数据类型

目录 Contents

数据存储类型介绍

业务数据的特殊性

1. 原始业务功能设计
    - 秒杀
    - 618活动
    - 双11活动
    - 排队购票
2. 运营平台监控到的突发高频访问数据
    - 突发时政要闻，被强势关注围观
3. 高频、复杂的统计数据
    - 在线人数
    - 投票排行榜

数据存储类型介绍

Redis 数据类型（5种常用）

- string
- hash
- list
- set
- sorted_set/zset（应用性较低）

数据存储类型介绍

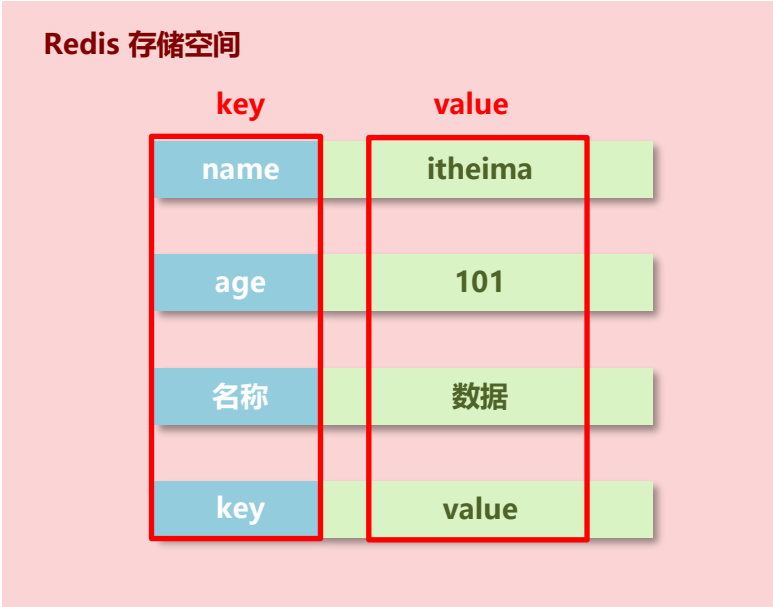
小节

- 业务数据
- Redis数据类型（5种）

目录 Contents

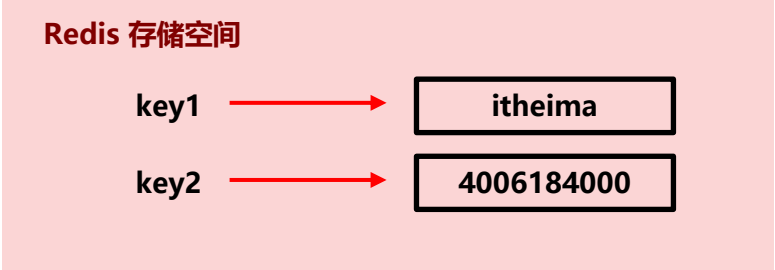
redis 数据存储格式

- redis 自身是一个 Map，其中所有的数据都是采用 key : value 的形式存储
- 数据类型指的是存储的数据的类型，也就是 value 部分的类型，key 部分永远都是字符串



string 类型

- 存储的数据：单个数据，最简单的数据存储类型，也是最常用的数据存储类型
- 存储数据的格式：一个存储空间保存一个数据
- 存储内容：通常使用字符串，如果字符串以整数的形式展示，可以作为数字操作使用



