# **[Redis的五种数据结构的内部编码](https://www.cnblogs.com/yangmingxianshen/p/8054094.html)**

type命令实际返回的就是当前键的数据结构类型，它们分别是：string（字符串）、hash（哈希）、

list（列表）、set（集合）、zset（有序集合），但这些只是Redis对外的数据结构。



实际上每种数据结构都有自己底层的内部编码实现，而且是多种实现，这样Redis会在合适的场景选择合适的内部编码。

Redis会根据数据量自适应地选择较优化的内部编码格式。

可以看到**每种数据结构都有两种以上的内部编码实现**，例如string数据结构就包含了raw、int和embstr三种内部编码。

同时，**有些内部编码可以作为多种外部数据结构的内部实现**，例如ziplist就是hash、list和zset共有的内部编码。

我们可以通过object encoding命令查询内部编码：

[IMG_256](https://www.cnblogs.com/yangmingxianshen/p/javascript:void(0);)

127.0.0.1:6379> set set:1 hello

OK

127.0.0.1:6379> object encoding set:1

"embstr"

127.0.0.1:6379> hset user:1 name kebi

(integer) 1

127.0.0.1:6379> object encoding user:1

"ziplist"

[IMG_257](https://www.cnblogs.com/yangmingxianshen/p/javascript:void(0);)

