Experiment 4/5

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)。

在进行实验前，我在 Python 上部署了实验所需的 jcc 和 pylucene(版本号 4.9.0) 模块。

jcc 模块的功能是生成 C++ 代码并使 Python 能够调用 Java 代码。而 pylucene 则是 Lucene 在 Python 上的 API，能够用来建立完整的全文检索结构。

1.2 实验目的

在前两次实验中，我已经实现了从根网页出发，爬取有关网站上的网页内容的功能。但我爬取这些网站之后更关键的步骤在于，我要能够从这些纷繁复杂的内容当中提取出有价值的信息，并加以利用。这便是此次试验的目的——对网页信息进行检索。

在实验中，我首先尝试着对需要检索的文本进行索引，并将这些信息组织成多个文档以及每个文档下的 Field，然后实现检索功能。在这个过程中，我能够了解到索引的实现机制，并接触分词的相关技术细节。同时，这也有助于我消化对爬取到的原网页进行诸如编码处理的相关知识。

接下来，我实现更为精确的查找功能。我能够对搜索网站进行限制，并能够通过文本内容查找相关得图片。这有助于进一步提升我的信息检索能力。

1.3 实验原理

本次实验的核心内容为 Lucene 的使用。为此，我将着重围绕这一全文检索库介绍倒排索引的结构和构建、布尔检索模型两大内容。

1.3.1 倒排索引的结构和构建

倒排索引的组织结构大致由两部分组成：

- 词项字典
- 全体倒排记录表

两者中的元素一一对应，即词项字典中的每一个词项都有一个记录出现该此项的所有文档的列表，该表中的每个元素记录的是词项在某文档中的一次出现信息以及词项在文档中出现的位置，这个列表即成为倒排记录表。词典按照字母顺序进行排序，而倒排记录表则按照文档 ID 号进行排序。

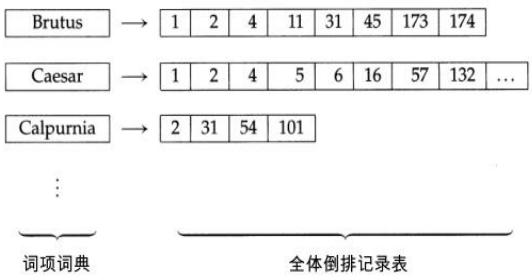


图 1-3 倒排索引的两个部分。词典部分往往放在内存中，而指针指向的每个倒排记录表则往往存放在磁盘上

建立索引的大致步骤如下：

1. 收集需要建立索引的文档
2. 将每篇文档转换成一个个词条的列表，即词条化操作
3. 进行语言学预处理，产生归一化的词条作为词项
4. 对所有文档按照其中出现的词项来建立倒排索引，索引中包括一部词项字典和一个全体倒排记录表

其中，在解决 (2) 的词条化问题时，使用东亚语言 (词之间不存在空格) 的我常常会求助于分词机制。

分词 在这里，我简单地讨论一下分词的主流方法。

分词的方法包括基于词典的最大匹配法（采用启发式规则来进行未定义词识别）和基于机器学习序列模型的方法（如隐马尔可夫模型或条件随机场模型）等，后者需要在手工切分好的语料上进行训练。由于存在多种切分可能，上述分词方法都有可能导致错误的切分结果，因此，永远不能保证只能得到一个完全一致的唯一切分结果。另一个解决方法则摒弃了基于词的索引策略而采用短字符序列的方法（如字符的 k-gram 方法）。这种方法并不关心词项是否会跨越词的边界。

