1+X Python程序开发（中级）

一、 证书标准解读

1.1 考试级别

Python开发职业技能分为初、中、高三个等级

1.2 考核方式与题型

考核方式为闭卷考试，采用上机考试形式。考试包括理论考试和实操考试两部分。理论考试时长90分钟，试卷满分100分，共50道试题，其中单选题30题，多选10题，判断10题；实操考试时长150分钟，试卷满分100分，试卷含5道实践性试题，试题形式包括案例分析、软件代码编码或是网页效果呈现等。

1.3 合格标准

理论考试试卷满分为100分，权重40%;实操考试试卷满分为100分，权重60%。综合成绩等于理论和实操考试成绩的加权之和，综合成绩合格标准为大于等于60分，综合成绩合格的学员可以获得相应级别的职业技能等级证书。

1.4 考试计算机环境配置

1.4.1 硬件

CPU：四核以上，主频2. 0GHz以上

内存：4GB

硬盘：50G空闲

网卡：1000M

1.4.2 软件

操作系统：Window10

开发工具：PyCharm2019以上

输入法要求：安装英文、拼音、五笔输入。

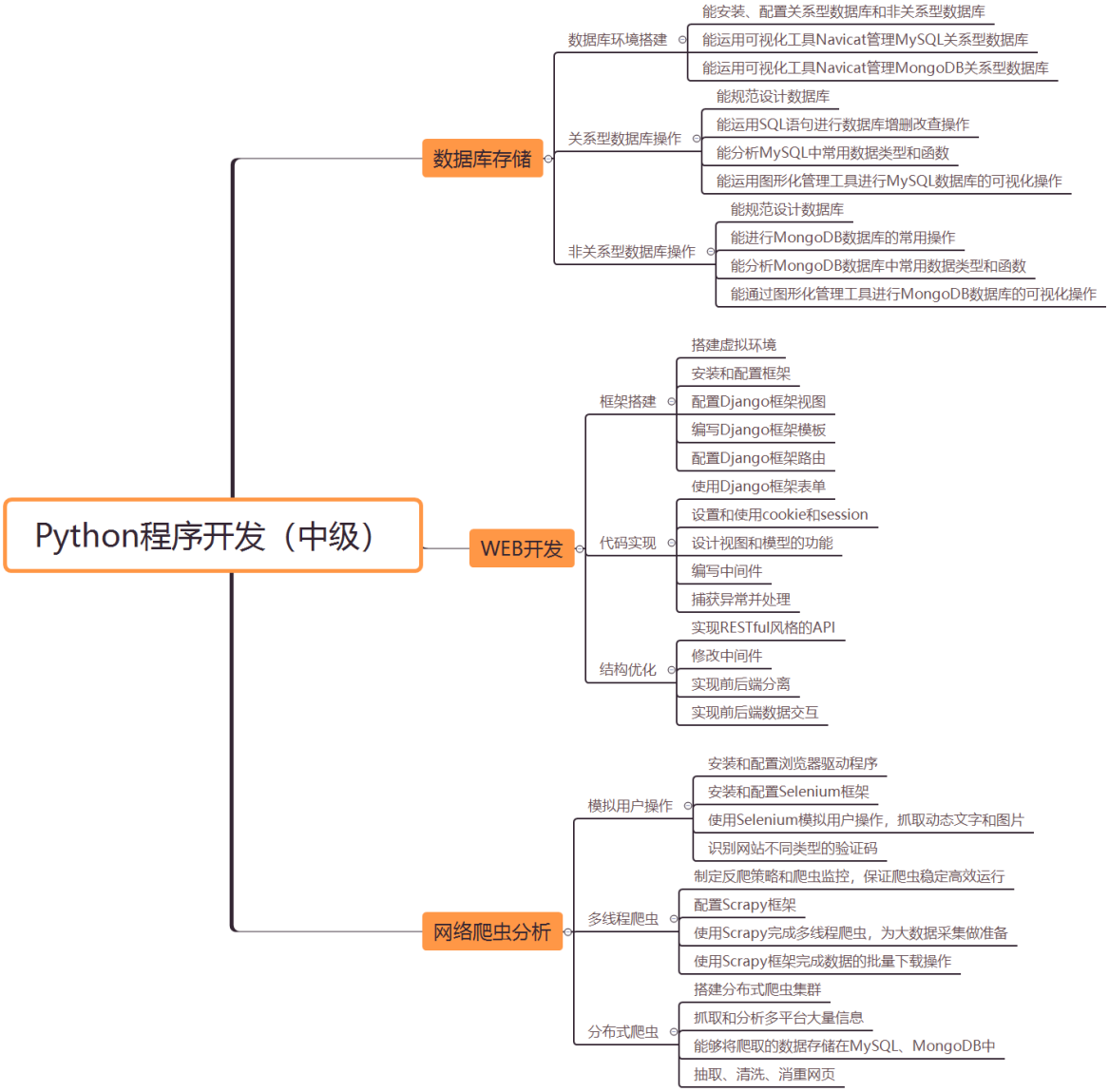
浏览器要求：谷歌稳定版75.0以上版本

1.5 考试内容、题型及分值

（1）理论：（单选题30题（60分））、（多选题10题（20分））、（判断题10题（20分））

（2）实操：程序填空（5个左右案例题）

1.6 中级证书涵盖的主要知识点



二、MySQL数据库

2.1 数据库基本概念

- **1.什么是数据库**

数据库（DataBase，DB），用于存储和管理数据的仓库。是指存储计算机内，按一定规则进行组织，可供共享的数据集合。数据库是数据的仓库,与普通文件不同，数据库依靠一定的数据结构来组织存储数据，以实现高效的数据存储与查询。

- **2.关系型数据库**

381 systems in ranking, November 2021

Rank			DBMS	Database Model	Score		
Nov 2021	Oct 2021	Nov 2020			Nov 2021	Oct 2021	Nov 2020
1.	1.	1.	Oracle	Relational, Multi-model	1272.73	+2.38	-72.27
2.	2.	2.	MySQL	Relational, Multi-model	1211.52	-8.25	-30.12
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	954.29	-16.32	-83.35
4.	4.	4.	PostgreSQL	Relational, Multi-model	597.27	+10.30	+42.22
5.	5.	5.	MongoDB	Document, Multi-model	487.35	-6.21	+33.52
6.	6.	↑ 7.	Redis	Key-value, Multi-model	171.50	+0.15	+16.08
7.	7.	↓ 6.	IBM Db2	Relational, Multi-model	167.52	+1.56	+5.90
8.	8.	8.	Elasticsearch	Search engine, Multi-model	159.09	+0.84	+7.54
9.	9.	9.	SQLite	Relational	129.80	+0.43	+6.48
10.	10.	10.	Cassandra	Wide column	120.88	+1.61	+2.13

关系型数据库，是指使用二维表格关系模型组织数据的数据库，通过行和列的形式表示和存储数据，数据库由行与列组成的表格构成。

主流的关系型数据库产品有Oracle、DB2、MySQL、Microsoft SQL Server等，每种数据库的的语法、功能和特性也各具特色。但一般都支持结构化查询语言（Structured Query Language，SQL）对数据库系统的维护管理和数据的插入、修改、查询、删除操作。

Oracle数据库是由甲骨文公司开发的，Oracle数据库在集群技术、高可用性、安全性、系统管理等方面都有较好性能表现，支持跨平台运行，是目前大型高性能商务数据库的首选。

MySQL是一款开源关系型数据库管理系统，具备体积小、速度快、部署成本低等优点，备受中小型企业欢迎。

2.2 MySQL数据库的安装以及管理

- **1.MySQL和可视化工具Navicat的安装**

略，见安装视频

- **2.MySQL数据库服务的启动与停止**

1）手动方式

右键我的电脑-->管理-->服务与应用程序-->服务-->双击最大化--> m找到以M开头的服务-->找到MySQL -->右键然后选择启动或停用

2）命令行方式

输入cmd，右键，以管理员身份启动，输入net start mysql启动MySQL服务，输入net stop mysql停止MySQL服务

- **3.MySQL数据库登录与退出**

1）**登录**

mysql -uroot -p密码

mysql -hip -uroot -p连接目标的密码

mysql --host=ip --user=root --password=连接目标的密码

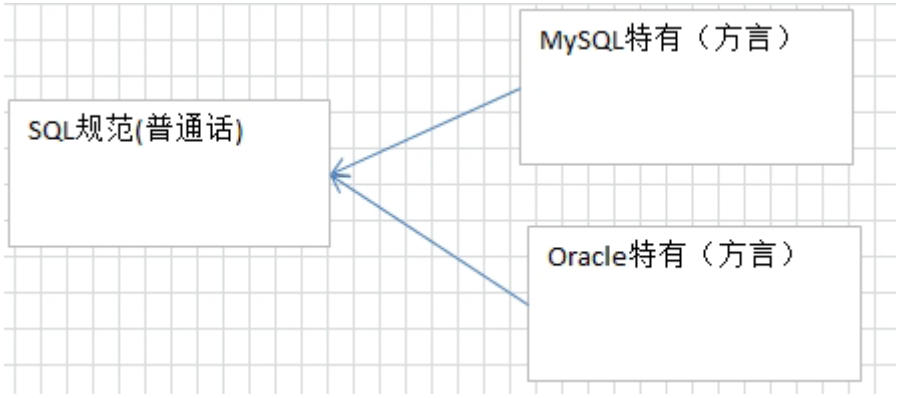
2）退出

exit 或者 quit

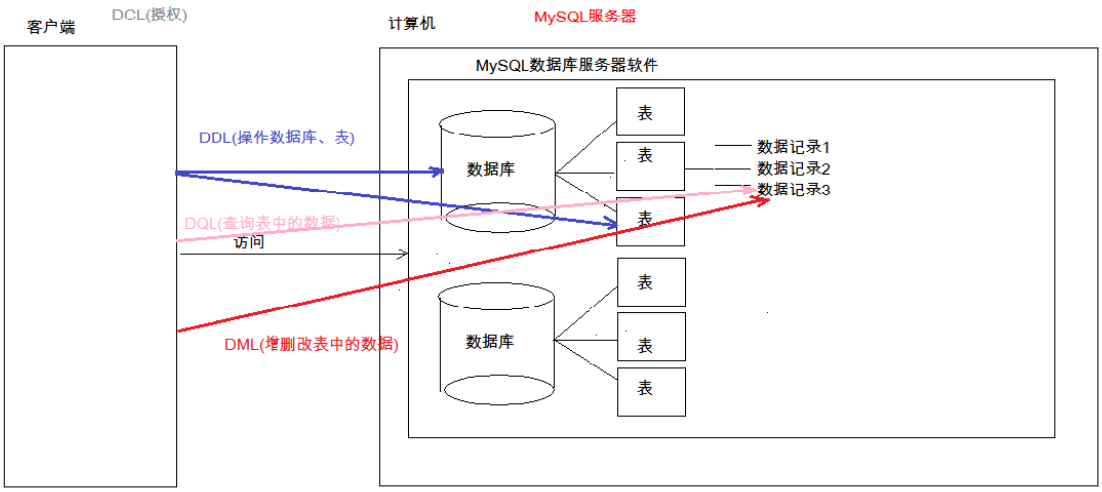
2.3 SQL的概念

- **1.什么是SQL**

Structured Query Language：结构化查询语言，其实就是定义了操作所有关系型数据库的规则。每一种数据库操作的方式存在不一样的地方，称为“方言”。



- **2.SQL分类**



1. DDL(Data Definition Language)数据定义语言
   用来定义数据库对象：数据库，表，列等。关键字：create, drop,alter 等
2. DML(Data Manipulation Language)数据操作语言
   用来对数据库中表的数据进行增删改。关键字：insert, delete, update 等
3. DQL(Data Query Language)数据查询语言
   用来查询数据库中表的记录(数据)。关键字：select, where 等
4. DCL(Data Control Language)数据控制语言(了解)；用来定义数据库的访问权限和安全级别，及创建用户。关键字：GRANT， REVOKE 等

2.4 DDL操作

2.4.1 DDL操作数据库

- **1.创建数据库**

```
CREATE DATABASE studb;
```

注意：studb为数据库名，MySQL命令不区分大小写

- **2.查看数据库**

```
SHOW DATABASES;
```

- **3.选择数据库**

```
USE dbname;
```

- **4.删除数据库**

```
DROP DATABASE DbName;
```

2.4.2 DDL操作数据表

- **1.显示表**

```
SHOW TABLES;
```

- **2.创建表**

```
create table 表名(
    列名1 数据类型1,
    列名2 数据类型2,
    ....
    列名n 数据类型n
    );
```

注意：最后一列，不需要加逗号（,）

- **3.MySQL数据类型和约束**

1）数值数据类型

重点关注**FLOAT、INT、DOUBLE**

类型	大小（Byte）	范围（有符号）	范围（无符号）	用途
TINYINT	1	(-128，127)	(0，255)	小整数值
SMALLINT	2	(-32 768，32 767)	(0，65 535)	大整数值
MEDIUMINT	3	(-8 388 608，8 388 607)	(0，16 777 215)	大整数值
INT 或 INTEGER	4	(-2 147 483 648，2 147 483 647)	(0，4 294 967 295)	大整数值
BIGINT	8	(-9 223 372 036 854 775 808，9 223 372 036 854 775 807)	(0，18 446 744 073 709 551 615)	极大整数值
FLOAT	4	(-3.402 823 466 E+38，-1.175 494 351 E-38)，0，(1.175 494 351 E-38，3.402 823 466 351 E+38)	0，(1.175 494 351 E-38，3.402 823 466 E+38)	单精度浮点数值
DOUBLE	8	(-1.797 693 134 862 315 7 E+308，-2.225 073 858 507 201 4 E-308)，0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308)	0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308)	双精度浮点数值
DECIMAL	DECIMAL(M,D)，依赖于M和D的值，若M>D，大小为M+2，否则为D+2			小数值

2）日期/时间数据类型

MySQL提供**DATE、TIME、YEAR、DATETIME、TIMESTAMP** 5种基本的日期和时间数据类型。

类型	大小（Byte）	范围	格式	用途
DATE	3	1000-01-01/9999-12-31	YYYY-MM-DD	日期值
TIME	3	'-838:59:59'/'838:59:59'	HH:MM:SS	时间值或持续时间
YEAR	1	1901/2155	YYYY	年份值
DATETIME	8	1000-01-01 00:00:00/9999-12-31 23:59:59	YYYY-MM-DD HH:MM:SS	混合日期和时间值
TIMESTAMP	4	1970-01-01 00:00:00/2038 结束时间是第 2147483647 秒， 北京时间 2038-1-19 11:14:07， 格林尼治时间 2038-1-19 03:14:07	YYYYMMDD HHMMSS	混合日期和时间值，时间戳

3）字符串数据类型

MySQL提供定长和变长字符串存储方式，分别用**CHAR(n)**和**VARCHAR(n)**表示

类型	大小（字节）	用途
CHAR	0-255	定长字符串
VARCHAR	0-65535	变长字符串
TINYBLOB	0-255	不超过 255 个字符的二进制字符串
TINYTEXT	0-255	短文本字符串
BLOB	0-65 535	二进制形式的长文本数据
TEXT	0-65 535	长文本数据
MEDIUMBLOB	0-16 777 215	二进制形式的中等长度文本数据
MEDIUMTEXT	0-16 777 215	中等长度文本数据
LONGBLOB	0-4 294 967 295	二进制形式的极大文本数据

4）约束

约束（Constraint）是数据库设计过程中为保证数据完整性、一致性、有效性的限制规则。MySQL常见的约束包含以下几个。

非空约束（Not Null），不允许插入空值。

唯一性约束（Unique），允许为空，但不能重复。

主键约束（Primary Key,PK），字段添加主键约束之后，该字段不能重复也不能为空，自动添加索引(index)。

外键约束（Foreign Key,FK），外键约束主要用来维护两个表之间数据的一致性。

缺省约束（Default），缺省约束用来指定某列的默认值。

5）例：**创建学生信息表**

```
CREATE TABLE stuinfo(
    id int not null auto_increment, -- 自增
    name varchar(255) not null,
    sex tinyint null,
    birth date null,
    address varchar(255),
    PRIMARY KEY(id) -- 主键
);
```

- **4.查看表结构**

```
SHOW COLUMNS FROM TablesName;
```

- **5.修改表**

```
ALTER TABLE TableName AlterCommand;
```

其中，TableName为要修改的表名，AlterCommand为相应的修改命令，如ADD、DROP、MODIFY/CHANGE子句，分别表示增加、删除、修改

- **6.删除表**

```
DROP TABLES;
```

2.5 DML数据增删改

2.5.1 插入数据

```
insert into 表名(列名1,列名2,...列名n) values(值1,值2,...值n);
```

注意:

1） 列名和值要一 一对应。

2）如果表名后，不定义列名，则默认给所有列添加值

3）除了数字类型，其他类型需要使用引号(单双都可以)引起来

2.5.2 修改数据

```
UPDATE TableName SET field1=newvalue1, field2=newvalue2... WHERE 条件;
```

