2"
127.0.0.1:6379>
```

### 4.3.6 获得/设置指定索引的元素值

获得或者设置指定索引的元素值分别使用 lindex 和 lset 命令。

范例：获取 nums 列表中索引 2 对应的数据

```
127.0.0.1:6379> lrange nums 0 -1
1) "1"
2) "2"
3) "3"
4) "4"
5) "5"
127.0.0.1:6379> lindex nums 2
"3"
127.0.0.1:6379>
```

范例：将 nums 中索引 2 对应的数据改为 x

```
127.0.0.1:6379> lset nums 2 x
OK
127.0.0.1:6379> lrange nums 0 -1
1) "1"
2) "2"
3) "x"
4) "4"
5) "5"
127.0.0.1:6379>
```

### 4.3.7 只保留列表指定片段

List 中使用 ltrim 命令对列表中的数据进行截取操作 ,只保留指定数据片段。

语法：ltrim key start stop。

范例：截取 nums 列表中索引从 1 开始到 3 的数据

```
127.0.0.1:6379> lrange nums 0 -1
1) "1"
2) "2"
3) "x"
4) "4"
5) "5"
127.0.0.1:6379> ltrim nums 1 3
OK
127.0.0.1:6379> lrange nums 0 -1
1) "2"
2) "x"
3) "4"
127.0.0.1:6379>
```

### 4.3.8 向列表中插入元素

向列表中指定位置插入元素可以使用 linsert 命令，语法：LINSERT key BEFORE|AFTER pivot value。

范例：向 nums 列表中的 x 元素后面插入元素 y

```
127.0.0.1:6379> lrange nums 0 -1
1) "2"
2) "x"
3) "4"
127.0.0.1:6379> linsert nums after x y
(integer) 4
127.0.0.1:6379> lrange nums 0 -1
1) "2"
2) "x"
3) "y"
4) "4"
127.0.0.1:6379>
```

### 4.3.9 将元素从一个列表转移到另一个列表中

使用 RPOPLPUSH 命令可以将元素从一个列表转移到另一个列表中。语法：RPOPLPUSH source destination。

范例：将元素从一个列表转移到另一个列表中

```
127.0.0.1:6379> lrange list 0 -1
1) "5"
2) "4"
3) "3"
4) "2"
5) "1"
127.0.0.1:6379> rpoplpush list newList
"1"
127.0.0.1:6379> lrange list 0 -1
1) "5"
2) "4"
3) "3"
4) "2"
127.0.0.1:6379> lrange newList 0 -1
1) "1"
127.0.0.1:6379>
```

## 4.4 set 类型

集合类型的常用操作是向集合中加入或删除元素、判断某个元素是否存在等，由于集合类型的 Redis 内部是使用值为空的散列表实现，所有这些操作的时间复杂度都为 0(1)。 Redis 还提供了多个集合之间的交集、并集、差集的运算。

set 类型的的常用命令有：

### 4.4.1 增加/删除元素

集合中添加数据使用 sadd 命令，语法 sadd key member [member...].

范例：向 set 集合中添加数据

```
127.0.0.1:6379> sadd set 10
(integer) 1
127.0.0.1:6379> sadd set 20
(integer) 1
127.0.0.1:6379>
```

### 4.4.2 获得集合中的所有元素

获取集合中所有元素使用 SMEMBERS 命令，语法：SMEMBERS key

范例：获取 set 集合中的所有元素

```
127.0.0.1:6379> smembers set
1) "10"
2) "20"
127.0.0.1:6379>
```

### 4.4.3 集合的差集运算

属于 A 并且不属于 B 的元素构成的集合 ,使用命令 SDIFF ,语法 :SDIFF key [key ...]

范例：集合的差集

```
127.0.0.1:6379> sadd a 10 20 30 40
(integer) 4
127.0.0.1:6379> sadd b 20 40 60
(integer) 3
127.0.0.1:6379> sdiff a b
1) "10"
2) "30"
127.0.0.1:6379>
```

### 4.4.4 集合的交集运算

属于 A 并且也属于 B 的元素构成的集合 , 使用命令 SINTER , 语法 : SINTER key [key ...]

范例：集合的交集

```
127.0.0.1:6379> sinter a b
1) "20"
2) "40"
127.0.0.1:6379>
```

### 4.4.5 集合的并集运算

属于 A 并且或者属于 B 的元素构成的集合，使用命令 SUNION，语法：SUNION key [key ...]

范例：集合的并集

