# Redis开发与运维

# 初始Redis

## 1.2 Redis特性

Redis全称：Remote Dictionary Server

8个重要特性

1. 速度快：所有数据都是存放在内存中的、是用C语言实现的、使用了单线程架构
2. 基于键值对的数据结构服务器：字符串、哈希、列表、集合、有序集合，在字符串基础上演变出了位图(Bitmaps)和HyperLogLog的数据结构，Redis3.2版本中加入GEO(地理信息定位)的功能。
3. 丰富的功能：键过期功能、发布订阅功能、支持Lua脚本功能、简单的事务功能、提供了流水线(Pipeline)功能
4. 简单稳定：代码5万行，相对其他产品较少，使用单线程模型。模型简单
5. 客户端语言多：Java，PHP，Python、C、C++、Nodejs
6. 持久化：两种持久化方式：RDB和AOF。
7. 主从复制：
8. 高可用和分布式：Redis2.8高可用实现Redis Sentinel。Redis3.0提供了分布式实现Redis Cluster

## 1.3 Redis使用场景

### 1.3.1 可以做什么

1. 缓存
2. 排行榜系统：列表和有序集合数据结构
3. 计数器应用：视频播放数，颠勺网站浏览数
4. 社交网络：赞/踩、粉丝、共同好友/喜好、推送、下拉刷新等
5. 消息对列系统

### 1.3.1 不可以做什么

1. 大数据规模不适合：每天有几亿的用户行为数据
2. 冷数据不适合：不经常使用的数据

# API的理解和使用

## 2.1 预备

### 2.1.1 全局命令

1. 查看所有键

keys \*

1. 键总数：dbsize

dbsize是直接获取Redis内置的键中暑变量，时间复杂度是O(1)，keys遍历所有键，时间复杂度是O(n)，当Redis保存大量键时，线上环境禁止使用

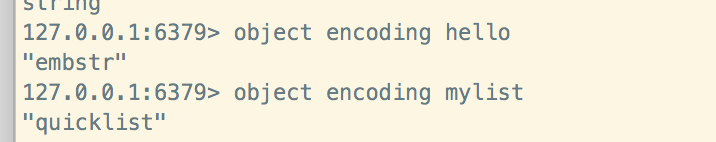
1. 检查键是否存在 exists key
2. 删除键：del key [key....]
3. 键过期：expire key seconds

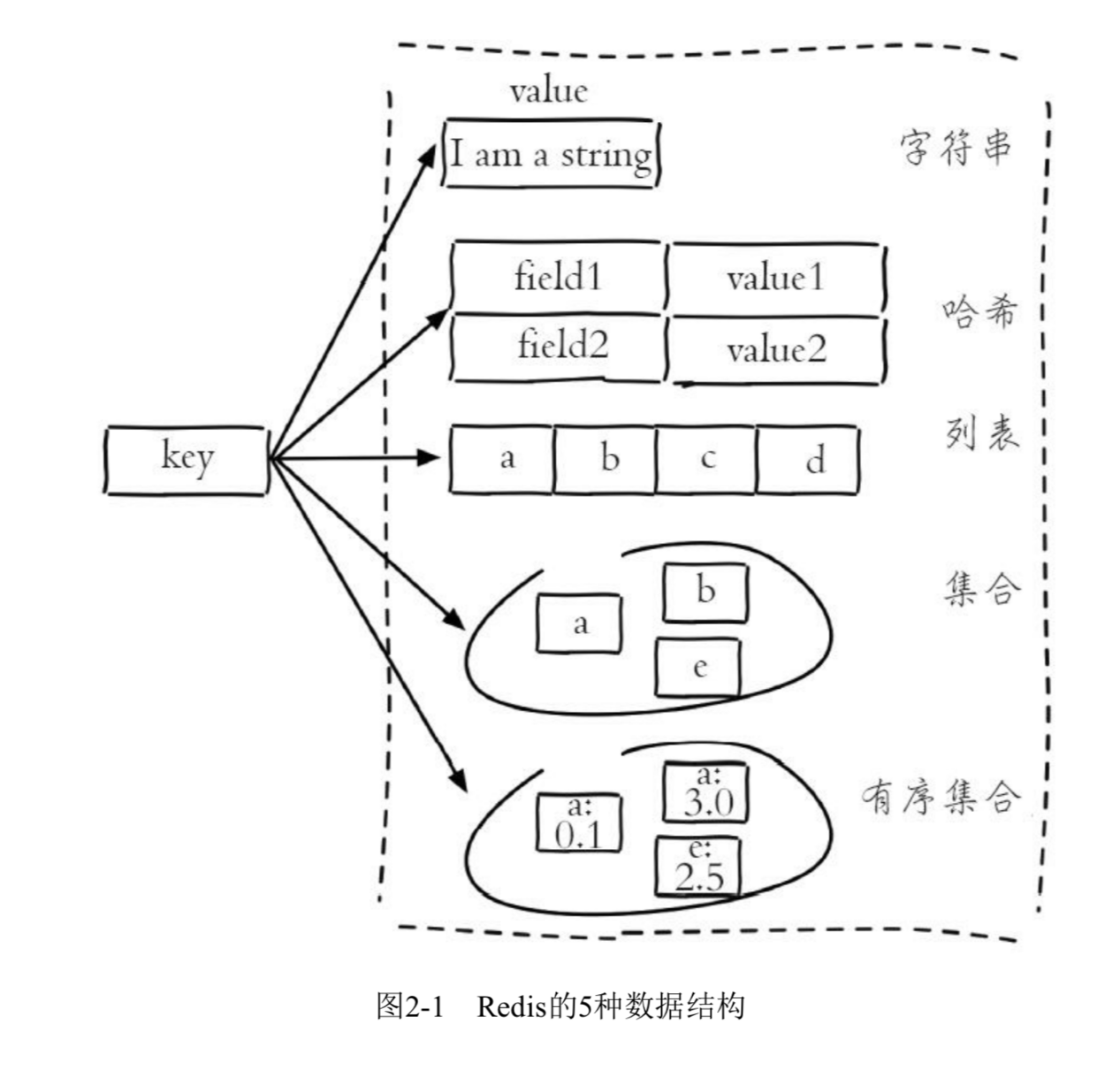
tt命令返回键的剩余过期时间，大于等于0的整数：剩余的过期时间，-1：没设置过期时间，-2：键不存在

