**Redis**

**学习手册**

**V1.0**

**张镇**

**目 录**

[1 简介 1](#_Toc490143608)

[1.1 定义 1](#_Toc490143609)

[1.2 支持语言 2](#_Toc490143610)

[1.3 Redis与其他数据库和软件的对比 2](#_Toc490143611)

[1.4 附加特性 4](#_Toc490143612)

[1.5 使用Redis的理由 4](#_Toc490143613)

[2 安装及配置 6](#_Toc490143614)

[2.1 在Ubuntu上安装Redis 6](#_Toc490143615)

[2.2 Redis配置 7](#_Toc490143616)

[2.3 编辑配置 11](#_Toc490143617)

[2.4 存储 12](#_Toc490143618)

[2.5 Redis常用内存优化手段与参数 12](#_Toc490143619)

[3 Redis数据结构简介 15](#_Toc490143620)

[3.1 Redis中的字符串 16](#_Toc490143621)

[3.2 Redis中的列表 17](#_Toc490143622)

[3.3 Redis的散列 18](#_Toc490143623)

[3.4 Redis的集合 20](#_Toc490143624)

[3.5 Redis的有序集合 21](#_Toc490143625)

[4 Redis命令 23](#_Toc490143626)

[4.1 常用命令 23](#_Toc490143627)

[4.2 键命令 24](#_Toc490143628)

[5 你好Redis 26](#_Toc490143629)

[5.1 对文章进行投票 27](#_Toc490143630)

[5.2 发布并获取文章 29](#_Toc490143631)

[5.3 对文章进行分组 30](#_Toc490143632)

[6 Redis进阶 33](#_Toc490143633)

[6.1 Redis发送订阅 33](#_Toc490143634)

[6.2 Redis事务 33](#_Toc490143635)

[6.3 Redis脚本 34](#_Toc490143636)

[6.4 Redis连接 35](#_Toc490143637)

[6.5 Redis服务器 35](#_Toc490143638)

[6.6 Redis备份 38](#_Toc490143639)

[6.7 Redis客户端连接 39](#_Toc490143640)

[6.8 Redis管道 40](#_Toc490143641)

[6.9 Redis分区 41](#_Toc490143642)

[7 各类语言连接Redis 43](#_Toc490143643)

[7.1 Java连接Redis 43](#_Toc490143644)

[7.2 PHP连接Redis 46](#_Toc490143645)

[7.3 C#连接Redis 49](#_Toc490143646)

# 简介

Redis是一个开源的使用ANSI [C语言](https://baike.baidu.com/item/C%E8%AF%AD%E8%A8%80)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](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93)，并提供多种语言的API，是一个开源，高级的键值存储和一个适用的解决方案，用于构建高性能，可扩展的Web应用程序。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。

Redis是一个远程内存数据库，它不仅性能强劲，而且还具有复制特性以及为解决问题而生的独一无二的数据模型。Redis提供了5种不同类型的数据结构，各式各样的问题都可以很自然地映射到这些数据结构上：Redis的数据结构致力于帮助用户解决问题，而不会像其他数据库那样，要求用户扭曲问题来适应数据库。除此之外，通过复制、持久化（persistence）和客户端分片（client-side sharding）等特性，用户可以很方便地将Redis扩展成一个能够包含数百GB数据、每秒处理上百万次请求的系统。

Redis Redis有三个主要特点，使它优越于其它键值数据存储系统：

* Redis将其数据库完全保存在内存中，仅使用磁盘进行持久化。
* 与其它键值数据存储相比，Redis有一组相对丰富的数据类型。
* Redis可以将数据复制到任意数量的从机中。

Redis官方网网站是：<http://www.redis.io/>

## 定义

Redis是一个速度非常快的非关系数据库（non-relational database），它可以存储键（key）与5种不同类型的值（value）之间的映射（mapping），包括string(字符串)、list([链表](https://baike.baidu.com/item/%E9%93%BE%E8%A1%A8))、set(集合)、zset(sorted set --有序集合)和hash（哈希类型），这些[数据类型](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E7%B1%BB%E5%9E%8B)都支持push/pop、add/remove及取交集并集和差集及更丰富的操作，而且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。与memcached一样，为了保证效率，数据都是缓存在内存中。区别的是redis会周期性的把更新的数据写入磁盘或者把修改操作写入追加的记录文件，并且在此基础上实现了master-slave(主从)同步。可以使用复制特性来扩展读性能，还可以使用客户端分片1来扩展写性能。

Redis 是一个高性能的key-value数据库。 Redis的出现，很大程度补偿了[memcached](https://baike.baidu.com/item/memcached)这类key/value存储的不足，在部分场合可以对关系数据库起到很好的补充作用。它提供了Java，C/C++，C#，PHP，JavaScript，Perl，Object-C，Python，Ruby，Erlang等客户端，使用很方便。

Redis支持主从同步。数据可以从主服务器向任意数量的从服务器上同步，从服务器可以是关联其他从服务器的主服务器。这使得Redis可执行单层树复制。存盘可以有意无意的对数据进行写操作。由于完全实现了发布/订阅机制，使得从数据库在任何地方同步树时，可订阅一个频道并接收主服务器完整的消息发布记录。同步对读取操作的可扩展性和数据冗余很有帮助。

Redis的官网地址是redis.io。（域名后缀io属于国家域名，是british Indian Ocean territory，即英属印度洋领地）目前，Vmware在资助着redis项目的开发和维护。

## 支持语言

许多语言都包含Redis支持，包括：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| * [ActionScript](https://baike.baidu.com/item/ActionScript) * [C](https://baike.baidu.com/item/C) * [C++](https://baike.baidu.com/item/C%2B%2B) * [C#](https://baike.baidu.com/item/C%23) * [Clojure](https://baike.baidu.com/item/Clojure) | * [Common Lisp](https://baike.baidu.com/item/Common%20Lisp) * [Dart](https://baike.baidu.com/item/Dart) * [Erlang](https://baike.baidu.com/item/Erlang) * [Go](https://baike.baidu.com/item/Go) * [Haskell](https://baike.baidu.com/item/Haskell) | * [Haxe](https://baike.baidu.com/item/Haxe) * [Io](https://baike.baidu.com/item/Io) * [Java](https://baike.baidu.com/item/Java) * [Node.js](https://baike.baidu.com/item/Node.js) * [Lua](https://baike.baidu.com/item/Lua) | * [Objective-C](https://baike.baidu.com/item/Objective-C) * [Perl](https://baike.baidu.com/item/Perl) * [PHP](https://baike.baidu.com/item/PHP) * Pure Data * [Python](https://baike.baidu.com/item/Python) | * [R](https://baike.baidu.com/item/R) * [Ruby](https://baike.baidu.com/item/Ruby) * [Scala](https://baike.baidu.com/item/Scala) * [Smalltalk](https://baike.baidu.com/item/Smalltalk) * [Tcl](https://baike.baidu.com/item/Tcl) |

## Redis与其他数据库和软件的对比

如果你熟悉关系数据库，那么你肯定写过用来关联两个表的数据的SQL查询。而Redis则属于人们常说的NoSQL数据库或者非关系数据库：Redis不使用表，它的数据库也不会预定义或者强制去要求用户对Redis存储的不同数据进行关联。

高性能键值缓存服务器memcached也经常被拿来与Redis进行比较：这两者都可用于存储键值映射，彼此的性能也相差无几，但是Redis能够自动以两种不同的方式将数据写入硬盘，并且Redis除了能存储普通的字符串键之外，还可以存储其他4种数据结构，而memcached只能存储普通的字符串键。这些不同之处使得Redis可以用于解决更为广泛的问题，并且既可以用作主数据库（primary database）使用，又可以作为其他存储系统的辅助数据库（auxiliary database）使用。

一般来说，许多用户只会在Redis的性能或者功能是必要的情况下，才会将数据存储到Redis里面：如果程序对性能的要求不高，又或者因为费用原因而没办法将大量数据存储到内存里面，那么用户可能会选择使用关系数据库，或者其他非关系数据库。在实际中，读者应该根据自己的需求来决定是否使用Redis，并考虑是将Redis用作主存储还是辅助存储，以及如何通过复制、持久化和事务等手段保证数据的完整性。

表1-1展示了一部分在功能上与Redis有重叠的数据库服务器和缓存服务器，从这个表可以看出Redis与这些数据库及软件之间的区别。

**表1-1　一些数据库和缓存服务器的特性与功能**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **名称** | **类型** | **数据存储选项** | **查询类型** | **附加功能** |
| Redis | 使用内存存储（in-memory）的非关系数据库 | 字符串、列表、集合、散列表、有序集合 | 每种数据类型都有自己的专属命令，另外还有批量操作（bulk operation）和不完全（partial）的事务支持 | 发布与订阅，主从复制（master/slave replication），持久化，脚本（存储过程，stored procedure） |
| memcached | 使用内存存储的键值缓存 | 键值之间的映射 | 创建命令、读取命令、更新命令、删除命令以及其他几个命令 | 为提升性能而设的多线程服务器 |
| MySQL | 关系数据库 | 每个数据库可以包含多个表，每个表可以包含多个行；可以处理多个表的视图（view）；支持空间（spatial）和第三方扩展 | SELECT、 INSERT、 UPDATE、 DELETE、函数、存储过程 | 支持ACID性质（需要使用InnoDB），主从复制和主主复制 （master/master replication） |
| PostgreSQL | 关系数据库 | 每个数据库可以包含多个表，每个表可以包含多个行；可以处理多个表的视图；支持空间和第三方扩展；支持可定制类型 | SELECT、 INSERT、 UPDATE、 DELETE、内置函数、自定义的存储过程 | 支持ACID性质，主从复制，由第三方支持的多主复制（multi-master replication） |
| MongoDB | 使用硬盘存储（on-disk）的非关系文档存储 | 每个数据库可以包含多个表，每个表可以包含多个无schema（schema-less）的BSON文档 | 创建命令、读取命令、更新命令、删除命令、条件查询命令等 | 支持map-reduce操作，主从复制，分片，空间索引（spatial index） |

* Redis是键值数据库系统的不同进化路线，它的值可以包含更复杂的数据类型，可在这些数据类型上定义原子操作。
* Redis是一个内存数据库，但在磁盘数据库上是持久的，因此它代表了一个不同的权衡，在这种情况下，在不能大于存储器(内存)的数据集的限制下实现非常高的写和读速度。
* 内存数据库的另一个优点是，它与磁盘上的相同数据结构相比，复杂数据结构在内存中存储表示更容易操作。 因此，Redis可以做很少的内部复杂性。

## 附加特性

在使用类似Redis这样的内存数据库时，一个首先要考虑的问题就是“当服务器被关闭时，服务器存储的数据将何去何从呢？”

Redis拥有两种不同形式的持久化方法，它们都可以用小而紧凑的格式将存储在内存中的数据写入硬盘：

* 第一种持久化方法为时间点转储（point-in-time dump），转储操作既可以在“指定时间段内有指定数量的写操作执行”这一条件被满足时执行，又可以通过调用两条转储到硬盘（dump-to-disk）命令中的任何一条来执行；
* 第二种持久化方法将所有修改了数据库的命令都写入一个只追加（append-only）文件里面，用户可以根据数据的重要程度，将只追加写入设置为从不同步（sync）、每秒同步一次或者每写入一个命令就同步一次。

另外，尽管Redis的性能很好，但受限于Redis的内存存储设计，有时候只使用一台Redis服务器可能没有办法处理所有请求。因此，为了扩展Redis的读性能，并为Redis提供故障转移（failover）支持，Redis实现了主从复制特性：执行复制的从服务器会连接上主服务器，接收主服务器发送的整个数据库的初始副本（copy）；之后主服务器执行的写命令，都会被发送给所有连接着的从服务器去执行，从而实时地更新从服务器的数据集。因为从服务器包含的数据会不断地进行更新，所以客户端可以向任意一个从服务器发送读请求，以此来避免对主服务器进行集中式的访问。

## 使用Redis的理由

以下是**Redis**的一些优点。

* **异常快** - Redis非常快，每秒可执行大约110000次的设置(SET)操作，每秒大约可执行81000次的读取/获取(GET)操作。
* **支持丰富的数据类型** - Redis支持开发人员常用的大多数数据类型，例如列表，集合，排序集和散列等等。这使得Redis很容易被用来解决各种问题，因为我们知道哪些问题可以更好使用地哪些数据类型来处理解决。
* **操作具有原子性** - 所有Redis操作都是原子操作，这确保如果两个客户端并发访问，Redis服务器能接收更新的值。
* **多实用工具** - Redis是一个多实用工具，可用于多种用例，如：缓存，消息队列(Redis本地支持发布/订阅)，应用程序中的任何短期数据，例如，web应用程序中的会话，网页命中计数等。