WHERE子句指定符合条件的数据，如果没有指定WHERE条件子句，则修改表中所有记录

2.5.3 删除数据

```
DELETE FROM TableName WHERE 条件;
```

如果没有指定 WHERE 条件子句，MySQL 表中的所有记录将被删除，需要慎重使用删除语句

2.6 DQL查询数据

```
select
        字段列表
from
        表名列表
where
        条件列表
group by
        分组字段
having
        分组之后的条件
order by
        排序
limit
        分页限定
```

2.7 Python与MySQL交互

2.7.1 库的介绍和安装

- **1.Python操作MySQL库**

MySQLdb是流行的Python操作MySQL第三方库，目前只支持Python 2版本。

mysqlclient是Python 3版本下MySQLdb的分支，解决了MySQLdb与Python 3版本不兼容问题，仍使用MySQLdb库名。mysqlclient由C语言编写，速度快。

pymysql由Python编写实现，对Python环境支持较好，但是速度比mysqlclient慢。

- **2.库的安装**

可使用pip install mysqlclient命令安装mysqlclient

可使用pip install pymysql命令安装pymysql

2.7.2 pymysql的使用

```
import pymysql
# 创建数据库连接对象
db = pymysql.connect(host='主机名',
    user='用户',
    password='密码',
    db='数据库名',
    charset='编码方式')
# 创建游标对象
cur = db.cursor()
# 执行SQL语句
cur.execute(sql)
# 获取查询结果集的一条数据(当为查询SQL语句时)
cur.fetchone()
# 获取查询结果集的多条数据(当为查询SQL语句时)
cur.fetchmany(n)
# 获取查询结果集的所有数据(当为查询SQL语句时)
cur.fetchall()
# 提交到数据库执行
db.commit()
# 关闭游标对象
```

```
cur.close()
# 关闭数据库连接对象
db.close()
```

2.7.3 项目实战：Python操作MySQL数据库

需求描述

在实际工作中，经常有通过Python程序访问数据库并对数据库进行增删改查操作的需求，实现这一系列的功能，需要借助一个第三方库pymysql来实现。本项目的任务是通过pymysql提供的相关函数，实现连接mysql数据库，并对表中的数据进行增删改查。

代码实现

```
import pymysql

# 数据库连接
Db = pymysql.connect(host='localhost',
                     user='root',
                     password='123456',
                     db='studb',
                     charset='utf8',
                     cursorclass=pymysql.cursors.DictCursor)
# 指定游标类型为字典格式返回数据。
cur = Db.cursor()
# 查询MySQL版本
SQL = "SELECT VERSION()"
res = cur.execute(SQL)
ver = cur.fetchone()
print("读取版本号:%s" % ver)

# 插入一条数据（要求：姓名为侯美汐，性别为女（1表示男，0表示女），出生日期为2001-5-3，住址为广东中山中路，班级为2班。
SQL = "insert into stuinfo (name,sex,birthday,address,class) values('侯美汐',0,'2001-5-3','广东中山中路',2)"
res = cur.execute(SQL)
Db.commit()
print("插入数据执行成功，插入了%d条数据" % res)

# 删除姓名中含“小”的数据。
SQL = "delete from stuinfo where name like '%小%'"
res = cur.execute(SQL)
Db.commit()
print("删除数据执行成功,删除了%d条数据" % res)

# 修改数据（要求：将侯美汐的姓名改为侯妤汐，住址改为四川成都天府大道。
SQL = "update stuinfo set name='侯妤汐',address='四川成都天府大道' where name='侯美汐'"
res = cur.execute(SQL)
Db.commit()
print("修改数据执行成功,修改了%d条数据" % res)

# 查询所有学生的姓名、出生日期和住址信息。
SQL = "select name,birthday,address from stuinfo"
res = cur.execute(SQL)
# 获取所有查询结果
data = cur.fetchall()
```

```
print("查询数据结果为:%s" % data)
# 关闭游标对象
cur.close()
Db.close()
```

三、MongoDB数据库

3.1 非关系型数据库

实际应用中存在很多非结构化的数据，例如文本、图像、可扩展的树形结构数据等，这些不规则或不完整，没有预定义的数据模型，难以使用关系型数据库的二维表格模型进行表示和存储。

为解决传统数据库处理非结构化数据的局限，大量高并发非结构化数据应用难题，非关系型数据库幸运而生，通常用NoSQL泛指非关系型数据库，即“non-relational”，也经常被解释为“Not Only SQL”。NoSQL数据库的最大特点是去掉关系数据库的关系型特性，数据非常容易扩展，带来了NoSQL数据库**可扩展性、大数据量、高性能**的特点。

3.2 MongoDB数据库简介

MongoDB是一种基于分布式文件的开源数据库系统，MongoDB由C++编写，可添加节点保证服务器性能，提供可扩展的高性能数据存储解决方法。

MongoDB提供面向文档的数据存储方式，数据结构由键值对组成，可使用JSON方式进行数据存储，支持灵活快捷的Web开发。

概念	说明
database	数据库，相当于 SQL 数据库中的数据库
collection	集合，相当于 SQL 数据库中的表
document	文档，相当于 SQL 数据库中的行，一个文档由多个字段 b 组成，并采用类似 JSON 格式的 BJSON 表示
field	字段，相当于 SQL 中的列，比 SQL 数据库更加灵活，支持嵌套的文档、数组等

3.3 MongoDB安装及启动

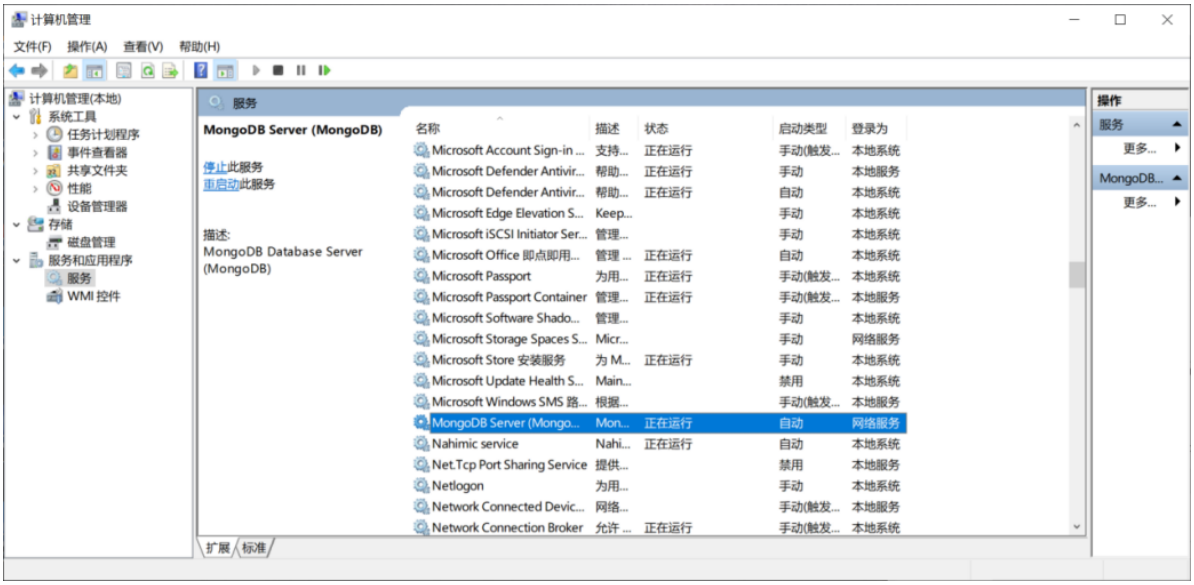
- **1.MongoDB的安装**

略，见安装视频

- **2.MongoDB的启动**

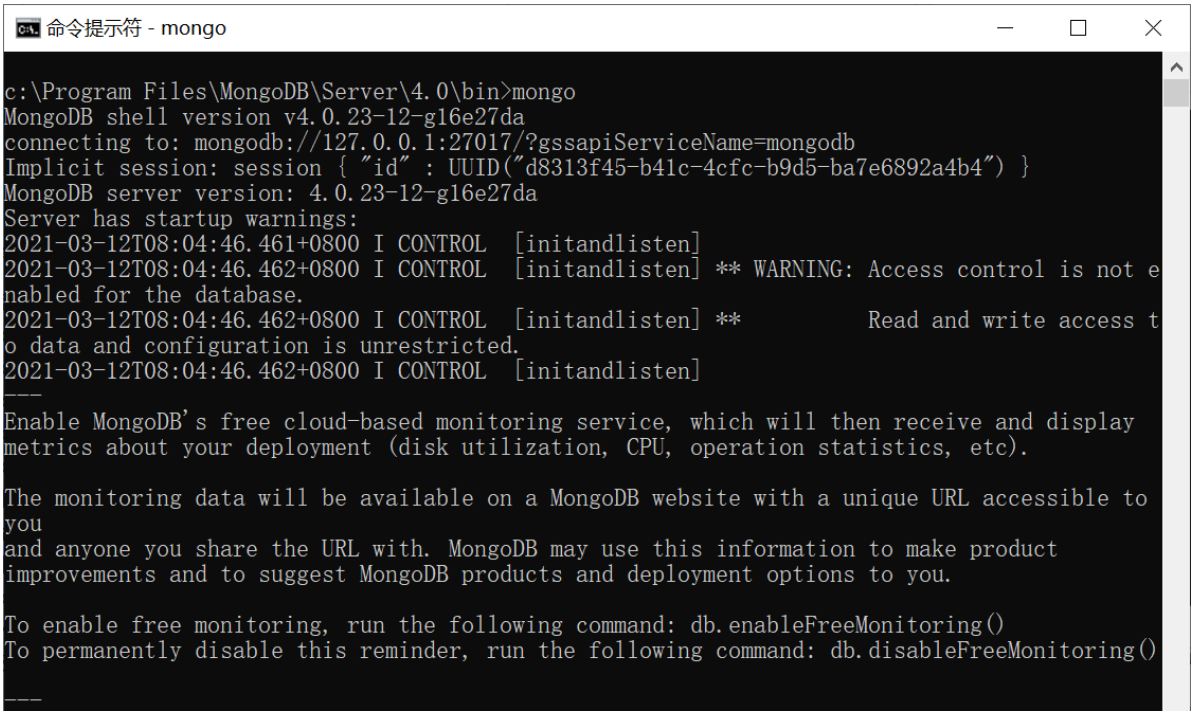
1）启动服务器

如果安装过程选择作为Windows服务安装，则Windows已经将MongoDB作为系统服务启动，可以通过鼠标右键启动或停止MongoDB服务。



2）启动客户端

mongo.exe是MongoDB的客户端程序，另外开启一个CMD命令行窗口，执行mongo.exe即可启动MongoDB客户端。



3.4 MongoDB数据库操作

- **1.显示数据库**

```
show dbs或show databases
```

- **2.切换数据库**

```
use 数据库名
```

MongoDB提供非常灵活的数据库操作，用户可以切换至一个尚未创建的数据库，当执行数据库写入操作后自动建立该数据库（也会自动创建一个和数据库同名的集合）。

```
use newdatabase
db.newdatabase.insert({"num":"123","name":"中慧","age":"40"})
show dbs
```

- **3.查看当前数据库信息**

```
db.stats()
```

- **4.删除数据库**

```
db.dropDatabase()
```

3.5 MongoDB集合操作

- **1.显示集合**

```
show collections 或 show tables
```

- **2.创建集合**

```
db.createCollection(集合名,可选参数)
```

参数说明：

capped，是否创建固定集合。如参数值为 true，则创建固定集合。固定集合指的是事先创建，并且大小固定的集合。即假设一个集合设置了固定大小为100，再添加一条文档的时候，会把最前面的文档剔除，永远只保留100条数据。一般来说，固定集合适用于任何想要自动淘汰过期属性的场景。比如日志文件，聊天记录，通话信息记录等只需保留最近某段时间内的应用场景，都会使用到MongoDB的固定集合。

size，最大字节数；

max，最大文档数。

MongoDB**支持自动创建集合**，向尚不存在集合写入记录后，在数据库中自动创建该集合。

- **3.删除集合**

```
db.CollectionName.drop()
```

其中CollectionName为集合名，如删除成功，返回true；否则返回false。

3.6 MongoDB文档操作

MongoDB支持文档使用键-值对存储数据，支持字符串、数值、日期、对象等多种数据类型，写入数据时，MongoDB会根据给定的键值自动设置该键对应的数据类型，可以使用typeof判断数据类型，如命令“typeof 25”执行结果为“number”。

- **1.插入文档**

```
db.CollectionName.insert(文档描述)
```

例

```
db.mycol.insert(
{ "num" : "2020001",
 "age" : 18,
 "address" : ["江苏常州"],
 "score" : { "ch" : 120, "en" : 85.5, "math" : 127,"grade":"A" }
})
db.mycol.find()
```

- **2.查询文档**

```
db.CollectionName.find({key:value})
```

其中，CollectionName为集合名，查询条件{key:value}为键-值对组合，使用多个键-值对组合查询时，中间使用逗号分开。如果省略查询条件，则find()操作返回当前集合下的所有记录。

- **3.删除文档**

```
db.CollectionName.remove({key:value}, {<justOne:false>})
```

其中，第一个参数{}key:value}表示查询条件，可选参数justOne表示是否只删除一条记录，默认值为false。

- **4.修改文档**

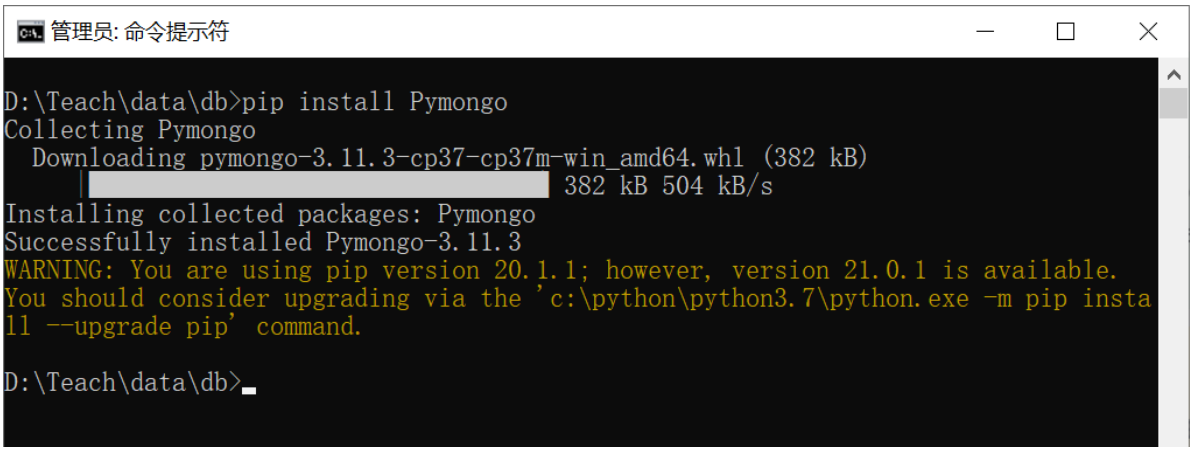
```
db.CollectionName.update(
{key:value},
{$set:{key1:value1,key2:value2,…}},
{< upsert :false>,< multi:false >}
)
```

参数{key:value}表示查询条件，参数{$set:{key1:value1,key2:value2,...}}表示执行修改操作符和待修改文档内容。除了$set命令，还有incr（自增）、$rename（修改字段名）等操作符。**可选参数upsert表示如果查询不存在是否插入指定修改内容为新的记录，默认为false**，可选参数multi表示是否修改多条记录，默认为false。

3.7 MongoDB与Python交互

3.7.1 库的安装

Python通过pymongo库实现MongoDB数据库的交互操作，可使用pip install pymongo命令安装pymongo库。



3.7.2 pymongo的使用

```
# 连接数据库得到MongoClient对象
# mon_client = pymongo.MongoClient(host='localhost', port=27017)
mon_client = pymongo.MongoClient("mongodb://localhost:27017/")
# 数据库对象,studb为数据库名
mon_db = self.mon_client["studb"]
# 数据库对象的数据集子对象，stuinfo为集合名
mon_col = self.mon_db["stuinfo"]
# 插入一条记录
mon_col.insert_one(col_document)
# 插入多条记录
mon_col.insert_many([col_document1, col_document2,…,col_documentn])
# 查找满足条件的所有记录
mon_col.find({Key:Value})
# 查找满足条件的一条记录
mon_col.find_one({Key:Value})
# 删除满足条件的一条记录
mon_col.delete_one({Key:Value})
# 删除满足条件的所有记录
mon_col.delete_ many({Key:Value})
# 修改满足条件的一条记录
mon_col.update_one(queryKey,updateKey)
# 修改满足条件的全部记录
mon_col.update_many(queryKey,updateKey)
```

四、Redis数据库

4.1 Redis简介

远程字典服务器(Remote Dictionary Server,Redis)，是开源协议的NoSQL技术，提供基于单线程的内存数据结构存储，支持Java、Python、Lua等多种语言API调用，可以作为数据库、缓存服务器使用。Redis具备多种优势，在高速Web开发等领域有广泛的应用前景。

