Python操作Redis

连接 Redis

import redis

r = redis.StrictRedis(**"localhost"**,6379,password=**'123456'**)

一 STRING 字符串的操作

1. r.set h设置值

```
#在Redis中设置值，默认不存在则创建，存在则修改
r.set('name', 'zhangsan')
'''参数:
     set(name, value, ex=None, px=None, nx=False, xx=False)
     ex，过期时间（秒）
     px，过期时间（毫秒）
     nx，如果设置为True，则只有name不存在时，当前set操作才执行,同
setnx(name, value)
     xx，如果设置为True，则只有name存在时，当前set操作才执行'''
```

2. r.get 获取值

```
r.get('name')
```

3. mset 批量设置值

```
#批量设置值
r.mset(name1='zhangsan', name2='lisi')
#或
r.mset({"name1":'zhangsan', "name2":'lisi'})
```

4. mget(keys, *args) 批量获取

```
#批量获取
print(r.mget("name1","name2"))
#或
li=["name1","name2"]
print(r.mget(li))
```

5. getset 设置新值，打印原值

```
#设置新值，打印原值
getset(name, value)

print(r.getset("name1","wangwu")) #输出:zhangsan
print(r.get("name1")) #输出:wangwu
```

6. getrange 根据字节获取子序列

```
#根据字节获取子序列
getrange(key, start, end)

r.set("name","zhangsan")
print(r.getrange("name",0,3))#输出:zhan
```

7. setrange 修改字符串内容

从指定字符串索引开始向后替换，如果新值太长时，则向后添加

```
#修改字符串内容，从指定字符串索引开始向后替换，如果新值太长时，则向后添加
setrange(name, offset, value)

r.set("name","zhangsan")
r.setrange("name",1,"z")
print(r.get("name")) #输出:zzangsan
r.setrange("name",6,"zzzzzzz")
print(r.get("name")) #输出:zzangszzzzzzz
```

8. strlen(name)

返回name对应值的字节长度

```
#返回name对应值的字节长度（一个汉字3个字节）
r.set("name","zhangsan")
print(r.strlen("name")) #输出:8
```

9. incr

值的累加 amount为累加的值

```
#自增mount对应的值，当mount不存在时，则创建mount＝amount，否则，则自增,amount为自增数(整数)
incr(self, name, amount=1)

print(r.incr("mount",amount=2))#输出:2
print(r.incr("mount"))#输出:3
print(r.incr("mount",amount=3))#输出:6
print(r.incr("mount",amount=6))#输出:12
print(r.get("mount")) #输出:12
```

10. append

在name对应的值后面追加内容

```
#在name对应的值后面追加内容
append(name, value)

r.set("name","zhangsan")
print(r.get("name"))    #输出:'zhangsan
r.append("name","lisi")
print(r.get("name"))    #输出:zhangsanlisi
```

11.type 查看类型

```
r.type(name)
```

二 Hash 操作

redis中的Hash 在内存中类似于一个name对应一个dic来存储

1. hset

name对应的hash中设置一个键值对（不存在，则创建，否则，修改）

```
#name对应的hash中设置一个键值对（不存在，则创建，否则，修改）
r.hset(name, key, value)
r.hset("dic_name","a1","aa")
```

2. hget

在name对应的hash中根据key获取value

```
r.hset("dic_name","a1","aa")
#在name对应的hash中根据key获取value
hget(name,key)

print(r.hget("dic_name","a1"))#输出:aa
```

3. hgetall 获取name对应hash的所有键值

```
#获取name对应hash的所有键值
hgetall(name)

print(r.hgetall("dic_name"))
```

4. hmset

在name对应的hash中批量设置键值对,mapping:字典

```
#在name对应的hash中批量设置键值对,mapping:字典
hmset(name, mapping)

dic={"a1":"aa","b1":"bb"}
r.hmset("dic_name",dic)
print(r.hget("dic_name","b1"))#输出:bb
```

