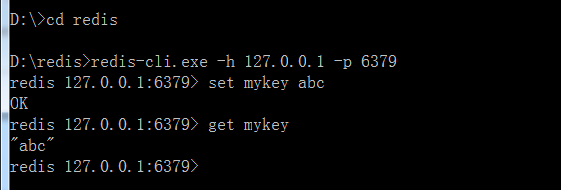
**Redis**

# Redis简介与安装

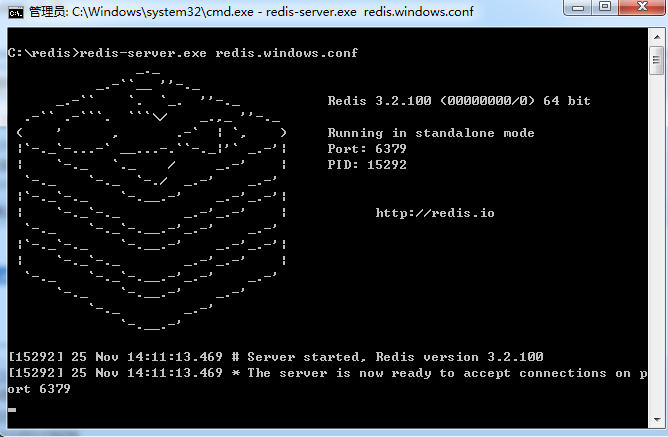
**下载地址:** [**http://github.com/dmajkic/redis/downloads**](http://github.com/dmajkic/redis/downloads)

下载到的Redis支持32bit和64bit。根据自己实际情况选择，我选择32bit。把32bit文件内容拷贝到需要安装的目录下,比如：D:\dev\redis-2.4.5。  
打开一个cmd窗口，使用cd命令切换到指定目录（D:\dev\redis-2.4.5）运行 redis-server.exe redis.conf 。  
重新打开一个cmd窗口，使用cd命令切换到指定目录（D:\dev\redis-2.4.5）运行 redis-cli.exe -h [127.0.0.1](https://www.baidu.com/s?wd=127.0.0.1&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLmHTYuWbvP1msn10LPyfL0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHbYPWfdrHb3rj6vn1RvnHb3r0) -p 6379，其中 [127.0.0.1](https://www.baidu.com/s?wd=127.0.0.1&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLmHTYuWbvP1msn10LPyfL0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHbYPWfdrHb3rj6vn1RvnHb3r0)是本地ip，6379是redis服务端的默认端口。  
这样，Redis windows环境下搭建已经完成。  
环境已经搭建好，总得测试下吧。比如：存储一个key为test，value为hello word的字符串，然后获取key值。

****

打开一个 **cmd** 窗口 使用cd命令切换目录到 **C:\redis** 运行 **redis-server.exe redis.windows.conf** 。

如果想方便的话，可以把 redis 的路径加到系统的环境变量里，这样就省得再输路径了，后面的那个 redis.windows.conf 可以省略，如果省略，会启用默认的。输入之后，会显示如下界面：

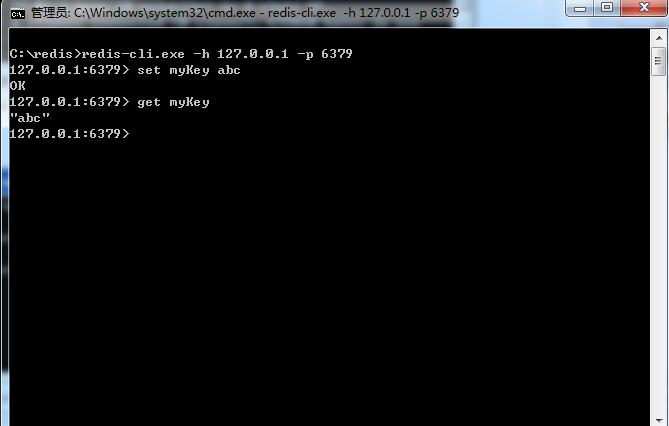


这时候另启一个cmd窗口，原来的不要关闭，不然就无法访问服务端了。

切换到redis目录下运行 **redis-cli.exe -h 127.0.0.1 -p 6379** 。

设置键值对 **set myKey abc**

取出键值对 **get myKey**



# Redis数据类型

Redis支持五种数据类型:string(字符串)、hash(哈希)、list(列表)、set(集合)以及zset(sorted set:有序集合)。

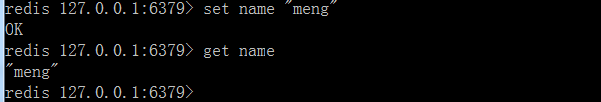
## String(字符串)