Redis使用NoSQL的键-值对存储数据，支持字符串、哈希、列表、集合等数据结构。

Redis数据运行在内存中，支持多种数据结构的高速读/写操作，性能优越，远超数据库中开源数据结构存储技术。

Redis支持内存数据的磁盘持久化存储，并支持主从模式的数据备份。

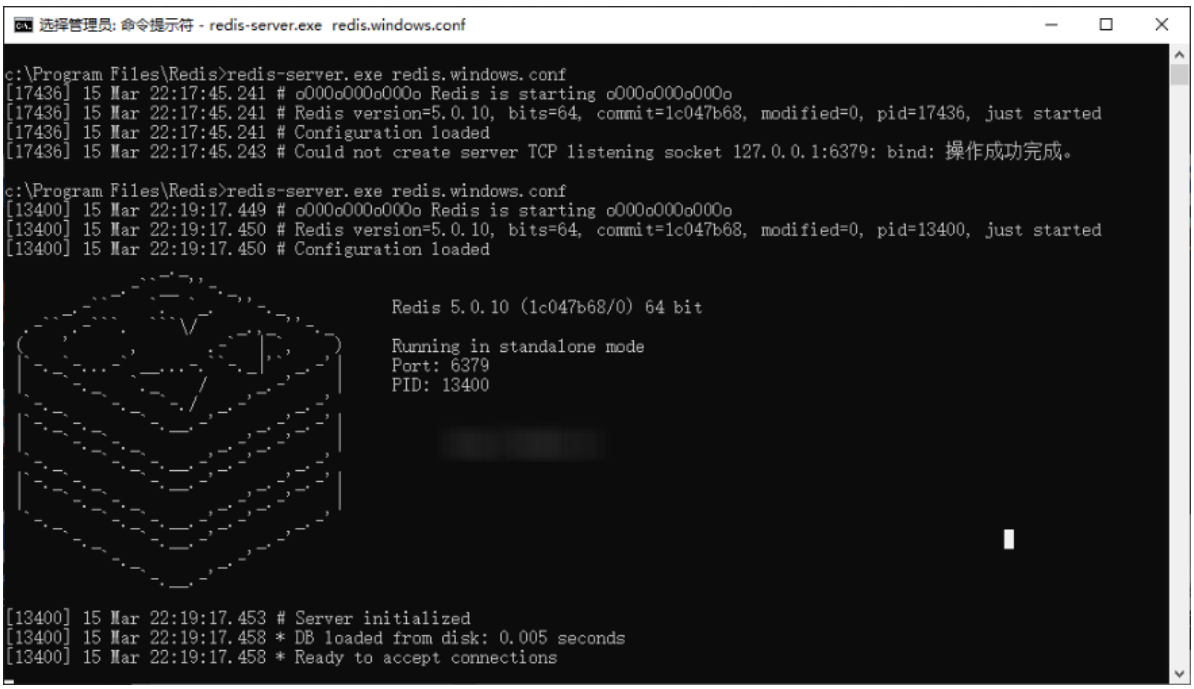
4.2 Redis的安装与管理

- **1.Redis安装**

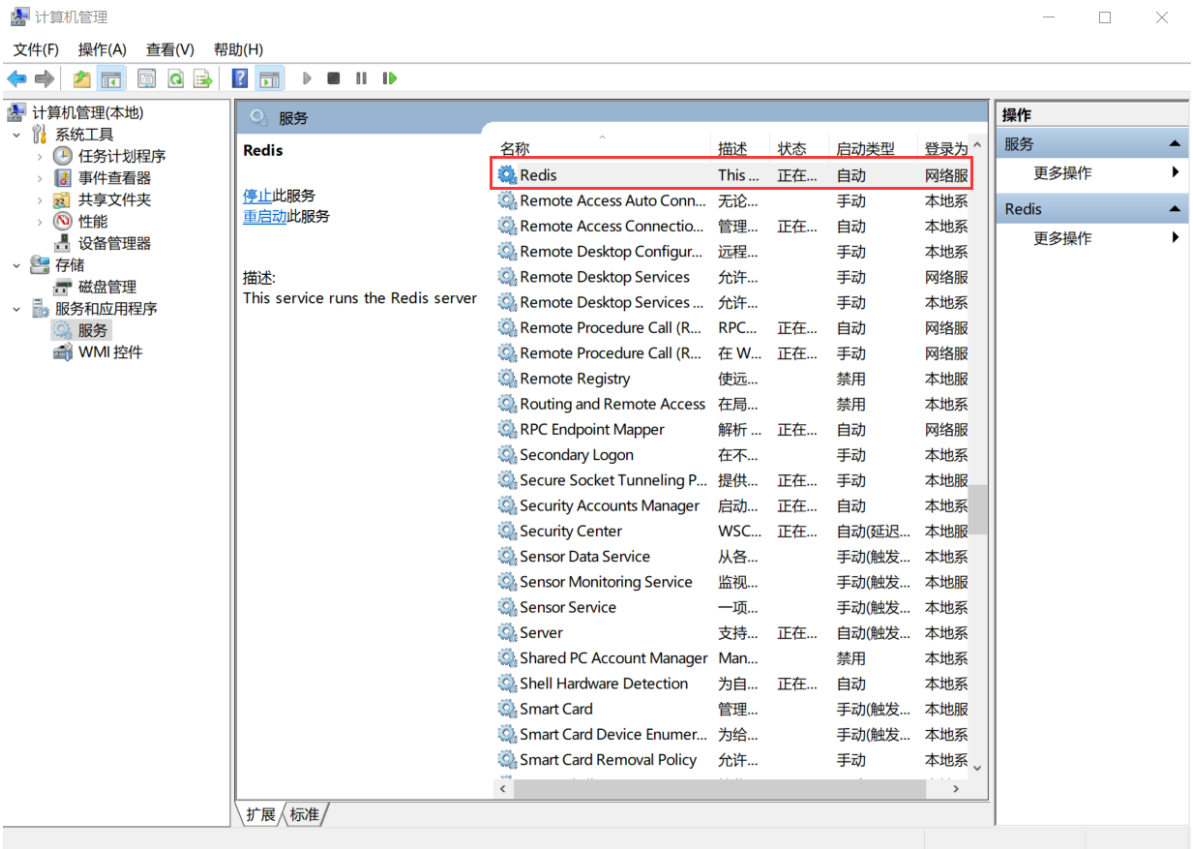
略，见安装视频

- **2.Redis的管理**

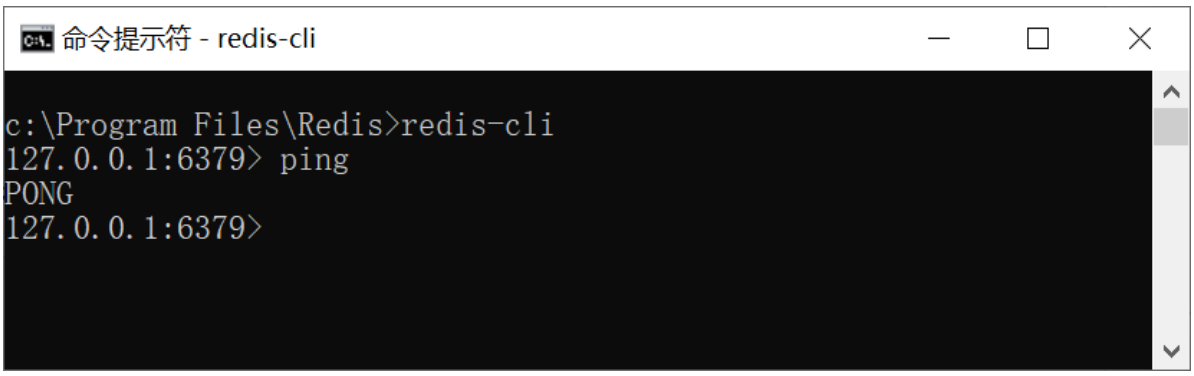
Redis安装完成后，使用CMD终端cd命令进入安装目录,Redis服务端程序为redis-server.exe，如果Redis服务未启动，可在CMD终端执行命令“redis-server.exe redis.windows.conf”启动服务，参数“redis.windows.conf”指定Redis的服务器配置。出现下图所示界面时，命令输出显示，表示Redis服务启动成功。



如果安装时选择将Redis作为Windows服务安装，则不需要每次手动启动服务端程序。



Redis客户端程序为redis-cli.exe，在CMD终端执行redis-cli.exe命令，出现下图所示界面的服务器地址和端口时，表示Redis客户端启动成功。可发送PING命令测试服务器，正常服务器回复PONG。



4.3 Redis基本命令

- **1.PING**

PING命令用于测试服务器是否连接成功，如果没有参数，服务器返回“PONG”，如果带参数，则返回参数

- **2.KEYS**

Keys 命令用于查找所有符合给定模式 pattern 的 key

- **3.SELECT**

SELECT命令用于选择当前数据库，Redis默认有16个数据库，SELECT使用参数 0 ~ 15切换当前工作数据库。

- **4.DBSIZE**

DBSIZE 命令用于查看当前数据库的记录数

- **5.FLUSHDB与FLUSHALL**

FLUSHDB命令清除当前数据库记录，FLUSHALL命令清除所有数据库记录

- **6.TYPE**

TYPE命令查看Key的存储类型

注意：Redis不区分命令大小写

4.4 Redis数据结构

4.4.1 String

- **1.SET**

基本功能：设置一个String对象的值，如果该对象已经存在，则进行修改，否则新增一个String对象。

```
SET KeyName Value
```

- **2.GET**

基本功能：获取一个String对象的值，如果该对象不存在，返回nil。

```
GET KeyName
```

- **3.APPEND**

基本功能：字符串追加。

```
APPEND KeyName Value
```

- **4.SETRANGE**

基本功能：修改字符串子串。

```
SETRANGE KeyName OffSet Value
```

从偏移位OffSet开始，使用Value修改字符串KeyName的值。

4.4.2 List

List是Redis基于链表实现的一种基本数据结构，用于存储有序元素集合。List支持从左、右两端的入栈、出栈、索引等操作，使用非常灵活，可以作为队列、栈等数据结构使用。

- **1.LPUSH和RPUSH**

基本功能：从左端或右端添加新元素

```
LPUSH ListName Value1 Value2…
RPUSH ListName Value1 Value2…
```

其中，ListName为列表名，Value1、Value2...为指定的元素值，至少指定一个添加的元素值

- **2.LRANGE**

基本功能：读取指定范围的元素。

```
LRANGE ListName Start End
```

其中，ListName为列表名，Start、End分别为开始与结束范围，End值指定为-1时，表示取值范围到列表尾部。

- **3.LPOP和RPOP**

基本功能：从左端或右端删除一个元素，并返回该元素值，如果列表为空，则返回nil

```
LPOP ListName
RPOP ListName
```

其中ListName为列表名。

- **4.LLEN**

基本功能：获取列表长度。

```
LEN ListName
```

- **5.LINDEX**

基本功能：获取列表指定元素值。

```
LINDEX ListName Index
```

其中，ListName为列表名，Index为位置。Index为正值表示从表头开始向表尾搜索，为负值表示从表尾向表头搜索

4.4.3 Hash

Hash是Redis的一种字典存储数据结构，一个Hash对象可以存储多个键-值对元素，底层由哈希表实现。Hash包括HSET、HGET和HMST等基本操作，分别实现字典的添加、获取、删除等操作。

- **1.HSET和HMSET**

基本功能：设置字典的一个或多个键-值对元素。

```
HSET DictName KeyName Value
HMSET DictName KeyName1 Value1 KeyName2 Value2…
```

其中，DictName为字典名。KeyName和Value是其中的一个键-值对，HMSET命令可以同时设置多个键-值对。

- **2.HGET和HGETALL**

基本功能：获取字典的一个或所有键-值对。

```
HGET DictName KeyName
HGETALL DictName
```

其中，DictName为字典名。HGET命令须指定键名，如果字典中存在该键，则返回元素对应的值，否则返回nil。HGETALL命令返回该字典的所有键-值。

4.4.4 Set

Set是Redis用于存储无序非重复元素的一种数据结构。集合最多可存储2**32-1个元素成员，由于集合元素是不重复的，通过哈希实现的Set的读写和查找操作非常快，时间复杂度仅为（O/1）。Set包括SADD、SCARD、SMEMBER、SDIFF、SINTER和SUNION等基本操作。

- **1.SADD**

基本功能：将元素值添加到集合中。

```
SADD SetName Value1 Value2 …
```

其中，SetName为集合名，Value1 Value2 ...要添加的元素值，如果指定元素值不符合Set规则，则自动忽略。

- **2.SCARD**

基本功能：获取集合的元素个数。

```
SCARD SetName
```

其中，SetName为集合名，返回值为集合的元素个数。

- **3.SMEMBERS**

基本功能：获取集合元素。

```
SMEMBERS SetName
```

其中，SetName为集合名，返回值为集合的元素值。

4.4.5 Sorted Set

Sorted Set是Redis用于存储有序集合的一种数据结构。有序集合最多可存储2**32-1个元素成员，集合元素不允许重复，集合中每个元素值对应一个分数值，元素按分数有序存储。ZADD、ZRANGE、ZREM等命令实现有序集合的元素添加、元素显示 、元素移除等基本操作。

- **1.ZADD**

基本功能：将元素值添加到有序集合中。

```
ZADD ZSetName Score1 Value1 Score2 Value2 …
```

其中，ZSetName为集合名，Score1Value1为对应的元素分数与元素值，可以一次将多组元素分数和元素值存入有序集合中。返回值为添加成功的元素个数。如果指定元素值不符合有序集合规则，则自动忽略。

- **2.ZCARD**

基本功能：用于计算集合中元素的数量。

```
ZCARD KEY_NAME
```

- **3.ZRANGE**

基本功能：显示有序集合的元素。

```
ZRANGE ZSetName Start End WITHSCORES
```

其中，ZSetName为集合名，Start、End两个参数指定起始元素位置，参数WithScore指定显示元素对应分数，如果省略该参数，则只显示元素值。

- **4.ZREM**

基本功能：移除有序集合的一个或多个元素。

```
ZREM ZSetName Value1 Value2…
```

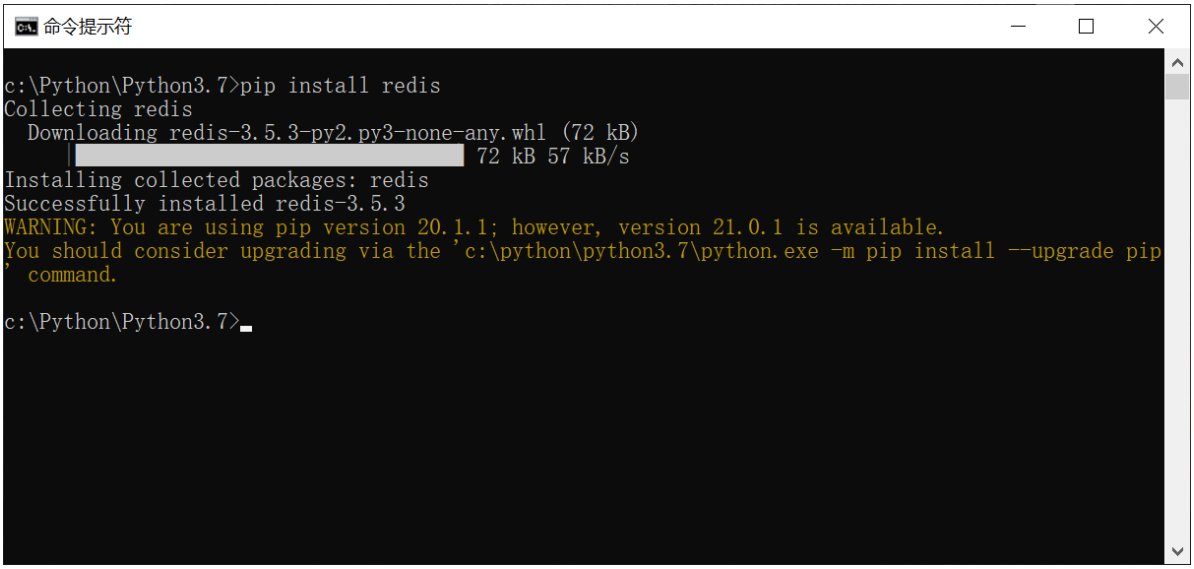
其中，ZSetName为集合名，Value1 Value2...为要移除的元素值，如果指定元素不存在，则自动忽略。

4.5 Redis与Python的交互

4.5.1 库的安装

Redis支持包括Python、Java、PHP在内的多种语言接口支持。Python环境下非常容易实现Redis连接、读写。

可使用pip install redis命令安装Python Redis模块。



4.5.2 redis库的使用

```
import redis

# 创建连接池对象,连接本机,数据库1，Redis返回值格式默认为字节形式可使用参数decode_responses指定为字符串
pool = redis.ConnectionPool(host='127.0.0.1', port=6379, db=1,
decode_responses=True)
# 通过连接池创建连接对象
cli = redis.Redis(connection_pool=pool)

# 写入键值对{"name": "张三"}
res = cli.set("name", "张三")
print(res)

# 获取键"name"对应的值
res = cli.get("name")
print(res)
```

