****

Nosql

课程设计报告

|  |  |
| --- | --- |
| **题目** | **英语听力网站** |
| **专业** | **信息与计算科学** |
| **班级** | **本科二班** |
| **姓名** | **李正强** |
| **教师** | **陈刚** |

2018年 12月 23日

# 系统整体设计

1. 爬虫部分

使用python+scrapy实现对可可英语网站音频的爬取并结构化数据自动保存至数据库，关键点为多级爬取链接。

1. Web部分

使用python+flask实现web界面，登陆展示内容等，使用elasticSearch+ik分词实现关键字搜索。

# 系统功能实现及主要代码展示

1. 多级爬取

class KekespiderSpider(scrapy.Spider):  
 name = 'kekespider'  
 allowed\_domains = ['www.kekenet.com']  
 start\_urls = ['http://www.kekenet.com/Article/chuji/List\_292.shtml']  
  
 # 开始访问  
 def start\_request(self):  
 yield scrapy.Request(self.start\_urls, callback=self.parse,dont\_filter=True)  
  
 # 一级页面  
 def parse(self, response):  
 for msg in response.xpath('//ul[@id="menu-list"]/li'):  
 inurl = msg.xpath('h2 / a[2]/@href').extract()  
 yield scrapy.Request(inurl[0], callback=self.parse1)  
 nowIndex = response.xpath('//div[@class="lastPage\_left"]/div[@class="page th"]/b/text()').extract()  
 print(nowIndex)  
 next\_page = 'http://www.kekenet.com/Article/chuji/List\_{0}.shtml'.format(str(int(nowIndex[0])-2))  
 print(next\_page)  
 if next\_page:  
 request\_url = next\_page  
 print(request\_url)  
 yield scrapy.Request(request\_url, callback=self.parse)  
  
 # 二级访问  
 def parse1(self,response):  
 src = response.xpath('//div[@class="lastPage\_left"]/div[3]/span/a/@href').extract()  
 src = response.urljoin(src[0])  
 print("二级界面"+src)  
 yield scrapy.Request(src, callback=self.parse2)  
  
 # 三级访问  
 def parse2(self, response):  
 item= KekeItem()  
 item['file\_name'] = response.xpath('//div[@class="lastPage\_left"]/div[@class="list\_box\_2"]/ul[1]/table[1]/tr[1]/td[2]/a/text()').extract();  
 item['src'] = response.xpath('//div[@class="lastPage\_left"]/div[@class="list\_box\_2"]/ul[1]/table[1]/tr[4]/td[1]/a/@href').extract()  
 yield item

1. 数据库存储

import pymongo  
  
class KekePipeline(object):  
 id = 0  
 def \_\_init\_\_(self):  
  
 host = '127.0.0.1'  
 port = 27017  
 db\_name = '201802final'  
 client = pymongo.MongoClient(host=host, port=port)  
 db = client[db\_name]  
 self.post = db['data\_link']  
  
 # 捕捉到item的时候执行  
 def process\_item(self, item,spider):  
 book\_info = dict(item)  
 self.post.insert(book\_info)  
 return item

1. es导入

from pymongo import MongoClient  
from elasticsearch import Elasticsearch  
import traceback  
  
# 创建连接  
\_db = MongoClient('mongodb://127.0.0.1:27017')['201802final']  
es = Elasticsearch('127.0.0.1:9200')  
# \_db.update({}, {"$set": {"src": "20130408120000"}})  
# 初始化索引的Mappings设置  
\_index\_mappings = {  
 "settings": {  
 "number\_of\_shards": "1",  
 "analysis": {  
 "analyzer": {  
 "ik": {  
 "tokenizer": "ik\_smart"  
 }  
 }  
 }  
 },  
 "mappings": {  
 "listen1": {  
 "properties": {  
 "file\_name": { "type": "text" },  
 "src": { "type": "text" },  
 }  
 }  
 },  
  
}  
# 创建索引  
if es.indices.exists(index='listen1\_index') is not True:  
 es.indices.create(index='listen1\_index', body=\_index\_mappings)  
  
