第4天-操作数据库

一、 操作 MySQL

1. 基本介绍

Python3 操作 MySQL 数据库 可以使用的模块是 `pymysql` 和 `MySQLdb`。

这个两个模块都是通过自己的 API 执行原生的 SQL 语句实现的。

MySQLdb 是最早出现的一个操作 MySQL 数据库的模块，核心由C语言编写，接口精炼，性能最棒，缺点是环境依赖较多，安装稍复杂，特别是 windows 中不好安装。更新较慢。

`pymysql` 为替代 `MySQLdb` 而生，纯 `Python` 实现，API 的接口与 `MySQLdb` 完全兼容，安装方便。

```
default charset utf8mb4 collate utf8mb4_general_ci;
```

2. 安装包 `pymysql`

pymsql是Python中操作MySQL的模块

```
shell> pip3 install pymysql
```

3. 基本操作

示例数据

```
info = {
    "base_info": {
        "manufacturer": "VMware, Inc.",
        "pod_name": "VMware7,1",
        "sn": "VMware-56 4d 2b 4b 91 1e 48 15-5b d2
73 9c ec 98 da 22",
        "host_name": "qfedu.com",
        "kernel": "3.10.0-957.el7.x86_64",
        "os": "CentOS Linux release 7.6.1810 (Core)"
    },
    "cpu": {
        "cpu_name": "Intel(R) Core(TM) i5-5350U CPU
@ 1.80GHz",
        "cpu_pyc": 1,
        "cpu_cores_each": 1
    },
    "memory": [
        {
            "capacity": " 8192 MB",
            "slot": " DIMM_B2",
            "model": " DDR3",
            "speed": " 1333 MT/s",
            "manufacturer": " 00CE00B380CE",
            "sn": " 82B79F71"
        },
        {
            "capacity": " 8192 MB",
            "slot": " DIMM_B3",
            "model": " DDR3",
            "speed": " 1333 MT/s",
            "manufacturer": " 00CE00B380CE",
            "sn": " 32CDDE81"
        },
        {
```

```
            "capacity": " No Module Installed",
            "slot": " DIMM_B4",
            "model": " DDR3",
            "speed": " Unknown",
            "manufacturer": "",
            "sn": ""
        },
        {
            "capacity": " 8192 MB",
            "slot": " DIMM_B5",
            "model": " DDR3",
            "speed": " 1333 MT/s",
            "manufacturer": " 00CE04B380CE",
            "sn": " 85966B82"
        },
        {
            "capacity": " 8192 MB",
            "slot": " DIMM_B6",
            "model": " DDR3",
            "speed": " 1333 MT/s",
            "manufacturer": " 000000B380CE",
            "sn": " 00000000"
        }
    ]
}
```


3.1 创建表

```
import pymysql

# 创建连接
```

```
conn = pymysql.connect(host='172.16.153.10',
                       port=3306,
                       user='root',
                       passwd='123',
                       db='shark_db',
                       charset='utf8mb4')
# 获取游标对象
cursor =
conn.cursor(cursor=pymysql.cursors.DictCursor)

# 定义 sql 语句, 创建第一个表 服务器基础信息表 base_info
sql1 = """create table base_info
 (id int auto_increment primary key,
  host_name varchar(64) not null,
  kernel varchar(64),
  os varchar(64),
  manufacturer varchar(32),
  pod_name varchar(64),
  sn varchar(128),
  cpu_name varchar(64),
  cpu_pyc int not null,
  cpu_cores_each int not null)"""

# 创建第二个表 内存表  memory
sql2 = '''
create table memory(
    id int auto_increment primary key,
    capacity varchar(32),
    slot varchar(16),
    model varchar(4),
    speed varchar(32),
    manufacturer varchar(128),
    sn varchar(128),
    server_id int
)
```

```
'''

# 执行 sql 语句
cursor.execute(sql1)
cursor.execute(sql2)


# 提交更改
conn.commit()

# 关闭游标对象
cursor.close()

# 关闭连接对象
conn.close()
```


