目录

[一、绪论 1](#_Toc482609403)

[（一）课题意义 1](#_Toc482609404)

[（二）研究背景和研究目的 1](#_Toc482609405)

[（三）开发环境以及关键技术介绍 2](#_Toc482609406)

[1.Centos 发行版7.3.1611 2](#_Toc482609407)

[2.Vim 2](#_Toc482609408)

[3.MYSQL 2](#_Toc482609409)

[4.网络爬虫 2](#_Toc482609410)

[5.Scrapy 2](#_Toc482609411)

[6.宽度优先搜索算法 4](#_Toc482609412)

[7.深度优先搜索算法 4](#_Toc482609413)

[二、系统分析 5](#_Toc482609414)

[（一）可行性分析 5](#_Toc482609415)

[1.技术可行性 5](#_Toc482609416)

[2.经济可行性 5](#_Toc482609417)

[（二）需求分析 5](#_Toc482609418)

[1.业务需求 5](#_Toc482609419)

[2.性能需求 5](#_Toc482609420)

[（三）结构分析 6](#_Toc482609421)

[1.系统用例图 6](#_Toc482609422)

[2.爬虫下载模块 6](#_Toc482609423)

[三、总体设计 8](#_Toc482609424)

[（一）体系设计 8](#_Toc482609425)

[（二）总体结构框架 8](#_Toc482609426)

[（三）数据库设计 8](#_Toc482609427)

[1.关系模式 9](#_Toc482609428)

[2.数据库表结构 9](#_Toc482609429)

[3.运行设计 9](#_Toc482609430)

[4.维护设计 10](#_Toc482609431)

[5.SQLalchemy 10](#_Toc482609432)

[四、详细设计 11](#_Toc482609433)

[（一）Scrapy顺序图 11](#_Toc482609434)

[（二）爬取item设计 11](#_Toc482609435)

[（三）SQLalchemy对应数据库类的设计 12](#_Toc482609436)

[（四）URL去重 12](#_Toc482609437)

[（四）反爬虫设计 13](#_Toc482609438)

[（五）爬虫爬行策略 13](#_Toc482609439)

[（六）Scrapy selectors 13](#_Toc482609440)

[五、豆瓣电影爬虫实现 15](#_Toc482609441)

[（一）爬虫实现 15](#_Toc482609442)

[（二）代理池的实现 16](#_Toc482609443)

[（三）URL的去重实现 16](#_Toc482609444)

[（四）Mysql数据库存储数据 17](#_Toc482609445)

[六、运行和测试 18](#_Toc482609446)

[（一）试验运行以及结果 18](#_Toc482609447)

[（二）实验分析 19](#_Toc482609448)

[七、总结 20](#_Toc482609449)

[参考文献 21](#_Toc482609450)

# 绪论

## （一）课题意义

当前互联网处于大数据时代，基于神经网络的深度学习技术也日益成熟，随着人工智能越来越火，使得现阶段深度学技术对数据量的需求也是响应的膨胀。从而导致现在是一个数据为王的时代， 所以目前如何快速的获取大量的数据成为了决定一个互联网企业是否能快速占领市场的标志。而作为搜索引擎、门户网站、社交平台和金融平台等拥有众多数据信息资源的平台，拥有了所谓的数据红利。可是就人工智能的初创企业而言，基于神经网络的人工智能对训练样本的需求是非常巨大的，由于缺乏相应的信息资源，因此这些初创企业从互联网上获取大量的数据成本非常巨大。所以可以通过网络爬虫从网络上批量的抓取相关所需要的数据[1]。

作为普通的网民对于信息的获取，对获取要求的精度、准确度和个性化的需求也是日益提升的，而通用搜索引擎获取信息的广度、有效性、更新速率在当前却不能满足这些用户，所以网络爬虫对于这些用户而言也是有很大需求的。

python语言作为一种动态语言用来编写网络爬虫是非常强大的，拥有很多通用的函数库，可以实现敏捷性开发一个网络爬虫系统。而网络开源爬虫框架Scrapy正是用python开发的，因此该系统的开发是基于这个网络爬虫框架的。

## （二）研究背景和研究目的

随着大数据时代的到来，基于深度学习人工智能技术的发展，人工智能可以说步入了一个新的纪元。在2011年Andrew Ng在google深度学习项目中进行的著名的“猫脸识别”实验，使用了16000个计算芯片然后接入了youtube。Youtube作为全球最大的视频网站门户，本身而言拥有非常巨大的视频图片信息。而谷歌自身虽然是搜索引擎平台，但是也拥有自己的数据中心，故这次“猫脸识别”实验虽然非常成功，可是背后是依靠庞大的训练样本数据才能获得如此的成果。由此可见数据对于人工智能而言是多么的重要。

在去年12月27日，中国电影报、人民日报手机客户端和CCTV-6声讨豆瓣电影的评价体系，对目前网络水军对影评的恶意评价以及刷分表示不满，通过该爬虫爬取热点影评，以及电影信息等数据，方便第三方对恶意差评刷分进行甄别。所以这里选择爬取豆瓣电影：