# 从MongoDB中查询数据，由于在Elasticsearch使用自动生成\_id，因此从MongoDB查询  
# 返回的结果中将\_id去掉。  
# data\_cursor = \_db.data\_link.find({}, projection={'\_id': False})  
data\_cursor = \_db.data\_link.find({}, projection={'\_id': False})  
data\_docs = [x for x in data\_cursor]  
# 批量导入  
processed = 0  
for \_doc in data\_docs:  
 try:  
 processed += 1  
 es.index(index='listen1\_index',doc\_type='listen1',body=\_doc,id=str(processed))  
 print('Processed:'+str(processed),flush=True)  
 except:  
 traceback.print\_exc()

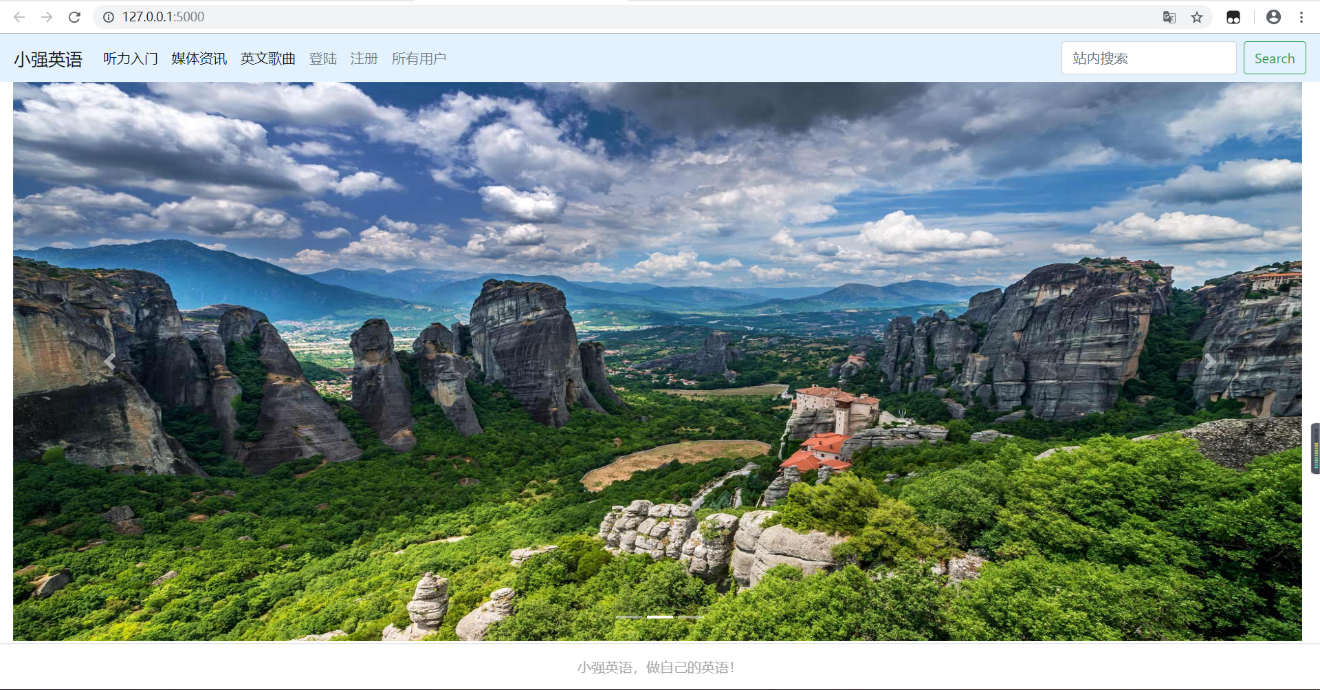
1. 关键字查询

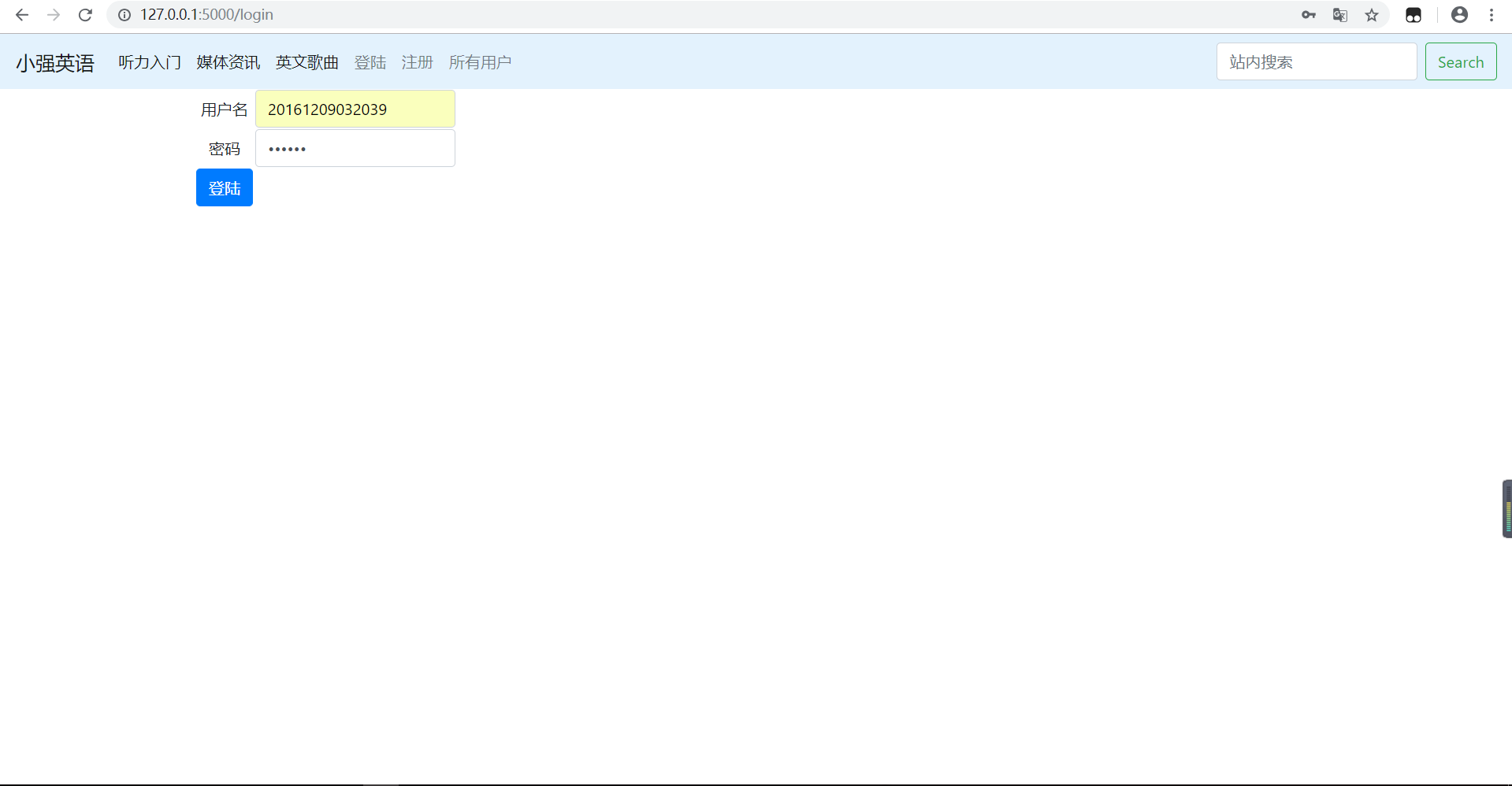
from elasticsearch import Elasticsearch  
  
# 此方法为搜索关键词时调用  
def do\_search(key\_word):  
 # 连ElasticSearch  
 es = Elasticsearch('127.0.0.1:9200')  
 # 关键字信息  
 print('Search name contains '+key\_word, flush=True)  
 # 查询规则  
 \_query\_name\_contains = {  
 'query': {  
 'match': {  
 'file\_name': key\_word  
 }  
 }  
 }  
 # 进行搜索,索引为entry\_level\_index,doc\_type为entry\_level,  
 \_searched = es.search(index='listen1\_index', doc\_type='listen1', body=\_query\_name\_contains)  
 # print(\_searched, flush=True)  
 #  
 # for hit in \_searched['hits']['hits']:  
 # print(hit['\_source'], flush=True)  
  