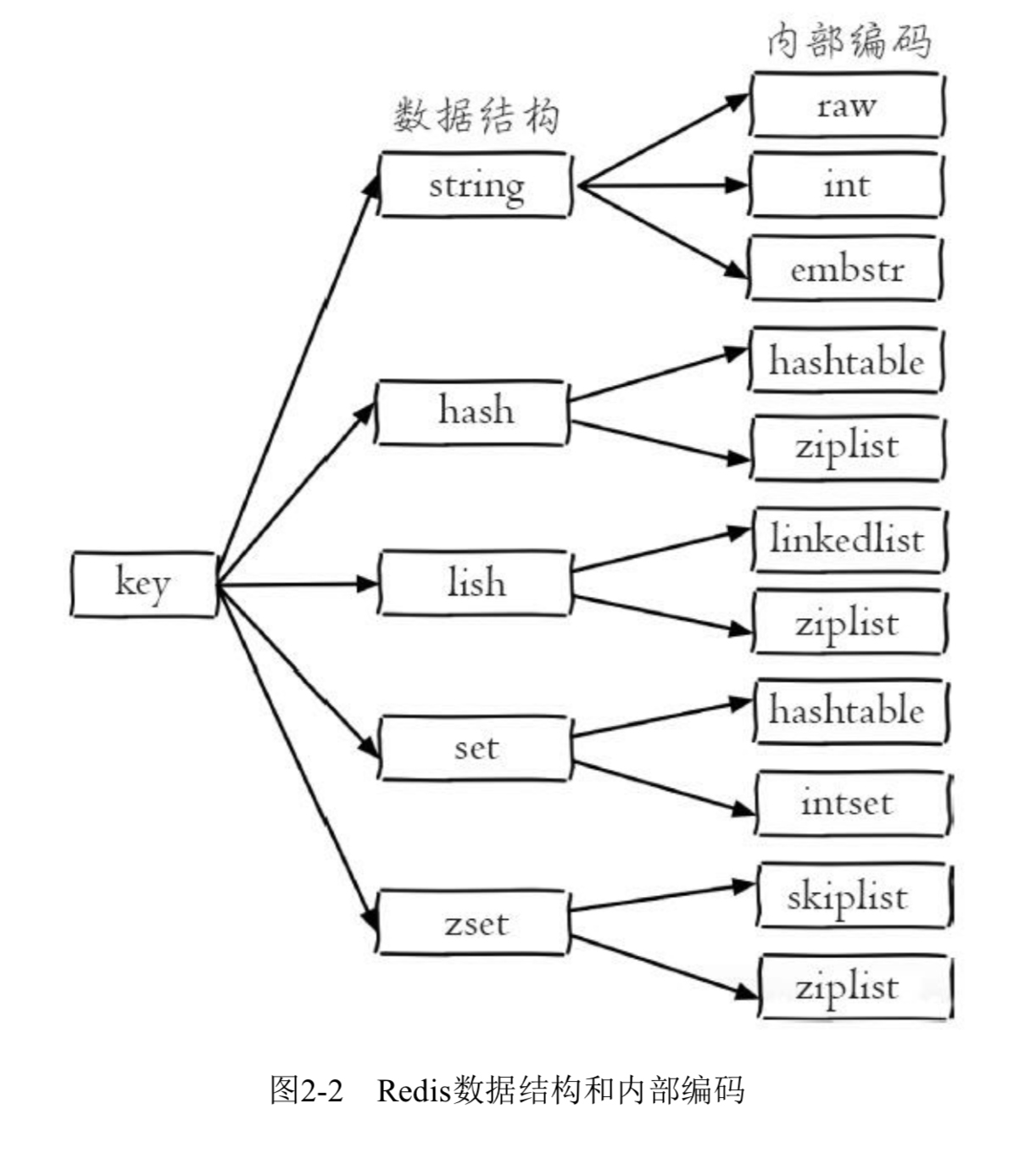
1. 键的数据结构类型：type keys

### 2.1.2 数据结构和内部编码

可以使用object encoding key 查看内部编码







### 2.1.3 单线程架构

Redis使用单线程架构和IO多路复用模型来实现高新能的内存数据库服务。

多个客户端的多条命令到Redis服务，命令会被放到一个对列中，然后单线程执行。

为什么单线程还能这么块？

1. 纯内存访问，Redis将所有数据放在内存中，内存响应时间约韦100纳秒。
2. 非阻塞IO，Redis使用epoll作为IO多路复用技术的实现。
3. 单线程避免了线程的切花和静态产生的消耗

单线程的问题？

对于每个命令的执行时间是由要求的。如果某个命令执行过长，会造成其他命令的阻塞。

## 2.2 字符串

字符串的值最大不能超过512M

### 2.2.1 命令

1. **常用命令**
2. 设置值：set key value [ex seconds] [px millseconds] [nx|xx]