五、Django框架

5.1 Django介绍和安装

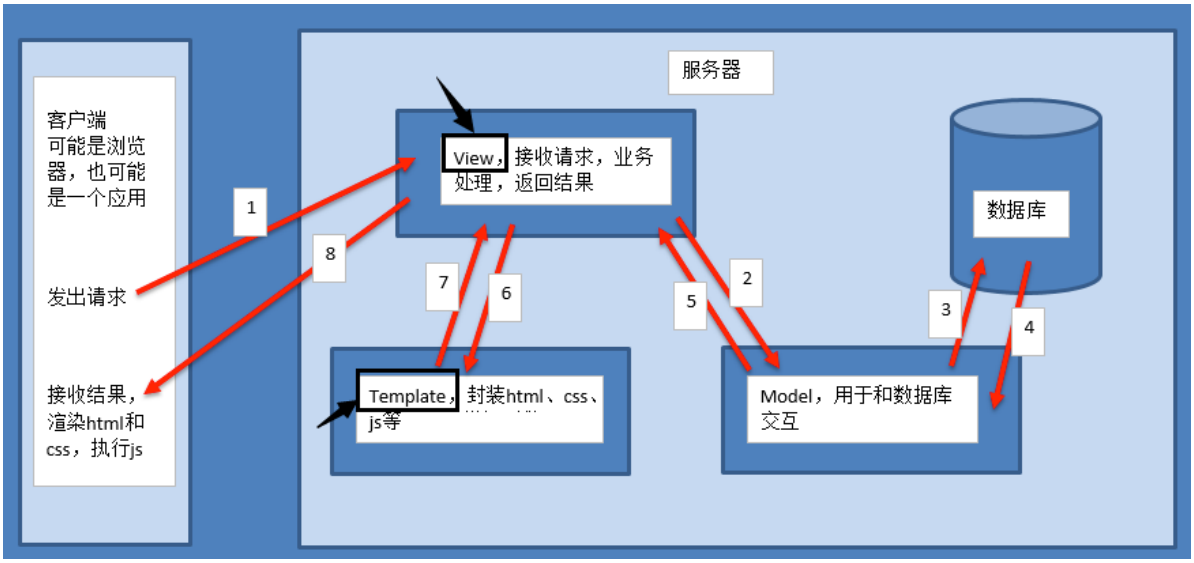
5.1.1 MTV框架模式

Django，发音为[`dʒæŋgəʊ]，是用python语言写的开源web开发框架，劳伦斯出版集团为了开发以新闻内容为主的网站，而开发出来了这个框架。它遵循MVC设计，并且有一个专有名词：MVT。

M全拼为Model，与MVC中的M功能相同，负责和数据库交互，进行数据处理。

T全拼为Template，与MVC中的V功能相同，负责封装构造要返回的html。

V全拼为View，与MVC中的C功能相同，接收请求，进行业务处理，返回应答



5.1.2 Django的安装

pip install django

5.2 Django项目的创建

5.2.1 创建项目

- **1.流程**

1.规划好一个文件夹

2.在文件夹中打开cmd命令行

3.执行命令django-admin startproject Login

4.现在可以用Pycharm来打开这个项目来进行开发了

- **2.项目目录结构**

1.manage.py是项目管理文件，是Django用于管理本项目的命令行工具，后续的站点运行、 数据库自动生成、 静态文件收集等工作都需要通过该文件完成。

2.一个与项目同名的目录,这里是Login

2.1.*init*.py是一个空文件，告诉Python该目录是一个Python包，创建后暂无内容。

2.2.settings.py是Django项目的配置文件，里面包含了项目引用的Django组件、项目名等。后续开发中，还需要进行数据库参数配置、导入其他包等操作。

2.3.urls.py是项目的URL配置文件。

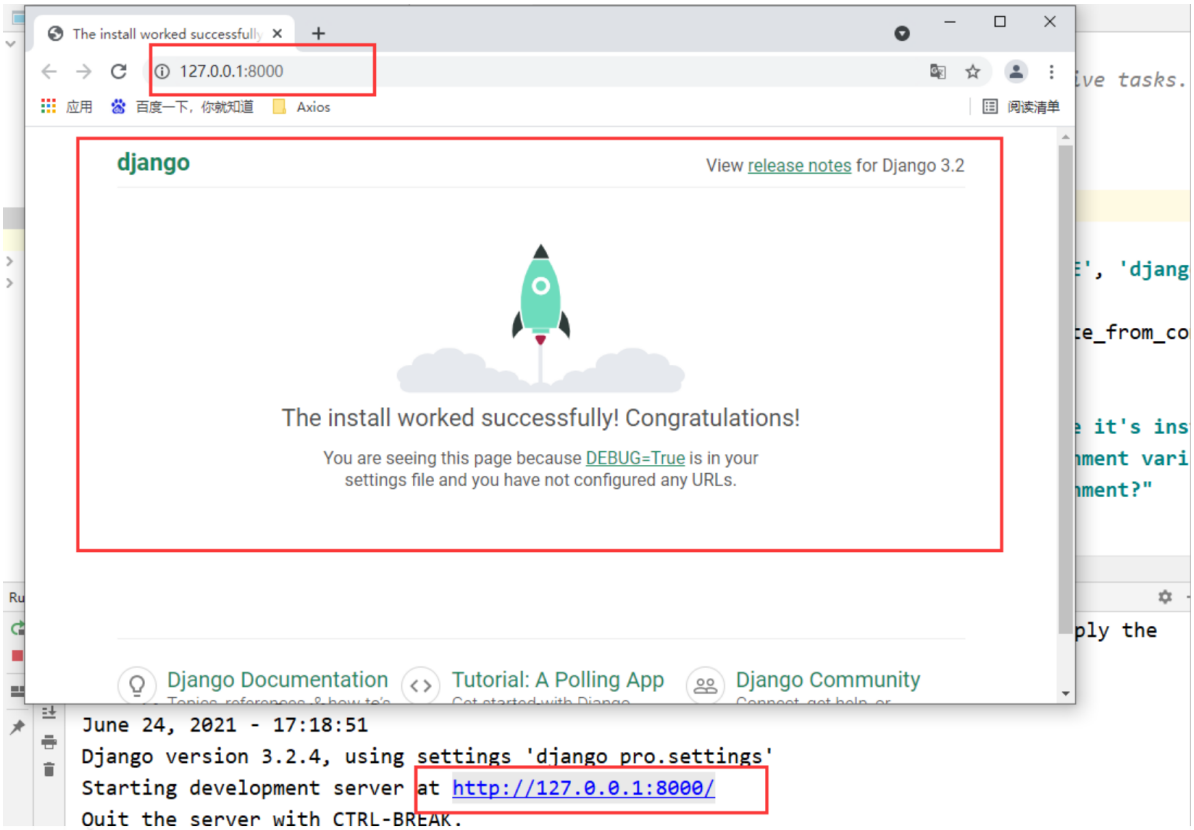
2.4.wsgi.py是定义WSGI的接口信息，用于与其他Web服务器集成，一般无需改动。

2.5 asgi.py与ASGI兼容的web服务器，为项目提供服务的入口点，以便运行项目

5.2.2 启动项目

进入到manage.py同级目录，然后执行命令**python manage.py runserver**启动项目

点击“http://127.0.0.1:8000/”网址处，将跳出如下界面



5.2.3 创建应用

在Django中,一个模块是由一个Django应用来开发的,比如电商网站有用户模块(负责用户注册、登录等)、商品模块(负责商品信息展示等),一个项目由很多个应用组成的，每一个应用完成一个特定的功能。

- **1.语法**

1）先进入到项目文件夹（manage.py同级目录）

2）python manage.py startapp 应用名

python manage.py startapp app1　创建app1应用

- **2.应用目录结构**

init.py是一个空文件，表示当前目录app1可以当作一个python包使用。

tests.py文件用于开发测试用例，在实际开发中会有专门的测试人员，这个事情不需要我们来做。

models.py文件跟数据库操作相关。

views.py文件跟接收浏览器请求，进行处理，返回页面相关。

admin.py文件跟网站的后台管理相关。

migrations文件夹存放数据库的迁移文件。

apps.py文件为应用的属性配置文件

- **3.应用的注册**

应用创建成功后，需要安装才可以使用，也就是建立应用和项目之间的关联。修改settings.py中的INSTALLED_APPS配置项,加上你的应用即可。

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'app1'
]
```

5.3 视图

对于django的设计框架MVT，用户在URL中请求的是视图，视图接收请求后进行处理，并将处理的结果返回给请求者

使用视图时需要进行两步操作：1.定义视图函数 2.配置URLconf

5.3.1 定义视图函数

视图就是一个Python函数，被定义在views.py中。

视图的必须有一个参数，一般叫request，视图必须返回HttpResponse对象，HttpResponse中的参数内容会显示在浏览器的页面上。

打开app1/views.py文件，定义视图index如下。

```
from django.http import HttpResponse
def index(request):
    return HttpResponse("首页")
```

5.3.2 配置URLconf

请求者在浏览器地址栏中输入url，请求到网站后，获取url信息，然后与编写好的URLconf逐条匹配，如果匹配成功则调用对应的视图函数，如果所有的URLconf都没有匹配成功，则返回404错误。

需要两步完成URLconf

- **1.应用中创建urls.py**

app1/urls.py中的代码如下

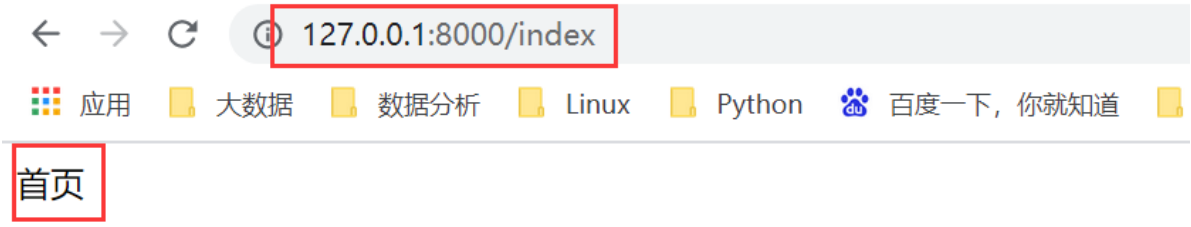
```
from django.urls import path
from app1 import views

urlpatterns = [
    path('index', views.index),
]
```

- **2.项目中进行路由分发**

Skin/urls.py中的代码如下

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('app1.urls'))
]
```

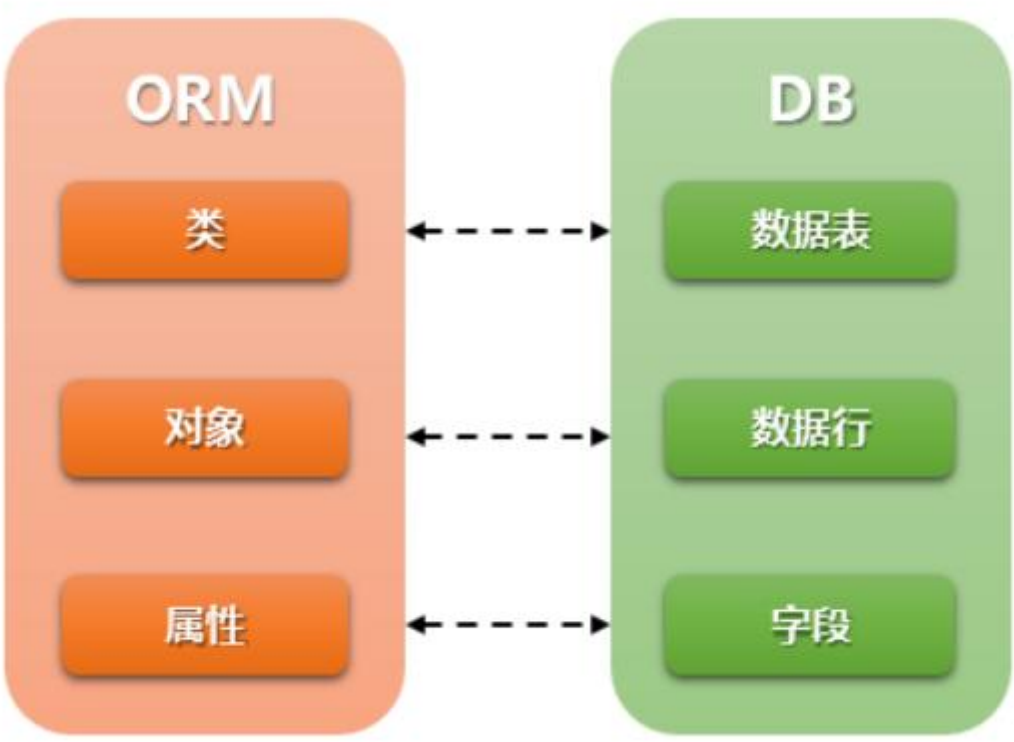
此时在浏览器输入http://127.0.0.1:8000/index，结果如下图



5.4 模型

5.4.1 ORM

O是object，也就类对象的意思，R是relation，翻译成中文是关系，也就是关系数据库中数据表的意思，M是mapping，是映射的意思。在ORM框架中，它帮我们把类和数据表进行了一个映射，可以让我们通过类和类对象就能操作它所对应的表格中的数据。



5.4.2 通过模型类生成数据表

- **1.数据库模块安装**

```
pip install  pymysql
```

- **2.创建数据库login**

创建语法:

```
create database login charset='utf8';
```

- **3.settings配置数据库**

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'login',
        'USER': 'root',
        'PASSWORD': '123456',
        'HOST': 'localhost',
        'PORT': 3306
    }
}
```

- **4.settings同级init.py配置**

```
import pymysql
pymysql.install_as_MySQLdb()
```

- **5.在app1应用下的models中创建模型类**

```
# 创建模型类
class User(models.Model):
    # 用户名
    uname = models.CharField(max_length=50)
    # 密码
    upwd = models.CharField(max_length=50)
    # 在数据库中生成的表名为应用名_类名小写,可通过Meta来规定数据库中表的名字
    class Meta:
        db_table = "user"
```

- **6.数据库迁移**

1）先生成本地数据库文件

```
python manage.py makemigrations
```

2）推送到MySQL数据库

```
python manage.py migrate
```

5.4.3 通过模型类操作数据表

- **1.添加数据**

进入项目shell的命令

python manage.py shell

在交互式shell终端中演示

向user表中插入一条数据。

```
from app1.models import User
u = User()
u.name = '张三'
u.password = '123456'
# 调用save()方法才会将数据存入数据库
u.save()
# 用此种方法也可实现添加数据
User.objects.create(name="李四", password="123")
```

- **2.查询数据**

```
# 查询id为1的用户
u1 = User.objects.get(id=1)
u1.name  # '张三'
```

- **3.修改数据**

```
u1.name = "bbb"
# 调用save()方法才能更新数据表中数据
u1.save()
```

- **4.删除数据**

```
 u1.delete()
```

- **5.查询所有内容**

```
users = User.objects.all()  # 得到一个查询集
users.values()  # <QuerySet [{'id': 2, 'name': '李四', 'password': '123',
'email': '3345@qq.com', 'phone': '135555555'}]>
```

5.4.4 两张表示一对多关系模型类操作

- **1.创建模型类**

在多的类中添加一个models.ForeignKey()字段,例如，SmallClass类和ShopDetail类是一对多的关系，则在创建模型类时，应当在ShopDetail类中添加ForeignKey()字段

```
class SmallClass(models.Model):
    '''它是多，BigClass是一'''
    name = models.CharField(max_length=20)
    desc = models.CharField(max_length=20)
    # 外键关联大类表
```

```
    sm_big = models.ForeignKey('BigClass', models.CASCADE)

    class Meta:
        db_table = "small_class"

    def __str__(self):
        return self.name


class ShopDetail(models.Model):
    '''它是多,SmallClass和Popular是一'''
    name = models.CharField(max_length=50)
    desc = models.CharField(max_length=100)
    # 价格
    price = models.FloatField()
    # 日期
    date = models.DateTimeField(auto_now_add=True)
    # 图片名称
    img = models.CharField(max_length=20)
    # 外键关联小类表
    shop_small = models.ForeignKey('SmallClass', models.CASCADE)
```

- **2.查询**

由多查一,可以通过关联属性查询，例如，通过一个商品查询它的分类

```
shop_datail = ShopDetail.objects.get(id=1)
shop_datail.shop_small
```

由一查多，通过属性(关联的类名小写_set.all())

```
sc = SmallClass.objects.get(id=1)
sc.shopdetail_set.all()
```

5.5 模板

5.5.1 模板的作用

在Django中，将前端的内容定义在模板中，然后再把模板交给视图调用，各种漂亮、炫酷的效果就出现了

5.5.2 模板创建

- **1.创建模板文件夹**

在项目目录内创建一个名为templates的文件夹，和应用文件夹同级

- **2.配置模板目录**

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

- **3.模板的加载**

在templates文件夹中新建tem.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>首页</title>
</head>
<body>
<h1>这是首页</h1>
</body>
</html>
```