string是redis最基本的类型，你可以理解成与Memcached一模一样的类型，一个key对应一个value。

string类型是二进制安全的。意思是redis的string可以包含任何数据。比如jpg图片或者序列化的对象。

string类型是Redis最基本的数据类型，一个键最大存储5 12MB。

**实例**

****

在以上实例中我们使用了Redis的**SET**和**GET**命令。键为name,对应的值为**meng**。

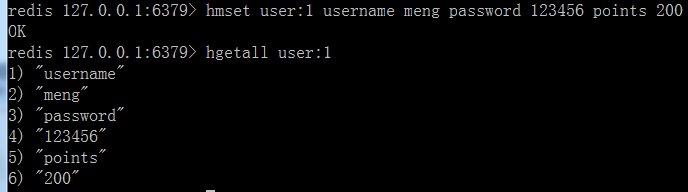
**注意:**一个键最大存储512MB

## Hash(哈希)

Redis hash是一个键值对集合

Redis hash是一个string类型的field和value的映射表，hash特别适合用于存储对象

**实例:**

****

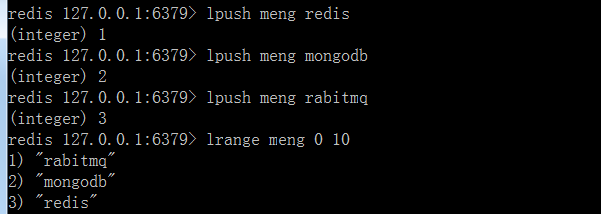
以上实例中hash数据类型存储了包含用户脚本信息的用户对象。实例中我们使用了Redis **HMSET,HGETALL**命令,**user:1**为键值

每个hash可以存储232-1键值对(40多亿)[上标Ctrl+Shift+=][下标Ctrl+=]

## List(列表)

Redis列表是简单的字符串列表，按照插入顺序排序。你可以添加一个元素到列表的头部(左边)或者尾部(右边)。

**实例**

****

列表最多可存储232-1元素(4294967295,每个列表可存储)

## Set(集合)

Redis的Set是string类型的无序集合。

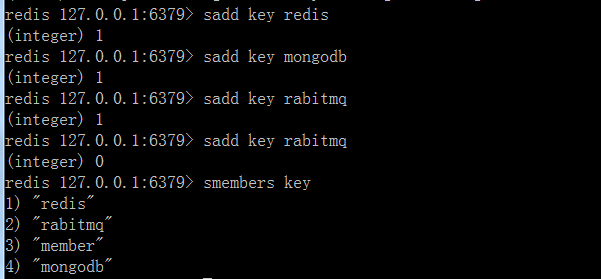
集合是通过哈希表实现的，所有添加，删除，查找的复杂度都是O(1)。

**sadd命令**

添加一个string元素到,key对应的set集合中，成功返回1，如果元素已经在集合中返回0,key对应的set不存在返回错误。



**实例**

****

**注意:** 以上实例中rabitmq添加了两次，但根据集合内元素的唯一性，第二次插入的元素将被忽略。集合中最大的成员数为232-1(4194967295,每个集合可存储40多亿个成员)。

## zset(sorted set:有序集合)

Redis zset和set一样也是string类型元素的集合，且不允许重复的成员。

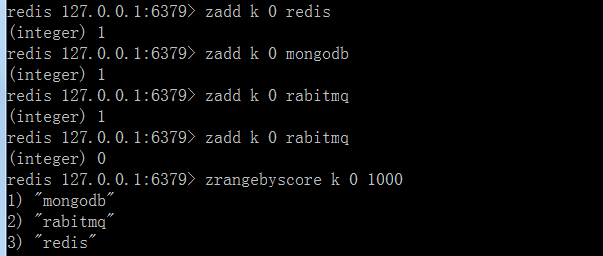
不同的是每个元素都会被关联一个double类型的分数。redis正式通过分数来为集合中的成员进行从小到大的排序。zset的成员是唯一的，但分数(score)却可以重复。

**zadd命令**

添加元素到集合，元素在集合中存在则更新对应score



**实例**

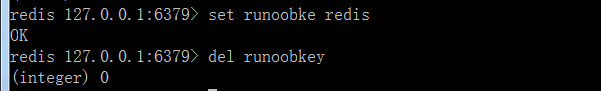


# Redis键(key)

Redis键命令用于管理redis的键

**语法**

Redis键命令的基本语法如下:



在以上实例中**DEL**一个命令，**runoobkey**是一个键。如果键被删除成功，命令执行后输出(**integer**) 1,否则将输出**(interger) 0**

