

**本科实验报告**

|  |  |
| --- | --- |
| 课程名称： | B/S体系软件设计 |
| 实验名称： | 商品比价网站 |
| 姓 名： | 周玥儿 |
| 学 院： | 计算机学院 |
| 系： | 计算机科学与技术 |
| 专 业： | 计算机科学与技术 |
| 学 号： | 3220102179 |
| 指导教师： | 胡晓军 |

2024年 12月 23 日

**目录**

[实验记录 3](#_Toc185843845)

[用户注册与登录 3](#_Toc185843846)

[用户搜索 4](#_Toc185843847)

[价格提醒 8](#_Toc185843848)

[遇到的困难 15](#_Toc185843849)

[跨域问题 15](#_Toc185843850)

[页面设计 15](#_Toc185843851)

[flask 和 celery 模块化后循环引入的问题 15](#_Toc185843852)

[短信发送 17](#_Toc185843853)

[try...catch无法捕捉promise.reject异常 17](#_Toc185843854)

[京东爬虫失效 17](#_Toc185843855)

[心得体会 19](#_Toc185843856)

[需求分析 19](#_Toc185843857)

[前端开发 20](#_Toc185843858)

[后端开发 21](#_Toc185843859)

[docker部署 22](#_Toc185843860)

[存在的不足 22](#_Toc185843861)

[收获 23](#_Toc185843862)

[总结 23](#_Toc185843863)

## 实验记录

### 用户注册与登录

使用令牌进行用户状态管理。令牌是一种无状态的用户认证方式，它通常采用JSON Web Token（JWT）格式。这种机制将用户认证信息直接嵌入令牌中，从而无需服务器存储会话信息。

#### 登陆流程

1. **用户登录**：用户通过登录接口提交用户名和密码。
2. **生成Token**：后端验证用户身份成功后，生成一个令牌。令牌通常包含用户信息、有效期、签名等。
3. **返回Token**：令牌返回给客户端，由客户端（如浏览器或移动应用）保存在localStorage中。

#### 后续请求

1. **请求携带Token**：客户端每次请求时，将Token放在HTTP请求头（Authorization: Bearer <token>）中发送到服务器。
2. **Token验证**：后端验证Token的签名和有效期，确认用户身份。
3. **Token失效处理**：当Token过期后，用户需要重新登录以获取新的令牌，或通过刷新Token机制（如Refresh Token）来延长会话。

// 登陆的信息接口  
export interface LoginInfo {  
 uid: string,  
 username: string,  
 password: string,  
 email: string,  
 avatar: string,  
 isLogin: boolean,  
}  
  
// 用户登陆时发送的请求以及收到的回应信息接口  
export interface LoginData{  
 name: string;  
 password: string;  
}  
  
export interface LoginResponse{  
 token: string;  
}  
  
export interface RegisterInfo{  
 name: string;  
 password: string;  
 email: string;  
}

### 用户搜索

前端通过搜索栏输入keyword后按下搜索按钮发起搜索请求，请求被封装成SearchPramas的形式，包含关键词、结果排序、分页数、分类、筛选等条件。

后端接收请求后，进行搜索。

* 首先调用创建好的爬虫对象更新搜索关键字，然后从网站抓取商品数据获取商品信息列表。
* 检查每个商品的SKU是否已经存在于数据库中。如果存在，则更新商品的价格信息；如果不存在，则添加新商品进数据库，在添加。
* 如果在爬虫抓取数据过程中发生异常（比如当前时间段网站繁忙），就从数据库中查找匹配关键字的商品。

