# 基本介绍

Redis的全称是REmote Dictionary Server，是基于**键值对**的NoSQL数据库。

值的类型：

**5**种基本数据类型：

* *String(字符串)*
* *Hash(哈希)*
* *List(列表)*
* *Set(集合)*
* *ZSet(有序集合)*

其他数据类型：

*Bitmaps（位图）*

*HyperLogLog*

*GEO（地理信息定位）*

其他功能：

键过期、发布订阅、事务、流水线、Lua脚本等

应用场景

可做

1.缓存

2.排行榜系统（ZSet）

3.计数器应用（String）

4.社交网络（Set）

5.消息队列系统（List）

不可做

存储冷数据

## 架构

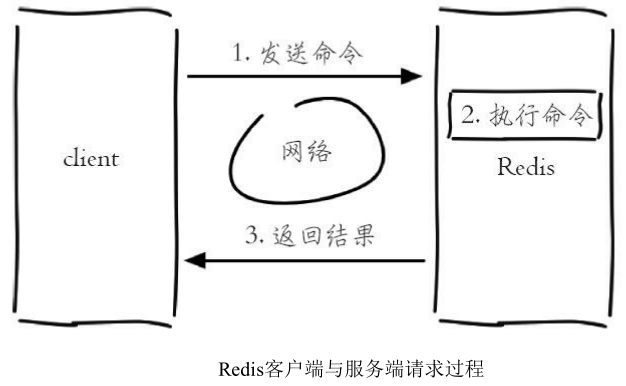
Redis使用**单线程架构**和**I/O多路复用模型**来实现高性能的内存数据库服务。

下图是redis客户端向redis服务发送请求的过程：

1.客户端发送命令

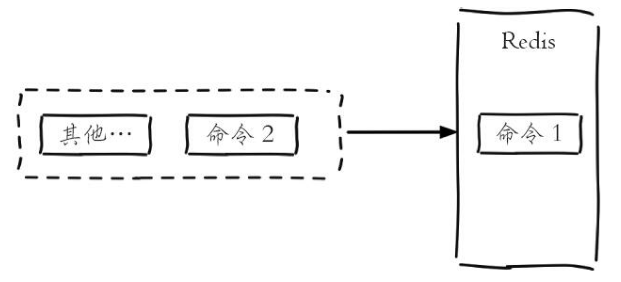
2.redis执行命令

3.返回结果



关键点在第2步：

redis是单线程来处理命令的，一个命令到达redis服务后，并不会立即执行，而是进入一个**队列**中，然后逐个执行。这就保证了线程安全的问题了，执行的顺序是以队列的顺序为主。



redis之所以快的原因：

1.纯内存访问，Redis将所有数据放在内存中，内存的响应时长大约为100纳秒，这是Redis达到每秒万级别访问的重要基础。

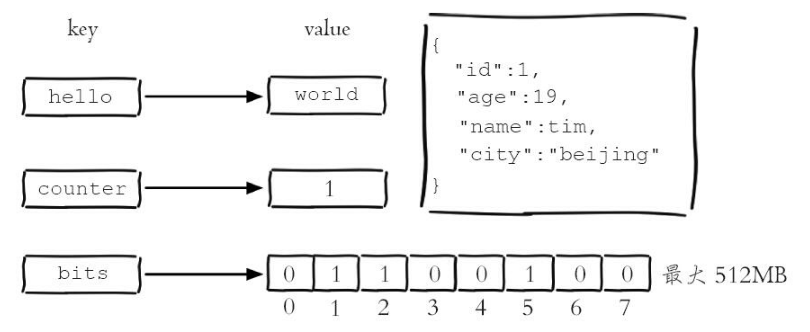
2.非阻塞I/O，Redis使用epoll作为I/O多路复用技术的实现，再加上Redis自身的事件处理模型将epoll中的连接、读写、关闭都转换为事件，不在网络I/O上浪费过多的时间.

3.单线程避免了线程切换和竞态产生的消耗。

**缺陷**

单线程会有一个问题。每个命令的执行时间是有要求的。如果某个命令执行过长，会造成其他命令的阻塞，对Redis这种高性能的服务来说是致命的，所以Redis是面向快速执行场景的数据库。

# String字符串



应用场景

1.缓存功能

2.计数

redis.incr(key)

