redis_day01回顾

Redis的特点

```
1、基于key-value的非关系型数据库
2、基于内存存储，速度很快
3、基于内存存储，经常当作缓存型数据库使用，常用信息缓存在redis数据库中
```

五大数据类型

```
1、字符串类型（string）
2、列表类型（list）
3、哈希类型（hash）
4、集合类型（set）
5、有序集合类型（sorted set）
```

字符串类型

```
# 设置key相关操作
1、set key value
2、setnx key value
3、mset k1 v1 k2 v2 k3 v3
4、set key value ex seconds
5、set key value
5、expire key 5
5、pexpire key 5
5、ttl key
5、persist key
# 获取key相关操作
6、get key
7、mget k1 k2 k3
8、strlen key
# 数字相关操作
7、incrby key 步长
8、decrby key 步长
9、incr key
10、decr key
11、incrbyfloat key number
```

列表类型

```
# 插入元素相关操作
1、LPUSH key value1 value2
2、RPUSH key value1 value2
3、RPOPLPUSH source destination
4、LINSERT key after|before value newvalue
# 查询相关操作
5、LRANGE key start stop
6、LLEN key
# 删除相关操作
7、LPOP key
8、RPOP key
9、BLPOP key timeout
10、BRPOP key timeout
11、LREM key count value
12、LTRIM key start stop
# 修改指定元素相关操作
13、LSET key index newvalue
```

思考:

Redis列表如何当做共享队列来使用？？？

```
# 同学你好，你还记得小米应用商店爬取URL地址的案例吗?
1、生产者消费者模型
2、生产者进程在列表中 LPUSH | RPUSH 数据，消费者进程在列表中 RPOP | LPOP 数据
```

Python与redis交互注意

```
1、r.set('name','Tom',ex=5,nx=True)
2、r.mset({'user1:name':'Tom','user1:age':'25'})
# 有元素时返回弹出元素,否则返回None
3、r.brpop('mylist')
```

redis_day02笔记

位图操作bitmap

定义

```
1、位图不是真正的数据类型，它是定义在字符串类型中
2、一个字符串类型的值最多能存储512M字节的内容，位上限：2^32
# 1MB = 1024KB
# 1KB = 1024Byte(字节)
# 1Byte = 8bit(位)
```

强势点

```
可以实时的进行统计，极其节省空间。官方在模拟1亿2千8百万用户的模拟环境下，在一台MacBookPro上，典型的统计如“日用户数”的时间消耗小于50ms，占用16MB内存
```

设置某一位上的值（setbit）

```
# 设置某一位上的值（offset是偏移量，从0开始）
setbit key offset value
# 获取某一位上的值
GETBIT key offset
# 统计键所对应的值中有多少个 1
BITCOUNT key
```

示例

```
# 默认扩展位以0填充
127.0.0.1:6379> set mykey ab
OK
127.0.0.1:6379> get mykey
"ab"
127.0.0.1:6379> SETBIT mykey 0 1
(integer) 0
127.0.0.1:6379> get mykey
"\xe1b"
127.0.0.1:6379>
```

获取某一位上的值

GETBIT key offset

```
127.0.0.1:6379> GETBIT mykey 3
(integer) 0
127.0.0.1:6379> GETBIT mykey 0
(integer) 1
127.0.0.1:6379>
```

bitcount

统计键所对应的值中有多少个 1

```
127.0.0.1:6379> SETBIT user001 1 1
(integer) 0
127.0.0.1:6379> SETBIT user001 30 1
(integer) 0
127.0.0.1:6379> bitcount user001
(integer) 2
127.0.0.1:6379>
```