有memcached使用经验的读者可能知道，用户只能用APPEND命令将数据添加到已有字符串的末尾。memcached的文档中声明，可以用APPEND命令来管理元素列表。这很好！用户可以将元素追加到一个字符串的末尾，并将那个字符串当作列表来使用。但随后如何删除这些元素呢？memcached采用的办法是通过黑名单（blacklist）来隐藏列表里面的元素，从而避免对元素执行读取、更新、写入（包括在一次数据库查询之后执行的memcached写入）等操作。相反地，Redis的LIST和SET允许用户直接添加或者删除元素。

使用Redis而不是memcached来解决问题，不仅可以让代码变得更简短、更易懂、更易维护，而且还可以使代码的运行速度更快（因为用户不需要通过读取数据库来更新数据）。除此之外，在其他许多情况下，Redis的效率和易用性也比关系数据库要好得多。

数据库的一个常见用法是存储长期的报告数据，并将这些报告数据用作固定时间范围内的聚合数据（aggregates）。收集聚合数据的常见做法是：先将各个行插入一个报告表里面，之后再通过扫描这些行来收集聚合数据，并根据收集到的聚合数据来更新聚合表中已有的那些行。之所以使用插入行的方式来存储，是因为对于大部分数据库来说，插入行操作的执行速度非常快（插入行只会在硬盘文件末尾进行写入）。不过，对表里面的行进行更新却是一个速度相当慢的操作，因为这种更新除了会引起一次随机读（random read）之外，还可能会引起一次随机写（random write）。而在Redis里面，用户可以直接使用原子的（atomic）INCR命令及其变种来计算聚合数据，并且因为Redis将数据存储在内存里面，而且发送给Redis的命令请求并不需要经过典型的查询分析器（parser）或者查询优化器（optimizer）进行处理，所以对Redis存储的数据执行随机写的速度总是非常迅速的。

使用 Redis 而不是关系数据库或者其他硬盘存储数据库，可以避免写入不必要的临时数据，也免去了对临时数据进行扫描或者删除的麻烦，并最终改善程序的性能。虽然上面列举的都是一些简单的例子，但它们很好地证明了“工具会极大地改变人们解决问题的方式”这一点。

# 安装及配置

## 在Ubuntu上安装Redis

要在Ubuntu上安装Redis，打开终端并键入以下命令 -

[yiibai@ubuntu:~]$ sudo apt-get update

[yiibai@ubuntu:~]$ sudo apt-get install redis-server

这将在Ubuntu机器上安装Redis。

**启动Redis**

[yiibai@ubuntu:~]$ redis-server

[2988] 07 Feb 17:09:42.485 # Warning: no config file specified, using the default config. In order to specify a config file use redis-server /path/to/redis.conf

[2988] 07 Feb 17:09:42.488 # Unable to set the max number of files limit to 10032 (Operation not permitted), setting the max clients configuration to 3984.

[2988] 07 Feb 17:09:42.490 # Warning: 32 bit instance detected but no memory lim

\_.\_

\_.-``\_\_ ''-.\_

\_.-`` `. `\_. ''-.\_ Redis 2.8.4 (00000000/0) 32 bit

.-`` .-```. ```\/ \_.,\_ ''-.\_

( ' , .-` | `, ) Running in stand alone mode

|`-.\_`-...-` \_\_...-.``-.\_|'` \_.-'| Port: 6379

| `-.\_ `.\_ / \_.-' | PID: 2988

`-.\_ `-.\_ `-./ \_.-' \_.-'

|`-.\_`-.\_ `-.\_\_.-' \_.-'\_.-'|

| `-.\_`-.\_ \_.-'\_.-' | http://redis.io

`-.\_ `-.\_`-.\_\_.-'\_.-' \_.-'

|`-.\_`-.\_ `-.\_\_.-' \_.-'\_.-'|

| `-.\_`-.\_ \_.-'\_.-' |

`-.\_ `-.\_`-.\_\_.-'\_.-' \_.-'

`-.\_ `-.\_\_.-' \_.-'

`-.\_ \_.-'

`-.\_\_.-'

[2988] 07 Feb 17:09:42.581 # Server started, Redis version 2.8.4

[2988] 07 Feb 17:09:42.582 # WARNING overcommit\_memory is set to 0! Background s ' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.overcommit\_m

[2988] 07 Feb 17:09:42.582 \* The server is now ready to accept connections on po

**检查Redis是否正在工作**

[yiibai@ubuntu:~]$ redis-cli

这将打开一个**redis**提示，如下所示 -

redis 127.0.0.1:6379>

在上面的提示中，127.0.0.1是计算机的IP地址，6379是运行**Redis**服务器的端口。 现在键入以下PING命令。

redis 127.0.0.1:6379> ping

PONG

这表明**Redis**已成功在您的计算机上安装了。

在Ubuntu上安装**Redis桌面管理**

要在Ubuntu上安装**Redis**桌面管理器，可从 <http://redisdesktop.com/download> 下载该软件包，安装即可。

打开下载的软件包并安装。

**Redis桌面管理器**将提供用于管理Redis的键和数据的UI。

## Redis配置

在Redis中，在Redis的根目录下有一个配置文件(redis.conf)。当然您可以通过Redis CONFIG命令获取和设置所有的**Redis**配置。

**语法**

以下是Redis中的CONFIG命令的基本语法。

redis 127.0.0.1:6379> CONFIG GET CONFIG\_SETTING\_NAME

**示例**

redis 127.0.0.1:6379> CONFIG GET loglevel

1) "loglevel"

2) "notice"

要获取所有配置设置，请使用\*代替CONFIG\_SETTING\_NAME

**示例**

redis 127.0.0.1:6379> CONFIG GET \*

1) "dbfilename"

2) "dump.rdb"

3) "requirepass"

4) ""

5) "masterauth"

6) ""

7) "unixsocket"

8) ""

9) "logfile"

10) "/var/log/redis/redis-server.log"

11) "pidfile"

12) "/var/run/redis/redis-server.pid"

13) "maxmemory"

14) "3221225472"

15) "maxmemory-samples"

16) "3"

17) "timeout"

18) "0"

19) "tcp-keepalive"

20) "0"

21) "auto-aof-rewrite-percentage"

22) "100"

23) "auto-aof-rewrite-min-size"

24) "67108864"

25) "hash-max-ziplist-entries"

26) "512"

27) "hash-max-ziplist-value"

28) "64"

29) "list-max-ziplist-entries"

30) "512"

31) "list-max-ziplist-value"

32) "64"

33) "set-max-intset-entries"

34) "512"

35) "zset-max-ziplist-entries"

36) "128"

37) "zset-max-ziplist-value"

38) "64"

39) "lua-time-limit"

40) "5000"

41) "slowlog-log-slower-than"

42) "10000"

43) "slowlog-max-len"

44) "128"

45) "port"

46) "6379"

47) "databases"

48) "16"

49) "repl-ping-slave-period"

50) "10"

51) "repl-timeout"

52) "60"

53) "repl-backlog-size"

54) "1048576"

55) "repl-backlog-ttl"

56) "3600"

57) "maxclients"

58) "3984"

59) "watchdog-period"

60) "0"

61) "slave-priority"

62) "100"

63) "min-slaves-to-write"

64) "0"

65) "min-slaves-max-lag"

66) "10"

67) "hz"

68) "10"

69) "no-appendfsync-on-rewrite"

70) "no"

71) "slave-serve-stale-data"

72) "yes"

73) "slave-read-only"

74) "yes"

75) "stop-writes-on-bgsave-error"

76) "yes"

77) "daemonize"

78) "yes"

79) "rdbcompression"

80) "yes"

81) "rdbchecksum"

82) "yes"

83) "activerehashing"

84) "yes"

85) "repl-disable-tcp-nodelay"

86) "no"

87) "aof-rewrite-incremental-fsync"

88) "yes"

89) "appendonly"

90) "no"

91) "dir"

92) "/var/lib/redis"

93) "maxmemory-policy"

94) "noeviction"

95) "appendfsync"

96) "everysec"

97) "save"

98) "900 1 300 10 60 10000"

99) "loglevel"

100) "notice"

101) "client-output-buffer-limit"

102) "normal 0 0 0 slave 268435456 67108864 60 pubsub 33554432 8388608 60"

103) "unixsocketperm"

104) "0"

105) "slaveof"

106) ""

107) "notify-keyspace-events"

108) ""

109) "bind"

110) "127.0.0.1"

## 编辑配置

要更新配置，可以直接编辑redis.conf文件，也可以通过CONFIG set命令更新配置。

**语法**

以下是CONFIG SET命令的基本语法。

redis 127.0.0.1:6379> CONFIG SET CONFIG\_SETTING\_NAME NEW\_CONFIG\_VALUE

**示例**

redis 127.0.0.1:6379> CONFIG SET loglevel "notice"

OK

redis 127.0.0.1:6379> CONFIG GET loglevel

1) "loglevel"

2) "notice"

## 存储

redis使用了两种[文件格式](https://baike.baidu.com/item/%E6%96%87%E4%BB%B6%E6%A0%BC%E5%BC%8F)：全量数据和增量请求。

* 全量数据格式是把内存中的数据写入磁盘，便于下次读取文件进行加载；
* 增量请求文件则是把内存中的数据序列化为操作请求，用于读取文件进行replay得到数据，序列化的操作包括SET、RPUSH、SADD、ZADD。

redis的存储分为内存存储、[磁盘存储](https://baike.baidu.com/item/%E7%A3%81%E7%9B%98%E5%AD%98%E5%82%A8)和log文件三部分，配置文件中有三个参数对其进行配置。

* save seconds updates，save配置，指出在多长时间内，有多少次更新操作，就将[数据同步](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%90%8C%E6%AD%A5)到数据文件。这个可以多个条件配合，比如默认配置文件中的设置，就设置了三个条件。
* appendonly yes/no ，appendonly配置，指出是否在每次更新操作后进行日志记录，如果不开启，可能会在断电时导致一段时间内的数据丢失。因为redis本身同步数据文件是按上面的save条件来同步的，所以有的数据会在一段时间内只存在于内存中。
* appendfsync no/always/everysec ，appendfsync配置，no表示等[操作系统](https://baike.baidu.com/item/%E6%93%8D%E4%BD%9C%E7%B3%BB%E7%BB%9F)进行[数据缓存](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E7%BC%93%E5%AD%98)同步到磁盘，always表示每次更新操作后手动调用fsync()将数据写到磁盘，everysec表示每秒同步一次。

## Redis常用内存优化手段与参数

通过我们上面的一些实现上的分析可以看出redis实际上的内存管理成本非常高，即占用了过多的内存，作者对这点也非常清楚，所以提供了一系列的参数和手段来控制和节省内存，我们分别来讨论下。

首先最重要的一点是不要开启Redis的VM选项，即虚拟内存功能，这个本来是作为Redis存储超出物理内存数据的一种数据在内存与磁盘换入换出的一个持久化策略，但是其内存管理成本也非常的高，并且我们后续会分析此种持久化策略并不成熟，所以要关闭VM功能，请检查你的redis.conf文件中 vm-enabled 为 no。

其次最好设置下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。

另外Redis为不同数据类型分别提供了一组参数来控制内存使用，我们在前面详细分析过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：

1. hash-max-zipmap-entries 64
2. hash-max-zipmap-value 512
3. hash-max-zipmap-entries

含义是当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。

**同样类似的参数**

* list-max-ziplist-entries 512

说明：list数据类型多少节点以下会采用去指针的紧凑存储格式。

* list-max-ziplist-value 64

说明：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。

* set-max-intset-entries 512

说明：set数据类型内部数据如果全部是数值型，且包含多少节点以下会采用紧凑格式存储。

最后想说的是Redis内部实现没有对内存分配方面做过多的优化，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。