3.共享Session

将用户信息保存到redis中

4.限速

短信获取频率，设置键过期与自增控制

5.分布式锁

**值最大不能超过512Mb**

**存储**：

**字符串**(简单的字符串、复杂的字符串(例如JSON、XML))

**数字**(整数、浮点数)

**二进制**(图片、音频、视频)

**实现分布式锁**

set命令

ex seconds：为键设置秒级过期时间

px milliseconds：为键设置毫秒级过期时间

nx：键必须不存在，才可以设置成功，用于添加

xx：与nx相反，键必须存在，才可以设置成功，用于更新

**内部编码**

int：8个字节的长整型

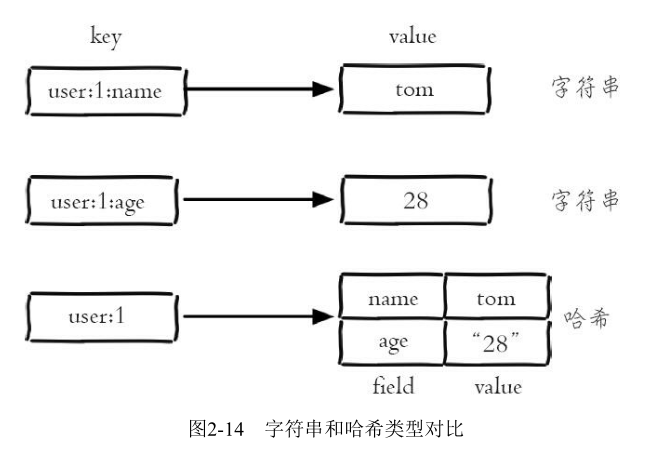
embstr：小于等于39个字节的字符串

raw：大于39个字节的字符串

# Hash哈希

哈希类型中的映射关系叫作field-value，注意这里的value是指field对应

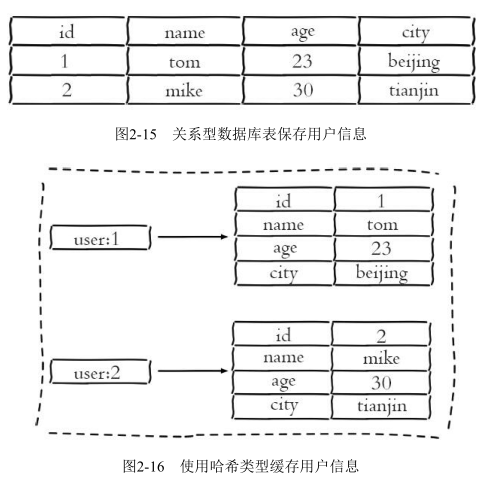
的值，不是键对应的值，请注意value在不同上下文的作用。



应用场景

1.记录对象信息

2.甚至模拟关系型数据库



内部编码

ziplist（压缩列表）

当哈希类型元素个数小于hash-max-ziplist-entries配置（默认512个）

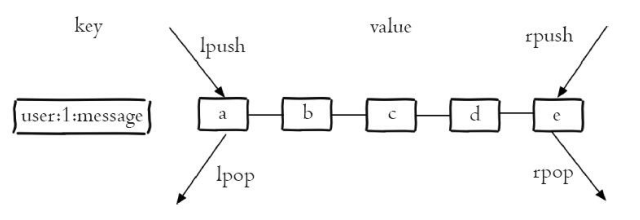
所有值都小于hash-max-ziplist-value配置（默认64字节）

hashtable（哈希表）

哈希类型无法满足ziplist的条件时

# List列表

**特点**：**元素有序，可重复**



应用场景

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）*// blpop和brpop是lpop和rpop的阻塞版本*