#### 全文搜索

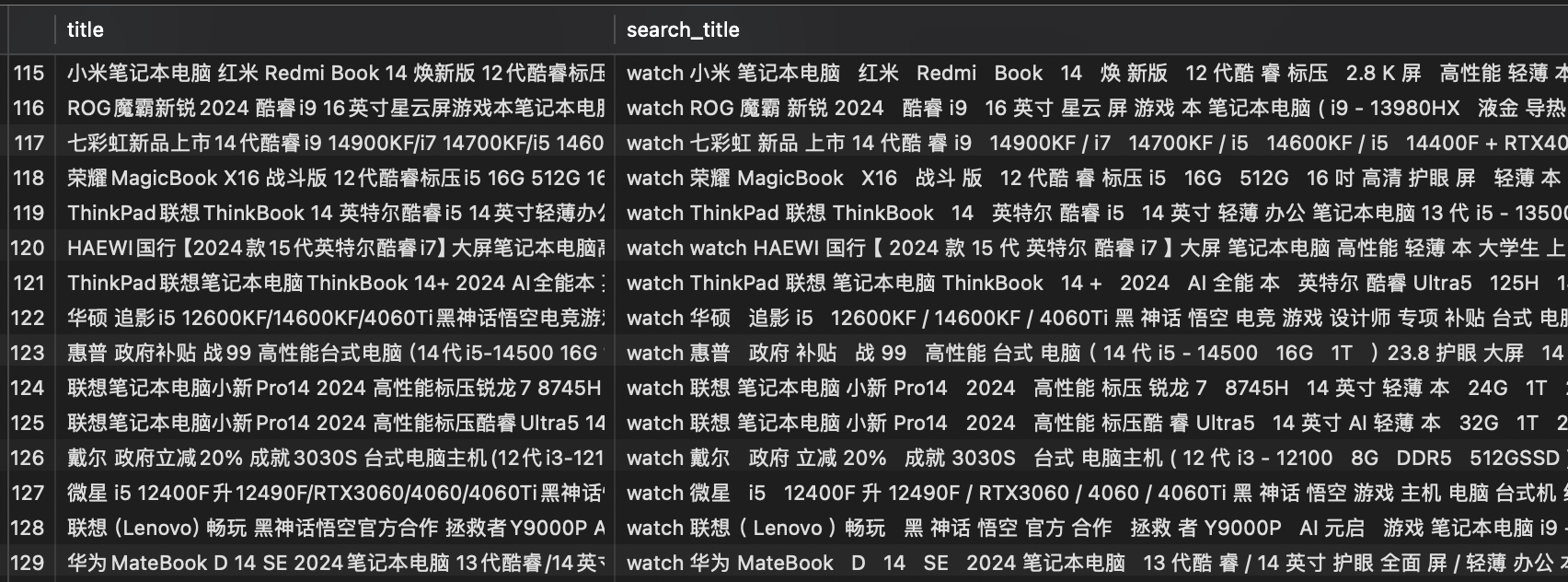
##### 分词处理

mysql全文检索是以词为基础的。MySQL默认的分词实现把**所有非字母和数字的特殊符号**都看作分词符。为了实现关键词的全文搜索，在加入商品时首先要对商品标题进行**分词处理**。

这里使用了jieba，并且以空格作为示意mysql分词的提示。

def segment\_text(text):  
 return ' '.join(jieba.cut(text))

在数据库中的存储类似于。



##### 关联优化

在mysql中有两种模式的全文搜索可选：

* **NATURAL LANGUAGE MODE**：按相关性排序，基于词频，适合一般的全文搜索。
* **BOOLEAN MODE**：可以使用布尔运算符来精确控制查询，适合需要更复杂条件的查询。

为了优化搜索结果的相关性，每次搜索后，在搜索索引中加入查询关键词再更新/加入数据库，以提高搜索结果的关联度。

search\_title = (keyword+' '+segment\_text(result['title']))[:255]

全文搜索操作通过封装fulltextsearch 函数实现

def fulltextsearch(keyword):  
 query = text("SELECT \* FROM items WHERE MATCH(search\_title) AGAINST(:keyword IN NATURAL LANGUAGE MODE)")  
 result = db.session.execute(query, {'keyword': keyword})  
 return result.fetchall()