另外redis 的6种过期策略redis 中的默认的过期策略是volatile-lru 。设置方式

　　config set maxmemory-policy volatile-lru

maxmemory-policy 六种方式

* volatile-lru：只对设置了过期时间的key进行LRU（默认值）
* allkeys-lru ： 是从所有key里 删除 不经常使用的key
* volatile-random：随机删除即将过期key
* allkeys-random：随机删除
* volatile-ttl ： 删除即将过期的
* noeviction ： 永不过期，返回错误
* maxmemory-samples 3 是说每次进行淘汰的时候 会随机抽取3个key 从里面淘汰最不经常使用的（默认选项）

# Redis数据结构简介

正如之前的表1-1所示，Redis可以存储键与5种不同数据结构类型之间的映射，这5种数据结构类型分别为STRING（字符串）、LIST（列表）、SET（集合）、HASH（散列）和ZSET（有序集合）。有一部分Redis命令对于这5种结构都是通用的，如DEL、TYPE、RENAME等；但也有一部分Redis命令只能对特定的一种或者两种结构使用。

大部分程序员应该都不会对Redis的STRING、LIST、HASH这3种结构感到陌生，因为它们和很多编程语言内建的字符串、列表和散列等结构在实现和语义（semantics）方面都非常相似。有些编程语言还有集合数据结构，在实现和语义上类似于Redis的SET。ZSET在某种程度上是一种Redis特有的结构，但是当你熟悉了它之后，就会发现它也是一种非常有用的结构。表1-2对比了Redis提供的5种结构，说明了这些结构存储的值，并简单介绍了它们的语义。

**表1-2　Redis提供的5种结构**

|  |  |  |
| --- | --- | --- |
| **结构类型** | **结构存储的值** | **结构的读写能力** |
| STRING | 可以是字符串、整数或者浮点数 | 对整个字符串或者字符串的其中一部分执行操作；对整数和浮点数执行自增（increment）或者自减（decrement）操作 |
| LIST | 一个链表，链表上的每个节点都包含了一个字符串 | 从链表的两端推入或者弹出元素；根据偏移量对链表进行修剪（trim）；读取单个或者多个元素；根据值查找或者移除元素 |
| SET | 包含字符串的无序收集器（unordered collection），并且被包含的每个字符串都是独一无二、各不相同的 | 添加、获取、移除单个元素；检查一个元素是否存在于集合中；计算交集、并集、差集；从集合里面随机获取元素 |
| HASH | 包含键值对的无序散列表 | 添加、获取、移除单个键值对；获取所有键值对 |
| ZSET（有序集合） | 字符串成员（member）与浮点数分值（score）之间的有序映射，元素的排列顺序由分值的大小决定 | 添加、获取、删除单个元素；根据分值范围（range）或者成员来获取元素 |

## Redis中的字符串

Redis中的字符串是一个字节序列。Redis中的字符串是二进制安全的，这意味着它们的长度不由任何特殊的终止字符决定。因此，可以在一个字符串中存储高达512兆字节的任何内容。Redis的STRING和其他编程语言或者其他键值存储提供的字符串非常相似。STRING拥有一些和其他键值存储相似的命令，比如GET（获取值）、SET（设置值）和DEL（删除值）。

**表1-3　字符串命令**

| **编号** | **命令** | **描述说明** |
| --- | --- | --- |
| 1 | [SET key value](http://www.yiibai.com/redis/strings_set.html) | 此命令设置指定键的值。 |
| 2 | [GET key](http://www.yiibai.com/redis/strings_get.html) | 获取指定键的值。 |
| 3 | [GETRANGE key start end](http://www.yiibai.com/redis/strings_getrange.html) | 获取存储在键上的字符串的子字符串。 |
| 4 | [GETSET key value](http://www.yiibai.com/redis/strings_getset.html) | 设置键的字符串值并返回其旧值。 |
| 5 | [GETBIT key offset](http://www.yiibai.com/redis/strings_getbit.html) | 返回在键处存储的字符串值中偏移处的位值。 |
| 6 | [MGET key1 [key2..]](http://www.yiibai.com/redis/strings_mget.html) | 获取所有给定键的值 |
| 7 | [SETBIT key offset value](http://www.yiibai.com/redis/strings_setbit.html) | 存储在键上的字符串值中设置或清除偏移处的位 |
| 8 | [SETEX key seconds value](http://www.yiibai.com/redis/strings_setex.html) | 使用键和到期时间来设置值 |
| 9 | [SETNX key value](http://www.yiibai.com/redis/strings_setnx.html) | 设置键的值，仅当键不存在时 |
| 10 | [SETRANGE key offset value](http://www.yiibai.com/redis/strings_setrange.html) | 在指定偏移处开始的键处覆盖字符串的一部分 |
| 11 | [STRLEN key](http://www.yiibai.com/redis/strings_strlen.html) | 获取存储在键中的值的长度 |
| 12 | [MSET key value [key value …]](http://www.yiibai.com/redis/strings_mset.html) | 为多个键分别设置它们的值 |
| 13 | [MSETNX key value [key value …]](http://www.yiibai.com/redis/strings_msetnx.html) | 为多个键分别设置它们的值，仅当键不存在时 |
| 14 | [PSETEX key milliseconds value](http://www.yiibai.com/redis/strings_psetex.html) | 设置键的值和到期时间(以毫秒为单位) |
| 15 | [INCR key](http://www.yiibai.com/redis/strings_incr.html) | 将键的整数值增加1 |
| 16 | [INCRBY key increment](http://www.yiibai.com/redis/strings_incrby.html) | 将键的整数值按给定的数值增加 |
| 17 | [INCRBYFLOAT key increment](http://www.yiibai.com/redis/strings_incrbyfloat.html) | 将键的浮点值按给定的数值增加 |
| 18 | [DECR key](http://www.yiibai.com/redis/strings_decr.html) | 将键的整数值减1 |
| 19 | [DECRBY key decrement](http://www.yiibai.com/redis/strings_decrby.html) | 按给定数值减少键的整数值 |
| 20 | [APPEND key value](http://www.yiibai.com/redis/strings_append.html) | 将指定值附加到键 |

除了能够GET、SET和DEL字符串值之外，Redis还提供了一些可以对字符串的其中一部分内容进行读取和写入的命令，以及一些能对字符串存储的数值执行自增或者自减操作的命令。

**示例**

redis 127.0.0.1:6379> set name "yiibai.com"

OK

redis 127.0.0.1:6379> get name

"yiibai.com"

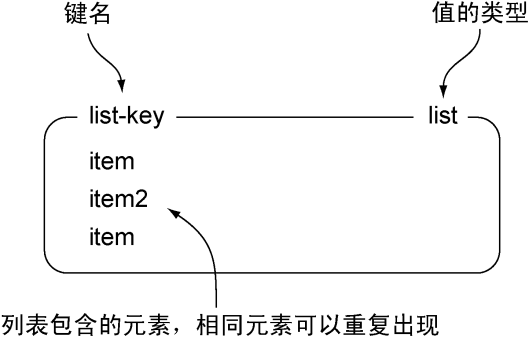
在上面的示例中，set和get是Redis命令，name是Redis中使用的键，yiibai.com是存储在Redis中的字符串的值。

注 - Redis命令不区分大小写，如SET,Set和set都是同一个命令。字符串值的最大长度为 512MB。

## Redis中的列表

Redis列表只是字符串列表，按插入顺序排序。您可以向Redis列表的头部或尾部添加元素。

Redis对链表（linked-list）结构的支持使得它在键值存储的世界中独树一帜。一个列表结构可以有序地存储多个字符串，和表示字符串时使用的方法一样。



**图1-2　list-key是一个包含3个元素的列表键，注意列表里面的元素是可以重复的**

Redis列表可执行的操作和很多编程语言里面的列表操作非常相似：LPUSH命令和RPUSH命令分别用于将元素推入列表的左端（left end）和右端（right end）；LPOP命令和RPOP命令分别用于从列表的左端和右端弹出元素；LINDEX命令用于获取列表在给定位置上的一个元素；LRANGE命令用于获取列表在给定范围上的所有元素。代码清单1-2展示了一些列表命令的使用示例，表1-4简单介绍了示例中用到的各个命令。

**表1-4　列表命令**

|  |  |
| --- | --- |
| **命令** | **行为** |
| RPUSH | 将给定值推入列表的右端 |
| LRANGE | 获取列表在给定范围上的所有值 |
| LINDEX | 获取列表在给定位置上的单个元素 |
| LPOP | 从列表的左端弹出一个值，并返回被弹出的值 |

即使Redis的列表只支持以上提到的几个命令，它也已经可以用来解决很多问题了，但Redis并没有就此止步——除了上面提到的命令之外，Redis列表还拥有从列表里面移除元素的命令、将元素插入列表中间的命令、将列表修剪至指定长度（相当于从列表的其中一端或者两端移除元素）的命令，以及其他一些命令。

**示例**

redis 127.0.0.1:6379> lpush alist redis

(integer) 1

redis 127.0.0.1:6379> lpush alist mongodb

(integer) 2

redis 127.0.0.1:6379> lpush alist sqlite

(integer) 3

redis 127.0.0.1:6379> lrange alist 0 10

1) "sqlite"

2) "mongodb"

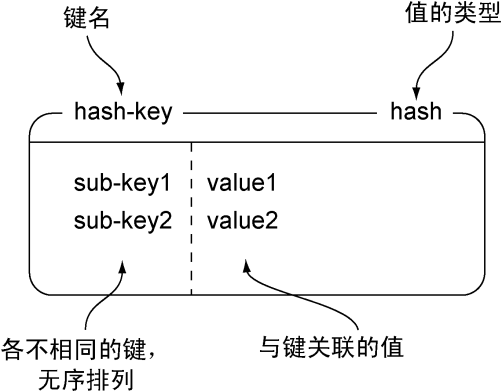
3) "redis"

列表的最大长度为2^32 - 1个元素(4294967295，每个列表可容纳超过40亿个元素)。

## Redis的散列

Redis散列/哈希(Hashes)是键值对的集合，Redis散列/哈希是字符串字段和字符串值之间的映射，因此，它们用于表示对象。每个散列/哈希可以存储多达2^32 - 1个健-值对(超过40亿个)。

和字符串一样，散列存储的值既可以是字符串又可以是数字值，并且用户同样可以对散列存储的数字值执行自增操作或者自减操作。图1-4展示了一个包含两个键值对的散列。



**图1-4　hash-key是一个包含两个键值对的散列键**

散列在很多方面就像是一个微缩版的Redis，不少字符串命令都有相应的散列版本。代码清单1-4展示了怎样对散列执行插入元素、获取元素和移除元素等操作，表1-6简单介绍了代码清单里面用到的各个命令。

**表1-6　散列命令**

|  |  |
| --- | --- |
| **命令** | **行为** |
| HSET | 在散列里面关联起给定的键值对 |
| HGET | 获取指定散列键的值 |
| HGETALL | 获取散列包含的所有键值对 |
| HDEL | 如果给定键存在于散列里面，那么移除这个键 |

熟悉文档数据库的读者可以将Redis的散列看作是文档数据库里面的文档，而熟悉关系数据库的读者则可以将Redis的散列看作是关系数据库里面的行，因为散列、文档和行这三者都允许用户同时访问或者修改一个或多个域（field）。

**示例**

redis 127.0.0.1:6379> HMSET myhash name "redis tutorial"

description "redis basic commands for caching" likes 20 visitors 23000

OK

127.0.0.1:6379> HGETALL myhash

1) "field1"

2) "Hello"

3) "field2"

4) "World"

5) "name"

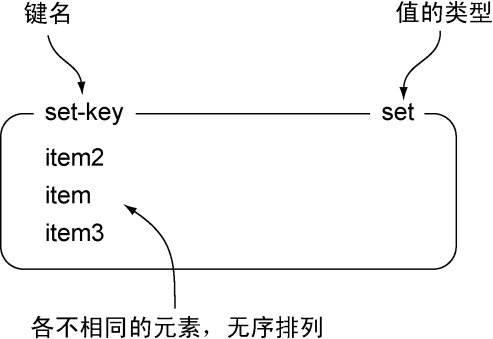
6) "redis tutorial"

在上面的例子中，在名称为’myhash‘的哈希中设置了Redis教程的详细信息(名称，描述，喜欢，访问者)。

## Redis的集合

Redis集合是字符串的无序集合。在Redis中，您可以添加，删除和测试成员存在的时间O(1)复杂性。

Redis 的集合和列表都可以存储多个字符串，它们之间的不同在于，列表可以存储多个相同的字符串，而集合则通过使用散列表来保证自己存储的每个字符串都是各不相同的（这些散列表只有键，但没有与键相关联的值）。



