# 爬虫下载PDF并且处理中文分词

## 思路

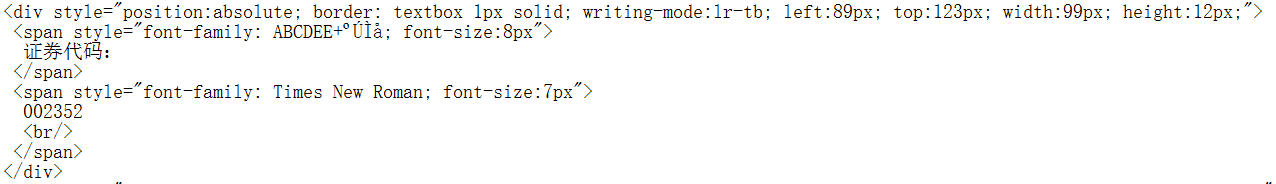
1. 首先使用useDownloadList.py，下载downlist.txt中的pdf文件到pdfStores文件夹，我在pdfStore中提供了几个下载好的pdf文件
2. Dir文件夹是一个package，包含所有的工具函数。所有的工具函数基本都在searchingTool.py，所以在searchControl.py前面要用from bin.searchingTool import \*导入所有函数\。所有py文件的执行都是在cmd中，cd到pdf处理集合文件夹执行
3. 安装pdfminer3k, 利用dir文件夹中convertPdf.Py中的pdf2html函数把pdf转成html
4. 安装BS4，可以利用beautifulSoup的prettify 函数输出漂亮的html文件（进行正确的indent），在txtAndXls中的htmlPretty.py中实现功能
5. 利用dir中htmlFixer.py文件，文件的最下面有单独处理一个html的代码，把html分解成txt和excel，见下面分解原理
6. 使用fixTextControl.py对txtAndXls中的txt进行细节修复（见下面）,并把修复的txt存储在fixedTrainingTxt文件夹
7. 对txt文件进行全文搜索,whoosh+jieba。使用searchControl.py文件，querys可以写入多个关键词，进行迭代寻找，在上一个关键词的筛选结果中继续寻找
8. 正则匹配搜索:reSearchControl.py

其实这是个笔记文档，利用搜索关键词找到你想找的东西，可能不太全，主要借助百度2333



### 分解原理

★观察html文件中可以看到每个文字块都是利用html中的div标签确定其块的宽度（width）高度(height)，起始行(top)和起始列(left)位置：（html的知识在“html内容详解”）



★因此如果是表格，肯定有多个div块的top是相等的，left不同

★继续观察发现有的表格是top稍微不同，他们占据的块的高度是从top到top+height，所以认为top到top+height有重合的div块是属于同一个表格行。

★没有重合的属于txt，每个文档段用’\n---------\n’分割开，有重合的按left从左到右放入xls文件

### Txt的细节修复

1. , 主要是去除了page页码后把跨页的文档段融合。并且发现如果在pdf中某行文字中包含数字（年月日元股）会导致该行被列为单独的文档段，形成语义分割，使用了正则表达式识别并且融合。

### 全文搜索原理

1. 利用前面设置的’\n---------\n’分隔符，用re.split(’\n---------\n’, txtString) 分解整个fixed.txt文件的字符串为一段段的中文自然段，存储为字符串列表，然后利用whoosh搜索引擎+jieba分词器对列表中的每个字符串进行分词后的匹配
2. ReSearchControl.py的搜索方式是：利用分隔符得到字符串列表，对每个字符串进行正则匹配，为了提高效率才用map函数并发，可以观察ReSearchControl.py中的parallelRe函数，map函数原理可以在这个word文件中搜索关键词map
3. 因为从html转换出的txt自然段，经过fixText处理，仍然是一个自然段有多个句子，所以在搜索的过程中会把自然段分割成句子，在reSearchControl中，传入walkAll的spliter是句号和’\n---------\n’分隔符。
4. 如果有分号的情况，在自然段分割为句子时会重组句子，比如：

下面责任未履行：

1. xxx;2.yyy;3.zzzz;

会重组为：

下面责任未履行：1.xxx;

下面责任未履行：2.xxx;

下面责任未履行：3.xxx;

在reSearchControl中的rearrangeStr实现这个功能。

## 如何调用另外py文件的函数

from dir.pruneText import \*#即使跟prunText在同一文件夹，仍然需要通过dir package去调用另一个py文件的全部函数

如果是在文件中有if \_\_name\_\_ == '\_\_main\_\_': 则cmd执行该文件，只会执行该处的内容。如果该文件是被别的py文件import的，则不会执行该处文件

