

Redis应用十讲

《Redis应用实例》配套课程

黄健宏



本课程为《Redis应用实例》一书的配套课程，全课程共包含10节课：

1. 使用缓存提高访问数据的速度
2. 使用锁保证重要资源的独占使用权
3. 使用先进先出队列解决抢购和秒杀问题
4. 使用简单计数器和唯一计数器进行计数
5. 使用排行榜对元素进行有序排列
6. 使用自动补全为用户提供输入建议
7. 构建类似Stack Overflow等网站的投票系统
8. 使用分页和时间线排列并管理大量元素
9. 使用社交关系程序存储用户间的社交关系
10. 使用地理位置程序记录用户的位置信息



黄健宏

计算机技术图书作者、译者，关注的领域包括数据库、编程语言、操作系统以及算法和数据结构等。

从Redis 2.2版本开始关注Redis，翻译过Redis的文档，注释过Redis 2.6和3.0的源代码，发表了大量关于Redis的文章和多部Redis相关图书。

著有《Redis应用实例》《Redis设计与实现》《Redis使用手册》。

译有《SQL实战（第2版）》《Redis实战》《Go语言趣学指南》《Go Web编程》。

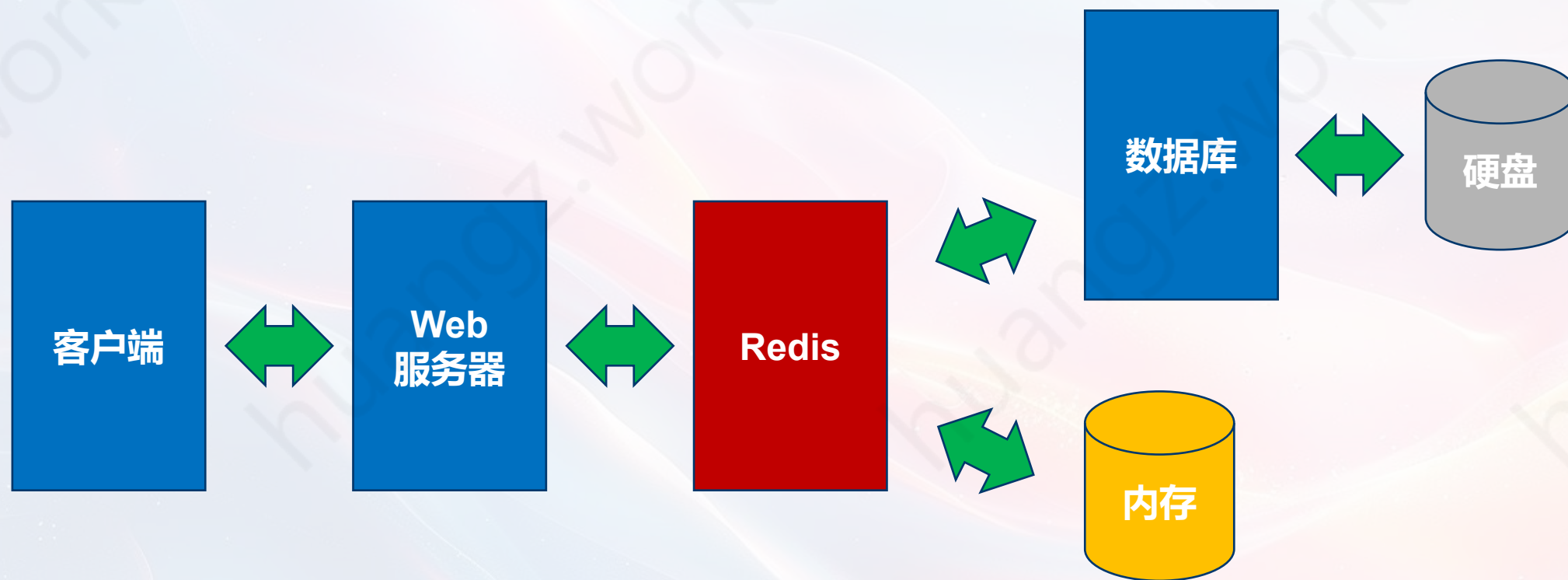


第1讲 使用缓存提高 访问数据的速度

传统数据库使用硬盘存储数据，它的问题是——不够快



Redis将数据存储在内存中，它可以用作传统数据库的缓存



使用Redis作为缓存的多种方式

Redis为用户提供了丰富的数据结构，比如字符串、哈希、列表、有序集合等等。

使用Redis作为缓存的多种方式

Redis为用户提供了丰富的数据结构，比如字符串、哈希、列表、有序集合等等。

不同的数据结构可以用于缓存不同的数据，比如：

- 字符串可以用作HTML页面的缓存；
- 哈希可以用作SQL表数据的缓存，又或者键值对文档的缓存；
- 列表可以用作博客文章列表、微博消息列表、新闻消息列表的缓存；
- 有序集合可以用作排行榜的缓存；
- 诸如此类。

使用Redis作为缓存的多种方式

Redis为用户提供了丰富的数据结构，比如字符串、哈希、列表、有序集合等等。

不同的数据结构可以用于缓存不同的数据，比如：

- 字符串可以用作HTML页面的缓存；
- 哈希可以用作SQL表数据的缓存，又或者键值对文档的缓存；
- 列表可以用作博客文章列表、微博消息列表、新闻消息列表的缓存；
- 有序集合可以用作排行榜的缓存；
- 诸如此类。

在这一课，我们将要学习的是如何实现最基本的缓存方式：

1. 缓存单项文本数据；
2. 缓存多项文本数据。

方法：使用字符串键缓存单项文本数据

方法：使用字符串键缓存单项文本数据

设置缓存：

SET <key> <value>

方法：使用字符串键缓存单项文本数据

设置缓存：

```
SET <key> <value>
```

设置缓存和过期时间：

```
SET <key> <value> EX <ttl>
```

```
SET <key> <value> PX <ttl>
```


方法：使用字符串键缓存单项文本数据

设置缓存：

SET <key> <value>

设置缓存和过期时间：

SET <key> <value> EX <ttl>

SET <key> <value> PX <ttl>

获取缓存：

GET <key>

方法：使用字符串键缓存单项文本数据

设置缓存：

SET <key> <value>

设置缓存和过期时间：

SET <key> <value> EX <ttd>

SET <key> <value> PX <ttd>

获取缓存：

GET <key>

删除缓存：

DEL <key>

实例：使用字符串键缓存HTML页面

实例：使用字符串键缓存HTML页面

Web框架为了生成客户端所需的HTML页面，往往需要完成大量工作：

1. Web框架接收客户端发送的请求；
2. 框架通过路由器将请求交给指定处理器进行处理；
3. 处理器访问数据库以获取用户所需的动态数据；
4. 处理器将所得的动态数据与模板文件结合，创建出HTML页面，然后将该页面返回给发送请求的客户端。

实例：使用字符串键缓存HTML页面

Web框架为了生成客户端所需的HTML页面，往往需要完成大量工作：

1. Web框架接收客户端发送的请求；
2. 框架通过路由器将请求交给指定处理器进行处理；
3. 处理器访问数据库以获取用户所需的动态数据；
4. 处理器将所得的动态数据与模板文件结合，创建出HTML页面，然后将该页面返回给发送请求的客户端。

为了加速上述过程、避免资源重复浪费，可以将框架生成的HTML页面缓存起来，从而在一段时间内避免重复执行相同的操作：

```
SET page:index "<html>...</html>" EX 300
```

方法：使用哈希键缓存多项文本数据

方法：使用哈希键缓存多项文本数据

设置缓存：

```
HSET <hash> <field1> <value1> <field2> <value2> ...
```

方法：使用哈希键缓存多项文本数据

设置缓存：

```
HSET <hash> <field1> <value1> <field2> <value2> ...
```

为缓存设置过期时间：

```
EXPIRE <hash> <ttd>
```

```
PEXPIRE <hash> <ttd>
```


方法：使用哈希键缓存多项文本数据

设置缓存：

```
HSET <hash> <field1> <value1> <field2> <value2> ...
```

为缓存设置过期时间：

```
EXPIRE <hash> <ttd>
```

```
PEXPIRE <hash> <ttd>
```

获取整个缓存：

```
HGETALL <hash>
```

方法：使用哈希键缓存多项文本数据

设置缓存：

```
HSET <hash> <field1> <value1> <field2> <value2> ...
```

为缓存设置过期时间：

```
EXPIRE <hash> <ttd>
```

```
PEXPIRE <hash> <ttd>
```

获取整个缓存：

```
HGETALL <hash>
```

获取缓存中的部分数据：

```
HMGET <hash> <field1> <field2> <field3> ...
```


方法：使用哈希键缓存多项文本数据

设置缓存：

```
HSET <hash> <field1> <value1> <field2> <value2> ...
```

为缓存设置过期时间：

```
EXPIRE <hash> <ttd>
```

```
PEXPIRE <hash> <ttd>
```

获取整个缓存：

```
HGETALL <hash>
```

获取缓存中的部分数据：

```
HMGET <hash> <field1> <field2> <field3> ...
```

删除缓存：

```
DEL <hash>
```

实例：使用哈希键缓存表数据或键值对文档

实例：使用哈希键缓存表数据或键值对文档

通过使用Redis的哈希键，可以很容易地缓存SQL数据库中的表或者文档数据库中的键值对文档。

实例：使用哈希键缓存SQL表数据或键值对文档

通过使用Redis的哈希键，可以很容易地缓存SQL数据库中的表或者文档数据库中的键值对文档。

比如说，为了缓存student表中的以下行数据：

属性（键）	值
id	10086
name	“peter”
age	20
gender	“male”

实例：使用哈希键缓存SQL表数据或键值对文档

通过使用Redis的哈希键，可以很容易地缓存SQL数据库中的表或者文档数据库中的键值对文档。

比如说，为了缓存student表中的以下行数据：

属性（键）	值
id	10086
name	“peter”
age	20
gender	“male”

可以使用以下代码：

```
HSET student:10086 id 10086 name “peter” age 20 gender “male”
```


缓存程序的Python实现

关联章：

- 第1章 “缓存文本数据”
- 第2章 “缓存二进制数据”

代码实现：

- 第1章分别使用Redis的字符串键和哈希键实现了用于缓存文本数据的Cache类、JsonCache类和HashCache类，它们都具有设置缓存的set()方法和获取缓存的get()方法；
- 第2章展示了如何使用BinaryCache类来缓存图片、视频、音频等二进制数据，这个类同样具有set()、get()两个方法。

