影视爬虫与检索系统设计文档

一、总体简介

本实验通过python爬虫来建立本地影视检索数据库，并学习利用django搭建网站来进行可视化检索。

二、影视爬虫部分

**1.实现方法**

影视爬虫选择的爬虫网站为豆瓣电影，爬虫所使用的python库为requests何beautifulsoup4，前者用来拉取网页到本地，并储存为txt文件，后者用来分析网页，并将分析结果储存在数据库中。

爬虫首先通过：

https://movie.douban.com/j/new\_search\_subjects?sort=T&tags=%E6%AC%A7%E7%BE%8E&start=0&limit=1000拉取豆瓣欧美电影的top1000。该网站可以拉取1000条电影的json文档，通过对json文档的分析，将相应的电影名、评分、URL等数据存储在数据库中。通过电影页面的URL爬取1000个电影的HTML页面，从中提取到电影导演、编剧、演员（及演员界面的URL，<=10）、语言、制片国家、又名、分类、市场、上映日期、IMDb链接、简介、评论（5条）。通过演员的URL抓取演员的网页，并从中抓取演员英文名、性别、星座、出生日期、出生地、职业、别名、家庭成员、IMdb链接、简介。

其中对于没有爬虫头的爬虫是不能爬到任何内容的，我设置的hearders为：

headers = {  
 "Accept": "application/json, text/javascript, \*/\*; q=0.01",  
 "X-Requested-With": "XMLHttpRequest",  
 "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.83 Safari/537.36",  
 "Content-Type": "application/x-www-form-urlencoded; charset=UTF-8"}

但是豆瓣还是会封，因此我使用了IP代理的技术爬完了共6000+条数据。

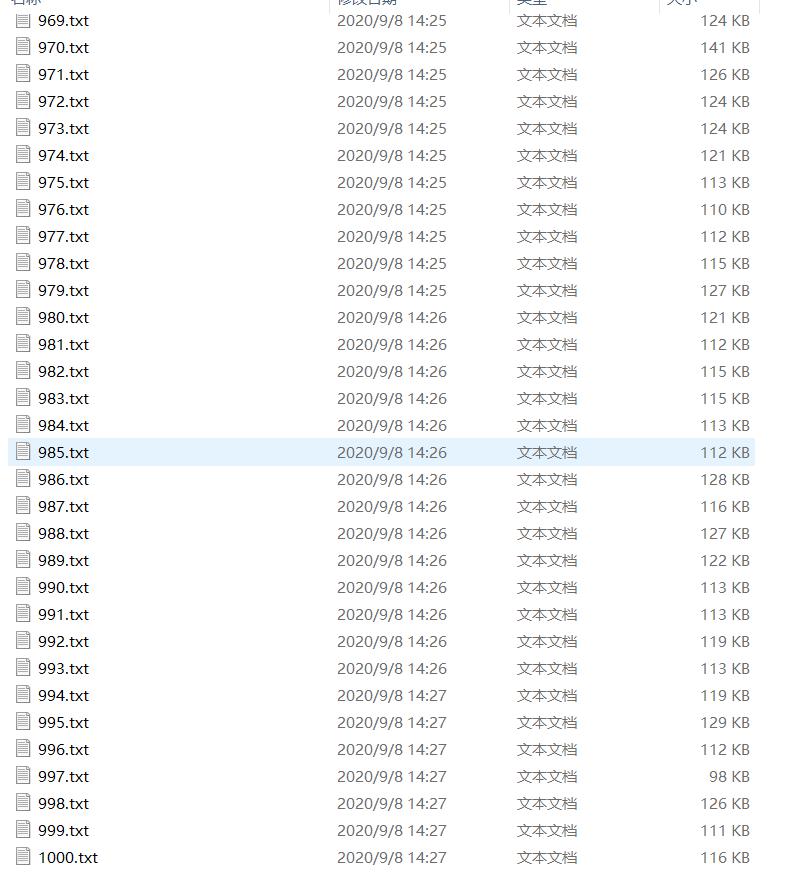
数据储存方面，通过django的ORM技术，我设计了三个表：

class Movie(models.Model):  
 id = models.AutoField(primary\_key=True)  
 url = models.CharField(max\_length=100)  
 rate = models.FloatField()  
 title = models.CharField(max\_length=100, null=True)  
 summary = models.CharField(max\_length=6000, null=True)  
  