获取热点影评，方便电影评价网站对恶意差评，刷分的甄别，减少网络水军恶意营销对电影票房带来的经济损失。

若有其他极客打算进行文本语言分析的人工智能的开发，可以提供获取大量优质文本数据的途径。

## （三）开发环境以及关键技术介绍

### 1.Centos 发行版7.3.1611

Centos是红帽公司的一个免费发行版的linux系统，Centos是RHEL的克隆版本但是得不到red hat企业的有偿技术和升级服务。而Linux是一个可以自由发布的类UNIX内核实现，并且基于POSIX和UNIX，毕竟和UNIX一个体系，所以对UNIX网络编程也支持，而且支持多线程，在linux中一切皆文件，从而降低了网络编程的难度。不过这里为了节约资源，该环境没有安装图形界面。

### 2.Vim

Vim是vi的进阶版本，而现在类unix系统安装好后会内建vi文本编辑器，具有语法高亮。虽然只是一个文本编译器，可是功能强大的它不逊色任何一个代码编写工具。在linux中用vim对程序的编写非常快捷。而且通过修改配置文件，可以让python程序的语法缩进变的非常方便，而且可以通过自定义添加插件来提供自己所需求的功能。只是快捷命令较多，学习起来较为困难，可是如果利用得当，这些快捷命令可以让开发的速度成倍的提高。

### 3.MYSQL

MYSQL是一个多线程，多用户的SQL服务器，它是以C/S为结构实现的，通过mysql.sock套接字文件实现客户端和服务端的交互，其主要服务通过msqld\_safe作为一个守护进程常驻后台。并且可以支持在linux环境中工作，而且就性能和可靠性来说也是没有多大问题的，毕竟商用化很久。

MYSQL作为一个关系型数据，因为该爬虫系统爬取豆瓣信息的数据进行结构化存储，所以利用mysql关系型数据库非常的便捷。而且通过SQL语句也很方便对存储的数据进行查询、读取。

### 4.网络爬虫

网络爬虫通俗的讲就是可以在互联网上通过超链接的跳转定向或者非定向的抓取数据的应用。其专业的描述是，抓取特定网页的HTML数据，从一个或者多个的初始的URL进行下载，一般一个web页面中拥有跳转到其他页面的超链接，将这些超链接URL放入URL请求队列中，然后对每个URL进行以上的操作。

### 5.Scrapy

Scrapy，是用python开发的一个爬虫框架，但是并不负责爬虫的业务，整个框架只是控制数据流向。可以通过快速开发实现爬虫在网页中提取需要的数据。该框架可用于数据挖掘、检测和自动化测试[2]。Scrapy的特点在于，他只是一个框架并且提供了多种类型爬虫的基类如BaseSpider、CrawlSpider等通过继承对方法的覆盖然后实现自己的功能。

Scrapy结构如图1-1



图1-1 Scrapy结构图

调度器(Scheduler)：所有的请求都在这里进行一个队列的存储直到下载器会对队列的请求进行下载，然后进行出队操作。这里请求队列主要就是URL队列，所以有很多也不会对性能造成影响，而且可以通过配置文件的修改决定是FIFO还是LIFO队列实现相应的业务。

下载器(Downloader)：下载器负责从将返回的http body即html文档丢给蜘蛛去处理。

蜘蛛(Spiders)：Spider是用户编写用于从请求返回中提取item(即需要抓取的数据)，这里可以支持编写多个spider用来处理不同的业务，这里负责从页面提取数据的业务。

项目管道(Item Pipline)：负责分析、保存、过滤从蜘蛛中获取的数据，其中这些数据保存在item对象中。

中间件：Scrapy通过中间件来实现用户自定义的功能，保持对Scrapy框架的扩展。

数据流程：

首先从一个URL或者一个URL列表开始，引擎会将该URL会将下载器，让下载器对该web站点发送HTTP请求，下载之后会将HTTP的返回body交给蜘蛛进行数据的抽取，蜘蛛从该HTML文档主抽取两种数据：一种是可以跳转到其他需求页面的超链接，例如“下一页”的链接，这些东西会被传回引擎存入请求队列中 ；另一种是需要抓取的数据，这些数据通过item对象传递给项目通道，这里对数据进行详细分析、过滤或者保存[3]。而且可以在中间件中添加相应的扩展。该爬虫系统就是依靠Scrapy框架来实现主要的业务逻辑，其中需求的扩展通过中间件来添加。

Scrapy依赖于twisted网络库，twisted是一个用python语言实现的基于事件驱动的网络框架，支持很多协议。主要通过事件的触发和回调来完成业务的实现。而且Scrapy基于并发性的考虑由非阻塞(即异步)的实现[4]。

这里考虑使用Scrapy框架实现爬虫而不是python多线程实现爬虫，因为虽然python语言这里本身支持多线程编程，但是存在一个性能的问题。因为python多线程的实现依靠了一个GIL的全局锁伪代码如下