获取代码：

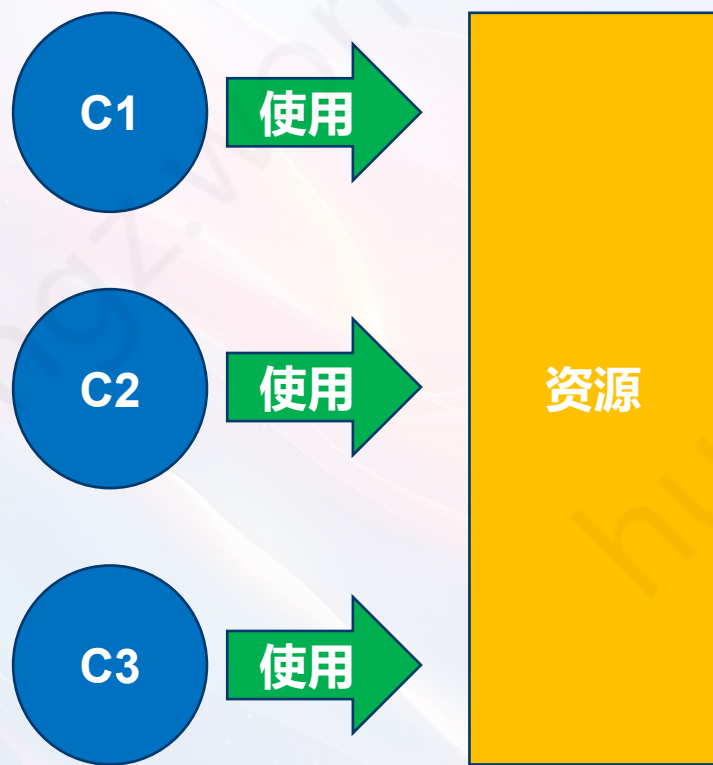
```
git clone git@github.com:huangzworks/rediscookbook.git
```