## 利用Pdfminer将pdf转成html和txt

### 下载安装

<https://pypi.python.org/pypi/pdfminer3k/>

python3 版本

python软件包的安装方法：

打开cmd，输入下面命令

cd dirName

dirName把当前文件夹到解压安装包后包含的setup.py目录

python setup.py install

### 转换成html

#### 利用tool文件夹下的pdf2txt

Cmd命令，必须cd到tool文件夹：

python pdf2txt.py -o output.html pdfName

pdfName是要转换的pdf文件名，

#### 按照pdf2txt自己编写

Bin中convertPdf.py, 要使用只需要在自己的python代码前加上

From bin.convertPdf import all

Pdf2html(pdfpath, htmlPath)

Code：

import sys

import io

# import getopt

from pdfminer.pdfinterp import PDFResourceManager, process\_pdf

from pdfminer.pdfdevice import TagExtractor

from pdfminer.converter import XMLConverter, HTMLConverter, TextConverter

from pdfminer.layout import LAParams

# from pdfminer.utils import set\_debug\_logging

def pdf2html(pdfPath, htmlPath):

caching = True

rsrcmgr = PDFResourceManager(caching=caching)

scale = 1

layoutmode = 'noraml'

laparams = LAParams()

outdir = None

debug = False

outfp = io.open(htmlPath, 'wt', encoding = 'utf-8', errors = 'ignore')

device = HTMLConverter(rsrcmgr, outfp, scale=scale, layoutmode=layoutmode,

laparams=laparams, outdir=outdir, debug=debug)

pagenos = set()

maxpages = 0

password = ''

fp = io.open(pdfPath, 'rb')

process\_pdf(rsrcmgr, device, fp, pagenos, maxpages=maxpages, password=password,

caching=caching, check\_extractable=True)

fp.close()

outfp.close()

### 其他pdf转换工具

#### ABBYY FineReader

#### Pypdf

<http://pybrary.net/pyPdf/>

## BeautifulSoup

### 关于查找功能的详细介绍

<http://blog.csdn.net/abclixu123/article/details/38502993>

### BS处理网页

<http://www.tuicool.com/articles/RNFVrm>

Python的BeautifulSoup包大家都知道吧，

**import** BeautifulSoup

**soup** = BeautifulSoup.BeautifulSoup(html)

利用这个包先把html里script，style给清理了：

[**script.extract**() for script in soup.findAll('script')]

[**style.extract**() for style in soup.findAll('style')]

清理完成后，这个包有一个prettify()函数，把代码格式给搞的标准一些：

**soup**.prettify()

然后用正则表达式，把所有的HTML标签全部清理了：

reg1 = re.compile("<[^>]\*>")

content = reg1.sub('',soup.prettify())

剩下的都是纯文本的文件了，通常是一行行的，把空白行给排除了，这样就会知道总计有多少行，每行的字符数有多少，我用excel搞了一些每行字符数的统计，如下图：

### 利用URL下载文件的几种方式

下载的三种方式

<http://www.open-open.com/lib/view/open1420378937984.html>

urllib.request.urlretrieve(url, ‘example.pdf’)

例子：抓取巨潮资讯网的草案页面下pdf下载

from bs4 import BeautifulSoup

import urllib.request

import re

import io

import sys

sys.stdout = io.TextIOWrapper(sys.stdout.buffer,encoding='gb18030') #改变标准输出的默认编码,输出就正常了

urlhand = urllib.request.urlopen(‘http://www.cninfo.com.cn/cninfo-new/disclosure/sse/bulletin\_detail/true/1202367076?announceTime=2016-06-14’)

soup = BeautifulSoup(content, 'html.parser')

# print(soup.get\_text())#cmd中显示仍然乱码

getContent = soup.find(attrs = {'class':r'btn-blue bd-btn'})

# print(getContent)

tags = getContent('a')

# print(len(tags))

pattern = r'"/(cninfo-new.+?)"'

stringFinded = re.findall(pattern, str(tags))#用search的结果难处理，用findall得到list

downloadUrl = 'http://www.cninfo.com.cn/' + stringFinded[0]

print(downloadUrl)

##获得文件名

getName = soup.find('h2')

print(getName.encode('utf-8'))

pattern = r'(\d+?.+?)<br/>[\n\t]+?([\u4e00-\u9fa5\w]+?.+?)[\n\t]'#遇到%号,[\u4e00-\u9fa5\w]是用于匹配中文的，就会停下，点匹配符（.）能匹配任何内容除了\n

stringFinded = re.findall(pattern, str(getName))

print(stringFinded)

saveName = stringFinded[0][0]+stringFinded[0][1] +'.pdf'

urllib.request.urlretrieve(downloadUrl, saveName)

