Experiment 6/7

517030910356 谢知晖

目录

1 实验准备

1.1 实验环境

本次实验环境依旧为 Ubuntu 平台 (版本号 18.04) 下的 Python(版本号 2.7)。

实验中，我们使用 web.py 框架来开发我们的搜索网页，并结合之前的 lucene 来实现搜索结果的呈现。

1.2 实验目的

这次的实验没有太多的新内容，旨在整合前面学过的内容，将我们之间建立的搜索结构进行更好地呈现。

我们开始尝试建立搜索引擎的雏形。像百度那样，针对用户的搜索进行本地查找，并实时将结果送到我们的网页当中。这有助于我们在之后建立完整的网站。

同时，我们实现了对于网页和图片两种对象的搜索，做到了功能化的区分。我们也针对网页内容的需求对网页的呈现形式进行定制，使网页更加美观。

1.3 实验原理

本次实验中主要利用 web.py 来开发网页前端，而在后端我们用 lucene 进行查询。由于 lucene 我们在之前的实验中已经介绍过，这次我们仅对 web.py 框架以及一些 DIV/CSS 的基础知识进行介绍。

1.3.1 web.py

web.py 是一个轻量级 Python web 框架，简单且功能强大。我们简单地介绍用 web.py 实现的网页的组成形式。

URL 映射 URL 映射就是一个 URL 请求由哪块代码 (类、函数) 来处理。web.py 的 URL 控制模式简单的、强大的、灵活。在每个应用的最顶部，通常会看到整个 URL 调度模式被定义在元组中, 如:

```
urls = (
    '/', 'Index',
    '/view/(\d+)', 'View',
    '/new', 'New',
    '/delete/(\d+)', 'Delete',
    '/edit/(\d+)', 'Edit',
    '/login', 'Login',
    '/logout', 'Logout',
)
```

元组中由若干个"URL-处理类" 组成。我们可以利用强大的正则表达式去设计更灵活的 URL 路径。

且 URL 映射有 3 种类型：

1. URL 完全匹配 (如'/index','Index')
2. URL 模糊匹配 (如'/view/(+)', 'View')

3. URL 带组匹配 (如'/post2/(+)','Post2')

三种类型都对特定形式的 URL 使用对应的处理类。但第三类与其他两类不同的是它能够捕获 URL 最后的参数，而其他两类是无法处理参数的。

请求处理 URL 的请求参数，包括 URL 查询的参数 (? 后面的内容)，表单提交 (GET/POST 方法)，都是用"web.input" 函数取得的。

```
data = web.input()
```

得到的'data' 是一个字典，这有助于我们处理网页请求。

响应处理 对于 (通过表单等获得的) 用户的请求，我们需要给以其实时的反馈。为了让用户得到正确的反馈结果，数据的组织必须是有条理的，且能够根据用户的不同请求做出不同的反馈。在 web.py 中，这样的响应处理通常是由模板实现的。

一个典型的模板形式如下:

```
$def with (name)

$if name:
    I just wanted to say <em>hello</em> to $name.
$else:
    <em>Hello</em>, world!
```

模板看上去跟这 Python 文件很相似，以"def with" 语句开始，但在关键字前需要添加"$"。

模板首先接受调用时传入的参数。对于不带"$" 的内容，我们可以理解为 HTML 代码的内容。在执行模板时，每执行到一条不带"$" 的语句，便向 HTML 中加入这条语句。而以"$" 开头的内容则可以用在 Python 中的同样的处理方式处理。

模板调用时，对于由 HTML 文件 (而非简单的字符串) 组织的模板，我们需要先设置模板的路径:

```
render = web.template.render('templates')
```

在之后调用时，我们便可直接使用 render 下的模板。通过在处理类中进行 return 的方式，相应的模板便能够显示在当前网页上。

```
class s:
    def GET(self):
        user_data = web.input()
        a = func(user_data.keyword)
        return render.result(a, user_data.keyword)
```

1.3.2 DIV/CSS

CSS 全称为" 层叠样式表"(Cascading Style Sheets)，它主要是用于定义 HTML 内容在浏览器内的显示样式，如文字大小、颜色、字体加粗等用于设置页面的表现。

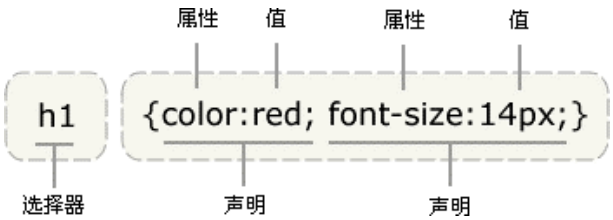
CSS 能够帮助我们将文档信息内容和如何展现它的细节相分离。与在 HTML 正文中定义样式相比，有以下几点明显的优势:

1. 避免重复
2. 更容易维护

3. 为不同的目的，使用不同的样式而内容相同

总的来说，使用了 CSS 之后，我们在 HTML 中使用标记语言只是用来描述内容而非样式，样式已经在正文中被去除了。

基本语法 CSS 规则由两个主要的部分构成：选择器，以及一条或多条声明:



选择器 指明网页中要应用样式规则的元素

声明 在英文大括号"" 中的的就是声明，属性和值之间用英文冒号":" 分隔。当有多条声明时，中间可以英文分号";" 分隔，如下所示:

```
p {
    color:red;
    text-align:center;
}
```

CSS 支持如标签选择器、类选择器等多种选择器，某些样式还具有继承性，能够将样式应用于某个元素的后代中。我们能够通过组合使用这些选择器来组织我们的网页样式。

样式引入 CSS 样式代码的引入方式分为以下三种:

1. 内联式 CSS 样式
2. 嵌入式 CSS 样式
3. 外部式 CSS 样式

其中，内联式 CSS 样式表就是把 CSS 代码直接写在现有的 HTML 标签中；嵌入式 CSS 样式，就是可以把 CSS 样式代码写在"style" 标签之间；而外部式则是把 CSS 代码写一个单独的外部文件中，使用"link" 标签将 CSS 样式文件链接到 HTML 文件内。

2 实验过程

2.1 Exp6-1

在此次练习当中，我们无需对我们的后端查询做大的改动，仅需要将之前在程序中执行的查询展现在网页当中。但针对网页中的高亮内容，我们还是有必要微调我们的索引结构。

2.1.1 实验步骤

索引调整 在之前的实验中，我们对于所查询的内容，是无需建立关于网页文本内容的 Field 的，因为我们并不需要向用户返回这些内容。而这次，为方便用户查看网页预览信息，我们还需要保存我们的文本内容。

这似乎并非十分必要，因为我们已经建立了分词好的文本信息，不妨在建立索引时直接只建立未分词的 Field，然后在查询时先提取出保存的文本，再分词。不过，由于分词也会消耗一定的时间，我还是选择同时建立分词和未分词的两个文本 Field，以加快查询速度。

网页结构 在本次练习中，我们的网页结构十分简单。

我们建立两个处理类来处理 URL:"index" 用来处理网页主 URL，"s" 用来处理网页搜索页面。

```
urls = (
    '/', 'index',
    '/s', 's'
)
```

在"index" 中，我们简单地调用了 web.py 的 form 模块建立 html 表单，以获取用户输入。

```
class index:
    def GET(self):
        f = login()
        return render.formtest(f)
```

而在"s" 中，我们则需要调用函数"getResult" 请求我们的本地数据。在得到用户的输入之后，我们将其传递给函数"getResult"，得到所需的 data。

```
class s:
    def GET(self):
        user_input = web.input().keyword
        search_result = getResult(user_input)
        return render.result(search_result, user_input)
```

数据处理 在"getResult" 中，我们和之前的实验中做的查询类似，对分词后的文本内容进行检索。在此我不再赘述，仅介绍高亮文本内容的实现方式。

```
# ...

for scoreDoc in scoreDocs:
    doc = searcher.doc(scoreDoc.doc)

    item = {}

    item['title'] = doc.get('title')
    item['url'] = doc.get('url')
    item['context'] = highlight(doc.get('content'), command)

    data.append(item)

# ...
```

函数"highlight" 接受查询对象 context 以及我们查询的关键词 keyword，并返回一个二维元组，内容为在 context 中 keyword 前后一定范围内的文本内容。这里我们将范围定在 keyword 前后 5 个字符。

其实现并不复杂，只需用到 string 中的函数"find" 即可。

```
def highlight(text, keyword):
    index = text.find(keyword)

    if index == -1:
        print "Not exist"
        return ''

    else:
        length = len(text)

        if index < 5:
            start = 0
        else:
            start = index - 5

        if length < index + len(keyword) + 5:
            end = -1
        else:
            end = index + len(keyword) + 5

        result = text[start:index], text[index + len(keyword):end]
        return result
```

内容呈现 在获取了我们的数据之后，我们便能够将数据反馈到网页当中。网页的样式被定义在"result.html" 中，我们可以直接向样式中填充数据。

```
$def with (data, input)
    $if data:
        <font size="20">Search Result for "$input"</font>
        <br>
        $for web in data:
            <font color="blue">
                <a href=$web['url'] target="_blank">$web['title']</a>
            </font>
            <br>
            $web['context'][0]
            <font color="red">
                $input
            </font>
            $web['context'][1]
            <br>
            <font color="green">$web['url']</font>
            <br>
            <br>
            <br>

    $else:
```

```
        No Results!
```

