实验 6-Flask

姓名：王煌基　　学号：519030910100　　班级：F1903004

一、实验概览

基于对 web 框架:Flask 的初步认识，学习简单的网站开发。通过利用 Flask 模板内丰富的函数、过滤器以及 Flask 表单的创建，结合 HTML、Lucene、中文分词等知识点，建立一个简单的搜索引擎

二、实验环境

1. 个人笔记本电脑
2. 操作系统：windows10 专业版
3. 使用软件：Visual Studio Code；Docker Desktop

三、实验过程

练习一：使用 Flask，结合前面学习的 HTML，Lucene，中文分词等知识点，根据上次实验爬取的网页，建立一个简单的搜索引擎。

【解】

基于在 lab4、lab5 中对 Lucene 的初步认识，我们已经能够简单地建立一定的索引并进行查找，这个任务的不同点在于将我们在终端上操作、显示的内容搬运到网页中，以网页作为载体进行任务实现。

通过对题意的理解,我知道应该需要 search.html 的初始网页、result.html 的结果网页，以及联系这两个 html 的 search.py 文件，并展开这个想法的具体实现：

①search.html：

需要实现简单的搜索框以及传参过程，所以只需要在 bio_form.html 上进行简单的修改即可，实现如右图所示。


```
<h1>Search</h1>
<form action="result">
    <label>keyword</label>
    <input type="name" name="keyword"><br>
                  
    <input type="submit" name="Search">
```

②result.html：

根据 PPT 内的结果展示，可以发现，首先我们需要保留在 search.html 内的搜索框，这一块的实现是简单的，完全复制 search.html 的相应代码块即可。

其次是实现 Search Result for “keyword”，而这很明显只需要一个 h1 标签的承载并且传入参数 keyword 即可：


```
<h1>Search Result for "{{ keyword }}"</h1>
```

最后是最重要的搜索结果的呈现，观察得搜索结果由：

1. 带超链接的标题
2. 关键词的上下文
3. 网页的具体链接

三部分构成，因此我们可以通过将我们 SearchFiles.py 的参数传入的方式来完成这个过程。

SearchFiles.py 内的参数传入我利用一个内嵌字典的元素实现，即：

```
TEXTS = [
    {
        ‘title’:…,
        ‘content’:…
        ‘url’:…
    },{
    }…
]
```

从而，我们只需要利用条件语句，调用起来便十分简单（下面是略写）：

```
{% for texts in TEXTS %}
    <div>
        <!-- 生成带有超链接的标题 -->
        <!-- 生成带有关键词上下文的内容 -->
        <!-- 生成网页链接 -->
    </div>
{% endfor %}
```

从而我们的 search.html 与 result.html 均已完成。

③search.py：

search.py 作为两个 html 间的联系枢纽，是需要接收它们的传参并输出结果的，所以对于 search.html 的参数，我们接收后可以重定向到我们的结果网页 result.html 中：

```
@app.route('/search', methods=['POST', 'GET'])
def search():
    if request.method == "POST":
        keyword = request.form['keyword']
        return redirect(url_for('result', keyword=keyword))
    return render_template("search.html")
```

较为复杂的是 result 函数，因为这个函数既要完成结果的搜索，又需要将得到的参数传入我们的 result.html 中，所以我们需要逐步分解来简化这份练习。

基于 lab4、lab5 的学习，对于搜索的条件，首先需要进行分词操作，因为我们建立好的

索引中本身就是以分词后的形式存储的：

```
keyword = request.args.get('keyword')
keyword = jieba.cut(keyword, cut_all=False)
keyword = " ".join(keyword)
```

并且，需要注意的是，当我们点击提交的时候，我们需要做好用户不输入的准备，即如果用户直接点击提交，我们需要让此次点击无效化，无效化即返回 search.html（刚开始我缺少这点的考虑导致这里总会导致在 query 空字符的时候异常返回）：

```
if(keyword == ""):
    return render_template("search.html")
```