string 类型数据的基本操作

- 添加/修改数据

```
set key value
```

- 获取数据

```
get key
```

- 删除数据

```
del key
```

- 判定性添加数据

```
setnx key value
```

string 类型数据的基本操作

- 添加/修改多个数据

```
mset key1 value1 key2 value2 …
```

- 获取多个数据

Multiple[ˈmʌltɪpl]

```
mget key1 key2 …
```

- 获取数据字符个数（字符串长度）

```
strlen key
```

- 追加信息到原始信息后部（如果原始信息存在就追加，否则新建）

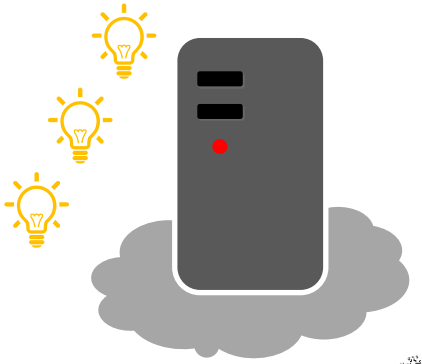
```
append key value
```

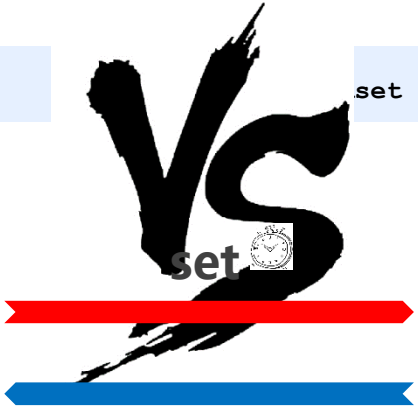
string 类型数据的基本操作

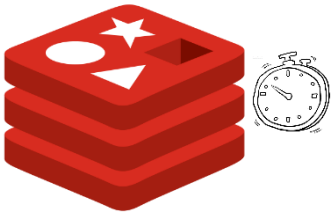
单数据操作与多数据操作的选择之惑

```
set key value
```

```
set key1 value1 key2 value2 …
```







result

单指令3条指令的执行过程 ×6 + ×3

多指令3条指令的执行过程 ×2 + ×3

string

小节

- string存储结构
- 数据操作
  - set
  - mset
  - del
  - setnx
  - append
- 查询操作
  - get
  - mget
  - strlen

string 类型数据的扩展操作

- 设置数值数据增加指定范围的值

```
incr key
incrby key increment
incrbyfloat key increment
```

- 设置数值数据减少指定范围的值

```
decr key
decrby key increment
```

- 设置数据具有指定的生命周期

```
setex key seconds value
psetex key milliseconds value
```

string 类型数据操作的注意事项

1. 数据操作不成功的反馈与数据正常操作之间的差异
   - ◆ 表示运行结果是否成功
     - (integer) 0 → false　　失败
     - (integer) 1 → true　　成功
   - ◆ 表示运行结果值
     - (integer) 3 → 3　　3个
     - (integer) 1 → 1　　1个
2. 数据未获取到时，对应的数据为（nil），等同于null
3. 数据最大存储量：512MB
4. string在redis内部存储默认就是一个字符串，当遇到增减类操作incr，decr时会转成数值型进行计算
5. 按数值进行操作的数据，如果原始数据不能转成数值，或超越了redis 数值上限范围，将报错

   9223372036854775807（java中Long型数据最大值，Long.MAX_VALUE）
6. redis所有的操作都是原子性的，采用单线程处理所有业务，命令是一个一个执行的，因此无需考虑并发带来的数据影响

小节

- 数值操作
  - incr
  - incrby
  - incrbyfloat
  - desc
  - descby
- 时效性操作
  - setex
  - psetex
- 注意事项

应用场景

主页高频访问信息显示控制，例如新浪微博大V主页显示粉丝数与微博数量

解决方案

- 在redis中为大V用户设定用户信息，以用户主键和属性值作为key，后台设定定时刷新策略即可

  eg: user:id:3506728370:fans → 12210947

  eg: user:id:3506728370:blogs → 6164

  eg: user:id:3506728370:focuses → 83

- 也可以使用json格式保存数据

  eg: user:id:3506728370 → {“fans”：12210947，“blogs”：6164，“focuses”：83}

key 的设置约定

- 数据库中的热点数据key命名惯例

	表名	主键名	主键值	字段名
eg1：	order	id	29437595	name
eg2：	equip	id	390472345	type
eg3：	news	id	202004150	title