#获得全部text

# fin = open('./textResult.txt', 'w', encoding = 'utf-8')#因为要写入字符串，用utf格式写入

# fin.write(soup.get\_text())#参数是str

# fin.close()

### 并行处理

<http://www.oschina.net/translate/python-parallelism-in-one-line>

没有尝试过，时间少不看，小量任务应该可以用map函数。

### 下载文件出错的处理思路

利用try except 捕捉错误并且写入noOklist列表，把正常获取到的下载文件保存。

注意：

1. 触发异常的语句必须在try里面才能被捕捉到
2. Except 后可以不写exception的种类，捕捉索引，但是如果写了种类，超出种类的就会截断程序。

例如：

#处理包含pdf的巨潮网页，获得pdf下载地址

import xlrd

import re

import socket

from bs4 import BeautifulSoup

import urllib.request

socket.setdefaulttimeout(10)#设置超时时间

#记录所有在excel中取得的数据

pageList = [‘http://www.cninfo.com.cn/cninfo-new/disclosure/sse/bulletin\_detail/true/1202367076?announceTime=2016-06-14’, ‘http://www.cninfo.com.cn/cninfo-new/disclosure/szse\_sme/bulletin\_detail/true/1202534048?announceTime=2016-08-03%2008:15’]

pageName = [‘haha’, ‘xixi’]

#记录获得的pdf下载地址，用于自动下载

downloadName = []

downloadList = []

#记录pdf下载失败的地址，手动下载

notOkList = []

notOkName = []

for index in range(numPageList):

url = pageList[index]

print(index, url)

name = pageName[index]

# print('i am try' + str(index))

try:

html = urllib.request.urlopen(url)

# print('i am hear' + str(index))

content = html.read()

soup = BeautifulSoup(content, 'html.parser')

getContent = soup.find(attrs = {'class' : r'btn-blue bd-btn'})

tags = getContent('a')

pattern = r'"/(cninfo-new.+?)"'

stringFinded = re.findall(pattern, str(tags))#用search的结果难处理，用findall得到list

dlAddress = 'http://www.cninfo.com.cn/' + stringFinded[0]

downloadList.append(dlAddress)

downloadName.append(name)

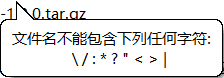
except TimeoutError:#如果不指明异常类型则会捕捉try内容内所有异常，可能是findall的异常，但是可以让程序通畅运行。缺点是无法得知是否出现除了timeout外的错误，所以一般在调试时建议保留异常类型，正式运行删去可以确保完整运行，捕捉所有异常到notOkList再另行处理

notOkList.append(url)

notOkName.append(name)

continue

### 保存文件名出错



一旦自定义的文件名中包含以上字符，则保存的文件名会只剩下左半部分，比如abc?d.pdf最后保存只剩下abc

处理方法：

**patternIllegal = r'[^\\/\\*\?"<>|]+'#[]内加^是代表了否定匹配，可以找出所有正常字符，最后再join就可以**

results = re.findall(patternIllegal, preDifinedName)

fileName = docAddress + ''.join(results)

### 自动循环下载

UseDownloadList.py

#使用parallel获得的downList

import urllib.request

import re

import socket

socket.setdefaulttimeout(10)

currRound = 20

print(currRound)

while (currRound > 0):

fhand = open('./downloadFile.txt', 'r', encoding = 'utf-8')

fContent = fhand.read()

fhand.close()

patternUrl = r'downloadList\n(.+?)\ndownloadName'

patternName = r'downloadName\n(.+)'

urlList = re.findall(patternUrl, fContent, re.S)

nameList = re.findall(patternName, fContent, re.S)

urls = urlList[0].splitlines()

names = nameList[0].splitlines()

docAddress = './pdfStores/'

failUrlList = []

failNameList = []

for index in range(len(urls)):

url = urls[index]

saveName = docAddress + names[index]

try:

print('hi', index)

urllib.request.urlretrieve(url, saveName)

except:

print(url)

print(saveName)

failUrlList.append(url)

failNameList.append(saveName)

continue

if (len(failUrlList) <= 5): break

downloadFile = open('downloadFile.txt', 'w', encoding = 'utf-8')

downloadFile.write('\ndownloadList\n')

downloadFile.write('\n'.join(failUrlList))

downloadFile.write('\ndownloadName\n')

downloadFile.write('\n'.join(failNameList))

downloadFile.close()

currRound -= 1

### 下载速度过慢

下文并没有讲到完全解决方案，但是提供几种尝试方法

<http://www.crifan.com/use_python_urllib-urlretrieve_download_picture_speed_too_slow_add_user_agent_for_urlretrieve/comment-page-1/>

本次下载速度还可以，但是有些防止爬虫的网址就不行，必须伪装

知乎讨论区。后几个有提供方法

<https://www.zhihu.com/question/28168585>

最后，再最后水一下不再更新的破爬虫相关博客：

 [爬虫必备——requests](https://zhuanlan.zhihu.com/p/20410446)

 [01. 准备](https://zhuanlan.zhihu.com/p/20413379)