 diretors = models.CharField(max\_length=200, null=True)  
 scriptwriters = models.CharField(max\_length=200, null=True)  
 categories = models.CharField(max\_length=200, null=True)  
 dates = models.CharField(max\_length=200, null=True)  
 language = models.CharField(max\_length=100, null=True)  
 country = models.CharField(max\_length=100, null=True)  
 other\_names = models.CharField(max\_length=100, null=True)  
 length = models.CharField(max\_length=100, null=True)  
 IMDb = models.CharField(max\_length=100, null=True)  
  
  
  
class Actor(models.Model):  
 id = models.AutoField(primary\_key=True)  
 url = models.CharField(max\_length=100)  
 name = models.CharField(max\_length=100, null=True)  
 eng\_name = models.CharField(max\_length=100, null=True)  
 movies = models.ManyToManyField(**'Movie'**, related\_name=**'actors'**)  
  
 gender = models.CharField(max\_length=100, null=True)  
 star\_sign = models.CharField(max\_length=100, null=True)  
 birth\_date = models.CharField(max\_length=100, null=True)  
 birth\_place = models.CharField(max\_length=100, null=True)  
 occupation = models.CharField(max\_length=100, null=True)  
 other\_names = models.CharField(max\_length=200, null=True)  
 families = models.CharField(max\_length=200, null=True)  
 summary = models.CharField(max\_length=6000, null=True)  
 IMDb = models.CharField(max\_length=100, null=True)  
 has\_img = models.BooleanField(null=True)  
 colleague = models.CharField(max\_length=100, null=True)  
  
class Comment(models.Model):  
 id = models.AutoField(primary\_key=True)  
 description = models.CharField(max\_length=2000, null=True)  
 movie = models.ForeignKey(**'Movie'**, on\_delete=models.CASCADE, related\_name=**'comments'**)

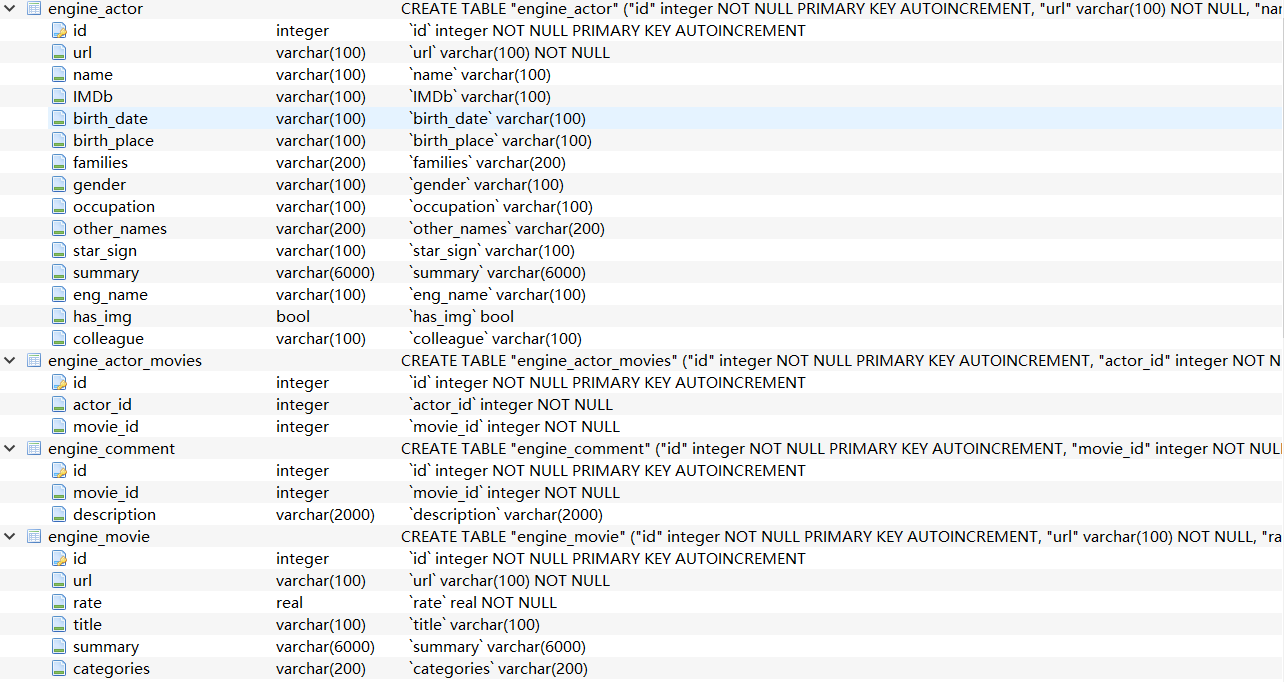
其中电影和评论是一对多的关系，电影和演员是多对多的关系，均通过外键来连接，在django储存时，Comment表包含了movie\_id的外键，同时新建了Movie\_Actor的表，用来存储多对多关系。