ex seconds: 为键设置妙计过期时间

px milliseconds: 为键设置毫秒级过去时间

nx :键必须不存在，才可以设置成功，用于添加

xx: 与nx想法，键必须存在，才可以设置成功，用于更新

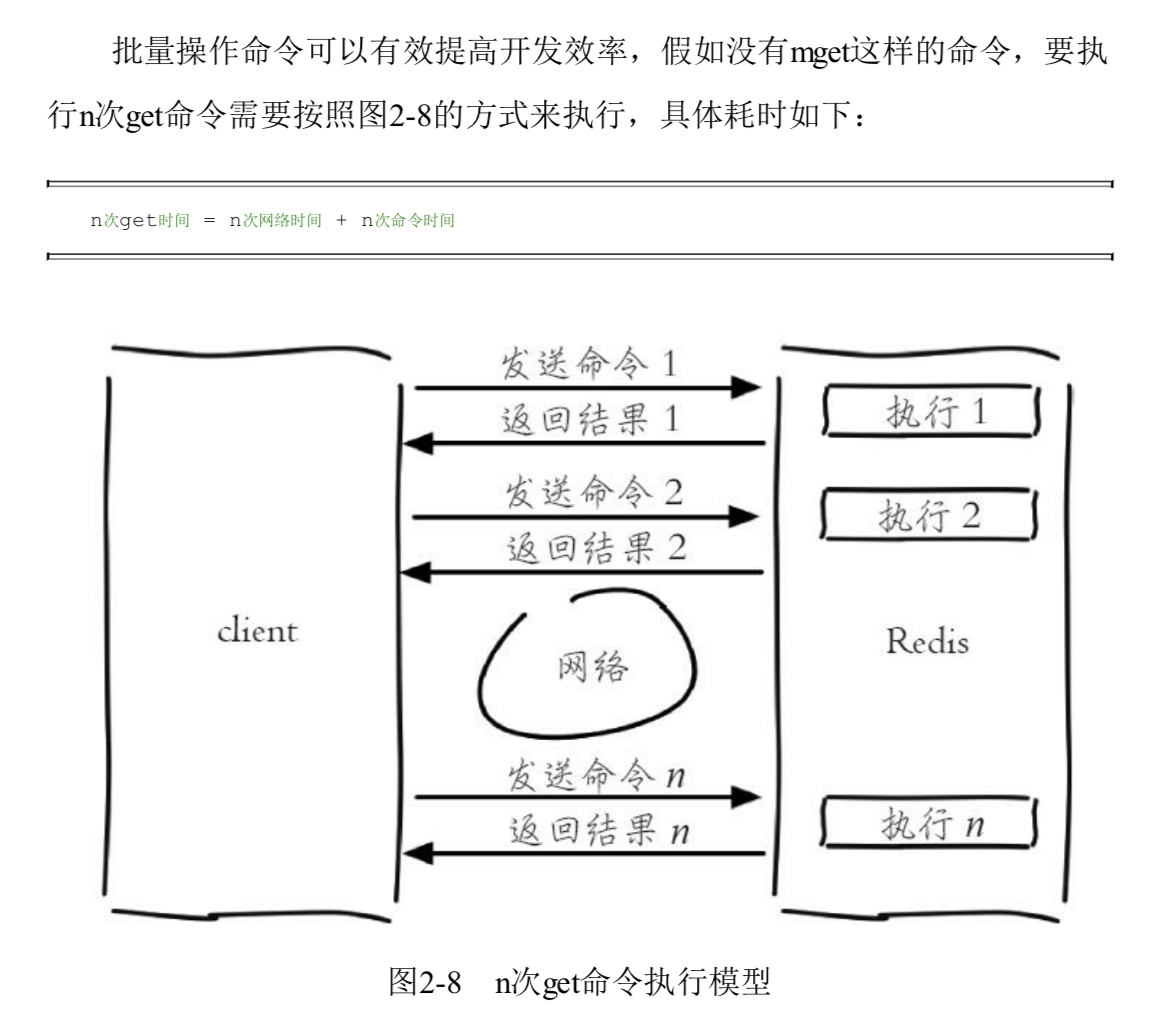
setex key seconds value

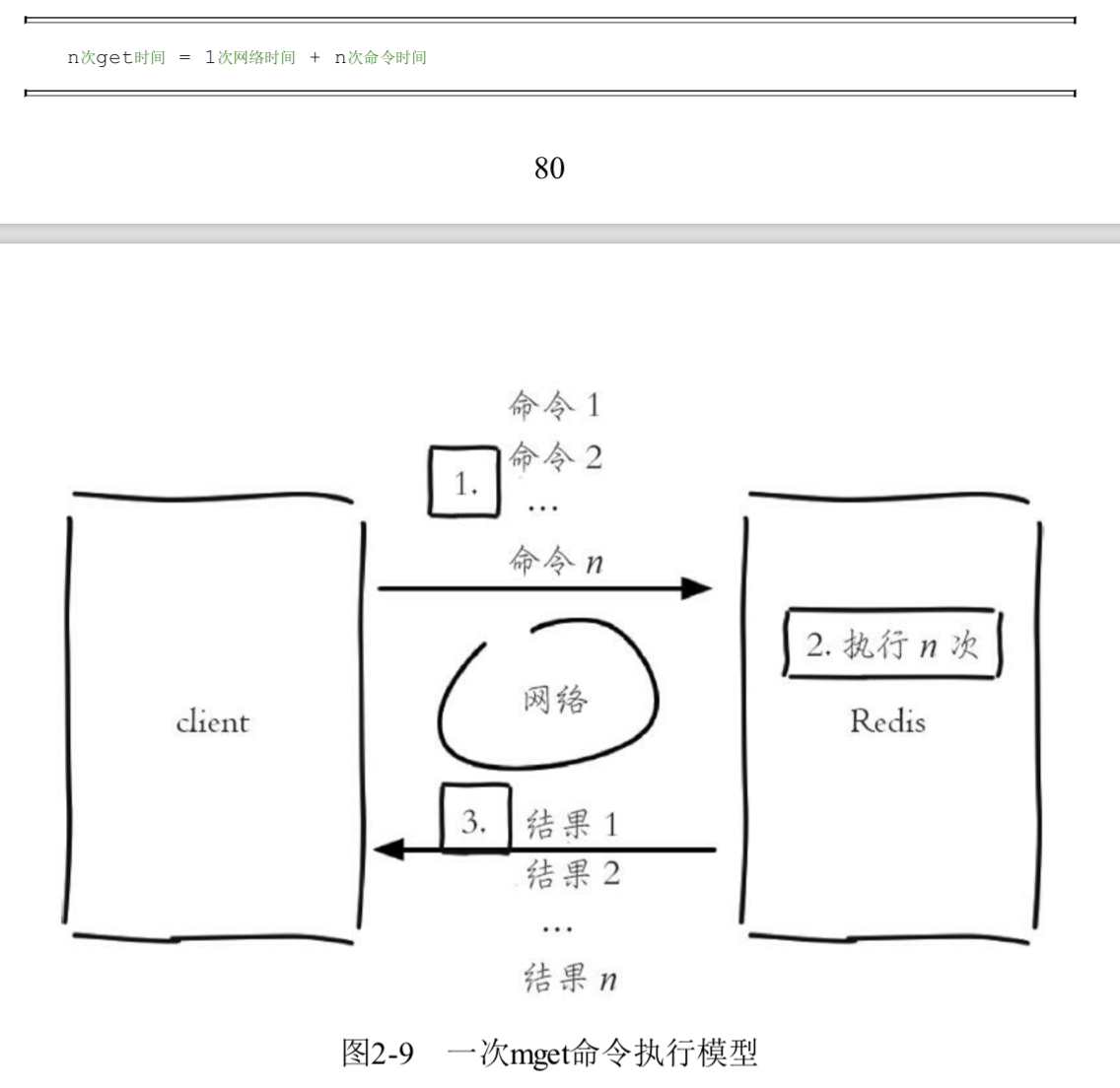
setnx key value

以上两个命令和ex和nx选项是一样的

setnx 可以作为分布式锁的一种实现方案

1. 获取值：get key
2. 批量设置值：mset key value [key value ....]
3. 批量获取值：mget key [key ....]





1. 计数：incr key

值不是整数：返回错误

值是整数：返回自增后的结果

键不存子啊，按照值为0自增，返回结果是1

Redis提供了decr(自减)、incrby(自增指定数字)、decrby(自减指定数字)、incrbyfloat(自增浮点数)

1. **不常用命令**
2. 追加值：append key value
3. 字符串长度：strlen key
4. 设置并返回值：getset key value :返回的是上一次的值
5. 设置指定位置的字符：setrange key offset value，
6. 获取部分字符串：getrange key start end



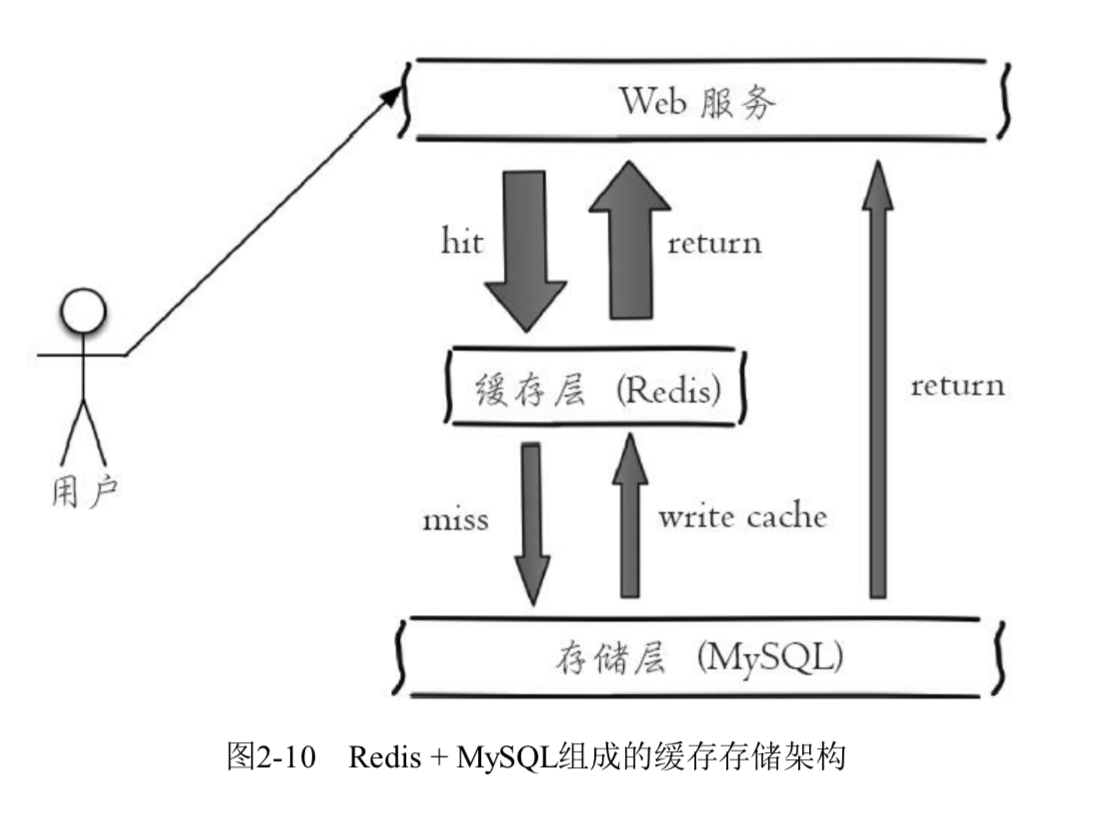
### 2.2.2 内部编码(3种)