**图1-3　set-key是一个包含3个元素的集合键**

因为Redis的集合使用无序（unordered）方式存储元素，所以用户不能像使用列表那样，将元素推入集合的某一端，或者从集合的某一端弹出元素。不过用户可以使用SADD命令将元素添加到集合，或者使用SRAM命令从集合里面移除元素。另外还可以通过SISMEMBER命令快速地检查一个元素是否已经存在于集合中，或者使用SMEMBERS命令获取集合包含的所有元素（如果集合包含的元素非常多，那么SMEMBERS命令的执行速度可能会很慢，所以请谨慎地使用这个命令）。代码清单1-3展示了一些集合命令的使用示例，表1-5简单介绍了代码清单里面用到的各个命令。

**表1-5　集合命令**

|  |  |
| --- | --- |
| **命令** | **行为** |
| SADD | 将给定元素添加到集合 |
| SMEMBERS | 返回集合包含的所有元素 |
| SISMEMBER | 检查给定元素是否存在于集合中 |
| SREM | 如果给定的元素存在于集合中，那么移除这个元素 |

跟字符串和列表一样，集合除了基本的添加操作和移除操作之外，还支持很多其他操作，比如SINTER、SUNION、SDIFF``这3个命令就可以分别执行常见的交集计算、并集计算和差集计算。

**示例**

redis 127.0.0.1:6379> sadd yiibailist redis

(integer) 1

redis 127.0.0.1:6379> sadd yiibailist mongodb

(integer) 1

redis 127.0.0.1:6379> sadd yiibailist sqlite

(integer) 1

redis 127.0.0.1:6379> sadd yiibailist sqlite

(integer) 0

redis 127.0.0.1:6379> smembers yiibailist

1) "sqlite"

2) "mongodb"

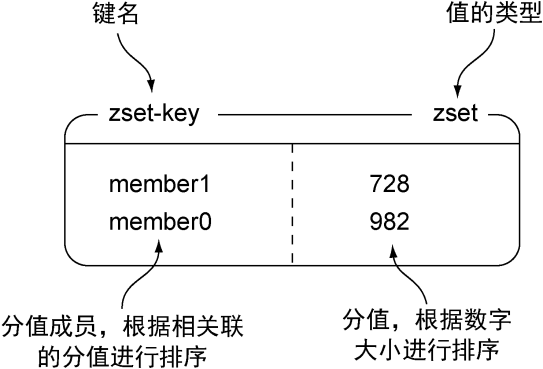
3) "redis"

注意 - 在上面的示例中，sqlite被添加了两次，但是由于集合的唯一属性，所以它只算添加一次。

一个集合中的最大成员数量为2^32 - 1(即4294967295，每个集合中元素数量可达40亿个)个。

## Redis的有序集合

有序集合和散列一样，都用于存储键值对：有序集合的键被称为成员（member），每个成员都是独一无二的，是不重复的字符集合；而有序集合的值则被称为分值（score），分值必须为浮点数。不同之处在于，排序集合的每个成员都与分数相关联，这个分数用于按最小分数到最大分数来排序的排序集合。虽然成员是唯一的，但分数值可以重复。有序集合是Redis里面唯一一个既可以根据成员访问元素（这一点和散列一样），又可以根据分值以及分值的排列顺序来访问元素的结构。图1-5展示了一个包含两个元素的有序集合示例。



**图1-5　zset-key是一个包含两个元素的有序集合键**

和Redis的其他结构一样，用户可以对有序集合执行添加、移除和获取等操作，代码清单1-5展示了这些操作的执行示例，表1-7简单介绍了代码清单里面用到的各个命令。

**表1-7　有序集合命令**

|  |  |
| --- | --- |
| **命令** | **行为** |
| ZADD | 将一个带有给定分值的成员添加到有序集合里面 |
| ZRANGE | 根据元素在有序排列中所处的位置，从有序集合里面获取多个元素 |
| ZRANGEBYSCORE | 获取有序集合在给定分值范围内的所有元素 |
| ZREM | 如果给定成员存在于有序集合，那么移除这个成员 |

现在读者应该已经知道有序集合是什么和它能干什么了。

**示例**

redis 127.0.0.1:6379> zadd yiibaiset 0 redis

(integer) 1

redis 127.0.0.1:6379> zadd yiibaiset 0 mongodb

(integer) 1

redis 127.0.0.1:6379> zadd yiibaiset 1 sqlite

(integer) 1

redis 127.0.0.1:6379> zadd yiibaiset 1 sqlite

(integer) 0

redis 127.0.0.1:6379> ZRANGEBYSCORE yiibaiset 0 1000

1) "mongodb"

2) "redis"

3) "sqlite"

因为 ‘sqlite‘ 的排序值是 1 ，其它两个元素的排序值是 0 ，所以 ‘sqlite‘ 排在最后一个位置上。

# Redis命令

## 常用命令

Redis命令是用于在Redis服务器上执行一些操作。

要在Redis服务器上运行命令，需要一个Redis客户端。Redis客户端在Redis包中有提供，这个包在我们前面的安装教程中就有安装过了。

**语法**

以下是Redis客户端的基本语法。

[yiibai@ubuntu:~]$ redis-cli

**示例**

以下示例说明了如何启动Redis客户端。

要启动Redis客户端，请打开终端并键入命令redis-cli。 这将连接到您的本地Redis服务器，现在可以运行任何的Redis命令了。

[yiibai@ubuntu:~]$redis-cli

redis 127.0.0.1:6379>

redis 127.0.0.1:6379> PING

PONG

在上面的示例中，连接到到在本地机器上运行的Redis服务器并执行PING命令，该命令检查服务器是否正在运行。

要在Redis远程服务器上运行命令，需要通过客户端redis-cli连接到服务器

**语法**

[yiibai@ubuntu:~]$ redis-cli -h host -p port -a password

**示例**

以下示例显示如何连接到Redis远程服务器，在主机(host)127.0.0.1，端口(port)6379上运行，并使用密码为 mypass。

[yiibai@ubuntu:~]$ redis-cli -h 127.0.0.1 -p 6379 -a "mypass"

redis 127.0.0.1:6379>

redis 127.0.0.1:6379> PING

PONG

不介绍mc里面已经有的东东，只列出特殊的：

* TYPE key — 用来获取某key的类型
* KEYS pattern — 匹配所有符合模式的key，比如KEYS \* 就列出所有的key了，当然，复杂度O(n)
* RANDOMKEY - 返回随机的一个key
* RENAME oldkey[newkey](https://baike.baidu.com/item/newkey)— key也可以改名

列表操作，精华

* RPUSH key string — 将某个值加入到一个key列表末尾
* LPUSH key string — 将某个值加入到一个key列表头部
* LLEN key — 列表长度
* LRANGE key start end — 返回列表中某个范围的值，相当于mysql里面的[分页](https://baike.baidu.com/item/%E5%88%86%E9%A1%B5)查询那样
* LTRIM key start end — 只保留列表中某个范围的值
* LINDEX key index — 获取列表中特定索引号的值，要注意是O(n)复杂度
* LSET key index value — 设置列表中某个位置的值
* LPOP key
* RPOP key — 和上面的LPOP一样，就是类似栈或队列的那种取头取尾指令，可以当成[消息队列](https://baike.baidu.com/item/%E6%B6%88%E6%81%AF%E9%98%9F%E5%88%97)来使用了

集合操作

* SADD key member — 增加元素
* SREM key member — 删除元素
* SCARD key — 返回集合大小
* SISMEMBER key member — 判断某个值是否在集合中
* SINTER key1 key2 ... keyN — 获取多个集合的交集元素
* SMEMBERS key — 列出集合的所有元素

还有Multiple DB的命令，可以更换db，数据可以隔离开，默认是存放在DB 0。

## 键命令

Redis键命令用于管理**Redis**中的键。以下是使用redis键命令的语法。

**语法**

redis 127.0.0.1:6379> COMMAND KEY\_NAME

**示例**

redis 127.0.0.1:6379> SET akey redis

OK

redis 127.0.0.1:6379> DEL akey

(integer) 1

127.0.0.1:6379> GET akey

(nil)

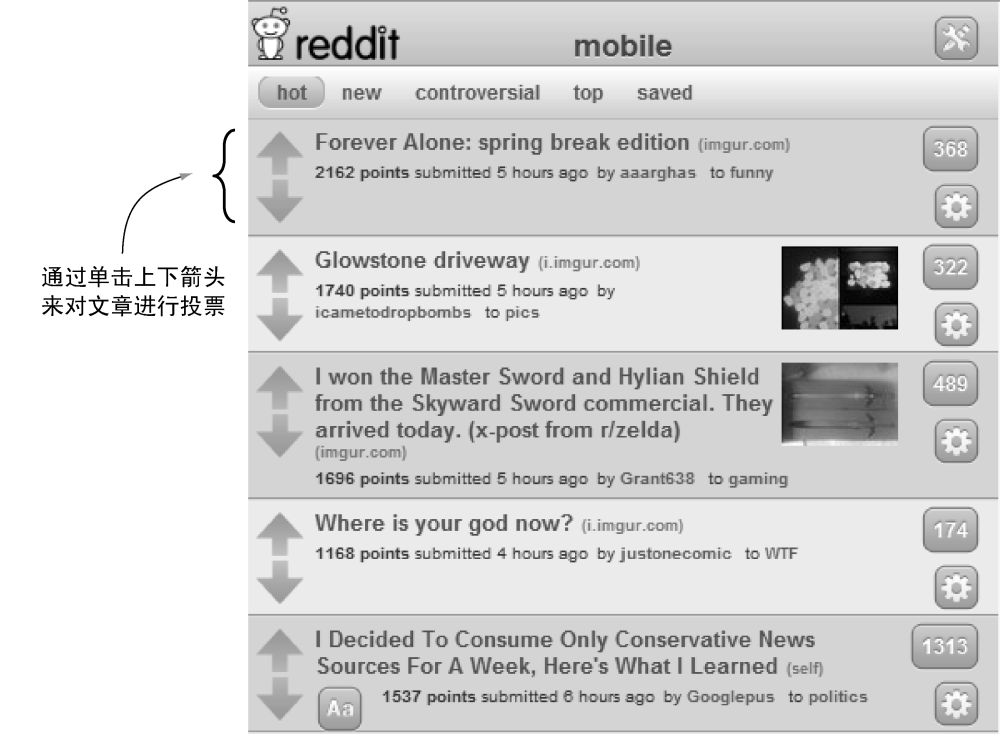
在上面的例子中，DEL是Redis的命令，而akey是键的名称。如果键被删除，则命令的输出将为(integer) 1，否则为(integer) 0。

下表列出了与键相关的一些基本命令。

| **编号** | **命令** | **描述** |
| --- | --- | --- |
| 1 | [DEL key](http://www.yiibai.com/redis/keys_del.html) | 此命令删除一个指定键(如果存在)。 |
| 2 | [DUMP key](http://www.yiibai.com/redis/keys_dump.html) | 此命令返回存储在指定键的值的序列化版本。 |
| 3 | [EXISTS key](http://www.yiibai.com/redis/keys_exists.html) | 此命令检查键是否存在。 |
| 4 | [EXPIRE key seconds](http://www.yiibai.com/redis/keys_expire.html) | 设置键在指定时间秒数之后到期/过期。 |
| 5 | [EXPIREAT key timestamp](http://www.yiibai.com/redis/keys_expireat.html) | 设置在指定时间戳之后键到期/过期。这里的时间是Unix时间戳格式。 |
| 6 | [PEXPIRE key milliseconds](http://www.yiibai.com/redis/keys_pexpire.html) | 设置键的到期时间(以毫秒为单位)。 |
| 7 | [PEXPIREAT key milliseconds-timestamp](http://www.yiibai.com/redis/keys_pexpireat.html) | 以Unix时间戳形式来设置键的到期时间(以毫秒为单位)。 |
| 8 | [KEYS pattern](http://www.yiibai.com/redis/keys_keys.html) | 查找与指定模式匹配的所有键。 |
| 9 | [MOVE key db](http://www.yiibai.com/redis/keys_move.html) | 将键移动到另一个数据库。 |
| 10 | [PERSIST key](http://www.yiibai.com/redis/keys_persist.html) | 删除指定键的过期时间，得永生。 |
| 11 | [PTTL key](http://www.yiibai.com/redis/keys_pttl.html) | 获取键的剩余到期时间。 |
| 12 | [RANDOMKEY](http://www.yiibai.com/redis/keys_randomkey.html) | 从Redis返回一个随机的键。 |
| 13 | [RENAME key newkey](http://www.yiibai.com/redis/keys_rename.html) | 更改键的名称。 |
| 14 | [PTTL key](http://www.yiibai.com/redis/keys_pttl.html) | 获取键到期的剩余时间(以毫秒为单位)。 |
| 15 | [RENAMENX key newkey](http://www.yiibai.com/redis/keys_renamenx.html) | 如果新键不存在，重命名键。 |
| 16 | [TYPE key](http://www.yiibai.com/redis/keys_type.html) | 返回存储在键中的值的数据类型。 |