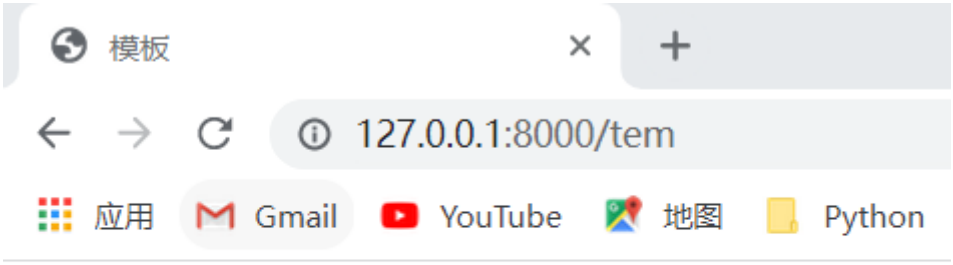
然后在视图函数tem()中

```
def tem(request):
    return render(request, 'tem.html')
```

再在app1的urls.py中添加路由

```
urlpatterns = [
    path('tem', views.tem)
]
```

在浏览器中输入http://127.0.0.1:8000/tem，如下所示



这是模板

5.5.3 模板传参

后端中必须将变量封装到字典中才允许传递到模板上

```
return render(request,'xx.html',字典数据)
```

5.5.4 模板语法

模板变量名是由数字，字母，下划线和点组成的，不能以下划线开头

- **1.模板变量使用**

{{ 模板变量名 }}

- **2.for循环**

{% for i in list %}

do something

{% endfor %}

- **3.模板变量解析顺序**

方式一：{{ shop.name }}

1）首先把shop当成一个字典，把name当成键名，进行取值shop['name']

2）把shop当成一个对象，把name当成属性，进行取值shop.name

3）把shop当成一个对象，把name当成对象的方法，进行取值shop.name

方式二：{{shop.0}}

1）首先把shop当成一个字典，把0当成键名，进行取值shop[0]

2）把shop当成一个列表，把0当成下标，进行取值shop[0]

如果解析失败，则产生内容时用空字符串填充模板变量。

5.5.5 反向解析

1.反向解析的定义

随着功能的增加会出现更多的视图，可能之前配置的路径匹配规则不够准确，于是就要修改，但是一旦修改了，之前所有对应的超链接都要修改，真是一件麻烦的事情，而且可能还会漏掉一些超链接忘记修改，有办法让链接根据路径匹配规则动态生成吗？ 就是用反向解析的办法。

2.反向解析应用范围

1）模板中超链接

2）视图中的重定向

3.使用方法

（1）项目urls.py分发时，需要为include定义namespace属性

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('app1.urls', namespace='app1'))
]
```

（2）应用的urls.py中，需要为url定义name属性，注意一定要添加app_name，否则会报错

```
app_name = "app1"

urlpatterns = [
    path('', views.index),
    path('login', views.show_login, name='show_login')
]
```

（3）模板中使用

```
<li> <a href="{% url 'app1:show_login' %}" target="_blank">登录</a></li>
```

（4）视图中的重定向

```
def test_redirect(request):
    url = reverse('app1:show_login')
    return redirect(url)
```

5.6 静态资源

静态资源配置

1.在项目下新建静态文件夹static，和应用的目录平级

2.settings.py中配置静态文件所在的物理目录

```
STATIC_URL = '/static/'
STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')]
```

3.在模板中引用静态资源

```
<!doctype html>
{% load static %}
<html>
<head>
<meta charset="utf-8">
<title>电商网页制作</title>
<link rel="stylesheet"  href="{% static 'css/index.css' %}">
```

5.7 Ajax发送请求

```
<script>
    $(function () {
        $('#btnLogin').click(function () {
            // 1.获取用户名和密码
            username = $('#exampleInputName2').val();
            password = $('#exampleInputPassword1').val();
            // 2.发起post请求 /login_ajax_check
            $.ajax({
                'url': 'login_ajax_check',
                'type': 'post',
                'data': {"username": username, 'password': password,
csrfmiddlewaretoken: '{{ csrf_token }}'},
                'dataType': 'json',
                'success':  function (data) {
                    //登录成功{'res': 1}
                    if (data.res == 0){
                        alert('用户名或密码错误')
                    }else {
                        alert('登录成功')
                        location.href='/index'
                    }

                }
            })
        })
    })

    $(function () {
        $('#btnRegister').click(function () {
            // 获取用户名和密码
            username = $('#exampleInputName2').val();
            password = $('#exampleInputPassword1').val();
            $.ajax({
                'url': 'register_ajax_check',
                'type': 'post',
                'data': {'username': username, 'password': password,
csrfmiddlewaretoken: '{{ csrf_token }}'},
                'dataType': 'json',
                'success': function (data) {
                    if(data.res=='0'){
                        alert("用户名已经存在，无法注册")
                    }else if (data.res == '2'){
                        alert("用户名或密码不能为空")
                    }else{
                        alert("注册成功")
                        location.href='/index'
                    }

                }
            })
        })
    })
</script>
```

5.8 项目实战:Django注册登录

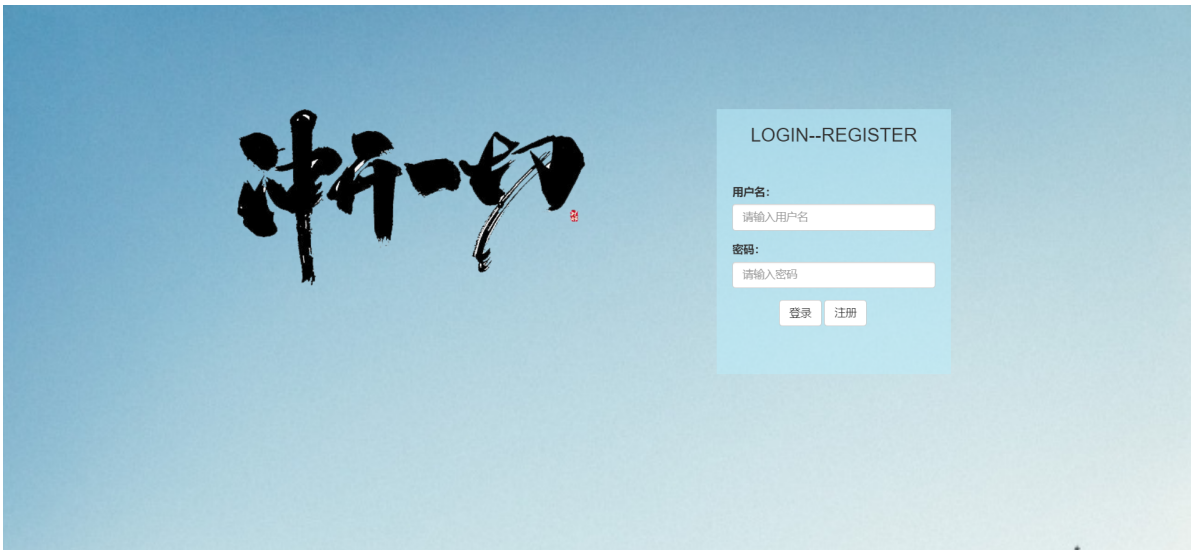
需求描述

注册登录是大多数网站所必备的功能，注册需要对输入的用户名进行校验，主要是看用户名是否合法以及该用户名有没有被注册，服务端应该根据用户输入的信息从而决定注册通过与否。登录同样需要对用户输入的用户名和密码进行比对，只有当输入的用户名和密码均正确时，才允许用户成功登录。

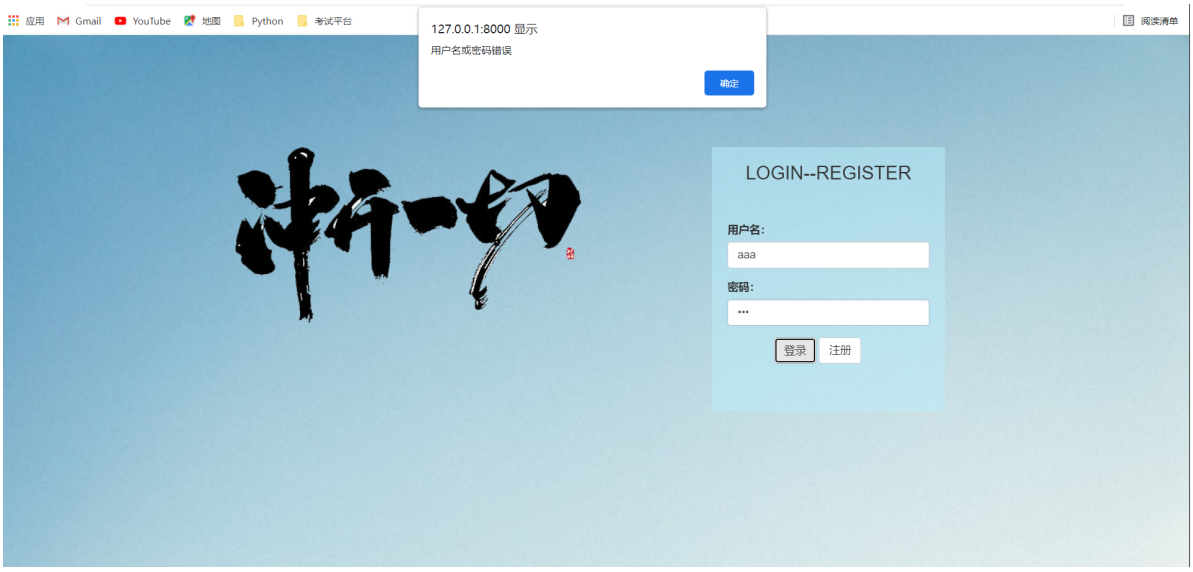
本项目中，我们将通过Django框架搭建一个网站，实现简单的用户注册登录功能，从前端获取用户输入的信息，传递到后端进行校验，并将校验后的结果返给前端。

首页效果如下

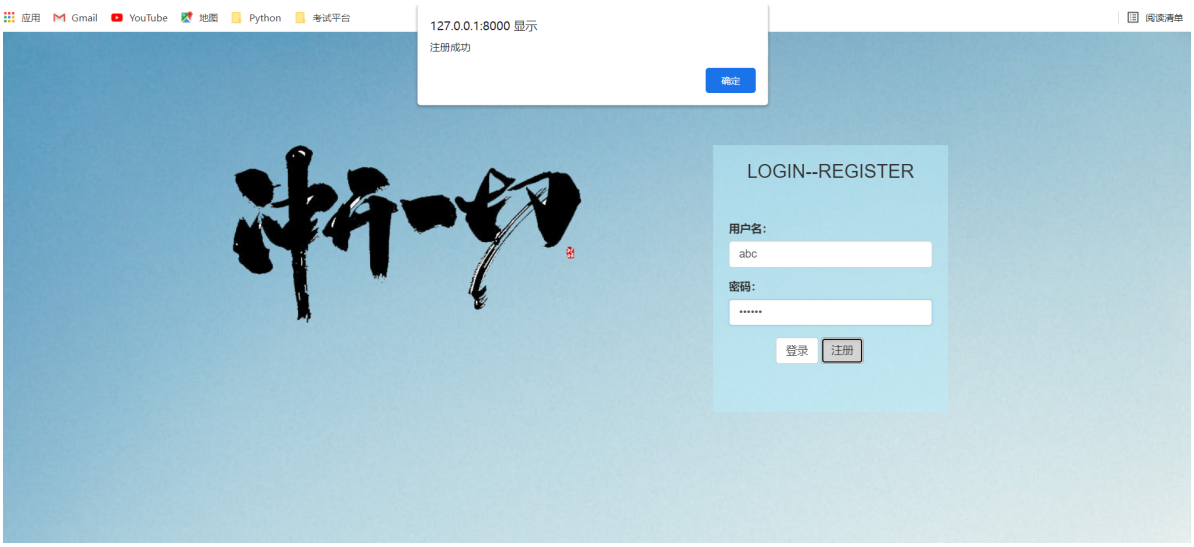
点击"登录/注册"后，效果如下



输入用户名和密码，点击登录，效果如下



注册效果如下



代码实现

1）创建模型类

```
from django.db import models


# 创建模型类
class User(models.Model):
    # 用户名
    uname = models.CharField(max_length=50)
    # 密码
    upwd = models.CharField(max_length=50)
    # 在数据库中生成的表名为应用名_类名小写,可通过Meta来规定数据库中表的名字

    class Meta:
        db_table = "user"
```

创建后通过迁移命令迁移到数据库

```
python manage.py makemigrations
python manage.py migrate
```

2）settings.py中进行配置

```
import os
from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-
9buex0n-$@wcn#kk923u$t*5h)g#q8n13l1_@9l0mzytqz@0s+'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'app1'
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'Login.urls'

# 配置模板
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
```

```
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'Login.wsgi.application'


# Database
# https://docs.djangoproject.com/en/3.2/ref/settings/#databases
# 数据库配置
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'login',
        'HOST': 'localhost',
        'USER': 'root',
        'PASSWORD': '123456',
        'PORT': '3306'
    }
}


# Password validation
# https://docs.djangoproject.com/en/3.2/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


# Internationalization
# https://docs.djangoproject.com/en/3.2/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True
```

```
USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.2/howto/static-files/

STATIC_URL = '/static/'
STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')]
# Default primary key field type
# https://docs.djangoproject.com/en/3.2/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

3）模板文件

tem.html中内容如下

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>模板</title>
</head>
<body>
<h1>这是模板</h1>
<h2>{{ name }}</h2>
<h2>{{ score }}</h2>
{% for foo in score %}
    {% if foo > 80 %}
        <p>{{ foo }}</p>
    {% else %}
        <p style="color:red">{{ foo }}</p>
    {% endif %}
{% endfor %}
<h3>{{ score.0 }}</h3>
</body>
</html>
```

index.html中内容如下

```
<!DOCTYPE html>
{% load static %}
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>首页</title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css"
integrity="sha384-
HSMxcRTRxnN+Bdg0JdbxYKrThecOKuH5zCYotlSAcp1+c8xmyTe9GYg1l9a69psu"
crossorigin="anonymous">
    <script src="/static/js/jquery-1.12.4.min.js"></script>
    <script src="/static/js/login_register.js"></script>
</head>
<body style="background-image: url('/static/img/jujia.jpg');background-size:
100%;background-repeat:repeat-y;">
```

```
<button type="button" class="btn btn-primary btn-lg active" id="ind_Login"
style="position: fixed;left: 60%">登录/注册</button>
<img src="/static/img/L.png" alt="index" style="margin: 20px 300px">

</body>

</html>
```

login_register_ajax.html中内容如下

```
<!DOCTYPE html>
{% load static %}
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>首页</title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css"
          integrity="sha384-
HSMxcRTRxnN+Bdg0JdbxYKrThecOKuH5zCYotlSAcp1+c8xmyTe9GYg1l9a69psu"
crossorigin="anonymous">
    <style>
        .Login {
            position: fixed;
            left: 60%;
            top: 20%;
            width: 300px;
            height: 340px;
            background-color: rgba(186, 237, 252, 0.5);
        }
    </style>
    <script src="/static/js/jquery-1.12.4.min.js"></script>
    <script src="/static/js/login_register.js"></script>
</head>
<body style="background-image: url('/static/img/bg.jpg');background-size:
100%;background-repeat:repeat-y;">
<img src="/static/img/col.png" alt="col" style="position: fixed;left: 20%;top:
20%">
<div class="Login">
    <div style="margin: 20px;">
        <h3 style="text-align: center">LOGIN--REGISTER</h3>
        <form style="padding-top: 40px" method="post">
            {% csrf_token %}
            <div class="form-group">
                <label for="exampleInputName2">用户名: </label>
                <input type="text" class="form-control" name="username"
id="exampleInputName2" placeholder="请输入用户名">
            </div>
            <div class="form-group">
                <label for="exampleInputPassword1">密码: </label>
                <input type="password" class="form-control" name="password"
id="exampleInputPassword1"
                       placeholder="请输入密码">
            </div>
            <button type="button" id="btnLogin" class="btn btn-default"
style="margin-left: 60px">登录</button>
```

```
            <button type="button" id="btnRegister" class="btn btn-default">注册
</button>
        </form>
    </div>
</div>
</body>
<script>
    $(function () {
        $('#btnLogin').click(function () {
            // 1.获取用户名和密码
            username = $('#exampleInputName2').val();
            password = $('#exampleInputPassword1').val();
            // 2.发起post请求 /login_ajax_check
            $.ajax({
                'url': 'login_ajax_check',
                'type': 'post',
                'data': {"username": username, 'password': password,
csrfmiddlewaretoken: '{{ csrf_token }}'},
                'dataType': 'json',
                'success':  function (data) {
                    //登录成功{'res': 1}
                    if (data.res == 0){
                        alert('用户名或密码错误')
                    }else {
                        alert('登录成功')
                        location.href='/index'
                    }