应用场景案例

网站用户的上线次数统计（寻找活跃用户）

用户名为key，上线的天作为offset，上线设置为1

示例: 用户名为 user001 的用户，今年第1天上线，第30天上线

SETBIT user1:login 1 1

SETBIT user1:login 30 1

BITCOUNT user1:login

代码实现

```python
import redis

r = redis.Redis(host='192.168.43.49',port=6379,db=0)

# user1, 一年之中第1天和第5天登录
r.setbit('user1:login',1,1)
r.setbit('user1:login',5,1)
# user2, 一年之中第100天和第200天登录
r.setbit('user2:login',100,1)
r.setbit('user2:login',200,1)
# user3, 一年之中好多天登录
for i in range(0,365,2):
    r.setbit('user3:login',i,1)
# user4, 一年之中好多天登录
for i in range(0,365,3):
    r.setbit('user4:login',i,1)

user_list = r.keys('user*')
print(user_list)

# 活跃用户
active_users = []
# 不活跃用户
noactive_user = []

for user in user_list:
    # 统计位图中有多少个 1
    login_count = r.bitcount(user)
    if login_count >= 100:
       active_users.append((user,login_count))
    else:
      noactive_user.append((user,login_count))
```

```python

# 打印活跃用户
for active in active_users:
    print('活跃用户:',active)
```

Hash散列数据类型

- **定义**

```
1、由field和关联的value组成的键值对
2、field和value是字符串类型
3、一个hash中最多包含2^32-1个键值对
```

- **优点**

```
1、节约内存空间
2、每创建一个键，它都会为这个键储存一些附加的管理信息（比如这个键的类型，这个键最后一次被访问的时间等）
3、键越多，redis数据库在储存附件管理信息方面耗费内存越多，花在管理数据库键上的CPU也会越多
```

- **缺点（不适合hash情况）**

```
1、使用二进制位操作命令:SETBIT、GETBIT、BITCOUNT等，如果想使用这些操作，只能用字符串键
2、使用过期键功能：键过期功能只能对键进行过期操作，而不能对散列的字段进行过期操作
```

基本命令操作

```
# 1、设置单个字段
HSET key field value
HSETNX key field value
# 2、设置多个字段
HMSET key field value field value
# 3、返回字段个数
HLEN key
# 4、判断字段是否存在（不存在返回0）
HEXISTS key field
# 5、返回字段值
HGET key field
# 6、返回多个字段值
HMGET key field filed
# 7、返回所有的键值对
HGETALL key
# 8、返回所有字段名
HKEYS key
# 9、返回所有值
HVALS key
# 10、删除指定字段
HDEL key field
# 11、在字段对应值上进行整数增量运算
HINCRBY key filed increment
# 12、在字段对应值上进行浮点数增量运算
HINCRBYFLOAT key field increment
```

python基本方法

```
# 1、更新一条数据的属性，没有则新建
hset(name, key, value)
# 2、读取这条数据的指定属性， 返回字符串类型
hget(name, key)
# 3、批量更新数据（没有则新建）属性,参数为字典
hmset(name, mapping)
# 4、批量读取数据（没有则新建）属性
hmget(name, keys)
# 5、获取这条数据的所有属性和对应的值，返回字典类型
hgetall(name)
# 6、获取这条数据的所有属性名，返回列表类型
hkeys(name)
# 7、删除这条数据的指定属性
hdel(name, *keys)
```

Python代码hash散列

```
import redis

r = redis.Redis(host="192.168.153.136", port=6379, db=0)
# 新建一条键名为"user1"的数据，包含属性name
r.hset("user1", "name", 'zhanshen001')
# 更改键名为"userinfo"的数据，更改属性username的值
r.hset("user1", "name", 'zhanshen002')

# 取出属性username的值
username = r.hget("user1", "name")

# 输出看一下
print('name',username)

# 属性集合
user_dict = {
    "password": "123456",
    "name": "Wang Success",
    "sex": "male",
    "height": '178',
    "Tel": '13838383888',
}
# 批量添加属性
r.hmset("user1", user_dict)
# 取出所有数据(返回值为字典)
all_data = r.hgetall("userinfo")
print('all_data:', all_data)
# 删除属性(可以批量删除)
r.hdel("user1", "Tel")
# 取出所有属性名 : 列表
h_keys = r.hkeys("user1")
print('all_key_name:',h_keys)
# 取出所有属性值 : 列表
h_values = r.hvals('user1')
print('all_values:',h_values)
```

应用场景：微博好友关注

```
1、用户ID为key，Field为好友ID，Value为关注时间
    user:10000 user:606 20190520
    user:10000 user:605 20190521
2、用户维度统计
   统计数包括：关注数、粉丝数、喜欢商品数、发帖数
   用户为key，不同维度为field，value为统计数
   比如关注了5人
    HSET user:10000 fans 5
    HINCRBY user:10000 fans 1
```