**2.结果展示**

爬取文件展示（部分）：

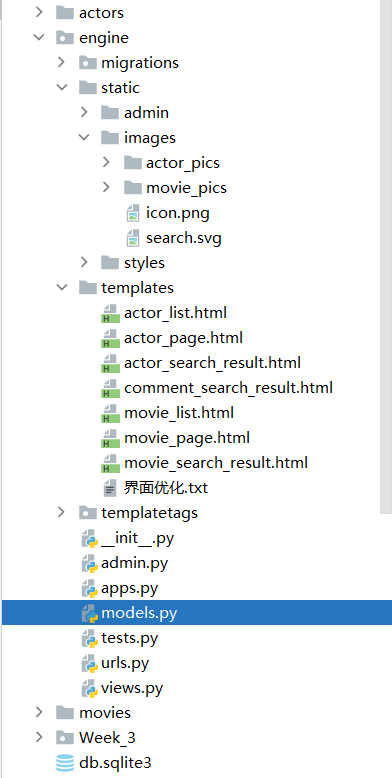


数据库展示：



三、检索系统部分

Django使用Pycharm直接创建项目而不用配置环境。创建新APP：engine并进行一系列环境配置之后，进行项目的编写，django项目的文件树为：



（其中movie和actor文件夹中为原始网页文件）

项目的url设置为search（搜索页面，电影、演员、评论，共三个）、display（列表页面，电影、演员，共两个）、detail（详细介绍页面，电影、演员，共两个），其中主页会自动跳转到影视列表页面。url设置为name?a=b&c=d的格式，使用repath正则表达式进行匹配，使用GET.get(‘a’)来获取属性a的值。

项目的网页均使用django模板进行渲染

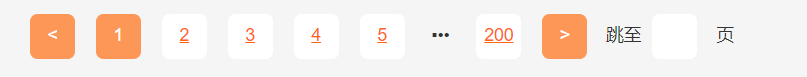
**1.导航栏**



包括搜索框和电影/演员首页跳转界面。在搜索框中输入相应的搜索字段，网站就会做相应的完全匹配，点击搜索按钮或者在搜索框中按回车将会跳转到搜索界面；点击右上角的导航可以转到电影和演员的列表页。

其中搜索框采用form实现，选择框使用select实现。为了方便起见，form的method均设置为get，同时通过为不同的控件设置相应的name实现自动跳转。按钮的type设置为submit即可实现点击按钮跳转。