                }
            })
        })
    })

    $(function () {
        $('#btnRegister').click(function () {
            // 获取用户名和密码
            username = $('#exampleInputName2').val();
            password = $('#exampleInputPassword1').val();
            $.ajax({
                'url': 'register_ajax_check',
                'type': 'post',
                'data': {'username': username, 'password': password,
csrfmiddlewaretoken: '{{ csrf_token }}'},
                'dataType': 'json',
                'success': function (data) {
                    if(data.res=='0'){
                        alert("用户名已经存在，无法注册")
                    }else if (data.res == '2'){
                        alert("用户名或密码不能为空")
                    }else{
                        alert("注册成功")
                        location.href='/index'
                    }

                }
            })
        })
    })
```

```
</script>
</html>
```

4）视图函数

```
from django.shortcuts import render
from django.http import HttpResponse, JsonResponse
from app1.models import User


# 定义视图函数
def index(request):
    # return HttpResponse("首页")
    return render(request, 'app1/index.html')


def tem(request):
    dic = {"name": "张三", "score": [88, 67, 92]}
    return render(request, 'tem.html', dic)


def login_register(request):
    return render(request, 'app1/login_register_ajax.html')


def login_ajax_check(request):
    """ajax登录校验"""
    # 1.获取用户名和密码
    username = request.POST.get('username')
    password = request.POST.get('password')
    # 查询,通过用户名查密码
    select_user = User.objects.filter(uname=username)
    if select_user:
        if select_user[0].upwd == password:
            return JsonResponse({'res': 1})
        else:
            return JsonResponse({'res': 0})
    else:
        return JsonResponse({'res': 0})


def register_ajax_check(request):
    # 获取用户名和密码
    username = request.POST.get('username')
    password = request.POST.get('password')
    select_user = User.objects.filter(uname=username)
    if username and password:
        # 说明用户已经存在
        if select_user:
            return JsonResponse({'res': 0})
        else:
            User.objects.create(uname=username, upwd=password)
            return JsonResponse({'res': 1})
    else:
        # 说明用户名或密码为空
        return JsonResponse({'res': 2})
```

5）路由

项目中的urls.py中内容如下

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('app1.urls'))
]
```

应用中的urls.py中的内容如下

```
from app1 import views
from django.urls import path

# http://127.0.0.1:8000/index,首先会将index拿去项目中的urls.py中进行匹配，匹配上之后，
# 删掉已经匹配的部分，再拿到应用中进行匹配，匹配上哪个就执行哪个视图函数
urlpatterns = [
    path('index', views.index),
    path('tem', views.tem),
    path('login_register', views.login_register),
    path('login_ajax_check', views.login_ajax_check),
    path('register_ajax_check', views.register_ajax_check)
]
```

六、Selenium

6.1 Selenium介绍和安装

6.1.1 网络爬虫回顾

爬虫就是模拟客户端发送网络请求，接收响应，一种按照一定的规则，自动地抓取互联网信息的程序

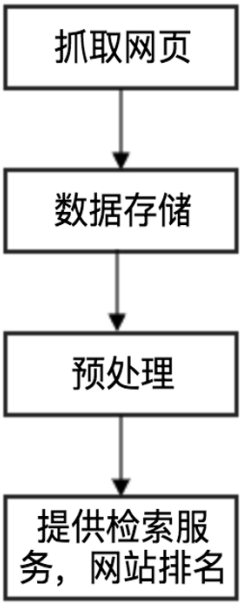
爬虫的分类

1.通用网络爬虫(搜索引擎使用,遵守robots协议)

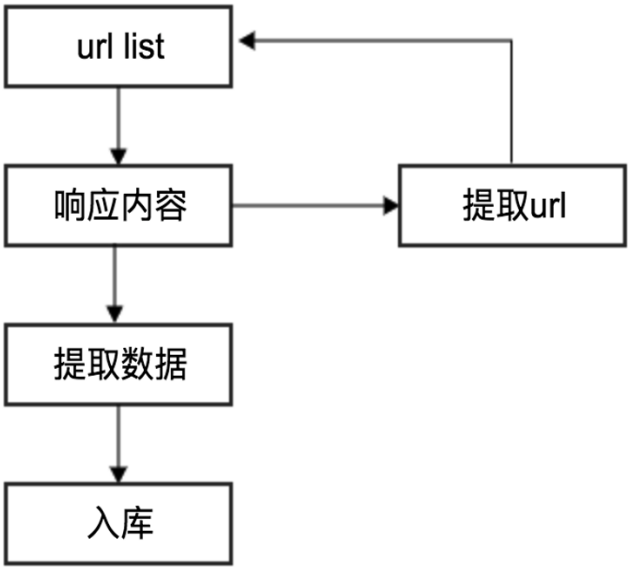
2.聚焦网络爬虫：针对特定网站的爬虫

爬虫的步骤

搜索引擎流程



聚焦爬虫流程



发送请求

```
import requests
res = requests.get(url,headers=headers)
```

提取数据

```
from lxml import etree
p = etree.HTML(response)
p.xpath('XPath表达式')
```

数据存储

CSV、TXT、JSON、MySQL、MongoDB等

6.1.2 Selenium简介

Selenium本身是一个Web的自动化测试工具，最初是为网站自动化测试而开发的，其就像游戏的遥控手柄，可以按指定的命令自动操作。如果作为爬虫框架，Selenium可以根据指令，让浏览器自动加载页面，获取需要的数据，甚至页面提交。Selenium支持众多浏览器包括IE浏览器、火狐浏览器、Safari和谷歌浏览器等。可以通过Selenium库中WebDriver与页面上的元素进行交互(包括发送文本、点击操作等)，以及执行其他动作来运行网络爬虫，就像真正的用户在操作一样。

6.1.3 安装Selenium

安装命令为：

```
pip install selenium
```

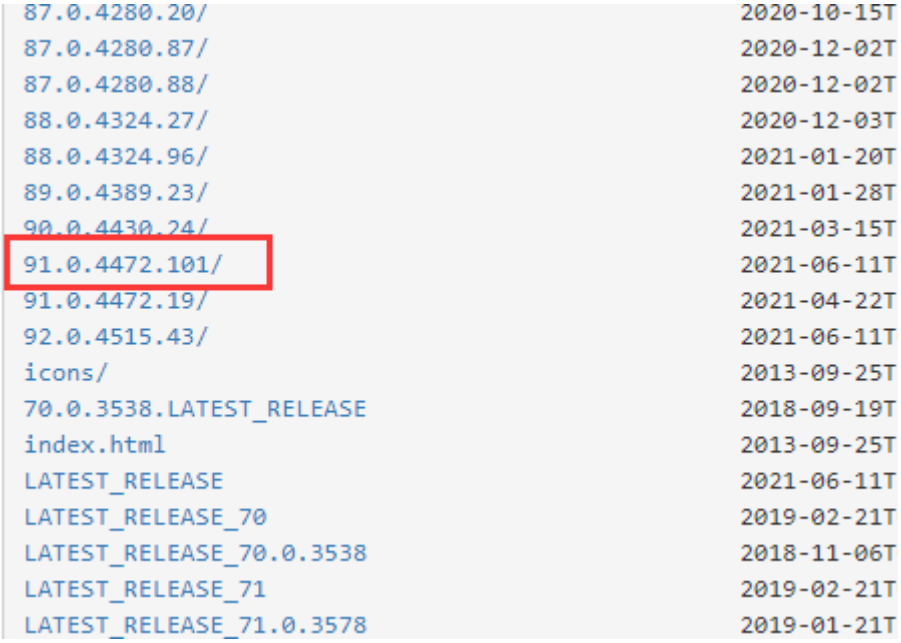
6.1.4 下载谷歌驱动

- **1.查看谷歌浏览器版本**：

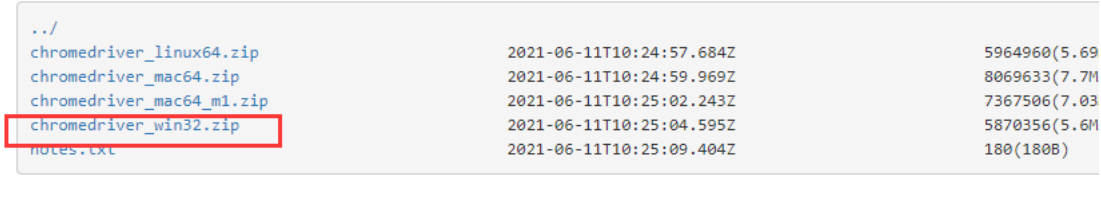
驱动地址：https://npm.taobao.org/mirrors/chromedriver/

- **2.下载**

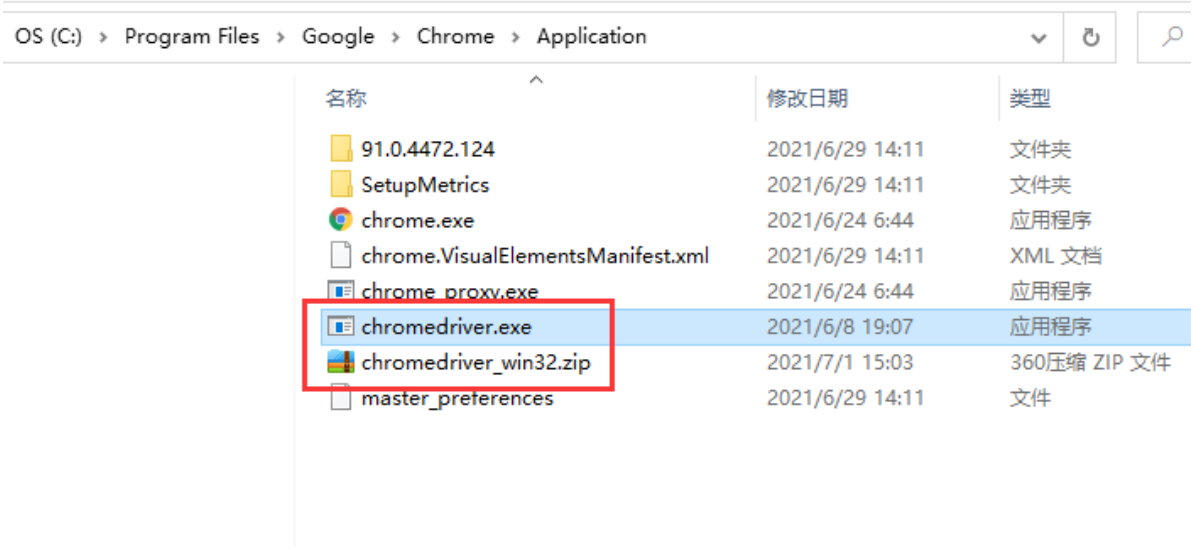
找到和浏览器对应的驱动版本



找到chromedriver_win32.zip进行下载

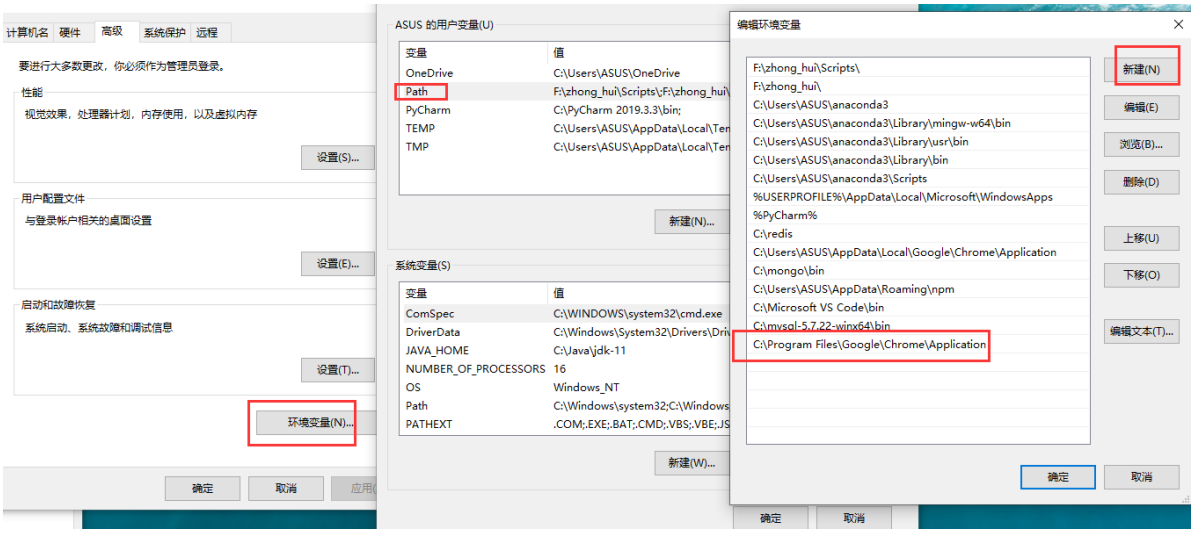


将下载后的文件进行解压，得到chromedriver.exe



- **3.配置系统环境变量**

现在需要为chromedriver.exe添加环境变量

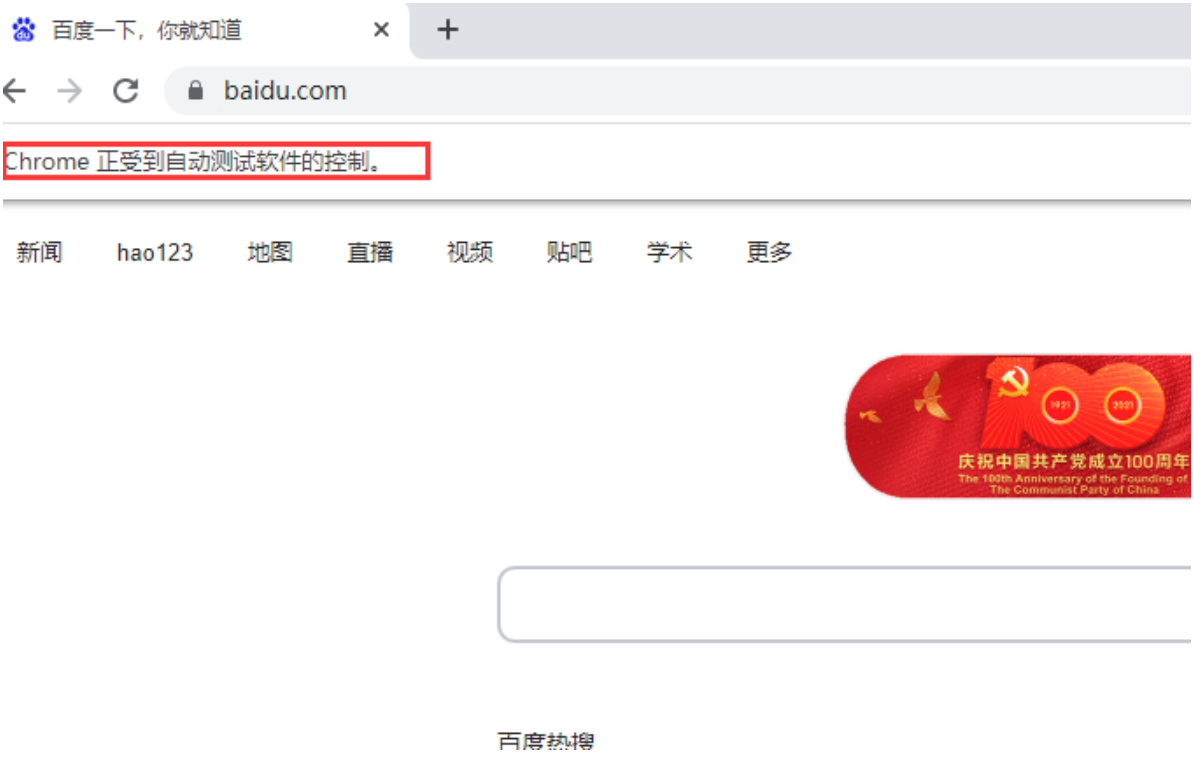


- **4.启动浏览器**

代码：

```
# 导入webdriver类
from selenium import webdriver
# 调用控制插件
driver = webdriver.Chrome()
# 启动浏览器
driver.get("http://www.baidu.com")
```

运行结果：



6.2 模拟用户操作

- **1.浏览器对象的方法**

1.browser = webdriver.Chrome()

2.browser.get(url) # 发送get请求

3.browser.page_source # HTML结构源码

4.browser.page_source.find('字符串') #从html源码中搜索指定字符串,没有找到返回-1

5.browser.quit() # 关闭浏览器

6.browser.execute_script('javascript')

7.browser. save_screenshot('baidu.png') 获取快照

- **2.定位操作**

find_element找到的是WebElement节点对象，但是没找到会抛出ElementException异常。
find_elements返回一个列表，列表里元素是WebElement节点对象，find_elements如果没有找到，它返回一个空列表

Webdriver By	含义
find_element_by_xpath()	根据xpath查找，获取一个对象
find_elements_by_xpath()	根据xpath查找，获取对象列表
find_element_by_id()	根据id查找，获取一个对象
find_element_by_class_name()	根据class属性名查找