 [02. 简单的尝试](https://zhuanlan.zhihu.com/p/20413828)

 [番外篇. 搭建称手的Python开发环境](https://zhuanlan.zhihu.com/p/20416894)

 [03. 豆瓣电影TOP250](https://zhuanlan.zhihu.com/p/20423182)

 [04. 另一种抓取方式](https://zhuanlan.zhihu.com/p/20430122)

 [05. 存储](https://zhuanlan.zhihu.com/p/20432575)

 [06. 海量数据的抓取策略](https://zhuanlan.zhihu.com/p/20435541)

 [07. 反爬机制<1>](https://zhuanlan.zhihu.com/p/20471442)

 [08. 模拟登录](https://zhuanlan.zhihu.com/p/20494731)

 [09. 通过爬虫找出我和轮子哥之间的最短关注链](https://zhuanlan.zhihu.com/p/20546546)

作者：xlzd  
链接：https://www.zhihu.com/question/28168585/answer/120205863  
来源：知乎  
著作权归作者所有，转载请联系作者获得授权。

### 系统自动暂停

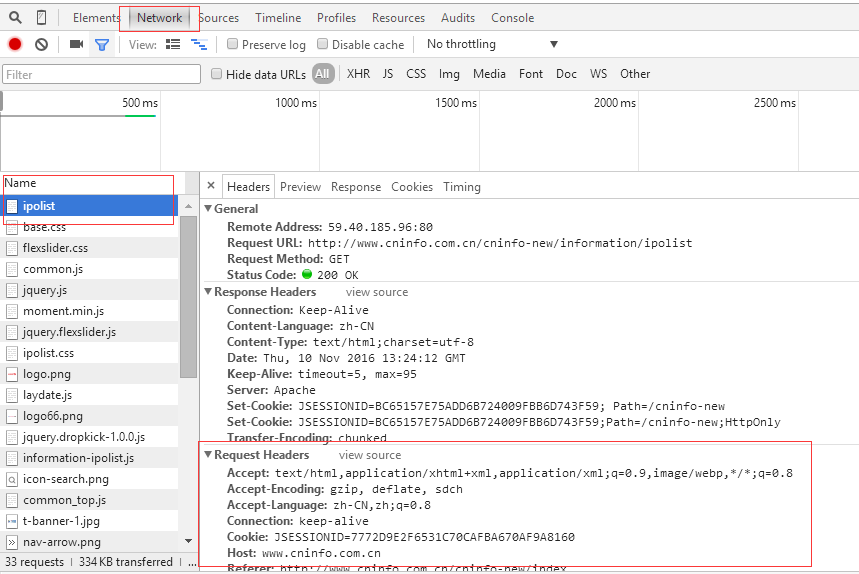
防止被检测下载频繁，可以设置爬虫暂停

Import time

time.sleep(500)

### 爬虫伪装

浏览器f12查看network中的情况。



Request header的情况累死

Accept:text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8

Accept-Encoding:gzip, deflate, sdch

Accept-Language:zh-CN,zh;q=0.8

Connection:keep-alive

Host:www.cninfo.com.cn

Upgrade-Insecure-Requests:1

User-Agent:Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.101 Safari/537.36

详细请看下面的讲解

<https://jecvay.com/2014/09/python3-web-bug-series3.html>

import urllib.request

url = 'http://www.baidu.com/'

req = urllib.request.Request(url, headers = {

'Connection': 'Keep-Alive',

'Accept': 'text/html, application/xhtml+xml, \*/\*',

'Accept-Language': 'en-US,en;q=0.8,zh-Hans-CN;q=0.5,zh-Hans;q=0.3',

'User-Agent': 'Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko'

})

oper = urllib.request.urlopen(req)

data = oper.read()

print(data.decode())

写入的话，因为是urlopen返回byte流，因此直接以byte形式写入, open参数写’wb’，

data = oper.read()

fhand = open(saveName, 'wb')

fhand.write(data)

fhand.close()

可以用type观察流的格式

>>> type(contents)

<class 'bytes'>

>>> type(contents.decode('utf-8'))

<class 'str'>

>>> type(contents.decode('utf-8','ignore'))

<class 'str'>

作者：Fel Peter

链接：https://www.zhihu.com/question/35838789/answer/65794367

来源：知乎

著作权归作者所有，转载请联系作者获得授权。

### 用BS4查找属性值未知的标签

<http://www.crifan.com/python_use_beautifulsoup_find_tag_with_unknown_attribute_value/>