3种：int：8个字节的长整型；embstr：小于等于39个字节的字符串，raw：大于39个字节的字符串。

object encoding key

### 2.2.3 典型使用场景

1. **缓存功能**



redis缓存键的命名方式：业务名:对象名:id:[属性]（也可以不用分号）

1. **计数**

共享session

1. **共享session**
2. **限速**

## 2.3 哈希

### 2.3.1 命令

1设置值：hset key field value

2 获取值：hget key field

3 删除field hdel key field [field ...]

4 计算field个数：hlen key

5 批量设置或获取field-value：hmget key field [field。。。] 、hmset key field value [field value ...]

6 判断field是否存在:hexists key field

7 获取所有field：hkeys key

8 获取所有value：hvals key

9 获取所有的field-value: hgetall key

10 hincrby hincrbyfloat：hincryby key field hincrbyfloat key field

就像incryby和incrbyfloat命令一样。

11 计算value的字符串长度 (需要redis3.2以上):hstrlen key field



### 2.3.2 内部编码（2种）

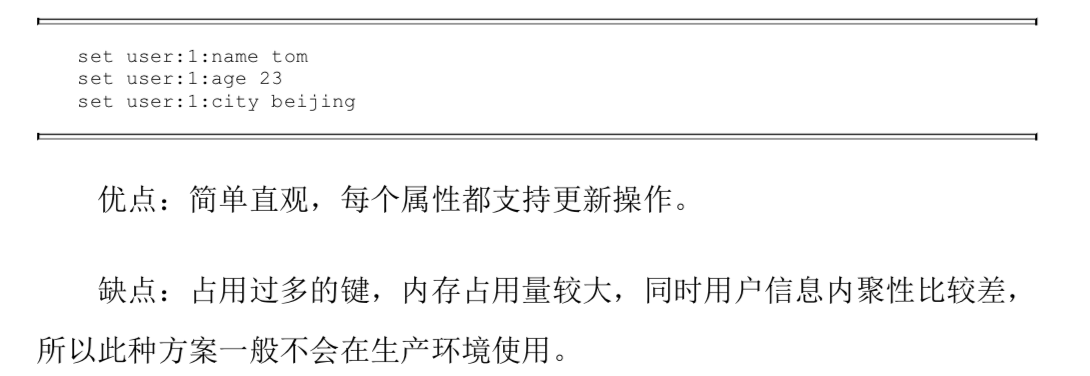
ziplist(压缩列表)：当哈希类型元素个数小鱼hash-max-ziplist-entries配置（默认512）同时所有值都小于hash-max-ziplist-value配置(默认64字节)时，Redis会使用ziplist作为哈希的内部实现。ziplist跟家紧凑。

hashtable(哈希表)：当哈希类型无法满足ziplist的条件时，Redis会使用hashtable作为哈希的内部实现，hashtable的读写时间复杂度O(1)；

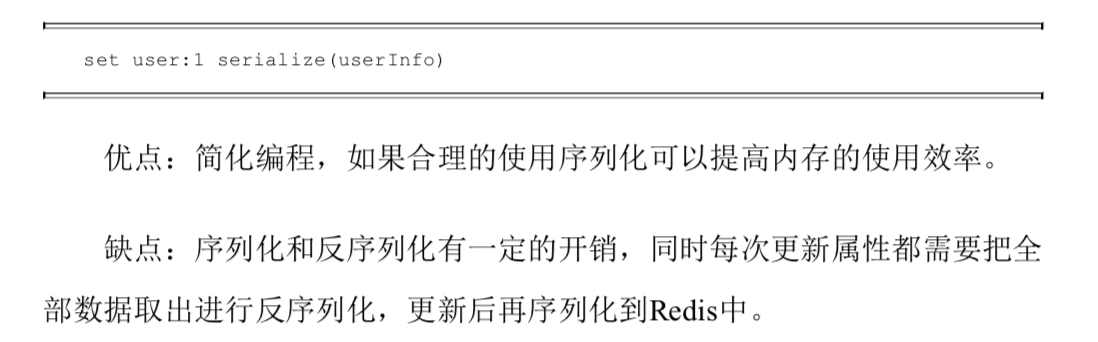
### 2.3.3 使用场景

三种方法缓存用户信息

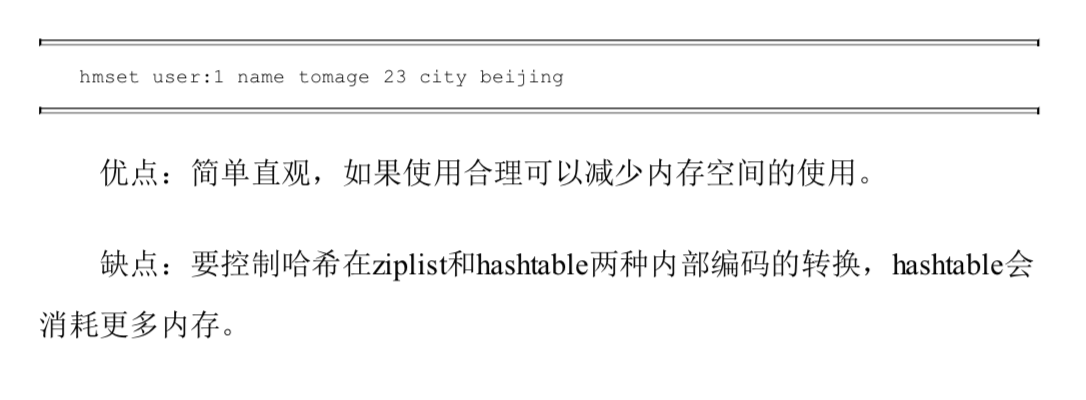
1. 原生字符串类型：每个属性一个键



1. 序列化字符串类型：将用户信息序列化后用一个键保存。



1. 哈希类型：每个用户属性使用一堆field-value，但是只用衣蛾键保存‘



## 2.4 列表

列表类型使用来存储多个有序的字符串，列表最多可以存储2^32 -1个元素；可以对列表的两端插入(push)和弹出(pop)，还可以获取指定范围的元素裂变、互殴指定索引下标的元素等。