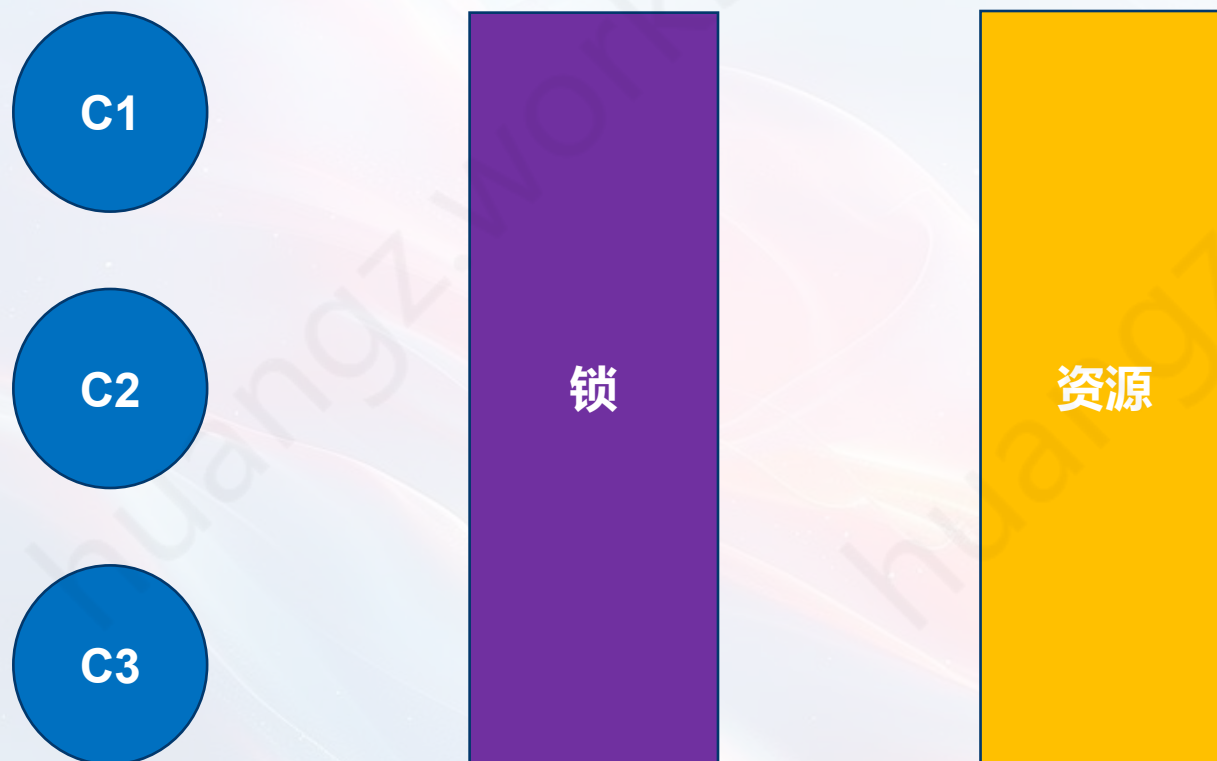
第2讲

使用锁保证重要资源 的独占使用权

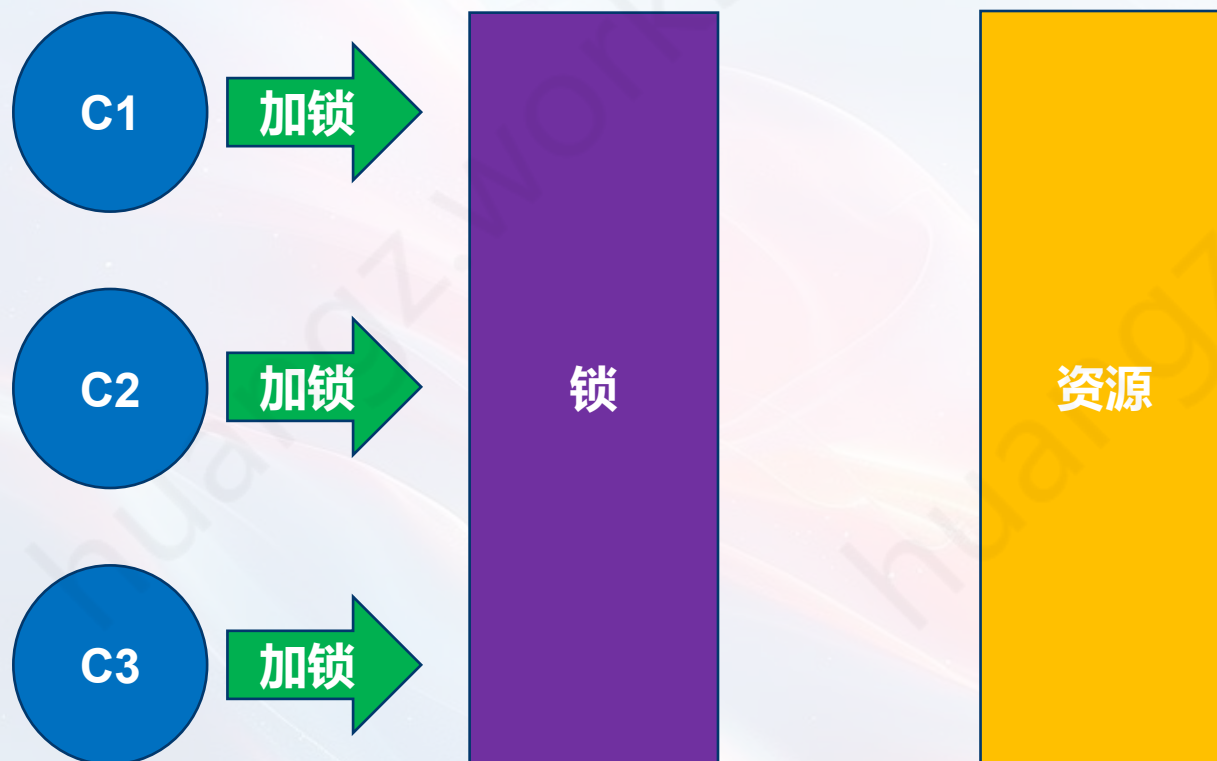
案例：多个客户端争抢使用唯一的资源



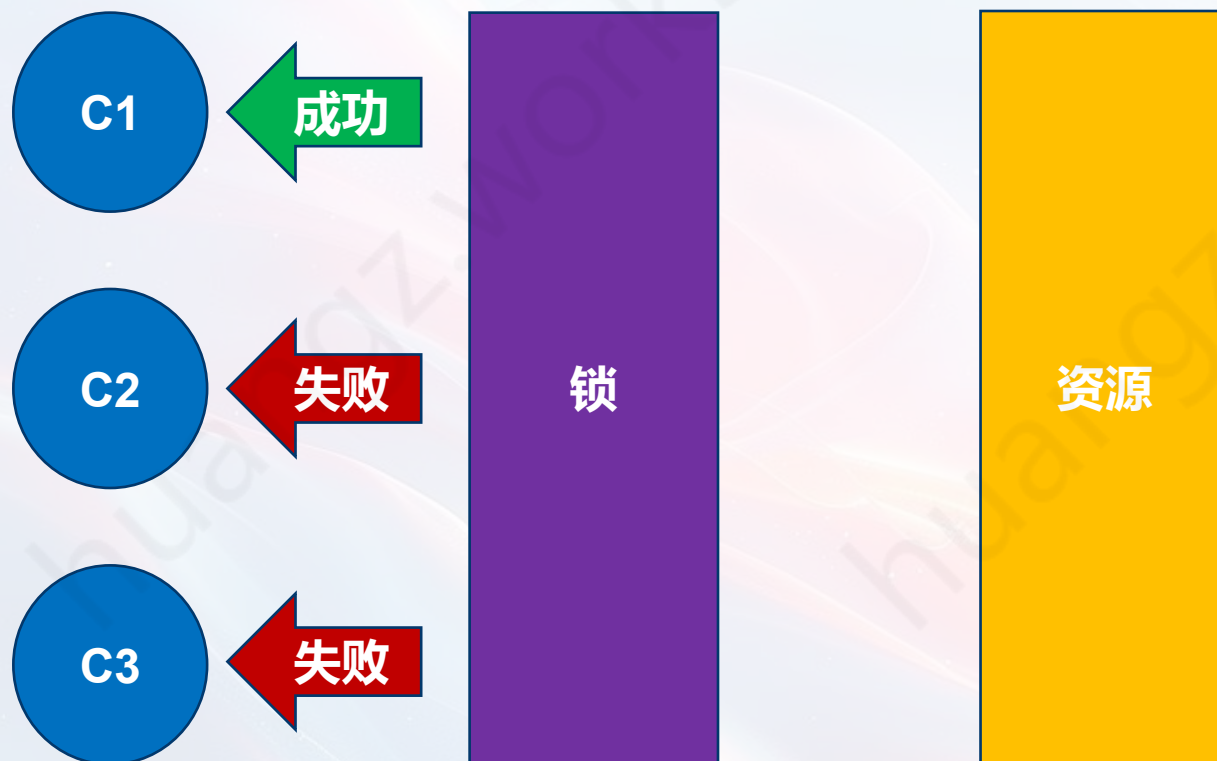
使用锁确保资源的独占使用权



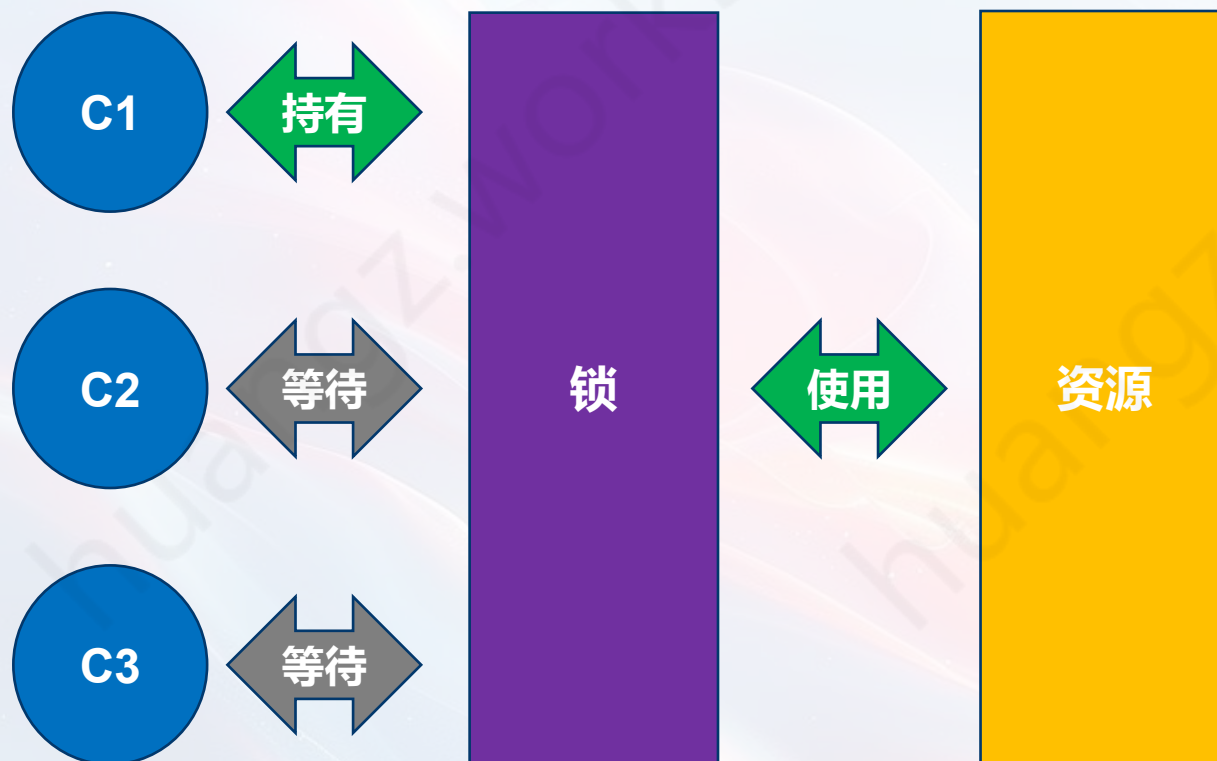
使用锁确保资源的独占使用权



使用锁确保资源的独占使用权



使用锁确保资源的独占使用权



方法：使用字符串键构建锁

方法：使用字符串键构建锁

加锁：

SET <key> “” NX

方法：使用字符串键构建锁

加锁：

SET <key> "" NX

加锁并设置解锁密码：

SET <key> <pswd> NX

方法：使用字符串键构建锁

加锁：

SET <key> "" NX

加锁并设置解锁密码：

SET <key> <pswd> NX

加锁并设置解锁密码和最大加锁时长（超时自动释放）：

SET <key> <pswd> NX EX <sec>

SET <key> <pswd> NX PX <ms>

方法：使用字符串键构建锁

加锁：

SET <key> "" NX

加锁并设置解锁密码：

SET <key> <pswd> NX

加锁并设置解锁密码和最大加锁时长（超时自动解锁）：

SET <key> <pswd> NX EX <sec>

SET <key> <pswd> NX PX <ms>

解锁：

DEL <key>

（如果是带密码的锁，那么在尝试解锁时需要使用事务或脚本来保证锁键不会在进行密码比对的过程中被其他客户端修改）

锁程序的Python实现

关联章：

- 第3章 “锁”
- 第4章 “带密码保护的锁”

代码实现：

- 第3章展示了基本的锁实现Lock类和带自动解锁功能的锁AutoReleaseLock类：
 - 这两者都带有加锁方法acquire()和解锁方法release();
 - AutoReleaseLock类的acquire()方法允许在加锁的同时设置最大加锁时长及其单位;
- 第4章展示了带密码保护功能的锁IdentityLock类：
 - 它的加锁方法acquire()和解锁方法release() 都接受一个密码作为参数，用作加锁和解锁时的密码。

获取代码：

```
git clone git@github.com:huangzworks/rediscookbook.git
```



第3讲 使用先进先出队列 解决抢购和秒杀问题

车票抢购和商品秒杀

<

北京 <> 上海

...

今天
08.23

周六
08.24

周日
08.25

周一
08.26

周二
08.27

周三
08.28

日历

直达

中转

☐只看高铁/动车

☐只看普通车

☐只看有票

筛选

北京南

北京

上海虹桥

上海

复兴号
高铁动车

06:10

G101

12:09

售罄

北京南

5时59分

上海虹桥

二等 候补

一等 候补

商务 候补

复兴号

06:20

G103

11:58

¥1035起

北京南

5时38分

上海虹桥

二等 候补

一等 1张

商务 15张

复兴号
高铁动车

07:00

G1

11:29

¥2331起

北京南

4时29分

上海

二等 候补

一等 候补

包廂软座 候补

商务 8张

01:20:46

22:00
即将开抢

00:00
即将开抢

08:00
即将开抢

1

精选

黑神话

家居家电

家庭清

专享价

维达 超韧手帕纸 4层18包

自营 1万+回头客 30天价保

¥6.9 6.97折 日常价 ¥9.9

已抢73%

抢

奥利奥 小饼干 55g*3杯

自营 1万+回头客 30天价保

¥9.9 7.68折 日常价 ¥12.9

已抢80%

抢

医用棉签100支

包邮 2千人加购 7天无理由退货

¥1.95 6.48折 日常价 ¥3.01

已抢46%

抢

白象刀削面 挂面

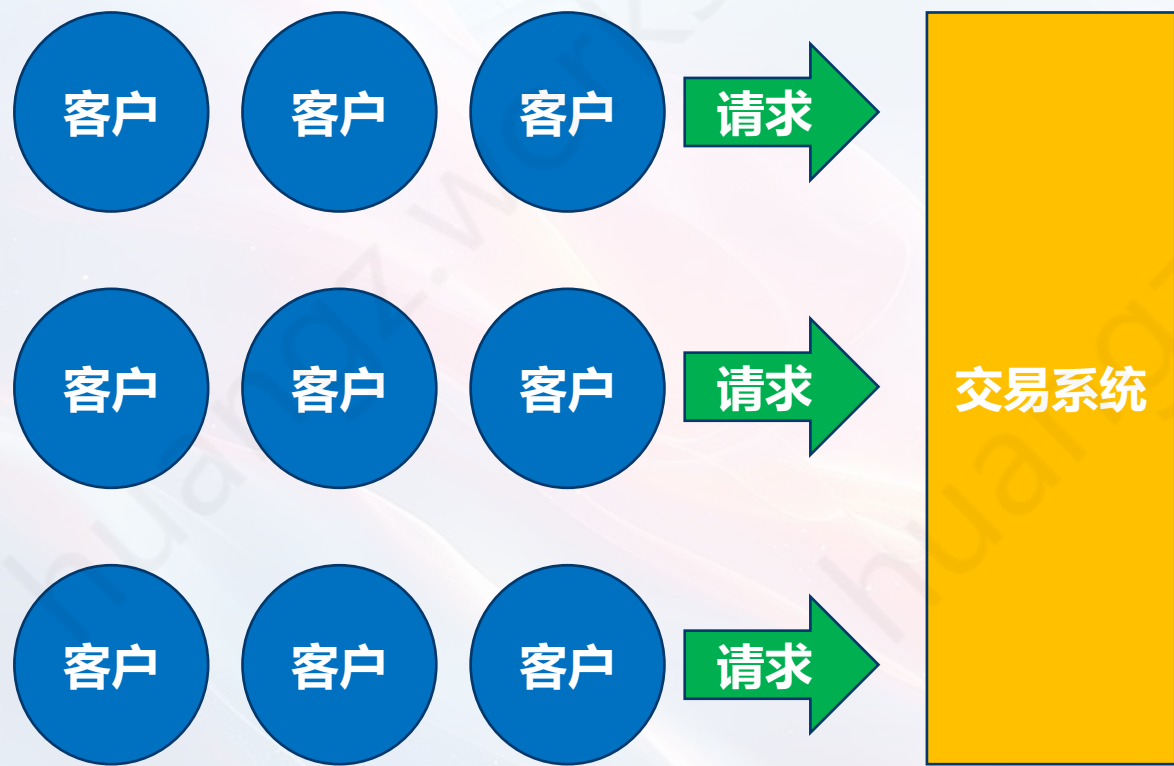
自营 30天价保 1万人加购

¥14.9 319天最低价

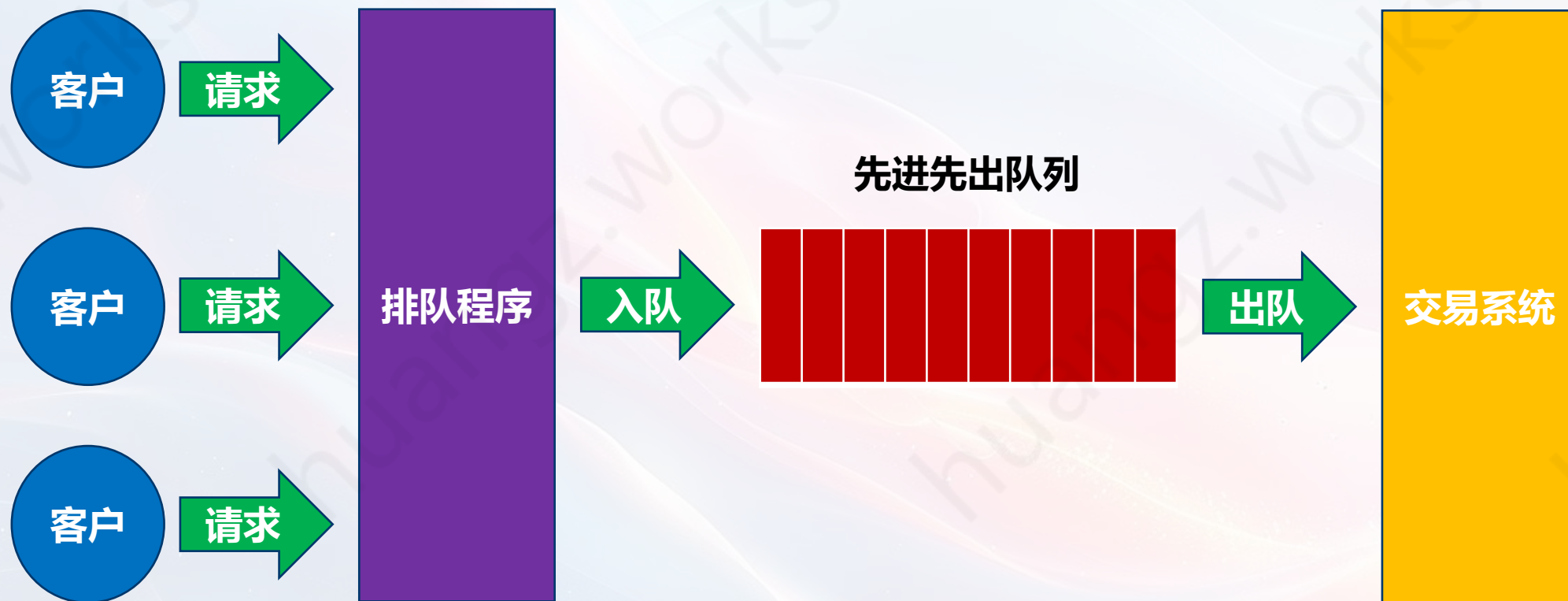
已抢75%

抢

大量客户同时发送请求，导致系统拥塞甚至崩溃



将请求放入到先进先出队列中，然后再一一交给系统处理



方法：使用Redis列表构建先进先出队列

方法：使用Redis列表构建先进先出队列

入队：

RPUSH <queue> <item>

方法：使用Redis列表构建先进先出队列

入队：

RPUSH <queue> <item>

出队：

LPOP <queue>

方法：使用Redis列表构建先进先出队列

入队：

RPUSH <queue> <item>

出队：

LPOP <queue>

查看队列长度：

LLEN <queue>

通过阻塞出队操作更高效地处理出队元素

```
while True:  
    item = queue.dequeue()  
    if item is not None:  
        handle(item)  
    else:  
        sleep(sometime)
```