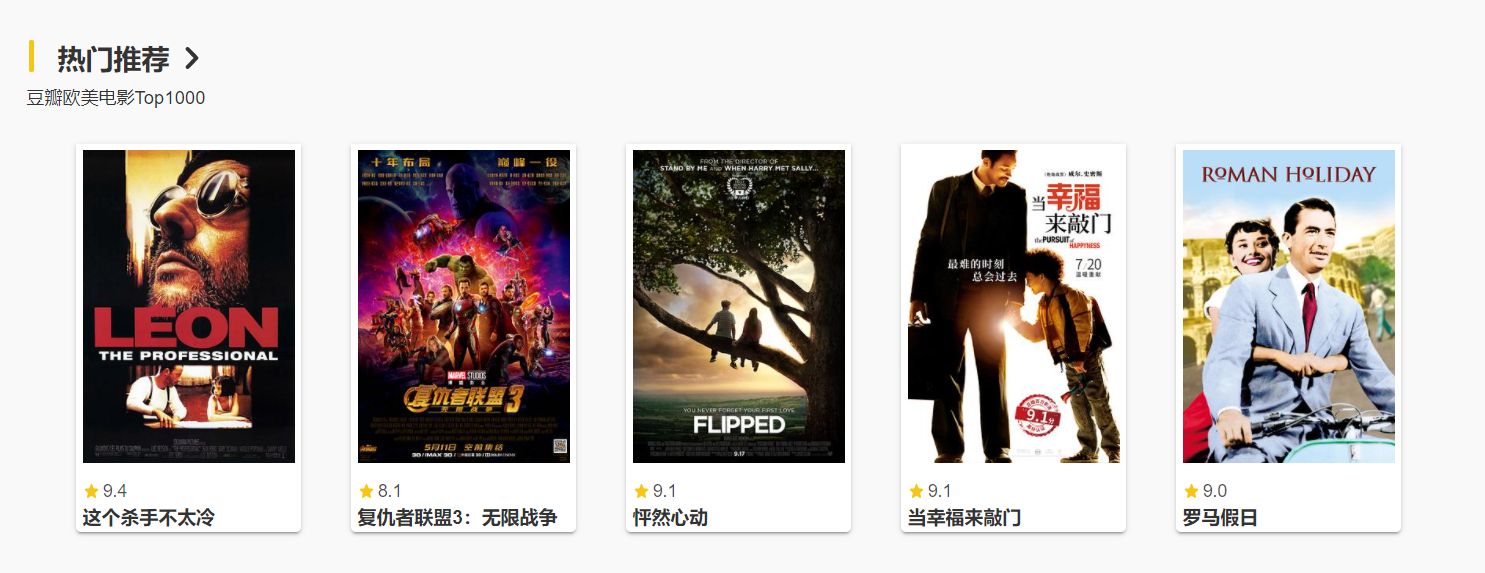
**2.页码**



页码包括上一页、下一页、该页为中心前后5页、首页、尾页、输入页码跳转按钮。

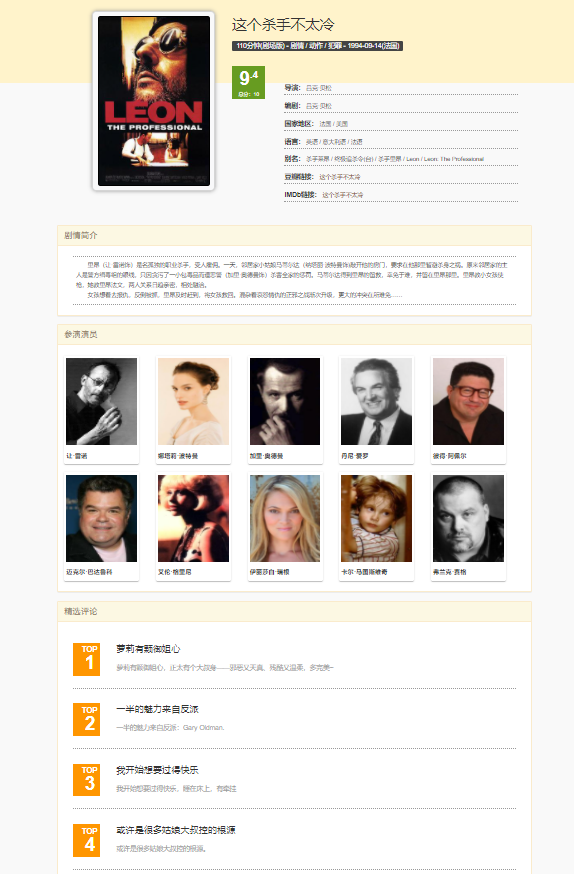
页码采用Django提供的Paginator类和Page类来进行设计，通过Paginator来自动为展示列表分页，利用Paginator.page获取的Page来读取当页的信息以及获取邻接页的信息，列表的生成通过Paginator.page\_range来获取元素。