```
127.0.0.1:6379> sunion a b
1) "10"
2) "20"
3) "30"
4) "40"
5) "60"
127.0.0.1:6379>
```

### 4.4.6 获得集合中元素的个数

获得集合中元素的个数，使用 SCARD 命令，如果集合不存在则返回 0，语法：SCARD key

范例：获取集合 set 中元素的个数

```
127.0.0.1:6379> sadd set 10 20 30 40 50
(integer) 5
127.0.0.1:6379> scard set
(integer) 5
127.0.0.1:6379> scard set1
(integer) 0
127.0.0.1:6379>
```

### 4.4.7 从集合中弹出一个元素

从集合中弹出一个元素使用 spop 命令，语法：spop key

范例：从 set 集合中弹出一个元素

```
127.0.0.1:6379> smembers set
1) "10"
2) "20"
3) "30"
4) "40"
5) "50"
127.0.0.1:6379> spop set
"10"
127.0.0.1:6379> smembers set
1) "20"
2) "30"
3) "40"
4) "50"
127.0.0.1:6379>
```

## 4.5 sorted set 类型

在集合类型的基础上有序集合类型为集合中的每个元素都关联一个分数，这

使得我们不仅可以完成插入、删除和判断元素是否存在在集合中，还能够获得分数最高或最低的前 N 个元素、获取指定分数范围内的元素等与分数有关的操作。在某些方面有序集合和列表类型有些相似。

1、二者都是有序的。

2、二者都可以获得某一范围的元素。

但是，二者有着很大区别：

1、列表类型是通过链表实现的，获取靠近两端的数据速度极快，而当元素增多后，访问中间数据的速度会变慢。

2、有序集合类型使用散列表实现，所有即使读取位于中间部分的数据也很快。

3、列表中不能简单的调整某个元素的位置，但是有序集合可以（通过更改分数实现）

4、有序集合要比列表类型更耗内存。

sorted set 类型的的常用命令有：

### 4.5.1 增加元素

向有序集合中加入一个元素和该元素的分数，如果该元素已经存在则会用新的分数替换原有的分数。返回值是新加入到集合中的元素个数，不包含之前已经存在的元素。语法：ZADD key score member [score member ...]

范例：向有序集合中添加数据

```
127.0.0.1:6379> zadd score 80 zhangsan 75 lisi 58 wangwu
(integer) 3
127.0.0.1:6379>
```

### 4.5.2 获得排名在某个范围的元素列表

获得排名在某个范围的元素列表使用 zrange 命令，语法：ZRANGE key start stop [WITHSCORES]

范例：获取 score 中的元素

```
127.0.0.1:6379> zrange score 0 -1
1) "wangwu"
2) "lisi"
3) "zhangsan"
127.0.0.1:6379> zrange score 0 -1 withscores
1) "wangwu"
2) "58"
3) "lisi"
4) "75"
5) "zhangsan"
6) "80"
127.0.0.1:6379>
```

以上是根据分数进行升序排列，如果要进行降序排列，我们使用命令 ZREVRANGE。

范例：降序获取 score 中的元素

```
127.0.0.1:6379> zrevrange score 0 -1
1) "zhangsan"
2) "lisi"
3) "wangwu"
127.0.0.1:6379> zrevrange score 0 -1 withscores
1) "zhangsan"
2) "80"
3) "lisi"
4) "75"
5) "wangwu"
6) "58"
127.0.0.1:6379>
```

### 4.5.3 删除元素

移除有序集 key 中的一个或多个成员使用命令 ZREM ,不存在的成员将被忽略。语法：ZREM key member [member ...]

范例：移除 score 中 zhangsan 元素

```
127.0.0.1:6379> zrange score 0 -1
1) "wangwu"
2) "lisi"
3) "zhangsan"
127.0.0.1:6379> zrem score zhangsan
(integer) 1
127.0.0.1:6379> zrange score 0 -1
1) "wangwu"
2) "lisi"
127.0.0.1:6379>
```

### 4.5.4 获得指定分数范围的元素

获得指定分数范围的元素使用 ZRANGEBYSCORE 命令，语法：ZRANGEBYSCORE key min max [WITHSCORES] [LIMIT offset count]

范例：获取 score 中分数在 60 – 80 分之间的数据元素。

```
127.0.0.1:6379> zrange score 0 -1 withscores
1) "wangwu"
2) "58"
3) "zhaoliu"
4) "60"
5) "lisi"
6) "75"
7) "qianqi"
8) "95"
127.0.0.1:6379> zrangebyscore score 60 80
1) "zhaoliu"
2) "lisi"
127.0.0.1:6379> zrangebyscore score 60 80 withscores
1) "zhaoliu"
2) "60"
3) "lisi"
4) "75"
127.0.0.1:6379>
```

### 4.5.5 增加某个元素的分数，返回值是更改后的分数

增加某个元素的分数，返回值是更改后的分数，使用命令 ZINCRBY，语法：ZINCRBY key increment member

范例：给 lisi 加 4 分