可以看到键set:1对应值的内部编码是“embstr”，键user:1对应值的内部编码是“ziplist”。

Redis这样设计有两个好处：

第一，**可以改进内部编码**，而对外的数据结构和命令没有影响，这样一旦开发开发出优秀的内部编码，无需改动外部数据结构和命令。

第二，**多种内部编码实现可以在不同场景下发挥各自的优势**。例如ziplist比较节省内存，但是在列表元素比较多的情况下，性能会有所下降，

　　这时候Redis会根据配置选项将列表类型的内部实现转换为linkedlist。

Redis可以支撑每秒数万的读写操作，但是这指的是Redis服务端的处理

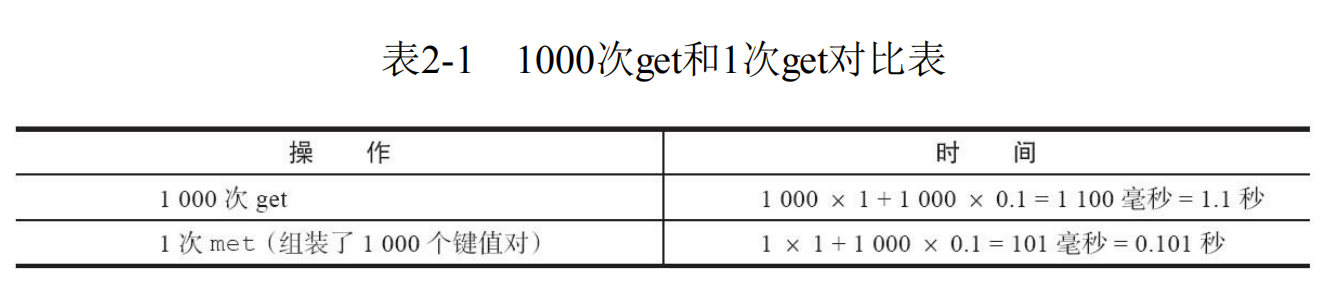
能力，对于客户端来说，一次命令除了命令时间还是有网络时间，假设网络

时间为1毫秒，命令时间为0.1毫秒（按照每秒处理1万条命令算），那么执

行1000次get命令和1次mget命令的区别如表2-1，因为Redis的处理能力已经

足够高，对于开发人员来说，网络可能会成为性能的瓶颈。

表2-1 1000次get和1次get对比表



学会使用批量操作，有助于提高业务处理效率，但是要注意的是每次批

量操作所发送的命令数不是无节制的，如果数量过多可能造成Redis阻塞或

者网络拥塞。

### Redis类型及内部编码

字符串



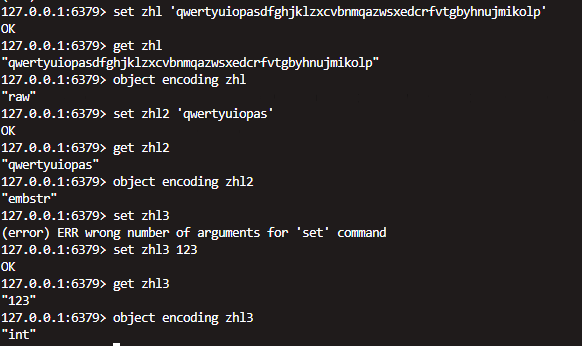
字符串类型的内部编码有3种：

·int：8个字节的长整型。

·embstr：小于等于39个字节的字符串。

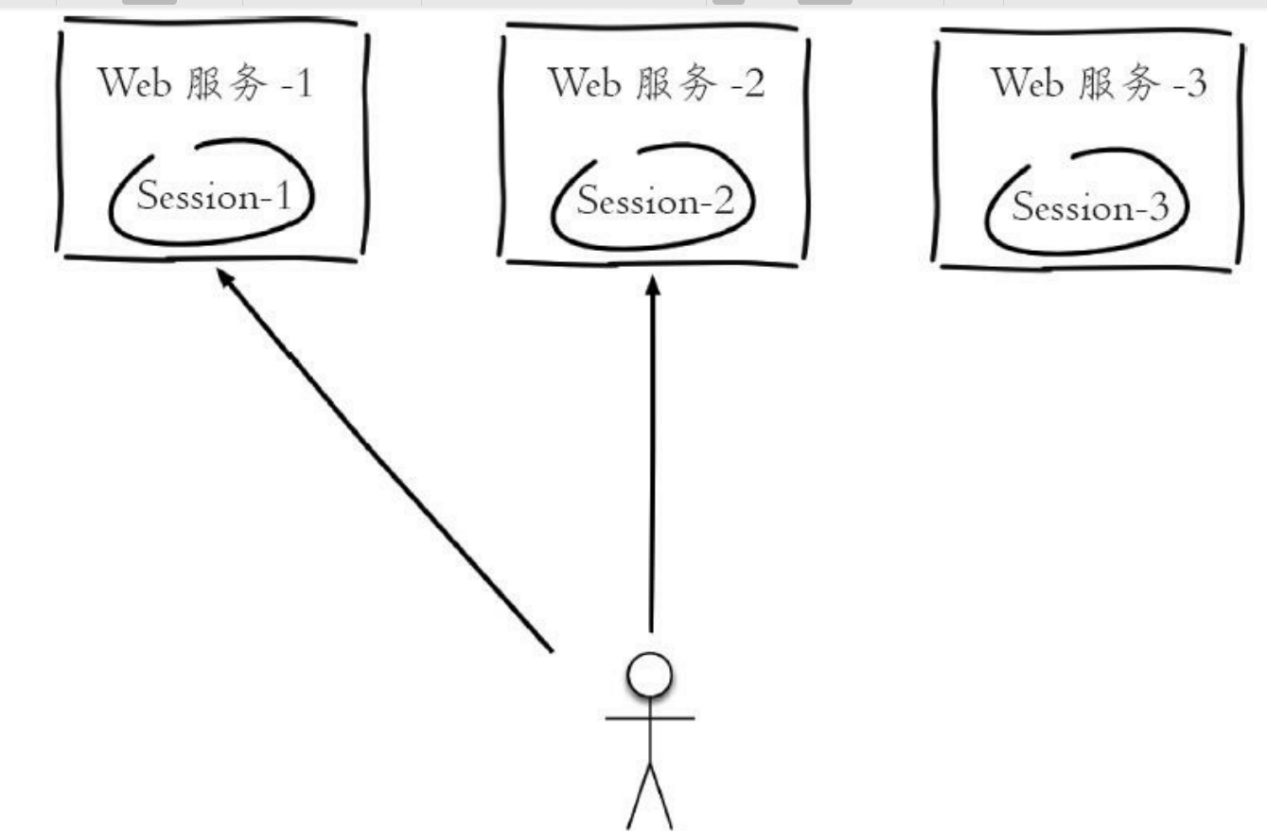
·raw：大于39个字节的字符串。

Redis会根据当前值的类型和长度决定使用哪种内部编码实现

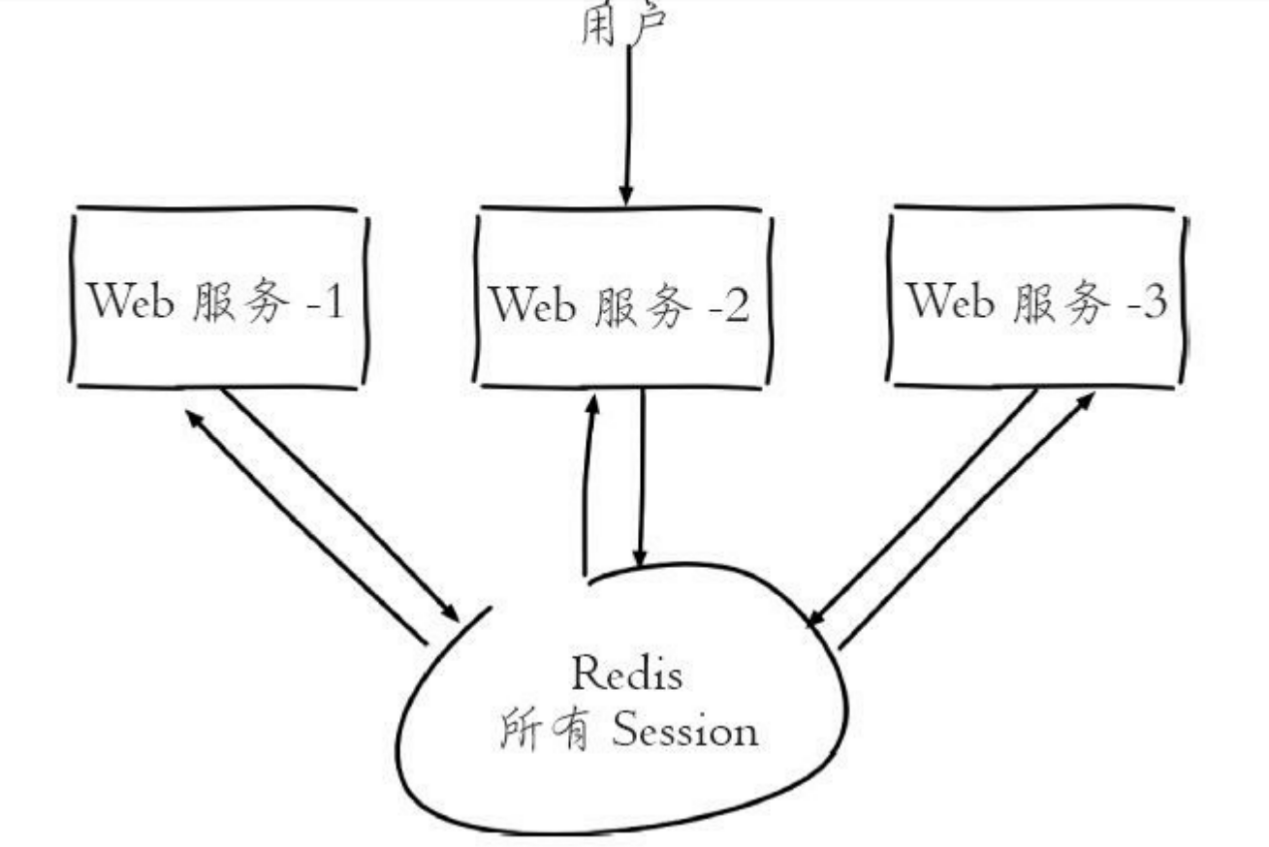


如果键名比较长，例如“user：{uid}：friends：messages：{mid}”，可以在能描述键含义的前提下适当减少键的长度，例如变为“u：{uid}：fr：m：{mid}”，从而减少由于键过长的内存浪费。