一个列表最多可以存储2^32 -1个元素

存储多个有序的字符串

内部编码

ziplist（压缩列表）

列表的元素个数小于list-max-ziplist-entries配置（默认512个）

列表中每个元素的值都小于list-max-ziplist-value配置时（默认64字节）

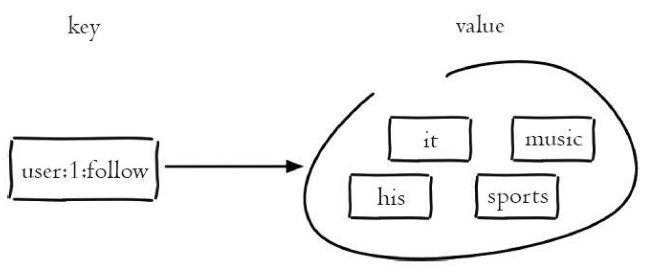
linkedlist（链表）

不满足ziplist的条件

# Set集合

特点：**唯一，无序**

支持多个集合取交集、并集、差集



应用场景

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，**求交集**）

一个集合最多可以存储2 32 -1个元素

存储多个有序的字符串

无序不重复

内部编码

intset（整数集合）

集合中的元素都是整数且元素个数小于set-max-intset-entries配置（默认512个）

hashtable（哈希表）

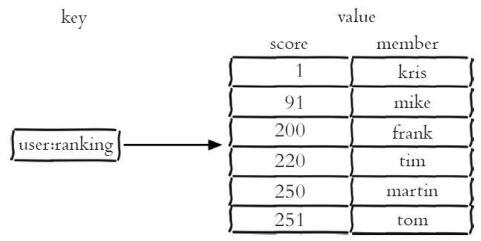
无法满足intset的条件时

# ZSet

特点：可以排序，元素唯一

Set == HasHSet

ZSet == TreeSet



应用场景

排行榜系统

添加用户赞数

取消用户赞数

展示获取赞数最多的十个用户

展示用户信息以及用户分数

元素不能重复，但有序

给每个元素设置一个分数（score）作为排序的依据（score可以重复）

内部编码

ziplist（压缩列表）

有序集合的元素个数小于zset-max-ziplist-entries配置（默认128个）

每个元素的值都小于zset-max-ziplist-value配置（默认64字节）

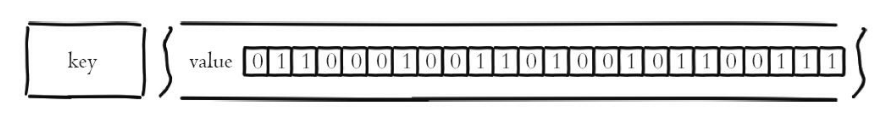
skiplist（跳跃表）

ziplist条件不满足时

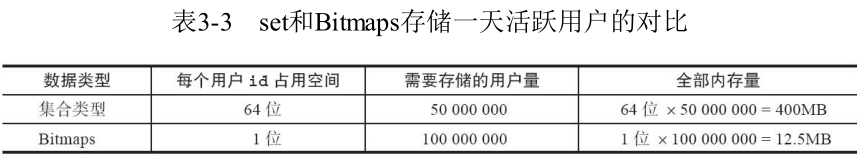
# 其他数据结构及功能

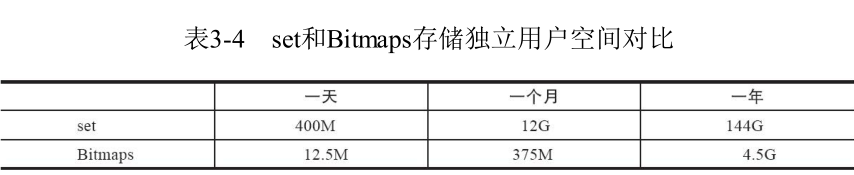
## BitMap

可以把Bitmaps想象成一个以**位**为单位的数组，数组的每个单元只能存储0和1，数组的下标在Bitmaps中叫做偏移量。



例子：统计用户活跃度功能：

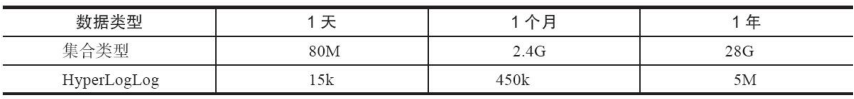




## TypeLogLog

HyperLogLog并不是一种新的数据结构（实际类型为字符串类型），而是一种**基数算法**，通过HyperLogLog可以利用极小的内存空间完成独立总数的统计。

例子：计算独立总数



*但是不是绝对正确的，Redis官方给出的数字是0.81%的失误率*。

## Pipeline

往返时间（Round Trip Time RTT）

1.发送命令

2.命令排队

3.命令执行

4.返回结果

解决一些没有提供批量操作的命令，减少RTT

与原生批量操作命令相比

原生批量命令是原子的，Pipeline是非原子的。

原生批量命令是一个命令对应多个key，Pipeline支持多个命令。

原生批量命令是Redis服务端支持实现的，而Pipeline需要服务端和客户端的共同实现。

## 事务与Lua

制作自己的专属原子命令，保证多条命令组合的原子性

操作过程

multi命令代表事务开始

exec命令代表事务结束

停止事务的执行，可以使用discard命令代替exec命令

# 8.缓存设计

## 缓存更新策略

内存淘汰策略

volatile-lru：从已设置过期时间的数据集中挑选最近最少使用的数据淘汰。

volatile-ttl：从已设置过期时间的数据集中挑选将要过期的数据淘汰。

volatile-random：从已设置过期时间的数据集中任意选择数据淘汰。

allkeys-lru：当内存不足以容纳新写入数据时移除最近最少使用的key。

allkeys-random：从数据集中任意选择数据淘汰。

no-enviction：当内存不足以容纳新写入数据时，新写入操作会报错。

## 缓存穿透

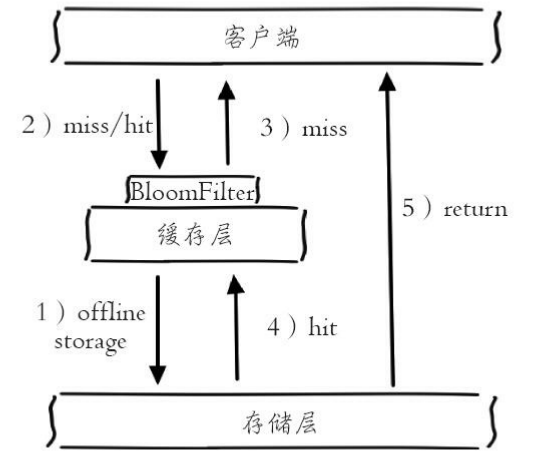
查询一个不存在的数据

解决方案

1.缓存空对象

2.布隆过滤器拦截





## 缓存雪崩

一批缓存因为某些原因突然消失，导致压力全部到了持久层

解决方案

1.缓存宕机

保证缓存层服务高可用性

2.依赖隔离组件为后端限流并降级

例如使用Hystrix

3.设置缓存不同的过期时间，防止同时过期

## 缓存击穿

一个热点Key突然消失，大量请求打到了数据库

解决方案

1.永不过期

2.互拆锁

只允许一个线程重建缓存，其他线程等待重建缓存的线程执行完，重新从缓存获取数据即可。

缓存雪崩和缓存击穿本质是同样的问题，只是量级有点不同。