对于 data 数组中的每一条数据，我用一个 for 语句来遍历填充到网页内容当中。

2.1.2 实验结果

现在基本的网页搜索已经构建完成，用户能够通过输入得到预期的结果。

值得说明的是，由于我所爬取的网站数量较多，建立的索引约有 120000 个，搜索时得到的结果较多，故只取前 10 个结果项。

搜索主页

搜索结果

2.2 Exp7-1

本次练习中，我们需要进一步美化网页，并加入图片搜索功能。

同时，为了丰富实验内容，我在网页中加入了简单的推荐功能，能够为用户提供基础的网站推荐。

2.2.1 实验步骤

网页结构 为了组织不同的搜索类型，我在原来网站的基础上添加了两个处理类。

```
urls = (
    '/', 'Index',
    '/pic', 'indexPic',
    '/s', 'Result',
    '/p', 'picResult'
)
```

四个处理类分别处理网页搜索主页、图片搜索主页、网页结果页、图片结果页。处理类的定义与之前并没有很大的区别。在此不再赘述。

数据处理 同样地，我们在本次练习当中的数据处理工作和之前也没有什么实质性的区别。

内容呈现 在加入了 DIV/CSS 之后，我们的网页内容的组织效率和美观度有了明显的改观。

在搜索主页当中，我主要针对搜索框和网页/图片切换做了美化，使之更加美观。网页内容整体居中，可定义 CSS 样式如下:

```
<style>
    body
    {
        text-align:center;
    }
</style>
```

而在搜索结果页中，网页的层次更加分明。主要的层次分为顶部搜索框、结果项以及推荐区域。

首先我定义了一个顶部 DIV 样式，在这个空间中放置跳转回主页的超链接，以及搜索框。

```
<div class="head">
    <div class="home">
        <a href="/">
            <--! logo -->
        </a>
    </div>

    <div class="searchbox">
        <form action="/p" method="GET">
            <input id="keyword" name="keyword" type="text" placeholder="Find something">
            <button id="Search" name="Search">Search</button>
        </form>
    </div>
</div>
```

而对于搜索结果项和推荐内容，我们定义一个主体区域放置，将页面分为左右两个部分，左边依次放置结果项，而右边放置我们的推荐内容。每一个结果项都由定义好的样式所填充。定义的样式较多，在此不详细讨论。

值得注意的是，我在 head 中添加了一个 meta 标签，以绕过防盗链。

接下来我主要介绍推荐网页的实现。

内容推荐 由于在本次实验中，我选用的网页为豆瓣上的音乐、图书、电影等内容，而所实现的检索是对于整个豆瓣的网页检索，我们可以在用户查找某一词条项时为其推荐同一类型的其他内容。时间所限，我仅实现了最基础的推荐功能，比如当用户搜索喜剧时，我们可以向他推荐豆瓣的电影专栏，而非图书或音乐专栏。

推荐的主要处理工作可以直接在原先的"getResult" 函数中完成。其思想是找到查询返回结果中所属专栏最多的网页，即"movie.douban.com"、"music.douban.com"、"book.douban.com"三种网页类型。

我定义了如下的辅助函数:

```
def getType(url):
    if url.find('https://movie') == 0:
        return 1

    if url.find('https://music') ==0:
        return 2

    return 3
```

而在"getResult" 中，我添加如下内容以实现推荐:

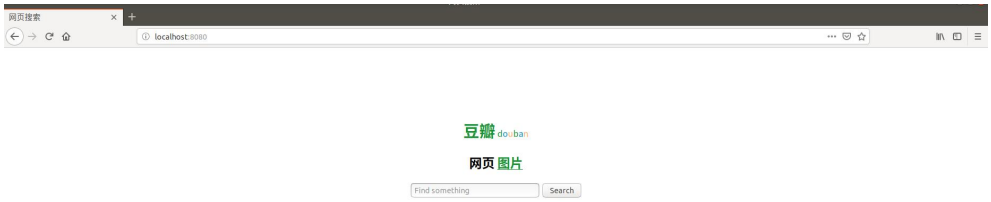
```
# ...

if getType(item['url']) == 1:
    count_movie += 1

else:
    if getType(item['url']) == 2:
        count_music += 1

    else:
        count_book += 1

data.append(item)

if count_movie > count_music:
    if count_movie > count_book:
        type = ["https://movie.douban.com/", u"豆瓣电影"]

    else:
        type = ["https://book.douban.com/", u"豆瓣读书"]

else:
    if count_music > count_book:
        type = ["https://music.douban.com/", u"豆瓣音乐"]

    else:
        type = ["https://book.douban.com/", u"豆瓣读书"]

# ...

return data, type
```

