目录

1 实验介绍

1.1 实验内容

1.1.1 实验六

使用 web.py, 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. 搜索结果中要求包含: 标题, 超链接, 关键词上下文以及网址.

1.1.2 实验七

制作一个图片加文字的搜索引擎, 作为中期整合. 即, 在上次的基础上, 加入图片搜索, 使用 css 制定样式.

自主调节 css 中的各项属性参数与 div 布局, 不存在标准答案, 只需根据个人喜好决定页面表示形式和显示哪些细节.

1.2 实验环境

操作系统:Ubuntu 18.04 LTS
Python 版本:3.7
PyLucene 版本:7.4.0
web.py 版本:0.40.dev1

1.3 所需技能

实验六主要涉及使用 Python 框架搭建简单的网站, 用到了 web.py, 需要学习其基本的工作方式与 API. 同时, 需要将之前实验中实现的多个功能模块, 如中文分词与 Lucene 索引, 进行整合使用.

实验七主要在于整合前期工作, 同时合理使用 HTML 和 CSS 布局来使得网页整齐美观. 这一过程需要综合考虑多个模块, 多次尝试调整, 以达到满意的效果.

2 实验过程

2.1 准备实验环境

相较于之前的实验, 本次实验仅需额外安装 web.py 即可准备好实验环境. 但需要说明的是, 由于 web.py 目前处于欠维护状态, 其对 Python 3 的支持并非十分完善. 如果通过 pip 安装, 需手动指定版本, 否则所安装的版本将仅对 Python 2 可用:

```
pip install web.py==0.40-dev1
```

更进一步的, 由于我所使用的 Python 版本为最新的 3.7, 这样安装后的 web.py 依旧存在问题: 框架报错而无法运行, 静态文件访问出错等. 这些问题主要是由于 Python 3.7 的一些变动导致的, 为解决这些问题, 需要手动从 Github 上下载最新的 web.py 源码安装, 如下:

```
git clone git://github.com/webpy/webpy.git
ln -s `pwd`/webpy/web .
```

2.2 后端构建

2.2.1 设置 JVM 与 Lucene

由于 PyLucene 仅仅是 Lucene 的 Python 接口,Lucene 本身是运行于 Java 虚拟机环境 (JVM) 中, 需要在 Python 程序中启动 JVM, 并且在每次执行搜索时调用. 对于 web.py, 在主程序中启动 Web 应用时, 会将当前所有的全局变量, 即 `globals()`, 打包传递给 web.py 程序, 并且在之后的每次响应时从中获取所需变量.

那么, 我们需要在主程序中启动 JVM, 作为全局变量供之后使用. 同时, 为了避免每次搜索时创建 Lucene 搜索器的开销, 可以在程序启动时就创建好搜索器, 同样作为全局变量传递. 如下:

```
if __name__ == '__main__':
    vm_env = lucene.initVM(vmargs=['-Djava.awt.headless=true'])
    base_dir = os.path.dirname(os.path.abspath(sys.argv[0]))
    directory = {
        "html": SimpleFSDirectory(Paths.get(os.path.join(base_dir,
        ↪  HTML_INDEX_DIR))),
        "image": SimpleFSDirectory(Paths.get(os.path.join(base_dir,
        ↪  IMAGE_INDEX_DIR))),
    }
    searcher = {
        "html": IndexSearcher(DirectoryReader.open(directory["html"])),
        "image": IndexSearcher(DirectoryReader.open(directory["image"])),
    }
    analyzer = StandardAnalyzer()
    app = web.application(urls, globals(), autoreload=False) #attention here
    app.run()
```

注意在上面的代码中, 需要禁用 web.py 的自动重载, 否则会导致上述代码失效. 可能的原因在于此种情况下主程序将不再是我们编写的该文件, 而是 web.py 自身, 则 `__name__!='__main__'`, 而是实际的文件名.