**Redis keys 命令**

下标给出了与Redis键相关的基本命令

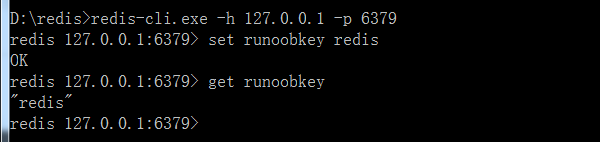
|  |  |
| --- | --- |
| 序号 | 命令及描述 |
| 1 | DEL KEY  该命令用于在key存在时删除Key |
| 2 | DUMP KEY  序列化给定key，并返回被序列化的值 |
| 3 | EXISTS KEY  检查给定key是否存在 |
| 4 | EXPIRE KEY　seconds  为给定key设置过期时间 |
| 5 | EXPIRE Key timestamp  EXPIREAT的作用和EXPIRE类似，都用于为key设置过期时间。不同在于EXPIREAT命令接受的时间参数是UNIX时间戳(unix timestamp)。 |
| 6 | PEXPIRE KEY milliseconds  设置key过期时间的时间戳(unix timestamp)以毫秒计 |
| 7 | PEXPIREAT key milliseconds-timestamp  设置key过期时间的时间戳(unix timestamp)以毫秒计 |
| 8 | Keys pattern  查找所有符合给定模式(pattern)的key |
| 9 | MOVE key db  将当前数据库的key移动到给定的数据库db当中 |
| 10 | PERSIST KEY  移除key的过期时间,key将持久保持 |
| 11 | PTTL key  以毫秒为单位返回key的剩余的过期时间 |
| 12 | TTL key  以秒为单位，返回给定key的剩余时间(TTL,time to live)。 |
| 13 | RANDOMKEY  从当前数据库中随机返回一个key |
| 14 | RENAME KEY newkey  修改key的名称 |
| 15 | Renameex key newkey  仅当newkey不存在时，将key改名为newkey |
| 16 | TYPE key  返回key所存储的值的类型 |

# Redis字符串(String)

Redis字符串命令用于在Redis管理字符串值。使用Redis字符串命令的语法如下所示:

**语法**

**redis 127.0.0.1:6379 > command key\_name**

****

在以上实例中我们使用**SET**和**GET**命令，键为**runoobkey**。

**Redis字符串命令**

下标列出了常用的redis字符串命令:

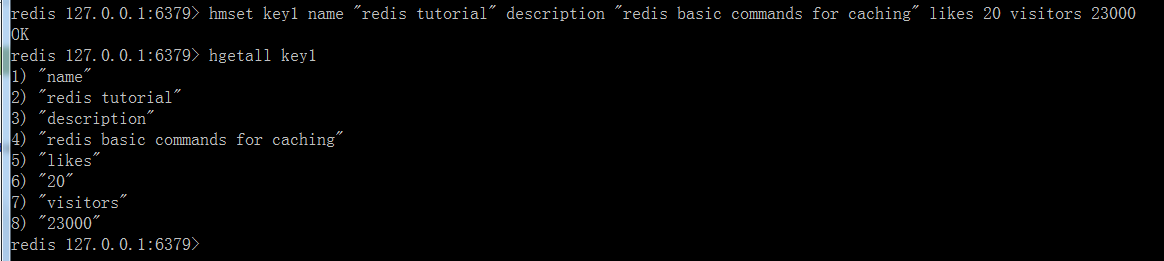
|  |  |
| --- | --- |
| 序号 | 命令及描述 |
| 1 | **SET key value**  设置指定key的值 |
| 2 | **GET key**  获取指定key的值 |
| 3 | **GETRANGE key start end**  返回key中字符串值的子字符 |
| 4 | **GETSET key value**  将给定key的值设为value，并返回Key的旧值(old value)。 |
| 5 | **GETBIT key offset**  对key所储存的字符串值，获取指定偏移量上的位(bit)。 |
| 6 | **MGET key1 [key2…]**  获取所有(一个或多个)给定Key的值 |
| 7 | **SETBIT key seconds value**  对key所储存的字符串值，设置或清除指定偏移量上的位(bit)。 |
| 8 | **SETEX key seconds value**  将值value关联到Key,并将Key的过期时间设为seconds(以秒为单位) |
| 9 | **SETNX key value**  只有在Key不存在时设置Key的值 |
| 10 | **SETRANGE key offset value**  用value参数覆写给定key所储存的字符串值，从偏移量offset开始 |
| 11 | **SERLEN KEY**  返回Key所储存的字符串值的长度 |
| 12 | **MSETNX key value [key value]**  同时设置一个或多个key-value对。 |
| 13 | **MSETNX key value [key value…]**  同时设置一个或多个key-value对，当且仅当所有给定key都不存在。 |
| 14 | **PSETEX key milliseconds value**  这个命令和setex命令相似，但它以毫秒为单位设置key的生存时间，而不是像setex命令那样，以秒为单位。 |
| 15 | **Incr key**  将key中储存的数字值增一。 |
| 16 | **Incrby key increment**  将key所储存的值加上给定的增量值(increment) |
| 17 | **Incrbyfloat key increment**  将Key所储存的值加上给定的浮点增量值(increment) |
| 18 | **Decr key**  将Key中储存的数字值减一。 |
| 19 | **Decrby key decrement**  Key所储存的值减去给定的减量值(decrement) |
| 20 | **Append key value**  如果key已经存在并且是一个字符串，append命令将value追加到Key原来的值的末尾。 |