处于负载均衡的考虑，分布式服务会将用户的访问均衡到不同服务器上<但是分布式Web服务将用户的Session信息（例如用户登录信息）保存在各自服务器中>



在这种模式下只要保证Redis是高可用和扩展性的，每次用户更新或者查询登录信息都直接从Redis中集中获取。



为了短信接口不被频繁访问，会限制用户每分钟获取验证码的频率，例如一分钟不能超过5次

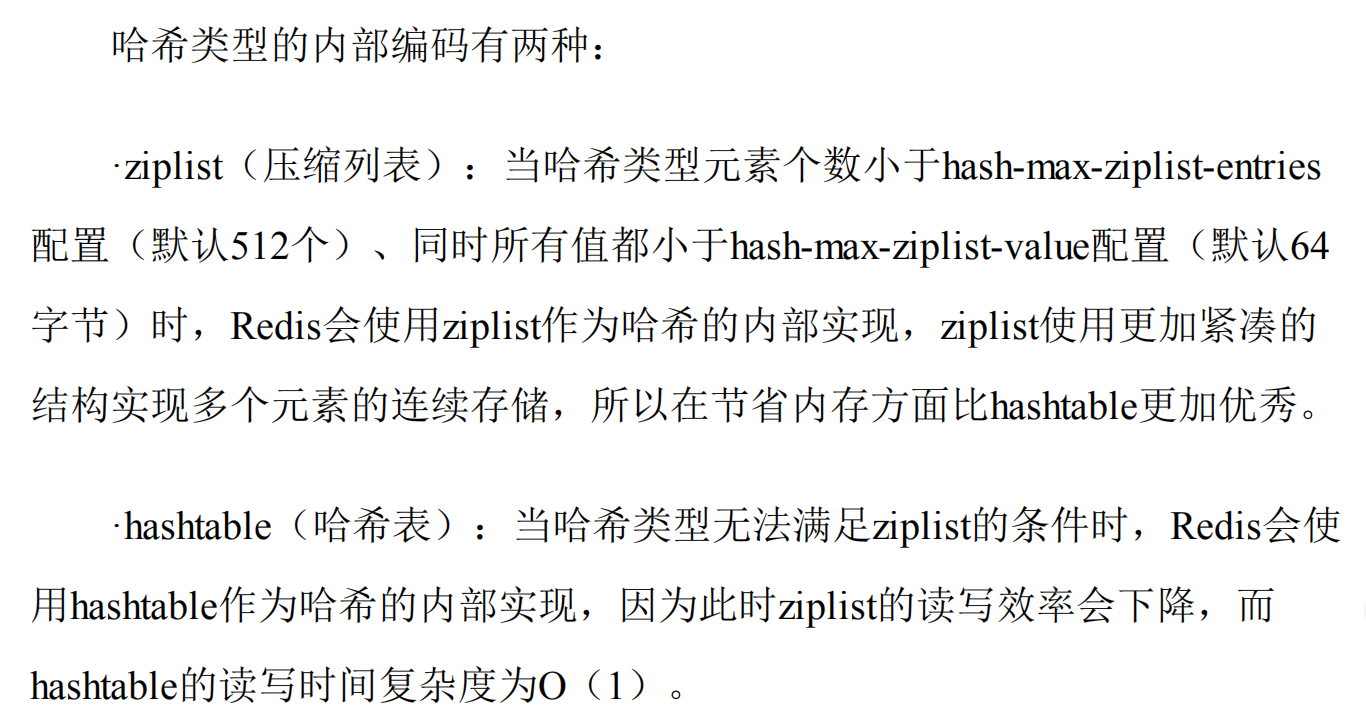
哈希

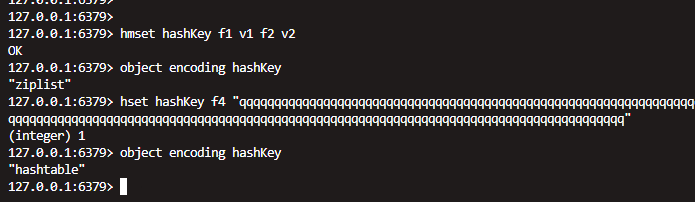
在使用hgetall时，如果哈希元素个数比较多，会存在阻塞Redis的可能。

如果开发人员只需要获取部分field，可以使用hmget，如果一定要获取全部

field-value，可以使用hscan命令，该命令会渐进式遍历哈希类型。







1. 当field个数比较少且没有大的value时，内部编码为ziplist;
2. 当有value大于64字节，内部编码会由ziplist变为hashtable;
3. 当field个数超过512，内部编码也会由ziplist变为hashtable.