5. hmget

在name对应的hash中获取多个key的值

```
# 在name对应的hash中获取多个key的值
hmget(name, keys, *args)

li=["a1","b1"]
print(r.hmget("dic_name",li))
print(r.hmget("dic_name","a1","b1"))
```

6.

hlen 获取hash中键值对的个数

hkeys 获取hash中所有的key的值

hvals 获取hash中所有的value的值

```
dic={"a1":"aa","b1":"bb"}
r.hmset("dic_name",dic)

#hlen(name) 获取hash中键值对的个数
print(r.hlen("dic_name"))

#hkeys(name) 获取hash中所有的key的值
print(r.hkeys("dic_name"))

#hvals(name) 获取hash中所有的value的值
print(r.hvals("dic_name"))
```

7. hexists

检查name对应的hash是否存在当前传入的key

```
#检查name对应的hash是否存在当前传入的key
hexists(name, key)

print(r.hexists("dic_name","a1"))#输出:True
```

8. hdel

删除指定name对应的key所在的键值对

```
#删除指定name对应的key所在的键值对
hdel(name,*keys)

r.hdel("dic_name","a1")
```

9. hincrby

自增hash中key对应的值，不存在则创建key=amount(amount为整数)

```
#自增hash中key对应的值，不存在则创建key=amount(amount为整数)
hincrby(name, key, amount=1)

print(r.hincrby("demo","a",amount=2))
```

10.hincrbyfloat

自增hash中key对应的值，不存在则创建key=amount(amount为浮点数)

自增hash中key对应的值，不存在则创建key=amount(amount为浮点数)

hincrbyfloat(name, key, amount=1.0)

三 List 操作

redis中的List在内存中按照一个name对应一个List来存储

1. lpush

在name对应的list中添加元素，每个新的元素都添加到列表的最左边

```
# 在name对应的list中添加元素，每个新的元素都添加到列表的最左边
lpush(name,values)

r.lpush("list_name",2)
r.lpush("list_name",3,4,5)#保存在列表中的顺序为5，4，3，2
```

2.rpush

同lpush，但每个新的元素都添加到列表的最右边

```
#同lpush，但每个新的元素都添加到列表的最右边
rpush(name,values)
```

3. lpushx

在name对应的list中添加元素，只有name已经存在时，值添加到列表的最左边

```
#在name对应的list中添加元素，只有name已经存在时，值添加到列表的最左边
 lpushx(name,value)
```

4. rpushx

在name对应的list中添加元素，只有name已经存在时，值添加到列表的最右边

```
#在name对应的list中添加元素，只有name已经存在时，值添加到列表的最右边
 rpushx(name,value)
```

5. llen

name对应的list元素的个数

```
# name对应的list元素的个数
llen(name)

print(r.llen("list_name"))
```

6. linsert

在name对应的列表的某一个值前或后插入一个新值

```
# 在name对应的列表的某一个值前或后插入一个新值
 linsert(name, where, refvalue, value))
r.linsert("list_name","BEFORE","2","SS")#在列表内找到第一个元素2，在它前面插入SS

'''参数:
    name: redis的name
    where: BEFORE（前）或AFTER（后）
    refvalue: 列表内的值
    value: 要插入的数据'''
```

7. r.lset

对list中的某一个索引位置重新赋值

```
#对list中的某一个索引位置重新赋值
r.lset(name, index, value)

r.lset("list_name",0,"bbb")
```

8. r.lrem

删除name对应的list中的指定值

```
#删除name对应的list中的指定值
r.lrem(name, count, value)
r.lrem("list_name",3,'ssss')

''' 参数:
    name:  redis的name
    value: 要删除的值
    num:   num=0 删除列表中所有的指定值;
           num=2 从前到后，删除2个;
           num=-2 从后向前，删除2个'''
```

9. lpop

移除列表的左侧第一个元素，返回值则是第一个元素

```
#移除列表的左侧第一个元素，返回值则是第一个元素
lpop(name)

print(r.lpop("list_name"))
```

10. lindex

根据索引获取列表内元素

```
#根据索引获取列表内元素
lindex(name, index)

print(r.lindex("list_name",1))
```