# Redis哈希(Hash)

Redis hash 是一个String类型的field和value的映射表，hash特别适合用于存储对象。

Redis中每个hash可以存储232-1键值对(40多亿)。

**实例**



Redis hash命令

下标列出了redis hash基本的相关命令

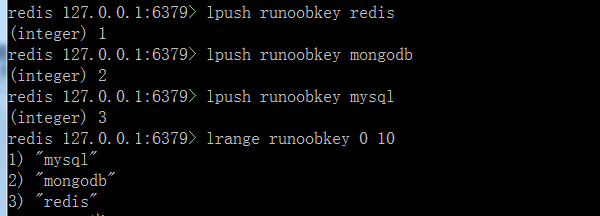


# Redis列表(List)

Redis列表是简单的字符串列表，按照插入顺序排序。你可以添加一个元素岛列表的头部(左边)或者尾部(右边)。

一个列表最多可以包含232-1个元素(4294967295,每个列表超过40亿个元素)。

**实例**

****

在以上实例我们使用**LPUSH**将三个值插入了名为**runoobkey**的列表当中。

**Redis列表命令**

下标列出了列表相关的基本命令:



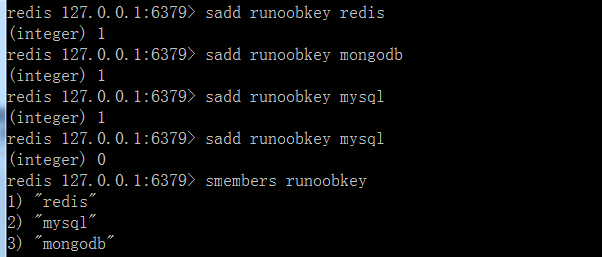
# Redis集合(Set)

Redis的Set是string类型的无序集合。集合成员是唯一的，这就意味这集合不能出现重复的数据。

Redis中集合是通过哈希表实现的，所以添加，删除，查找的复杂度都是O(1)。

集合中最大的成员数为232-1（4294967295，每个集合可存储40多亿个成员）。

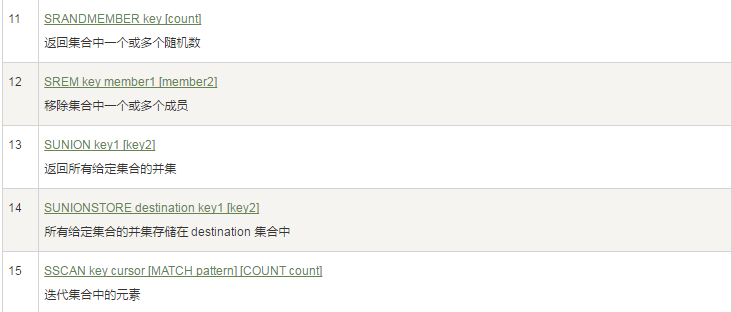
**实例**

****

在以上实例中我们通过**SADD**命令向名为**runoobkey**的集合插入的三个元素。

**Redis集合命令**

下标列出了Redis集合基本命令



# Redis有序集合(sorted set)

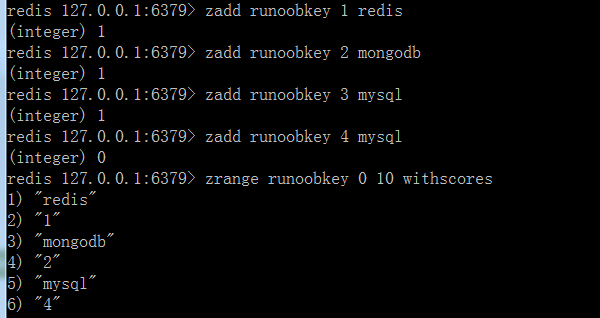
Redis有序集合和集合一样也是string类型元素的集合，且不允许重复的成员。

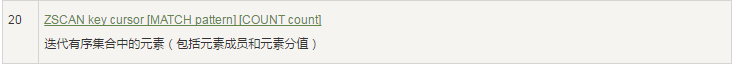
不同的是每个元素都会关联一个double类型的分数。redis正是通过分数来为集合中的成员进行从小到大的排序。