在初始化 JVM 后, 只需在每次调用时:

```
vm_env.attachCurrentThread()
```

2.2.2 设置路由

由于需要区分网页搜索和图片搜索 (当然也可以不区分, 只需在搜索时额外传递一个类型参数), 则对二者的 Index 页面各自分配一个处理类 (原因在于这两个类都非常简洁), 但对于搜索结果页面, 我选择共用一个类, 通过 URL 来区分类别 (原因在于处理网页和图片的搜索请求流程基本相同, 可以通过共用类来减少冗余代码). 如下:

```
urls = (
    "/", "Index",
    "/image", "IndexImage",
    "/search/(.+)", "Search",
)

class Index:
    def GET(self):
        return render.index()

class IndexImage:
    def GET(self):
        return render.index_image()

class Search:
    def GET(self, search_type="html"):
        search_data = web.input()
        query_string = search_data.get("s", "")
        if search_type not in ("html", "image"):
            search_type = "html"
        if not query_string:
            return render.index_image() if search_type == "image" else
            ↪  render.index()
        search_result = search_function[search_type](query_string)
        return result_template[search_type](query_string, search_result)
```

注意以上代码的最后两行, 通过表驱动法避免了 if-else 语句的泛滥, 对于目前仅有的两种搜索类型可能优势并不明显, 但可以方便地添加新的搜索类型而无需修改此处逻辑, 有利于日后的功能增加.

2.2.3 网页搜索

对网页的搜索, 在往次实验的基础上, 只需对原有程序作出一些修改, 即可基本实现, 故不再此赘述. 主要的问题在于, 如何实现关键词上下文以及关键词高亮.

让我们稍做分析:Lucene 所创建的索引中, 并未保存网页的完整内容 (`t.setStored(False)`), 而是对其进行提取处理后保存了索引信息 (如倒排索引). 那么, 自然是难以通过 Lucene 直接对关键词进行高亮处理, 除非重新创建索引并且保存网页的内容信息.

那么, 我采用了十分朴素的高亮方式: 打开检索结果对应的本地 HTML 文件, 对其内容进行去 HTML 标签及分词处理, 得到一个词汇列表, 然后将其与输入的搜索字符串分词后的词汇列表对比, 找到第一个重合项, 然后将此项的前后一定数量的词汇合并为所展示的内容摘要. 如下:

```
with open(doc.get("path"), mode="r", encoding="utf8") as file:
    content = file.read()
html2text = HTML2Text()
html2text.ignore_links = True
html2text.ignore_images = True
content = html2text.handle(content)
cutted_content = jieba.cut(content)
flag = False
if cutted_query:
    word_num, cnt = 20, 0
    for x in cutted_content:
        if not x.strip():
            continue
        if not flag and x in cutted_query:
            flag = True
            content = ""
        if flag and cnt < word_num:
            cnt += 1
            content += (x if x not in cutted_query else
                        "<span class='highlight'>{0}</span>".format(x)
                        )
        elif cnt >= word_num:
            break
single_result["content"] = content if flag else content[:100]
```

这样的做法可能有些粗糙, 但效果尚可接受, 在后续的实验中, 必要时需要对其做出修改.

2.2.4 图片搜索

类似的, 图片搜索基本上可以利用之前实验的代码实现, 仅需做出一些简单的修改.

```
def search_image(query_string, result_num=10):
    vm_env.attachCurrentThread()
```

```
    cutted = [x for x in jieba.cut_for_search(query_string) if x.strip()]
    command = " ".join(cutted)
    query = QueryParser("content", analyzer).parse(command)
    scoreDocs = searcher["image"].search(query, result_num).scoreDocs
    result = list()
    for num, scoreDoc in enumerate(scoreDocs):
        doc = searcher["image"].doc(scoreDoc.doc)
        single_result = {
            "url": doc.get("url"),
            "description": doc.get("raw_content"),
        }
        result.append(single_result)
    return result
```