由于Flask-SQLAlchemy不支持创建全文索引，需要在初始化数据库的时候手动执行mysql语句创建索引

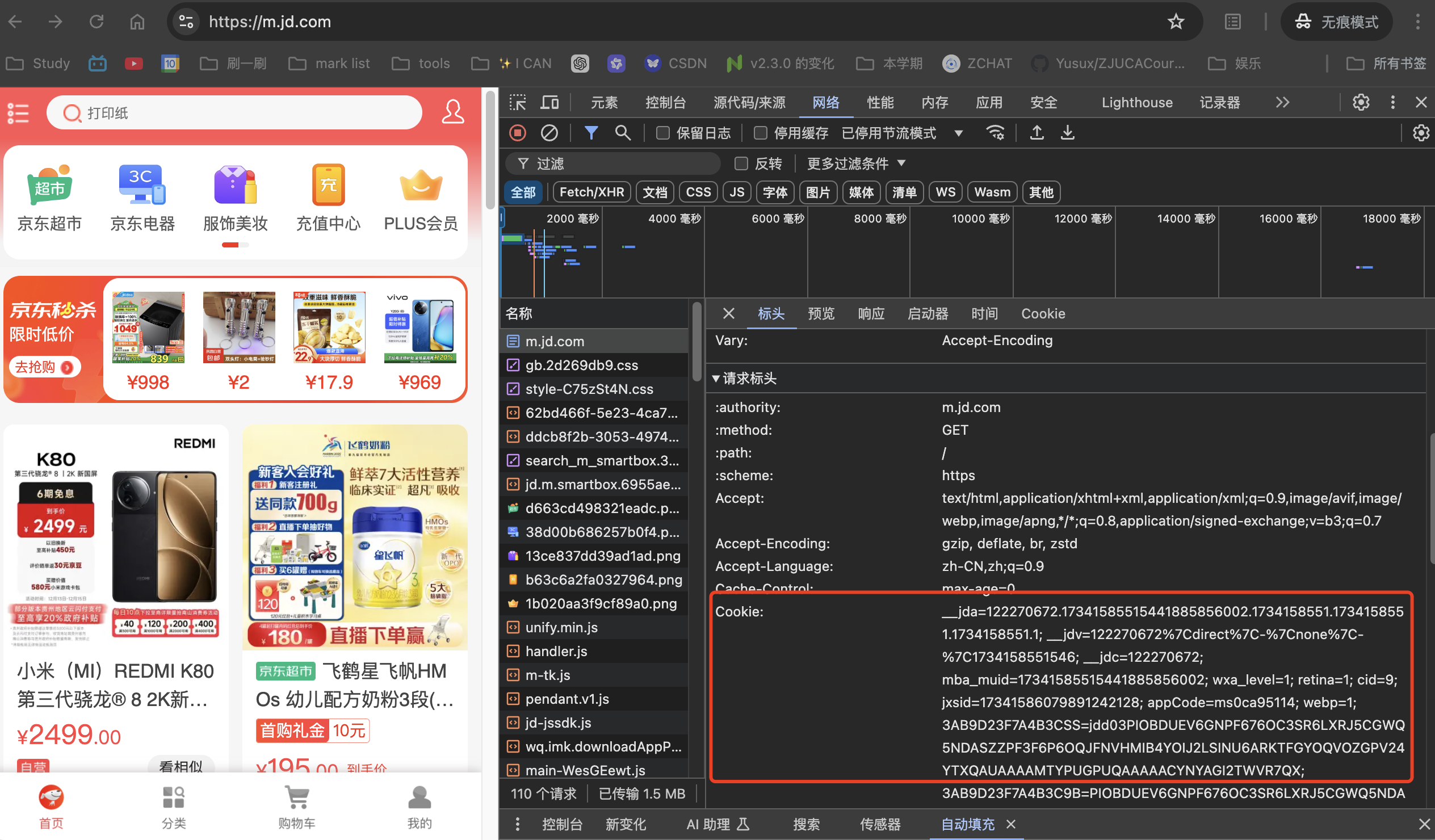
# app/\_\_init\_\_.py  
  
def db\_init(app):  
 with app.app\_context():  
 db.drop\_all()  
 db.create\_all() # 数据库初始化  
 db.session.execute(text('CREATE FULLTEXT INDEX idx\_title ON items(search\_title);'))  
 print('Database initialized')