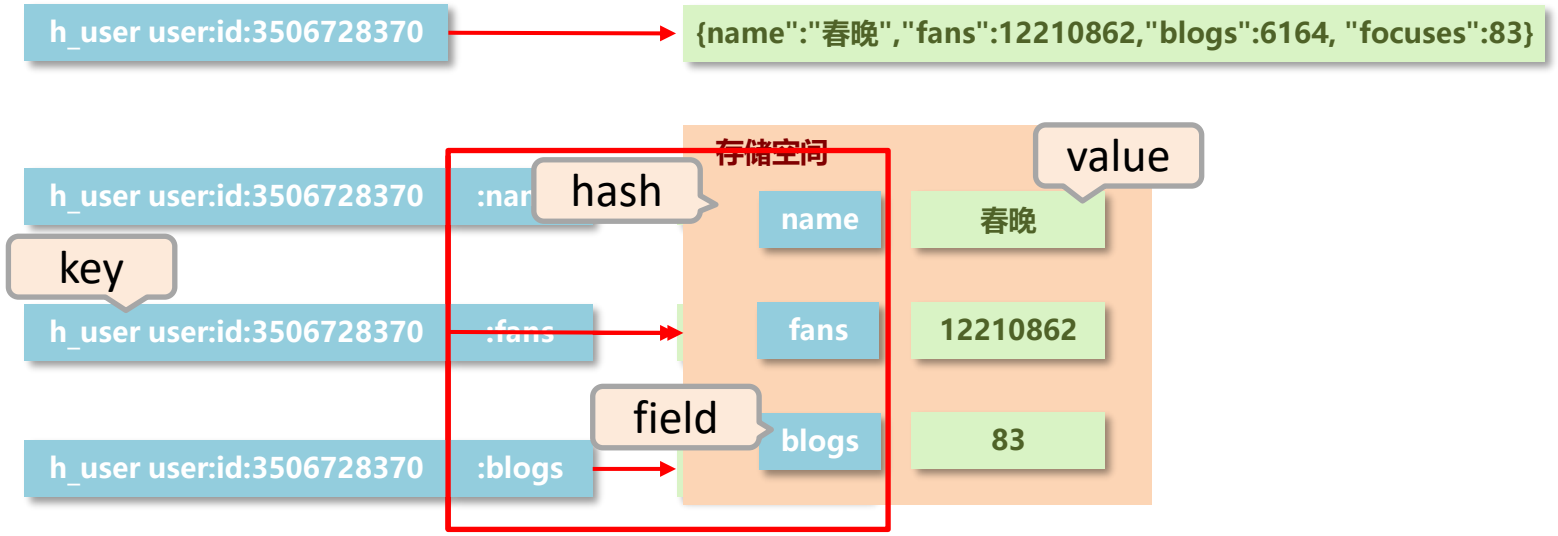
小节

- string类型数据的应用场景
- key命名习惯

目录 Contents

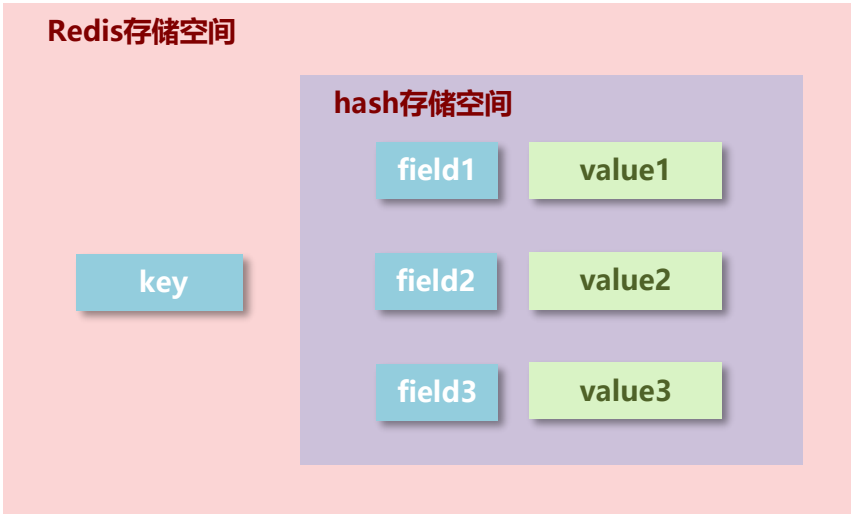
数据存储的困惑

对象类数据的存储如果具有较频繁的更新需求操作会显得笨重



hash 类型

- 新的存储需求：对一系列存储的数据进行编组，方便管理，典型应用存储对象信息
- 需要的存储结构：一个存储空间保存多个键值对数据
- hash类型：底层使用哈希表结构实现数据存储



hash存储结构优化

- 如果field数量较少，存储结构优化为类数组结构
- 如果field数量较多，存储结构使用HashMap结构

hash 类型数据的基本操作

- 添加/修改数据

```
hset key field value
```

- 获取数据

```
hget key field
hgetall key
```

- 删除数据

```
hdel key field1 [field2]
```

- 设置field的值，如果该field存在则不做任何操作

```
hsetnx key field value
```

hash 类型数据的基本操作

- 添加/修改多个数据

```
hmset key field1 value1 field2 value2 …
```

- 获取多个数据

```
hmget key field1 field2 …
```

- 获取哈希表中字段的数量

```
hlen key
```

- 获取哈希表中是否存在指定的字段

```
hexists key field
```

hash

小节

- hash存储结构
- 数据操作
  - hset
  - hmset
  - hdel
  - hsetnx
- 查询操作
  - hget
  - hmget
  - hgetall
  - hlen
  - hexists

hash 类型数据扩展操作

- 获取哈希表中所有的字段名或字段值

```
hkeys key
hvals key
```

- 设置指定字段的数值数据增加指定范围的值

```
hincrby key field increment
hincrbyfloat key field increment
```