有序集合的成员是唯一的,但分数(score)却可以重复。

集合是通过哈希表实现的，所以添加，删除，查找的复杂度都是O(1)。 集合中最大的成员数为 232 - 1 (4294967295, 每个集合可存储40多亿个成员)。

**实例**

****

****

# Redis HyperLogLog

Redis 在 2.8.9 版本添加了 HyperLogLog 结构。

Redis HyperLogLog 是用来做基数统计的算法，HyperLogLog 的优点是，在输入元素的数量或者体积非常非常大时，计算基数所需的空间总是固定 的、并且是很小的。

在 Redis 里面，每个 HyperLogLog 键只需要花费 12 KB 内存，就可以计算接近 2^64 个不同元素的基 数。这和计算基数时，元素越多耗费内存就越多的集合形成鲜明对比。

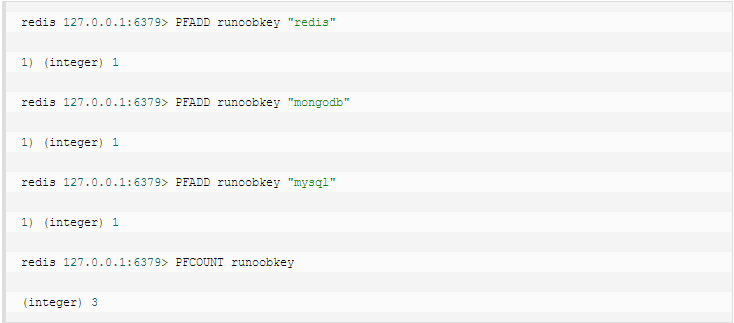
但是，因为 HyperLogLog 只会根据输入元素来计算基数，而不会储存输入元素本身，所以 HyperLogLog 不能像集合那样，返回输入的各个元素。

**什么是基数？**

比如数据集{1,3,5,7,5,7,8},那么这个数据集的基数集为{1,3,5,7,8}，基数(不重复元素)为5。基数估计就是在误差可接收的范围内，快速计算基数。

**实例**

以下实例演示了HyperLogLog的工作过程:



# Redis发布订阅

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 客户端可以订阅任意数量的频道。

下图展示了频道 channel1 ， 以及订阅这个频道的三个客户端 —— client2 、 client5 和 client1 之间的关系：



当有新消息通过 PUBLISH 命令发送给频道 channel1 时， 这个消息就会被发送给订阅它的三个客户端：





# Redis事务

Redis 事务可以一次执行多个命令， 并且带有以下两个重要的保证：

* 事务是一个单独的隔离操作：事务中的所有命令都会序列化、按顺序地执行。事务在执行的过程中，不会被其他客户端发送来的命令请求所打断。
* 事务是一个原子操作：事务中的命令要么全部被执行，要么全部都不执行。