接下去则是我们所熟悉的分词后检索的操作，与 lab4、lab5 的 SearchFiles.py 是几乎一样的，只是比较不同的点在于我们需要建立一个列表型元素并将字典型元素传入列表中（带有关键词上下文的内容比较麻烦，我在**四、实验问题及解决**中展开）：

```
TEXTS = []
for scoreDoc in scoreDocs:
    doc = searcher.doc(scoreDoc.doc)
    dic = {}
    dic['title'] = doc.get("title")
    dic['url'] = doc.get("url")
    TEXTS.append(dic)
```

并且最后将 TEXTS 的参数传入 result.html 即可：

```
return render_template("result.html", keyword=tmp, TEXTS=TEXTS)
```

从而我们的任务得到了解决。

四、实验问题及解决

问题一：在实验过程中，出现了虚拟机初始化被反复调用的过程，导致文件无法正常运行，如何处理？（代码为虚拟机的初始化代码）

```
lucene.initVM(vmargs=['-Djava.awt.headless=true'])
```

【解】

最初我尝试利用如 try 等方法来解决这个问题，但都无法成功，具体原因在于：

对于 try，当我进行 try 的时候，实际上会造成“内部变量影响外部变量”的 BUG，即我定义的变量是内部的，内部结束之后是与外部毫无关系的，因此无法解决这个问题。

```
    query = QueryParser("contents", analyzer).parse(command)
RuntimeError: attachCurrentThread() must be called first
```

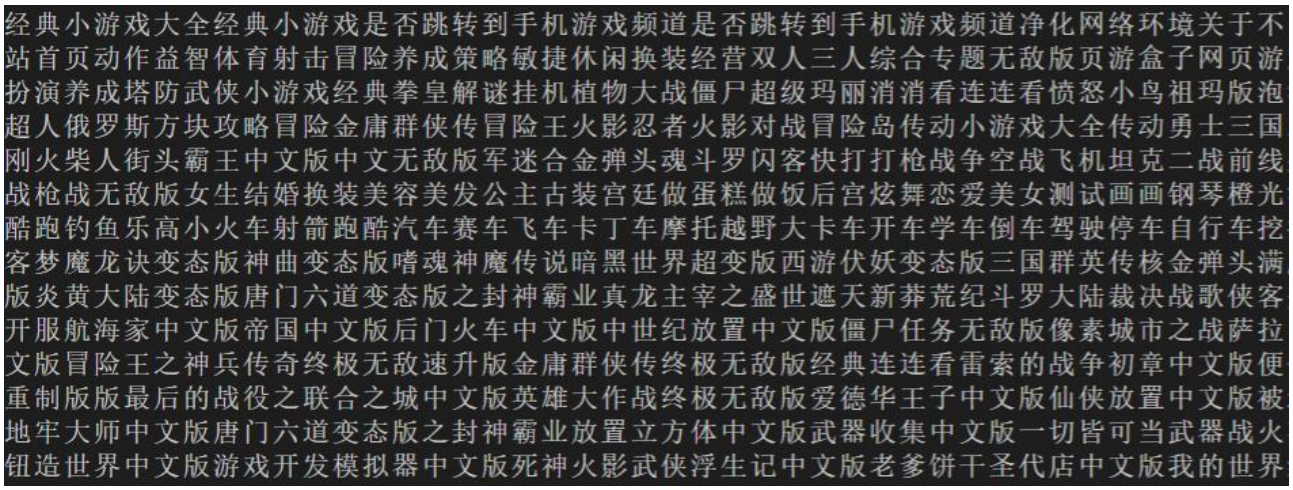
因此，通过其报错的原因界面，我去网上查找与 attachCurrentThread 函数相关的资料，可以发现，由于虚拟机的初始化只能被执行一次，我们只能在 main 函数内进行声明，并且将其的具体调用放在我们的函数内部：

```
@app.route('/result', methods=['POST','GET'])
def result():
    ...
    s.attachCurrentThread()
    ...
if __name__ == '__main__':
    s = lucene.initVM(vmargs=['-Djava.awt.headless=true'])
```