1.3.2 布尔检索模型

在建立的我的索引结构后，我便可以尝试着进行查询操作。

在这里我重点介绍的是" 与" 查询。在很多情况下，查询往往是由简单的" 与" 操作构成的。

对于两个词条 (A, B) 的与的结果，我有如下分析：

1. 在词典中定位 A
2. 返回其倒排索引表
3. 在词典中定位 B
4. 返回其倒排索引表
5. 对两个倒排索引表求交集

其中，交集操作十分关键，这是因为我必须快速将倒排记录表求交集以尽快找到哪些文档同时包括两个词项。

一种简单的实现方法是对每个有序列表都维护一个位置指针，并让两个指针同时在两个列表中后移。每一步我都比较两个位置指针所指向的文档 ID，如果两者一样，则将该 ID 输出到结果表中，然后同时将两个指针后移一位。如果两个文档 ID 不同，则将较小的 ID 所对应的指针后移。

对于更为复杂的查询，进一步优化的方法是将个词项按照词项的文档频率（也就是倒排记录表的长度）从小到大依次进行处理，如果我先合并两个最短的倒排记录表，那么所有中间结果的大小都不会超过最短的倒排记录表，这样处理所需要的工作量很可能最少。

2 实验过程

2.1 Exp4-1

在练习 4-1 中，我被要求对我爬取下来的网页建立索引并进行检索。

2.1.1 实验步骤

创建索引 我将关注的重点放在"IndexFiles" 类的成员函数'indexDocs' 上，这是我定义索引方式的地方。该函数接受三个参数'root'，'writer'，'urlDic'。其中'root' 是存放待索引文件的文件夹，'writer' 是用来写索引文件的'IndexWriter' 对象，而'urlDic' 则是存放文件名与真实url 一一对应的字典。它定义在'indexDocs' 之前，以便于之后索引 Field'url'。

我定义了一个函数'getUrlDic'。该函数提取我之前爬取的网址信息'index.txt'，返回所需的字典，即传入'indexDocs' 的'urlDic'

```
def getUrlDic(filename):
file = open(filename, 'r')
urlDic = {}
for line in file.readlines():
    line = line.replace('\n', '')
    line = line.split('\t')
    urlDic[line[1]] = line[0]

file.close()

return urlDic
```

在'indexDocs' 中，我首先定义三个 Field 类。

t1 是"name"，"path"，"title"，"url" 的 FieldType 类型。由于这四种 Field 不出现在查询当中，仅作为返回结果显示使用，故我不建立倒排索引和分词，但完整存储其内容。

t2 是"contents" 的 FieldType 类型。它需要被查询，因此我设置了倒排索引和分词，但不完整储存其内容。

```
t1 = FieldType()
t1.setIndexed(False)
t1.setStored(True)
t1.setTokenized(False)

t2 = FieldType()
t2.setIndexed(True)
t2.setStored(False)
t2.setTokenized(True)
t2.setIndexOptions(FieldInfo.IndexOptions.DOCS_AND_FREQS_AND_POSITIONS)
```

接着，我么读取之前存储的文档信息。方便起见，我仅读取文件夹中的所有以"html"、"htm"、"com"、"cn" 结尾的文件。

```
for root, dirnames, filenames in os.walk(root):
    for filename in filenames:
        if not filename.endswith('.htm') and not filename.endswith('.html') and not
                                                    filename.endswith('.com') and not
                                                    filename.endswith('.cn'):
            continue
        print "adding", filename
```

我从这些文件中进一步处理，得到我想要的 Field 内容。

其中，对于"url"，我使用了之前传入的字典进行匹配。对于"title" 和"contents"，我则首先用"BeautifulSoup" 处理为 BeautifulSoup 对象，进而使用其内置的方法得到网页的"title" 和预

处理后的"contents"。对于"contents"，我最后又使用了"jieba" 库的分词系统为其分词。这里我使用了'WhitespaceAnalyzer' 作为后续的分词器。