针对：

<div aria-label="5星, 747 份评分" class="rating" role="img" tabindex="-1">

<div>

<span class="rating-star">

</span>

<span class="rating-star">

</span>

<span class="rating-star">

</span>

<span class="rating-star">

</span>

<span class="rating-star">

</span>

</div>

<span class="rating-count">

747 份评分

</span>

</div>

可以通过：

[?](http://www.crifan.com/python_use_beautifulsoup_find_tag_with_unknown_attribute_value/)

|  |  |
| --- | --- |
| 1 | soup.findAll("div", attrs={"aria-lable": True}); |

去查找到属性包含aria-lable的div标签的。

解释：特殊值True和None更让人感兴趣。 True匹配给定属性为任意值的标签，None匹配那些给定的属性值为空的标签。

2.有属性值，要查找特定属性值形式，使用re.compile

patternPageNum = re.compile(r'.+top:.+;')

results = soup.findAll('div', attrs = {'style':patternPageNum})#find可以找到

#### 使用re.compile的好处:

<http://blog.csdn.net/wusuopubupt/article/details/29379367>

对于程序频繁使用的表达式，编译这些表达式会更有效。compile()函数会把一个表达式字符串转化成为一个RegexObject。

模块级函数会维护已编译表达式的一个缓存。不过，这个缓存的大小是有限的，直接使用已编译的表达式可以避免缓存查找开销。使用已编译表达式的另一个好处是，通过在加载模块时预编译所有表达式，可以把编译工作转到应用开始时，而不是当程序响应一个用户动作时才进行编译。

### 用bs的prettify输出漂亮的html

import re

from bs4 import BeautifulSoup

fhand = open('./merge2.html', 'r', encoding = 'utf-8')

htmlContent = fhand.read()

fhand.close()

soup = BeautifulSoup(htmlContent, 'html.parser')

prettyHtml =soup.prettify()

print(type(prettyHtml))

fwrite = open('./merge2.txt', 'w', encoding = 'utf-8')

fwrite.write(prettyHtml)

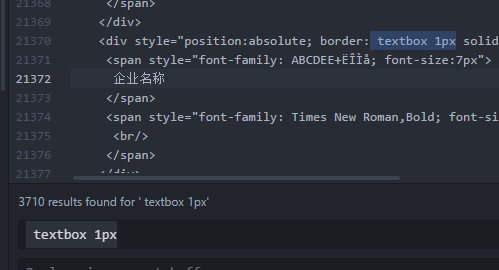
fwrite.close()

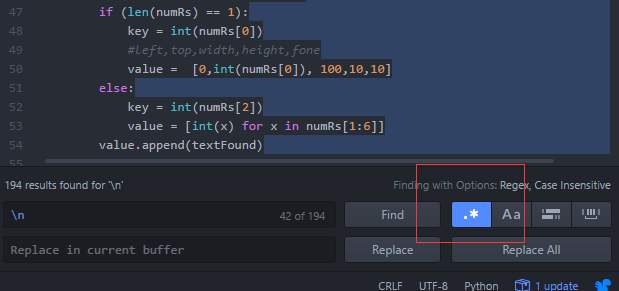
### 利用atom搜索的正则匹配功能

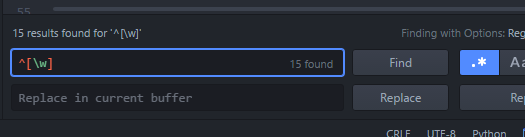
详细探索atom的博文

<http://blog.csdn.net/u010494080/article/details/50615953>

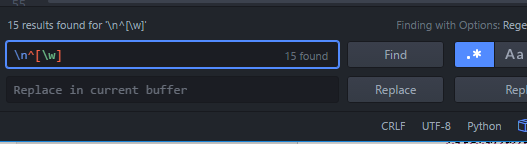
利用atom的find功能对比







支持跨行匹配



Textbox标签是最少的，left多了很多换页时的空白行，但是这些空白行是span标签，top更多，比left多了用于标识页码

### 匹配所有含top的div标签

<http://blog.csdn.net/abclixu123/article/details/38502993>

先确定标签类型是div，然后利用正则表达式匹配属性style中的内容

patternStyle = re.compile(r'.+top:42435px.+;')

results = soup.find(‘div’, attrs = {'style':patternStyle})

print(results)

print(type(results))

print(results.get\_text())

tagValue = re.findall(r'(\d+)px', str(results))

print(tagValue)

<div style="position:absolute; border: textbox 1px solid; writing-mode:lr-tb; left:92px; top:42435px; width:13px; height:24px;"><span style="font-family: ABCDEE+????????; font-size:7px">序

<br>号</br></span><span style="font-family: Times New Roman,Bold; font-size:6px">

<br/></span></div>

<class 'bs4.element.Tag'>

序

号

['1', '92', '42435', '13', '24', '7', '6']

### 否定匹配

[^一二三四]+

不能出现一二三四的字符串

### 字典排序

按key排序

tagSortedList = sorted(tagDict.items(), key = lambda d:d[0])