一个事务从开始到执行会经历以下三个阶段：

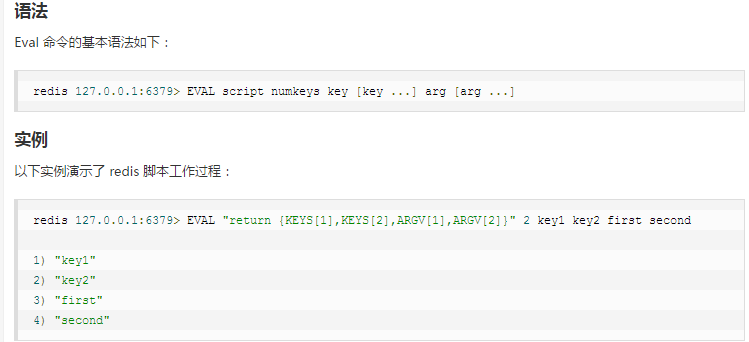
* 开始事务。
* 命令入队。
* 执行事务。

实例



# Redis脚本

Redis 脚本使用 Lua 解释器来执行脚本。 Reids 2.6 版本通过内嵌支持 Lua 环境。执行脚本的常用命令为 **EVAL**。



# Redis连接

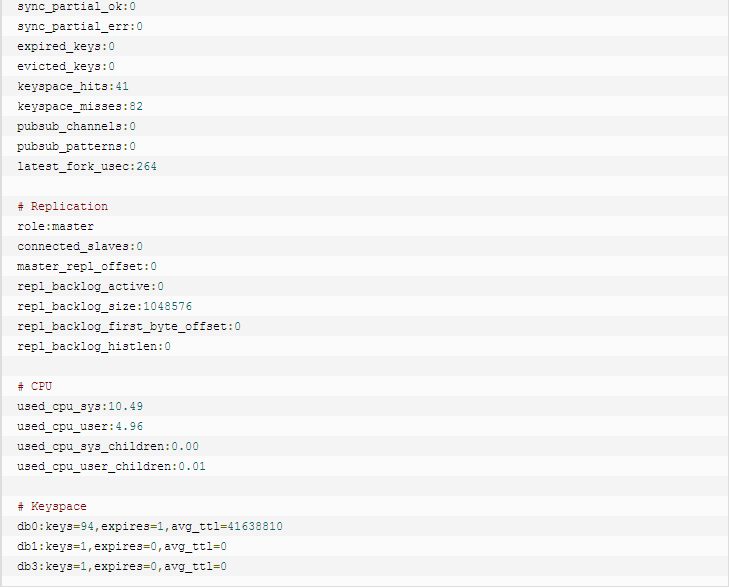
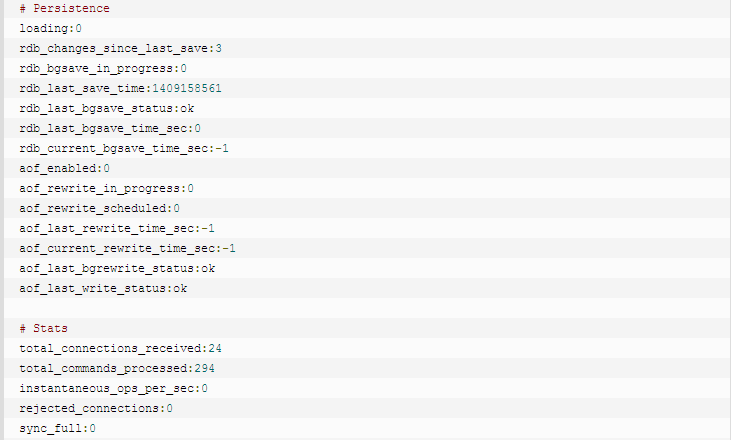
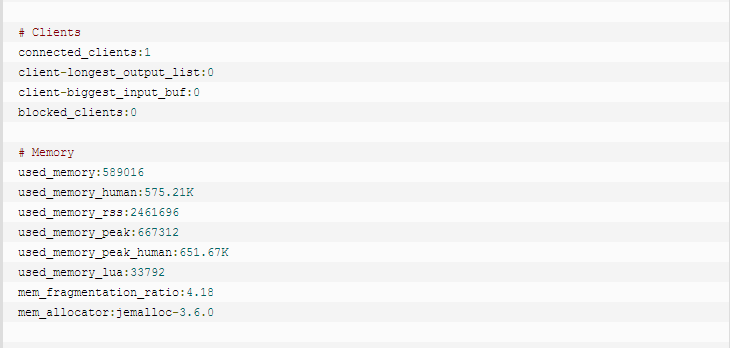
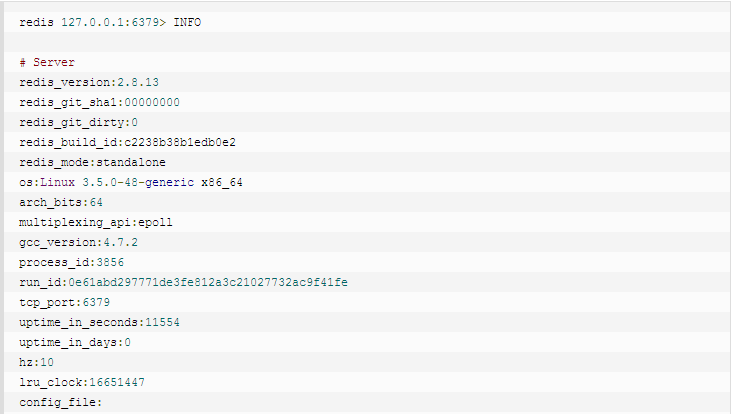