处理完成后，我即可将这些 Field 写入文档。

```
try:
    path = os.path.join(root, filename)

    url = urlDic[filename]

    file = open(path)
    contents = file.read()
    file.close()

    soup = BeautifulSoup(contents, features='html.parser')
    title = soup.title.string
    title = unicode(title).encode('utf-8')
    title = title.replace("\n", '')

    contents = soup.get_text().encode('utf-8')
    seg_list = jieba.cut(contents)
    contents = " ".join(seg_list)

    doc = Document()
    doc.add(Field("name", filename, t1))
    doc.add(Field("path", path, t1))
    doc.add(Field("title", title, t1))
    doc.add(Field("url", url, t1))
    if len(contents) > 0:
        doc.add(Field("contents", contents, t2))
    else:
        print "warning: no content in %s" % filename
    writer.addDocument(doc)
except Exception, e:
    print "Failed in indexDocs:", e
```

值得一提的是，由于我使用了"BeautifulSoup" 来处理网页文件，而之前爬取的文件是 string 类型存储的，我不需要提取出网页中的"charset" 来进行编码。但是我不妨实现这个功能，以便后续需要。

函数'get_charset' 基于正则表达式查找网页的"charset" 信息。

```
def get_charset(content):
    pattern = re.compile(r'charset=([0-9a-zA-Z\_\-\.\/]*)')
    result = pattern.search(content)
    return result.group(1)
```

"main" 函数中，我先初始化 JAVA 虚拟机，然后即可开始创建索引。

```
if __name__ == '__main__':
    dic = getUrlDic('index.txt')

    lucene.initVM(vmargs=['-Djava.awt.headless=true'])
    print 'lucene', lucene.VERSION
    start = datetime.now()
```

```
    try:
        IndexFiles('html', "index", 'index.txt')
        end = datetime.now()
        print end - start
    except Exception, e:
        print "Failed: ", e
        raise e
```

搜索程序 搜索程序的实现相对比较简单。由于在"lucene" 中已经封装好了'QueryParser' 方法和一些 searcher，我只需要将输入的内容转为符合格式的查询 Field 和查询内容即可。

而在这次的练习中,我仅针对 contents 进行搜索,所以我只需将输入传递给'QueryParser'。对查询到的结果，我显示它的"path" 等信息。

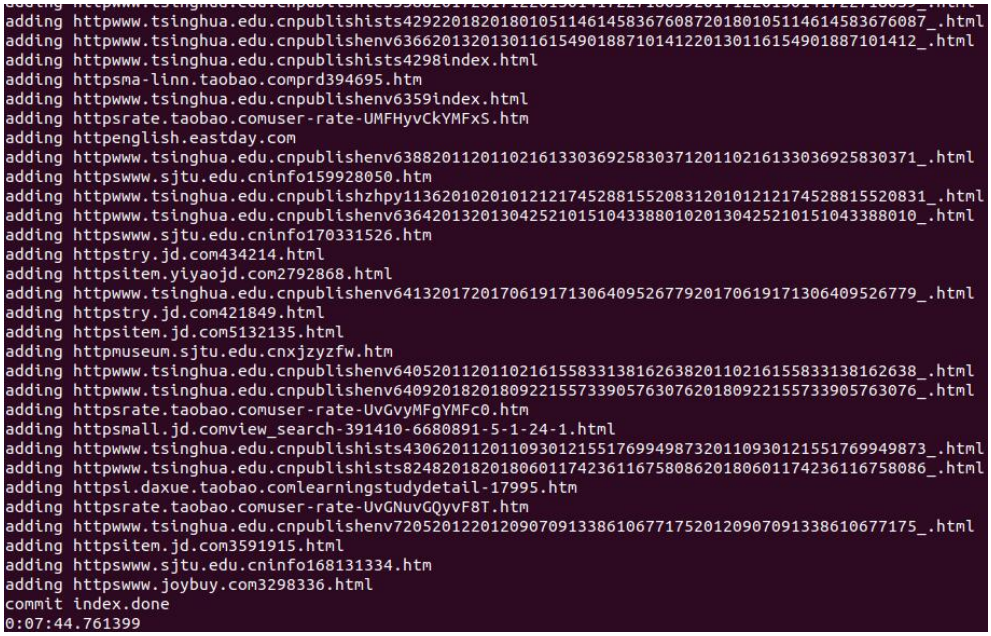
```
def run(searcher, analyzer):
    while True:
        print
        print "Hit enter with no input to quit."
        command = raw_input("Query:")
        command = unicode(command, 'utf-8')
        if command == '':
            return

        print
        print "Searching for:", command
        query = QueryParser(Version.LUCENE_CURRENT, "contents",
                            analyzer).parse(command)
        scoreDocs = searcher.search(query, 10).scoreDocs
        print "%s total matching documents." % len(scoreDocs)

        for i, scoreDoc in enumerate(scoreDocs):
            doc = searcher.doc(scoreDoc.doc)
            print 'path:', doc.get("path"), 'title:', doc.get("title"), \
                'url:', doc.get("url"), 'name:', doc.get("name")
```