使用场景：

哈希类型和关系型数据库有两点不同之处：

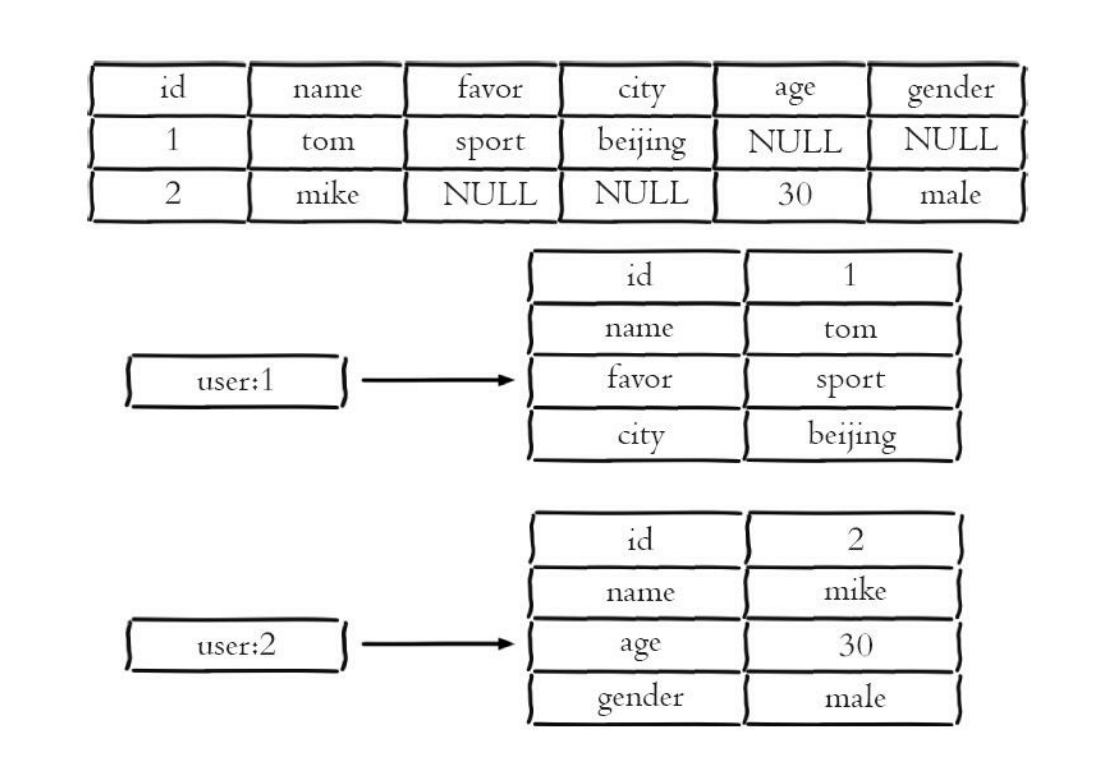
·哈希类型是稀疏的，而关系型数据库是完全结构化的，例如哈希类型

每个键可以有不同的field，而关系型数据库一旦添加新的列，所有行都要为

其设置值（即使为NULL），如图2-17所示。

·关系型数据库可以做复杂的关系查询，而Redis去模拟关系型复杂查询

开发困难，维护成本高。



我们已经能够用三种方法缓存用户信息，下面给出三种方

案的实现方法和优缺点分析。

1）原生字符串类型：每个属性一个键。

set user:1:name tom

set user:1:age 23

set user:1:city beijing

优点：简单直观，每个属性都支持更新操作。

缺点：占用过多的键，内存占用量较大，同时用户信息内聚性比较差，

所以此种方案一般不会在生产环境使用。

2）序列化字符串类型：将用户信息序列化后用一个键保存。

set user:1 serialize(userInfo)

优点：简化编程，如果合理的使用序列化可以提高内存的使用效率。

缺点：序列化和反序列化有一定的开销，同时每次更新属性都需要把全

部数据取出进行反序列化，更新后再序列化到Redis中。

3）哈希类型：每个用户属性使用一对field-value，但是只用一个键保

存。

hmset user:1 name tomage 23 city beijing

优点：简单直观，如果使用合理可以减少内存空间的使用。

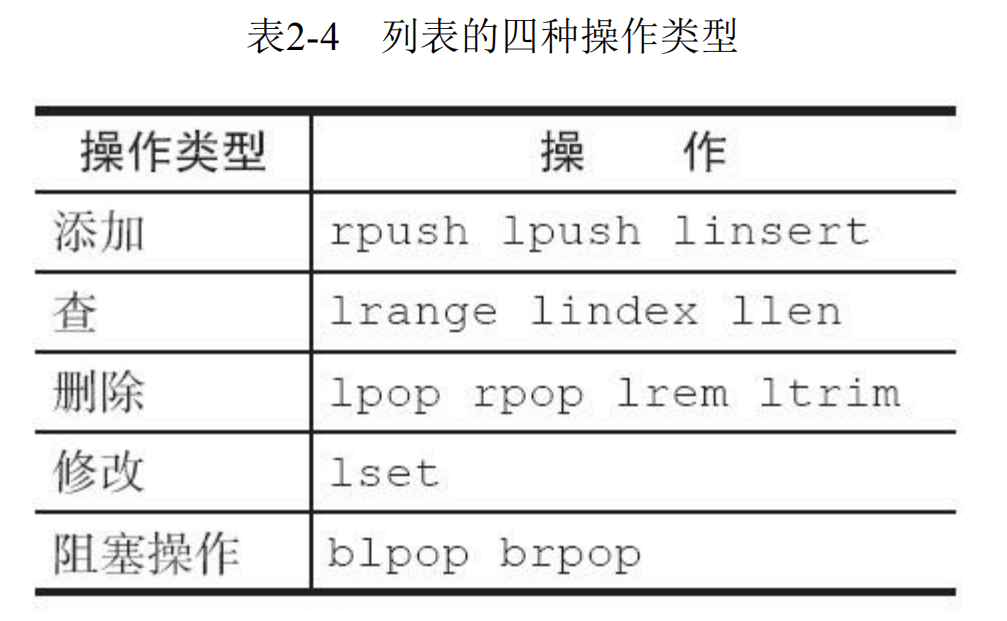
缺点：要控制哈希在ziplist和hashtable两种内部编码的转换，hashtable会