3.2 插入数据

一次插入一条数据

```
# 一次插入一条数据，并且使用 pymysql 定义的变量占位符
insert_data_sql = '''insert into base_info(
                        {},{},{},{},{},{},{},{},{}
                ) values(
                %s, %s, %s, %s, %s, %s, %s, %s, %s);'''

# 注意这里不是 Python 的字符串格式化，所以传值的时候不需要使用 %
cursor.execute(insert_data_sql, tuple(base_info.values()))

# 数据库中受影响的行数
```

```
print(cursor.rowcount)

conn.commit()
cursor.close()
conn.close()
```

一次插入多条数据

```
# 处理数据

mem_info =  info["memory"]
mem_li = []
server_id = 1
for mem in mem_info:
    mem["server_id"] = server_id
    v = tuple(mem.values())
    mem_li.append(v)

# 获取对应的 key
mem = mem_info[0]
m_keys = mem.keys()

# 一次插入一条数据，并且使用 pymysql 定义的变量占位符 %s
sql = '''insert into memory({},{},{},{},{},{},{}
        ) values(%s, %s, %s, %s, %s, %s, %s);'''

sql = sql.format(*m_keys)
# print(sql)

# 语法:
# cursor.executemany(sql, [("v1","v2"),("v3",
"v4")])

```

```
cursor.executemany(sql, mem_li)

conn.commit()
cursor.close()
conn.close()
```

3.3 查询数据

3.3.1 获取到的数据是元组类型

```
定义一个查询语句
query_sql = "select id, host_name, os from base_info;"

执行查询语句，并且返回得到结果的行数
row_nums = cursor.execute(query_sql, ('shark1'))

"""
获取到数据结果集具有迭代器的特性：
1. 可以通过索引取值，可以切片
2. 结果集中的数据每次取出一条就少一条
"""

获取结果集中的第一条数据， 注意不是整个表的第一条数据
one_data = cursor.fetchone()

获取结果集中接下来的第 2 条和 第 3 条 数据
many_data = cursor.fetchmany(2)

获取结果集中剩余的全部数据
all_data = cursor.fetchall()

cursor.close()
```

```
conn.close()
print(row_nums)
print(one_data)
print(many_data)
print(all_data)
```

二、redis 的安装和基本使用

1. 简单介绍

redis是一个key-value存储系统，和Memcached类似。它支持存储的value类型相对更多，包括string(字符串)、list(链表、set(集合)、zset(sorted set --有序集合)和hash（哈希类型）。

这些数据类型都支持push/pop、add/remove及取交集并集和差集及更丰富的操作，而且这些操作都是原子性的。在此基础上，redis支持各种不同方式的排序。

与memcached一样，为了保证效率，数据都是缓存在内存中。区别的是redis会周期性的把更新的数据写入磁盘或者把修改操作写入追加的记录文件，并且在此基础上实现了master-slave(主从)同步。

2. 软件包

```
yum install  epel-release   ## 保证安装 epel 源
yum install redis
```

3. 启动服务

```
[root@kube-master ~]# systemctl start redis
```

检查服务状态

```
[root@kube-master ~]# systemctl status redis
● redis.service - Redis persistent key-value database
   Loaded: loaded (/usr/lib/systemd/system/redis.service; disabled; vendor pr
ed)
  Drop-In: /etc/systemd/system/redis.service.d
           └─limit.conf
   Active: active (running) since Mon 2019-12-23 07:53:17 CST; 4s ago
 Main PID: 22732 (redis-server)
    Tasks: 3
   Memory: 4.2M
   CGroup: /system.slice/redis.service
           └─22732 /usr/bin/redis-server 127.0.0.1:6379
```

检查端口

Redis 默认监听 6379 端口

```
[root@kube-master ~]#
[root@kube-master ~]# lsof -i :6379          方式一
COMMAND     PID  USER   FD   TYPE   DEVICE SIZE/OFF NODE NAME
redis-ser 22732 redis    4u  IPv4 30673512      0t0  TCP localhost:6379 (LISTEN)
[root@kube-master ~]# ss -ntal |grep 6379    方式二
LISTEN     0      128    127.0.0.1:6379                 *:*
[root@kube-master ~]#
```