While(1)

{

Lock(&GIL);

Work();

Unlock(&GILl);

}

看起来只是多个线程抢占同一个全局锁，当IO线程等待时，可以休息让其他线程开始工作，在单核心情况下看起来没问题。但是在多个核心的情况下就会出问题了，注意这里解开全局锁和加锁之间几乎有间隔，可能存在当前线程解锁之后唤醒了其他线程，可是其他线程被唤醒之后还没来得及抢占这个锁，就又被当前线程继续抢占，而这时被唤醒的线程已经被唤醒了，在其工作时间内又没有抢占到这个锁而白白等待自己的生存时间。而线程切换的会消耗cpu资源，从而引起额外的性能的开销会导致传说中的线程颠簸(thrashing)。

所以本系统不采用python爬虫采用多线程的设计。

### 6.宽度优先搜索算法

宽度优先搜索算法，缩写BFS，是一种对图的遍历策略。该算法属于盲目搜索，因为对于爬虫来说可以覆盖尽可能多的页面，所以在该爬虫系统中用来实现豆瓣电影全网搜索的需求，这里通过宽度优先遍历获得尽可能多的URL，然后通过Scrapy自带的rules通过正则表达式将其中无关的URL过滤掉后放入到待爬取队列中等待下载。也就是说这里通过将宽度优先搜索和网页过滤技术结合使用来完成豆瓣电影全网资源的爬取。

### 7.深度优先搜索算法

深度优先搜索就是尽可能深的去搜索图。就是通过对一个html文档进行url的抓取，找到第个url之后再重复上面动作，如果没有其他的URL可供爬取时，递归回溯然后继续，如果当所有HTML都进行完没有发现URL可以提供爬取时说明搜索已经结束。而爬虫陷阱对于深度优先搜索的爬虫来说是致命的，使得爬虫陷入其中，这里通过显示爬虫爬取网页的深度来解决该问题。本爬虫系统通过深度优先搜索算法实现了爬取豆瓣电影同类电影信息的需求。

# 二、系统分析

## （一）可行性分析

该可行性是指在当前情况下，研制该爬虫系统是否必要，是否具有必备的条件。系统的可行性分析包含技术上，经济上的必要条件，同时还要进行判断在目前情况下是否可以按时按量完成该爬虫系统。而且为了避免重复造轮子的情况，需要通过研究进行可行性研究分析。

### 1.技术可行性

从目前系统的功能、各个场景的需求来考量我们目前具有的技术是否能够支持该爬虫系统的开发。现在从总体的框架的设计、构建可以目测可以满足本爬虫系统所需要的的条件。

### 2.经济可行性

本系统采用阿里云服务器搭建linux系统环境，学生特惠只需9.9元一月，其余使用框架、技术皆为开源软件，无需其他的成本投入，唯独需要开发的时间成本。因此来说该爬虫系统的经济可行性是可行的。

## （二）需求分析

### 1.业务需求

业务需求即分析该爬虫系统爬取这些信息是否合理。目前因为电商和电影届的合作导致大量水军涌入，使得豆瓣电影的评价体系的可靠性降低。所以这里通过对豆瓣电影热电影评和电影评分的爬取，可以方便第三方鉴别是否该影评为恶意评论。而且热电影评无论是易读性还是文笔都较好，故后期可以通过添加NTLK自然语言分析框架，可以将爬取的文本数据作为其他深度学习人工智能的训练样本。

### 2.性能需求

(1)可维护性：本爬虫系统开发全部使用python语言，阅读性很好。而且开发环境通过租用阿里云服务器，所有的开发修改都可以通过ssh远程连接操作，方便后续开发人的维护和更新。

(2)可扩展性：本爬虫系统开发使用Scrapy框架+mysql的结构，所以这里业务需求的扩展只需要进行中间件自定义的编写即可。故该爬虫系统的扩展性通过中间件的机制得到保证。

(3)稳定性：本爬虫系统通过代理池和禁止COOKIES的机制来降低被爬取web站点的反爬虫机制发现的几率，通过mysql数据库实现了对爬取数据的存储，保证设备掉电重启后数据仍在存在。

## （三）结构分析

### 1.系统用例图

通过初始url下载

获取数据填充item

将符合规则url存入队列

创建数据库会话

将item存入数据库

图2-1 系统用例图

由于该爬虫系统并不存在所谓的注册登录对于用户的区分，所以这里的用户便可以理解为系统的拥有者或者管理员，而图中用例则标明了该爬虫系统所做的功能，也表明了该次开发所需要重点完成的需求。本次对爬虫系统的开发完全根据该用例图的建模来完成，其中URL下载以及请求队列的建立这里都通过scrapy框架自有的功能来完成，其中数据获取、URL规则匹配和代理池的实现这里都需要自己完成。而对于数据这里会话的建立以及操作都通过SQLalchemy提供的API接口来完成。数据库中存储数据表的建立则通过直接进入mysql中通过SQL语句进行。