hash 类型数据操作的注意事项

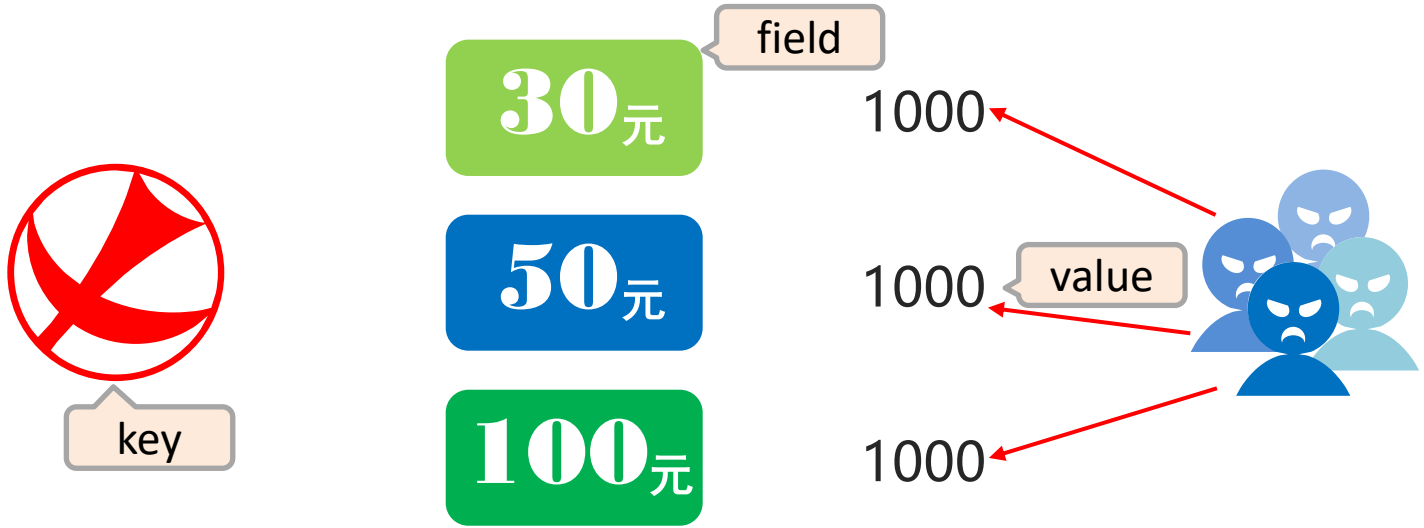
1. hash类型中value只能存储字符串，不允许存储其他数据类型，不存在嵌套现象。如果数据未获取到，对应的值为（nil）
2. 每个 hash 可以存储 2^{32} - 1 个键值对
3. hash类型十分贴近对象的数据存储形式，并且可以灵活添加删除对象属性。但hash设计初衷不是为了存储大量对象而设计的，切记不可滥用，更不可以将hash作为对象列表使用
4. hgetall 操作可以获取全部属性，如果内部field过多，遍历整体数据效率就很会低，有可能成为数据访问瓶颈