列表特点：列表的元素是有序的，列表的元素可以重复。

### 2.4.1命令



1. 添加操作
2. 从右边插入元素：rpush key value [value ....]
3. 从左边插入元素：lpush key value [value....]
4. 向某个元素前后者后插入元素：linsert key before|after pivot value
5. 查找
6. 获取指定范围内的元素列表

lrange key start end

索引下边从做导游分别是0到N-1 ,但是从右到做分别是-1到-N

lrange中的end选项包含了自身

1. 获取列表指定索引下标的元素：lindex key index
2. 获取列表长度：llen key
3. 删除
4. 从列表左侧弹出元素：lpop key
5. 从右侧弹出：rpop key
6. 删除指定元素：lrem key count value

count > 0 从左到右，删除最多Count个元素

count < 0 从右到左，删除最多count绝对值个元素。

count = 0 ,删除所有。

1. 按照索引范围修剪列表：ltrim key start end
2. 修改

lset key index newValue

1. 阻塞操作

阻塞式弹出如下：blpop key [key ...] timeout 或 brpop key [key...] timeout

timeout:阻塞的时间（单位秒）



### 2.4.2 内部编码(2-3种)

ziplist(压缩列表)：元素个数小于list-max-ziplist-entries配置（默认512个），同时列表每个元素的值都小于list-max-ziplist-value配置时（默认64字节）。

linkedlist（链表）：当列表类型无法满足ziplist的条件时，Redis会使用linkedlist作为列表的内部实现。

Redis3.2版本提供了quicklist内部编码，简单地说它是以一个ziplist为节点linkedlist。

### 2.4.3 使用场景

1. 消息对列

lpush+brpop命令组合即可实现阻塞对列，生产者客户端使用lrpush从对列左侧插入元素，多个消费者客户端使用brpop命令阻塞式“抢”列表尾部的元素。

1. 文章列表

lpush+lpop=Stack（栈）

lpush+rpop=Queue(对列)

lpush+ltrim=Capped Collection （有限集合）

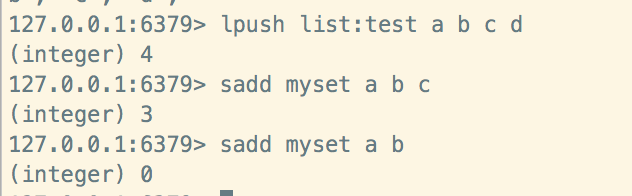
lpush+brpop=Message Queue（消息对列）

## 2.5 集合

集合中元素不能重复，无序。

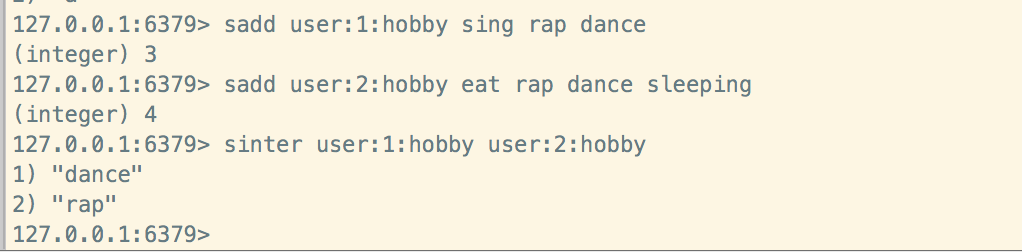
### 2.5.1 命令

1. 集合内操作
2. 添加元素：sadd key element [ ...]



1. 删除元素：srem key element [element ...]
2. 计算元素个数：scard key：四件负载读O(1)
3. 判断元素是否在集合中：sismember key element
4. 随机从集合返回指定个数元素：srandmember key [count]:count不写默认是1
5. 从集合随机弹出元素：spop key ：随机弹出一个元素
6. 获取所有元素：smembers key
7. 集合间操作
8. 求多个集合的交集

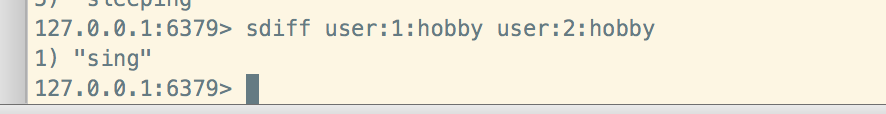
sinter key [key。。。]



1. 求多个集合的并集

sunion key [key....]

1. 求多个集合的差集:sdff key [key...] 以第一个key的集合的非交集部分的元素

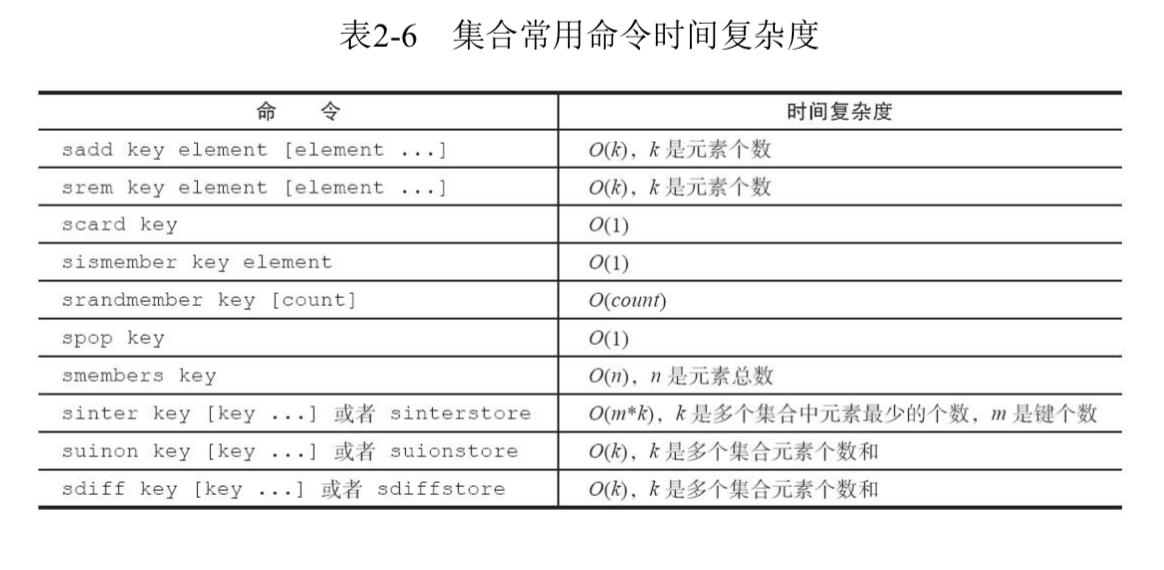


1. 将交集、并集、差集的结果保存

sinterstore destintion key [key ...]

suionstore destintion key [key ...]

sdiffstore destintion key [key ...]



### 2.5.2 内部编码(2种)

intset(整数集合)：当集合中的元素都是整数且元素个数小鱼set-max-intset-entries配置（默认512个）时，Redis会选用intset来作为集合的内部实现

hashtable(哈希表)：当集合类型无法满足intset的条件时，会使用hashtable

### 2.5.3 使用场景

比较典型的使用场景室标签(tag)。

sadd=Tagging(标签)

spop/srandmember = Random item（生成随机数，比如抽象）

sadd+sinter=Social Graph(社交需求)

## 2.6 有序集合

### 2.6.1 命令

1. 集合内
2. 添加成员：zadd key member [score member...]