#### 爬虫处理

爬虫有两种方案：

* 第一种方式是使用request模拟请求，并使用bs4解析respond得到数据。
* 第二种是使用selenium和无头浏览器，selenium自动化操作无头浏览器，由无头浏览器实现请求，对得到的数据进行解析。

##### 京东

* 京东的登录通常需要模拟用户输入用户名、密码以及验证码。可以手动登录京东，获取登录后 cookie，然后将 cookie 传递给爬虫使用。**使用程序自动登录**（例如使用 requests 或 selenium）：模拟登录过程，自动获取 cookie。但自动登录涉及处理验证码，这需要复杂的机制，可以用 selenium 实现模拟登录。
* 根据用户提供的关键词，构建搜索 URL 并请求搜索页面。京东的搜索链接通常包含 keyword（关键词）和 page（页数）参数。  
  
* keyword = "手机"  
  start\_url = 'https://search.jd.com/Search?keyword=' + keyword + '&enc=utf-8&wq=' + keyword  
  print("搜索链接：", start\_url)
* 使用 request 向搜索链接发送请求，需要将 cookie 添加到请求头中，以确保爬虫模拟登录后的状态。
* response = requests.get(start\_url, headers={  
   'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.149 Safari/537.36',  
   'cookie': cookie # 使用登录后的cookie  
  })  
  response.encoding = 'utf-8'
* 使用 BeautifulSoup 解析获取的 HTML 内容，以便从中提取需要的商品信息。在解析后的页面中，定位到包含商品信息的标签，例如商品的 sku、名称、价格 等。通过选择器来定位到包含商品数据的 HTML 元素。 京东的商品信息通常位于 #J\_goodsList > ul 标签下，每个商品的信息会在 li.gl-item 标签中。通过 soup.select() 或 soup.find\_all() 方法，定位到所有商品元素。
* # 使用 BeautifulSoup 解析页面内容  
  soup = BeautifulSoup(response.text, 'html.parser')  
   # 定位到商品列表  
  goods\_list = soup.select('#J\_goodsList > ul > li.gl-item')

### 价格提醒

#### Celery检查价格

通过设置一个celery队列 + reddis，来周期性地执行价格更新任务，并且根据更新的价格、提醒的设置决定是否要给用户发送价格提醒。

@celery.task(name='alert')  
def check\_price():  
 results = []  
 with db.session() as session: # 使用 Flask 的 SQLAlchemy  
 alerts = session.query(PriceAlert).all() # 查询所有 alertlist 条目  
 for alert in alerts:  
 if alert.enable is False:  
 continue  
 current\_price = get\_item\_details(alert.item\_id)['current\_price'] # 调用爬取价格的逻辑  
 if current\_price <= alert.target\_price:  
 if alert.notification\_method == 'email':  
 notify\_result = email\_notify(alert.user\_id, alert.item\_id, current\_price)  
 elif alert.notification\_method == 'sms':  
 notify\_result = sms\_notify(alert.user\_id, alert.item\_id, current\_price)  
 results.append({  
 ...  
 })  
 return results

每次使用时需要启动Reddis服务器、Celery任务和周期性Beat任务

#### 邮件发送

使用 Flask 为用户发送邮件提醒可以通过 Flask-Mail 插件实现。在 Flask 应用中配置邮件服务器的参数，SMTP 地址、端口和认证信息。这里我使用的是网易163邮箱提供的服务。



记录相关信息到配置中

MAIL\_SERVER="smtp.163.com"  
MAIL\_PORT=465  
MAIL\_USE\_SSL=True  
MAIL\_USERNAME="MyPrice2025@163.com" # 发送邮箱  
MAIL\_PASSWORD="\*\*\*\*" # 客户端授权码