小节

- 数值操作
  - hincrby
  - hincrbyfloat
- 特殊查询操作
  - hkeys
  - hvals
- 注意事项

应用场景

双11活动日，销售手机充值卡的商家对移动、联通、电信的30元、50元、100元商品推出抢购活动，每种商品抢购上限1000张



解决方案

- 以商家id作为key
- 将参与抢购的商品id作为field
- 将参与抢购的商品数量作为对应的value
- 抢购时使用降值的方式控制产品数量

注意：实际业务中还有超卖等实际问题，这里不做讨论

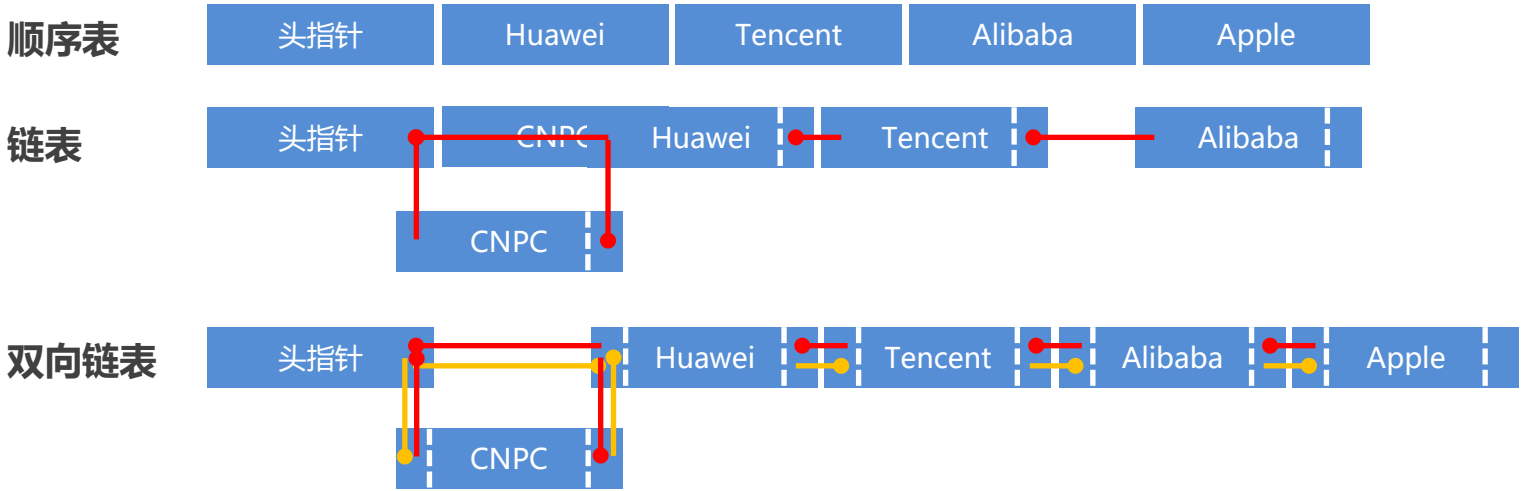
小节

- hash类型数据的应用场景

目录 Contents

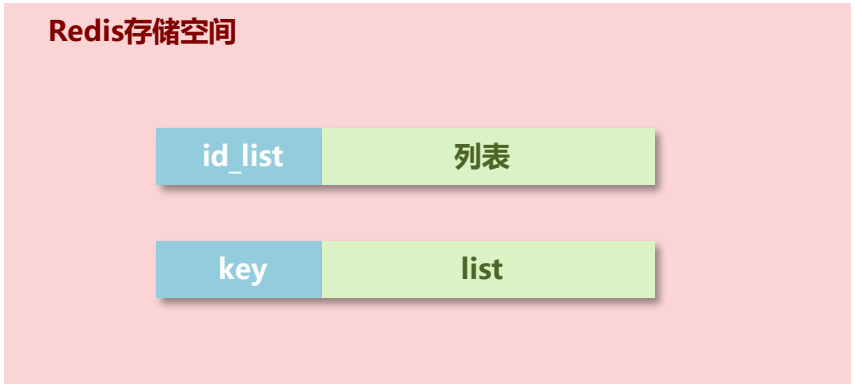
list 类型

- 数据存储需求：存储多个数据，并对数据进入存储空间的顺序进行区分
- 需要的存储结构：一个存储空间保存多个数据，且通过数据可以体现进入顺序
- list类型：保存多个数据，底层使用双向链表存储结构实现



list 类型

- 数据存储需求：存储多个数据，并对数据进入存储空间的顺序进行区分
- 需要的存储结构：一个存储空间保存多个数据，且通过数据可以体现进入顺序
- list类型：保存多个数据，底层使用双向链表存储结构实现



list 类型数据基本操作

- 添加/修改数据

```
lpush key value1 [value2] ……
rpush key value1 [value2] ……
```

- 获取数据

```
lrange key start stop
lindex key index
llen key
```

- 获取并移除数据

```
lpop key
rpop key
```

小节

- list存储结构
- 数据操作
  - lpush
  - rpush
  - lpop
  - rpop
- 查询信息
  - lrange
  - lindex
  - llen

list 类型数据扩展操作

- 移除指定数据

```
lrem key count value
```

- 规定时间内获取并移除数据

```
blpop key1 [key2] timeout
brpop key1 [key2] timeout
brpoplpush source destination timeout
```