# 你好Redis

在对Redis提供的5种结构有了基本的了解之后，现在是时候来学习一下怎样使用这些结构来解决实际问题了。最近几年，越来越多的网站开始提供对网页链接、文章或者问题进行投票的功能，其中包括图1-6展示的reddit以及图1-7展示的StackOverflow。这些网站会根据文章的发布时间和文章获得的投票数量计算出一个评分，然后按照这个评分来决定如何排序和展示文章。本节将展示如何使用Redis来构建一个简单的文章投票网站的后端。



**图1-6　Reddit是一个可以对文章进行投票的网站**



**图1-7　StackOverflow是一个可以对问题进行投票的网站**

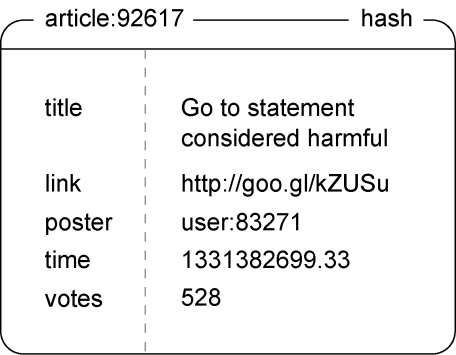
## 对文章进行投票

要构建一个文章投票网站，我们首先要做的就是为了这个网站设置一些数值和限制条件：如果一篇文章获得了至少200张支持票（up vote），那么网站就认为这篇文章是一篇有趣的文章；假如这个网站每天发布1000篇文章，而其中的50篇符合网站对有趣文章的要求，那么网站要做的就是把这50篇文章放到文章列表前100位至少一天；另外，这个网站暂时不提供投反对票（down vote）的功能。

为了产生一个能够随着时间流逝而不断减少的评分，程序需要根据文章的发布时间和当前时间来计算文章的评分，具体的计算方法为：将文章得到的支持票数量乘以一个常数，然后加上文章的发布时间，得出的结果就是文章的评分。

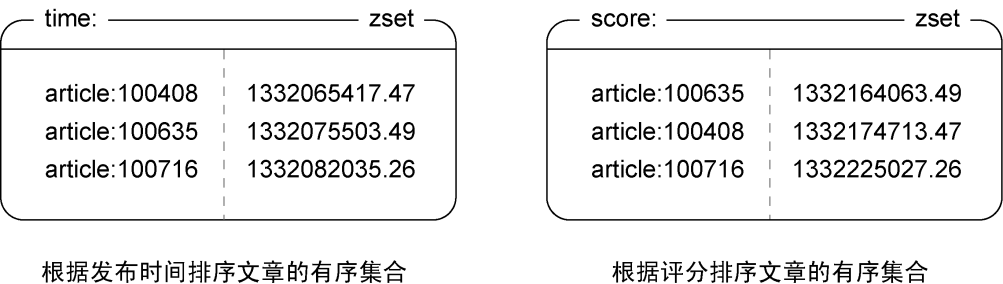
我们使用从UTC时区1970年1月1日到现在为止经过的秒数来计算文章的评分，这个值通常被称为Unix时间。之所以选择使用Unix时间，是因为在所有能够运行Redis的平台上面，使用编程语言获取这个值都是一件非常简单的事情。另外，计算评分时与支持票数量相乘的常量为432，这个常量是通过将一天的秒数（86 400）除以文章展示一天所需的支持票数量（200）得出的：文章每获得一张支持票，程序就需要将文章的评分增加432分。

构建文章投票网站除了需要计算文章评分之外，还需要使用Redis结构存储网站上的各种信息。对于网站里的每篇文章，程序都使用一个散列来存储文章的标题、指向文章的网址、发布文章的用户、文章的发布时间、文章得到的投票数量等信息，图1-8展示了一个使用散列来存储文章信息的例子。



**图1-8　一个使用散列存储文章信息的例子**

文章投票网站将使用两个有序集合来有序地存储文章：第一个有序集合的成员为文章 ID，分值为文章的发布时间；第二个有序集合的成员同样为文章 ID，而分值则为文章的评分。通过这两个有序集合，网站既可以根据文章发布的先后顺序来展示文章，又可以根据文章评分的高低来展示文章，图1-9展示了这两个有序集合的一个示例。

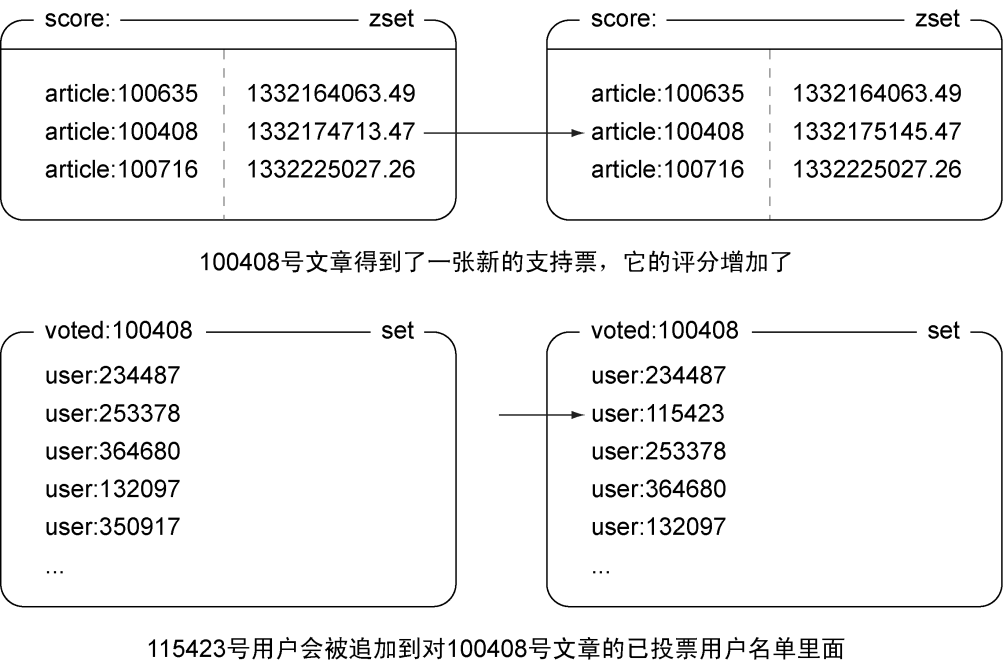


**图1-9　两个有序集合分别记录了根据发布时间排序的文章和根据评分排序的文章**

为了防止用户对同一篇文章进行多次投票，网站需要为每篇文章记录一个已投票用户名单。为此，程序将为每篇文章创建一个集合，并使用这个集合来存储所有已投票用户的ID。

为了尽量节约内存，我们规定当一篇文章发布期满一周之后，用户将不能再对它进行投票，文章的评分将被固定下来，而记录文章已投票用户名单的集合也会被删除。

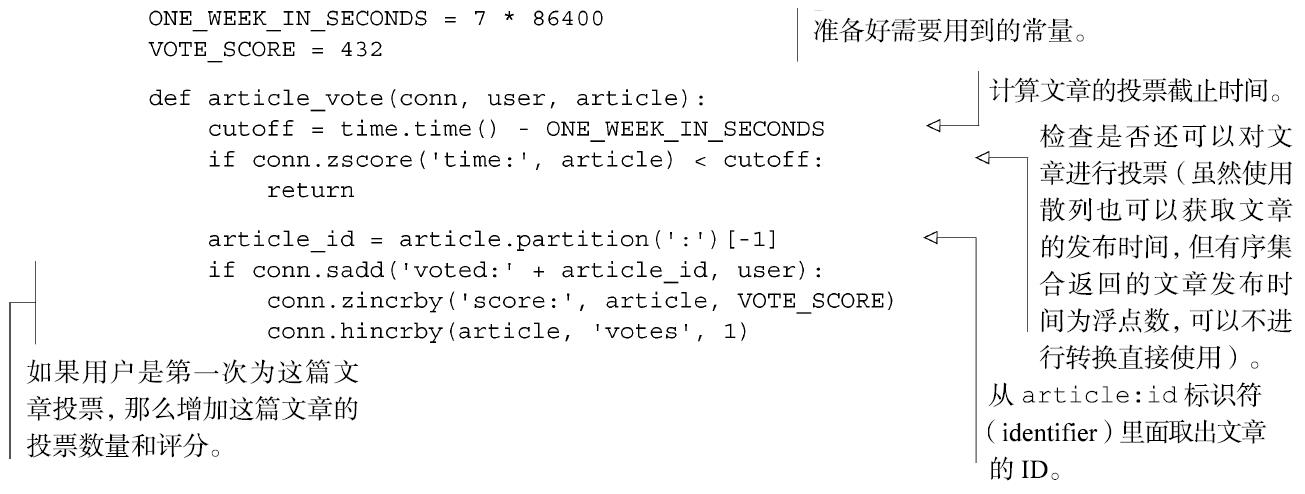
在实现投票功能之前，让我们来看看图 1-11：这幅图展示了当115423号用户给100408号文章投票的时候，数据结构发生的变化。



**图1-11　当115423号用户给100408号文章投票的时候，数据结构发生的变化**

既然我们已经知道了网站计算文章评分的方法，也知道了网站存储数据所需的数据结构，那么现在是时候实际地实现这个投票功能了！当用户尝试对一篇文章进行投票时，程序需要使用ZSCORE命令检查记录文章发布时间的有序集合，判断文章的发布时间是否未超过一周。如果文章仍然处于可以投票的时间范围之内，那么程序将使用SADD命令，尝试将用户添加到记录文章已投票用户名单的集合里面。如果添加操作执行成功的话，那么说明用户是第一次对这篇文章进行投票，程序将使用ZINCRBY命令为文章的评分增加432分（ZINCRBY命令用于对有序集合成员的分值执行自增操作），并使用HINCRBY命令对散列记录的文章投票数量进行更新（HINCRBY命令用于对散列存储的值执行自增操作），代码清单1-6展示了投票功能的实现代码。