消耗更多内存。

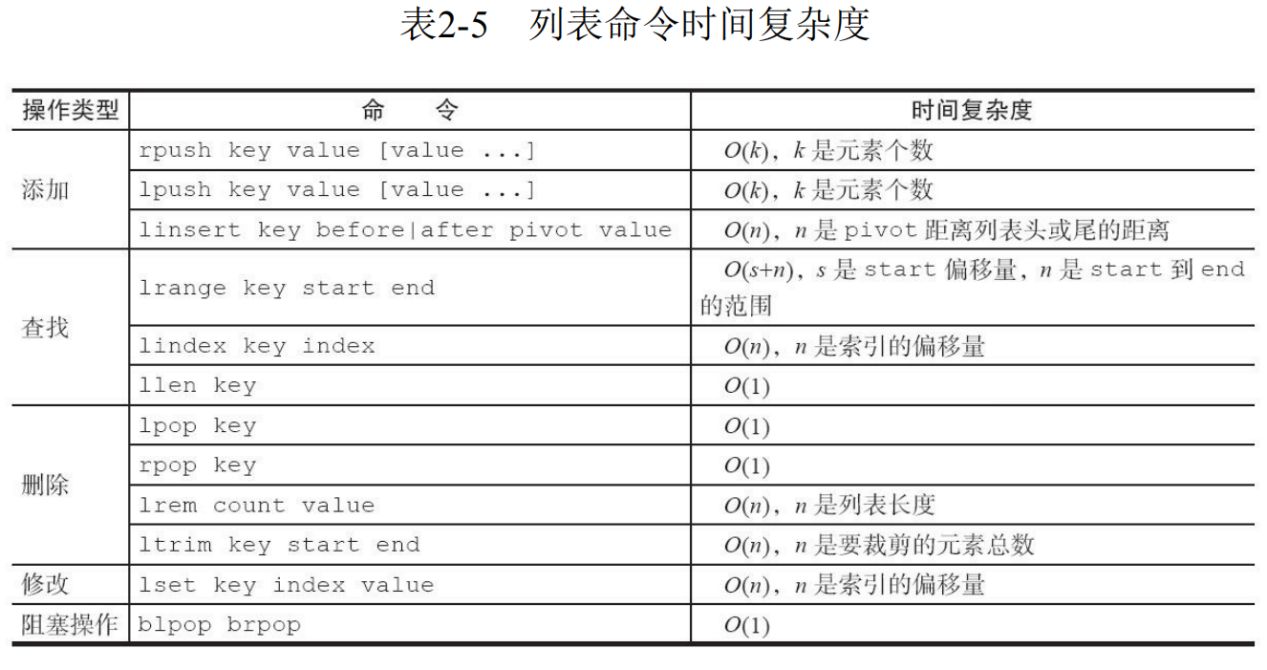
列表

列表（list）类型是用来存储多个有序的字符串，a、b、c、d、e五个元素从左到右组成了一个有序的列表，列表中的每个字符串称为元素（element），一个列表最多可以存储232 - 1 个元素 (4294967295, 每个列表超过40亿个元素)。

列表类型有两个特点：第一、列表中的元素是有序（先后顺序，所以可以弹入弹出对应的元素）的，第二、列表中的元素可以是重复的。







内部编码

列表类型的内部编码有两种。

·ziplist（压缩列表）：当列表的元素个数小于list-max-ziplist-entries配置

（默认512个），同时列表中每个元素的值都小于list-max-ziplist-value配置时

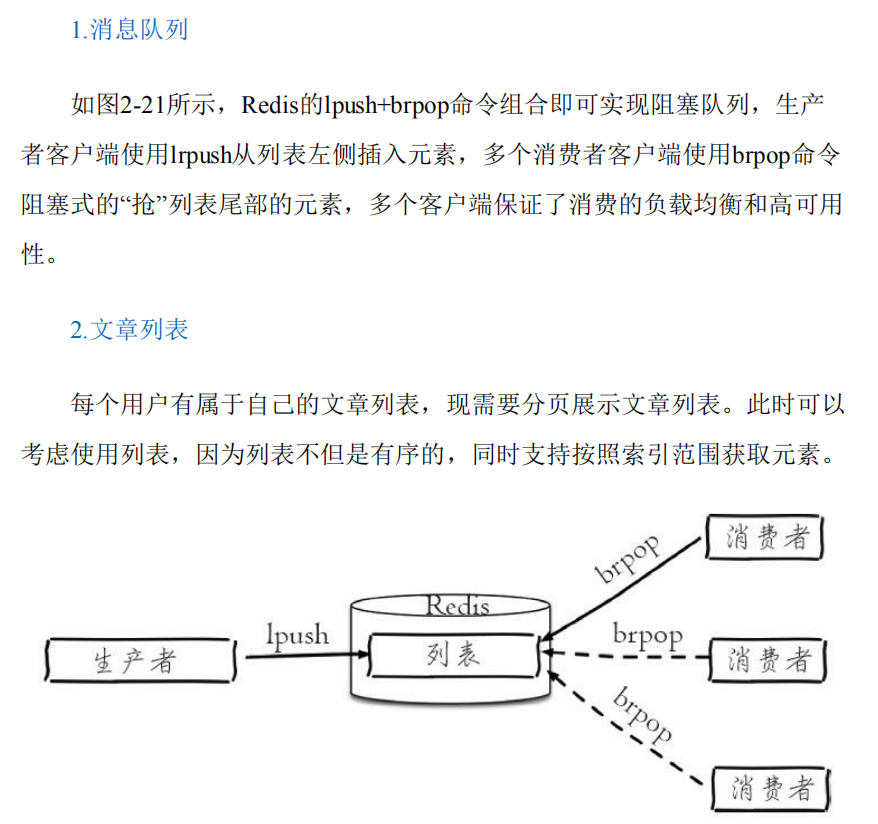
（默认64字节），Redis会选用ziplist来作为列表的内部实现来减少内存的使

