

Redis教程
Redis环境安装配置
Redis配置
Redis数据类型
Redis快速入门
Redis命令
Redis命令
Redis键命令
Redis字符串
Redis哈希
Redis列表
Redis集合
Redis HyperLogLog
Redis发送订阅
Redis事务
Redis脚本
Redis连接
Redis服务器
高级应用
Redis备份
Redis基准
Redis客户端连接
Redis管道
Redis分区
Java连接Redis
PHP连接Redis
Windows下安装Redis
C#连接Redis

## Redis快速入门

我要打赏

作者：初生不惑

Redis教程

评论：0 条

Windows产品技术QQ群：227270512


广告

Redis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。

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官方网站是：<http://www.redis.io/>，如下：



Commands Clients Documentation Community Download Support License

Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs and geospatial indexes with radius queries. Redis has built-in replication, Lua scripting, LRU eviction, transactions and different levels of on-disk persistence, and provides high availability via Redis Sentinel and automatic partitioning with Redis Cluster. [Learn more](#) →

### Try it

Ready for a test drive? Check this [interactive tutorial](#) that will walk you through the most important features of Redis.

### Download it

[Redis 3.2.8 is the latest stable version.](#) Interested in release candidates or unstable versions? [Check the downloads page.](#)

### Quick links

Follow day-to-day Redis on [Twitter](#) and [GitHub](#). Get help or help others by subscribing to [our mailing list](#), we are 5,000 and counting!

## Redis News



## Redis的优点

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。

## Redis与其他键值存储系统

- 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。

## 1. Redis环境安装配置

在本章中，您将了解和学习Redis的环境安装设置。

## 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
```

```
[2988] 07 Feb 17:09:42.488 # Unable to set the max number of files limit to 10032 (Operation not permitted), setting t
[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
[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。

## 2. 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) "activeremotehashing"
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"
```

### 3. Redis数据类型

Redis支持 5 种数据类型。

#### 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。

#### 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。

## 散列/哈希

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。

### 示例

```
redis 127.0.0.1:6379> HMSET ukey username "yiibai" password "passswd123" points 200
```

在上述示例中，散列/哈希数据类型用于存储包含用户的基本信息的用户对象。这里 `HMSET`，`HGETALL` 是Redis的命令，而 `ukey` 是键的名称。

每个散列/哈希可以存储多达  $2^{32} - 1$  个键-值对(超过 40 亿个)。

### 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。

### 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)。

## 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。

### 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。

## 可排序集合

Redis可排序集合类似于Redis集合，是不重复的字符集合。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。

### 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。

## 4. Redis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。

## 在远程服务器上运行命令

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
```

## 5. Redis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`。

## Redis键命令

下表列出了与键相关的一些基本命令。

编号	命令	描述
1	<a href="#">DEL key</a>	此命令删除一个指定键(如果存在)。
2	<a href="#">DUMP key</a>	此命令返回存储在指定键的值的序列化版本。
3	<a href="#">EXISTS key</a>	此命令检查键是否存在。
4	<a href="#">EXPIRE key seconds</a>	设置键在指定时间秒数之后到期/过期。
5	<a href="#">EXPIREAT key timestamp</a>	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	<a href="#">PEXPIRE key milliseconds</a>	设置键的到期时间(以毫秒为单位)。
7	<a href="#">PEXPIREAT key milliseconds-timestamp</a>	以Unix时间戳形式来设置键的到期时间(以毫秒为单位)。
8	<a href="#">KEYS pattern</a>	查找与指定模式匹配的所有键。
9	<a href="#">MOVE key db</a>	将键移动到另一个数据库。
10	<a href="#">PERSIST key</a>	删除指定键的过期时间，得永生。
11	<a href="#">PTTL key</a>	获取键的剩余到期时间。
12	<a href="#">RANDOMKEY</a>	从Redis返回一个随机的键。
13	<a href="#">RENAME key newkey</a>	更改键的名称。
14	<a href="#">PTTL key</a>	获取键到期的剩余时间(以毫秒为单位)。
15	<a href="#">RENAMENX key newkey</a>	如果新键不存在，重命名键。
16	<a href="#">TYPE key</a>	返回存储在键中的值的数据类型。

## 6. Redis字符串

Redis字符串命令用于管理Redis中的字符串值。以下是使用Redis字符串命令的语法。

```
redis 127.0.0.1:6379> COMMAND KEY_NAME
```

### 示例

```
redis 127.0.0.1:6379> SET mykey "redis"
OK
redis 127.0.0.1:6379> GET mykey
"redis"
```

在上面的例子中，`SET` 和 `GET` 是redis中的命令，而 `mykey` 是键的名称。

## Redis字符串命令

下表列出了一些用于在 **Redis** 中管理字符串的基本命令。

编号	命令	描述说明
1	<a href="#">SET key value</a>	此命令设置指定键的值。
2	<a href="#">GET key</a>	获取指定键的值。
3	<a href="#">GETRANGE key start end</a>	获取存储在键上的字符串的子字符串。
4	<a href="#">GETSET key value</a>	设置键的字符串值并返回其旧值。
5	<a href="#">GETBIT key offset</a>	返回在键处存储的字符串值中偏移处的位值。
6	<a href="#">MGET key1 [key2..]</a>	获取所有给定键的值
7	<a href="#">SETBIT key offset value</a>	存储在键上的字符串值中设置或清除偏移处的位