按照value排序

# tagSortedList = sorted(tagDict.items(), key = lambda d:d[1])

返回的是tuple组成的list,每个tuple是key:value对

### 写入excel

Pip install xlwt

1. encoding:utf-8       #设置编码方式
3. **import** xlwt
4. wbk = xlwt.Workbook(encoding='utf-8', style\_compression=0)
5. sheet = wbk.add\_sheet('sheet 1', cell\_overwrite\_ok=True)  ##第二参数用于确认同一个cell单元是否可以重设值。
7. sheet.write(0,0,'some text')
8. sheet.write(0,0,'this should overwrite')   ##重新设置，需要cell\_overwrite\_ok=True
10. style = xlwt.XFStyle()
11. font = xlwt.Font()
12. font.name = 'Times New Roman'
13. font.bold = True
14. style.font = font
15. sheet.write(0, 1, 'some bold Times text', style)
17. wbk.save('D:\TestData2.xls')    ##该文件名必须存在

### 获取文件夹下文件名

<http://blog.csdn.net/lsq2902101015/article/details/51305825>

1. # -\*- coding: utf-8 -\*-
3. **import** os
5. **def** file\_name(file\_dir):
6. **for** root, dirs, files **in** os.walk(file\_dir):
7. **print**(root) #当前目录路径
8. **print**(dirs) #当前路径下所有子目录
9. **print**(files) #当前路径下所有非目录子文件

## PDF2HTMLEX

下载sh文件然后在linux上用./xxx.sh直接运行

<https://gist.github.com/botmtl/fa44a0b9bb4a7ac64f2c4df5fe90d616>

或这个

<https://gist.github.com/rajeevkannav/d07f822e209a22d07176>

安装

<https://github.com/coolwanglu/pdf2htmlEX/wiki/Building>

### windows直接运行版

证实可以用

<http://soft.rubypdf.com/software/pdf2htmlex-windows-version>

使用方法为cd到exe文件当前目录，pdf2htmlex.exe pdfName

## Html内容详解

### div

DIV元素:是用来为HTML文档内大块（block-level）的内容提供结构和背景的元素。DIV的起始标签和结束标签之间的所有内容都是用来构成这个块的，其中所包含元素的特性由DIV标签的属性来控制，或者是通过使用样式表格式化这个块来进行控制。DIV标签称为区隔标记。作用：设定字、画、表格等的摆放位置。当你把文字、图象，或其他的放在 DIV 中，它可称作为“DIV block”，或“DIV element”或“CSS-layer”，或干脆叫“layer”。而中文我们把它称作“层次”。

定义

<div> 可定义文档中的分区或节（division/section）。

<div> 标签可以把文档分割为独立的、不同的部分。它可以用作严格的组织工具，并且不使用任何格式与其关联。如果用 id 或 class 来标记 <div>，那么该标签的作用会变得更加有效。

用法

DIV标签应用于 Style Sheet（样式表）方面会更显威力，它最终目的是给设计者另一种组织能力，有 Class、Style、title、ID 等属性。

<div> 是一个块级元素。这意味着它的内容自动地开始一个新行。实际上，换行是 <div> 固有的唯一格式表现。可以通过 <div> 的 class 或 id 应用额外的样式。

不必为每一个 <div> 都加上类或 id，虽然这样做也有一定的好处。

可以对同一个 <div> 元素应用 class 或 id 属性，但是更常见的情况是只应用其中一种。这两者的主要差异是，class 用于元素组（类似的元素，或者可以理解为某一类元素），而 id 用于标识单独的唯一的元素。

### Css

层叠样式表(英文全称：Cascading Style Sheets)是一种用来表现HTML（标准通用标记语言的一个应用）或XML（标准通用标记语言的一个子集）等文件样式的计算机语言。

### Css+ div 开发网页

<http://blog.csdn.net/mercop/article/details/7882000/>

### br标签

<br> 可插入一个简单的换行符。

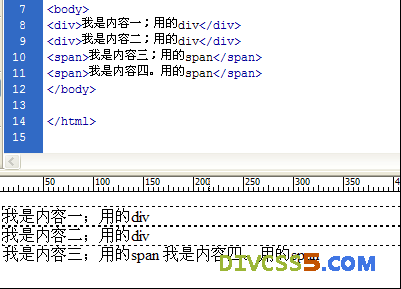
<br> 标签是空标签（意味着它没有结束标签，因此这是错误的：<br></br>）。在 XHTML 中，把结束标签放在开始标签中，也就是 <br />。

请注意，<br> 标签只是简单地开始新的一行，而当浏览器遇到 <p> 标签时，通常会在相邻的段落之间插入一些垂直的间距。

<br>和<br/>具有相同作用，但前者是老的html规范，新的规范（W3C//DTD HTML 4.0）要求一切都应象XML那样有结束符，如果没有也要硬加一个反斜杠。**象br这种不需成对使用的html符号，**就成了这个样子：<br/>。