用。

·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。

使用场景



使用场景

实际上列表的使用场景很多，在选择时可以参考以下口诀：

·lpush+lpop=Stack（栈）

·lpush+rpop=Queue（队列）

·lpsh+ltrim=Capped Collection（有限集合）

·lpush+brpop=Message Queue（消息队列）

集合



内部编码

集合类型的内部编码有两种：

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

现，从而减少内存的使用。

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

用hashtable作为集合的内部实现。

使用场景

集合类型比较典型的使用场景是标签（tag）。例如一个用户可能对娱

乐、体育比较感兴趣，另一个用户可能对历史、新闻比较感兴趣，这些兴趣

点就是标签。有了这些数据就可以得到喜欢同一个标签的人，以及用户的共

同喜好的标签，这些数据对于用户体验以及增强用户黏度比较重要。

可以使用sinter命令，来计算用户共同感兴趣的标签。

前面只是给出了使用Redis集合类型实现标签的基本思路，实际上一个

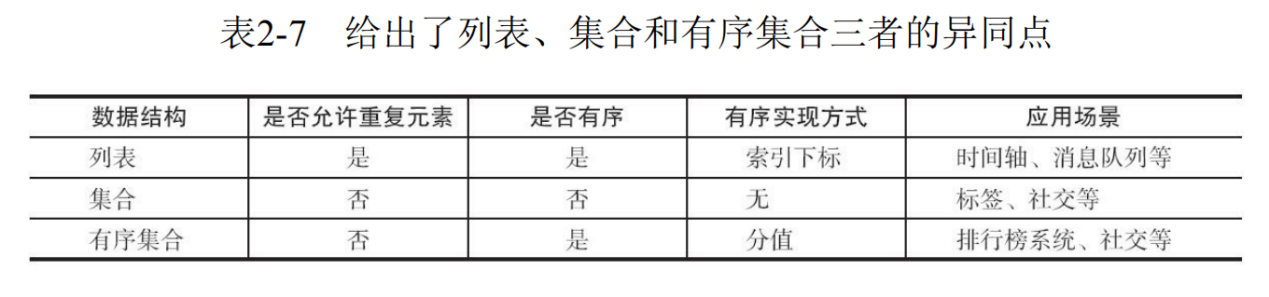
标签系统远比这个要复杂得多，不过集合类型的应用场景通常为以下几种：

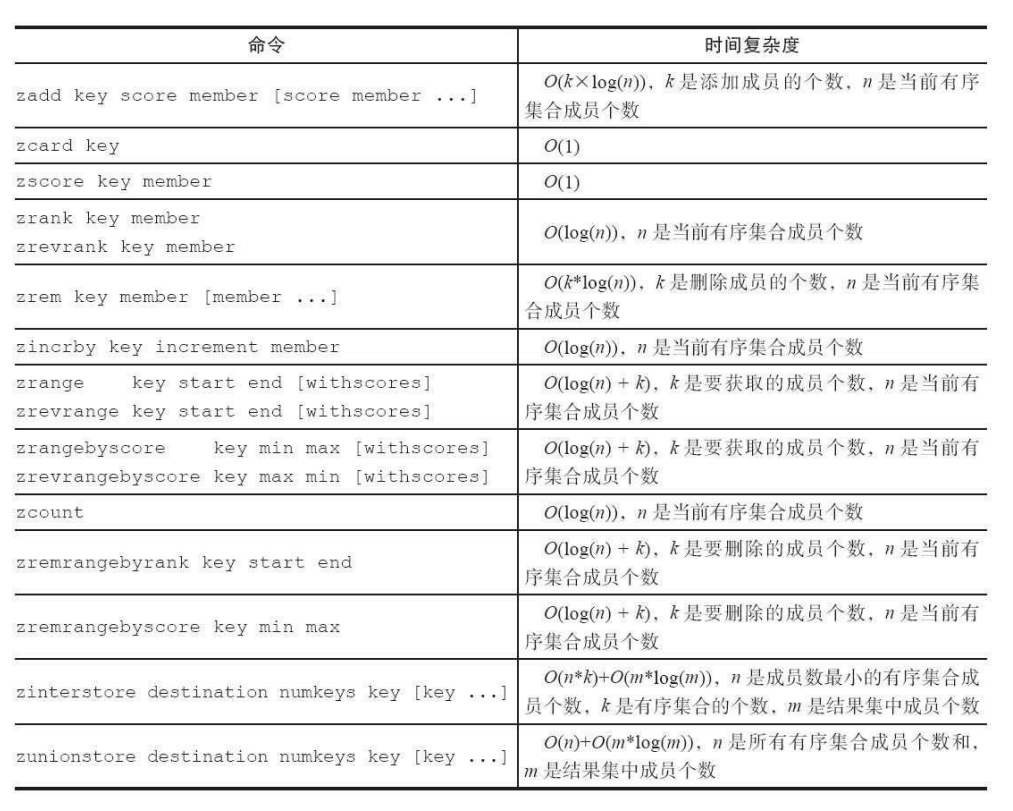
·sadd=Tagging（标签）

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

·sadd+sinter=Social Graph（社交需求）

有序集合





内部编码

有序集合类型的内部编码有两种：

·ziplist（压缩列表）：当有序集合的元素个数小于zset-max-ziplistentries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）时，Redis会用ziplist来作为有序集合的内部实现，ziplist

可以有效减少内存的使用。

·skiplist（跳跃表）：当ziplist条件不满足时，有序集合会使用skiplist作

为内部实现，因为此时ziplist的读写效率会下降。

迁移键

迁移键功能非常重要，因为有时候我们只想把部分数据由一个Redis迁

移到另一个Redis（例如从生产环境迁移到测试环境），Redis发展历程中提

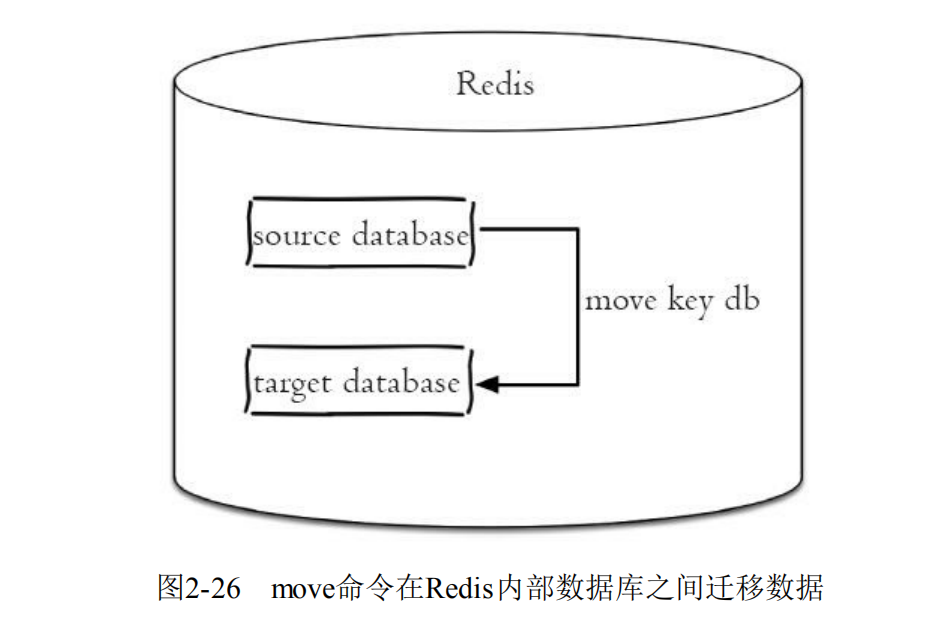
供了move、dump+restore、migrate三组迁移键的方法，它们的实现方式以及

使用的场景不太相同，下面分别介绍。

（1）move

move key db

如图2-26所示，move命令用于在Redis内部进行数据迁移，Redis内部可以有多个数据库，由于多个数据库功能后面会进行介绍，这里只需要知道Redis内部可以有多个数据库，彼此在数据上是相互隔离的，move key db就是把指定的键从源数据库移动到目标数据库中，但笔者认为多数据库功能不建议在生产环境使用，所以这个命令读者知道即可。