list 类型数据操作注意事项

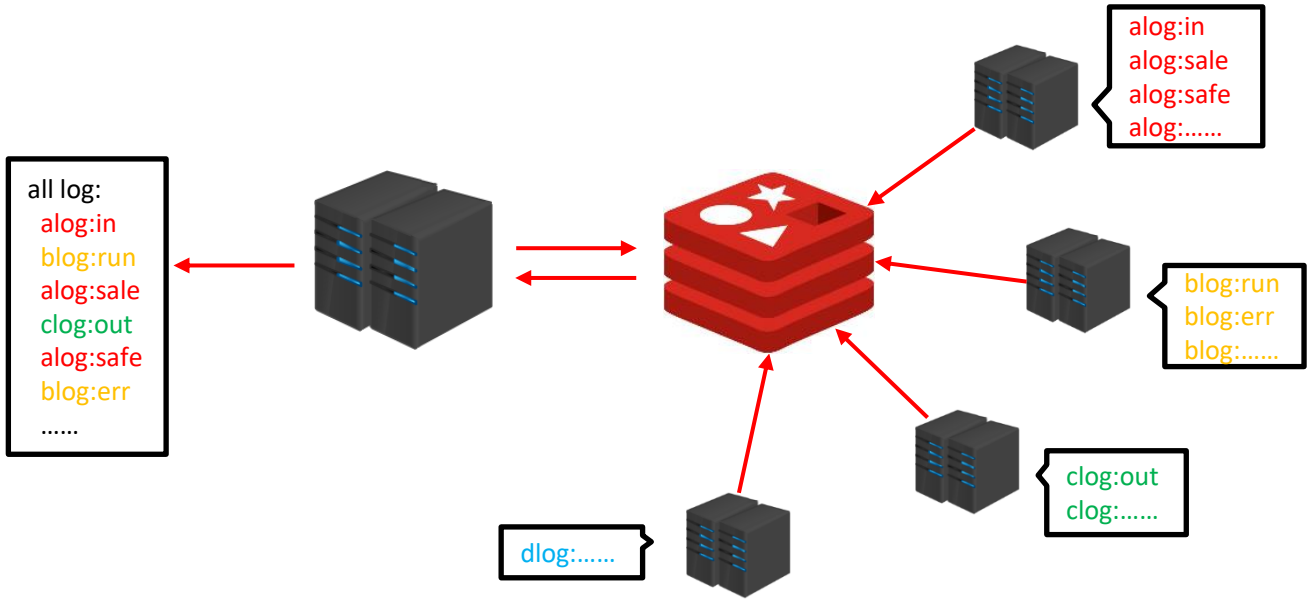
1. list中保存的数据都是string类型的，数据总容量是有限的，最多$2^{32}-1$ 个元素 (4294967295)。
2. list具有索引的概念，但是操作数据时通常以队列的形式进行入队出队操作，或以栈的形式进行入栈出栈操作
3. 获取全部数据操作结束索引设置为-1
4. list可以对数据进行分页操作，通常第一页的信息来自于list，第2页及更多的信息通过数据库的形式加载

小节

- 数据操作
  - lrem
  - blpop
  - brpop
  - brpoppush
- 注意事项

应用场景

企业运营过程中，系统将产生出大量的运营数据，如何保障多台服务器操作日志的统一顺序输出？



解决方案

- 依赖list的数据具有顺序的特征对信息进行管理
- 使用队列模型解决多路信息汇总合并的问题
- 使用栈模型解决最新消息的问题

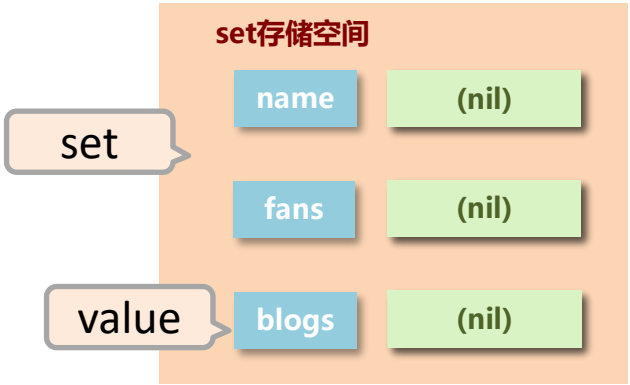
小节

- list类型数据的应用场景

目录 Contents

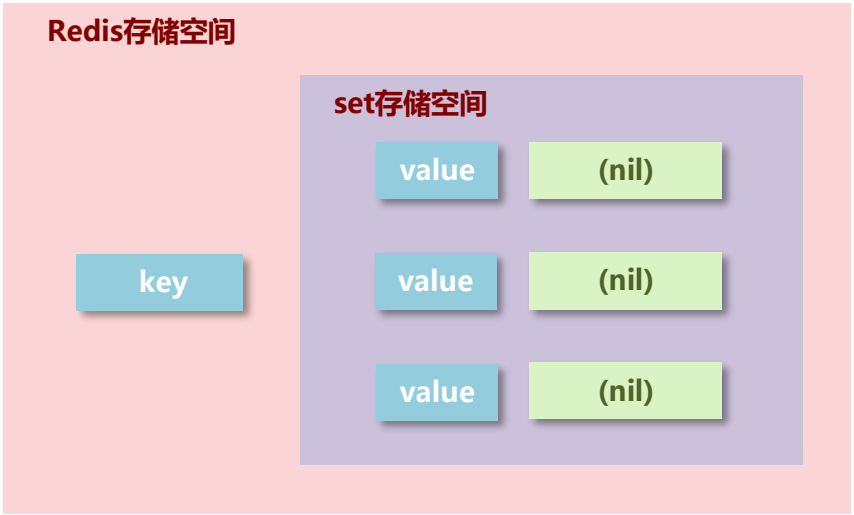
set 类型

- 新的存储需求：存储大量的数据，在查询方面提供更高的效率
- 需要的存储结构：能够保存大量的数据，高效的内部存储机制，便于查询
- set类型：与hash存储结构完全相同，仅存储键，不存储值（nil），并且值是不允许重复的