- **3.节点对象操作**

1.ele.send_keys('') 搜索框发送内容
2.ele.click() 点击
3.ele.text 获取文本内容，包含子节点和后代节点的文本内容
4.ele.get_attribute('src') 获取属性值

- **4.切换页面**

适用网站

页面中点开链接出现新的页面，但是浏览器对象browser还是之前页面的对象

应对方案

```
# 获取当前所有句柄（窗口）
all_handles = browser.window_handles
# 切换browser到新的窗口，获取新窗口的对象
browser.switch_to.window(all_handles[1])
```

6.3 项目实战:京东爬虫

需求描述

我们通过爬虫去互联网上抓取数据的时候，有时候遇到动态网页的时，通过requests库抓取数据很可能要通过破解js等一些列繁琐且高难度的操作才能获取到数据，这会花费我们大量的时间和精力。如果我们对抓取数据的速度没有太大的要求，这时候我们可以考虑用Selenium去完成。Selenium可以操控浏览器去加载某些页面，并提取页面中的数据，在应对复杂的反爬往往能起到很好的效果。

在本项目中，我们将通过Selenium控制浏览器，去京东网上搜索商品，并抓取商品的名字、价格、评论、商家等信息。

代码实现

```
from selenium import webdriver
from selenium.webdriver.common.by import By
import time


class JdSpider:
    def __init__(self):
        self.browser = webdriver.Chrome()
        self.browser.get('https://www.jd.com/')

    def get_html(self):
        self.browser.find_element_by_id('key').send_keys('龙泉武士')
        self.browser.find_element(By.CLASS_NAME, 'button').click()
        time.sleep(2)

    def get_one_page(self):

 self.browser.execute_script('window.scrollTo(0,document.body.scrollHeight)')
        time.sleep(2)
        li_list = self.browser.find_elements(By.XPATH, '//ul[@class="gl-warp
clearfix"]/li')
        for li in li_list:
            name = li.find_element(By.XPATH, './/div[@class="p-name p-name-type-
2"]/a/em').text
            price = li.find_element_by_xpath('.//div[@class="p-
price"]/strong').text
```

```
            comment = li.find_element_by_xpath('.//div[@class="p-
commit"]/strong/a').text
            shop = li.find_element_by_xpath('.//div[@class="p-
shop"]/span/a').text
            print(name, price, comment, shop)
        print("*" * 50)

    def get_all_page(self):
        while True:
            self.get_one_page()
            self.browser.find_element_by_class_name('pn-next').click()
            if self.browser.page_source.find('pn-next disabled') != -1:
                self.get_one_page()
                print("抓取结束")
                return

    def run(self):
        self.get_html()
        self.get_all_page()
        self.browser.quit()


if __name__ == "__main__":
    spider = JdSpider()
    spider.run()
```

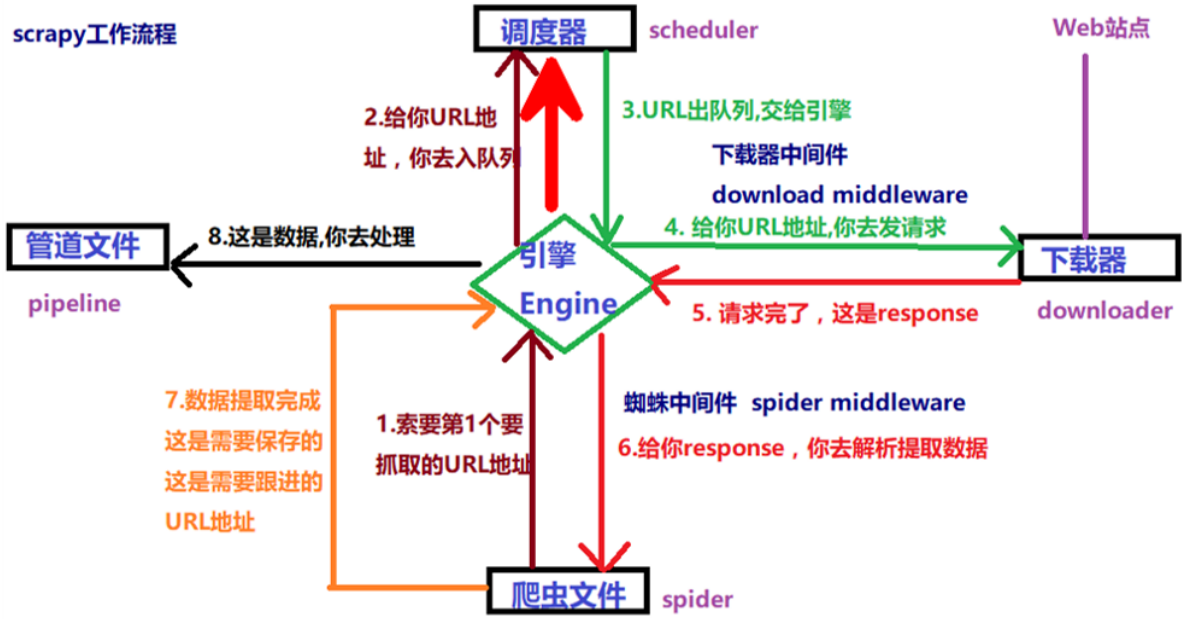
七、Scrapy框架

7.1 Scrapy框架介绍与安装

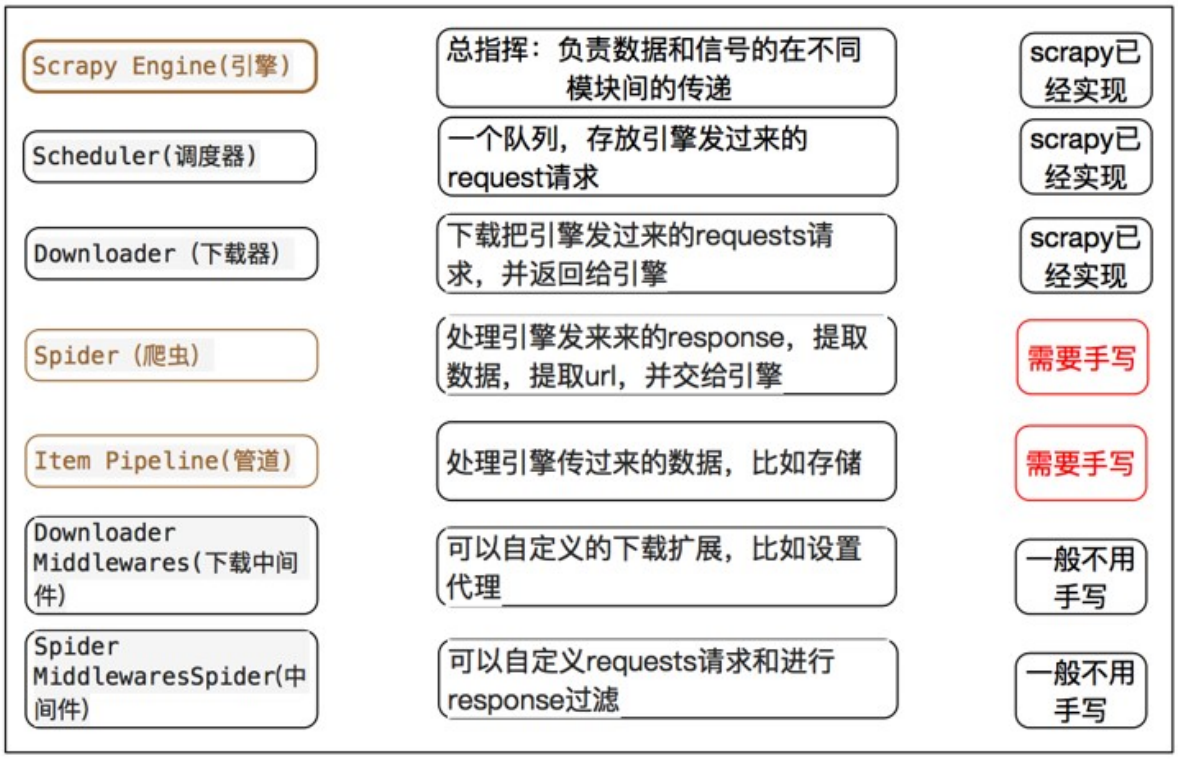
Scrapy框架是一个异步处理框架,可配置和可扩展程度非常高,Python中使用最广泛的爬虫框架，在使用之前需要安装。

安装命令为:pip install scrapy

7.2 Scrapy框架核心组件



Scrapy框架包括五大核心组件，分别是引擎、调度器、下载器、爬虫文件、管道。此外下载器中间件(Downloader Middlewares) 位于引擎和下载器,用于包装请求(随机代理等)，蜘蛛中间件(Spider Middlewares) 位于引擎和爬虫文件,可修改响应对象属性。



7.3 创建Scrapy项目

通过命令创建

```
scrapy startproject 项目名字
cd 项目名
scrapy genspider 爬虫名 域名
```

项目目录结构

Baidu 项目文件夹

|------Baidu 项目目录

| |------items.py 定义抓取的数据结构

| |------middlewares.py 中间件

| |------pipelines.py 数据处理

| |------settings.py 全局配置

| |------spiders 自己定义的spider文件夹

| |-----baidu.py 定义的爬虫文件

|--------scrapy.cfg 项目基本配置文件

运行爬虫

scrapy crawl 爬虫名

7.4 编写爬虫代码

- **1.settings.py配置文件**

1）定义User-Agent
USER_AGENT = 'Mozilla/5.0'
2）是否遵循robots协议，一般设置为False
ROBOTSTXT_OBEY = False
3）最大并发量，默认为16
CONCURRENT_REQUESTS = 32
4）下载延迟时间
DOWNLOAD_DELAY = 1
5）请求头，此处也可以添加User-Agent
DEFAULT_REQUEST_HEADERS={}
6）项目管道,300为优先级,1-1000,数字越小,优先级越高
ITEM_PIPELINES={'项目目录名.pipelines.类名':300}
7）是否启用Cookies,设置为Fasle表示启用
COOKIES_ENABLED = False

- **2.items.py**

其作用为定义要抓取的数据结构

例如:

```
name = scrapy.Field()
title = scrapy.Field()
```

- **3.爬虫文件**

1)创建爬虫生成的class XXXSpider(scrapy.Spider)类中几个重要的属性

name：是字符串，用于设置spider的名字，实际中一般为每个独立网站创建一个spider。

allowed_domains：是一个字符串列表。规定了允许爬取的网站域名，非域名下的网页将被自动过滤。

strart_urls 是一个必须定义包含初始请求页面URL的列表。start_requests()方法会引用该属性，发出初始的Request。

parse(self,response)：引擎默认的响应处理函数。如果没有在Request中指定响应处理函数，那么引擎会调用这个函数。

2）parse(self,response)中的response对象

response对象常用属性如下表所示

属性	作用
body	HTTP响应正文，bytes类型
text	文本形式的HTTP响应正文，str类型，由response.body解码得到
xpath(query)	通过xpath表达式从下载的页面中提取数据

3）数据交给管道

```
from ..items import xxxItem

item=xxxItem()

yield item
```

4）请求交给调度器

```
yield scrapy.Request(url=url,callback=self.parse)
```

- **4.管道文件pipelines.py**

管道文件会对爬虫文件中传递过来的item数据做处理，将数据进行持久化存储，如果要存入数据库，需要写open_spider和close_spider方法

```
def open_spider(self,spider):
      爬虫开始执行1次,用于数据库连接

def close_spider(self,spider):
      爬虫结束时执行1次,用于断开数据库连接
```

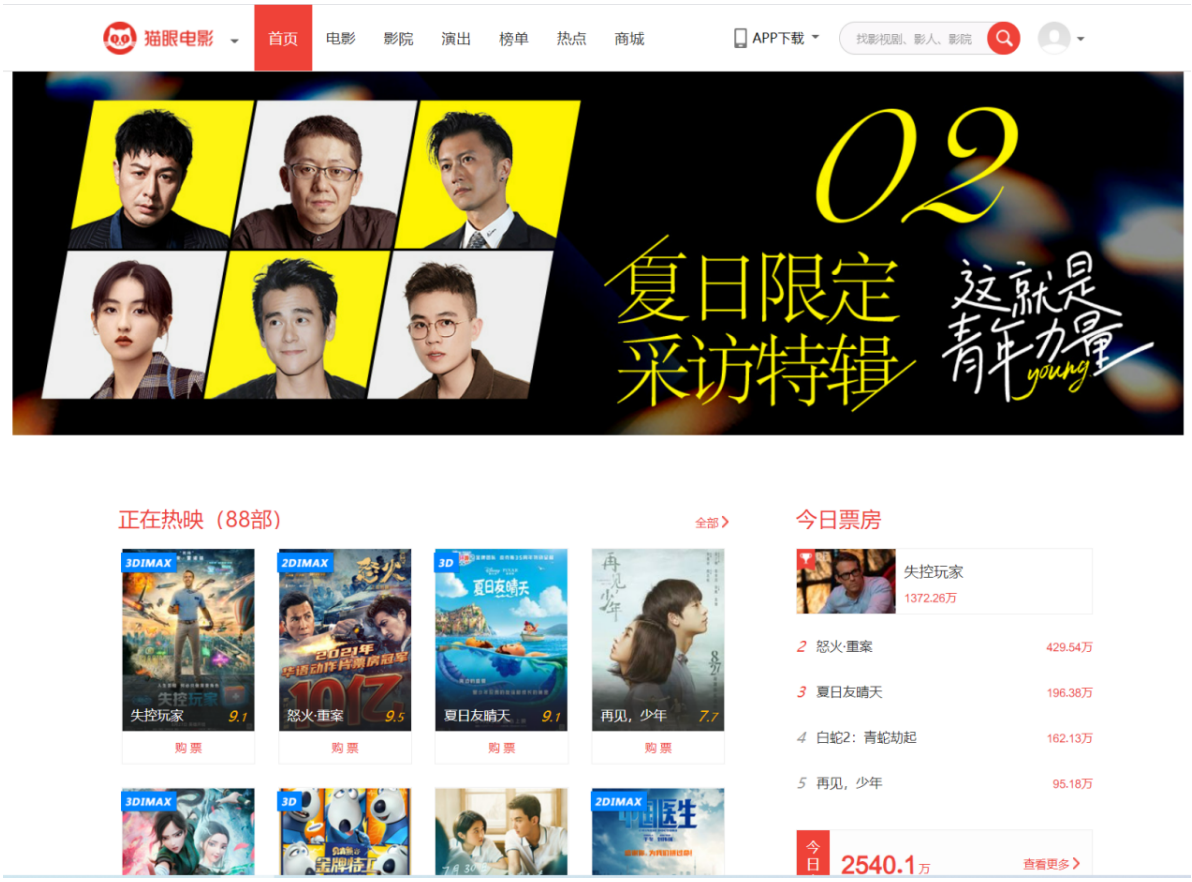
- **5.命令行数据存储**

```
# 导出CSV文件
scrapy crawl maoyan -o maoyan.csv
# 导出JSON文件
scrapy crawl maoyan -o maoyan.json

# settings.py中设置导出编码(针对json文件)
FEED_EXPORT_ENCODING = 'utf-8'
```