### 2.爬虫下载模块

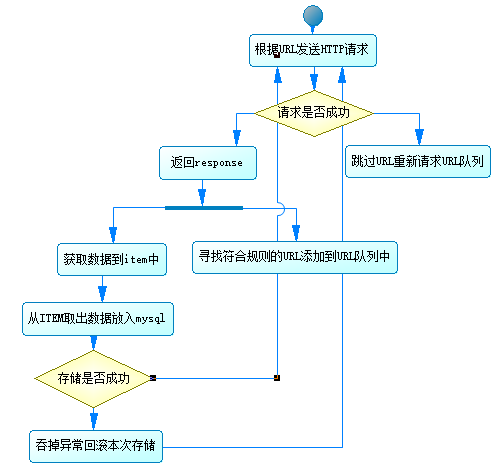


图2-2 爬虫模块活动图

从爬虫的模块活动图中可以看出，本爬虫系统所有的设计都是为了保持爬虫进程能够一直运行下去，所有可能出现异常控制流的模块，这里都用过对异常的抓取和忽视来保证。而吞下异常后URL队列重复这里通过在数据表中建立unique字段来二次保证，而数据库存储失败则通过抓取该异常，然后进行事务回滚后继续进行后续的操作。总之，爬虫模块的活动的对异常的设计就是为了保证不会打断爬虫本身业务的进行，

# 三、总体设计

## （一）体系设计

该爬虫系统总体主要分为爬取数据和保存数据，保存数据这里通过mysql数据库来做保证数据的非易失性。而爬取数据这里需要通过Scrapy的中间件来扩展URL去重和用户代理池的功能，有效URL的匹配通过rule规则的编写依靠正则表达式来实现。系统总体功能模块如下图：

代理池

下载网页

爬虫下载

爬虫系统

数据存储

网页链接查重

匹配规则链接

提取数据

电影影评存储

电影信息存储

图3-1 总体功能模块

## （二）总体结构框架

Scrapy

download

SQLalchemy

Mysql

extractData

session

insert

Info\_table

Mov\_table

图3-2 总体结构框架图

其中上层通过Scrapy来爬取url并且从中抽象出数据，中间层通过SQLalchemy通过建立会话保证和数据库的以及存储操作，底层的mysql通过两张表来保存电影数据和信息。整体结构比较稳健，不直接通过中间层访问数据库来隔离数据。虽然Scrapy这里作为顶层而且是本系统的主要业务层，但是由于SQLalchemy中间层的添加降低了顶层的扇出，保证的系统的稳定性。

## （三）数据库设计

数据库这里通过info和mov两个表来分别保存电影信息和电影影评。数据库表引擎使用InnoDB，因为该引擎支持字段自增，并且支持事务的回滚。采用UTF-8来支持中文[5]。

### 1.关系模式

(1)电影信息表(序号，电影名称，电影导演，豆瓣电影评分，电影在影评中的名称)

(2)影评信息表(序号，电影名称，影评名称，影评)

### 2.数据库表结构

表3-1 info 表格

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field | Type | Null | Key | Default | Extra |
| id | int(11) | NO | PRI | NULL | auto\_increment |
| name | varchar(200) | YES | UNI | NULL |  |
| score | varchar(10) | YES |  | NULL |  |
| atrrs | varchar(100) | YES |  | NULL |  |
| m\_name | varchar(200) | YES |  | NULL |  |

Id: 就是序号，这里采用自增长。

Name: 电影的名字，因会有的日文电影名字很长，故采取varchar(200)

Score: 豆瓣电影的分数，因是浮点数，mysql对浮点数的记录需要精度，但这里只需要小数点后一位的精度，所以这里选择用varchar(10)类型来记录

Atrrs：电影的导演，担心某些导演奇怪的名字，故这里采用varchar(100)

M\_name: 电影名字会包含英文名或者其他语言的翻译，这里通过字符串的截取来保留下前面中文电影名字的翻译，可以方便后续对mov表格的查找。

表3-2 mov 表格

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field | Type | Null | Key | Default | Extra |
| id | int(11) | NO | PRI | NULL | Auto\_increment |
| name | varchar(200) | YES |  | NULL |  |
| title | varchar(200) | YES |  | NULL |  |
| content | text | YES |  | NULL |  |

Id:就是序号，这里采用自增长。

Name: 电影的名字，理由同上。

Title：这里是影评的标题，这里担心有的影评标题会过长故采用varchar(200)。

Content: 影评，主要存贮的文本，也不会通过影评建立索引，所以这里采用text。

### 3.运行设计

实际运行环境：

操作系统：centos 发行版7.3.1611

硬件：处理器 Intel(R) Xeon(R) CPU E5-2682 v4 @ 2.50GHz 1核心

内存：2GB

网络：带宽1兆

该爬虫爬取较少数据时，比如top250电影信息和热电影评时，可以通过直接在命令行中运行scrapy crawl movie，可是过在运行全网爬行数据时，如果还在运行时还要占据shell，那当前终端没有办法断开，所以通过命令nohup scrapy crawl movie & 来将爬虫的进程交接给pid 1进程来接管，从而脱离了该终端。