通过阻塞出队操作更高效地处理出队元素

```
while True:  
    item = queue.dequeue()  
    if item is not None:  
        handle(item)  
    else:  
        sleep(sometime)
```

阻塞出队:

```
BLPOP <queue> <timeout>
```

通过阻塞出队操作更高效地处理出队元素

```
while True:
    item = queue.dequeue()
    if item is not None:
        handle(item)
    else:
        sleep(sometime)
```

阻塞出队:

BLPOP <queue> <timeout>

```
while True:
    item = queue.dequeue(0) # 阻塞并等待可处理的新元素出现
    handle(item)
```


先进先出队列程序的Python实现

关联章：

- 第26章 “先进先出队列”
- 第27章 “定长队列和淘汰队列”

代码实现：

- 第26章展示了先进先出队列FifoQueue类，它提供了：
 - 将一个或多个元素入队的enqueue()方法；
 - 将元素出队的dequeue()方法，阻塞功能可选；
 - 查看队列长度的length()方法。
- 第27章展示了带有固定长度限制功能的先进先出队列FixedLengthQueue类，还有自动淘汰旧元素并保留新元素的先进先出队列FadedQueue类。

获取代码：

```
git clone git@github.com:huangzworks/rediscookbook.git
```



第4讲

使用简单计数器和唯一计数器 进行计数

简单计数器



构建简单计数器



构建简单计数器

使用字符串键实现

增加计数:

INCRBY <counter> <n>

减少计数:

DECRBY <counter> <n>

获取当前计数值:

GET <counter>

重置计数:

SET <counter> <n> GET

构建简单计数器

使用字符串键实现

增加计数:

INCRBY <counter> <n>

减少计数:

DECRBY <counter> <n>

获取当前计数值:

GET <counter>

重置计数:

SET <counter> <n> GET

使用哈希键实现

增加计数:

HINCRBY <hash> <counter> <n>

减少计数:

HINCRBY <hash> <counter> **<-n>**

获取当前计数值:

HGET <hash> <counter>

重置计数:

MULTI

HGET <hash> <counter> -- 旧值

HSET <hash> <counter> <n>

EXEC

唯一计数器 (基数计数器)



构建唯一计数器



构建唯一计数器

使用集合键实现

增加计数:

`SADD <set> <item>`

减少计数:

`SREM <set> <item>`

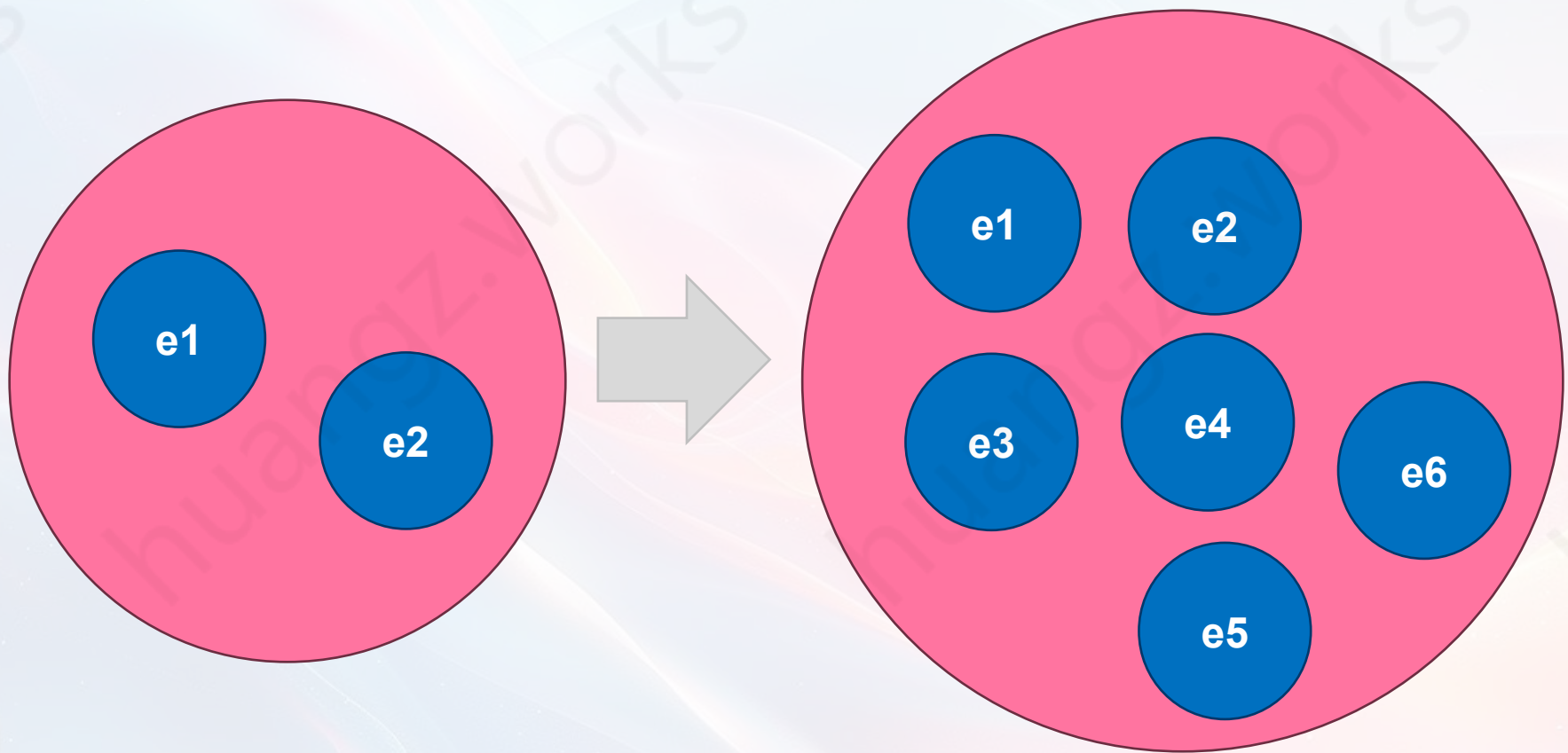
获取当前计数值:

`SCARD <set>`

重置计数:

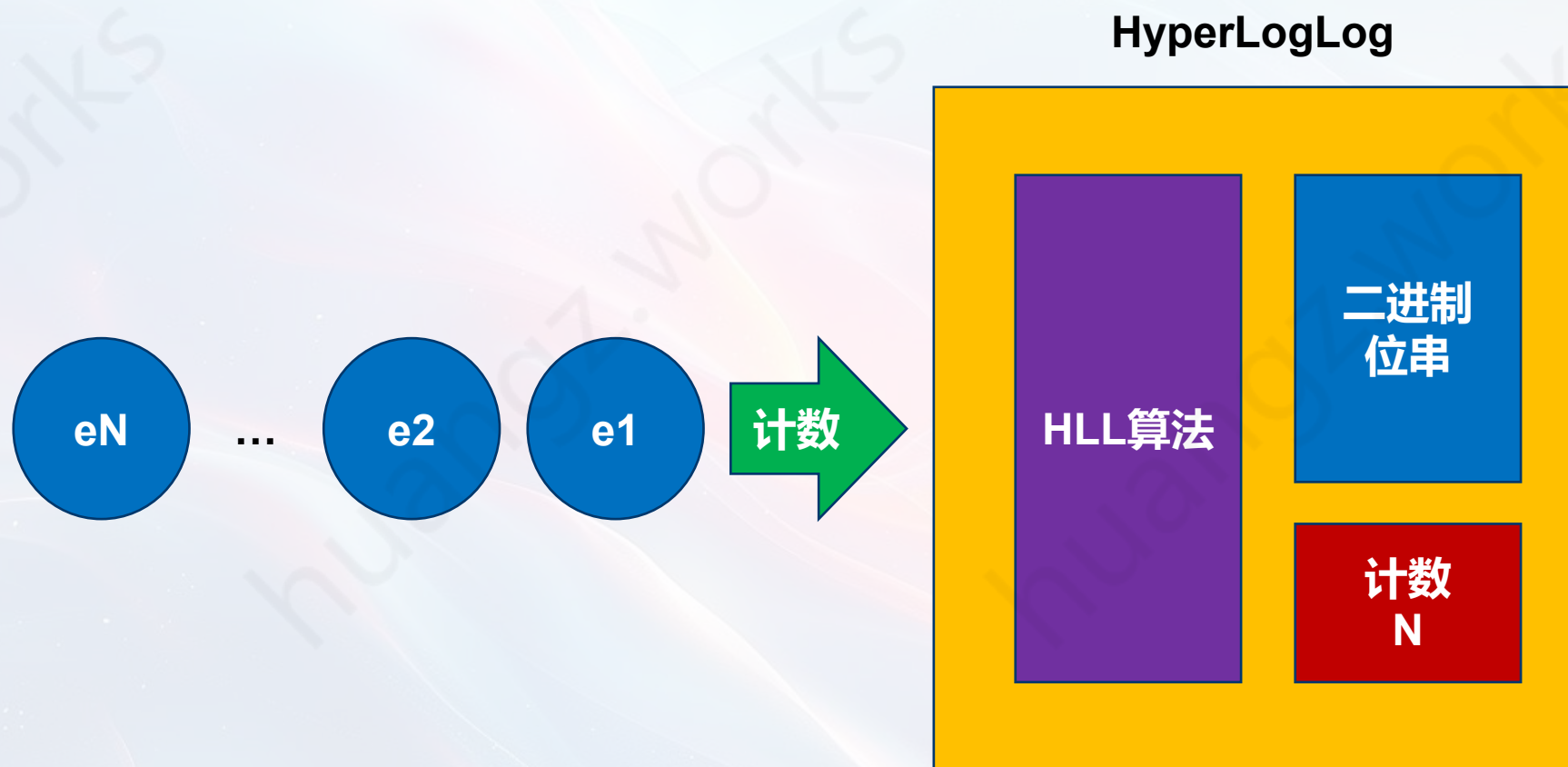
`DEL <set>`

使用集合键实现唯一计数器的问题



集合计数的元素越多，它的体积就越大

使用HyperLogLog键改善计数所需的内存



无论计数多少元素，HLL的体积都是固定的，且是微小的（几十KB）

构建唯一计数器

使用集合键实现

增加计数:

SADD <set> <item>

减少计数:

SREM <set> <item>

获取当前计数值:

SCARD <set>

重置计数:

DEL <set>

使用HyperLogLog键实现

增加计数:

PFADD <hll> <item>

减少计数:

N/A

获取当前计数值:

PFCOUNT <hll>

重置计数:

DEL <hll>

计数器程序的Python实现

关联章：

- 第6章 “计数器”
- 第7章 “唯一计数器”

代码实现：

- 第6章展示了使用Redis字符串实现的计数器Counter类和使用Redis哈希实现的计数器HashCounter类，它们具有：
 - 增加计数的increase()方法和减少计数的decrease()方法；
 - 获取现有计数值的get()方法；
 - 重置计数并返回旧计数值的reset()方法。
- 第7章展示了使用Redis集合实现的唯一计数器UniqueCounter类，它具有：
 - 对元素进行计数的include()方法和取消对元素进行计数的exclude()方法；
 - 获取当前计数值的count()方法；
- 此外，第7章还展示了使用HyperLogLog实现的唯一计数器HllUniqueCounter类，它具有除exclude()方法之外UniqueCounter类拥有的所有方法。

获取代码：

```
git clone git@github.com:huangzworks/rediscookbook.git
```



第5讲 使用排行榜 对元素进行有序排列

排行榜示例

←

🔗

⋮

巅峰榜 · 每日更新

热歌榜

最近更新 8月26日

🔴

快来巅峰潮流榜支持你喜爱的歌曲登...