（2）dump+restore

dump key

restore key ttl value

dump+restore可以实现在不同的Redis实例之间进行数据迁移的功能，整

个迁移的过程分为两步：

1）在源Redis上，dump命令会将键值序列化，格式采用的是RDB格式。

2）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。

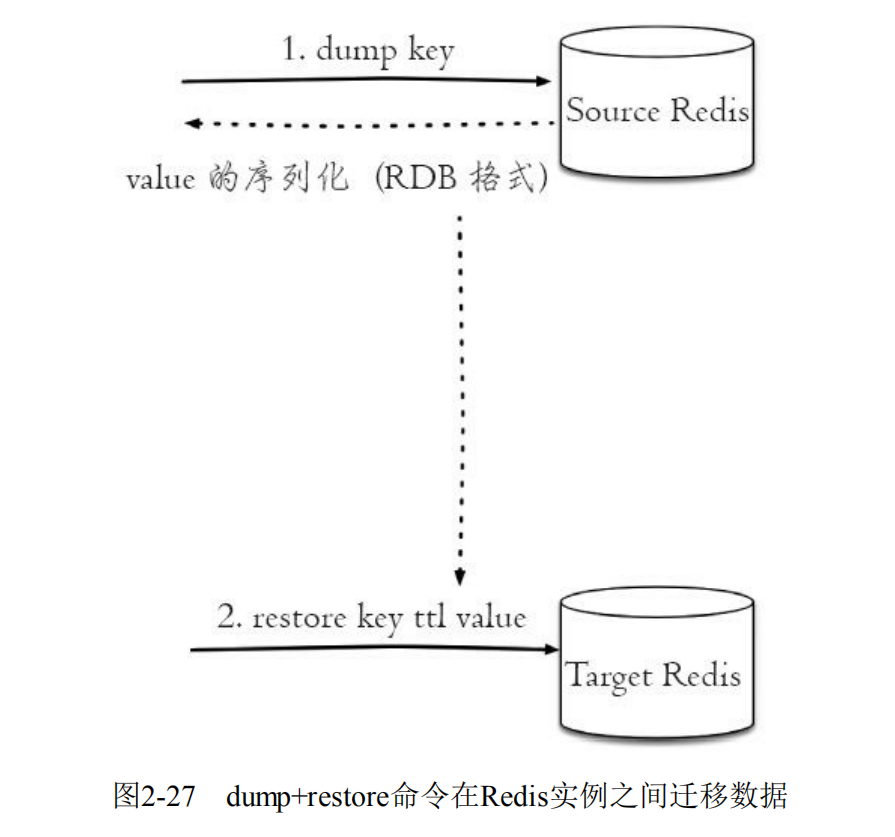


图2-27 dump+restore命令在Redis实例之间迁移数据

有关dump+restore有两点需要注意：第一，整个迁移过程并非原子性

的，而是通过客户端分步完成的。第二，迁移过程是开启了两个客户端连

接，所以dump的结果不是在源Redis和目标Redis之间进行传输，下面用一个

例子演示完整过程。

1）在源Redis上执行dump：

redis-source> set hello world

OK

redis-source> dump hello

"\x00\x05world\x06\x00\x8f<T\x04%\xfcNQ"

2）在目标Redis上执行restore：

155

redis-target> get hello

(nil)

redis-target> restore hello 0 "\x00\x05world\x06\x00\x8f<T\x04%\xfcNQ"

OK

redis-target> get hello

"world"

上面2步对应的伪代码如下：

Redis sourceRedis = new Redis("sourceMachine", 6379);

Redis targetRedis = new Redis("targetMachine", 6379);

targetRedis.restore("hello", 0, sourceRedis.dump(key));