2.1.2 实验结果

我成功地建立对绝大多数文档建立了索引，并能够查询到制定的内容。



索引建立

搜索程序

2.2 Exp5-1

本练习中，我将实现基本的组合查询，对搜索的网站进行主域名的限制。

2.2.1 实验步骤

更新索引 由于这次我需要针对主域名进行组合查询，我需要在索引中对每个文档加入"site"的 Field。这样的更新其实相当于重新创建整个索引，因此实际上我操作时是直接在原有的创建索引方法上加上"site" 这个 Field 再重新索引。但我在这里也给出更新索引的方法。

我定义个一个类"IndexUpdate" 来完成索引工作。类下的方法包括'testDelete'，'testAdd' 等。

'testDelete' 函数接受一个需要指定搜索的 Field，和相应的搜索内容。Term 对象用于定位文档，找到在 Field 中含有 searchString 的文档。

```
    def testDelete(self, fieldName, searchString):
        config = IndexWriterConfig(Version.LUCENE_CURRENT, self.getAnalyzer())
        config.setOpenMode(IndexWriterConfig.OpenMode.APPEND)
        writer = IndexWriter(self.dir, config)
        writer.deleteDocuments(Term(fieldName, searchString))
        writer.close()
```

根据我的需要，'testAdd' 函数接受保存指定添加的文档内容的文件路径、之前提到的存放文件名与真实 url 一一对应的字典，以及根目录。

```
def testAdd(self, filepath, urlDic, root):
    # ...
```

函数的主体与之前的索引创建相仿，故在此略过。这里只介绍"site" 的提取工作。

我用到"urllib" 的'splittype' 和'splithost'，切分 Url，得到 site。

```
proto, rest = urllib.splittype(url)
site, rest = urllib.splithost(rest)
```

之后，我定义了第三个 FieldType t3，用来存放 site。这种类型需要被索引和储存，但无需分词。

```
t3 = FieldType()
t3.setIndexed(True)
t3.setStored(True)
t3.setTokenized(False)
# ...
doc.add(Field("site", site, t3))
# ...
```

搜索程序 为了实现组合查询功能，我需要首先能够处理输入的格式，转换为对应的查询语句。

'parseCommand' 完成了原始输入转化到查询语句的过程。它接受一串字符，返回布尔查询的字典。它首先确定了能够组合查询的 Field，然后对输入进行切分。

```
def parseCommand(command):
    allowed_opt = ['site']
    command_dict = {}
    opt = 'contents'
    for i in command.split(' '):
        if ':' in i:
            opt, value = i.split(':')[:2]
            opt = opt.lower()
            if opt in allowed_opt and value != '':
                command_dict[opt] = command_dict.get(opt, '') + ' ' + value
        else:
            command_dict[opt] = command_dict.get(opt, '') + ' ' + i
    return command_dict
```