➤

🎵

全部播放(100)

📁

☰

1

🎵

青花

VIP

🔒

👤

🔗

⋮

2

🎵

感谢你曾来过

VIP

🔒

👤

🔗

⋮

3

🎵

爱你

VIP

🔒

👤

🔗

⋮

4

🎵

若月亮没来 (若是月亮...

VIP

🔒

👤

🔗

⋮

5

🎵

偏爱

VIP

🔒

👤

🔗

⋮

6

🎵

如果可以

VIP

🔒

👤

🔗

⋮

←

🔍

全区排行榜

全站

番剧

国创

国创相关

纪录片

电影

1

🎬

"独属于中国人浪漫的彩蛋"

👤

👁 651.9万

📅 4173

⋮

显示此UP主全部上榜视频

2

🎬

相片漂流瓶之冰棍老头

👤

👁 265.5万

📅 2092

⋮

3

🎬

纯黑《黑神话：悟空》全程无伤攻略解说 第二期

👤

👁 277.5万

📅 2.7万

⋮

4

🎬

黑神话凭什么这么火

👤

👁 149万

📅 2979

⋮

推荐

应用榜

游戏榜

鸿蒙先锋

🔍

浏览器

热门榜

综合榜

飙升榜

匠心榜

1

📱

工具

Kimi 是一个脑容量...

安装

2

📱

旅游

随时随地预订酒店、...

安装

3

📱

推荐 视频

记录美好生活

安装

4

📱

聊天

同城单身玩法多。距...

安装

5

📱

租房买房

找二手房新房租房, ...

安装

6

📱

安装

为排行榜构建对应的有序集合



与歌曲排行榜对应的有序集合（逆序视角）

索引	分值（热度）	成员（歌曲ID）	歌名
0	5500	123456	青花
1	4800	100860	感谢你曾来过
2	4200	666777	爱你
3	4100	255255	若月亮没来
4	3500	333000	偏爱
5	3300	512512	如果可以
.....	

使用有序集合实现排行榜

添加新元素（如元素已存在则更新它的权重）：
ZADD <ranking> <weight> <item>

获取排名前N的元素：
ZRANGE <ranking> 0 <N-1> REV WITHSCORES

修改已有元素的权重（如元素不存在则创建）：
ZINCRBY <ranking> <change> <item>

获取单个元素的权重：
ZSCORE <ranking> <item>

移除指定元素：
ZREM <ranking> <item>

与歌曲排行榜对应的有序集合（逆序视角）

索引	分值（热度）	成员（歌曲ID）	歌名
0	5500	123456	青花
1	4800	100860	感谢你曾来过
2	4200	666777	爱你
3	4100	255255	若月亮没来
4	3500	333000	偏爱
5	3300	512512	如果可以
.....	

排行榜程序的Python实现

关联章:

- 第22章 “排行榜”

代码实现:

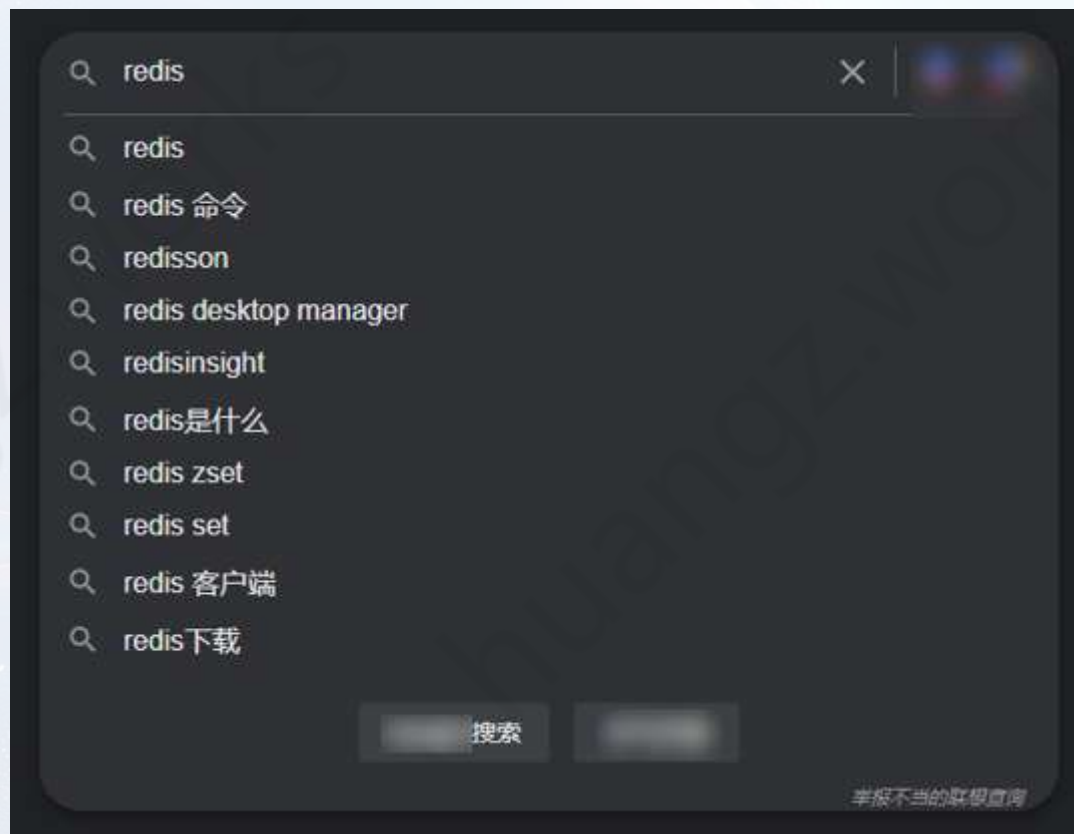
- 第22章展示了使用Redis有序集合实现的排行榜程序Ranking类, 它具有:
 - 为排行榜中指定成员设置权重的set_weight()方法;
 - 获取排行榜中指定成员权重的get_weight()方法;
 - 为排行榜中指定成员更新权重的update_weight()方法;
 - 获取排行榜前N个成员的top()方法和获取排行榜后N个成员的bottom()方法等。

获取代码:

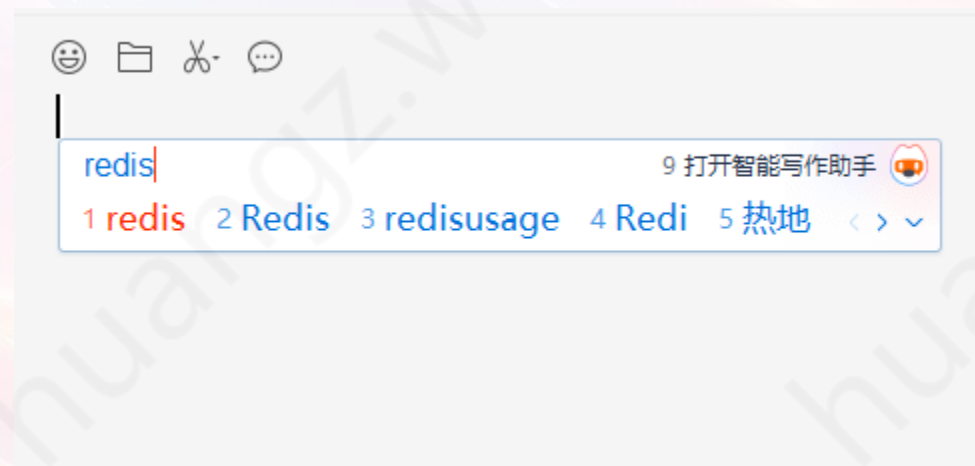
```
git clone git@github.com:huangzworks/rediscookbook.git
```


第6讲 使用自动补全 为用户提供输入建议

自动补全示例 (1/2)



自动补全示例 (2/2)



使用有序集合构建自动补全建议表



使用有序集合实现的建议表（逆序视角）

索引	分值（权重）	成员（建议项）
0	2000	“redis”
1	1500	“redis是什么”
2	1350	“redis教程”
3	1200	“redis分布式锁”
4	1100	“redis下载安装教程”
5	800	“rediscover”
.....

根据用户输入动态生成自动补全建议表的方法：以“Redis”为例

根据用户输入动态生成自动补全建议表的方法：以“Redis”为例

1. 枚举构成输入“redis”的每个组成片段
[“r”, “re”, “red”, “redi”, “redis”]

根据用户输入动态生成自动补全建议表的方法：以“Redis”为例

1. 枚举构成输入“redis”的每个组成片段
[“r”, “re”, “red”, “redi”, “redis”]
2. 根据每个片段构建自动补全建议表的键名
[“AutoComplete:r”,
“AutoComplete:re”,
“AutoComplete:red”,
“AutoComplete:redi”,
“AutoComplete:redis”]

根据用户输入动态生成自动补全建议表的方法：以“Redis”为例

1. 枚举构成输入“redis”的每个组成片段
[“r”, “re”, “red”, “redi”, “redis”]
2. 根据每个片段构建自动补全建议表的键名
[“AutoComplete:r”,
“AutoComplete:re”,
“AutoComplete:red”,
“AutoComplete:redi”,
“AutoComplete:redis”]
3. 在每个建议表对应的有序集合中，对输入“redis”成员的分值加上权重增量（如1.0）
ZINCRBY AutoComplete:r 1.0 “redis”
ZINCRBY AutoComplete:re 1.0 “redis”
ZINCRBY AutoComplete:red 1.0 “redis”
ZINCRBY AutoComplete:redi 1.0 “redis”
ZINCRBY AutoComplete:redis 1.0 “redis”

根据输入动态构建建议表示例



根据输入动态构建建议表示例

用户输入:

- “redis”
- “redis”
- “reveal”
- “react”
- “respect”
- “redis”
- “react”
- “redis”
- “react”
- “reveal”

根据输入动态构建建议表示例

用户输入:

- “redis”
- “redis”
- “reveal”
- “react”
- “respect”
- “redis”
- “react”
- “redis”
- “react”
- “reveal”



AutoComplete:r有序集合 (逆序视角)

索引	分值 (权重)	成员 (建议项)
0	4	“redis”
1	3	“react”
2	2	“reveal”
3	1	“respect”

根据输入动态构建建议表示例

用户输入:

- “redis”
- “redis”
- “reveal”
- “react”
- “respect”
- “redis”
- “react”
- “redis”
- “react”
- “reveal”



AutoComplete:r有序集合 (逆序视角)

索引	分值 (权重)	成员 (建议项)
0	4	“redis”
1	3	“react”
2	2	“reveal”
3	1	“respect”