体系结构：Scrapy

DBMS：MYSQL(数据库引擎InnoDB)

### 4.维护设计

由于服务器在公网上，所以可随时连接至服务器检查环境是否出现问题。对于爬虫对数据的爬取，进行日志文件的写入，包括查bug、对爬虫进度的查看都可以通过查看日志文件来实现。对于该爬虫来说，维护工作主要依靠日志文件。

### 5.SQLalchemy

用python语言实现，使用ORM技术：Object-Relational Mapping，把关系数据库的表结构映射到对象上[6]，将数据库中的表与面向对象语言中的类建立了一种对应关系。相当于通过ORM帮我们把对象转化成SQL然后跟数据库交互，可以不许用将SQL语句写死在代码中，防止SQL语句的注入攻击，而且代码跨平台，可以对接多种数据库。使得开发人员并不需要对具体的SQL语句进行学习，而且更换数据库也不需要大幅度的变动代码。在当前爬虫系统中，对接mysql数据库，倘若以后因为扩展需求或者平台需求需要更换数据库，其逻辑语句并不需要做很大的变动，提高的扩展性。

SQLalchemy拥有两种对数据库表格建立映射的初始化方式，一种是使用table定义表，其仅仅是与数据库做了关联，如果想以OO的思想对其进行操作，需要设计对应的类，并且通过mapper(from sqlalchemy.orm) 对其做关联映射，是通过类——映射——数据库表，来实现所谓的ORM思想的。第二种是通过类声明数据库表、字段，相当于直接实现了类&映射——数据库表。在本爬虫系统中我们选择使用第二种定义表的方式。SQLalchemy对数据库所有的操作都是通过create\_session()函数创建的会话来完成的，也可以通过该会话完成相应的数据库的事务操作。

# 四、详细设计

## （一）Scrapy顺序图

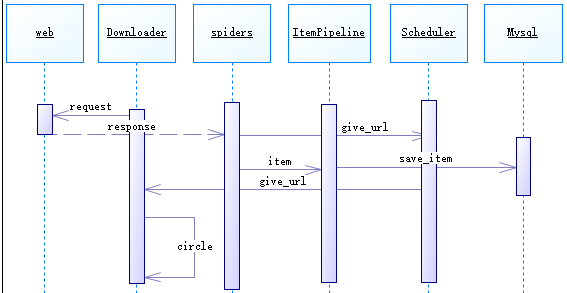


图4-1 Scrapy工作时序图

所以这里需要在调度器将URL丢给下载器下载之前来添加代理池中的用户代理，在spiders解析出URL存放到URL队列时进行URL的查重，这里通过中间件来扩展。

## （二）爬取item设计

由于使用scrapy框架所以这里首先定义爬虫要抓取的数据项目，这里类似与ORM思想。

定义了两个item MoiveContentItem 与 MoiveItem。前者是用来保存影评信息，影评标题，以及电影名字，这里是用来保存电影信息，电影导演，电影豆瓣评分[7]。

class MoiveContentItem(scrapy.Item):

# define the fields for your item here like:

# name = scrapy.Field()

name = scrapy.Field()

title = scrapy.Field()

content = scrapy.Field()

m\_name = scrapy.Field()

class MoiveItem(scrapy.Item):

name = scrapy.Field()

atrrs = scrapy.Field()

score = scrapy.Field()

这里将这两个项目和成同一个爬虫来写，并没有分成两个爬虫，是因为这两个信息的网页地图很多都是重复的，毕竟影评是依附于电影的信息的。如果分成两个爬虫来写，会导致很多url爬取都是重复的。反而影响性能。

## （三）SQLalchemy对应数据库类的设计

这里通过对数据库表字段通过类的声明直接完成了类&映射——数据库表。

class Info(Base):

\_\_tablename\_\_ = 'info'

id = Column(Integer, primary\_key=True)

name = Column(String(200))

score = Column(String(10))

atrrs = Column(String(50))

class Mov(Base):

\_\_tablename\_\_ = 'mov'

id = Column(Integer, primary\_key=True)

name = Column(String(200))

title = Column(String(200))

content = Column(TEXT)

## （四）URL去重

在Scrapy框架中也自己带有链接去重的功能，可使它的去重算法是通过对request的16位hash指纹来判断的，他的去重是通过对(method+url+body+header)的请求用一个哈希算法，所以实际上在我们的业务中能够去重的比例有待考究。而作为爬虫来说URL去重的意义是非常重大的。

1.如果URL不去重，URL待爬取队列就会被占用很多重复的URL，导致会产生会数据资源的重复获取，这是对内存对网络都会产生巨大的浪费。

2.URL去重还有一个原因是避免爬虫陷入所谓的爬虫陷阱。在豆瓣电影的页面中，同类电影之间会存在，A电影的页面中有B，而B电影的页面中有A的链接，这样就会导致爬虫在A于B之间来回的爬取。而且全网爬行的话，会产生很多这样的环状连接。从而导致尽管爬虫爬取URL的速率很快，可是存储至数据库中有效数据却很少，这种环状爬取如果不解决的话，对内存的损耗是非常巨大的，可以说不解决这种环状爬取，对内存的占用并不是能通过加物理内存来解决的。