**代码清单1-6**article\_vote()**函数**

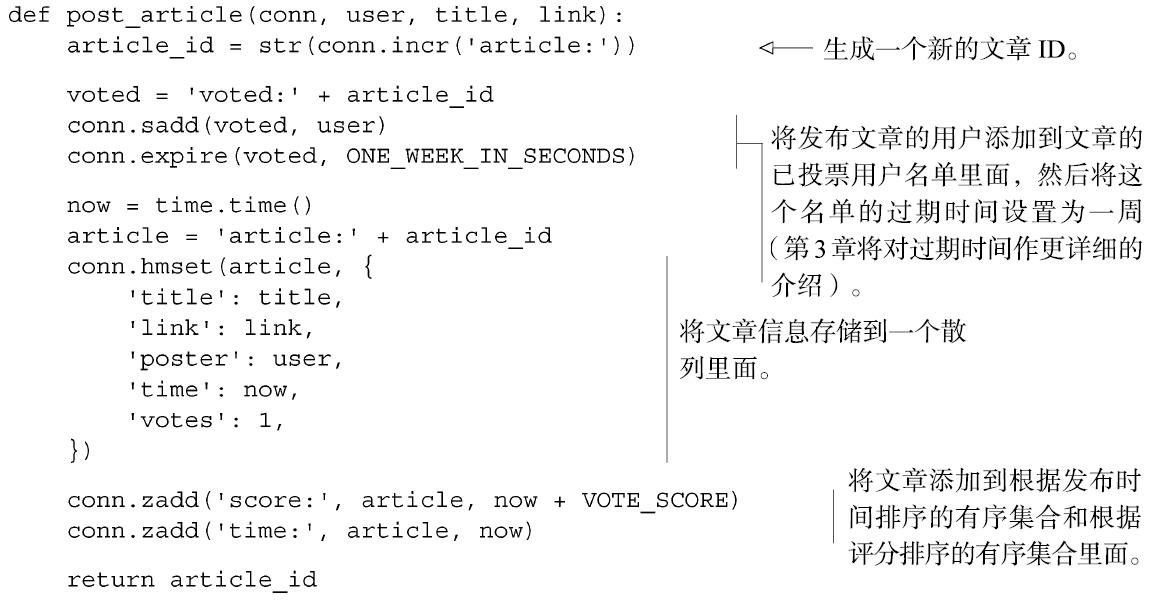


这个投票功能还是很不错的，对吧？那么发布文章的功能要怎么实现呢？

## 发布并获取文章

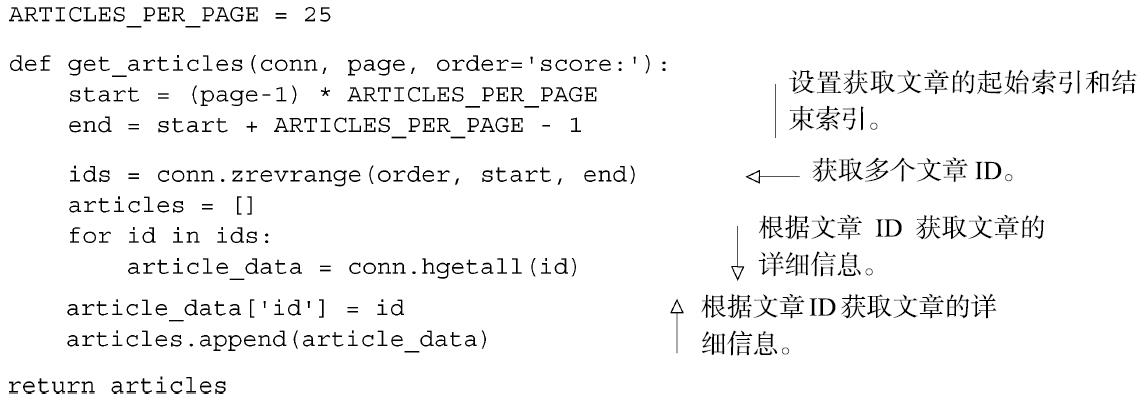
发布一篇新文章首先需要创建一个新的文章ID，这项工作可以通过对一个计数器（counter）执行INCR命令来完成。接着程序需要使用SADD将文章发布者的ID添加到记录文章已投票用户名单的集合里面，并使用EXPIRE命令为这个集合设置一个过期时间，让Redis在文章发布期满一周之后自动删除这个集合。之后，程序会使用HMSET命令来存储文章的相关信息，并执行两个ZADD命令，将文章的初始评分（initial score）和发布时间分别添加到两个相应的有序集合里面。代码清单1-7展示了发布新文章功能的实现代码。

**代码清单1-7**post\_article()**函数**



好了，我们已经陆续实现了文章投票功能和文章发布功能，接下来要考虑的就是如何取出评分最高的文章以及如何取出最新发布的文章了。为了实现这两个功能，程序需要先使用ZREVRANGE命令取出多个文章ID，然后再对每个文章ID执行一次HGETALL命令来取出文章的详细信息，这个方法既可以用于取出评分最高的文章，又可以用于取出最新发布的文章。这里特别要注意的一点是，因为有序集合会根据成员的分值从小到大地排列元素，所以使用ZREVRANGE命令，以“分值从大到小”的排列顺序取出文章ID才是正确的做法，代码清单1-8展示了文章获取功能的实现函数。

**代码清单1-8**get\_articles()**函数**



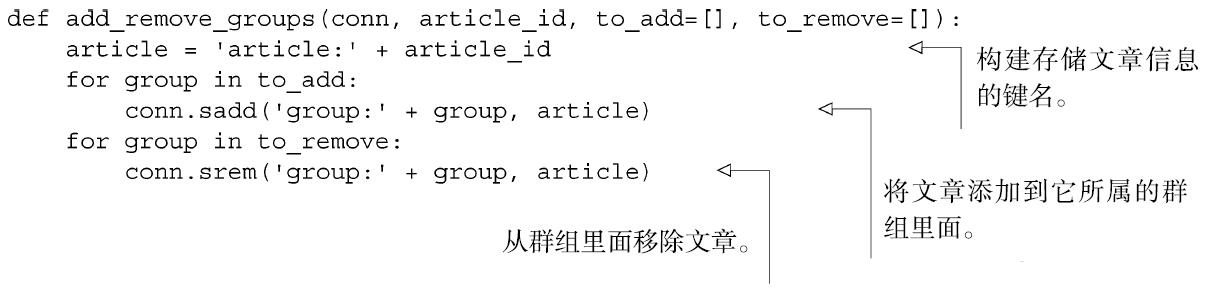
**Python的默认值参数和关键字参数**　代码清单1-8中的get\_articles()函数为order参数设置了默认值score:。Python语言的初学者可能会对“默认值参数”以及“根据名字（而不是位置）来传入参数”的一些细节感到陌生。如果读者在理解函数定义或者参数传递方面有困难，可以考虑去看看《Python语言教程》，教程里面对这两个方面进行了很好的介绍，点击以下短链接就可以直接访问教程的相关章节：<http://mng.bz/KM5x>。

虽然我们构建的网站现在已经可以展示最新发布的文章和评分最高的文章了，但它还不具备目前很多投票网站都支持的群组（group）功能：这个功能可以让用户只看见与特定话题有关的文章，比如与“可爱的动物”有关的文章、与“政治”有关的文章、与“Java编程”有关的文章或者介绍“Redis用法”的文章等等。接下来的一节将向我们展示为文章投票网站添加群组功能的方法。

## 对文章进行分组

群组功能由两个部分组成，一个部分负责记录文章属于哪个群组，另一个部分负责取出群组里面的文章。为了记录各个群组都保存了哪些文章，网站需要为每个群组创建一个集合，并将所有同属一个群组的文章ID都记录到那个集合里面。代码清单1-9展示了将文章添加到群组里面的方法，以及从群组里面移除文章的方法。

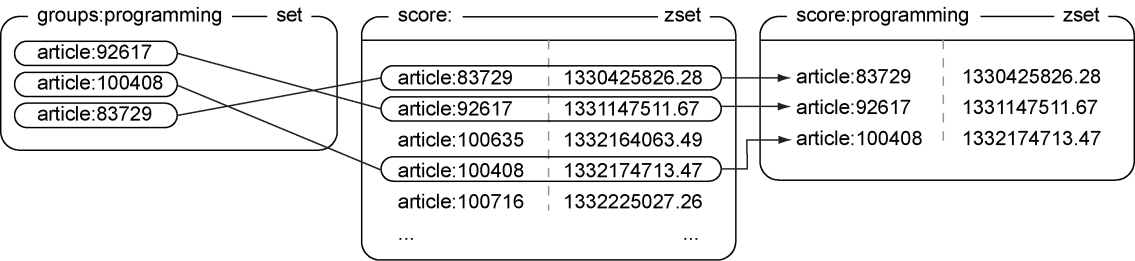
**代码清单1-9**add\_remove\_groups()**函数**



初看上去，可能会有读者觉得使用集合来记录群组文章并没有多大用处。到目前为止，读者只看到了集合结构检查某个元素是否存在的能力，但实际上Redis不仅可以对多个集合执行操作，甚至在一些情况下，还可以在集合和有序集合之间执行操作。

为了能够根据评分对群组文章进行排序和分页（paging），网站需要将同一个群组里面的所有文章都按照评分有序地存储到一个有序集合里面。Redis的ZINTERSTORE命令可以接受多个集合和多个有序集合作为输入，找出所有同时存在于集合和有序集合的成员，并以几种不同的方式来组合（combine）这些成员的分值（所有集合成员的分值都会被视为是1）。对于我们的文章投票网站来说，程序需要使用ZINTERSTORE命令选出相同成员中最大的那个分值来作为交集成员的分值：取决于所使用的排序选项，这些分值既可以是文章的评分，也可以是文章的发布时间。

图 1-12 展示了对一个包含少量文章的群组集合和一个包含大量文章及评分的有序集合执行ZINTERSTORE命令的过程，注意观察那些同时出现在集合和有序集合里面的文章是怎样被添加到结果有序集合里面的。

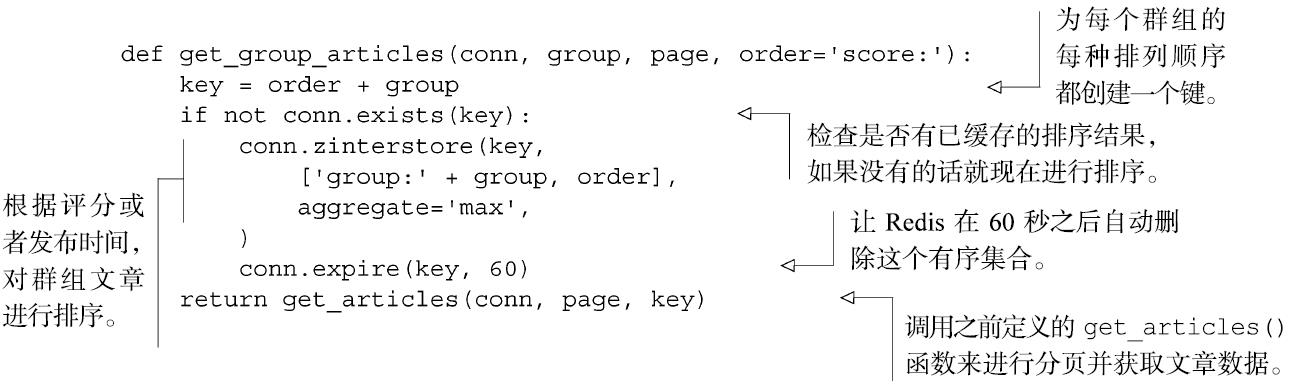


**图1-12**

对集合groups:programming和有序集合score:进行交集计算得出了新的有序集合score:programming，它包含了所有同时存在于集合groups:programming和有序集合score:的成员。因为集合groups:programming的所有成员的分值都被视为是1，而有序集合score:的所有成员的分值都大于1，并且这次交集计算挑选的分值为相同成员中的最大分值，所以有序集合score:programming的成员的分值实际上是由有序集合score:的成员的分值来决定的

通过对存储群组文章的集合和存储文章评分的有序集合执行ZINTERSTORE命令，程序可以得到按照文章评分排序的群组文章；而通过对存储群组文章的集合和存储文章发布时间的有序集合执行ZINTERSTORE命令，程序则可以得到按照文章发布时间排序的群组文章。如果群组包含的文章非常多，那么执行ZINTERSTORE命令就会比较花时间，为了尽量减少Redis的工作量，程序会将这个命令的计算结果缓存60秒。另外，我们还重用了已有的get\_articles()函数来分页并获取群组文章，代码清单1-10展示了网站从群组里面获取一整页文章的方法。

**代码清单1-10**get\_group\_articles()**函数**



有些网站只允许用户将文章放在一个或者两个群组里面（其中一个是“所有文章”群组，另一个是最适合文章的群组）。在这种情况下，最好直接将文章所在的群组记录到存储文章信息的散列里面，并在article\_vote()函数的末尾增加一个ZINCRBY命令调用，用于更新文章在群组中的评分。但是在这个示例里面，我们构建的文章投票网站允许一篇文章同时属于多个群组（比如一篇文章可以同时属于“编程”和“算法”两个群组），所以对于一篇同时属于多个群组的文章来说，更新文章的评分意味着程序需要对文章所属的全部群组执行自增操作。在这种情况下，如果一篇文章同时属于很多个群组，那么更新文章评分这一操作可能会变得相当耗时，因此，我们在get\_group\_articles()函数里面对ZINTERSTORE命令的执行结果进行了缓存处理，以此来尽量减少ZINTERSTORE命令的执行次数。开发者在灵活性或限制条件之间的取舍将改变程序存储和更新数据的方式，这一点对于任何数据库都是适用的，Redis也不例外。