set 类型

- 新的存储需求：存储大量的数据，在查询方面提供更高的效率
- 需要的存储结构：能够保存大量的数据，高效的内部存储机制，便于查询
- set类型：与hash存储结构完全相同，仅存储键，不存储值（nil），并且值是不允许重复的



set 类型数据的基本操作

- 添加数据

```
sadd key member1 [member2]
```

- 获取全部数据

```
smembers key
```

- 删除数据

```
srem key member1 [member2]
```

set 类型数据的基本操作

- 获取集合数据总量

```
scard key
```

- 判断集合中是否包含指定数据

```
sismember key member
```

- 随机获取集合中指定数量的数据

```
srandmember key [count]
```

- 随机获取集合中的某个数据并将该数据移出集合

```
spop key [count]
```

小节

- set存储结构
- 数据操作
  - sad
  - srem
  - spop
- 查询操作
  - smember
  - scard
  - sismember
  - srandmember

set 类型数据的扩展操作

- 求两个集合的交、并、差集

```
sinter key1 [key2 …]
sunion key1 [key2 …]
sdiff key1 [key2 …]
```

- 求两个集合的交、并、差集并存储到指定集合中

```
sinterstore destination key1 [key2 …]
sunionstore destination key1 [key2 …]
sdiffstore destination key1 [key2 …]
```

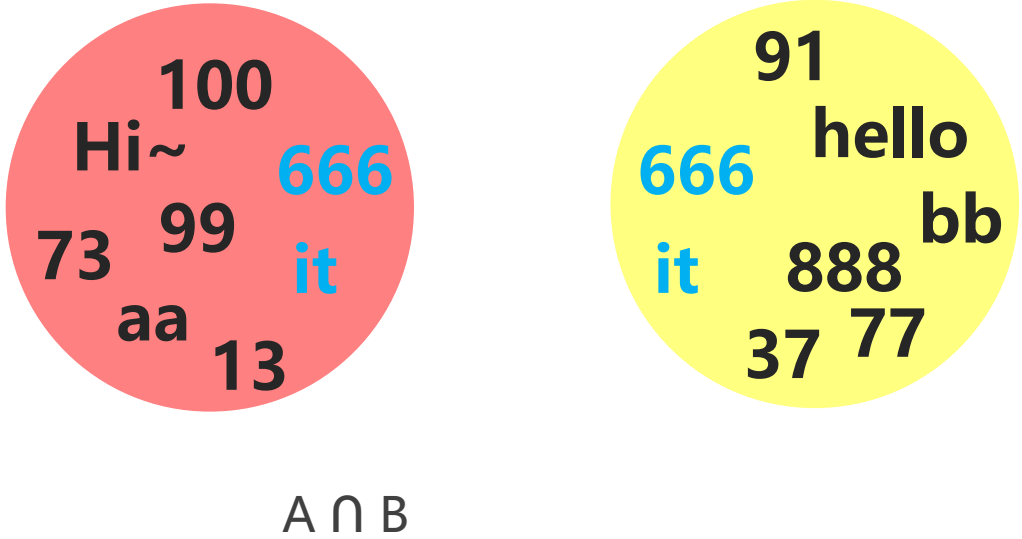
- 将指定数据从原始集合中移动到目标集合中

```
smove source destination member
```

set 类型数据的扩展操作

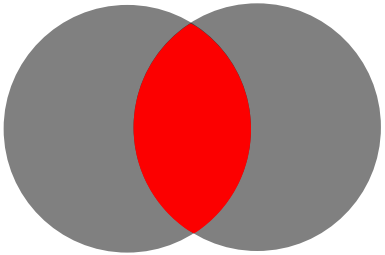
set 类型数据的扩展操作

set 类型数据的扩展操作

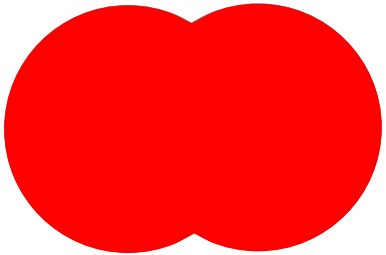


A ∩ B

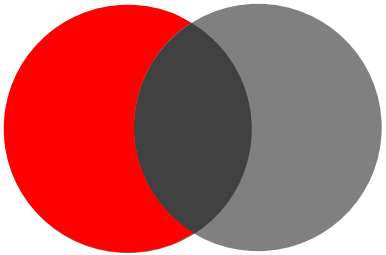
set 类型数据的扩展操作



A ∩ B



A ∪ B



A - B

set 类型数据的扩展操作

- 求两个集合的交、并、差集

```
sinter key1 [key2 …]
sunion key1 [key2 …]
sdiff key1 [key2 …]
```