所以这里对于url的去重，是通过继承RFPDuperFilter类进行的，仅仅根据URL本身的相同与否进行去重，通过Scrapy中间件的扩展当获取到新的URL时[8]，再插入URL队列时，如果待爬队列里已经有了就不进行插入。从而实现了URL边爬边去重。

## （四）反爬虫设计

(1)爬虫实现登录可以通过保存浏览器手工登录的cookies来保证登录的状态，可是维持着cookies会很容易让网页辨识出这是同一个爬虫，便会禁止掉当前爬虫，所以该爬虫系统选择禁用cookies防止用cookies识别爬虫轨迹的网站发觉。

(2)如果只是增加爬虫的间隔，也很容易被禁止掉，这里仅仅通过设置爬虫爬取url的间隔，而且还将时间间隔随机化，防止被禁止掉的几率。

(3)代理池设计，因为一些爬虫由于设计的不合理会曾经会对某些网站进行疯狂的爬取导致网站服务器压力暴增甚至宕机，所以一些网站禁止掉使用python默认代理的爬虫程序。而且使用自定义的用户代理代替默认的代理可以更好的模拟自然用户浏览该网站的行为。而且如果一个爬虫爬取一个网站虽然会给该站点带来额外的流量，但是同样的也对该web服务器带来了负载，所以一个爬虫如果某个时间段爬取该web站点频率过高，就存在被禁止的风险[9]。而通过代理池，让爬虫在通过URL发送相应的http请求时，在该请求头部随机的从代理池中找出一个代理然后构造其中，就会让同时间段某个用户代理访问web站点的几率降低，从而减少了被禁止的风险。

## （五）爬虫爬行策略

实现同类型电影的电影信息和影评的爬取采用深度遍历的方式。

实现豆瓣电影全网电影信息和影评的爬取采用宽度遍历的方式。

## （六）Scrapy selectors

因为网络爬虫只是从web站点下载URL对应的html文档，而从html文档中选择所需要的数据，这里就需要selector选择器发挥作用，不同网页站点的html都会有自己的结构或者规则，而从中批量的提取数据并且能够保证通用性，正是selector选择器的用武之地。

Scrapy Selectors是Scrapy内置的一种基于XPath和CSS表达式机制的selector选择器，安装好Scrapy框架的环境既可使用，无需安装，上手快，简单粗暴。所以该爬虫系统选取它作为selector选择器来爬取数据。而不选取其他选择器的原因如下。目前从html文档爬虫获取数据有这么几种方式，Lxml，beautiful soup，正则表达式，Scrapy selectors。

表4-1 selector选择器对比

|  |  |  |  |
| --- | --- | --- | --- |
| 抓取方式 | 性能 | 使用难度 | 安装难度 |
| 正则表达式 | 快 | 困难 | 简单(内置模块) |
| Beautiful Soup | 慢 | 简单 | 简单(纯python) |
| Lxml | 快 | 简单 | 相对困难(依赖较多) |
| [Scrapy Selectors](http://scrapy-chs.readthedocs.io/zh_CN/latest/topics/selectors.html" \l "topics-selectors) | 快 | 简单 | Scrapy内置 |

正则表达式，依赖较少，性能很好，但是在html中匹配数据过于依赖网页的具体结构，倘若网页有变动就会比较尴尬，不太适用于全网的爬取。对于爬取大批量数据，多数据正则表达式并不适合，因为影评相对其他信息来说作为毕竟作为一个文本，而且每个人有自己的格式，所以这里不选取用正则表达式。

Beautiful soup是一个从html文档或者XML文件中提取所需要数据的python库，安装简单，使用方便，只是性能相比其他选择器来说稍差，因为他的本质其实是将整个的HTML文档构造成一个对象，然后在该对象中调用相应的查找方法对需要的内容进行匹配，在对于豆瓣电影全网信息爬取的场景中，虽然满足该场景的需求，但是性能相比较差，会影响爬取速度。

LXML性能很好，但是需要依赖的库很多，scrapy本身依赖twisted库，安装就比较麻烦，在已经安装好Scrapy的环境下进行LXML的安装，如果依赖的库有问题，很容易把环境装挂掉[10]，对其他模块造成影响，可能会造成重新安装Scrapy，所以不选择该匹配方式。