4. 基本使用

4.1 连接到 redis-server

```
使用默认端口连接到本地 redis 服务
[root@kube-master ~]# redis-cli
127.0.0.1:6379>
```

> 退出输入客户端 `exit`

4.2 增删改查数据

4.2.1 增加

`set`

```
127.0.0.1:6379> help set
SET key value [EX seconds] [PX milliseconds] [NX|XX]
```

> 在 Redis 中设置值，默认，不存在则创建，存在则修改 参数： ex，过期时间（秒） px，过期时间（毫秒） nx，假如设置为True，则只有 name 不存在时，当前 set 操作才执行 xx，假如设置为True，则只有 name 存在时，当前 set 操作才执行

Example

```
127.0.0.1:6379> set name shark EX 10
OK

```

4.2.2 获取

`get`

```
127.0.0.1:6379> get name
"shark"
```

4.2.3 查询过期时间

`ttl`

查看一个 key 的过期时间

```
127.0.0.1:6379> ttl name
(integer) 2    # 距离过期时间，剩余的秒数
```

- -1 永不过期
- -2 已经过期

4.2.4 指定过期时间

`expire`

设置一个 key 的过期时间（单位: 秒）

```
127.0.0.1:6379> set age 18
OK
127.0.0.1:6379> ttl age
(integer) -1
127.0.0.1:6379> EXPIRE age 20
(integer) 1
127.0.0.1:6379> ttl age
(integer) 18
127.0.0.1:6379> ttl age
(integer) 14
127.0.0.1:6379>
```

4.2.5 删除

`del`

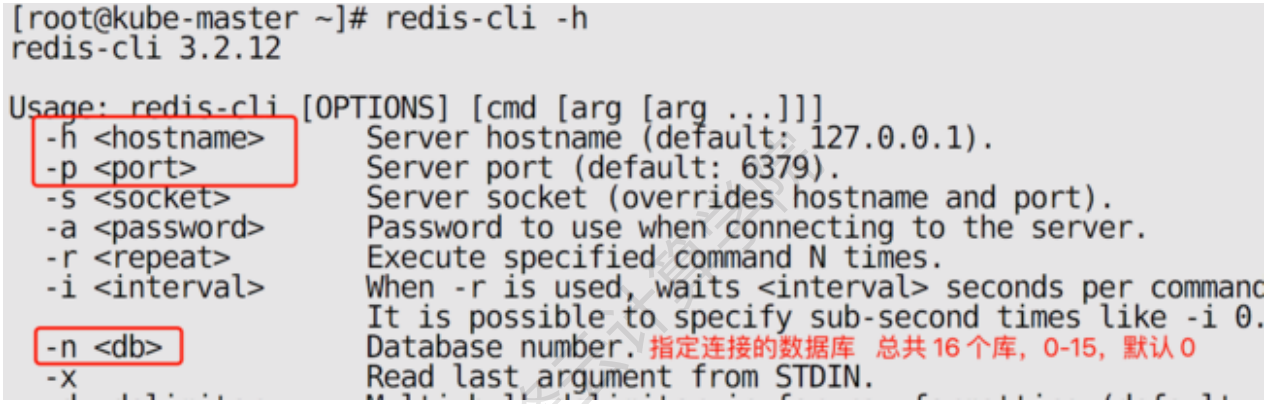
```
[root@kube-master ~]# redis-cli set name shark
OK
[root@kube-master ~]# redis-cli get name
"shark"
[root@kube-master ~]# redis-cli del name
(integer) 1
[root@kube-master ~]# redis-cli get name
(nil)
```


4.3 配置文件

路径 `/etc/redis.conf`

```
# 设置监听地址
bind 127.0.0.1

# 设置监听端口
port 6379

# 设置密码为 foo
requirepass foo

```

4.4 其他连接方式（扩展自修）



三、python3 操作 redis

1. 安装包

```
pip3 install redis
```

2. 连接

```
In [1]: import redis

In [2]: rs = redis.StrictRedis(host='192.168.1.37',
                               port=6379,
                               db=0,
                               decode_responses=True
In [3]: rs.set("QF", "www.qfedu.com")
Out[3]: True

In [4]: rs.get("QF")
Out[4]: 'www.qfedu.com'
```