应用场景: redis+mysql+hash组合使用

- 原理

```
用户想要查询个人信息
1、到redis缓存中查询个人信息
2、redis中查询不到，到mysql查询，并缓存到redis
3、再次查询个人信息
```

- 代码实现

```python
import redis
import pymysql

# 1、到redis中查询个人信息
# 2、redis中查询不到，到mysql查询，并缓存到redis
# 3、再次查询个人信息


r = redis.Redis(host='192.168.153.136',port=6379,db=0)

username = input('请输入用户名:')
# 如果redis中没有缓存，则返回空字典{}
result = r.hgetall(username)
print('redis中找到:',result)

# mysql中表字段: username、password、gender、age
if not result:
    db = pymysql.connect('192.168.153.136','tiger','123456','spider',charset='utf8')
    cursor = db.cursor()
    cursor.execute('select gender,age from user where username=%s',[username])
    # (('m',30),)
    userinfo = cursor.fetchall()
    if not userinfo:
        print('MySQL中用户信息不存在')
    else:
        dict = {
            'gender':userinfo[0][0],
            'age':userinfo[0][1]
        }
        # hmset第二个参数为字典
        r.hmset(username,dict)
```

```
        # 设置过期时间为5分钟
        r.expire(username,60*5)
        print('redis缓存成功')
```

mysql数据库中数据更新信息后同步到redis缓存

用户修改个人信息时，要将数据同步到redis缓存

```
import redis
import pymysql

# 当用户修改个人信息时，要同步更新到redis缓存中

def update_mysql(username,new_age):
    #  连接MySQL
    db = pymysql.connect('192.168.153.136','tiger','123456','spider',charset='utf8')
    cursor = db.cursor()
    cursor.execute('update user set age=%s where username=%s',[new_age,username])
    db.commit()
    cursor.close()
    db.close()

def update_redis(username,new_age):
    #  连接redis
    r = redis.Redis(host='192.168.153.136', port=6379, db=0)
    # 同步更新redis缓存
    r.hset(username,'age',new_age)
    print('已同步到redis缓存')
    # 设置过期时间为5分钟
    r.expire(username,60*5)
    # 从redis中打印查看
    print(r.hget(username,'age'))

if __name__ == '__main__':
    username = input('请输入用户名:')
    new_age = input('请输入新年龄:')
    update_mysql(username,new_age):
    update_redis(username,new_age)
```

集合数据类型（set）

- 特点

```
1、无序、去重
2、元素是字符串类型
3、最多包含2^32-1个元素
```

- 基本命令

```
# 1、增加一个或者多个元素,自动去重
```

```
SADD key member1 member2
# 2、查看集合中所有元素
SMEMBERS key
# 3、删除一个或者多个元素，元素不存在自动忽略
SREM key member1 member2
# 4、元素是否存在
SISMEMBER key member
# 5、随机返回集合中指定个数的元素，默认为1个
SRANDOMMEMBER key [count]
# 6、返回集合中元素的个数，不会遍历整个集合，只是存储在键当中了
SCARD key
# 7、把元素从源集合移动到目标集合
SMOVE source destination member
# 8、差集(number1 1 2 3 number2 1 2 4)
SDIFF key1 key2
# 9、差集保存到另一个集合中
SDIFFSTORE destination key1 key2
# 10、交集
SINTER key1 key2
SINTERSTORE destination key1 key2
# 11、并集
SUNION key1 key2
SUNIONSTORE destination key1 key2
```

案例: 新浪微博的共同关注

需求: 当用户访问另一个用户的时候，会显示出两个用户共同关注过哪些相同的用户

设计: 将每个用户关注的用户放在集合中，求交集即可

实现:

user001 = {'peiqi','qiaozhi','danni'}

user002 = {'peiqi','qiaozhi','lingyang'}

user001和user002的共同关注为:

SINTER user001 user002

结果为: {'peiqi','qiaozhi'}

python操作set

```
# 1、给name对应的集合中添加元素
sadd(name,values)
r.sadd("set_name","tom")
r.sadd("set_name","tom","jim")

# 2、获取name对应的集合的所有成员：集合
smembers(name)

# 3、获取name对应的集合中的元素个数
scard(name)
r.scard("set_name")

# 4、检查value是否是name对应的集合内的元素:True|False
```

```
sismember(name, value)

# 5、随机删除并返回指定集合的一个元素
spop(name)

# 6、删除集合中的某个元素
srem(name, value)
r.srem("set_name", "tom")

# 7、获取多个name对应集合的交集
sinter(keys, *args)

r.sadd("set_name","a","b")
r.sadd("set_name1","b","c")
r.sadd("set_name2","b","c","d")

print(r.sinter("set_name","set_name1","set_name2"))
#输出：{b'b'}

# 8、获取多个name对应的集合的并集
sunion(keys, *args)
r.sunion("set_name","set_name1","set_name2")
```

python代码实现微博关注

```
import redis

r = redis.Redis(host='192.168.153.136',port=6379,db=0)

# 用户1关注的人
r.sadd('user_first','peiqi','qiaozhi','danni')
# 用户2关注的人
r.sadd('user_second','peiqi','qiaozhi','lingyang')

# user_first和user_second的共同关注的人为？？求差集
result = r.sinter('user_first','user_second')
# 把集合中的每个元素转为string数据类型
focus_on_set = set()
for r in result:
    focus_on_set.add(r.decode())

print(focus_on_set)
```

有序集合sortedset

- 特点

```
1、有序、去重
2、元素是字符串类型
3、每个元素都关联着一个浮点数分值(score)，并按照分支从小到大的顺序排列集合中的元素（分值可以相同）
4、最多包含2^32-1元素
```

- 示例

一个保存了水果价格的有序集合

分值	2.0	4.0	6.0	8.0	10.0
元素	西瓜	葡萄	芒果	香蕉	苹果

一个保存了员工薪水的有序集合

分值	6000	8000	10000	12000
元素	lucy	tom	jim	jack

一个保存了正在阅读某些技术书的人数

分值	300	400	555	666	777
元素	核心编程	阿凡提	本拉登	阿姆斯特朗	比尔盖茨

- **增加**

zadd key score member

```
# 在有序集合中添加一个成员
zadd key score member
# 查看指定区间元素（升序）
zrange key start stop [withscores]
# 查看指定区间元素（降序）
ZREVRANGE key start stop [withscores]
# 查看指定元素的分值
ZSCORE key member
# 返回指定区间元素
# offset : 跳过多少个元素
# count : 返回几个
# 小括号 : 开区间  zrangebyscore fruits (2.0 8.0
zrangebyscore key min max [withscores] [limit offset count]
# 删除成员
zrem key member
# 增加或者减少分值
zincrby key increment member
# 返回元素排名
zrank key member
# 返回元素逆序排名
zrevrank key member
# 删除指定区间内的元素
zremrangebyscore key min max
# 返回集合中元素个数
```

```
zcard key
# 返回指定范围中元素的个数
zcount key min max
zcount fruits 4 7
zcount fruits (4 7
# 并集
zunionstore destination numkeys key [weights 权重值] [AGGREGATE SUM|MIN|MAX]
# 交集：和并集类似，只取相同的元素
ZINTERSTORE destination numkeys key1 key2 WEIGHTS weight AGGREGATE SUM|MIN|MAX
```

- **查看: 指定索引区间元素（升序）**

zrange key start stop [withscores]

```
127.0.0.1:6379> ZRANGE salary 0 -1
1) "lucy"
2) "tom"
3) "jim"
4) "jack"
127.0.0.1:6379> ZRANGE salary 0 -1 withscores
1) "lucy"
2) "6000"
3) "tom"
4) "8000"
5) "jim"
6) "10000"
7) "jack"
8) "12000"
127.0.0.1:6379>
```

- **查看: 指定索引区间元素（降序）**

  ZREVRANGE key start stop [withscores]

- **显示指定元素的分值**

  ZSCORE key member

```
127.0.0.1:6379> zscore salary jack
"14000"
127.0.0.1:6379>
```