执行命令 ZRANGE AutoComplete:r 0 N REV即可查找输入“r”的建议项

通过过期时间特性自动删除冷门建议表

通过过期时间特性自动删除冷门建议表

随着程序运行时间不断增长，上述方法将创建大量建议表。

比如说，光为了补全“redis”一个单词，就需要创建至少5个表，而每个表又会包含多个成员。

通过过期时间特性自动删除冷门建议表

随着程序运行时间不断增长，上述方法将创建大量建议表。

比如说，光为了补全“redis”一个单词，就需要创建至少5个表，而每个表又会包含多个成员。

为此，可以在对建议项更新权重的同时，给建议项所在的表设置过期时间：

```
ZINCRBY AutoComplete:r 1.0 "redis"
```

```
ZINCRBY AutoComplete:re 1.0 "redis"
```

```
ZINCRBY AutoComplete:red 1.0 "redis"
```

```
ZINCRBY AutoComplete:redi 1.0 "redis"
```

```
ZINCRBY AutoComplete:redis 1.0 "redis"
```

```
EXPIRE AutoComplete:r 300
```

```
EXPIRE AutoComplete:re 300
```

```
EXPIRE AutoComplete:red 300
```

```
EXPIRE AutoComplete:redi 300
```

```
EXPIRE AutoComplete:redis 300
```


通过过期时间特性自动删除冷门建议表

随着程序运行时间不断增长，上述方法将创建大量建议表。

比如说，光为了补全“redis”一个单词，就需要创建至少5个表，而每个表又会包含多个成员。

为此，可以在对建议项更新权重的同时，给建议项所在的表设置过期时间：

```
ZINCRBY AutoComplete:r 1.0 "redis"
ZINCRBY AutoComplete:re 1.0 "redis"
ZINCRBY AutoComplete:red 1.0 "redis"
ZINCRBY AutoComplete:redi 1.0 "redis"
ZINCRBY AutoComplete:redis 1.0 "redis"
EXPIRE AutoComplete:r 300
EXPIRE AutoComplete:re 300
EXPIRE AutoComplete:red 300
EXPIRE AutoComplete:redi 300
EXPIRE AutoComplete:redis 300
```

在Redis过期时间机制的支持下，热门输入所创建的建议表的过期时间将不断地更新续期，而冷门输入对应的建议表则会随着过期时间到达而自动被删除。

自动补全程序的Python实现

关联章:

- 第16章 “自动补全”

代码实现:

- 第16章展示了使用Redis有序集合实现的自动补全程序AutoComplete类, 它带有:
 - 接受用户输入并自动创建相应建议表的feed()方法, 还有可以自动对冷门建议表进行自动过期的feedex()方法;
 - 根据给定前缀获取相应输入建议的hint()方法;
 - 根据已有权重直接为指定内容设置权重的set()方法。

获取代码:

```
git clone git@github.com:huangzworks/rediscookbook.git
```


第7讲

构建类似Stack Overflow等 网站的投票系统

投票示例：Stack Overflow网站

How to make a great R reproducible example

Asked 13 years, 4 months ago Modified 4 months ago Viewed 472k times ★ Part of R Language Collective



2464



This question's answers are a [community effort](#). Edit existing answers to improve this post. It is not currently accepting new answers or interactions.



When discussing performance with colleagues, teaching, sending a bug report or searching for guidance on mailing lists and here on Stack Overflow, a [reproducible example](#) is often asked and always helpful.

What are your tips for creating an excellent example? How do you paste data structures from `r` in a text format? What other information should you include?

Are there other tricks in addition to using `dput()`, `dump()` or `structure()`? When should you include `library()` or `require()` statements? Which reserved words should one avoid, in addition to `c`, `df`, `data`, etc.?

How does one make a great `r` reproducible example?

R

r

r-faq

投票示例：知识问答网站



MySQL+Java+Redis+算法+网络+Linux等一个都讲不清



面试大概九十分钟，问的东西很全面，需要做充足准备，就是除了概念以外问的有点  了（呜呜呜~）。回来之后把这些题目做了一个分类并整理出答案（强迫症的我~狂补知识~）分为 MySQL+Java+Redis+... [阅读全文](#) 

 赞同 2399

 139 条评论

 分享

 收藏



 已认证账号



今天，我面试了某大厂的java开发岗位，迎面走来一位风尘仆仆的中年男子，手里拿着屏幕还亮着的mac，他冲着我礼貌的笑了笑，然后说了句“不好意思，让你久等了”，然后示意我坐下，说：“我们开始吧。看了你的... [阅读全文](#) 

 赞同 2357

 125 条评论


 分享

 收藏

投票示例：书评网站

★★★★★ 2017-04-03 21:58:53

| 这篇书评可能有关键情节透露

本来是为应付面试读的(几次面试当中问到了好多Redis实现方面的问题:Redis如何实现expire、如何实现集群、sentinel作用、事务实现、哪类持久化更安全、各种数据结构的底层实现...),结果在看了几章之后完全被吸引住了,深入浅出,读来流畅痛快,有种“’的感觉。一方面,... (展开)

△ 18 ▽ 1 1回应

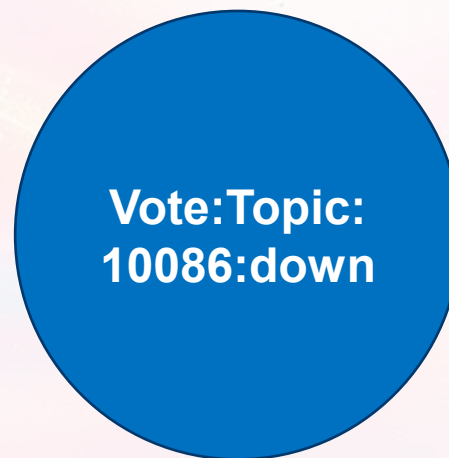
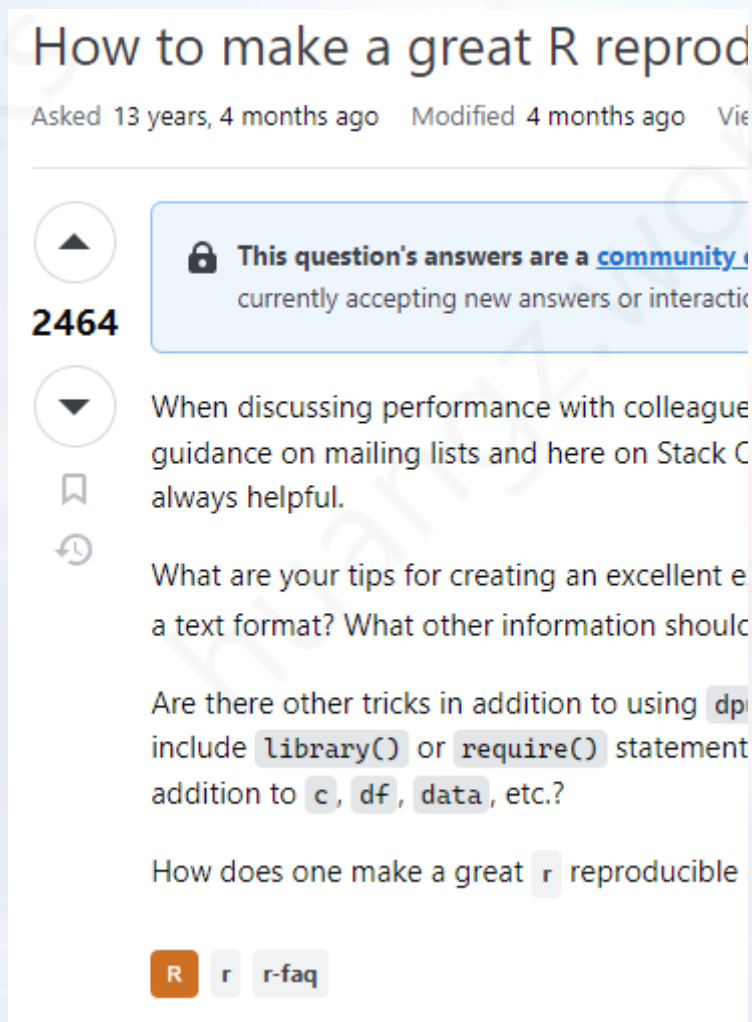
★★★★★ 2014-06-29 16:28:09

很好的设计原理剖析

第一时间入手,花了一个周末读完。总体来说很棒,比网站上的内容丰富了很多。亮点: 1. 在源码层面,对Redis几乎所有特性都做了深入分析。2. 每个章节都有很多生动的配图,便于理解。3. 附带了注释版的Redis3.0源码,结合着看很爽。挑刺: 1. 基本是讲原理,涉及实战经验较... (展开)

△ 10 ▽ 3回应

使用两个集合分别记录主题的支持票和反对票



投票操作的实现 (1/2)

投支持票:

MULTI

SADD Vote:<topic>:up <user>

SREM Vote:<topic>:down <user>

EXEC

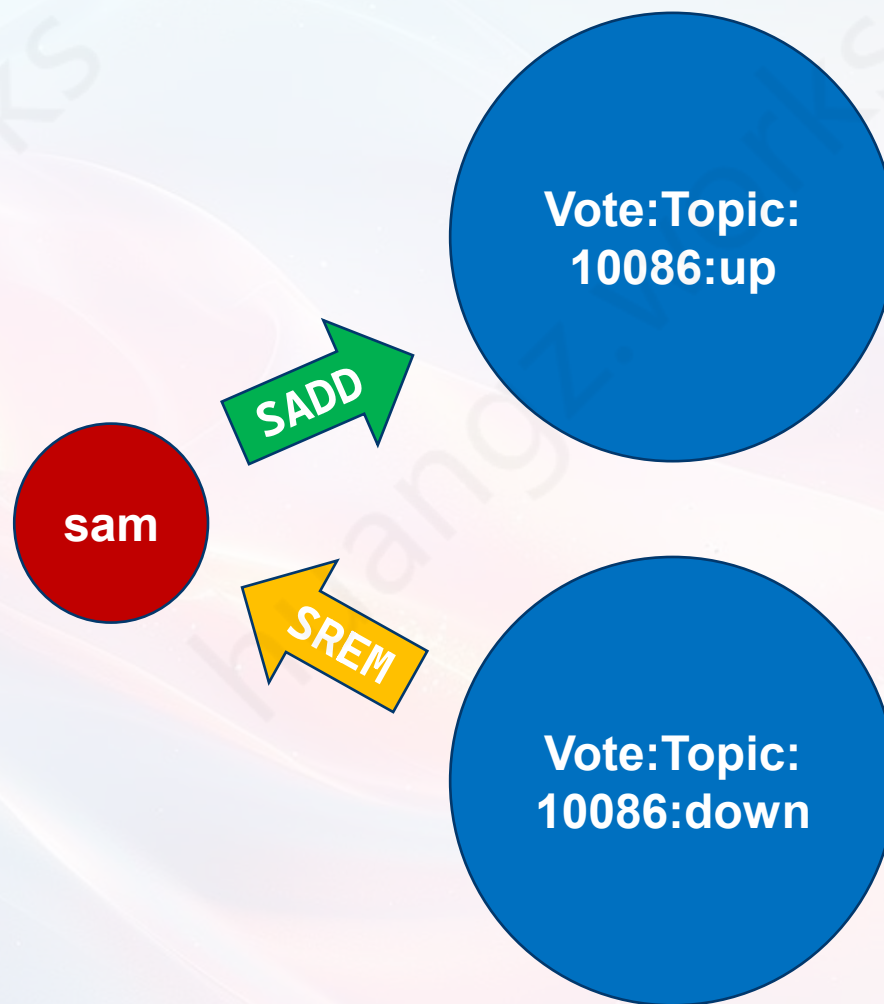
投反对票:

MULTI

SADD Vote:<topic>:down <user>

SREM Vote:<topic>:up <user>

EXEC



投票操作的实现 (2/2)

查看指定用户是否已投票:

```
SISMEMBER Vote:<topic>:up <user>
```

```
SISMEMBER Vote:<topic>:down <user>
```

查看已投支持票或反对票的用户数量:

```
SCARD Vote:<topic>:up
```

```
SCARD Vote:<topic>:down
```

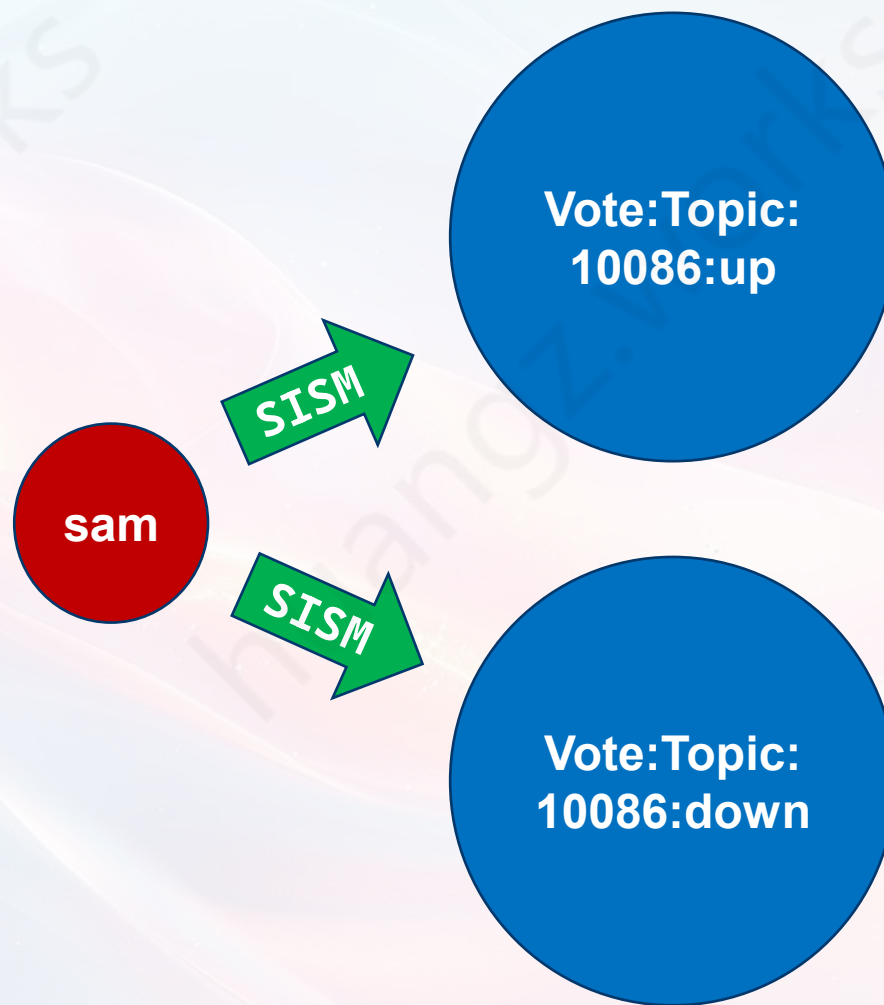
取消投票:

```
MULTI
```

```
SREM Vote:<topic>:up <user>
```

```
SREM Vote:<topic>:down <user>
```

```
EXEC
```



投票程序的Python实现

关联章:

- 第21章 “投票” , 第7章 “唯一计数器” , 第9章 “二元操作记录器”

代码实现:

- 第21章展示了使用Redis集合实现的投票类Vote, 它提供了:
 - 让用户投支持票的up()方法和投反对票的down()方法;
 - 检查用户是否已投票的is_voted()方法;
 - 取消投票的unvote()方法等。
- 第7章的HllUniqueCounter或第9章的BinaryRecorder方法可以用于优化投票程序的内存占用。

获取代码:

```
git clone git@github.com:huangzworks/rediscookbook.git
```



第8讲

使用分页和时间线 排列并管理大量元素

分页示例（移动应用）



分页示例（传统网页）



使用Redis列表构建分页列表

使用Redis列表构建分页列表



使用Redis列表构建分页列表



索引	项（视频ID）
0	10086
1	12345
2	25525
3	51251
4	78978
5	66666

第一页

使用Redis列表构建分页列表



索引	项（视频ID）
0	10086
1	12345
2	25525
3	51251
4	78978
5	66666

第一页



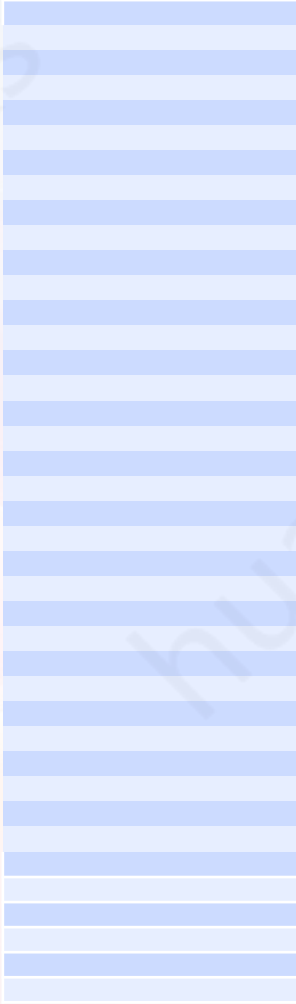
视频ID列表

第一页

第二页

第三页

最后页



使用Redis列表构建分页列表

使用Redis列表构建分页列表

将元素放入分页列表:

```
LPUSH <list> <item>
```

使用Redis列表构建分页列表

将元素放入分页列表:

```
LPUSH <list> <item>
```

从分页列表中获取被分页的项:

```
LRANGE <list> <start> <end>
```

计算分页项索引的公式:

```
start = (page-1)*size
```

```
end = page*size-1
```

其中page为想要获取的页数, size为想要获取的项数量。

使用Redis列表构建分页列表

将元素放入分页列表:

`LPUSH <list> <item>`

从分页列表中获取被分页的项:

`LRANGE <list> <start> <end>`

计算分页项索引的公式:

`start = (page-1)*size`

`end = page*size-1`

其中page为想要获取的页数, size为想要获取的项数量。

获取列表总长度:

`LLEN <list>`

按时间而不仅仅是索引来获取分页列表中的元素



The image shows a screenshot of the Weibo (微博) Advanced Search (高级搜索) interface. At the top, there are four tabs: 精选 (Selected), 微博 (Weibo), 视频 (Video), and 相册 (Albums). The 微博 tab is currently selected and highlighted with an orange underline. Below the tabs, the title '高级搜索' (Advanced Search) is displayed on the left, and a small upward-pointing triangle icon is on the right. Under the title, there is a section for '类型' (Type) with six checkboxes, all of which are checked: 原创 (Original), 转发 (Retweet), 纯文字 (Pure Text), 含图片 (Contains Image), 含视频 (Contains Video), and 含音乐 (Contains Music). Below this is a '关键字' (Keywords) section with a search input field containing the placeholder text '搜索我的微博' (Search my Weibo). At the bottom, there is a '时间' (Time) section with a date range filter. This section is highlighted with a red rectangular box. It consists of the label '时间', followed by a date input field showing '2024/01/01', the word '到' (to), another date input field showing '2024/09/09', and an orange '搜索' (Search) button to the right.

精选 微博 视频 相册

高级搜索 ▲

类型 ☒ 原创 ☒ 转发 ☒ 纯文字 ☒ 含图片 ☒ 含视频 ☒ 含音乐

关键字

时间 2024/01/01 到 2024/09/09 搜索

使用Redis有序集合构建时间线



索引	分值 (时间戳)	成员 (微博ID)
0	1725840554	wb528129312
1	1725794028	wb508231234
2	1725685375	wb490920302
3	1725421036	wb450292302
4	1725300301	wb442912312
.....

使用Redis有序集合构建时间线



使用Redis有序集合构建时间线

将元素放入时间线:

```
ZADD <timeline> <time> <item>
```

使用Redis有序集合构建时间线

将元素放入时间线:

```
ZADD <timeline> <time> <item>
```

按索引区间从时间线中获取被分页的成员:

```
ZRANGE <timeline> <start> <end> REV
```

使用Redis有序集合构建时间线

将元素放入时间线:

```
ZADD <timeline> <time> <item>
```

按索引区间从时间线中获取被分页的成员:

```
ZRANGE <timeline> <start> <end> REV
```

按时间戳区间从时间线中获取成员

```
ZRANGE <timeline> <start-time> <end-time> BYSCORE REV LIMIT offset count
```


使用Redis有序集合构建时间线

将元素放入时间线:

```
ZADD <timeline> <time> <item>
```

按索引区间从时间线中获取被分页的成员:

```
ZRANGE <timeline> <start> <end> REV
```

按时间戳区间从时间线中获取成员

```
ZRANGE <timeline> <start-time> <end-time> BYSCORE REV LIMIT offset count
```

获取时间线长度

```
ZCARD <timeline>
```

分页程序和时间线程序的Python实现

关联章:

- 第23章 “分页” , 第24章 “时间线”

代码实现:

- 第23章展示了使用Redis列表实现的分页程序Paging类, 它具有:
 - 将项添加至分页列表add()方法;
 - 从分页列表中获取被分页项的get()方法;
 - 获取分页列表长度的length()方法和获取分页数量的number()方法等。
- 第24章展示了使用Redis有序集合实现的时间线程序Timeline类, 它具有:
 - 将成员添加至时间线的add()方法;
 - 按索引区间获取时间线成员的get()方法;
 - 按时间戳区间获取时间线成员的get_with_time()方法等。