从而，我们的问题得到了解决。

问题二：如何输出关键词的上下文？

【解】



由于我对网页内所有中文的保存是一大段的以空格分词的字符串，所以我刚开始思考的是通过找上一个空格、下一个空格的方式来“切出来”我们要的片段，但是经过具体实现，这样的方法不仅麻烦，而且调试起来非常不容易。

之后，我想到了可以将我们这一大段以空格分词的字符串进行切割，转化为列表型，然后通过遍历的方式，先找到该关键词的具体位置，然后再往它的上下各爬取 4 个左右的关键词，从而就能凑出一个完整的关键词上下文，而这个方法的实现会相对简单很多：

```
content = doc.get("contents")
content = content.split(' ')
j = 0
test = 1
#找到关键词左边的内容
solutionl = []
#找到关键词右边的内容
solutionr = []
flag = False
for i in search_word:
    #已经找到了一次就直接退出，提高效率
    if(flag):
        break
```

```
    #一个一个词找过去比对
    for j in range(0,len(content)):
        if(flag):
            break
        #找到之后，爬取上下文
        if(i==content[j]):
            flag = True
            if(j>0):
                test = 4
                while(test>0):
                    if(j-test>=0):
                        solutionl.append(content[j-test])
                    test -= 1
            if(j<len(content)):
                test = 1
                while(j+test<len(content) and test<=4):
                    solutionr.append(content[j+test])
                    test += 1
#上下文保存并用 dic 存入
solutionl = "".join(solutionl)
solutionr = "".join(solutionr)
dic['left'] = solutionl
dic['right']= solutionr
```

从而，我的关键词上下文获取得以实现。

五、实验结果

【练习一：建立一个简单的搜索引擎】

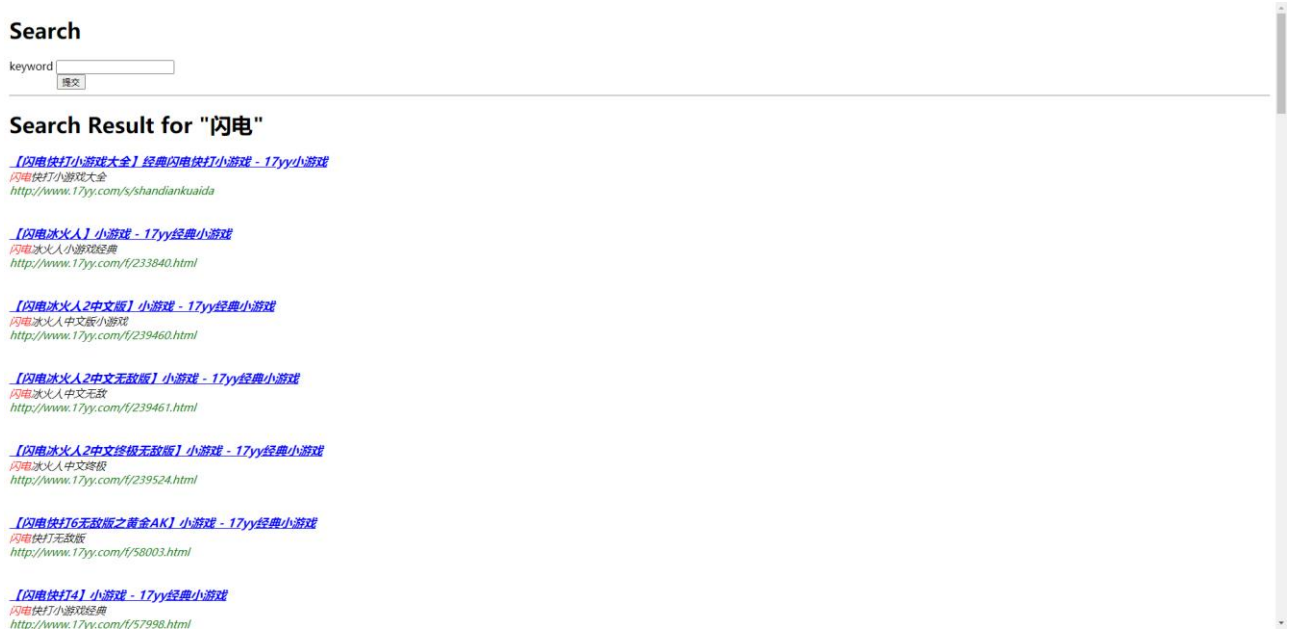
运行 search.py 建立网页：

```
^Croot@ac2d234ef3c5:/workspaces/code# python search.py
 * Serving Flask app "search" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:8080/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 203-876-595
127.0.0.1 - - [23/Oct/2020 07:51:57] "GET /search HTTP/1.1" 200 -
```