- **返回指定区间元素**

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

  offset : 跳过多少个元素

  count : 返回几个

  小括号 : 开区间 zrangebyscore fruits (2.0 8.0

```
127.0.0.1:6379> ZRANGEBYSCORE salary (8000 12000
1) "jim"
2) "jack"
127.0.0.1:6379> ZRANGE salary 0 -1 withscores
1) "lucy"
2) "6000"
3) "tom"
4) "8000"
5) "jim"
6) "10000"
7) "jack"
8) "12000"
```

- **删除**

zrem key member

```
127.0.0.1:6379> ZREM salary jim
(integer) 1
127.0.0.1:6379> ZRANGE salary 0 -1 withscores
1) "lucy"
2) "6000"
3) "tom"
4) "8000"
5) "jack"
6) "12000"
127.0.0.1:6379>
```

- 增加或者减少分值

  zincrby key increment member

```
127.0.0.1:6379> ZINCRBY salary 2000 jack
"14000"
127.0.0.1:6379> ZRANGE salary 0 -1 withscores
1) "lucy"
2) "6000"
3) "tom"
4) "8000"
5) "jack"
6) "14000"
127.0.0.1:6379>
```

- 返回元素的排名（索引）

zrank key member

```
127.0.0.1:6379> zrank salary jack
(integer) 2
127.0.0.1:6379>
```

- 返回元素逆序排名

zrevrank key member

```
127.0.0.1:6379> ZREVRANK salary jack
(integer) 0
127.0.0.1:6379> ZREVRANK salary lucy
(integer) 2
127.0.0.1:6379>
```

- 删除指定区间内的元素

zremrangebyscore key min max

```
127.0.0.1:6379> ZREMRANGEBYSCORE salary 4000 6000
(integer) 1
127.0.0.1:6379> ZRANGE salary 0 -1 withscores
1) "tom"
2) "8000"
3) "jack"
4) "14000"
127.0.0.1:6379>
```

- 返回集合中元素个数

  zcard key

```
127.0.0.1:6379> ZCARD salary
(integer) 2
127.0.0.1:6379>
```

- 返回指定范围中元素的个数

  zcount key min max

  zcount fruits 4 7

  zcount fruits (4 7

```
127.0.0.1:6379> ZRANGE salary 0 -1 withscores
1) "tom"
2) "8000"
3) "jack"
4) "14000"
127.0.0.1:6379> zcount salary 8000 14000
(integer) 2
# 不包含8000，包含14000
127.0.0.1:6379> zcount salary (8000 14000
(integer) 1
127.0.0.1:6379>
```

- 并集

  zunionstore destination numkeys key [weights] [AGGREGATE SUM|MIN|MAX]

```
127.0.0.1:6379> zadd stu_score1 60 tom 70 jim
(integer) 2
127.0.0.1:6379> zadd stu_score2 80 tom 90 lucy
(integer) 2
# 默认为SUM
```

```
127.0.0.1:6379> ZUNIONSTORE stu_score3 2 stu_score1 stu_score2
(integer) 3
127.0.0.1:6379> ZRANGE stu_score3 0 -1 withscores
1) "jim"
2) "70"
3) "lucy"
4) "90"
5) "tom"
6) "140"
127.0.0.1:6379>
# WEIGHTS 和 AGGREGATE
127.0.0.1:6379> ZRANGE stu_score1 0 -1 withscores
1) "tom"
2) "60"
3) "jim"
4) "70"
127.0.0.1:6379> ZRANGE stu_score2 0 -1 withscores
1) "tom"
2) "80"
3) "lucy"
4) "90"
# 权重1给stu_score1，权重0.5给stu_score2，算完权重之后求和SUM
127.0.0.1:6379> ZUNIONSTORE stu_score8 2 stu_score1 stu_score2 weights 1 0.5 AGGREGATE SUM
(integer) 3
127.0.0.1:6379> ZRANGE stu_score8 0 -1 withscores
1) "lucy"
2) "45"
3) "jim"
4) "70"
5) "tom"
6) "100"
127.0.0.1:6379>
```