- 求两个集合的交、并、差集并存储到指定集合中

```
sinterstore destination key1 [key2 …]
sunionstore destination key1 [key2 …]
sdiffstore destination key1 [key2 …]
```

- 将指定数据从原始集合中移动到目标集合中

```
smove source destination member
```

set 类型数据操作的注意事项

- set 类型不允许数据重复，如果添加的数据在 set 中已经存在，将只保留一份
- set 虽然与hash的存储结构相同，但是无法启用hash中存储值的空间

小节

- 集合操作
  - sinter
  - sunion
  - sdiff
  - sinterstore
  - sunionstore
  - sdiffstore
  - smove
- 注意事项

应用场景

黑名单

资讯类信息类网站追求高访问量，但是由于其信息的价值，往往容易被不法分子利用，通过爬虫技术，快速获取信息，个别特种行业网站信息通过爬虫获取分析后，可以转换成商业机密进行出售。例如第三方火车票、机票、酒店刷票代购软件，电商刷评论、刷好评。

同时爬虫带来的伪流量也会给经营者带来错觉，产生错误的决策，有效避免网站被爬虫反复爬取成为每个网站都要考虑的基本问题。在基于技术层面区分出爬虫用户后，需要将此类用户进行有效的屏蔽，这就是**黑名单**的典型应用。

ps:不是说爬虫一定做摧毁性的工作，有些小型网站需要爬虫为其带来一些流量。

白名单

对于安全性更高的应用访问，仅仅靠黑名单是不能解决安全问题的，此时需要设定可访问的用户群体，依赖**白名单**做更为苛刻的访问验证。

解决方案

- 基于经营战略设定问题用户发现、鉴别规则
- 周期性更新满足规则的用户黑名单，加入set集合
- 用户行为信息达到后与黑名单进行比对，确认行为去向
- 黑名单过滤IP地址：应用于开放游客访问权限的信息源
- 黑名单过滤设备信息：应用于限定访问设备的信息源
- 黑名单过滤用户：应用于基于访问权限的信息源

小节

- set类型数据的应用场景

目录 Contents

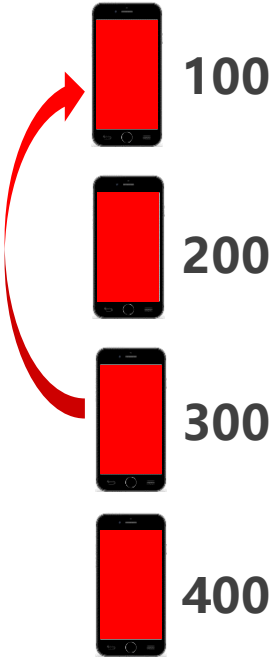
数据类型实践案例

业务场景

使用微信的过程中，当微信接收消息后，会默认将最近接收的消息置顶，当多个好友及关注的订阅号同时发送消息时，该排序会不停的进行交替。同时还可以将重要的会话设置为置顶。一旦用户离线后，再次打开微信时，消息该按照什么样的顺序显示?

数据类型实践案例

业务分析



300发送消息

400发送消息

200发送消息

200发送消息

300发送消息

数据类型实践案例

解决方案

- 依赖list的数据具有顺序的特征对消息进行管理，将list结构作为栈使用
- 对置顶与普通会话分别创建独立的list分别管理
- 当某个list中接收到用户消息后，将消息发送方的id从list的一侧加入list（此处设定左侧）
- 多个相同id发出的消息反复入栈会出现问题，在入栈之前无论是否具有当前id对应的消息，先删除对应id
- 推送消息时先推送置顶会话list，再推送普通会话list，推送完成的list清除所有数据
- 消息的数量，也就是微信用户对话数量采用计数器的思想另行记录，伴随list操作同步更新

数据类型实践案例

小节

- 数据类型实践案例

数据类型

1. string
2. hash
3. list
4. set
5. 数据类型实践案例

传智播客旗下高端IT教育品牌