# Redis服务器

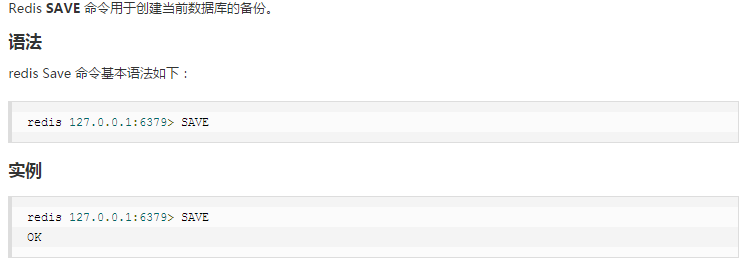
Redis 服务器命令主要是用于管理 redis 服务。

**实例**

以下实例演示了如何获取 redis 服务器的统计信息：

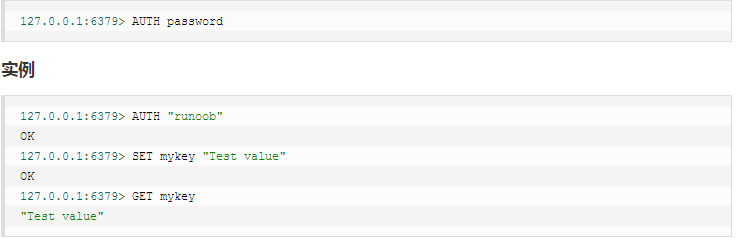


# Redis数据备份与回复

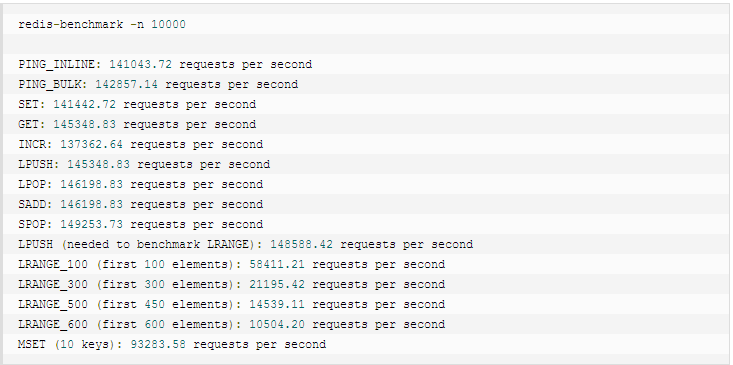
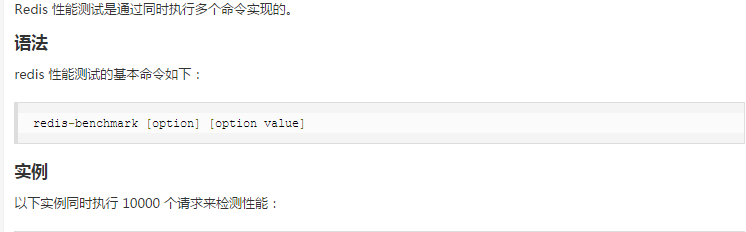


# Redis安全

我们可以通过 redis 的配置文件设置密码参数，这样客户端连接到 redis 服务就需要密码验证，这样可以让你的 redis 服务更安全。



# Redis性能测试



# Redis客户端连接

Redis 通过监听一个 TCP 端口或者 Unix socket 的方式来接收来自客户端的连接，当一个连接建立后，Redis 内部会进行以下一些操作：

* 首先，客户端 socket 会被设置为非阻塞模式，因为 Redis 在网络事件处理上采用的是非阻塞多路复用模型。
* 然后为这个 socket 设置 TCP\_NODELAY 属性，禁用 Nagle 算法
* 然后创建一个可读的文件事件用于监听这个客户端 socket 的数据发送

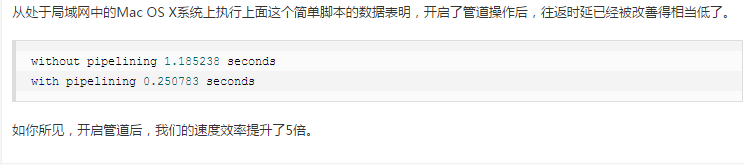
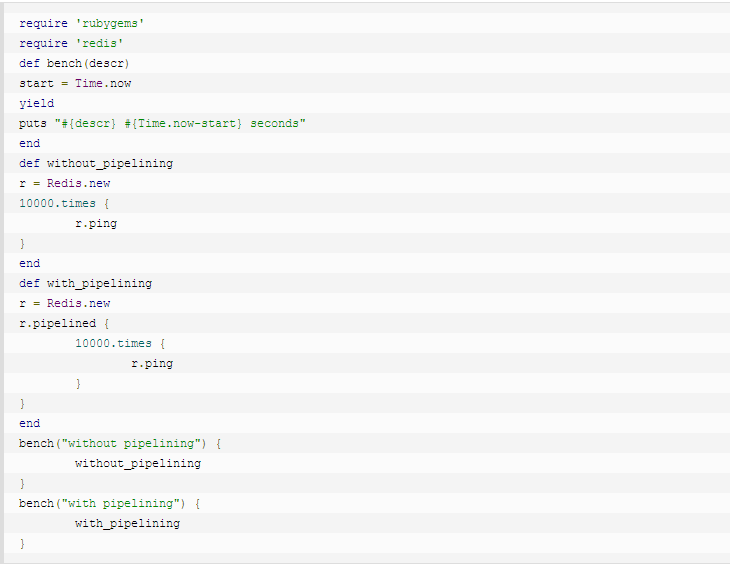
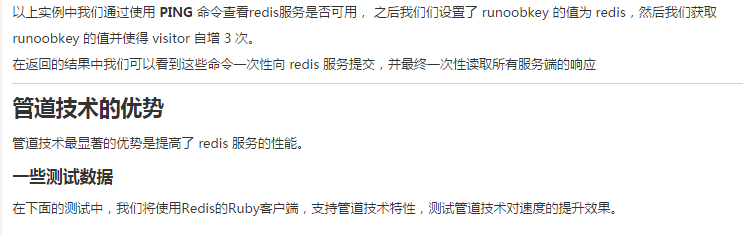