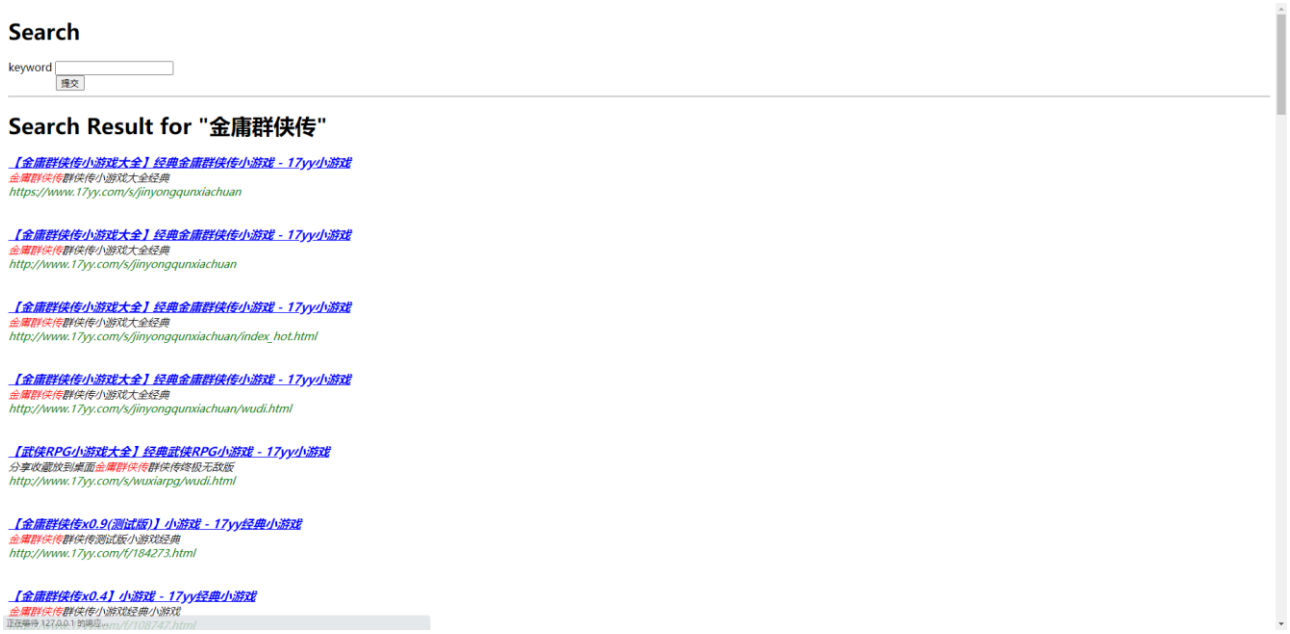
查看 127.0.0.1:8080/search：

输入为空，返回原网页：

输入为闪电：



输入为金庸群侠传：



六、实验体会

这是本课程的第五次实验作业，第一次接触 HTML 语言让我感到新奇而且充满动力，而且本次的难度层层递进，具有挑战感，提升了我的获得感。

在本次实验中，我再一次加深了对 doc 的存储方式的理解以及对 HTML 语言的应用，从而能够很好地实现一个完整的网站。

当然，本次实验也存在不足之处，主要是由于查找关键词的上下文方式过于简单，并不能很好地还原原网页的全貌，而且只找到其中一项便输出，当遇到多项匹配的时候只会输出第一项，因而该提取方法仍需要改进。