编号	命令	描述说明
8	SETEX key seconds value	使用键和到期时间来设置值
9	SETNX key value	设置键的值，仅当键不存在时
10	SETRANGE key offset value	在指定偏移处开始的键处覆盖字符串的一部分
11	STRLEN key	获取存储在键中的值的长度
12	MSET key value [key value ...]	为多个键分别设置它们的值
13	MSETNX key value [key value ...]	为多个键分别设置它们的值，仅当键不存在时
14	PSETEX key milliseconds value	设置键的值和到期时间(以毫秒为单位)
15	INCR key	将键的整数值增加 <b>1</b>
16	INCRBY key increment	将键的整数值按给定的数值增加
17	INCRBYFLOAT key increment	将键的浮点值按给定的数值增加
18	DECR key	将键的整数值减 <b>1</b>
19	DECRBY key decrement	按给定数值减少键的整数值
20	APPEND key value	将指定值附加到键

### 7. Redis哈希

Redis Hashes是字符串字段和字符串值之间的映射(类似于PHP中的数组类型)。 因此，它们是表示对象的完美数据类型。在Redis中，每个哈希(散列)可以存储多达4亿个键-值对。

#### 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)。

### 8. Redis列表

Redis列表只是字符串列表，按插入顺序排序。可以在列表的头部或尾部添加Redis列表中的元素。列表的最大长度为 **2^32 - 1** 个元素(即 **4294967295**，每个列表可存储超过40亿个元素)。

#### 示例

```
redis 127.0.0.1:6379> LPUSH mylist "redis"
(integer) 1
redis 127.0.0.1:6379> LPUSH mylist "mongodb"
(integer) 2
redis 127.0.0.1:6379> LPUSH mylist "mysql"
(integer) 3
redis 127.0.0.1:6379> LRANGE mylist 0 10
1) "mysql"
2) "mongodb"
3) "redis"
```

在上面的示例中，通过命令 **LPUSH** 将三个值插入到名称为 **mylist** 的Redis列表中。

### 8. Redis集合

Redis集合是唯一字符串的无序集合。唯一值表示集合中不允许键中有重复的数据。在Redis中设置添加，删除和测试成员的存在(恒定时间O(1)，而不考虑集合中包含的元素数量)。列表的最大长度为 **2^32 - 1** 个元素(即4294967295，每组集合超过40亿个元素)。

#### 示例

```
redis 127.0.0.1:6379> SADD myset "redis"
(integer) 1
redis 127.0.0.1:6379> SADD myset "mongodb"
(integer) 1
redis 127.0.0.1:6379> SADD myset "mysql"
(integer) 1
redis 127.0.0.1:6379> SADD myset "mysql"
(integer) 0
redis 127.0.0.1:6379> SMEMBERS "myset"
1) "mysql"
2) "mongodb"
3) "redis"
```

在上面的示例中，通过命令 **SADD** 将三个值插入到名称为 **myset** 的Redis集合中。

### 9. 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"
```

## 10. 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。

### 语法示例

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
```

## 11. 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。

[重庆小面学习](#) [集成墙面怎么样](#) [武汉二手车交易市场](#) [自助洗车机](#) [早点培训班](#) [月嫂培训班多少钱](#) [农村投资好项目](#)

```
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"
```

## 12. Redis连接

Redis中的连接命令基本上用于管理与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	<a href="#">AUTH password</a>	使用给定的密码验证服务器
2	<a href="#">ECHO message</a>	打印给定的字符串信息
3	<a href="#">PING</a>	检查服务器是否正在运行
4	<a href="#">QUIT</a>	关闭当前连接

序号	命令	说明
5	<a href="#">SELECT index</a>	更改当前连接的所选数据库

### 13. 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>
```

### 14. 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"
```

## 15. 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。

### 最大客户端数

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 <b>CLIENT SETNAME</b> 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。

## 16. 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。

## 17. 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。

## 18. 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 successfully
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 successfully");

        //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 successfully
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 successfully");
        //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 successfully
List of stored keys:: tutorial-name
List of stored keys:: tutorial-list
```

## 19. 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>。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();
$redis->connect('127.0.0.1', 6379);
echo "Connection to server successfully";
//check whether server is running or not
echo "Server is running: ".$redis->ping();
?>
```

当程序执行时，将产生以下结果。

```
Connection to server successfully
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
```

20. 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
            {
                client.Add("Test", "Learninghard", timeOut);
            }
        }
    }
}
```

```

        {
            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 hashValues = client.GetHashValues("HashId");
    foreach (var value in hashValues)
    {
        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.Learngard"); // 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
    }

    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
    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

```

```
*/
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/1sjp55U>

本文属作者原创，转载请注明出处：[易百教程](#) » [Redis快速入门](#)



广告

#### 热门教程实例

- Java教程
- Python教程
- MySQL教程
- Spring教程
- MySQL教程
- SQL Server教程
- SQL教程

#### 最新更新

- traceroute命令
- ping命令
- route命令
- ifconfig命令
- 资源下载
- lsof命令
- lostat命令
- vmstat命令

#### 站点信息

- 意见反馈
- 免责声明
- 关于我们
- 关于捐助
- [所有实例代码下载](#)

#### QQ学习群

微软技术群：227270512  
数据库群：418407075

关注微信公众号