在app的初始化时加入Mail功能，在需要发送邮件的场合里调用flask相关函数即可。

def email\_notify(user\_id, item\_id, price):  
 title = "[My Price Notification] 您收藏的商品价格下降啦！"  
 email = get\_user\_email(user\_id)  
 product = get\_item\_details(item\_id)  
 alert = query\_alert(user\_id=user\_id, item\_id=item\_id)  
 content = f"您收藏的商品 {product['title']} 价格已经降到 {price} 元！您预期的价格曾经是 {alert.target\_price} 元。\n 点击链接查看详情：{product['link']}"  
 print(f"Sending notification to {email} for product {item\_id} at price {price}, title: {product['title']}, content: {content}")  
 message = Message(title, recipients=[email], sender=Config.MAIL\_USERNAME, body=content)  
 with current\_app.app\_context():  
 mail = Mail()  
 mail.send(message)  
 # 这里写通知逻辑，例如发送邮件或短信  
 alert.enable = False # 关闭提醒  
 add\_alert\_history({  
 'alert\_id': alert.id,  
 'price\_after': price  
 })  
 print(f"User {user\_id} notified for product {item\_id} at price {price}.")  
 return True



#### 短信发送

使用[短信宝](https://console.smsbao.com/" \l "/general/log/mt)提供的服务，从短信服务提供商的管理界面获取必要的API密钥、账户ID等信息。使用的API接口为

# coding=utf-8  
import urllib  
import urllib.request  
import hashlib  
   
def md5(str):  
 import hashlib  
 m = hashlib.md5()  
 m.update(str.encode("utf8"))  
 return m.hexdigest()  
   
statusStr = {  
 ...  
}  
   
smsapi = "http://api.smsbao.com/"  
# 短信平台账号  
user = '\*\*\*'  
# 短信平台密码  
password = md5('\*\*\*\*')  
# 要发送的短信内容  
content = '短信内容'  
# 要发送短信的手机号码  
phone = '\*\*\*\*\*'  
   
data = urllib.parse.urlencode({'u': user, 'p': password, 'm': phone, 'c': content})  
send\_url = smsapi + 'sms?' + data  
response = urllib.request.urlopen(send\_url)  
the\_page = response.read().decode('utf-8')  
print (statusStr[the\_page])

在celery task中调用提醒函数即可

def sms\_notify(user\_id, item\_id, price):  
 product = get\_item\_details(item\_id)  
 alert = query\_alert(user\_id=user\_id, item\_id=item\_id)  
 if len(product['title']) > 15:  
 product['title'] = product['title'][:15] + '...'  
 content = f"【云比价】商品{product['title']}价格当前为 {price} 元！您的预期价格为 {alert.target\_price} 元。"  
 print(f"Sending SMS notification to user {user\_id} for product {item\_id} at price {price}, content: {content}")  
  
 # 获取用户的手机号码  
 phone\_number = get\_user\_phone(user\_id)  
 if not phone\_number:  
 print(f"User {user\_id} has no phone number, skip sending SMS.")  
 return False  
 # 使用 Twilio 发送短信  
 data = urllib.parse.urlencode({'u': Config.SMS\_USER, 'p': Config.SMS\_PASSWORD, 'm': phone\_number, 'c': content})  
 send\_url = Config.SMSAPI + 'sms?' + data  
 response = urllib.request.urlopen(send\_url)  
 the\_page = response.read().decode('utf-8')  
 print (Config.SMS\_STATUS[the\_page])  
 alert.enable = False # 关闭提醒  
 add\_alert\_history({  
 'alert\_id': alert.id,  
 'price\_after': price  
 })  
 print(f"User {user\_id} notified for product {item\_id} at price {price}.")  
 return True