Redis3.2 为zadd命令添加了nx、xx、ch、incr四个选项

nx：member必须不存在，才可以设置成功，用于添加

xx：member必须存在，才可以设置成功，用于更新

ch：返回此操作后，有序集合元素和分数发生变化的个数

incr：对score做增加，相当于后面介绍的zincrby

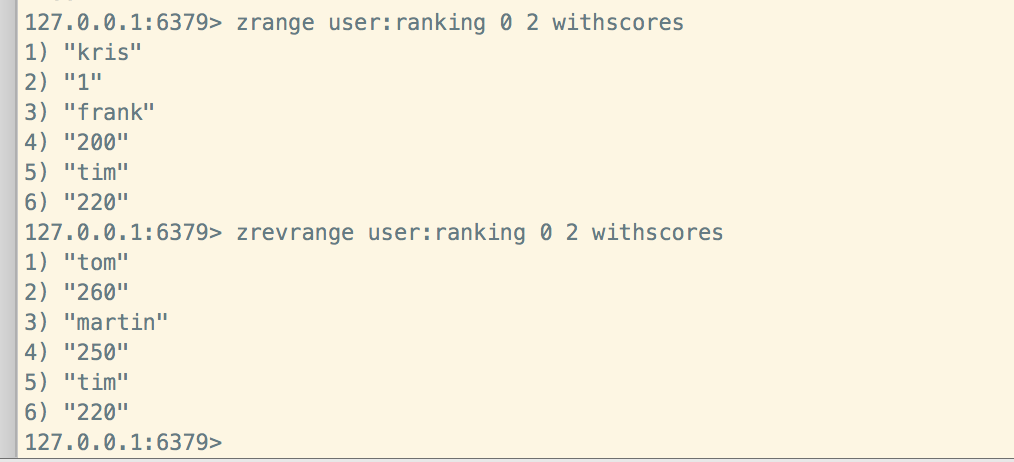
zadd的时间服战度O(logn),sadd的时间复杂度为O(l)

1. 计算成员个数：zcard key
2. 计算某个成员的分数：zscore key member。

1. 计算成员的排名：zrank key member 或者 zrevrank key member

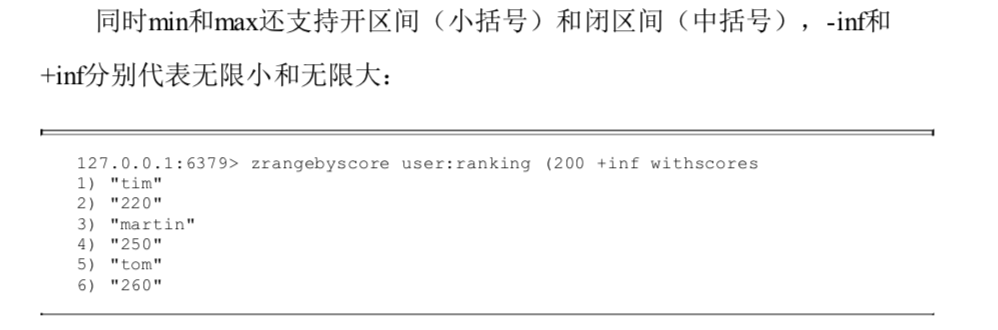
zrank:从分数从低到搞=高返回排名，zrevrank反之

1. 删除成员：zrem key member [member ...]
2. 增加成员的分数：zincrby key increment member
3. 返回指定排名范围的成员：zrange key start end [withscores] 或者 zrevrange key start end [withscores]



1. 返回指定分数范围的成员：zrangebyscore key min max [withscores] [limit offset count]

zrevrangebyscore key min max [withscores] [limit offset count]



1. 返回指定分数范围成员个数：zcount key min max
2. 删除指定排名内的升序元素：zremrangebyrank key start end
3. 删除指定分数范围的成员：zremrangebyscore key min max
4. 集合间的操作

zadd user:ranking:1 1 kris 91 mike 200 frank 220 tim 250 martin 251 tom

zadd user:ranking:2 8 james 77 mike 625 martin 888 tom

1. 交集：zinterstore destiontion numkeys key [key ....] [weights weighr [weight ...]] [aggregate sum|min|max]

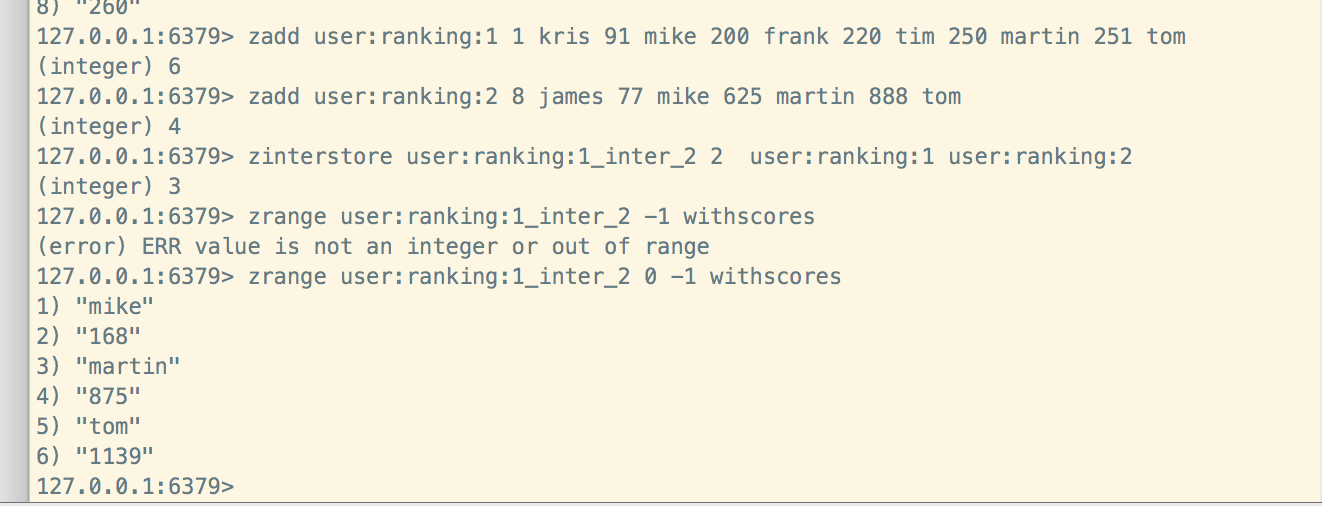
destination:交集计算结果保存到这个键中

numkeys：需要做交集计算键的个数。

key[key....]：需要做交集计算的键

weights weight[weight...]：每个键的权重，在做交集计算时，每个键中的每个member会将自己分数乘以这个权重，每个键的权重默认是1.

aggregate sum|min|max:计算成员交集后，分值可以按照sum(和)、min（最小值），max（最大值）做汇总。默认值sum



1. 并集

zunionstore destination numkeys key [key...] [weights weight [weight ...]] [aggregate sum|min|max]