 # 返回dict字典 提取主要信息  
 return \_searched['hits']['hits']

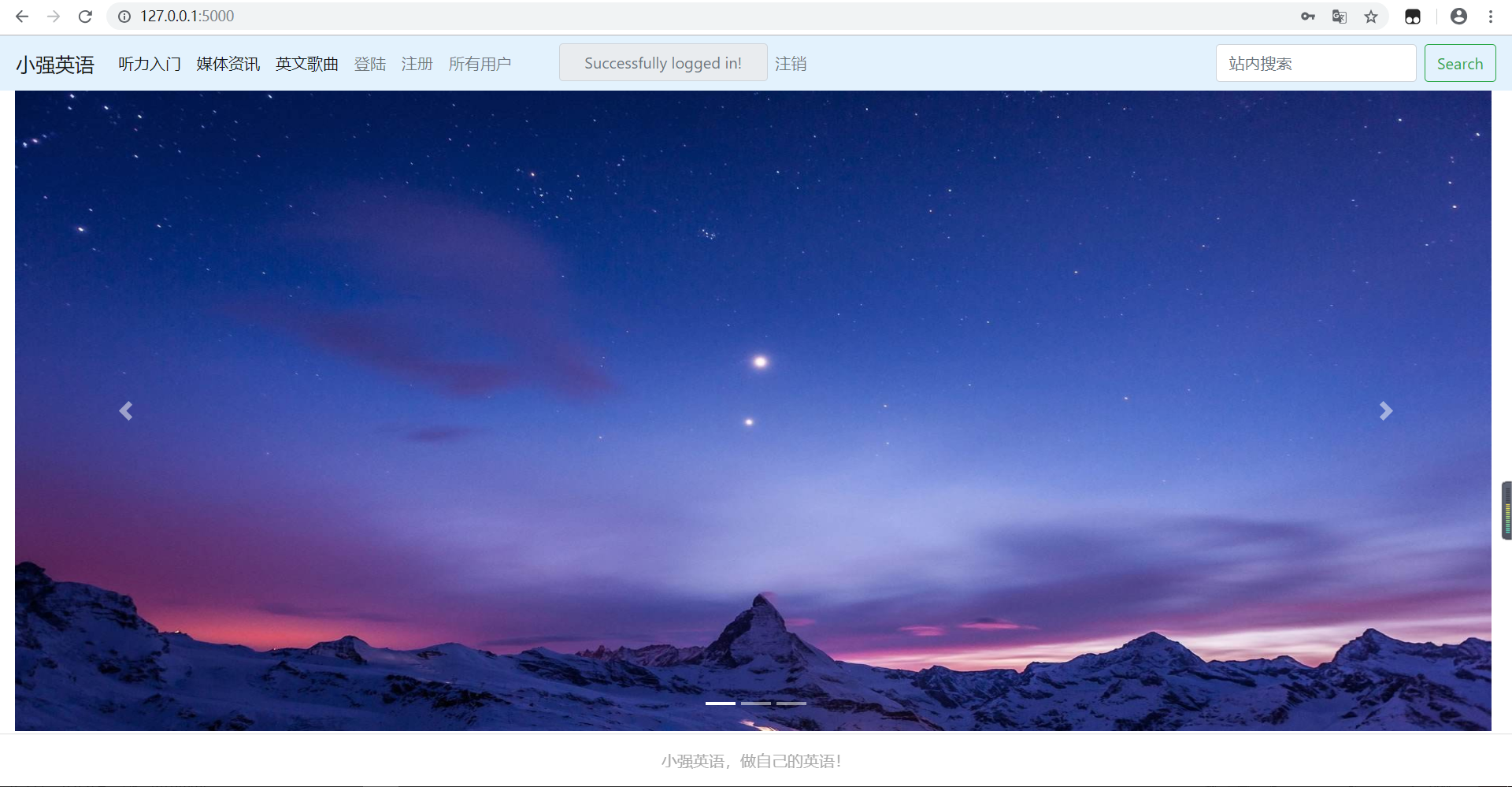
1. web页面后台处理

# 创建蓝图  
main=Blueprint('main',\_\_name\_\_)  
  
# 权限控制 相当于jfinal中的拦截器  
def admin\_login\_req(f):  
 @wraps(f)  
 def decorated\_function(\*args, \*\*kwargs):  
 # 检查session是否是空的,每一个线程都对应一个session  
 if session.get('user\_id') == '' or session.get('user\_id') == None:  
 # 消息闪现 可以作为提示信息等 基于session实现  
 flash('请登陆!')  
 # 跳转页面 重定向 注意 url\_for 的参数为方法名称  
 return redirect(url\_for("main.login"))  
 return f(\*args, \*\*kwargs)  
 return decorated\_function  
  