## 遇到的困难

### 跨域问题

localhost：5000收不到换成127.0.0.1:5000即可

withCredential

token加在header里的格式，需要前后端一致。比如说在authorization字段下。

### 页面设计

### flask 和 celery 模块化后循环引入的问题

分为两次注册app，第一次用于注册celery，第二次在主程序中提供上下文环境并且注册蓝图

# app/\_\_init\_\_,py  
  
  
def register\_celery(celery, app):  
 class ContextTask(celery.Task):  
 abstract = True  
 def \_\_call\_\_(self, \*args, \*\*kwargs):  
 with app.app\_context():  
 return self.run(\*args, \*\*kwargs)  
 celery.Task = ContextTask  
   
   
def create\_app(celery=None,db\_init=False,register\_celery\_blueprint=False):  
 app = Flask(\_\_name\_\_)  
 app.config.from\_object(Config)  
 db.init\_app(app)  
 CORS(app, supports\_credentials=True, resources={r"/\*": {"origins": "http://localhost:5173"}})  
 if celery:  
 register\_celery(celery=celery, app=app) # >> 注册celery  
 if register\_celery\_blueprint:  
 from celery\_route import celery\_bp  
 app.register\_blueprint(celery\_bp)  
 app.register\_blueprint(api) # /api/search, /api/login # 注册蓝图  
 if db\_init:  
 with app.app\_context():  
 db.drop\_all()  
 db.create\_all() # 数据库初始化  
 # db.session.execute(text('CREATE FULLTEXT INDEX idx\_title ON items(search\_title);'))  
 db.session.execute(text('ALTER TABLE items ADD FULLTEXT INDEX ngram\_index (title) WITH PARSER ngram;'))  
 print('Database initialized')  
 return app  
  
   
def make\_celery(app\_name):  
 celery = Celery(app\_name,  
 broker=celeryconfig.broker\_url,  
 backend=celeryconfig.result\_backend)  
 celery.config\_from\_object(celeryconfig)  
 return celery  
  
celery = make\_celery(\_\_name\_\_)  
app = create\_app(celery=celery, db\_init=False, register\_celery\_blueprint=False)

[notes/flask/03-2 使用 Celery 执行周期任务.md at master · newbieof410/notes · GitHub](https://github.com/newbieof410/notes/blob/master/flask/03-2%20%E4%BD%BF%E7%94%A8%20Celery%20%E6%89%A7%E8%A1%8C%E5%91%A8%E6%9C%9F%E4%BB%BB%E5%8A%A1.md)

# run.py  
import os  
project\_root = os.path.abspath(os.path.dirname(\_\_file\_\_))  
os.environ['PROJECT\_ROOT'] = project\_root  
  
from app import create\_app, celery  
  
app = create\_app(db\_init=False, register\_celery\_blueprint=True)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 for rule in app.url\_map.iter\_rules():  
 print(rule)  
 app.run(debug=True)

### 短信发送

* 一开始准备使用twilio提供的服务，但是不能正常接收到短信，后面换成使用短信宝提供的服务。
* 服务签名一开始使用形如【比价提醒】的格式，但是过于中性，被要求换成公司商标。经测试，【云比价】可以正常通过发送。

### try...catch无法捕捉promise.reject异常

根据逻辑用最外层的处理函数来处理，不能过早地在内部就直接进行捕捉后处理。

### 京东爬虫失效

12/17 京东疑似调整了页面结构和登陆手段，导致爬虫失效，换成使用比价网站的内容。

## 心得体会

在选课之前就已经对B/S 体系软件设计有所耳闻，学长们都称这是一门真正的计算机专业课，一门“开局一个大作业，其他全靠自己写”的课程。

我在上这门课之前接触过一些vue的前端知识——曾经我加入了学校的某个学生组织，在里面参与过一些网站开发的小练手。在部门学长的建议下学习了vue3的教程，但这些也都是“纸上谈兵”。理论知识过不了多久在没有实践的情况下就被忘在了脑后。

后来就是数据库系统课上被要求写一个“图书管理系统”，并且完成相应的前后端。那次开发是在学长提供的框架之下，后端只用按需完成部分填空即可。前端的灵活度也被限制在了仅仅更换某些组件。也没有给出完整的开发框架。

所以本次开发可以算是我第一次完完整整地从零开始架构，独立分析需求、拟定设计文档、开发、测试、部署的项目经历。以下将从开发的各个阶段来简短抒发下心得。（因为其他期末考试还没复习呜呜呜，要来不及了）