```
127.0.0.1:6379> zrange score 0 -1 withscores
1) "wangwu"
2) "58"
3) "zhaoliu"
4) "60"
5) "lisi"
6) "75"
7) "qianqi"
8) "95"
127.0.0.1:6379> zincrby score 4 lisi
"79"
127.0.0.1:6379> zrange score 0 -1 withscores
1) "wangwu"
2) "58"
3) "zhaoliu"
4) "60"
5) "lisi"
6) "79"
7) "qianqi"
8) "95"
127.0.0.1:6379>
```

### 4.5.6 获得集合中元素的数量

获取集合中元素的数量使用 ZCARD 命令，语法：ZCARD key。

范例：获取 score 集合中元素的个数

```
127.0.0.1:6379> zcard score
(integer) 4
127.0.0.1:6379>
```

### 4.5.7 获得指定分数范围内的元素个数

获得指定分数范围内的元素个数使用命令 zcount ,语法 :ZCOUNT key min max。

范例：获取 score 中分数在 70-90 之间的元素个数

```
127.0.0.1:6379> zrange score 0 -1 withscores
1) "wangwu"
2) "58"
3) "zhaoliu"
4) "60"
5) "lisi"
6) "79"
7) "qianqi"
8) "95"
127.0.0.1:6379> zcount score 70 90
(integer) 1
127.0.0.1:6379>
```

### 4.5.8 按照排名范围删除元素

按照排名范围删除元素使用命令 ZREMRANGEBYRANK，语法：ZREMRANGEBYRANK key start stop

范例：删除 score 中排名前两位的元素

```
127.0.0.1:6379> zrange score 0 -1 withscores
1) "wangwu"
2) "58"
3) "zhaoliu"
4) "60"
5) "lisi"
6) "79"
7) "qianqi"
8) "95"
127.0.0.1:6379> zremrangebyrank score 0 1
(integer) 2
127.0.0.1:6379> zrange score 0 -1 withscores
1) "lisi"
2) "79"
3) "qianqi"
4) "95"
127.0.0.1:6379>
```

### 4.5.9 按照分数范围删除元素

按照分数范围删除元素使用命令 ZREMRANGEBYSCORE，语法：ZREMRANGEBYSCORE key start stop

范例：删除 score 中分数在 60 -80 之间的元素

```
127.0.0.1:6379> zrange score 0 -1 withscores
 1) "wangwu"
 2) "60"
 3) "zhangsan"
 4) "65"
 5) "lisi"
 6) "79"
 7) "zhaoliu"
 8) "80"
 9) "qianqi"
10) "95"
127.0.0.1:6379> zremrangebyscore score 60 80
(integer) 4
127.0.0.1:6379> zrange score 0 -1 withscores
1) "qianqi"
2) "95"
127.0.0.1:6379>
```

### 4.5.10 获取元素的排名

获取元素的排名使用 zrank 命令，语法：ZRANK key member

范例：查看 score 中 qianqi 的排名

```
127.0.0.1:6379> zrange score 0 -1 withscores
1) "wangwu"
2) "60"
3) "zhaoliu"
4) "80"
5) "qianqi"
6) "95"
127.0.0.1:6379> zrank score qianqi
(integer) 2
127.0.0.1:6379>
```

以上是升序后的排名，如果要使用倒序排名使用命令 ZREVRANK。

范例：查看 score 中 qianqi 的倒序排名

```
127.0.0.1:6379> zrange score 0 -1 withscores
1) "wangwu"
2) "60"
3) "zhaoliu"
4) "80"
5) "qianqi"
6) "95"
127.0.0.1:6379> zrank score qianqi
(integer) 2
127.0.0.1:6379> zrevrank score qianqi
(integer) 0
127.0.0.1:6379>
```

# 5. redis 的其他命令

## 5.1 设置 key 的生存时间

Redis 在实际使用过程中更多的用作缓存，然而缓存的数据一般都是需要设置生存时间的，即：到期后数据销毁。

| | |
|---|---|
| EXPIRE key seconds | 设置 key 的生存时间（单位：秒）key 在多少秒后会自动删除 |
| TTL key | 查看 key 生于的生存时间 |
| PERSIST key | 清除生存时间 |
| PEXPIRE key milliseconds | 生存时间设置单位为：毫秒 |

范例：key 生存时间的设置