而搜索程序的主体和之前 Exp4-1 中的程序并没有很大的区别。我把切分好的查询语句构造为两个"QueryParser" 对象，分别指向内容和 site，并把它们都添加到布尔容器"BooleanQuery" 中，取两者的交集。

```
def run(searcher, analyzer):
    while True:
        print
        print "Hit enter with no input to quit."
        command = raw_input("Query:")
        command = unicode(command, 'utf-8')
        if command == '':
            return
```

```
        print "Searching for:", command

        command_dict = parseCommand(command)
        querys = BooleanQuery()
        for k, v in command_dict.iteritems():
            print k, v
            query = QueryParser(Version.LUCENE_CURRENT, k,
                                analyzer).parse(v)
            querys.add(query, BooleanClause.Occur.MUST)
        scoreDocs = searcher.search(querys, 10).scoreDocs
        print "%s total matching documents." % len(scoreDocs)

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

            print "------------------------"
            print 'path:', doc.get("path")
            print 'name:', doc.get("name")
            print 'title:', doc.get('title')
            print 'url:', doc.get("url")
```

2.2.2 实验结果

我现在能够搜索指定 host 内的网站内容。
以搜索某一淘宝中的球鞋内容为例，可以得到如下结果：

```
Hit enter with no input to quit.
Query:球鞋 site:42shoes.taobao.com
Searching for: 球鞋 site:42shoes.taobao.com
site  42shoes.taobao.com
contents  球鞋
5 total matching documents.
------------------------
path: html/https42shoes.taobao.com
name: https42shoes.taobao.com
title: 首页-42运动家 SNEAKER-淘宝网
url: https://42shoes.taobao.com
------------------------
path: html/https42shoes.taobao.comindex.htm
name: https42shoes.taobao.comindex.htm
title: 首页-42运动家 SNEAKER-淘宝网
url: https://42shoes.taobao.com/index.htm
------------------------
path: html/https42shoes.taobao.comprd369217.htm
name: https42shoes.taobao.comprd369217.htm
title: 售后及退换货流程-42运动家 SNEAKER-淘宝网
url: https://42shoes.taobao.com/p/rd369217.htm
------------------------
path: html/https42shoes.taobao.comprd119846.htm
name: https42shoes.taobao.comprd119846.htm
title: 真假及常见问题-42运动家 SNEAKER-淘宝网
url: https://42shoes.taobao.com/p/rd119846.htm
------------------------
path: html/https42shoes.taobao.comprd263725.htm
name: https42shoes.taobao.comprd263725.htm
title: 会员中心-42运动家 SNEAKER-淘宝网
url: https://42shoes.taobao.com/p/rd263725.htm
```

搜索程序

2.3 Exp5-2

本次实验中，我在原有网站中提取图片网址和周围的文本内容，实现一个图片索引。

出于兴趣原因，我这次选择虎扑论坛作为目标爬取网站进行分析。

2.3.1 实验步骤

首先，我对虎扑论坛的网页结构进行了一定分析。

虽然很容易实现提取工作，但在用户的论坛发帖当中，图片并没有保存相关得 title 或 alt 信息。鉴于论坛的回帖机制，除了可以提取网页标题和图片周围文本内容之外，我们可以通过检索回帖文本来获得额外的信息。

进一步分析，由于在该论坛内用户回帖都由统一的标签约束，我们可以很容易地找到相应内容。

创建索引 和之前类似，我还是先对对目录中的每一个文件进行信息提取。这次我对 Exp1 中的'parseIMG' 做了稍许改动，使其能够找到网页中的所有"img" 标签，并提取出其"src" 属性和与该标签相近的文本内容。

函数是根据该论坛的网页结构设计的。该论坛的帖子下的图片都包含在一对" "," " 中，然后再往外的 parent 标签分为两种：若为主楼，则 p 标签的 parent 为一对 class 为"quote-content" 的 div 标签；若为回帖，则 parent 为一对"<td>"，"</td>"。

找到该 parent 后，我们不难用"BeautifulSoup" 中 tag 对象的 strings 成员来遍历其下的所有文本内容。这些文本和图片是同一级的，且有很大的关联度。