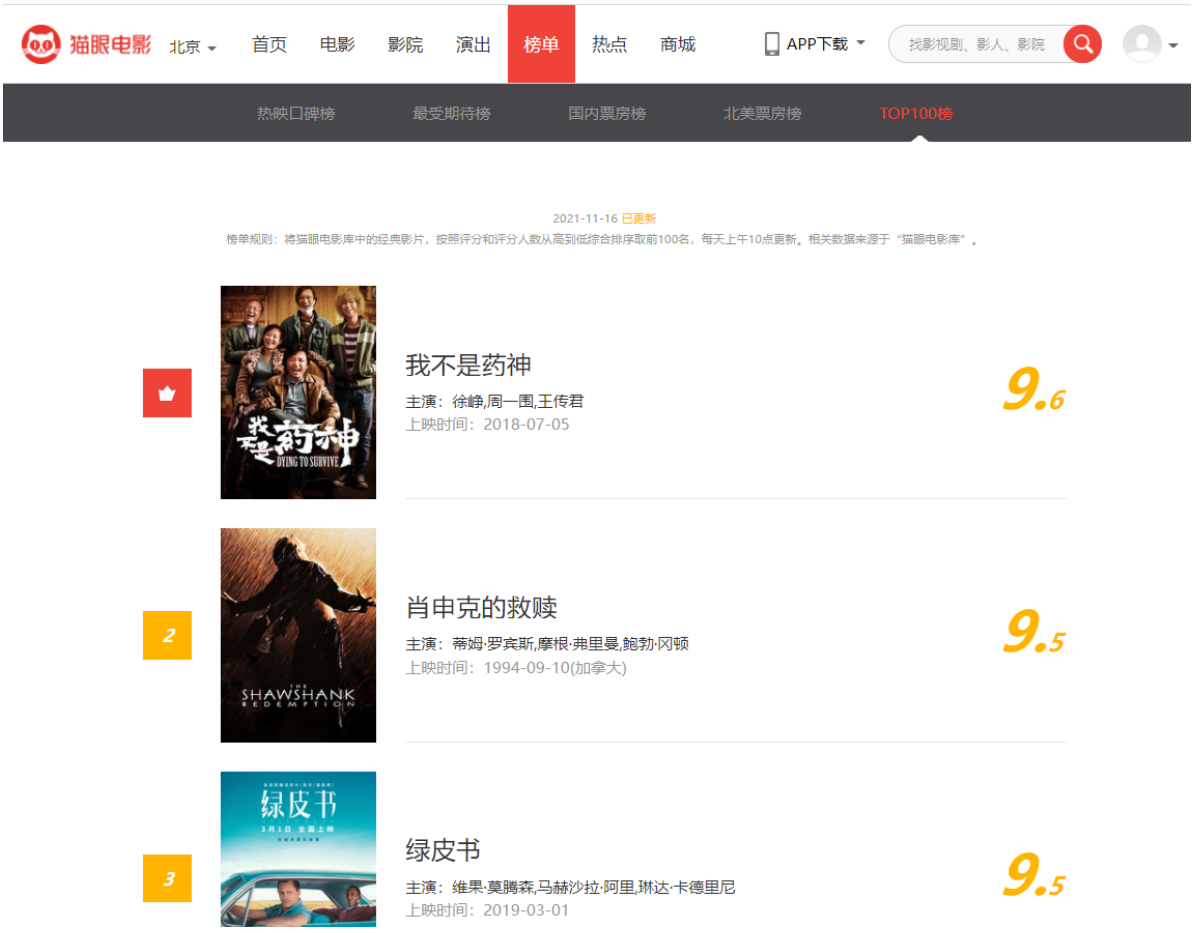
7.5 项目实战:猫眼爬虫

需求描述

猫眼电影是一个电影网站，里面有大量的电影资源，包括电影、影院、演出、榜单、热点等，如下所示。



现搭建一个Scrapy爬虫框架，爬取榜单里面的TOP100榜，爬取的内容包括电影名、主演、上映时间等，如下图所示。



代码实现

1）items.py文件

```
import scrapy

class MaoyanItem(scrapy.Item):
    # define the fields for your item here like:
    name = scrapy.Field()
    actor = scrapy.Field()
    on_time = scrapy.Field()
```

2）爬虫文件

```
import scrapy
from ..items import MaoyanItem

class MaoyanSpider(scrapy.Spider):
    name = 'maoyan'
    allowed_domains = ['www.maoyan.com']
    start_urls = ['https://www.maoyan.com/board/4']
    base_url = "https://www.maoyan.com/board/4?offset={}"
    i = 0

    def parse(self, response):
        dd_list = response.xpath('//dl[@class="board-wrapper"]/dd')
        for dd in dd_list:
            item = MaoyanItem()
            item['name'] =
dd.xpath('.//p[@class="name"]/a/text()').get().strip()
            item['actor'] = dd.xpath('.//p[@class="star"]/text()').get().strip()
            item['on_time'] =
dd.xpath('.//p[@class="releasetime"]/text()').get().strip()
            yield item
        if self.i < 90:
            self.i += 10
            url = self.base_url.format(self.i)
            yield scrapy.Request(url=url, callback=self.parse)
```

3）settings.py文件

```
BOT_NAME = 'Maoyan'

SPIDER_MODULES = ['Maoyan.spiders']
NEWSPIDER_MODULE = 'Maoyan.spiders'


# Crawl responsibly by identifying yourself (and your website) on the user-agent
#USER_AGENT = 'Maoyan (+http://www.yourdomain.com)'

# Obey robots.txt rules
ROBOTSTXT_OBEY = False

# Configure maximum concurrent requests performed by Scrapy (default: 16)
CONCURRENT_REQUESTS = 1

# Configure a delay for requests for the same website (default: 0)
# See https://docs.scrapy.org/en/latest/topics/settings.html#download-delay
# See also autothrottle settings and docs
DOWNLOAD_DELAY = 2
```

```
# The download delay setting will honor only one of:


# Override the default request headers:
DEFAULT_REQUEST_HEADERS = {
  'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
  'Accept-Language': 'en',
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/94.0.4606.54 Safari/537.36'
}


ITEM_PIPELINES = {
   'Maoyan.pipelines.MaoyanPipeline': 300,
  'Maoyan.pipelines.MaoyanMysqlPpeline': 200
}


FEED_EXPORT_ENCODING = 'utf-8'
```

4）管道文件

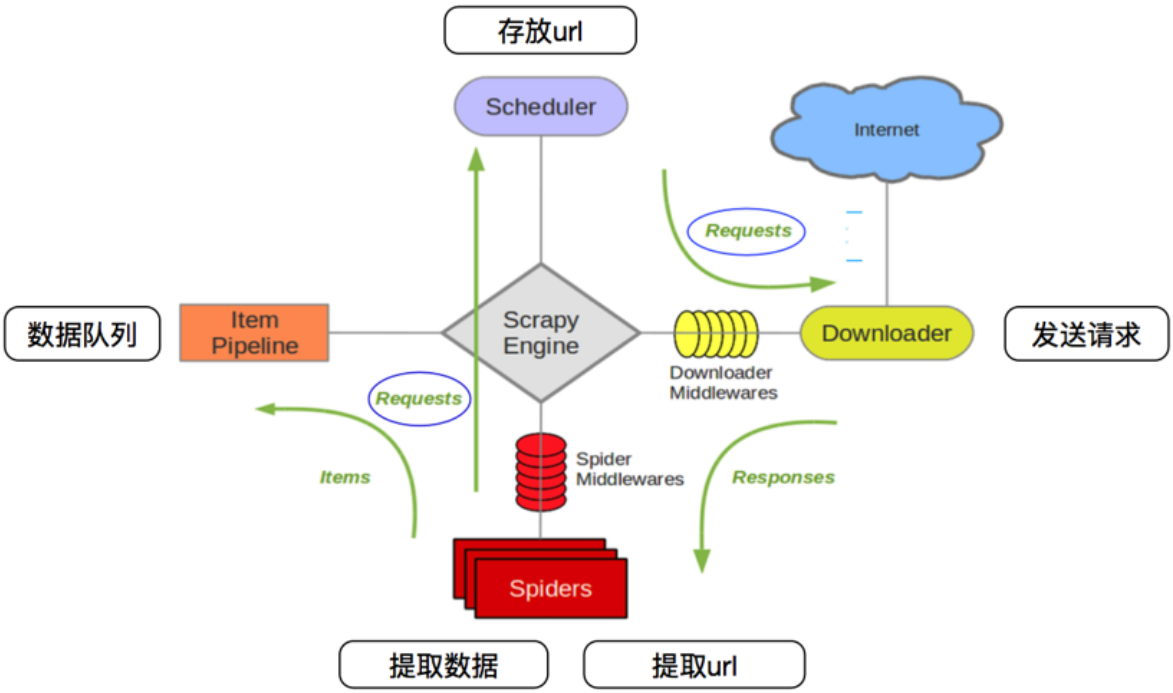
```
import pymysql


class MaoyanPipeline:
    def process_item(self, item, spider):
        print(item)
        return item


class MaoyanMysqlPpeline:
    def open_spider(self, spider):
        print("开启爬虫")
        self.db = pymysql.connect(host='localhost', user='root',
password='123456',
                                  database='maoyan', port=3306, charset='utf8')
        self.cursor = self.db.cursor()

    def process_item(self, item, spider):
        sql_insert = "insert into maoyantab values(%s,%s,%s)"
        self.cursor.execute(sql_insert, [item['name'], item['actor'],
item['on_time']])
        self.db.commit()
        return item

    def close_spider(self, spider):
        print("关闭爬虫")
        self.cursor.close()
        self.db.close()
```

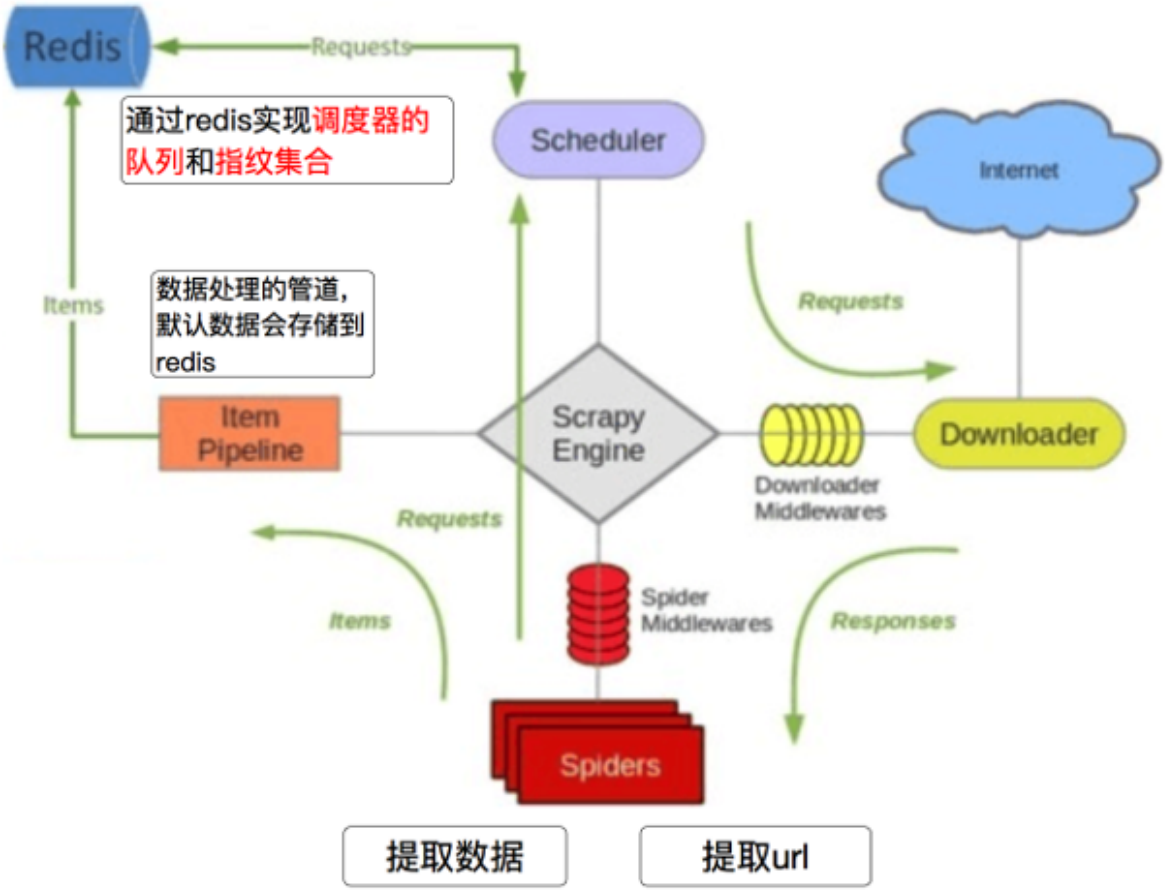
八、分布式爬虫框架Scrapy-Redis

8.1 Scrapy-Redis简介

Scrapy-Redis(Redis-based components for Scrapy)在scrapy的基础上实现了更多，更强大的功能，具体体现在：reqeust去重，爬虫持久化，和轻松实现分布式。

Scrapy-redis的思路是改变了Scrapy的队列调度，将起始的网址从start_urls里分离出来，改为从Redis读取，这样多个客户端可以同时读取同一个Redis；同时分布式爬虫让所有机器上的爬虫程序，从同一个queue队列中获取request请求，并且每个机器取出request，直到所有的request被请求完毕。





8.2 Scrapy-Redis安装

可以通过命令pip install scrapy-redis 和pip install scrapy-redis-cluster进行Scrapy-Redis的安装。

8.3 项目搭建

1）通过命令创建项目

同Scrapy框架创建项目一样

```
scrapy startproject 项目名字
cd 项目名
scrapy genspider 爬虫名 域名
```

2）爬虫文件中的变化

爬虫类继承自RedisSpider，将start_urls去掉，增加redis_key

```
import scrapy
from scrapy_redis.spiders import RedisSpider


class MaoyanSpider(RedisSpider):
    name = 'maoyan'
    allowed_domains = ['maoyan.com']
    redis_key = 'maoyan'
```

3）settings.py中需要增加如下配置项

```
# 去重类
DUPEFILTER_CLASS = "scrapy_redis.dupefilter.RFPDupeFilter"
# 调度器类
SCHEDULER = "scrapy_redis.scheduler.Scheduler"
# 要持久化
SCHEDULER_PERSIST = True
# Redis数据库配置
REDIS_URL = "redis://127.0.0.1:6379"
# 管道配置
ITEM_PIPELINES = {
   'MaoyanRedis.pipelines.MaoyanredisPipeline': 300,
   # 如果需要将数据保存到redis数据库中，添加此项配置，否则不用添加
    'scrapy_redis.pipelines.RedisPipeline': 400
}
```

8.4项目实战：分布式猫眼爬虫

需求描述

将前面的猫眼爬虫改为通过Scrapy-Redis框架实现

代码实现

1）items.py

```
import scrapy

class MaoyanredisItem(scrapy.Item):
    # define the fields for your item here like:
    name = scrapy.Field()
    actor = scrapy.Field()
    on_time = scrapy.Field()
```

2）爬虫文件

```
import scrapy
from scrapy_redis.spiders import RedisSpider
from ..items import MaoyanredisItem


class MaoyanSpider(RedisSpider):
    name = 'maoyan'
    allowed_domains = ['maoyan.com']
    redis_key = 'maoyan'
    base_url = "https://www.maoyan.com/board/4?offset={}"
    i = 0

    def parse(self, response):
        dd_list = response.xpath('//dl[@class="board-wrapper"]/dd')
        for dd in dd_list:
            item = MaoyanredisItem()
            item['name'] =
dd.xpath('.//p[@class="name"]/a/text()').get().strip()
            item['actor'] = dd.xpath('.//p[@class="star"]/text()').get().strip()
            item['on_time'] =
dd.xpath('.//p[@class="releasetime"]/text()').get().strip()
            yield item
        if self.i < 90:
            self.i += 10
            url = self.base_url.format(self.i)
            yield scrapy.Request(url=url, callback=self.parse)
```

3）settings.py文件

```
BOT_NAME = 'MaoyanRedis'

SPIDER_MODULES = ['MaoyanRedis.spiders']
NEWSPIDER_MODULE = 'MaoyanRedis.spiders'


# Obey robots.txt rules
ROBOTSTXT_OBEY = False

# 去重类
DUPEFILTER_CLASS = "scrapy_redis.dupefilter.RFPDupeFilter"
# 调度器类
SCHEDULER = "scrapy_redis.scheduler.Scheduler"
# 要持久化
SCHEDULER_PERSIST = True
# Redis数据库配置
REDIS_URL = "redis://127.0.0.1:6379"
```

```
# Configure maximum concurrent requests performed by Scrapy (default: 16)
CONCURRENT_REQUESTS = 1

# Configure a delay for requests for the same website (default: 0)
DOWNLOAD_DELAY = 2
# The download delay setting will honor only one of:
#CONCURRENT_REQUESTS_PER_DOMAIN = 16
#CONCURRENT_REQUESTS_PER_IP = 16

# Disable cookies (enabled by default)
#COOKIES_ENABLED = False

# Disable Telnet Console (enabled by default)
#TELNETCONSOLE_ENABLED = False

# Override the default request headers:
DEFAULT_REQUEST_HEADERS = {
  'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
  'Accept-Language': 'en',
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/94.0.4606.54 Safari/537.36'
}


#SPIDER_MIDDLEWARES = {
#    'MaoyanRedis.middlewares.MaoyanredisSpiderMiddleware': 543,
#}

# Enable or disable downloader middlewares
# See https://docs.scrapy.org/en/latest/topics/downloader-middleware.html
#DOWNLOADER_MIDDLEWARES = {
#    'MaoyanRedis.middlewares.MaoyanredisDownloaderMiddleware': 543,
#}

# Enable or disable extensions
# See https://docs.scrapy.org/en/latest/topics/extensions.html
#EXTENSIONS = {
#    'scrapy.extensions.telnet.TelnetConsole': None,
#}

# Configure item pipelines
# See https://docs.scrapy.org/en/latest/topics/item-pipeline.html
ITEM_PIPELINES = {
   'MaoyanRedis.pipelines.MaoyanredisPipeline': 300,
    'scrapy_redis.pipelines.RedisPipeline': 400
}
```

4）管道文件

```
class MaoyanredisPipeline:
    def process_item(self, item, spider):
        print(item)
        return item
```

5）启动爬虫

先编写run.py文件，使用命令启动爬虫，和Scrapy框架一致

```
from scrapy import cmdline

cmdline.execute('scrapy crawl maoyan'.split())
```

然后在redis_key中推入url,命令如下

```
lpush maoyan https://www.maoyan.com/board/4
```