（3）migrate

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

migrate命令也是用于在Redis实例间进行数据迁移的，实际上migrate命

令就是将dump、restore、del三个命令进行组合，从而简化了操作流程。

migrate命令具有原子性，而且从Redis3.0.6版本以后已经支持迁移多个键的

功能，有效地提高了迁移效率，migrate在10.4节水平扩容中起到重要作用。

整个过程如图2-28所示，实现过程和dump+restore基本类似，但是有3点

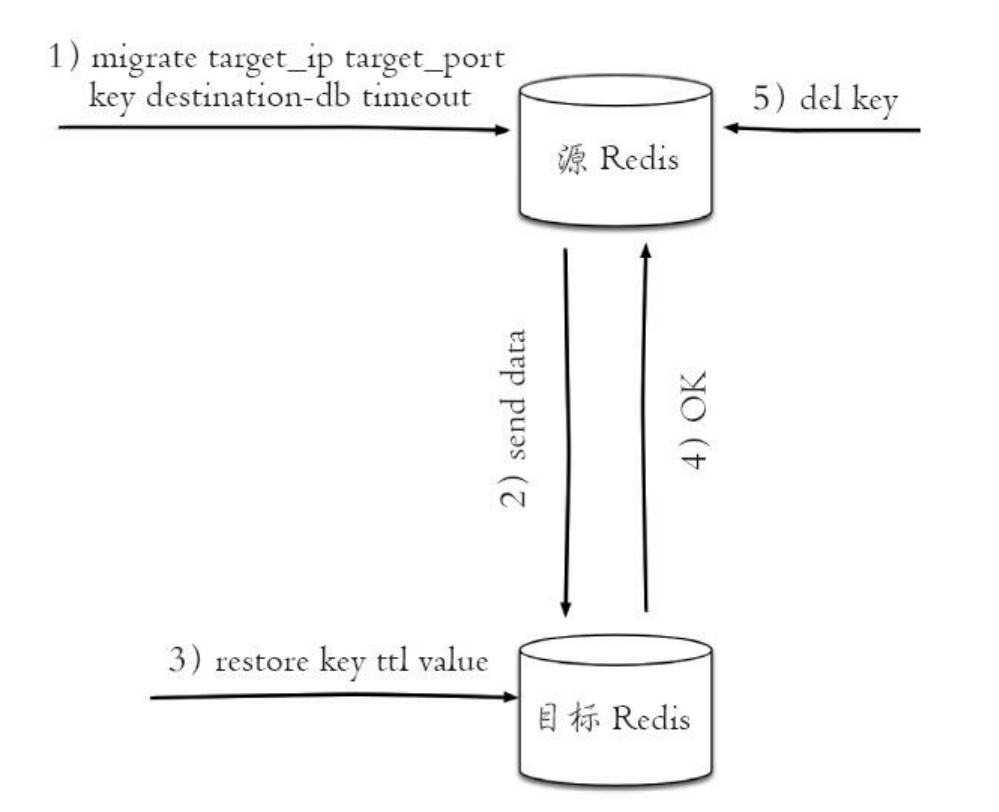
不太相同：第一，整个过程是原子执行的，不需要在多个Redis实例上开启

客户端的，只需要在源Redis上执行migrate命令即可。第二，migrate命令的

数据传输直接在源Redis和目标Redis上完成的。第三，目标Redis完成restore

后会发送OK给源Redis，源Redis接收后会根据migrate对应的选项来决定是否

在源Redis上删除对应的键。



下面对migrate的参数进行逐个说明：

·host：目标Redis的IP地址。

·port：目标Redis的端口。

·key|""：在Redis3.0.6版本之前，migrate只支持迁移一个键，所以此处是

要迁移的键，但Redis3.0.6版本之后支持迁移多个键，如果当前需要迁移多

个键，此处为空字符串""。

·destination-db：目标Redis的数据库索引，例如要迁移到0号数据库，这

157

里就写0。

·timeout：迁移的超时时间（单位为毫秒）。

·[copy]：如果添加此选项，迁移后并不删除源键。

·[replace]：如果添加此选项，migrate不管目标Redis是否存在该键都会

正常迁移进行数据覆盖。

·[keys key[key...]]：迁移多个键，例如要迁移key1、key2、key3，此处填

写“keys key1 key2 key3”。

下面用示例演示migrate命令，为了方便演示源Redis使用6379端口，目

标Redis使用6380端口，现要将源Redis的键hello迁移到目标Redis中，会分为

如下几种情况：

情况1：源Redis有键hello，目标Redis没有：

127.0.0.1:6379> migrate 127.0.0.1 6380 hello 0 1000

OK

情况2：源Redis和目标Redis都有键hello：

127.0.0.1:6379> get hello

"world"

127.0.0.1:6380> get hello

"redis"

如果migrate命令没有加replace选项会收到错误提示，如果加了replace会

返回OK表明迁移成功：

127.0.0.1:6379> migrate 127.0.0.1 6379 hello 0 1000

(error) ERR Target instance replied with error: BUSYKEY Target key name already exists.

158

127.0.0.1:6379> migrate 127.0.0.1 6379 hello 0 1000 replace

OK

情况3：源Redis没有键hello。如下所示，此种情况会收到nokey的提

示：

127.0.0.1:6379> migrate 127.0.0.1 6380 hello 0 1000

NOKEY

下面演示一下Redis3.0.6版本以后迁移多个键的功能。

·源Redis批量添加多个键：

127.0.0.1:6379> mset key1 value1 key2 value2 key3 value3

OK

·源Redis执行如下命令完成多个键的迁移：

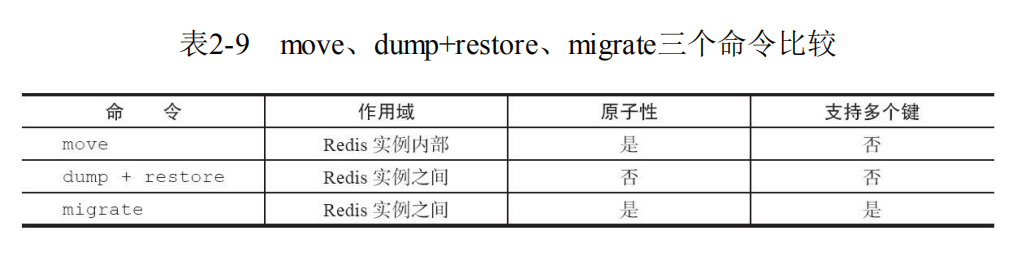
127.0.0.1:6379> migrate 127.0.0.1 6380 "" 0 5000 keys key1 key2 key3

OK

至此有关Redis数据迁移的命令介绍完了，最后使用表2-9总结一下

move、dump+restore、migrate三种迁移方式的异同点，笔者建议使用migrate

命令进行键值迁移。



Redis3.0中已经逐渐弱化这个功能，例如Redis的分布式实现Redis

Cluster只允许使用0号数据库，只不过为了向下兼容老版本的数据库功能，

该功能没有完全废弃掉，下面分析一下为什么要废弃掉这个“优秀”的功能

呢？总结起来有三点：

·Redis是单线程的。如果使用多个数据库，那么这些数据库仍然是使用

一个CPU，彼此之间还是会受到影响的。

·多数据库的使用方式，会让调试和运维不同业务的数据库变的困难，

假如有一个慢查询存在，依然会影响其他数据库，这样会使得别的业务方定

位问题非常的困难。

·部分Redis的客户端根本就不支持这种方式。即使支持，在开发的时候

来回切换数字形式的数据库，很容易弄乱。

flushdb/flushall命令用于清除数据库，两者的区别的是flushdb只清除当

前数据库，flushall会清除所有数据库。

本章重点回顾

1）Redis提供5种数据结构，每种数据结构都有多种内部编码实现。

2）纯内存存储、IO多路复用技术、单线程架构是造就Redis高性能的三个因素。

3）由于Redis的单线程架构，所以需要每个命令能被快速执行完，否则会存在阻塞Redis的可能，理解Redis单线程命令处理机制是开发和运维Redis的核心之一。

4）批量操作（例如mget、mset、hmset等）能够有效提高命令执行的效率，但要注意每次批量操作的个数和字节数。

5）了解每个命令的时间复杂度在开发中至关重要，例如在使用keys、hgetall、smembers、zrange等时间复杂度较高的命令时，需要考虑数据规模对于Redis的影响。

6）persist命令可以删除任意类型键的过期时间，但是set命令也会删除字符串类型键的过期时间，这在开发时容易被忽视。

7）move、dump+restore、migrate是Redis发展过程中三种迁移键的方式，其中move命令基本废弃，migrate命令用原子性的方式实现了dump+restore，并且支持批量操作，是Redis Cluster实现水平扩容的重要工具。

8）scan命令可以解决keys命令可能带来的阻塞问题，同时Redis还提供了hscan、sscan、zscan渐进式地遍历hash、set、zset。