- **交集**

  ZINTERSTORE destination numkeys key1 key2 WEIGHTS weight AGGREGATE SUM|MIN|MAX

  和并集类似，只取相同的元素

python操作sorted set

```python
import redis

r = redis.Redis(host='192.168.43.49',port=6379,password='123456',db=0)
# 注意第二个参数为字典
r.zadd('salary',{'tom':6000,'jim':8000,'jack':12000})
# 结果为列表中存放元组[(),(),()]
print(r.zrange('salary',0,-1,withscores=True))
print(r.zrevrange('salary',0,-1,withscores=True))
# start:起始值,num:显示条数
print(r.zrangebyscore('salary',6000,12000,start=1,num=2,withscores=True))
# 删除
r.zrem('salary','tom')
print(r.zrange('salary',0,-1,withscores=True))
# 增加分值
r.zincrby('salary',5000,'jack')
print(r.zrange('salary',0,-1,withscores=True))
```

```
# 返回元素排名
print(r.zrank('salary','jack'))
print(r.zrevrank('salary','jack'))
# 删除指定区间内的元素
r.zremrangebyscore('salary',6000,8000)
print(r.zrange('salary',0,-1,withscores=True))
# 统计元素个数
print(r.zcard('salary'))
# 返回指定范围内元素个数
print(r.zcount('salary',6000,20000))
# 并集
r.zadd('salary2',{'jack':17000,'lucy':8000})
r.zunionstore('salary3',('salary','salary2'),aggregate='max')
print(r.zrange('salary3',0,-1,withscores=True))
# 交集
r.zinterstore('salary4',('salary','salary2'),aggregate='max')
print(r.zrange('salary4',0,-1,withscores=True))
```

案例1：网易音乐排行榜

```
1、每首歌的歌名作为元素（先不考虑重复）
2、每首歌的播放次数作为分值
3、使用ZREVRANGE来获取播放次数最多的歌曲
```

代码实现

```
import redis

r = redis.Redis(host='192.168.43.49',port=6379,password='123456',db=0)

r.zadd('ranking',{'song1':1,'song2':1,'song3':1,'song4':1})
r.zadd('ranking',{'song5':1,'song6':1,'song7':1})
r.zadd('ranking',{'song8':1,'song9':1})

r.zincrby('ranking',50,'song3')
r.zincrby('ranking',60,'song5')
r.zincrby('ranking',80,'song7')
# 获取前10名
rlist = r.zrevrange('ranking',0,2,withscores=True)

i = 1
for r in rlist:
    print('第%d名:%s' % (i,r[0].decode()))
    i += 1
```

案例2: 京东商品畅销榜

```
# 第1天
ZADD mobile-001 5000 'huawei' 4000 'oppo' 3000 'iphone'
# 第2天
ZADD mobile-002 5200 'huawei' 4300 'oppo' 3230 'iphone'
# 第3天
ZADD mobile-003 5500 'huawei' 4660 'oppo' 3580 'iphone'
问题：如何获取三款收集的销量排名?
ZUNIONSTORE mobile-001:003 mobile-001 mobile-002 mobile-003 # 可否?
# 正确
1、ZADD mobile-003 5500 'huawei' 4660 'oppo' 3580 'iphone'
2、ZUNIONSTORE mobile-001:003 mobile-001 mobile-002 mobile-003 AGGREGATE MAX
```

python代码实现

```
import redis

r = redis.Redis(host='192.168.43.49',port=6379,password='123456',db=0)

# 第1天
day01_dict = {
    'huawei' : 5000,
    'oppo'   : 4000,
    'iphone' : 3000
}
# 第2天
day02_dict = {
    'huawei' : 5200,
    'oppo'   : 4300,
    'iphone' : 3230
}
# 第3天
day03_dict = {
    'huawei' : 5500,
    'oppo'   : 4660,
    'iphone' : 3580
}
r.zadd('mobile-day01',day01_dict)
r.zadd('mobile-day02',day02_dict)
r.zadd('mobile-day03',day03_dict)

r.zunionstore('mobile-day01:03',('mobile-day01','mobile-day02','mobile-
day03'),aggregate='max')
rlist = r.zrevrange('mobile-day01:03',0,-1,withscores=True)

i = 1
for r in rlist:
    print('第{}名：{}'.format(i,r[0].decode()) )
```