# Redis进阶

## Redis发送订阅

Redis发布订阅(pub/sub)是一种消息通信模式：发送者(pub)发送消息，订阅者(sub)接收消息。

Redis 发布订阅(pub/sub)实现了消息系统，发送者(在redis术语中称为发布者)在接收者(订阅者)接收消息时发送消息。传送消息的链路称为信道。

在Redis中，客户端可以订阅任意数量的信道。

**示例**

以下示例说明了发布用户概念的工作原理。 在以下示例中，一个客户端订阅名为“redisChat”的信道。

redis 127.0.0.1:6379> SUBSCRIBE redisChat

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

1) "subscribe"

2) "redisChat"

3) (integer) 1

现在，两个客户端在名称为“redisChat”的相同信道上发布消息，并且上述订阅的客户端接收消息。

redis 127.0.0.1:6379> PUBLISH redisChat "Redis is a great caching technique"

(integer) 1

redis 127.0.0.1:6379> PUBLISH redisChat "Learn redis by yiibai"

(integer) 1

1) "message"

2) "redisChat"

3) "Redis is a great caching technique"

1) "message"

2) "redisChat"

3) "Learn redis by yiibai"

## Redis事务

Redis事务允许在单个步骤中执行一组命令。以下是事务的两个属性：

* 事务中的所有命令作为单个隔离操作并按顺序执行。不可以在执行Redis事务的中间向另一个客户端发出的请求。
* Redis事务也是原子的。原子意味着要么处理所有命令，要么都不处理。

**语法**

Redis事务由命令MULTI命令启动，然后需要传递一个应该在事务中执行的命令列表，然后整个事务由EXEC命令执行。

redis 127.0.0.1:6379> MULTI

OK

List of commands here

redis 127.0.0.1:6379> EXEC

**示例**

以下示例说明了如何启动和执行Redis事务。

redis 127.0.0.1:6379> MULTI

OK

redis 127.0.0.1:6379> SET mykey "redis"

QUEUED

redis 127.0.0.1:6379> GET mykey

QUEUED

redis 127.0.0.1:6379> INCR visitors

QUEUED

redis 127.0.0.1:6379> EXEC

1) OK

2) "redis"

3) (integer) 1

## Redis脚本

Redis脚本用于使用Lua解释器来执行脚本。从Redis 2.6.0版开始内置到Redis中。使用脚本的命令是**EVAL**命令。

**语法**

以下是EVAL命令的基本语法。

redis 127.0.0.1:6379> EVAL script numkeys key [key ...] arg [arg ...]

**示例**

以下示例说明了Redis脚本的工作原理。

redis 127.0.0.1:6379> EVAL "return {KEYS[1],KEYS[2],ARGV[1],ARGV[2]}" 2 key1

key2 first second

1) "key1"

2) "key2"

3) "first"

4) "second"

## Redis连接

Redis中的连接命令基本上是用于管理与Redis服务器的客户端连接。

**示例**

以下示例说明客户端如何向Redis服务器验证自身，并检查服务器是否正在运行。

redis 127.0.0.1:6379> AUTH "password"

OK

redis 127.0.0.1:6379> PING

PONG

**Redis连接命令**

下表列出了与Redis连接相关的一些基本命令。

| **序号** | **命令** | **说明** |
| --- | --- | --- |
| 1 | [AUTH password](http://www.yiibai.com/redis/connection_auth.html) | 使用给定的密码验证服务器 |
| 2 | [ECHO message](http://www.yiibai.com/redis/connection_echo.html) | 打印给定的字符串信息 |
| 3 | [PING](http://www.yiibai.com/redis/connection_ping.html) | 检查服务器是否正在运行 |
| 4 | [QUIT](http://www.yiibai.com/redis/connection_quit.html) | 关闭当前连接 |
| 5 | [SELECT index](http://www.yiibai.com/redis/connection_select.html) | 更改当前连接的所选数据库 |

## Redis服务器

Redis服务器命令基本上是用于管理Redis服务器。

**示例**

以下示例说明了如何获取有关服务器的所有统计信息和信息。

127.0.0.1:6379> info

# Server

redis\_version:2.8.4

redis\_git\_sha1:00000000

redis\_git\_dirty:0

redis\_build\_id:8f6097d7914679ca

redis\_mode:standalone

os:Linux 3.19.0-25-generic i686

arch\_bits:32

multiplexing\_api:epoll

gcc\_version:4.8.2

process\_id:1004

run\_id:1e53acea2aa628199c4e438a3ed815d96eebc036

tcp\_port:6379

uptime\_in\_seconds:888450

uptime\_in\_days:10

hz:10

lru\_clock:1861984

config\_file:/etc/redis/redis.conf

# Clients

connected\_clients:1

client\_longest\_output\_list:0

client\_biggest\_input\_buf:0

blocked\_clients:0

# Memory

used\_memory:424872

used\_memory\_human:414.91K

used\_memory\_rss:6709248

used\_memory\_peak:424464

used\_memory\_peak\_human:414.52K

used\_memory\_lua:22528

mem\_fragmentation\_ratio:15.79

mem\_allocator:jemalloc-3.4.1

# Persistence

loading:0

rdb\_changes\_since\_last\_save:0

rdb\_bgsave\_in\_progress:0

rdb\_last\_save\_time:1486607123

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

# Stats

total\_connections\_received:1

total\_commands\_processed:263

instantaneous\_ops\_per\_sec:0

rejected\_connections:0

sync\_full:0

sync\_partial\_ok:0

sync\_partial\_err:0

expired\_keys:0

evicted\_keys:0

keyspace\_hits:257

keyspace\_misses:0

pubsub\_channels:0

pubsub\_patterns:0

latest\_fork\_usec:4793

# 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:24.65

used\_cpu\_user:15.84

used\_cpu\_sys\_children:0.08

used\_cpu\_user\_children:0.00

# Keyspace

db0:keys=14,expires=0,avg\_ttl=0

db1:keys=1,expires=0,avg\_ttl=0

127.0.0.1:6379>

## Redis备份

Redis数据库可以使用安全的方案，使得进行连接的任何客户端在执行命令之前都需要进行身份验证。要保护Redis安全，需要在配置文件中设置密码。

**示例**

下面的示例显示了保护Redis实例的步骤。

127.0.0.1:6379> CONFIG get requirepass

1) "requirepass"

2) ""

默认情况下，此属性为空，这表示还没有为此实例设置密码。您可以通过执行以下命令更改此属性。

127.0.0.1:6379> CONFIG set requirepass "yiibai"

OK

127.0.0.1:6379> CONFIG get requirepass

1) "requirepass"

2) "yiibai"

设置密码后，如果任何客户端运行命令而不进行身份验证，则会返回一个**(error) NOAUTH Authentication required.**的错误信息。 因此，客户端需要使用AUTH命令来验证。

**语法**

以下是AUTH命令的基本语法。

127.0.0.1:6379> AUTH password

**示例**

127.0.0.1:6379> AUTH "yiibai"

OK

127.0.0.1:6379> SET mykey "Test value"

OK

127.0.0.1:6379> GET mykey

"Test value"

## Redis客户端连接

Redis在配置的监听TCP端口和Unix套接字上等待和接受客户端的连接(如果已启用)。 当接受新的客户端连接时，执行以下操作：

* 由于Redis使用复用和非阻塞I/O，因此客户端套接字处于非阻塞状态。
* 设置TCP\_NODELAY选项是为了确保连接不延迟。
* 创建可读文件事件，以便Redis能够在套接字上读取新数据时收集客户端查询。

**最大客户数**

在Redis配置文件(redis.conf)中，有一个名称为maxclients的属性，它描述了可以连接到Redis的客户端的最大数量。

以下是命令的基本语法。

127.0.0.1:6379> config get maxclients

1) "maxclients"

2) "3984"

默认情况下，此属性设置为10000(取决于操作系统的文件描述符限制的最大数量)，但您可以更改此属性。

**示例**

在以下示例中，我们已将客户端的最大数目设置为100000，并启动服务器。

yiibai@ubuntu:~$ redis-server --maxclients 100000

**客户端命令**

| **编号** | **命令** | **描述** |
| --- | --- | --- |
| 1 | CLIENT LIST | 返回连接到Redis服务器的客户端列表 |
| 2 | CLIENT SETNAME | 为当前连接分配/设置新的名称 |
| 3 | CLIENT GETNAME | 返回由CLIENT SETNAME设置的当前连接的名称 |
| 4 | CLIENT PAUSE | 这是一个连接控制命令，能够将所有Redis客户端按指定的时间量(以毫秒为单位)挂起 |
| 5 | CLIENT KILL | 此命令关闭指定的客户端连接。 |

## Redis管道

Redis是一个TCP服务器，支持请求/响应协议。 在Redis中，请求通过以下步骤完成：

* 客户端向服务器发送查询，并从套接字读取，通常以阻塞的方式，用于服务器响应。
* 服务器处理命令并将响应发送回客户端。

**管道的意义**

管道的基本含义是，客户端可以向服务器发送多个请求，而不必等待回复，并最终在一个步骤中读取回复。

**示例**

要检查Redis管道，只需启动Redis实例，并在终端中键入以下命令。

$(echo -en "PING\r\n SET tutorial redis\r\nGET tutorial\r\nINCR

visitor\r\nINCR visitor\r\nINCR visitor\r\n"; sleep 10) | nc localhost 6379

+PONG

+OK

redis

:1

:2

:3

在上面的例子中，我们将使用PING命令检查Redis连接。这里设置了一个名称为tutorial的字符串，值为redis。 然后得到键值，并增加 visitor 数量三次。 在结果中，我们可以看到所有命令都提交到Redis一次，Redis在一个步骤中提供所有命令的输出。

**管道的好处**

这种技术的好处是大大提高了协议性能。通过管道从连接到本地主机速度增加五倍，因特网连接的至少快一百倍。

## Redis分区

分区是将数据拆分为多个Redis实例的过程，因此每个实例只包含一部分键。

**分区的优点**

* 它允许更大的数据库，使用更多计算机的内存总和。如果没有分区，则限制为单个计算机可以支持的内存量。
* 它允许将计算能力扩展到多个核心和多个计算机，并将网络带宽扩展到多个计算机和网络适配器。

**分区的缺点**

* 通常不支持涉及多个键的操作。 例如，如果两个集合存储在映射到不同Redis实例的键中，则不能执行两个集合之间的交集操作。
* 不能使用涉及多个键的Redis事务。
* 分区粒度是关键，因此不可能使用单个巨大的键(如非常大的排序集合)来分割数据集。
* 使用分区时，数据处理更复杂。 例如，必须处理多个**RDB/AOF**文件，并获得数据的备份，您需要聚合来自多个实例和主机的持久性文件。
* 添加和删除容量可能很复杂。 例如，Redis Cluster支持大多数透明的数据重新平衡，具有在运行时添加和删除节点的能力。但是，其他系统(如客户端分区和代理)不支持此功能。但可以使用一种叫作**Presharding**的技术来处理这方面的问题。

**分区类型**

Redis中有两种类型的分区。假设有四个Redis实例：R0，R1，R2，R3以许多代表用户的键，如user：1，user：2，…等等。

* **范围分区**

范围分区通过将对象的范围映射到特定的Redis实例来实现。假设在上面示例中，从ID 0到ID 10000的用户将进入实例R0，而从ID 10001到ID 20000的用户将进入实例R1，以此类推。

* **哈希分区**

在这种类型的分区中，使用散列函数(例如，模函数)将键转换成数字，然后将数据存储在不同的Redis实例中。

# 各类语言连接Redis

## Java连接Redis

在Java程序中使用Redis之前，需要确保在机器上安装了Redis的Java驱动程序和Java环境。可以先在将Java电脑上并配置好环境。

**安装**

现在，让我们看看如何设置Redis Java驱动程序。

* 下载jedis.jar - <http://repo1.maven.org/maven2/redis/clients/jedis/2.1.0/jedis-2.1.0-sources.jar> ，确保下载的jedis.jar是最新版本。
* 将jedis.jar包含到类路径中。

**Java连接到Redis服务器**

请参考以下一个简单的示例代码 -

import redis.clients.jedis.Jedis;

public class RedisJava {

public static void main(String[] args) {

//Connecting to Redis server on localhost

Jedis jedis = new Jedis("localhost");

System.out.println("Connection to server sucessfully");

//check whether server is running or not

System.out.println("Server is running: "+jedis.ping());