所以相比其他Selector选择器来说，本爬虫由于使用Scrapy框架，所以直接使用Scrapy Selectors。

# 五、豆瓣电影爬虫实现

## （一）爬虫实现

通过scrapy startproject douban创建douban爬虫的scrapy项目，当前爬虫项目目录树见下图：

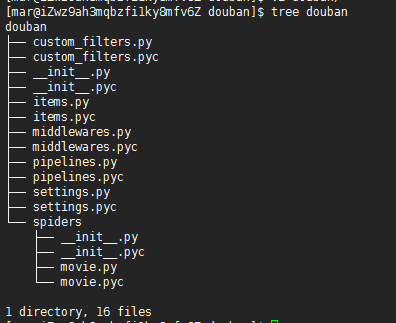


图5-1 爬虫项目目录树

当前爬虫具体的业务实现在spiders目录下的movie.py,DoubanSpider类通过继承了CrawlSpider来实现爬取的业务，其中属性name表明了该spider的名字，start\_urls就是起始url，可单个可多个，多个的形式写成一个列表即可，其中匹配记下来爬取url规则的定义如下在类DoubanSpider(CrawlSpider)中。

allowed\_domains = ['movie.douban.com']

rules = [

#Rule(LinkExtractor(allow=r"/top250\?start=")),

Rule(LinkExtractor(allow=r"/subject/\d+/reviews$")),

Rule(LinkExtractor(allow=r"/subject/\d+/\?from=subject-page$"),\

callback="movie\_item", follow=True),

Rule(LinkExtractor(allow=r"/subject/\d+/$"),\

callback="movie\_item", follow=True),

Rule(LinkExtractor(allow=r"/review/\d+/$"),\

callback="parse\_item", follow=False),

#Rule(LinkExtractor(allow=r"/subject/\d+/reviews\?start=\d+$")),

]

Allowed\_domains保证了爬取的页面不会跳出豆瓣电影。

其中注释掉的#Rule(LinkExtractor(allow=r"/top250\?start="))[11]是用来匹配豆瓣电影中top250的信息和影评的。

#Rule(LinkExtractor(allow=r"/subject/\d+/reviews\?start=\d+$"))是用来抓取当前电影中的每一条评论，不仅仅限于热点评论。

在当前规则中，如果满足相应的抓取数据的规则便会出发回调movie\_item与parse\_item，分别爬取电影信息和电影影评。

## （二）代理池的实现

将多个代理以python列表的形式写入到该项目的配置文件中，然后在middlewares.py通过对RandomUserAgent类的实现来完成对用户代理的随即提取，然后将其构造在http请求的头部以便于接下来下载器对URL的下载。

class RandomUserAgent(object):

def \_\_init\_\_(self, agents):

self.agents = agents

@classmethod

def from\_crawler(cls, crawler):

return cls(crawler.settings.getlist('USER\_AGENTS'))

def process\_request(self, request, spider):

request.headers.setdefault('User-Agent', random.choice(self.agents))

## （三）URL的去重实现

Scrapy自带URL去重处理，只不过是将生成的request请求进行16位hash后进行去重，这里我们需要进行URL本身的去重处理，所以这里需要自己拟写中间件custom\_filter.py，然后通过集成RFPDupeFilter类来实现[12]。

实现代码如下：

class SeenURLFilter(RFPDupeFilter):

def \_\_init\_\_(self, path=None):

self.urls\_seen = set()

RFPDupeFilter.\_\_init\_\_(self, path)

def request\_seen(self, request):

if request.url in self.urls\_seen:

return True

else:

self.urls\_seen.add(request.url)

## （四）Mysql数据库存储数据

该爬虫系统爬取的豆瓣影评和电影信息，为结构化的数据，故这里采用mysql关系型数据库进行存储，而连接数据库这里采用了SQLalchemy以实现ORM思想，使存储的过程更加的方便快捷，支持跨平台，无视数据库的差异，而且能够防止SQL语句注入的问题。

这里连接数据库，包括对数据库对象的映射都在piplines.py中实现。由于info表中对name字段属性为unique，所以如果碰到重名的电影名字插入会报错，由于采取的是对数据库的事务操作，所以这里通过session.rollback()进行一个事务回滚的操作，并且对报错进行日志的写入。

数据库存储信息如下图：（部分数据）



图5-2 info表中前10个存储的信息

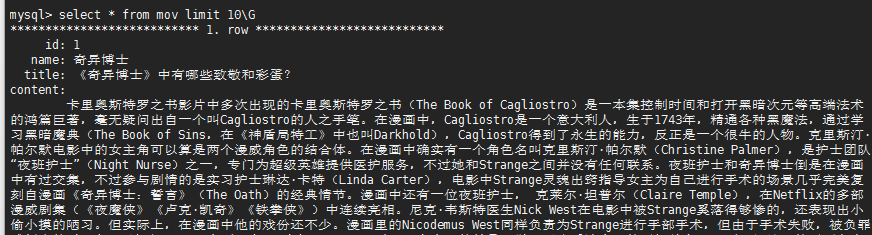


图5-3 mov表中部分储存的信息

# 六、运行和测试

## （一）试验运行以及结果

1.top250热点电影爬取情况

Top250电影的信息的获取情况如下：

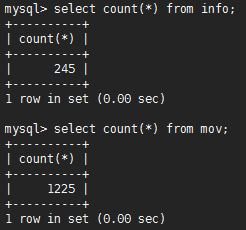


图6-1 top250爬取信息条目

这里只获取都245条记录，通过检查发现是豆瓣电影top250热度电影的网页中有5个电影的链接点开之后就是404的返回值，也就是说是本身网页的问题，爬虫的业务并没有什么不对，也没有出现丢数据的情况。而且对热点影评的检查也没有出现丢失的情况。