# 主页映射 注解 url\_map 将来注册蓝图之时,添加到url\_map映射中  
@main.route('/')  
def index():  
 # 实例化搜索表单  
 form\_search = SearchForm()  
 # 渲染页面 传递表单参数  
 return render\_template('index.html',form\_search=form\_search)  
  
# 登陆映射  
@main.route('/login',methods=["GET","POST"])  
def login():  
 # 判断请求的方式  
 if request.method == 'GET':  
 form\_search = SearchForm()  
 form = LoginForm()  
 return render\_template('login.html', form=form,form\_search=form\_search)  
 else:  
 form = LoginForm(request.form)  
 # 对表单内容是否通过验证进行判断  
 if form.validate():  
 # 获取表单的数据  
 data = form.data  
 # 数据库查询  
 admin = User.objects(user\_id=data['username'], user\_pwd=data['password']).first()  
 # 对查询结果进行判断  
 if not admin:  
 flash('密码错误或用户不存在！')  
 return redirect(url\_for('main.login'))  
 else:  
 flash('Successfully logged in!')  
 session['user\_id'] = data['username']  
 return redirect(url\_for('main.index'))  
 else:  
 return redirect(url\_for('main.login'))  
  
# 注销映射  
@main.route('/logout')  
def logout():  
 flash('已退出登陆！')  
 # 从session中删除即可  
 session.pop('user\_id')  
 return redirect(url\_for("main.index"))  
  
# 注册映射  
@main.route('/sign',methods=["GET","POST"])  
def sign():  
 # 判断请求方式  
 if request.method == 'GET':  
 form\_search = SearchForm()  
 form = SignForm()  
 return render\_template('sign.html', form=form,form\_search=form\_search)  
 else:  
 form = SignForm(request.form)  
 print('注册表单验证:',form.validate\_on\_submit())  
 # 判断表单是否通过验证  
 if form.validate():  
 data = form.data  
 # 通过查询结果判断 是否已存在该用户  
 admin = User.objects(user\_id=data['username']).first()  
 if not admin:  
 flash('注册成功!')  
 User(user\_id=data['username'],user\_pwd=data['password']).save()  
 return redirect(url\_for('main.index'))  
 else:  
 flash('用户已存在!')  
 return redirect(url\_for('main.sign'))  
 else:  
 flash('输入格式有误!')  
 return redirect(url\_for('main.sign'))  
  
# 为方便查看数据所有用户  
@main.route('/allUser')  
def all\_user():  
 form\_search = SearchForm()  
 users = User.objects().all()  
 return render\_template('allUser.html', users=users,form\_search=form\_search)  
  
# 听力入门页面 映射 以及拦截认证注解  
@main.route('/entryLevel')  
@admin\_login\_req  
def entry\_level():  
 form\_search = SearchForm()  
 # 获取get请求传过来的页数,没有传参数，默认为1  
 page = int(request.args.get('page', 1))  
 # 调用数据库分页的方法进行查询  
 paginate = db.view\_todos(page)  
 # Pagination object的内容  
 stus = paginate.items  
 return render\_template('entryLevel.html', files=stus, paginate=paginate,form\_search=form\_search)  
  
# 听的界面  
@main.route('/listenIt')  
def listen\_it():  
 form\_search = SearchForm()  
 args = dict(request.args)  
 data\_id = args['id']  
 print(data\_id[0])  
 # 判断是从搜索页面来的 还是通过听力入门界面来的  
 if len(data\_id[0]) == 24 :  
 file = Todo.objects(id=data\_id[0]).first()  
 else:  
 file = Todo.objects().skip(  
 int(data\_id[0])-1  
 ).first()  
 return render\_template('listenIt.html', file=file,form\_search=form\_search)  
  