此外，我们也将可能包含的图片 alt 信息添加到该图片的文本信息中。

```
def parseIMG(content, page):
imgset = {}
soup = BeautifulSoup(content, features='html.parser')
for link in soup.findAll('img'):
    text = ""

    src = link.get('src', '').encode('utf-8')
    alt = link.get('alt', '').encode('utf-8')
    text += alt
    parent = link.parent.parent
    try:
        if str(parent.name) == "td" or \
                (str(parent.name) == "div" and \
                str(parent['class'][0]) == "quote-content"):
            for string in parent.strings:
                text += string.replace(' ', '').encode('utf-8')
    except:
        pass

    imgset[src] = text.replace(' ', '').replace('\n', '')

return imgset
```

返回的结果是一个 imgurl-imgtext 的词典。利用这个函数，足够在 Exp4 的基础上建立图片索引。

索引建立过程和之前十分类似，在此不再赘述，仅说明 title 信息也包含在了 imgtext 中，imgtext 作为搜索内容进行分词、索引，其他信息仅完整保存。

```
# ...

url = urlDic[filename]

file = open(path)
contents = file.read()
file.close()

soup = BeautifulSoup(contents, features='html.parser')
title = soup.title.string
title = unicode(title).encode('utf-8')
title = title.replace("\n", '')

img = parseIMG(contents, url)

for imgurl, imgtext in img.items():
    if len(imgurl) < 5 or len(imgurl) > 500:    # simple filter
        continue
    imgtext += title
    seg_list = jieba.cut(imgtext)
    imgtext = " ".join(seg_list)

    doc = Document()
    doc.add(Field("imgurl", imgurl, t1))
    doc.add(Field("title", title, t1))
    doc.add(Field("url", url, t1))
    if len(imgtext) > 0:
        doc.add(Field("imgtext", imgtext, t2))
    else:
        print "warning: no content in %s" % filename
    writer.addDocument(doc)

# ...
```

搜索程序 搜索程序针对 imgtext 进行检索，返回相应结果。在此不再分析具体代码细节。

2.3.2 实验结果

程序能够根据文件内容建立索引，并执行搜索。

```
adding httpsbbs.hupu.com24007383.html
adding httpsbbs.hupu.com24041385.html
adding httpsbbs.hupu.com22749006.html
adding httpsbbs.hupu.com21083591.html
adding httpsbbs.hupu.com23906379-3.html
adding httpsbbs.hupu.com5048053-2.html
adding httpsbbs.hupu.com24000776_170652846900897.html
adding httpsbbs.hupu.com23866073-3.html
adding httpsbbs.hupu.com23428609.html
adding httpsbbs.hupu.com23879505.html
adding httpsbbs.hupu.com24045453.html
adding httpsbbs.hupu.com24035985.html
adding httpsbbs.hupu.com24019352_179328800552738.html
adding httpsbbs.hupu.com24040000_201037379931033.html
adding httpsbbs.hupu.com24045305_185430614110975.html
adding httpsbbs.hupu.com24021701.html
adding httpsbbs.hupu.com19654371.html
adding httpsbbs.hupu.com23593321-3.html
adding httpsbbs.hupu.com24043103.html
adding httpsbbs.hupu.com24000743_36678182950463.html
adding httpsbbs.hupu.com24016671-2.html
adding httpsbbs.hupu.com19480592_236124962254825.html
adding httpsbbs.hupu.com2807968-3.html
adding httpsbbs.hupu.com1412963.html
adding httpsbbs.hupu.com23584004-16.html
commit index..done
0:25:21.170875
```