### 需求分析

在拿到需求（作业要求）之后，我对项目有了粗浅的理解，认为这并不是一个非常复杂的项目，起码每个功能都有实现的思路。随后，我便开始了挑选前后端的开发技术。

在这一阶段，我也花了不少时间。先前对前后端的开发技术并不了解，只知道有很多种奇奇怪怪的搭配，但从来不知道他们各自在项目中起了什么作用。我先搜索了一些开发者论坛，确定了前端的框架。本来是想尝试下使用react来拓宽自己的技术栈，但是后面还是不想为难自己，准备沿用之前有一定基础的vue3框架。现在想想，也幸亏当时没有选择去学习新的技术，否则完成项目的难度又将提升一个阶梯。

对于后端的语言无非是在java和python之间做出选择。因为我有一定的python基础，所以果断选用了python。

在选择好技术栈之后，我的第一想法是先从github上找到一些开源的、和我使用相同技术栈的项目，学习他们的项目架构来进行改编。如果能找到类似的商城开发项目更是能为我锦上添花。不过很可惜，因为vue3是近几年更新的技术，所以找到的大部分项目使用的都是vue2，或者就是项目架构不够完整。接着，我又转念是否能找到商城框架的静态html，来自己把它转换成vue3的框架。后来，我很幸运地找到了一个别人跟着b站改编的商城项目练手框架，在学习之后我就在这个框架上进行前端开发。

### 前端开发

我的前端开发先从改动一些小组件开始，比如说header、搜索框、footer等。在读懂了别人的项目架构之后，我尝试按照自己的理念来编写页面，在此过程中逐渐熟悉了vue3和typescript的语法。但还是会有一些技术性要求比较高的地方一知半解，比如说vue的响应式处理。

在掌握完编程理论知识后，就要思考项目的逻辑层面需求了。根据以往的经验，我知道联系前后端最重要的便是——“接口设计”。我又去学习了一系列知识，花了一段时间分清前端的路由路径、后端的api路径、互联网url等等概念。后续参照别人的一些博客教程，尝试按照RESTFUL规范来设计。在这个阶段，我的每一次设计其实都充满了自我怀疑，因为我很担心在后端代码的编写中无法和前端的接口匹配而导致需要不停地修改代码。

我起初以为前端只需要负责思考页面的呈现即可，但是发现在引入pinia和axios这些功能之后还要去考虑参数的存储、和后端的交互。学习这一部分也花费了我不少时间，尤其是处理各个store的状态。

### 后端开发

实际上，我的前后端开发是同步进行的，因为比较担心全部写好之后再进行测试的debug工作量会很大。我的后端代码也经历了很多版本的迭代，从一开始的单文件，到后来的多文件，再到最后的不同文件模块下的多文件格式。代码的架构逐渐复杂化，也提高了我的开发难度。

我先根据需求文档设计好了数据库结构并且定义了modles，事实证明一开始很难想得非常全面，这个数据库的格式在后续开发中还是经历了多轮迭代修改。

我最初尝试实现的是登录和注册功能。为了更加规范，在这里我接触到了我很多不了解的概念，比如说登录后的JWT验证。我也不知道该怎么处理这些验证。不过在后续的学习中，我也做到了对登录信息进行了一定的处理，比如说对密码进行加密后传输、赋予登录的用户一个JWT token来验证其身份信息等等。

接着就来到本项目最难实现的部分了——爬虫。我早就对各大电商网站的反爬机制有所耳闻。先在网上找了一圈爬虫代码后，发现都无法处理淘宝的情况（request一直失败、selenium又依赖于手动过于缓慢），于是我转战京东。幸运的是，很快就发现了使用requsest就能成功处理京东的场合。但是根据这个原理我试图在淘宝上复现，还是无果。多次尝试后还是放弃了从淘宝获取数据。准备先用京东搭好基础框架。

其他的前后端交互（价格提醒，历史价格，详情页面）部分完成地都比较顺利，只需要确定好和前端之间数据传递的格式，返回前端需要使用的数据即可。需要注意的是，数据的格式真的非常非常重要，一不小心就会匹配错误！这段时间的开发非常消耗体力和耐心。