为此我向"result.html" 多传入一个数组 type。其中，我们可以直接读取 type 来完成推荐内容的呈现。

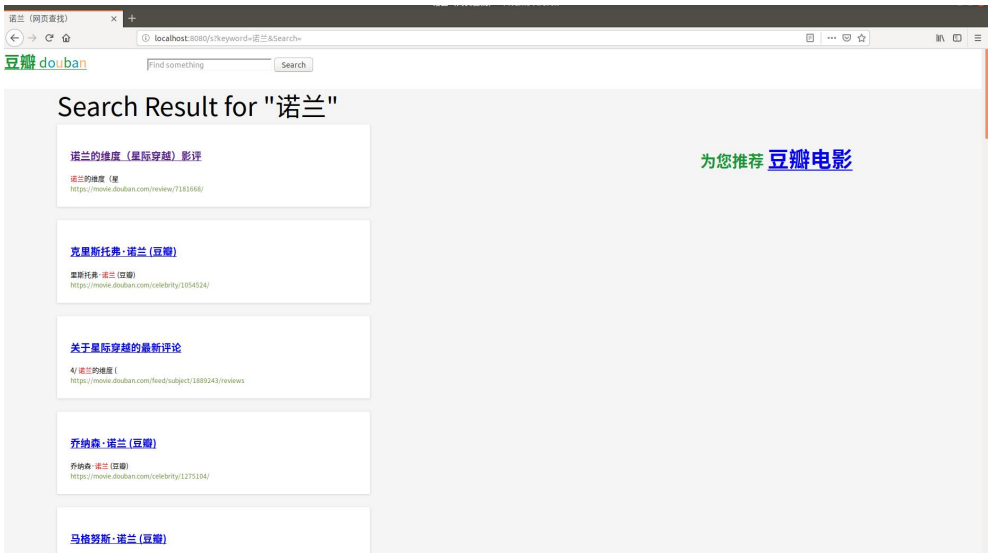
```
<div class="recommendation">
    <div align="center">
        <span style="font-size:30px;font-weight:bold;color:#1b9336;">
            为 您 推 荐
            <span style="font-size:40px;font-weight:bold;color:#6CA6CD;">
                <a href=$type[0]>$type[1]</a>
            </span>
        </span>
    </div>
</div>
```

2.2.2 实验结果

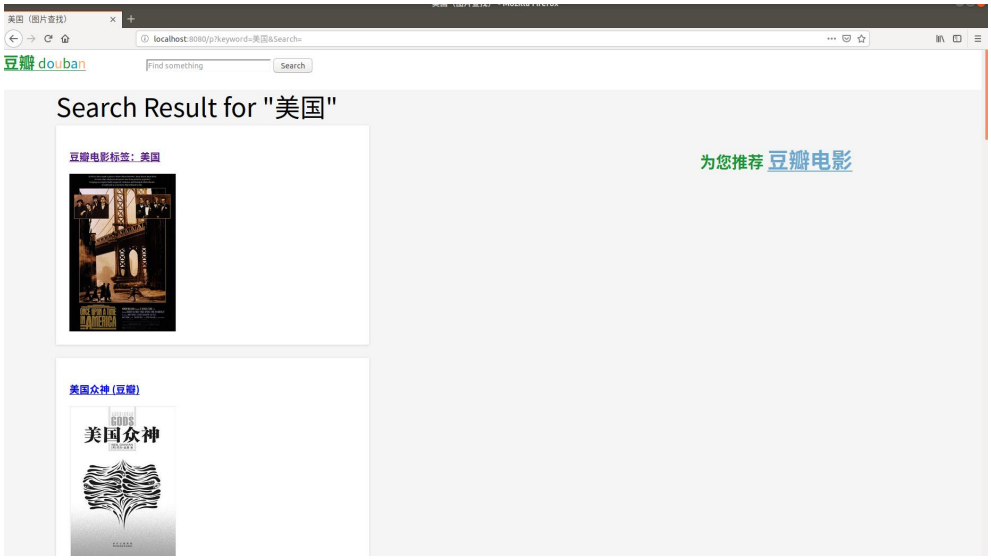
现在网站的功能都已基本实现，美化也得到了完善。



主页面



网页搜索结果



图片搜索结果

搜索结果较为精确，" 美国" 词条的前三项分别为美国专题、美国众神以及美国队长电影。搜索的呈现也比较简洁。

3 实验感想

本次实验中，主要的难点在于网页的内容整合。

对于后端查找获得的信息，前端给予怎样的表现形式十分重要，一个好的展现形式可能比更精确的查找更有价值。