**3.列表页（以电影为例）**



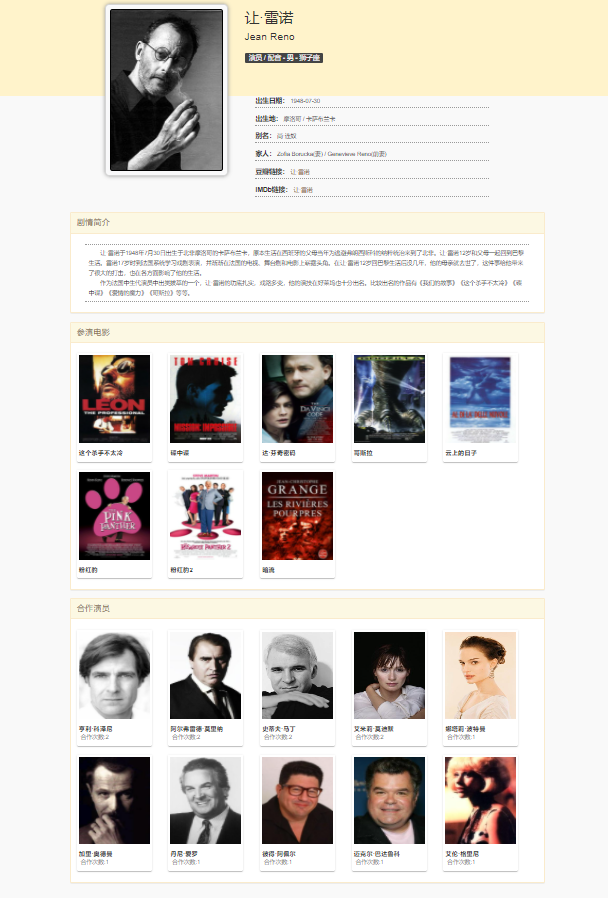
点击相应的图片或者标题就会跳转到相应的电影详细页面，每页最多展示40个。

**4.电影详细页**



该页面包括电影的资料、简介、参演演员和评论（5条）。点击参演演员的头像或者名字可以跳转到相应的演员界面。

**5.演员详细界面**



演员界面包括详细信息、简介、参演电影、合作演员（按找合作次数排序），其中合作次数通过对数据库的外键查找该演员的全部电影，再利用外键查找所有电影的所有演员，利用字典统计出现次数，获取钱10名演员。

**6.搜索页面（以电影为例）**



搜索结果展示电影名称、评分、类型、时长、主演，点击标题、图片以及演员的名字可以跳转到相应的详细界面。其中关键字会被标红，每页最多展示5条结果。

其中关键字标红使用javascript将相应的关键字keyword替换为<span class=”a”>keyword</span>的形式。搜索使用的语句为：

objects = Movie.objects.filter(Q(title\_\_contains=search\_text) | Q(actors\_\_name\_\_contains=search\_text)).distinct()，使用外键的包含查询以及去重（django涉及到连接查询和或查询会出现重复错误）。

四、爬虫数据总量以及性能

**1.数据统计**

根据数据库的统计，共爬到1000个电影网页，4602个演员网页以及对应数量的图片。





网页的查询性能：

**2.电影查询**

|  |  |
| --- | --- |
| 搜索字段 | 搜索用时 |
| 复仇 | 0.0259s |
| 雷神 | 0.028s |
| 这 | 0.0269s |
| 斯巴达 | 0.0299s |
| 西恩 | 0.0279s |

基本都可以在0.03s内完成查询

**3.演员查询**

|  |  |
| --- | --- |
| 搜索字段 | 搜索用时 |
| 汤姆 | 0.0299s |
| 查理 | 0.0299s |
| 特 | 0.0418s |
| 无限 | 0.028s |
| 太空 | 0.0269s |

基本都可以在0.05s内完成查询。

**4.评论查询**

|  |  |
| --- | --- |
| 搜索字段 | 搜索用时 |
| 评论 | 0.006s |
| 第三部 | 0.006s |
| 无聊 | 0.006s |
| 失望 | 0.008s |
| 太空 | 0.007s |

基本都可以在0.01s内完成查询。

五、提交文件说明

提交文件中loader.py为爬虫代码，其余文件为django的工程项目代码，数据库由于无法打包没有提交。