# 搜索处理方法  
@main.route('/search', methods = ['POST'])  
def search():  
 form = SearchForm(request.form)  
 form\_search = SearchForm()  
 data = form.data  
 key\_word = data['key\_word']  
 # 进行检索  
 results = do\_search(key\_word)  
 return render\_template('search\_results.html',form\_search=form\_search,results=results,query=key\_word)  
  
# 404页面  
@main.errorhandler(404)  
def page\_404(er):  
 return render\_template('404.html',er = er)

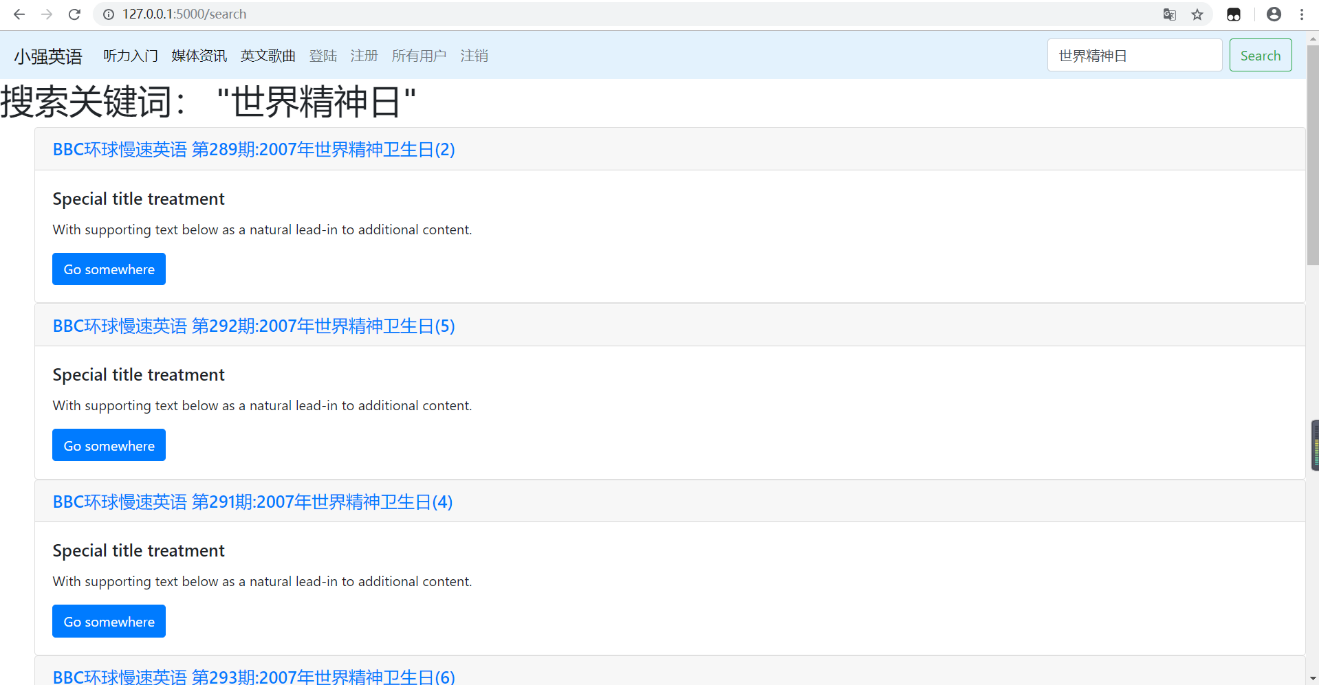
# 结果展示

1. 主页
2. 登陆功能展示





（3）关键字搜索功能展示



# 心得体会

本学期我们开设了Nosql这门课程，学习任何知识，仅从理论上去求知，而不去实践、探索是不够的，课程设计是很及时、很必要的。这样不仅能加深我们对Nosql数据库的理解，而且真正的做到了学以致用。通过我们不懈的努力与切实追求，终于做完了课程设计。学会了如何使用python中flask和scrapy框架去开发web网页以及多级爬虫，学会了如何使用elasticsearch+ik分词实现站内关键字搜索的功能。这次课程设计让我学到了很多，不仅巩固了先前学的数据库理论知识，而且也培养了我的动手能力，更令我的创造性思维得到拓展。希望今后类似这样课程设计、类似这样的锻炼机会能更多些!