创建索引

```
Hit enter with no input to quit.
Query:AF1

Searching for: AF1
30 total matching documents.

imgurl: https://i1.hoopchina.com.cn/user/356/28285356/28285356-1495419895.jpg@45h_45w_2e
url: https://bbs.hupu.com/23964540_25968507529372.html
urltitle: 各位大佬求类似富尔茨这条牛 - 我上过af1 - 潮流区 - 虎扑社区
------------------------------------------------------------------------
imgurl: https://i1.hoopchina.com.cn/user/default_small.jpg
url: https://bbs.hupu.com/23982582_217472468400665.html
urltitle: 求鉴定af1小麦高帮37.5 - 爱琴海的蓝 - 鉴定中心 - 虎扑社区
------------------------------------------------------------------------
imgurl: https://i1.hoopchina.com.cn/user/default_small.jpg
url: https://bbs.hupu.com/24003367_218763115438266.html
urltitle: 新入无盒af1小麦高帮求鉴定 - 菜鸟一ming - 鉴定中心 - 虎扑社区
------------------------------------------------------------------------
imgurl: https://i1.hoopchina.com.cn/user/default_small.jpg
url: https://bbs.hupu.com/23999808.html
urltitle: 求双af1小麦high 42码 - 求购区 - 虎扑社区
------------------------------------------------------------------------
imgurl: https://i1.hoopchina.com.cn/user/795/17619795/17619795_small_20.jpg
url: https://bbs.hupu.com/23999808.html
urltitle: 求双af1小麦high 42码 - 求购区 - 虎扑社区
```

搜索程序

3 实验感想

3.1 GZIP 压缩网页 & 动态网页

在实验过程中，我本来的设想是爬取京东的商品信息，但我遇到了两个问题：

- 爬取商品信息页时，返回的 content 为空。

- 返回的 content 不为空时，也有许多内容被折叠。

经分析，我之所以遇到这样的问题，一个是因为部分网站经过了 GZIP 压缩，一个是因为网页中存在大量的动态内容。

3.1.1 GZIP 压缩网页

HTTP 协议上的 GZIP 编码是一种用来改进 WEB 应用程序性能的技术。大流量的 WEB 站点常常使用 GZIP 压缩技术来让用户感受更快的速度。

经过 GZIP 压缩的网页，在读取时需要进行解压操作。

最简单的方式是使用"GZIP" 中的"decompress" 方法，但可惜的是在 python2.7 下并没有此模块。

查找有关资料知，"Requests" 库能够自动检测是否开启 GZIP 压缩并解压。因此我实现了另外一个版本的'get_page' 函数，以解决此问题。

```
def get_page(page):
    # ...

    req = requests.get(page)
    req.encoding = 'utf-8'

    content = req.text

    # ...
```

3.1.2 动态网页

目前有相当多的网站通过 js 动态加载网页内容，而这一部分内容的读取只通过 urllib 是无法实现的。

为此，我们需要进一步实现爬虫的功能，运用 json 来帮助我们实现动态网页内容的读取。

3.2 改进的爬虫实现

在之前的实验中，我通过字符串搜索进行网页内"href" 的搜集。由于没有进行相应的查询，我没有发觉我爬取网页的主要函数"get_all_links" 爬取功能并不理想。爬取的网页数量较少，且网址格式相近。

通过分析我使用的正则表达式发现，之前的表达式的匹配范围十分有限。

```
# ...

pattern = re.compile(r'\bhref=\"([0-9a-zA-Z\_\-\.\/]*)\"\s')

# ...
```

它只接受一定范围的 href 字符，且结尾被限定为空白符，这过滤掉了许多内容。

改进后的正则表达式仅仅过滤掉诸如"javascript void(0)" 以及"#" 这样的无意义的网址，并且允许结尾为">"，以匹配更多格式的 tag。

```
    # ...

    pattern = re.compile(r'\bhref=\"([^#\":]*)\"[\s>]')

    # ...
```

改进后，在爬取淘宝商品信息时匹配的内容增加了约 20 倍。这提示了我需要针对网页格式不断赶紧爬取功能，并在爬取不同类型的网站时对症下药，达到爬取的最大能力。

References

[1] Christopher D.Manning / Hinrich Schütze / Prabhakar Raghavan. Introduction to Information Retrieval