- `db=0` 指定使用那个数据库， `0-15` 共 16 个，默认是 `0`
- `decode_responses=True` redis 返回值默认是二进制 bytes 类型，设置为 `True` 可直接返回字符串，默认使用 `utf-8`

3. 基本操作

3.1 set 在 Redis 中设置键值对值，默认，不存在则创建，存在则修改

```
50
"""
 参数:
          ex，过期时间（秒）
          px，过期时间（毫秒）
          nx，假如设置为True，则只有 name 不存在时，当前 set 操作才执行
          xx，假如设置为True，则只有 name 存在时，当前 set 操作才执行
"""
```

- 示例

```
In [121]: rs.set("name", "shark",ex=50)
Out[121]: True

In [122]: rs.ttl("name")  # 查看过期时间
Out[122]: 41

In [123]: rs.get("name")
Out[123]: 'shark'
```

3.2 setex 设置键值对，并设置过期时间（单位秒）

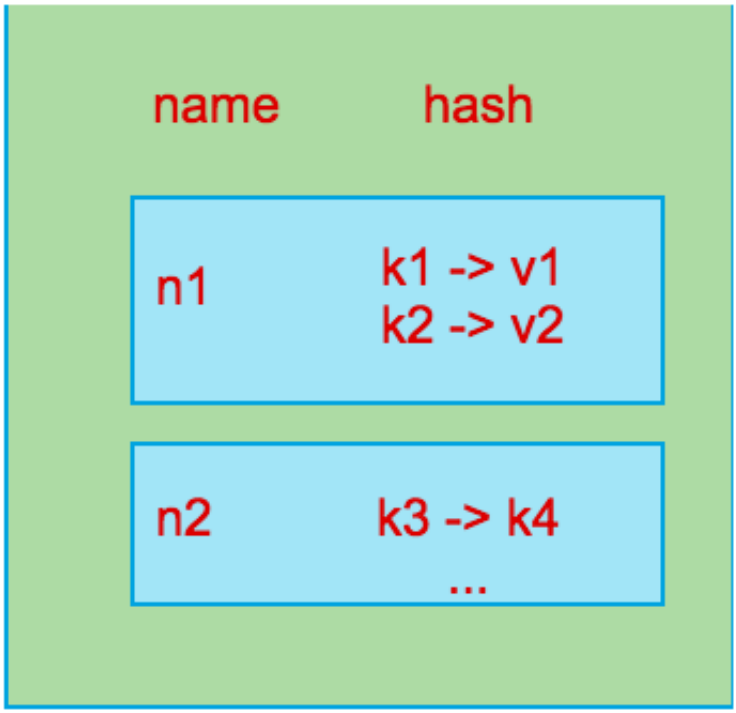
```
setex(name,  time, value)

"""
 参数:
     time，过期时间（数字秒）
"""
```

- 示例


```
In [26]: rs.setex('login', 30, 1)
Out[26]: True

In [27]: rs.ttl("login")
Out[27]: 27

In [28]: rs.get("login")
Out[28]: '1'

```

3.3 Hash 操作

Redis 中的 Hash 数据，很想 Python 中嵌套的字典：`{"n1": {"k1": "v1", "k2", "v2"}}`




```
In [29]: user_info = {"name": "shark", "age": 18}

In [30]: rs.hmset("shark", user_info)
Out [30]: True

In [31]: rs.hget(name, "age")
In [22]: 18
```