表6-1 爬取速率折线图

爬取速率查看日志16:41开始爬取一直到5:27爬取结束，总共爬取了1470条数据，1小时1800条数据还是可以接受的，因为这里设计每个url之间爬取的等待时间是1.5s，所以这里理论最多一小时能爬取2400条算其中对数据的处理时间再加上这个延时设置了随即的波动，故这里也和设计的理论速率相符合。

2.全网爬取情况

而全网爬取的数据目前已经爬取了8个h了，全网为了不被禁止这里设置延时为3s，爬取的影评和电影信息的数据量分别如下图：

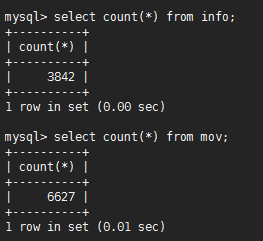


图6-2全网爬取信息条目

也就是平均1min爬取了21.8条数据，符合上面数据爬取的速率，故全网爬取速率也是符合要求的。

通过深度优先遍历可以实现同类电影的匹配，这里通过限制爬取的层数来决定爬取的精度，层数越少，爬取信息越少，但是精度越高。

## （二）实验分析

此次实验是通过同类型电影信息的爬取和全网电影信息爬取来做对比的，发现速率前后的速率是符合的，但是发现主要影响速率的部分是url爬取的间隔时间，可是这里如果时延的时间太短，就会被禁止掉，就是说时间越短，被禁止掉的概率越高，所以我这里选择了细水长流。而在设计中已经分析了这里并不能采用两个爬虫，反而会引起不必要的开销。所以这里不仅仅应该采用用户代理，还应该加上IP代理，然而IP代理的多半是收费的，而免费的IP代理速率难以得到保证，如果出现ping不通的情况，会引起数据的漏抓。如果这里采用商用的IP代理，减少url间的时延，能一定的优化目前的性能，但是ip代理的切换请求这里还是有一定的性能损失的。就目前而言1兆的带宽没有什么问题，就是直接用的本机IP，如果被网站封IP就会比较尴尬，会导致接下来的爬取没有办法进行。所以目前该爬虫为了稳定性而牺牲了部分性能，不过如果添加IP池的话，性能会得到一个提升。

# 七、总结

基于Python的Scrapy框架设计并实现的抓取豆瓣电影的爬虫，从技术上提供了提供了获取文本数据的方法。其主要特点：

1.使用方便，只需要提供一个电影或者多个电影的url就可以爬取同类电影的信息、影评。

2.规格化存储在mysql中，方便他人的查询，搜索。

3.也支持全网的电影信息的爬取，通过宽度优先遍历来实现，通过修改配置文件即可。

该爬虫通过对url进行去重，还通过给mysql的name字段设置unique属性，通过双重来保证获取到的电影信息不会重复。保证数据的信息不会发生重复。该爬虫系统后续可以去对接NTLK自然语言分析，然后和豆瓣电影评分作为对比来遍历全部影评来删除恶意的差评。

# 参考文献

[1] Lawson R 用Python写网络爬虫[M]李斌 北京：人民邮电出版社 2016.

[2] 于娟,刘强. 主题网络爬虫研究综述[J]. 计算机工程与科学,2015,(02):231-237.

[3] 吴剑兰 基于Python的新浪微博爬虫研究[J]无线互联科技 江苏2015(3):93-95.

[4] 陈琳,任芳. 基于Python的新浪微博数据爬虫程序设计[J]. 信息系统工程,2016,(09):97-99

[5] 孔雪娜,孙红. 中文微博文本采集与预处理综述[J]. 软件导刊,2017,(02):186-189.

[6] 夏火松,李保国. 基于Python的动态网页评价爬虫算法[J]. 软件工程,2016,(02):43-46.

[7] 杨华权. 论爬虫协议对互联网竞争关系的影响[J]. 知识产权,2014,(01):12-21.

[8] 袁威,薛安荣,周小梅. 基于Nutch的分布式爬虫的优化研究[J]. 无线通信技术,2014,(03):44-47+52.

[9] 史宝明,贺元香,吴崇正. 主题搜索引擎中爬虫搜索策略的研究[J]. 计算机工程与应用,2014,(02):116-119+128.

[10]鲁继文. 基于Scrapy的论文引用爬虫的设计与实现[J]. 现代计算机(专业版),2017,(09):131-133.

[11]黄聪,李格人,罗楚. 大数据时代下爬虫技术的兴起[J]. 计算机光盘软件与应用,2013,(17):79-80+83.

[12]赵本本,殷旭东,王伟. 基于Scrapy的GitHub数据爬虫[J]. 电子技术与软件工程,2016,(06):199-202.