11. lrange

分片获取元素

```
#分片获取元素
lrange(name, start, end)

print(r.lrange("list_name",0,-1))
```

12. ltrim

移除列表内没有在该索引之内的值(裁剪)

```
#移除列表内没有在该索引之内的值
ltrim(name, start, end)

r.ltrim("list_name",0,2)
```

13. rpoplpush(src, dst)

从一个列表取出最右边的元素，同时将其添加至另一个列表的最左边

```
# 从一个列表取出最右边的元素，同时将其添加至另一个列表的最左边
#src 要取数据的列表
#dst 要添加数据的列表
```

14. brpoplpush(src, dst, timeout=0)

```
#同rpoplpush，多了个timeout，timeout：取数据的列表没元素后的阻塞时间，0为一直阻塞
r.brpoplpush("list_name","list_name1",timeout=0)
```

15blpop(keys, timeout)

```
#将多个列表排列,按照从左到右去移除各个列表内的元素
r.lpush("list_name",3,4,5)
r.lpush("list_name1",3,4,5)

while True:
    print(r.blpop(["list_name","list_name1"],timeout=0))
    print(r.lrange("list_name",0,-1),r.lrange("list_name1",0,-1))

'''keys: redis的name的集合
   timeout: 超时时间，获取完所有列表的元素之后，阻塞等待列表内有数据的时间（秒）, 0 表示永远阻塞'''
```

16. r.brpop(keys, timeout)

```
#同blpop，将多个列表排列,按照从右像左去移除各个列表内的元素
```

四 Set 操作

Set集合就是不允许重复的列表

1. sadd(name,values)

给name对应的集合中添加元素

```
#给name对应的集合中添加元素
r.sadd("set_name","aa")
r.sadd("set_name","aa","bb")
```

2. smembers(name)

获取name对应的集合的所有成员

```
#获取name对应的集合的所有成员
```

3. scard(name)

获取name对应的集合中的元素个数

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

4. sdiff(keys, *args)

```
#在第一个name对应的集合中且不在其他name对应的集合的元素集合
r.sadd("set_name","aa","bb")
r.sadd("set_name1","bb","cc")
r.sadd("set_name2","bb","cc","dd")

print(r.sdiff("set_name","set_name1","set_name2"))#输出：{aa}
```

5. sdiffstore(dest, keys, *args)

```
#相当于把sdiff获取的值加入到dest对应的集合中
```

6. sinter(keys, *args)

获取多个name对应集合的交集

```
# 获取多个name对应集合的并集
r.sadd("set_name","aa","bb")
r.sadd("set_name1","bb","cc")
r.sadd("set_name2","bb","cc","dd")

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

7. sinterstore(dest, keys, *args)

```
#获取多个name对应集合的并集，再讲其加入到dest对应的集合中
```

8.sismember

检查value是否是name对应的集合内的元素

```
#检查value是否是name对应的集合内的元素
sismember(name, value)
```

9. smove(src, dst, value)

将某个元素从一个集合中移动到另外一个集合

```
#将某个元素从一个集合中移动到另外一个集合
```

10. spop(name)

从集合的右侧移除一个元素，并将其返回

```
#从集合的右侧移除一个元素，并将其返回
```

11. srandmember(name, numbers)

从name对应的集合中随机获取numbers个元素

```
# 从name对应的集合中随机获取numbers个元素
print(r.srandmember("set_name2",2))
```

12. srem(name, values)

删除name对应的集合中的某些值

```
#删除name对应的集合中的某些值
print(r.srem("set_name2","bb","dd"))
```

13. sunion(keys, *args)

获取多个name对应的集合的并集

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

14. sunionstore(dest,keys, *args)

获取多个name对应的集合的并集，并将结果保存到dest对应的集合中

```
#获取多个name对应的集合的并集，并将结果保存到dest对应的集合中
```

五 有序集合 zset

有序集合：

在集合的基础上，为每元素排序，元素的排序需要根据另外一个值来进行比较，所以，对于有序集合，每一个元素有两个值，即：值和分数，分数专门用来做排序。

1. zadd(name, *args, **kwargs)