# Redis管道技术

Redis是一种基于客户端-服务端模型以及请求/响应协议的TCP服务。这意味着通常情况下一个请求会遵循以下步骤：

* 客户端向服务端发送一个查询请求，并监听Socket返回，通常是以阻塞模式，等待服务端响应。
* 服务端处理命令，并将结果返回给客户端。





# Redis分区

分区是分割数据到多个Redis实例的处理过程，因此每个实例只保存key的一个子集。

## 分区的优势

* 通过利用多台计算机内存的和值，允许我们构造更大的数据库。
* 通过多核和多台计算机，允许我们扩展计算能力；通过多台计算机和网络适配器，允许我们扩展网络带宽。

分区的不足

redis的一些特性在分区方面表现的不是很好：

* 涉及多个key的操作通常是不被支持的。举例来说，当两个set映射到不同的redis实例上时，你就不能对这两个set执行交集操作。
* 涉及多个key的redis事务不能使用。
* 当使用分区时，数据处理较为复杂，比如你需要处理多个rdb/aof文件，并且从多个实例和主机备份持久化文件。
* 增加或删除容量也比较复杂。redis集群大多数支持在运行时增加、删除节点的透明数据平衡的能力，但是类似于客户端分区、代理等其他系统则不支持这项特性。然而，一种叫做presharding的技术对此是有帮助的。

## 分区类型

Redis 有两种类型分区。 假设有4个Redis实例 R0，R1，R2，R3，和类似user:1，user:2这样的表示用户的多个key，对既定的key有多种不同方式来选择这个key存放在哪个实例中。也就是说，有不同的系统来映射某个key到某个Redis服务。

## 范围分区

最简单的分区方式是按范围分区，就是映射一定范围的对象到特定的Redis实例。

比如，ID从0到10000的用户会保存到实例R0，ID从10001到 20000的用户会保存到R1，以此类推。

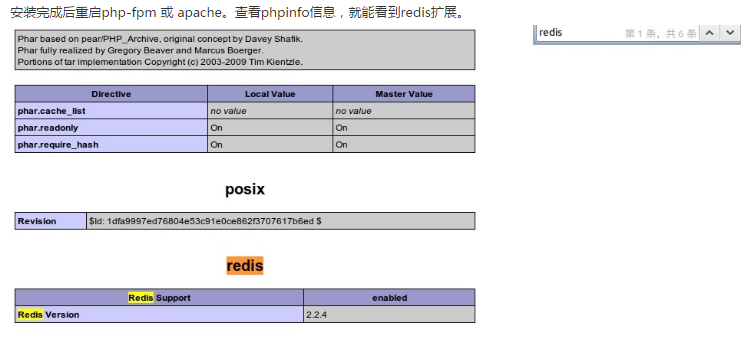
这种方式是可行的，并且在实际中使用，不足就是要有一个区间范围到实例的映射表。这个表要被管理，同时还需要各 种对象的映射表，通常对Redis来说并非是好的方法。

## 哈希分区

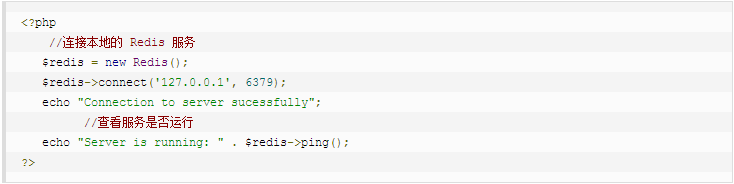
另外一种分区方法是hash分区。这对任何key都适用，也无需是object\_name:这种形式，像下面描述的一样简单：

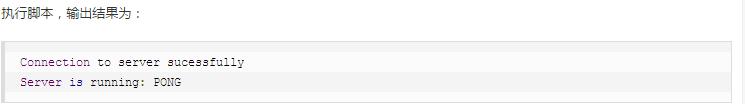
* 用一个hash函数将key转换为一个数字，比如使用crc32 hash函数。对key foobar执行crc32(foobar)会输出类似93024922的整数。
* 对这个整数取模，将其转化为0-3之间的数字，就可以将这个整数映射到4个Redis实例中的一个了。93024922 % 4 = 2，就是说key foobar应该被存到R2实例中。注意：取模操作是取除的余数，通常在多种编程语言中用%操作符实现。

# PHP使用Redis



## 连接到redis服务





## Redis PHP String(字符串)实例



## Redis PHP List(列表)实例



## Redis PHP Keys 实例