### Span

div与span区别  
div占用的位置是一行，  
span占用的是内容有多宽就占用多宽的空间距离，说明如下图

  
sapn标签使用案例截图

分析：从上图很容易知道“我是内容一；用的div”和“我是内容二；用的div”两个内容外部用的是**<div>标签**，他们得到样式是占用了一排空间（相当于换行一样）；而“我是内容三；用的span”和“我是内容四；用的span”则，文字内容有多宽，就占用多宽距离，使用<span>标签和不使用一样效果。

HTML <div> 元素是块级元素，它是可用于组合其他 HTML 元素的容器。

<div> 元素没有特定的含义。除此之外，由于它属于块级元素，浏览器会在其前后显示折行。

HTML <span> 元素是内联元素，可用作文本的容器。

例如：

<span style="font-family: ABCDEE+PMingLiU; font-size:7px">

萬科企業股份有限公司

</span>

<span style="font-family: Times New Roman,Bold; font-size:6px">

<br/>

</span>

文本内容7px，然后下面的空白行6px

### Div+span+br

Div开辟一块内容，限定了其开始行的高度(top)，开始列的位置（left）其宽度(width)，但是height限制不了整体高度，宽度不够字体会排列成很多行，跟下面的重复

div内中用span来写入内容，但是span不会换行，需要用br做一次换行,如果想要控值空白行的大小，需要再加一个br，这个br的font-size就决定了空白行的大小，第一个br跟span在同一行，所以大小好像不会影响?

例如：

<body>

<div style="position:absolute; border: textbox 1px solid; writing-mode:lr-tb; left:400px; top:0px; width:1110px; height:1000px;">

<span style="font-family: ABCDEE+PMingLiU; font-size:7px">

1萬科企業股份有限公司

</span>

<span style="font-family: Times New Roman,Bold; font-size:100px">

<br/>

</span>

<span style="font-family: ABCDEE+PMingLiU; font-size:7px">

2萬科企業股份有限公司

</span>

</div>

<div style="position:absolute; border: textbox 1px solid; writing-mode:lr-tb; left:400px; top:14px; width:110px; height:50px;">

<span style="font-family: ABCDEE+PMingLiU; font-size:7px">

3萬科企業股份有限公司

</span>

<span style="font-family: Times New Roman,Bold; font-size:100px">

<br/>

</span>

<span style="font-family: Times New Roman,Bold; font-size:100px">

<br/>

</span>

<span style="font-family: ABCDEE+PMingLiU; font-size:7px">

4萬科企業股份有限公司

</span>

</div>

<div style="position:absolute; border: textbox 1px solid; writing-mode:lr-tb; left:400px; top:14px; width:110px; height:50px;">

<span style="font-family: ABCDEE+PMingLiU; font-size:7px">

5萬科企業股份有限公司

</span>

<span style="font-family: Times New Roman,Bold; font-size:0px">

<br/>

</span>

<span style="font-family: ABCDEE+PMingLiU; font-size:7px">

6萬科企業股份有限公司

</span>

</div>

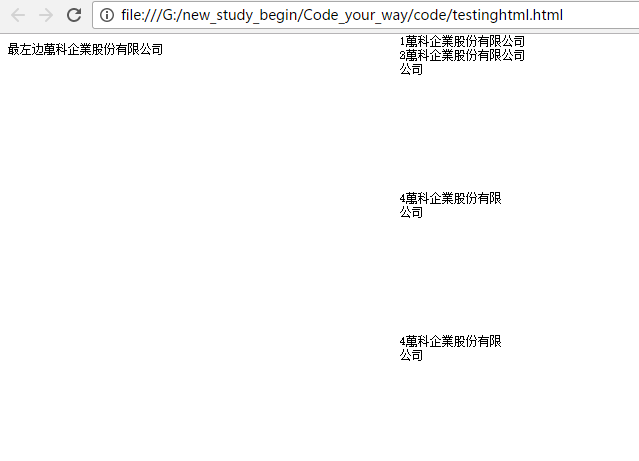
<span style="font-family: ABCDEE+PMingLiU; font-size:7px">

最左边萬科企業股份有限公司

</span>

</body>

效果：



## 文本挖掘

### 优秀博客

<http://blog.csdn.net/fighting_no1/article/details/50927162>

### 可能问题

数字有时是2,54而非254.