4. 使用场景

比如我们想存储用户的登录信息 一般用户的登录信息有一下几点：

- 用户名 name
- 加密的密码 password
- 会话 session_id
- 会话过期时间 expiration_date

那现在就来模拟一下一个用户从登录到登出（过期）的情况

预备知识，我们模拟用户登录情况需要使用如下几个模块

- hashlib
- string
- random

4.1 密码加密功能实现

第一步我们实现加密密码

> **`hashlib` 提供了常见的摘要算法，如MD5，SHA1等等。**
>
> 什么是摘要算法呢？摘要算法又称哈希算法、散列算法。它通过一个函数，把任意长度的数据转换为一个长度固定的数据串（通常用16进制的字符串表示）。 举个例子，假设有一段字符串 `data = 'hello'`，使用摘要算法函数 `f(data)` 进行计算，会返回一个固定长度的16进制数字 hexdigest。 摘要算法的好处是：
>
> 1. 无法反推，因为摘要函数是一个单向函数，计算f(data)很容易，但通过`hexdigest`反推`data`却非常困难。
> 2. 对原始数据做一个bit的修改，都会导致计算出的摘要完全不同。

```
In [40]: import hashlib

In [41]: hashlib.md5("hello".encode("utf-8"))
Out[41]: <md5 HASH object @ 0x10fc5deb8>

In [42]: hmd5 = hashlib.md5("hello".encode("utf-8"))  # 接收的是字节（二进制数据）

In [43]: hmd5.hexdigest()
Out[43]: '5d41402abc4b2a76b9719d911017c592'
```

```
In [44]: hmd5 = hashlib.md5("hello".encode("utf-8"))   # 多次计算，是等幂操作，加密的值是一样的

In [45]: hmd5.hexdigest()
Out[45]: '5d41402abc4b2a76b9719d911017c592'
```

4.2 生成会话 ID 随机数

`string` 可以产生有序的数字和英文大小写字母

```
In [55]: import string

In [56]: string.digits
Out[56]: '0123456789'

In [57]: string.ascii_letters
Out[57]: 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'

In [58]:
```


`random` 可以把一个序列类型的数据打乱，这样我们就可以得到一个随机字符串，用于会话 ID

```
In [65]: import random, string

In [66]: base_str = f"{string.digits + string.ascii_letters}"
```

```
In [67]: base_str
Out[67]: '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'

In [68]: random.sample(base_str, 6)   # 从传入的序列数据中产生 6 个随机数据
Out[68]: ['Q', '4', 'y', 'a', 'R', 'd']

In [69]: li = random.sample(base_str, 6)

In [70]: sessid = ''.join(li)

In [71]: sessid
Out[71]: 'B1umkn'

In [72]:
```


4.3 信息写入 Redis

当用户登录的时候我们需要添加多个数据，可以使用 `hmset()` 方法

```
"""
hmset(name, mapping)
name 就是一个hash 的名字 ， mapping 就是一个字典
在name对应的hash中批量设置键值对，没有的 key 就创建，已存在的 key 修改
"""

In [78]: import random, string, hashlib

In [79]: name="shark"
```

```
In [80]: base_str = f"{string.digits + string.ascii_letters}"

In [81]: sessid = ''.join(random.sample(base_str, 20))

In [82]: sessid
Out[82]: 'SFMNr3jzJZTRYLy64qfb'

In [83]: password = hashlib.md5("QFedu123!".encode('utf-8'))

In [84]: password = password.hexdigest()

In [85]: password
Out[85]: '19b169cff70f61b4e80663757025a17d'

In [86]: user_info = {"name": name, "password": password, "sessid": sessid}

In [87]: rs.hmset(name, user_info)
Out[87]: True

In [88]: rs.setex(f"{name}_sessid",86400,sessid))
Out[88]: True
```


每次用户再次跳转页面或者发送新的请求时，我们都应该查看他的会话是否超时。

首先，获取 redis 中 Hash 数据使用的是 `hget()` 和 `hmget` 方法

> ## hget(name,key)
>
> 在name对应的hash中获取根据key获取value

```
In [155]: rs.hget(name, 'sessid')
Out[155]: 'SFMNr3jzJZTRYLy64qfb'
```

hmget(name, keys, *args)

在name对应的hash中获取多个key的值 参数： name，reids对应的name keys，要获取key集合，例如：['k1', 'k2', 'k3'] *args，要获取的key，例如：k1,k2,k3

```
In [156]: rs.hmget(name, ["name", "sessid", "login"])
Out[156]: ['shark', 'SFMNr3jzJZTRYLy64qfb', 'True']

In [157]: name, sessid, login = rs.hmget(name,
["name", "sessid", "login"])

In [158]: login
Out[158]: 'True'
```