2.3 前端构建

2.3.1 Index 页面

网页搜索的 Index 页面主体部分都是一个文本输入框加上表单提交的按键. 为了使网页整齐, 使用 CSS+DIV 进行布局:

```
<!DOCTYPE html>
<html lang="zh">
<head>
    <meta charset="UTF-8">
    <link rel="stylesheet" href="/static/css/main.css">
    <title> 网页搜索 </title>
</head>
<body>
    <div id="magicWrapper">
    <div id="superCenter" align="center">
        <h1 style="font-size: 60px;" >
            <span>The</span><span class="beacon">Beacon</span>
        </h1>
        <div id="searchType">
            <span class="active"><a href="/"> 网页 </a></span>
            <span class="type"><a href="/image"> 图片 </a></span>
        </div>
        <div id="homeSearch" align="center">
            <form action="/search/html">
                <input type="text" id="homeInput" name="s" placeholder=" 搜点
                ↪  什么吧">
```

```
                <input type="submit" id="homeSubmit" value=" 搜索">
                <br>
            </form>
        </div>
    </div>
    </div>
</body>
</html>
```

具体的 CSS 属性不在此贴出, 最终实现的效果如下:

图 1: 网页搜索 Index 页面

而对于图片搜索, 其布局基本类似, 为了体现图片搜索的特点而有所区分, 对其设置了不同的背景, 效果如下:

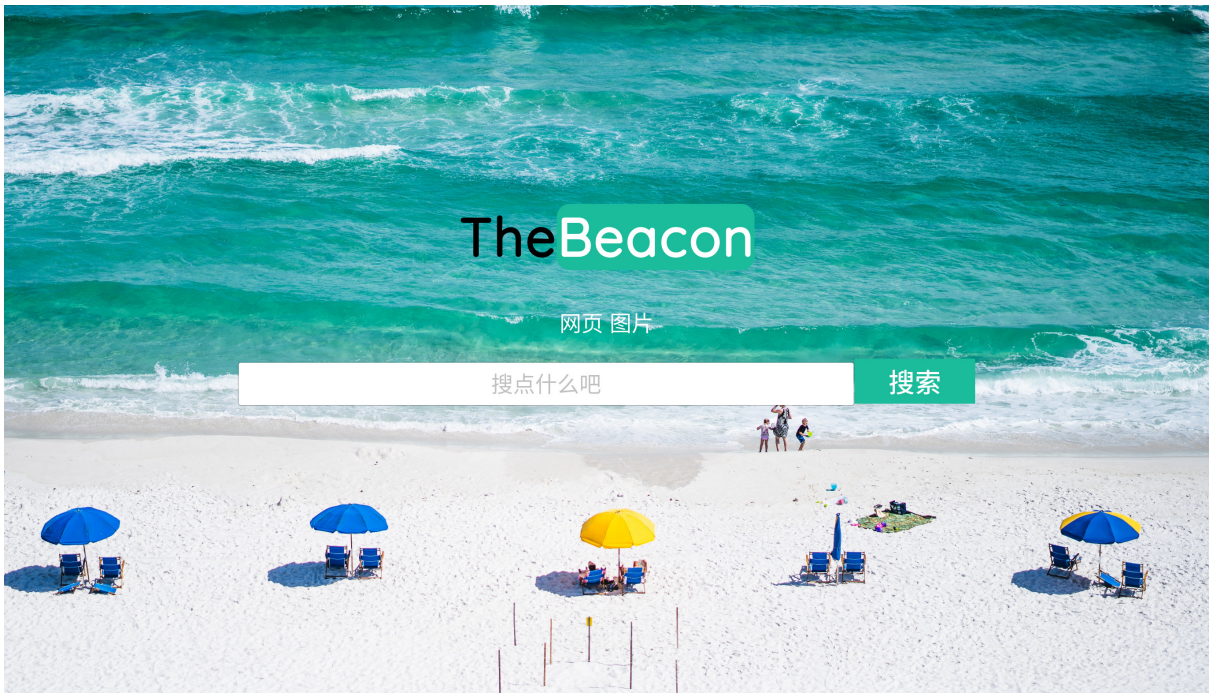

图 2: 图片搜索 Index 页面

2.3.2 网页搜索结果页面

搜索结果页面模板接受 Python 传来的参数: `query_string,search_result`, 类型分别为字符串以及字典组成的列表, 以此生成搜索页面. 则可布局为多个 DIV, 每个 DIV 为一条搜索结果. 同时, 为了方便使用, 可以设置一个始终悬浮于页面最上的输入框, 用于进行新的搜索.

```
$def with (query_string,search_result)
<!DOCTYPE html>
<html lang="zh">
<head>
    <meta charset="UTF-8">
    <link rel="stylesheet" href="/static/css/main.css">
    <title>$query_string 网页搜索结果 </title>
</head>
<body>
    <div class = "floatingBar">
        <form action="/search/html">
            <input type="text" name="s" class="floatingInput" value=
            ↪  "$query_string">
            <input type="submit" class="floatingSubmit" value=" 搜索">
        </form>
    </div>
    <div class = "resultContainer">
        $for item in search_result:
        <div class = "item">
```

```
            <div class = "title">
                <a href="$item['url']">$item["title"]</a>
            </div>
            <div class = "content">$:item["content"]</div>
            <div class = "url">
                <a href="$item['url']">$item["url"][:80]</a>
            </div>
        </div>
    </div>
</body>
</html>
```

用到的主要 CSS 属性如下:

```
div.item{
    margin-bottom: 10px;
    padding: 20px 15px 15px;
    border-radius: 5px;
    background-color: #fff;
    box-sizing: border-box;
    box-shadow: 0 0 20px 2px rgba(0, 0, 0, .1);
    -webkit-box-shadow: 0 0 20px 2px rgba(0, 0, 0, .1);
    -moz-box-shadow: 0 0 20px 2px rgba(0, 0, 0, .1);
}



.floatingBar{
    position: fixed;
    top: 10px;
    width:80%;
    margin-left: 10%;
    margin-right: 10%;
}
```

最终效果如图所示:

图 3: 网页搜索结果

2.3.3 图片搜索结果页面

对于图片的搜索结果, 每张图片都配有一定的描述文字, 可将其放在一个 DIV 中, 然后对所有的 DIV 进行排列. 一种比较美观的排列方式为瀑布流, 这部分的 CSS 布局代码参考了网上的代码.[1]

则 HTML 模板的主要部分形如:

```
<div class = "waterfall">
    $for item in search_result:
    <div class = "item">
        <img src="$item['url']">
        <div class = "url">$item["description"][:80]</div>
    </div>
</div>
```

最终效果如下:

[1] https://blog.csdn.net/m0_37568521/article/details/78487568

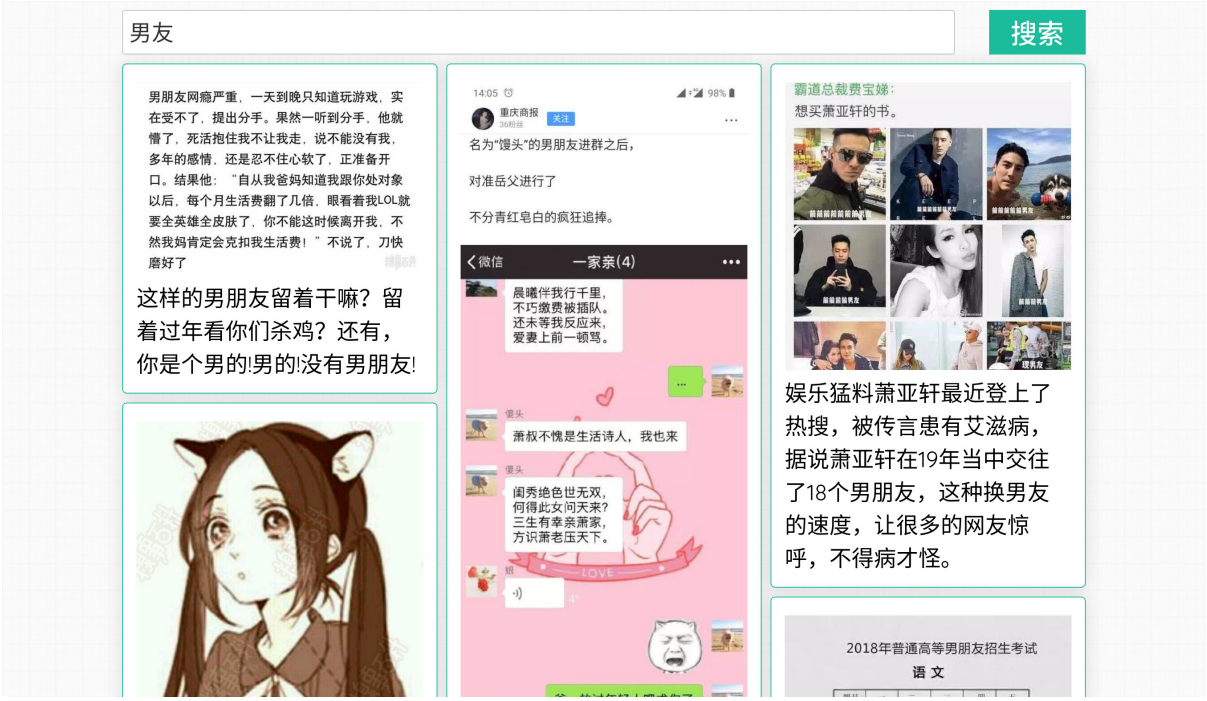
男友
搜索
男朋友网瘾严重，一天到晚只知道玩游戏，实在受不了，提出分手。果然一听到分手，他就慌了，死活抱住我不让我走，说不能没有我，多年的感情，还是忍不住心软了，正准备开口。结果他：“自从我爸妈知道我跟你处对象以后，每个月生活费翻了几倍，眼看着我LOL就要全英雄全皮肤了，你不能这时候离开我，不然我妈肯定会克扣我生活费！”不说了，刀快磨好了
这样的男朋友留着干嘛？留着过年看你们杀鸡？还有，你是个男的!男的!没有男朋友!
14:05
98%
重庆商报
关注
名为“馒头”的男朋友进群之后，
对准岳父进行了
不分青红皂白的疯狂追捧。
微信
一家亲(4)
晨曦伴我行千里，
不巧缴费被插队。
还未等我反应来，
爱妻上前一顿骂。
馒头
萧叔不愧是生活诗人，我也来
馒头
闺秀绝色世无双，
何得此女问天来?
三生有幸亲萧家，
方识萧老压天下。
LOVE
霸道总裁费宝娣：
想买萧亚轩的书。
娱乐猛料萧亚轩最近登上了热搜，被传言患有艾滋病，据说萧亚轩在19年当中交往了18个男朋友，这种换男友的速度，让很多的网友惊呼，不得病才怪。
2018年普通高等男朋友招生考试
语 文

图 4: 图片搜索结果

3 总结与分析

本次实验将之前实验内容进行了整合, 在这一过程中, 我发现了以前所写代码的一些缺陷, 诸如不能灵活配置参数, 过度依赖上下文等, 这些都阻碍着对其的整合. 因此, 在今后的编程中, 更需要注意模块化的设计思想.

此外, 在编写 HTML 和 CSS 代码时, 由于没有正式地学习过前端知识, 常常遇到一些难以处理的细节问题, 需要反复尝试并查找各种资料, 耗费了不少时间, 但也加深了对其的理解与掌握.