### 2.6.2 内部编码（2种）

ziplist（压缩列表）：元素个数小于zset-max-ziplist-enties配置（默认128个），同时每个元素的值都小于zset-max-ziplist-value配置（默认64字节）。

skiplist（跳跃表）：当ziplist条件不满足时

### 2.6.3 使用场景

1. 添加用户赞数
2. 取消用户赞数
3. 展示获取赞数最多的是个用户
4. 展示用户信息以及用户分数

## 2.7 键管理

### 2.7.1 单个键管理

1. 键重命名 rename key newkey

renamenx 确保只有newKey不存在时候才被覆盖。

1. 随机返回一个键：randomkey
2. 键过期：

expire key seconds：键在seconds秒后过期

expireat key timestamp：键在秒级时间戳timestamp后过期

pttl查询键的剩余过期时间，pttl进度更高，可以达到毫秒级别，有3种过期时间。

-1:键没有设置过期时间

-2:键不存在

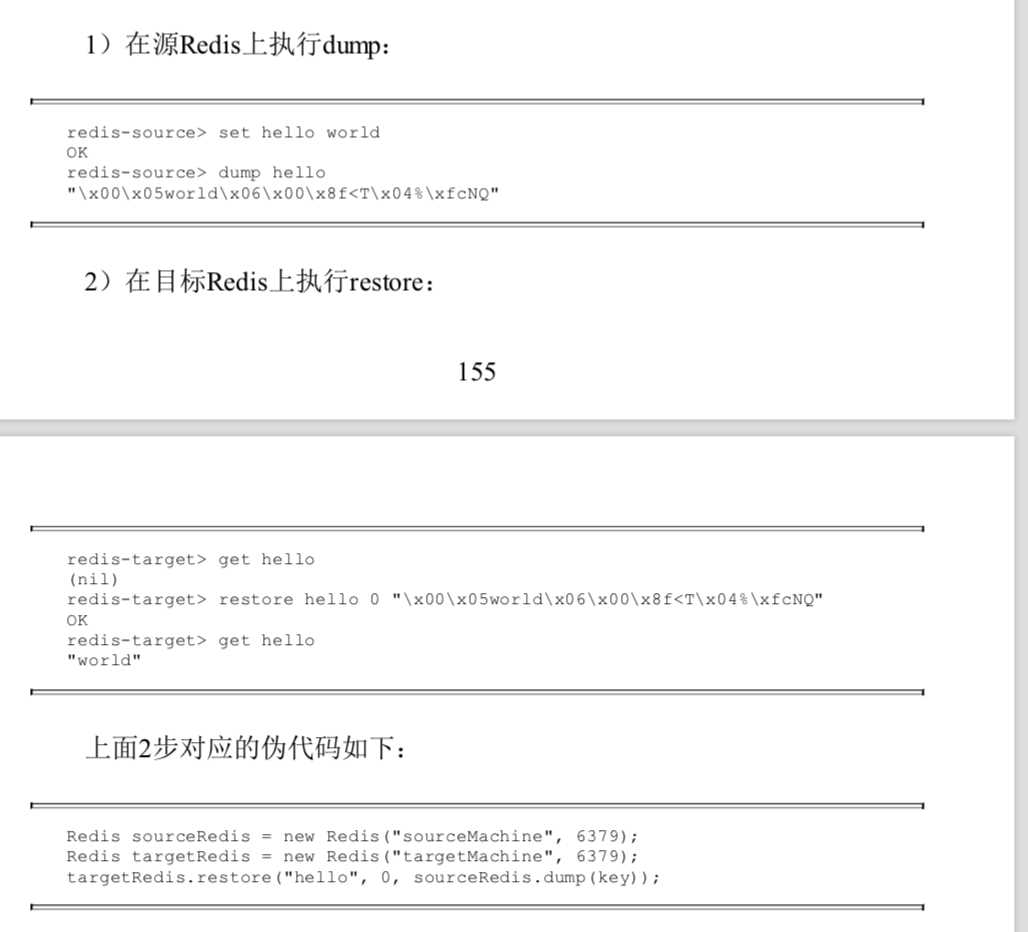
pexpire key milliseconds:键在milliseconds毫秒后过期

pexpireat key milluseconds-timestamp键在毫秒级时间戳timestamp后过期

1. 如果expire key键不存在，返回结果为0
2. 如果过期时间为负值，键会立即被删除，犹如使用del命令一样：
3. persist命令可以将键的过期时间清除：
4. 对于字符串类型键，执行set命令会去掉过期时间，这个问题很容易在开发中被忽视
5. Redis不支持二级数据结构(例如哈希、列表)内部元素的过期功能
6. setex命令作为set+expire的组合，不但是原子执行，同时减少了一次网络通讯的时间
7. 迁移键
8. move：move key db:redis内部有多个数据库
9. dump+restore：dump key，restore key ttl value

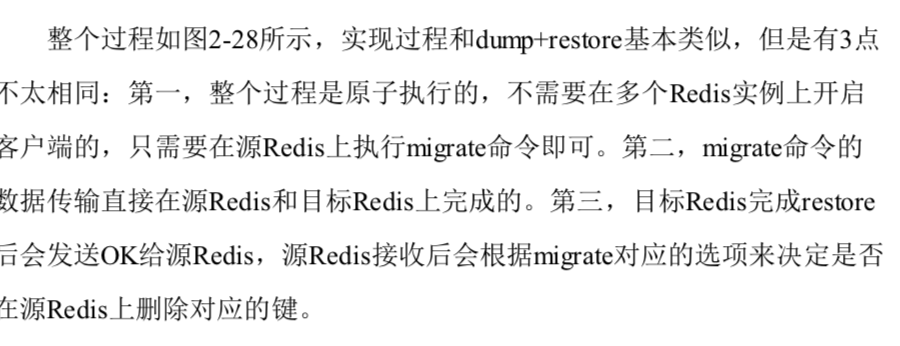
在原Redis上，dump命令会将键值序列化，格式采用的是RDB格式

在目标Redis上，restore命令将上面序列化的值进行复原，其中ttl参数代表过期时间，如果ttl=0代表没有过期时间



migrate host port key|”” destination-db timeout [copy] [replace] [keys key [key]]