模拟登录状态OK， 需要进一步检查 会话是否超时的情况

```
In [164]: rs.hmset(name, {"sessid": '', "login":
"False"})
Out[164]: True

In [165]: rs.hget(name,"sessid")
Out[165]: ''

In [166]: rs.hget(name,"login")
Out[166]: 'False'
```

三、小练习：

1. 需求

编写一个脚本，把上面我们将的用户信息存入到 Redis 中。 实现一个简单的登录验证功能。

一般用户的每次请求，都会携带自己会话（session） ID

这里就分两种情况：

- 用没有携带 session ID 就说明是第一次登录，就必须是携带用户名密码的。此时就走验证用户名密码的流程。
- 用户携带了 session ID 就走验证 session 是否过期的流程。

验证用户名密码流程:

1. 提示用户输入用户名密码
2. 不正确提示重新输入
3. 正确，创建用户 session id 和 有效时间，

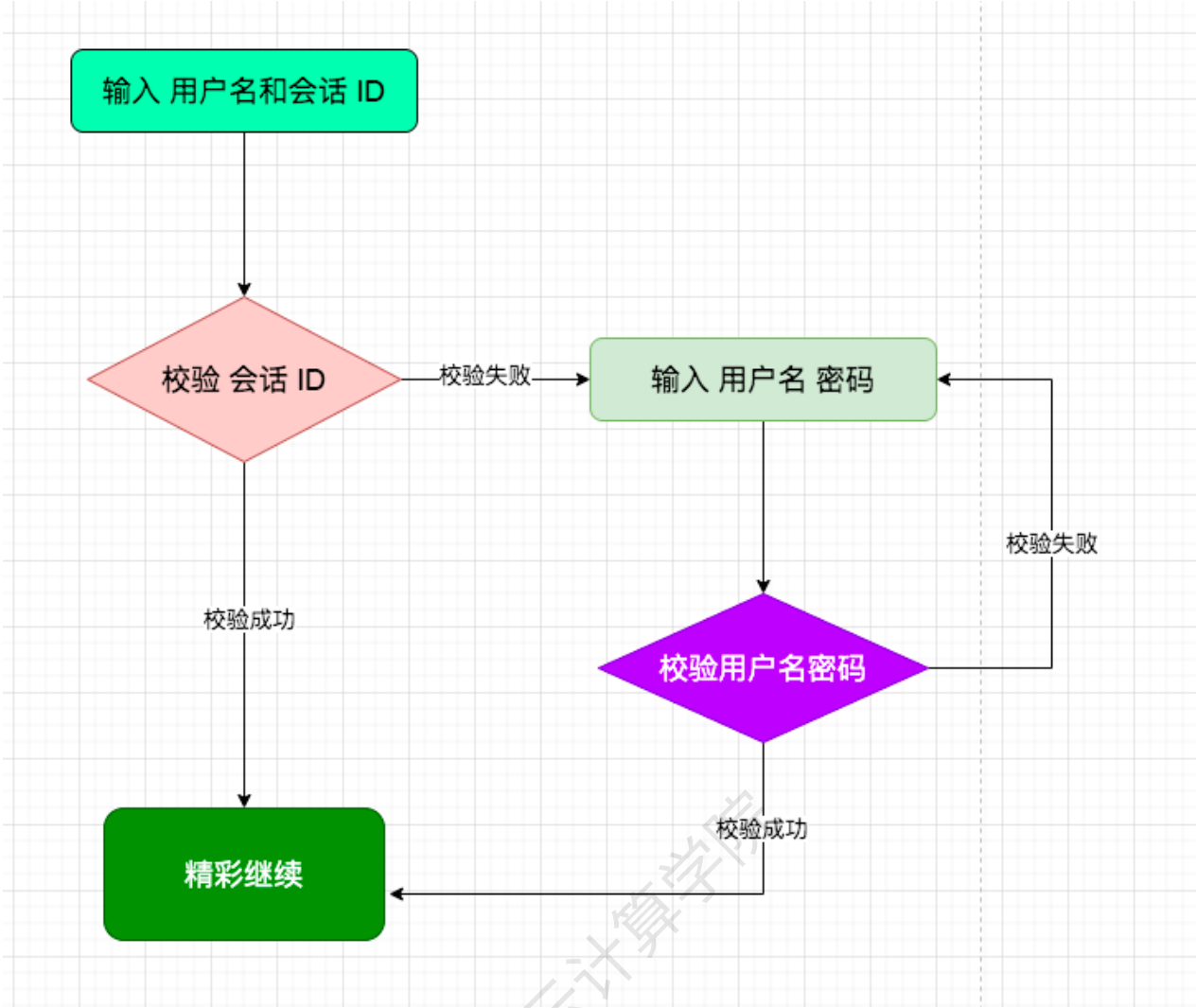
   并保存到 Redis 中。就显示”精彩继续”。

   由于模拟的原因，需要打印出 session ID

验证 session ID 是否过期流程:

1. 过期了，重新走用户登录流程
2. 不过期，就显示”精彩继续”

2. 流程图：



3. 运行时的效果

```
输入 会话 ID
格式： name sessionid>>:d d
输入用户名和密码,中间用空格隔开, 比如: shark 123
>>:shark QFedu123                    输入用户名 密码
shark MJ1hum69FY42nkGSoOji           输出用户名 密码
精彩继续
输入 会话 ID
格式： name sessionid>>:shark MJ1hum69FY42nkGSoOji    会话未超时
精彩继续
输入 会话 ID
格式： name sessionid>>:shark MJ1hum69FY42nkGSoOji    会话超时
输入用户名和密码,中间用空格隔开, 比如: shark 123
>>:
```

4. 参考源码

```python
import random, string, hashlib

import redis

rs = redis.StrictRedis(
    host="172.16.153.189",
    port=6379,db=0,
    decode_responses=True,
    password="foo")

def generate_sessid():
    """生产随机字符串
    Return: 返回 session id
    """

    # 获取基本的字符串0-9 a-z A-Z
    base_str = f"{string.digits + string.ascii_letters}"
    # 拼接
    sessid = ''.join(random.sample(base_str, 20))
    return sessid

def hash_pwd(pwd):
    """传入一串字符，进行 MD5 计算
    Keyword arguments:
    pwd -- 字符串
    Return: md5 加密值
    """

    password = hashlib.md5(pwd.encode('utf-8'))