```
127.0.0.1:6379> set k1 v1
OK
127.0.0.1:6379> get k1
"v1"
127.0.0.1:6379> expire k1 5
(integer) 1
127.0.0.1:6379> ttl k1
(integer) 2
127.0.0.1:6379> ttl k1
(integer) 1
127.0.0.1:6379> ttl k1
(integer) -2
127.0.0.1:6379> get k1
(nil)
127.0.0.1:6379>
```

## 5.2 返回满足给定 pattern 的所有 key

Redis 中可以使用 pattern 来获取键的名称。

范例：查看所有以 user 开头的键

```
127.0.0.1:6379> keys *
1) "k2"
2) "k1"
3) "userage"
4) "username"
127.0.0.1:6379> keys user*
1) "userage"
2) "username"
127.0.0.1:6379>
```

## 5.3 重命名 key

重命名 key 使用命令 rename，语法：rename name newName

范例：将名为 username 的键改为 user_name

```
127.0.0.1:6379> keys *
1) "k2"
2) "k1"
3) "userage"
4) "username"
127.0.0.1:6379> rename username user_name
OK
127.0.0.1:6379> keys *
1) "k1"
2) "k2"
3) "userage"
4) "user_name"
127.0.0.1:6379>
```

## 5.4 返回值的类型

使用 type 命令可以返回指定键的值的类型，语法：type key。

范例：获取 user_name 的数据的类型

```
127.0.0.1:6379> type user_name
string
127.0.0.1:6379>
```

## 5.5 返回当前数据库中 key 的数目

返回当前数据库中 key 的数目使用 dbsize 命令。

范例：获取当前数据库中 key 的数目

```
127.0.0.1:6379> keys *
1) "k1"
2) "k2"
3) "userage"
4) "user_name"
127.0.0.1:6379> dbsize
(integer) 4
127.0.0.1:6379>
```

## 5.6 获取服务器的信息和统计

获取服务器的信息和统计使用 info 命令。

范例：获取服务器信息

```
127.0.0.1:6379> info
# Server
redis_version:3.0.7
redis_git_sha1:00000000
redis_git_dirty:0
redis_build_id:9310b523e663e7ad
redis_mode:standalone
os:Linux 2.6.32-431.el6.i686 i686
arch_bits:32
multiplexing_api:epoll
gcc_version:4.4.7
process_id:1296
run_id:89f7fd18598e38523ccd3dd89b144a7013f3bef3
tcp_port:6379
uptime_in_seconds:28310
```

```
uptime_in_days:0
hz:10
lru_clock:16746523
config_file:/usr/local/6379/./redis.conf

# Clients
connected_clients:2
client_longest_output_list:0
client_biggest_input_buf:0
blocked_clients:0

# Memory
used_memory:658112
used_memory_human:642.69K
used_memory_rss:1904640
used_memory_peak:658328
used_memory_peak_human:642.90K
used_memory_lua:24576
mem_fragmentation_ratio:2.89
mem_allocator:jemalloc-3.6.0

# Persistence
loading:0
rdb_changes_since_last_save:0
rdb_bgsave_in_progress:0
rdb_last_save_time:1509918442
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:5
total_commands_processed:287
instantaneous_ops_per_sec:0
total_net_input_bytes:10452
total_net_output_bytes:13479
instantaneous_input_kbps:0.00
```

```
instantaneous_output_kbps:0.00
rejected_connections:0
sync_full:0
sync_partial_ok:0
sync_partial_err:0
expired_keys:1
evicted_keys:0
keyspace_hits:98
keyspace_misses:10
pubsub_channels:0
pubsub_patterns:0
latest_fork_usec:1469
migrate_cached_sockets:0

# 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

# CPU
used_cpu_sys:28.60
used_cpu_user:15.06
used_cpu_sys_children:0.03
used_cpu_user_children:0.01

# Cluster
cluster_enabled:0

# Keyspace
db0:keys=4,expires=0,avg_ttl=0
```

此时我们发现使用 info 获取的数据信息很丰富，如果只需要获取某一个方便的信息，可以使用 info 加上需要的信息的名称。

范例：获取数据库的 CPU 信息。