migrate命是将dump、restore、del三个命令进行组合、从而监护了操作流程。



### 2.7.2 遍历键

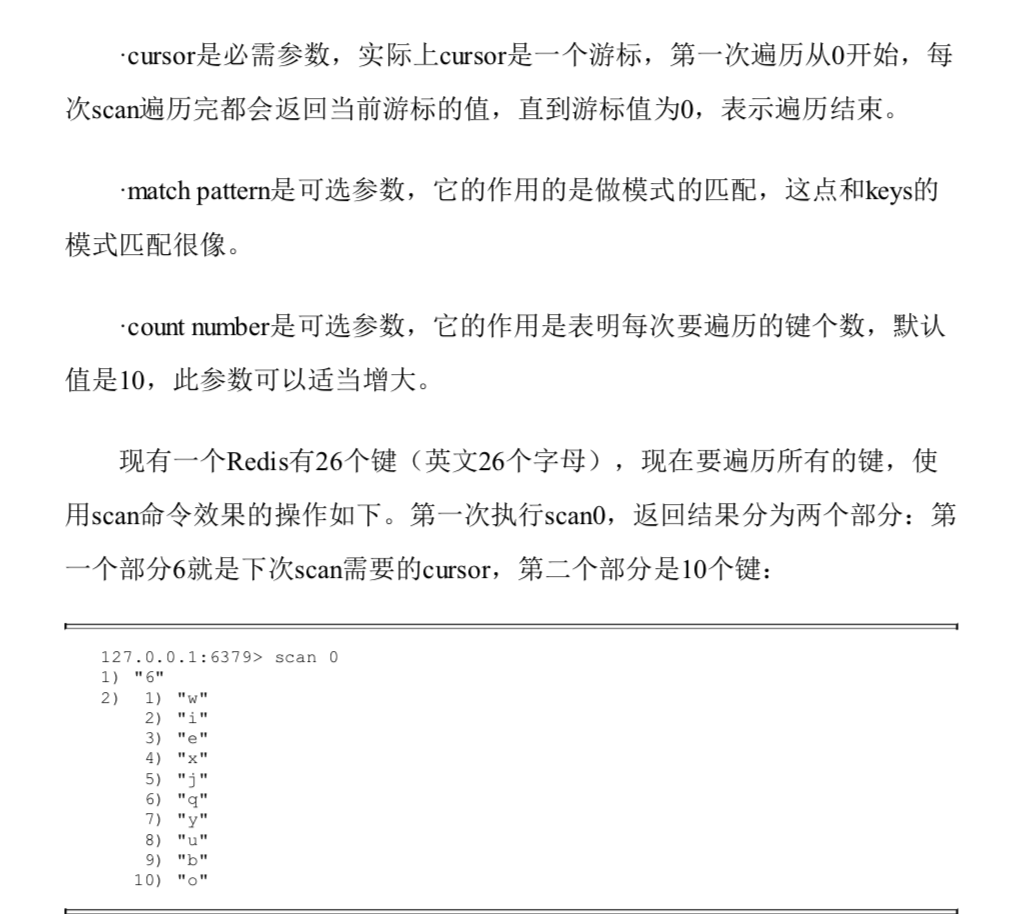
1.全量遍历键

keys pattern

1. 渐进式遍历

Redis存储键值对实际使用的是hashtable的数据结构

scan cursor [match parttern] [count number]



### 2.7.3 数据库管理

1.切换数据库

select dbIndex

1. flushdb/flushall

命令用于清除数据库，前者只清楚当前数据库，flushall会清除所有数据库

# 小功能大用处

Redis还提供了诸如慢查询分析、功能强大的Redis Shell、Pipeline、事务与Lua脚本

慢查询分析：通过满查询分析，找到有问题的命令进行优化。

Redis Shell：功能强大的Redis Shell 会有意想不到的使用功能。

Pipeline：通过Pipeline(管道或者流水线)机制有效提高客户端性能。

事务与Lua：制作自己的专属原子命令

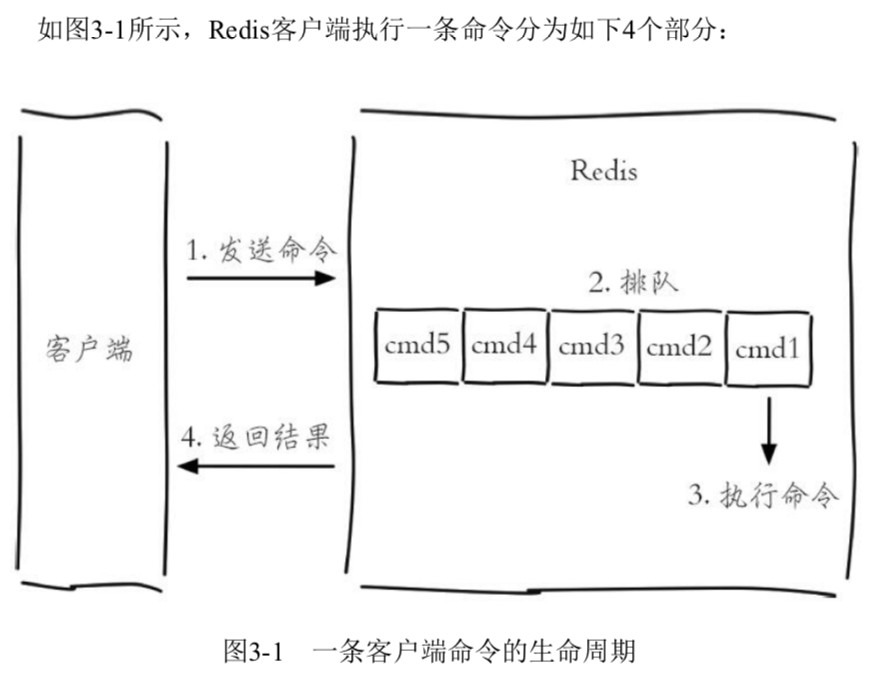
Bitmaps：通过在字符串数据结构上使用为操作，有效节省内存，为开发提供新的思路

HyperLogLog：一种基于概率的新算法，难以想象第节省内存空间。

发布订阅：基于发布订阅模式的消息通信机制。

GEO：Redis3.2提供了基于滴滴位置信息的功能。

## 3.1 慢查询分析



满查询只统计步骤3的时间，所以没有满查询并不代表客户端没有超时时间

### 3.1.1 慢查询的两个配置参数

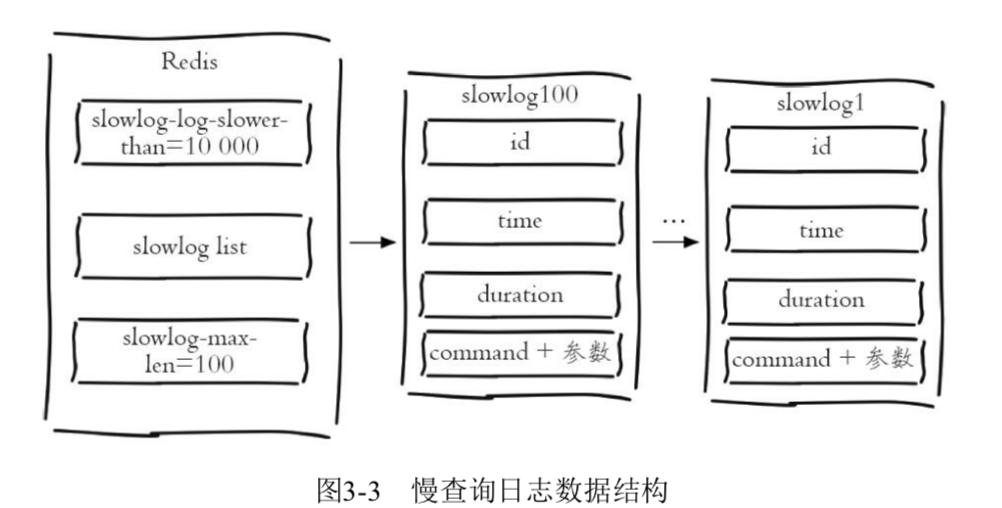
对于慢查询功能，需要明确两件事：预设阈值怎么设置？慢查询记录存放在哪？

showlog-log-slower-than（微秒，默认10000微秒）和slowlog-max-len配置来解决了两个问题。

showlog-log-slower-than=0会记录所有命令。showlog-log-slower-than<0对于任何命令都不会进行记录

slowlog-max-len是存慢查询记录的长度，用一个列表存储的。

1. 获取慢查询日志：slowlog get [n]



1. 获取慢查询日志列表当前的长度：slowlog len
2. 慢查询日志重置：slowlog reset 实际是对列表做清理操作

### 3.1.2 最佳实践

slowlog-max-len：Redis会对长命令做截断操作，线上可设置为1000以上

slowlog-log-slower-than：默认超过10毫秒。对于高流量场景，如果命令执行时间在1毫秒以上，那么Redis最多可制成OPS不到1000.

## 3.2 Redis Shell