}

}

现在，编译并运行上面的程序来测试与Redis服务器的连接。可以根据需要更改路径。假设jedis.jar的当前版本在当前路径中可以使用。

执行上面代码，将生成以下结果 -

$javac RedisJava.java

$java RedisJava

Connection to server sucessfully

Server is running: PONG

**Redis Java字符串示例**

import redis.clients.jedis.Jedis;

public class RedisStringJava {

public static void main(String[] args) {

//Connecting to Redis server on localhost

Jedis jedis = new Jedis("localhost");

System.out.println("Connection to server sucessfully");

//set the data in redis string

jedis.set("tutorial-name", "Redis tutorial");

// Get the stored data and print it

System.out.println("Stored string in redis:: "+ jedis.get("tutorialname"));

}

}

执行上面代码，将生成以下结果 -

$javac RedisStringJava.java

$java RedisStringJava

Connection to server sucessfully

Stored string in redis:: Redis tutorial

**Redis Java列表示例**

import redis.clients.jedis.Jedis;

public class RedisListJava {

public static void main(String[] args) {

//Connecting to Redis server on localhost

Jedis jedis = new Jedis("localhost");

System.out.println("Connection to server sucessfully");

//store data in redis list

jedis.lpush("tutorial-list", "Redis");

jedis.lpush("tutorial-list", "Mongodb");

jedis.lpush("tutorial-list", "Mysql");

// Get the stored data and print it

List<String> list = jedis.lrange("tutorial-list", 0 ,5);

for(int i = 0; i<list.size(); i++) {

System.out.println("Stored string in redis:: "+list.get(i));

}

}

}

执行上面代码，将生成以下结果 -

$javac RedisListJava.java

$java RedisListJava

Connection to server sucessfully

Stored string in redis:: Redis

Stored string in redis:: Mongodb

Stored string in redis:: Mysql

**Redis Java键示例**

import redis.clients.jedis.Jedis;

public class RedisKeyJava {

public static void main(String[] args) {

//Connecting to Redis server on localhost

Jedis jedis = new Jedis("localhost");

System.out.println("Connection to server sucessfully");

//store data in redis list

// Get the stored data and print it

List<String> list = jedis.keys("\*");

for(int i = 0; i<list.size(); i++) {

System.out.println("List of stored keys:: "+list.get(i));

}

}

}

执行上面代码，将生成以下结果 -

$javac RedisKeyJava.java

$java RedisKeyJava

Connection to server sucessfully

List of stored keys:: tutorial-name

List of stored keys:: tutorial-list

## PHP连接Redis

在php程序中使用Redis之前，需要确保在机器上安装了Redis的PHP驱动程序和PHP环境。可以先在将PHP电脑上并配置好环境。

**安装**

现在，让我们看看如何设置Redis PHP驱动程序。

从github库下载phpredis=> [http://github.com/nicolasff/phpredis。](http://github.com/nicolasff/phpredis%E3%80%82) 当下载它之后，提取文件到phpredis目录。在Ubuntu上，安装以下扩展。

cd phpredis

sudo phpize

sudo ./configure

sudo make

sudo make install

现在，将“modules”文件夹的内容复制并粘贴到PHP扩展目录中，并在php.ini中添加以下行。

extension = redis.so

现在，Redis PHP安装完成！

**使用连接到Redis服务器**

<?php

//Connecting to Redis server on localhost

$redis = new Redis();

$redis->connect('127.0.0.1', 6379);

echo "Connection to server sucessfully";

//check whether server is running or not

echo "Server is running: ".$redis->ping();

?>

当程序执行时，将产生以下结果。

Connection to server sucessfully

Server is running: PONG

**Redis PHP字符串示例**

<?php

//Connecting to Redis server on localhost

$redis = new Redis();

$redis->connect('127.0.0.1', 6379);

echo "Connection to server sucessfully";

//set the data in redis string

$redis->set("tutorial-name", "Redis tutorial");

// Get the stored data and print it

echo "Stored string in redis:: " .$redis→get("tutorial-name");

?>

执行上面代码，将生成以下结果 -

Connection to server sucessfully

Stored string in redis:: Redis tutorial

**Redis php列表示例**

<?php

//Connecting to Redis server on localhost

$redis = new Redis();

$redis->connect('127.0.0.1', 6379);

echo "Connection to server sucessfully";

//store data in redis list

$redis->lpush("tutorial-list", "Redis");

$redis->lpush("tutorial-list", "Mongodb");

$redis->lpush("tutorial-list", "Mysql");

// Get the stored data and print it

$arList = $redis->lrange("tutorial-list", 0 ,5);

echo "Stored string in redis:: ";

print\_r($arList);

?>

执行上面代码，将生成以下结果 -

Connection to server sucessfully

Stored string in redis::

Redis

Mongodb

Mysql

**Redis php键示例**

<?php

//Connecting to Redis server on localhost

$redis = new Redis();

$redis->connect('127.0.0.1', 6379);

echo "Connection to server sucessfully";

// Get the stored keys and print it

$arList = $redis->keys("\*");

echo "Stored keys in redis:: "

print\_r($arList);

?>

执行上面代码，将生成以下结果 -

Connection to server sucessfully

Stored string in redis::

tutorial-name

tutorial-list

## C#连接Redis

前面我们已经准备成功开启Redis服务，其端口号为6379，接下来我们就看看如何使用C#语言来操作Redis。就如MongoDB一样，要操作Redis服务，自然就需要下载C#的客户端，这里通过Nuget下载了“ServiceStack.Redis”客户端，引入成功之后，就可以使用C#来对Redis服务进行操作了。

由于Redis一般是用来作为缓存的，也就是一般我们把一些不经常改变的数据通过Redis缓存起来，之后用户的请求就不需要再访问数据库，而可以直接从Redis缓存中直接获取，这样就可以减轻数据库服务器的压力以及加快响应速度。既然是用来做缓存的，也就是通过指定key值来把对应Value保存起来，之后再根据key值来获得之前缓存的值。具体的操作代码如下所示，这里就不过多介绍了。

请参考以下代码 -

class Program

{

static void Main(string[] args)

{

//在Redis中存储常用的5种数据类型：String,Hash,List,SetSorted set

var client = new RedisClient("127.0.0.1", 6379);

//AddString(client);

//AddHash(client);

//AddList(client);

//AddSet(client);

AddSetSorted(client);

Console.ReadLine();

}

private static void AddString(RedisClient client)

{

var timeOut = new TimeSpan(0,0,0,30);

client.Add("Test", "Learninghard", timeOut);

while (true)

{

if (client.ContainsKey("Test"))

{

Console.WriteLine("String Key: Test -Value: {0}, 当前时间: {1}", client.Get<string>("Test"), DateTime.Now);

Thread.Sleep(10000);

}

else

{

Console.WriteLine("Value 已经过期了，当前时间：{0}", DateTime.Now);

break;

}

}

var person = new Person() {Name = "Learninghard", Age = 26};

client.Add("lh", person);

var cachePerson = client.Get<Person>("lh");

Console.WriteLine("Person's Name is : {0}, Age: {1}", cachePerson.Name, cachePerson.Age);

}

private static void AddHash(RedisClient client)

{

if (client == null) throw new ArgumentNullException("client");

client.SetEntryInHash("HashId", "Name", "Learninghard");

client.SetEntryInHash("HashId", "Age", "26");

client.SetEntryInHash("HashId", "Sex", "男");

var hashKeys = client.GetHashKeys("HashId");

foreach (var key in hashKeys)

{

Console.WriteLine("HashId--Key:{0}", key);

}

var haskValues = client.GetHashValues("HashId");

foreach (var value in haskValues)

{

Console.WriteLine("HashId--Value:{0}", value);

}

var allKeys = client.GetAllKeys(); //获取所有的key。

foreach (var key in allKeys)

{

Console.WriteLine("AllKey--Key:{0}", key);

}

}

private static void AddList(RedisClient client)

{

if (client == null) throw new ArgumentNullException("client");

client.EnqueueItemOnList("QueueListId", "1.Learnghard"); //入队

client.EnqueueItemOnList("QueueListId", "2.张三");

client.EnqueueItemOnList("QueueListId", "3.李四");

client.EnqueueItemOnList("QueueListId", "4.王五");

var queueCount = client.GetListCount("QueueListId");

for (var i = 0; i < queueCount; i++)

{

Console.WriteLine("QueueListId出队值：{0}", client.DequeueItemFromList("QueueListId")); //出队(队列先进先出)

}

client.PushItemToList("StackListId", "1.Learninghard"); //入栈

client.PushItemToList("StackListId", "2.张三");

client.PushItemToList("StackListId", "3.李四");

client.PushItemToList("StackListId", "4.王五");

var stackCount = client.GetListCount("StackListId");

for (var i = 0; i < stackCount; i++)

{

Console.WriteLine("StackListId出栈值：{0}", client.PopItemFromList("StackListId")); //出栈(栈先进后出)

}

}

//它是string类型的无序集合。set是通过hash table实现的，添加，删除和查找,对集合我们可以取并集，交集，差集

private static void AddSet(RedisClient client)

{

if (client == null) throw new ArgumentNullException("client");

client.AddItemToSet("Set1001", "A");

client.AddItemToSet("Set1001", "B");

client.AddItemToSet("Set1001", "C");

client.AddItemToSet("Set1001", "D");

var hastset1 = client.GetAllItemsFromSet("Set1001");

foreach (var item in hastset1)

{

Console.WriteLine("Set无序集合Value:{0}", item); //出来的结果是无须的

}

client.AddItemToSet("Set1002", "K");

client.AddItemToSet("Set1002", "C");

client.AddItemToSet("Set1002", "A");

client.AddItemToSet("Set1002", "J");

var hastset2 = client.GetAllItemsFromSet("Set1002");

foreach (var item in hastset2)

{

Console.WriteLine("Set无序集合ValueB:{0}", item); //出来的结果是无须的

}

var hashUnion = client.GetUnionFromSets(new string[] { "Set1001", "Set1002" });

foreach (var item in hashUnion)

{

Console.WriteLine("求Set1001和Set1002的并集:{0}", item); //并集

}

var hashG = client.GetIntersectFromSets(new string[] { "Set1001", "Set1002" });

foreach (var item in hashG)

{

Console.WriteLine("求Set1001和Set1002的交集:{0}", item); //交集

}

var hashD = client.GetDifferencesFromSet("Set1001", new string[] { "Set1002" }); //[返回存在于第一个集合，但是不存在于其他集合的数据。差集]

foreach (var item in hashD)

{

Console.WriteLine("求Set1001和Set1002的差集:{0}", item); //差集

}

}

/\*

sorted set 是set的一个升级版本，它在set的基础上增加了一个顺序的属性，这一属性在添加修改.元素的时候可以指定，

\* 每次指定后，zset(表示有序集合)会自动重新按新的值调整顺序。可以理解为有列的表，一列存 value,一列存顺序。操作中key理解为zset的名字.

\*/

private static void AddSetSorted(RedisClient client)

{

if (client == null) throw new ArgumentNullException("client");

client.AddItemToSortedSet("SetSorted1001", "A");

client.AddItemToSortedSet("SetSorted1001", "B");

client.AddItemToSortedSet("SetSorted1001", "C");

var listSetSorted = client.GetAllItemsFromSortedSet("SetSorted1001");

foreach (var item in listSetSorted)

{

Console.WriteLine("SetSorted有序集合{0}", item);

}

client.AddItemToSortedSet("SetSorted1002", "A", 400);

client.AddItemToSortedSet("SetSorted1002", "D", 200);

client.AddItemToSortedSet("SetSorted1002", "B", 300);

// 升序获取第一个值:"D"

var list = client.GetRangeFromSortedSet("SetSorted1002", 0, 0);

foreach (var item in list)

{

Console.WriteLine(item);

}

//降序获取第一个值:"A"

list = client.GetRangeFromSortedSetDesc("SetSorted1002", 0, 0);

foreach (var item in list)

{

Console.WriteLine(item);

}

}

}

class Person

{

public string Name { get; set; }

public int Age { get; set; }

}

如何要想查看自己操作是否成功，也可以像MongoDB那样下载一个客户端工具，这里推荐一款Redis Desktop Manager。这个工具就相当于SQL Server的客户端工具一样。通过这款工具可以查看Redis服务器中保存的数据和对应格式。其使用也非常简单，只需要添加一个Redis服务连接即可。该工具的下载地址为：<http://pan.baidu.com/s/1sjp55Ul>