```
127.0.0.1:6379> info CPU
# CPU
used_cpu_sys:28.81
used_cpu_user:15.13
used_cpu_sys_children:0.03
used_cpu_user_children:0.01
127.0.0.1:6379>
```

### 5.7 退出连接

# 6. redis 的持久化

Redis 的高性能是由于其将所有数据都存储在了内存中，为了使 Redis 在重启之后仍能保证数据不丢失，需要将数据从内存中同步到硬盘中，这一过程就是持久化。

Redis 支持两种方式的持久化，一种是 RDB 方式，一种是 AOF 方式。可以单独使用其中一种或将二者结合使用。

## 6.1 RDB 持久化

RDB 方式的持久化是通过快照（snapshotting）完成的，当符合一定条件时 Redis 会自动将内存中的数据进行快照并持久化到硬盘。

RDB 是 Redis 默认采用的持久化方式，在 redis.conf 配置文件中默认有此下配置：

```
save 900 1

save 300 10

save 60 10000
```

save 开头的一行就是持久化配置，可以配置多个条件（每行配置一个条件），每个条件之间是“或”的关系，“save 900 1”表示 15 分钟（900 秒钟）内至少 1 个键被更改则进行快照，“save 300 10”表示 5 分钟（300 秒）内至少 10 个键被更改则进行快照。

Redis 启动后会读取 RDB 快照文件，将数据从硬盘载入到内存。根据数据量

大小与结构和服务器性能不同，这个时间也不同。通常将记录一千万个字符串类型键、大小为 1GB 的快照文件载入到内存中需要花费 20～30 秒钟。

如果开启了 RDB，Redis 默认会在启动当前服务的目录自动创建数据文件。通过 RDB 方式实现持久化，一旦 Redis 异常退出，就会丢失最后一次快照以后更改的所有数据。这就需要开发者根据具体的应用场合，通过组合设置自动快照条件的方式来将可能发生的数据损失控制在能够接受的范围。如果数据很重要以至于无法承受任何损失，则可以考虑使用 AOF 方式进行持久化。

## 6.2 AOF 持久化

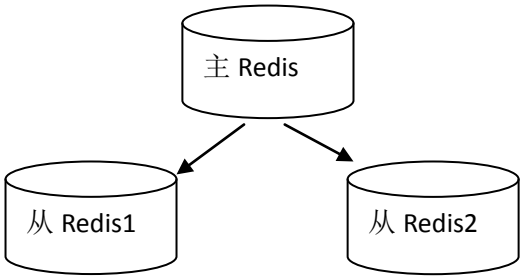
默认情况下 Redis 没有开启 AOF（append only file）方式的持久化，可以通过 appendonly 参数开启：appendonly yes

开启 AOF 持久化后每执行一条会更改 Redis 中的数据的命令，Redis 就会将该命令写入硬盘中的 AOF 文件。AOF 文件的保存位置和 RDB 文件的位置相同，都是通过 dir 参数设置的，默认的文件名是 appendonly.aof，可以通过 appendfilename 参数修改：appendfilename appendonly.aof

# 7. redis 的主从复制

## 7.1 什么是主从复制

持久化保证了即使 redis 服务重启也会丢失数据，因为 redis 服务重启后会将硬盘上持久化的数据恢复到内存中，但是当 redis 服务器的硬盘损坏了可能会导致数据丢失，如果通过 redis 的主从复制机制就可以避免这种单点故障。




说明：

- 主 redis 中的数据有两个副本（replication）即从 redis1 和从 redis2，即使一台 redis 服务器宕机其它两台 redis 服务也可以继续提供服务。
- 主 redis 中的数据和从 redis 上的数据保持实时同步，当主 redis 写入数据时通过主从复制机制会复制到两个从 redis 服务上。
- 只有一个主 redis，可以有多个从 redis。
- 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

## 7.2 主从复制实现

在 Redis 从服务器上添加其对应的主服务器的信息，修改从 redis 服务器上的 redis.conf 文件，添加 slaveof 主 redis 服务器的 ip 和端口号。

```
################################## REPLICATION #################################

# Master-Slave replication. Use slaveof to make a Redis instance a copy of
# another Redis server. A few things to understand ASAP about Redis replication.
#
# 1) Redis replication is asynchronous, but you can configure a master to
#    stop accepting writes if it appears to be not connected with at least
#    a given number of slaves.
# 2) Redis slaves are able to perform a partial resynchronization with the
#    master if the replication link is lost for a relatively small amount of
#    time. You may want to configure the replication backlog size (see the next
#    sections of this file) with a sensible value depending on your needs.
# 3) Replication is automatic and does not need user intervention. After a
#    network partition slaves automatically try to reconnect to masters
#    and resynchronize with them.
#
# slaveof <masterip> <masterport>
slaveof 192.168.5.128 6379
-- INSERT --
```

启动主从服务器之后即完成了对 Redis 主从复制的配置。