获取代码:

```
git clone git@github.com:huangzworks/rediscookbook.git
```



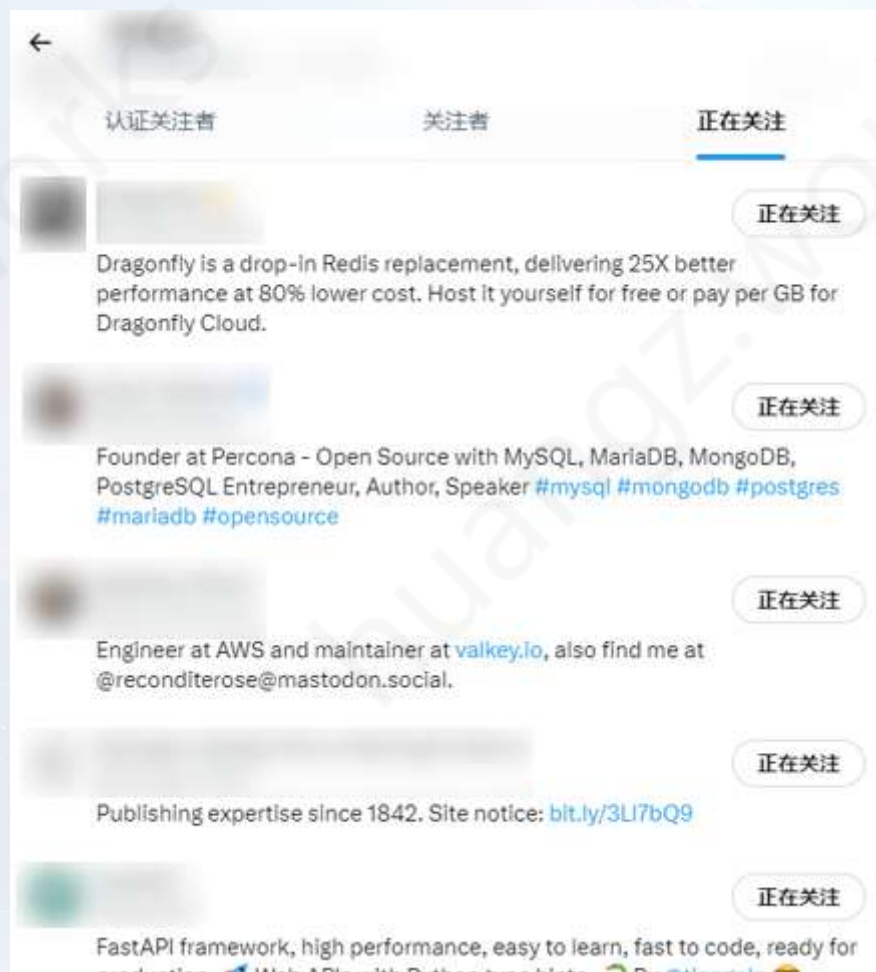

第9讲

使用社交关系程序 存储用户间的社交关系

社交关系示例



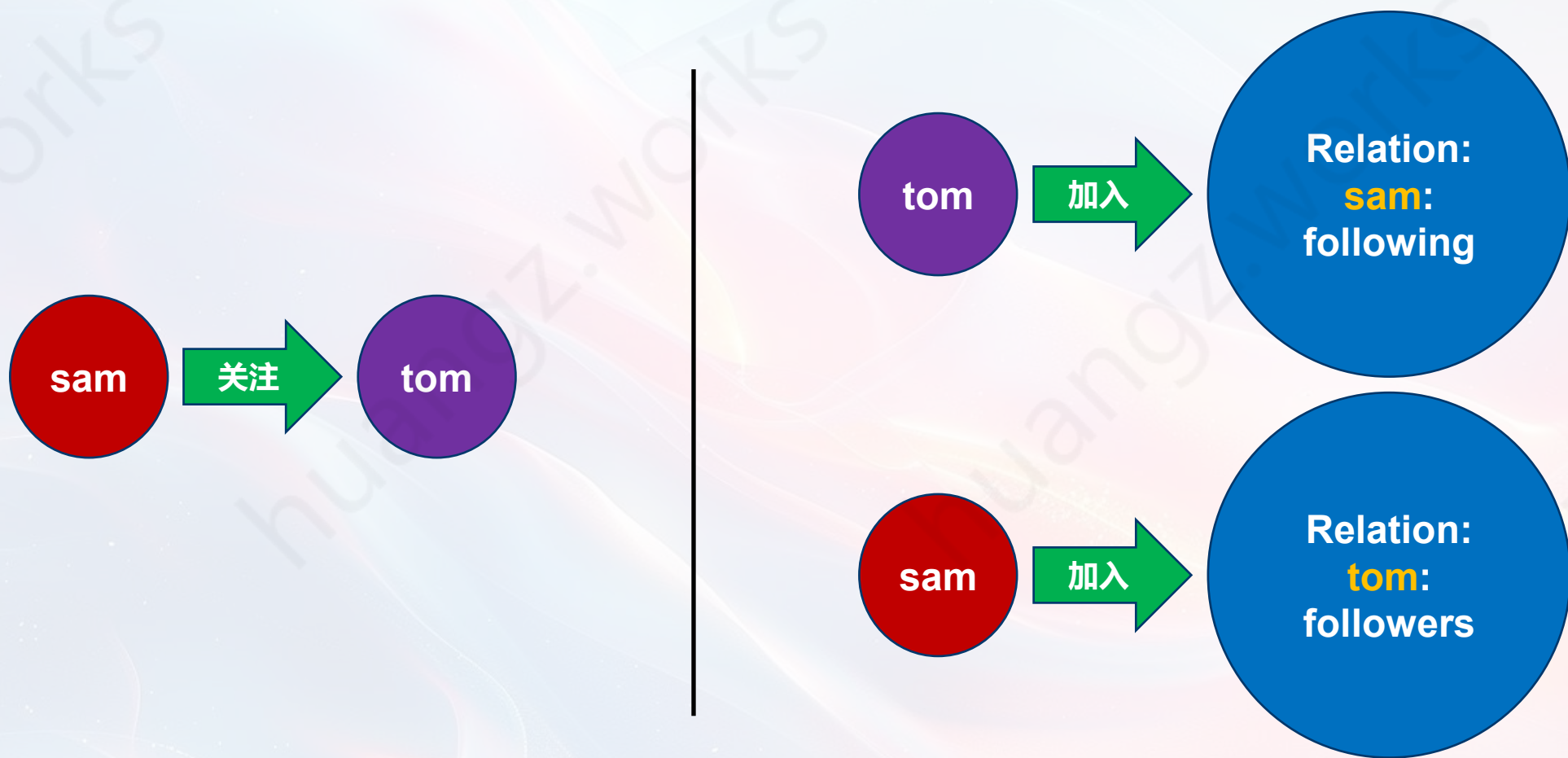
正在关注和关注者



正在关注和关注者



使用Redis集合存储用户的“正在关注”和“关注者”信息



使用Redis有序集合存储用户的“正在关注”和“关注者”信息



与“正在关注”名单对应的有序集合

索引	分值 (时间戳)	成员 (用户ID)
0	1725840554	uid123456
1	1725794028	uid255255
2	1725685375	uid345678
3	1725421036	uid666777
4	1725300301	uid512512
.....

使用Redis有序集合实现社交关系

huangz.works

huangz.works

huangz.works

huano

使用Redis有序集合实现社交关系

关注指定用户:

```
MULTI
```

```
ZADD <user_following> <time> <target>
```

```
ZADD <target_followers> <time> <user>
```

```
EXEC
```

使用Redis有序集合实现社交关系

关注指定用户:

```
MULTI
```

```
ZADD <user_following> <time> <target>
```

```
ZADD <target_followers> <time> <user>
```

```
EXEC
```

取消关注:

```
MULTI
```

```
ZREM <user_following> <target>
```

```
ZREM <target_followers> <user>
```

```
EXEC
```

使用Redis有序集合实现社交关系

关注指定用户:

```
MULTI
```

```
ZADD <user_following> <time> <target>
```

```
ZADD <target_followers> <time> <user>
```

```
EXEC
```

取消关注:

```
MULTI
```

```
ZREM <user_following> <target>
```

```
ZREM <target_followers> <user>
```

```
EXEC
```

是否关注指定用户:

```
ZRANK <user_following> <target> != nil
```


使用Redis有序集合实现社交关系

关注指定用户:

```
MULTI
```

```
ZADD <user_following> <time> <target>
```

```
ZADD <target_followers> <time> <user>
```

```
EXEC
```

取消关注:

```
MULTI
```

```
ZREM <user_following> <target>
```

```
ZREM <target_followers> <user>
```

```
EXEC
```

是否关注指定用户:

```
ZRANK <user_following> <target> != nil
```

正在关注数量:

```
ZCARD <user_following>
```

社交关系程序的Python实现

关联章:

- 第18章 “社交关系”

代码实现:

- 第18章展示了使用Redis有序集合实现的社交关系程序Relation类, 它具有:
 - 关注指定用户的follow()方法以及取关指定用户的unfollow()方法;
 - 检查是否关注了指定用户的is_following()方法以及是否被指定用户关注的is_following_by()方法;
 - 检查是否与指定用户形成了互相关注关系的following_each_other()方法;
 - 获取正在关注数量的following_count()方法和关注者数量的followers_count()方法。

获取代码:

```
git clone git@github.com:huangzworks/rediscookbook.git
```




第10讲 使用地理位置程序 记录用户的位置信息

地理位置示例



使用Redis地理空间索引（GEO）记录用户地理位置

使用Redis地理空间索引（GEO）记录用户地理位置

记录用户坐标：

```
GEOADD <location> <longitude> <latitude> <member>
```


使用Redis地理空间索引（GEO）记录用户地理位置

记录用户坐标：

```
GEOADD <location> <longitude> <latitude> <member>
```

获取用户坐标：

```
GEOPOS <location> <member>
```

使用Redis地理空间索引（GEO）记录用户地理位置

记录用户坐标:

```
GEOADD <location> <longitude> <latitude> <member>
```

获取用户坐标:

```
GEOPOS <location> <member>
```

计算两个用户之间的直线距离:

```
GEODIST <location> <member_a> <member_b> <unit>
```

使用Redis地理空间索引（GEO）记录用户地理位置

记录用户坐标：

```
GEOADD <location> <longitude> <latitude> <member>
```

获取用户坐标：

```
GEOPOS <location> <member>
```

计算两个用户之间的直线距离：

```
GEODIST <location> <member_a> <member_b> <unit>
```

搜索位于指定用户特定范围内的其他用户：

```
GEOSEARCH <location> FROMMEMBER <member> BYRADIUS <radius> <unit>
```


实现随机获取附近用户功能



实现随机获取附近用户功能

无缓存版本

直接执行GEOSEARCH命令并从中随机挑选结果

实现随机获取附近用户功能

无缓存版本

直接执行GEOSEARCH命令并从中随机挑选结果

缓存版本

使用**GEOSEARCHSTORE**命令，搜索指定用户的附近用户名单，并将其保存至指定键中：

MULTI

GEOSEARCHSTORE **<dest>** <location> FROMMEMBER <member> BYRADIUS <radius> <unit>

EXPIRE <dest> <ttd>

EXEC

实现随机获取附近用户功能

无缓存版本

直接执行GEOSEARCH命令并从中随机挑选结果

缓存版本

使用**GEOSEARCHSTORE**命令，搜索指定用户的附近用户名单，并将其保存至指定键中：

MULTI

GEOSEARCHSTORE **<dest>** <location> FROMMEMBER <member> BYRADIUS <radius> <unit>

EXPIRE <dest> <ttd>

EXEC

从附近用户名单中随机返回指定数量的用户：

ZRANDMEMBER <dest> <N>

(因为Redis GEO数据是以有序集合方式存储的，所以可以直接使用有序集合命令来处理它们)

地理位置程序的Python实现

关联章:

- 第25章 “地理位置”

代码实现:

- 第25章展示了地理位置程序Location类的具体定义, 它具有:
 - 记录用户位置的pin()方法;
 - 获取用户坐标的locate()方法;
 - 计算两个用户之间直线距离的distance()方法;
 - 以给定用户为圆心, 搜索指定公里数范围内其他用户的search()方法;
 - 随机返回给定用户1公里范围内其他用户的random_neighbour()方法及其缓存版本cached_random_neighbour()方法等。

获取代码:

```
git clone git@github.com:huangzworks/rediscookbook.git
```

32个实例
50+程序

异步图书
www.epubit.com

Redis 应用与实战丛书

Redis 应用实例

黄健宏 著

案例丰富，注重实战
简明易懂，代码详尽

中国工信出版集团 人民邮电出版社
POSTS & TELECOM PRESS

全面兼容
Redis 8



感谢观看！