```
# 在name对应的有序集合中添加元素
r.zadd("zset_name", 6,"a1", 2, "a2", 5,"a3")
#或
r.zadd('zset_name1', b1=10, b2=5)
```

2. zcard(name)

获取有序集合内元素的数量

```
#获取有序集合内元素的数量
```

3. zcount(name, min, max)

获取有序集合中分数在[min,max]之间的个数

```
#获取有序集合中分数在[min,max]之间的个数
print(r.zcount("zset_name",1,5))
```

4. zincrby(name, value, amount)

自增有序集合内value对应的分数

```
#自增有序集合内value对应的分数
r.zincrby("zset_name","a1",amount=2)#自增zset_name对应的有序集合里a1对应的分数
```

5. zrange(name, start, end, desc=False, withscores=False, score_cast_func=float)

```
# 按照索引范围获取name对应的有序集合的元素
aa=r.zrange("zset_name",0,1,desc=False,withscores=True,score_cast_fun
c=int)
print(aa)
'''参数:
    name    redis的name
    start   有序集合索引起始位置
    end     有序集合索引结束位置
    desc    排序规则，默认按照分数从小到大排序
    withscores  是否获取元素的分数，默认只获取元素的值
    score_cast_func 对分数进行数据转换的函数'''
```

6. zrevrange(name, start, end, withscores=False, score_cast_func=float)

```
#同zrange，集合是从大到小排序的
```

7. zrank(name, value)、zrevrank(name, value)

```
#获取value值在name对应的有序集合中的排行位置（从0开始）
print(r.zrank("zset_name", "a2"))

print(r.zrevrank("zset_name", "a2"))#从大到小排序
```

8. zscore(name, value)

获取name对应有序集合中 value 对应的分数

```
#获取name对应有序集合中 value 对应的分数
print(r.zscore("zset_name","a1"))
```

9. zrem(name, values)

删除name对应的有序集合中值是values的成员

```
#删除name对应的有序集合中值是values的成员
r.zrem("zset_name","a1","a2")
```

10. zremrangebyrank(name, min, max) 根据排行范围删除

```
#根据排行范围删除
```

11. zremrangebyscore(name, min, max) 根据分数范围删除

```
#根据分数范围删除
```

12. zinterstore(dest, keys, aggregate=None)

```
r.zadd("zset_name", "a1", 6, "a2", 2,"a3",5)
r.zadd('zset_name1', a1=7,b1=10, b2=5)

# 获取两个有序集合的交集并放入dest集合，如果遇到相同值不同分数，则按照aggregate进行操作
# aggregate的值为: SUM  MIN  MAX
r.zinterstore("zset_name2",
("zset_name1","zset_name"),aggregate="MAX")
print(r.zscan("zset_name2"))
```

13. zunionstore(dest, keys, aggregate=None)

```
#获取两个有序集合的并集并放入dest集合，其他同zinterstore,
```

其他常用操作

1. delete(*names)

根据name删除redis中的任意数据类型

```
#根据name删除redis中的任意数据类型
```

2. exists(name)

检测redis的name是否存在

```
#检测redis的name是否存在
```

3. keys(pattern='*')

根据*？等通配符匹配获取redis的name

```
#根据* ？等通配符匹配获取redis的name
```

4. expire(name ,time)

为某个name设置超时时间

```
# 为某个name设置超时时间
```

5. rename(src, dst) 重命名

```
# 重命名
```

6. move(name, db))

将redis的某个值移动到指定的db下

```
# 将redis的某个值移动到指定的db下
```

7. randomkey()

随机获取一个redis的name（不删除）

```
#随机获取一个redis的name（不删除）
```

8. type(name)

获取name对应值的类型

```
# 获取name对应值的类型
```

六、管道

缓存多条命令、依次执行，可以减少服务器和客户端之间的传输次数，从而提高效率

```
pipe = conn.pipeline()
pipe.set("code1", "1111")
pipe.set("code2", "2222")
pipe.set("code3", "3333")
pipe.set("code4", "4444")
pipe.excute()
```