    return password.hexdigest()

def save_session(name, sessid='', ex=7200):
    """保存会话信息，并存入缓存
```

```
    Keyword arguments:
    name -- 用户名
    pwd -- 加密过的密码
    sessid -- 会话 id
    Return:
    """
    rs.hset(name, "sessid", sessid)
    rs.setex(f"{name}_{sessid}", ex, sessid)


def check_session(name, sessid):
    if rs.ttl(f"{name}_{sessid}") in (-2, 0):
        return False
    else:
        return True


def login():
    """
    用户登录，验证，验证成功返回用户名
    :return 用户名和 会话 ID
    """
    while True:
        inp = input("输入用户名和密码,中间用空格隔开，比如: shark 123\n>>:").strip()
        user_pwd = inp.split()
        if len(user_pwd) == 2:
            name, source_pwd = user_pwd
        else:
            print("格式错误,请重新输入")
            continue
        # 获取加密密文
        pwd = hash_pwd(source_pwd)
        # 获取缓存中加密密文
        user = rs.hget(name, "name")
```

```
        redis_pwd = rs.hget(name, "password")

        # 验证用户名和密码
        if name == user and pwd == redis_pwd:
            return name, generate_sessid()
        else:
            print("用户名密码错误, 请重新输入")
            continue



def main():
    while True:
        inp = input("输入 会话 ID\n格式:  name sessionid>>:").strip()
        if len(inp.split()) == 2:
            name, sessid = inp.split()
            ex = check_session(name, sessid)
        else:
            print("格式错误 name  session")
            continue
        if ex:
            print("精彩继续")
        else:
            name, sessid = login()
            save_session(name, sessid,ex=10)
            print(name, sessid)
            print("精彩继续")

if __name__ == "__main__":
    main()
```