后来，就到了要实现异步处理价格提醒的部分了。这一阶段我使用了celery队列来完成，准备让他在后台定期执行任务。这又要求我学习新的技术，不过我也已经习惯了（笑）。在粗略地了解了celery和redis之后，我就尝试把它们揉和起来。过程中还是遇到了很多困难的，尤其是和flask app初始化循环引用的问题。不过在网上参照了别人的博客之后还是顺利结局了。在这段开发过程中，我还使用了postman不断进行测试，节省了很大的精力。

这一部分我还学习了如何使用python借助互联网协议发送email，如何调用其他短信运营商提供的接口发送短信等技术，收获颇丰。（但也真的很花时间😭）

终于，在项目的后期，我尝试接入另一个电商平台——亚马逊。亚马逊的反爬做得并不严格，所以处理起来比京东更快一些。其实后来还打算接入更多的平台，例如苏宁易购、唯品会等等。我分析了苏宁的接口请求感觉还是比较好爬的，不过没太多时间就先暂时搁置了。

### docker部署

这又是一个我完全没接触过的领域！唉，从头学习。

不过这一阶段我遇到比较大的问题就是的前端和后端请求一直匹配不上，以及后端和redis等环境不能成功对接。在debug的过程中我倒是对了网络交互有了更多的认识，包括nginx反向代理、localhost、端口、宿主机代理、docker网络等等有关计算机网络的知识。最后成功打包并且运行起来的时候还是很有成就感的。

最后尝试并且打包发布在了Docker Hub上。由于全部都是自己在网上学习，也不知道做的是否规范。

### 存在的不足

开发过程中最让人崩溃的无非就是**不断失灵的爬虫**！由于有时候的过多请求或者电商网站更换机制，爬虫总是会间歇性地失效，要求我不断去获取新的cookie或者调整其他代码。这一点也是我部署之后最担心的一点，也导致了我的网站无法长期的自主运营下去。需要我或者用户不断自己维护。

同时，由于并没有使用很好的搜索算法，只是简单地使用了mysql本身支持的全文索引，商品的搜索结果往往不尽如人意，无法按照相关度进行结果的优先级排序展示。

总的来说，我的网站总有很多还能优化的部分，比如对高并发的支持，以及一些细节处的维护（以免用户试出bug）。我总是对我的网站充满了担心，觉得他非常地破碎，害怕他运行着运行着就崩溃了。

### 收获

最大的感受是，一个人毫无经验的网页开发真的好难做到规范化，因为会缺少很多“常识性”的指导。从一开始，我就不停地在网上去搜索一些“开发规范”，但似乎大部分都没有一个明确的规定。找了很久才找到类似于，“设计文档怎么写”，“接口怎么设计”，这样的“授人以渔”的教程。我想，大部分的网页开发应该是在一个已经具有一定规模的团队内部进行的，而团队内部会从更加宏观的层面基于前人经验规定好开发要求。但是对于我这样一个还未入门的初学者来说，很难获取到相关信息，也不愿意投入很大的时间进行文档学习。所以大部分情况我都是借助ChatGPT等工具，希望他们为我提供一个“常识性”的解答。比如说，前端的架构该如何设计、后端的模块化应该怎么划分等等。

此外，我还领悟到，**学习就是要在动手中学**。不要等到学完了所有理论知识再动手，完全可以借助各种各样的工具、前人经验，先动手开始写，在一次次修改代码和debug中就能学到很多知识。（但是如果debug效果实在不好，还是得从理论层面来分析错误的）

### 总结

希望老师测试的时候手下留情呜呜呜！

以及真的非常感谢老师这学期的付出和解答，从这个作业中学习到了很多网页开发知识，也算是一种外力让我入门了一点点网页开发。总的来说是一次很有趣很有趣的项目经历，学习新技术很开心，但是真的很花时间。在这个过程中真的提升了很多个人的学习能力和代码能力！