### 英文分词 波特分词法

Python版本

http://blog.csdn.net/shuangshuang37278752/article/details/9314131

java版本

<http://blog.csdn.net/fighting_no1/article/details/51003386>

### 中文分词

1. 基于词典的方法（字符串匹配，机械分词方法）   
   定义:按照一定策略将待分析的汉字串与一个“大机器词典”中的词条进行匹配，若在词典中找到某个字符串，则匹配成功。   
   按照扫描方向的不同：正向匹配和逆向匹配   
   按照长度的不同：最大匹配和最小匹配   
   1.2基于统计的分词（无字典分词）   
   主要思想：上下文中，相邻的字同时出现的次数越多，就越可能构成一个词。因此字与字相邻出现的概率或频率能较好的反映词的可信度。   
   主要统计模型为：N元文法模型（N-gram）、隐马尔科夫模型(Hidden Markov Model, HMM)。   
   **最大正向匹配算法**:从左向右扫描寻找词的最大匹配。首先我们规定一个词的最大长度，每次扫描的时候寻找当前开始的这个长度的词来和字典中的词匹配，如果没有找到，就缩短长度继续寻找，直到找到或者成为单字。

#### 结巴分词器

<http://blog.csdn.net/Fighting_No1/article/details/50927180>

## 全文检索

Python2.7

<http://www.cnblogs.com/fuhuixiang/p/4488029.html>

基本上所有的开源搜索引擎都有python的封装，sphinx,xapian,solr等，还有纯python写的引擎woosh，性能没有测试过，估计比不上用c/c++引擎。

### Python3的woosh+逻辑查询

<http://blog.csdn.net/u012387575/article/details/52187920>

默认的  QueryParser 实现了一个与Lucene的查询语言很类似的语言。它让你用 AND 和 OR  将 terms 连接起来，用NOT 消除 terms，用括号将terms组成句子，做一些范围（range）、前缀（prefix）、通配符（wildcard）查询，明确搜索的不同field。默认情况下，经常与AND 合用(所以默认情况下, 你举出的所有terms 必须在文档中，以让文本匹配):

**[python]** [view plain](http://blog.csdn.net/u012387575/article/details/52188054) [copy](http://blog.csdn.net/u012387575/article/details/52188054)

[在CODE上查看代码片](https://code.csdn.net/snippets/1824765)

1. >>> **print**(parser.parse(u"render shade animate"))
2. And([Term("content", "render"), Term("content", "shade"), Term("content", "animate")])
4. >>> **print**(parser.parse(u"render OR (title:shade keyword:animate)"))
5. Or([Term("content", "render"), And([Term("title", "shade"), Term("keyword", "animate")])])
7. >>> **print**(parser.parse(u"rend\*"))
8. Prefix("content", "rend")

### 提供woosh的应用方法

<http://blog.csdn.net/wenxuansoft/article/details/8169842>

写一个分词器

from whoosh.analysis import Tokenizer,Token

class ChineseTokenizer(Tokenizer):

def \_\_call\_\_(self, value, positions=False, chars=False,

keeporiginal=False, removestops=True,

start\_pos=0, start\_char=0, mode='', \*\*kwargs):

assert isinstance(value, text\_type), "%r is not unicode" % value

t = Token(positions, chars, removestops=removestops, mode=mode,

\*\*kwargs)

seglist=jieba.cut(value,cut\_all=False) #使用结巴分词库进行分词

for w in seglist:

t.original = t.text = w

t.boost = 1.0

if positions:

t.pos=start\_pos+value.find(w)

if chars:

t.startchar=start\_char+value.find(w)

t.endchar=start\_char+value.find(w)+len(w)

yield t #通过生成器返回每个分词的结果token

def ChineseAnalyzer():

return ChineseTokenizer()

#重点在这里，将原先的RegexAnalyzer(ur”([\u4e00-\u9fa5])|(\w+(\.?\w+)\*)”),改成这句，用中文分词器代替原先的正则表达式解释器。

analyzer=ChineseAnalyzer()

<http://www.cnblogs.com/fuhuixiang/p/4488029.html>

**非常优秀的总结：提供并行分词思路**

<http://blog.csdn.net/xyw_blog/article/details/41893871>

### jieba自定义词典

<http://www.cnblogs.com/eastmount/p/5055906.html>

Python中结巴分词的准确性比较高，网上有详细的教程，包括自字义字典的使用方法。

最近在做实验室的一个小项目，其中有很多实体名不规则，需要使用自定义的字典，按照网上某些教程的方法，建立了自定义字典，该方法说只有词性是可选的，我就设置了词频，发现不管是提高还是降低数值，自己希望的自定义词仍然没有出现。后面上Git看了作者发的东西，说词频也是可选的，**然后就试着把词频也去掉了，自已定义的词就出现了。**

真是众里寻他千百度，蓦然回首，那人就站在我身后

词性标注（optional）

http://www.hankcs.com/nlp/part-of-speech-tagging.html

例子：

def paragraphParser(docs, searchKey, pathName, customDictOrNot = 1):

# if (customDictOrNot == 1):

# fhand = open("customDict.txt", 'w', encoding = 'utf-8')

# fhand.write(searchKey)

# fhand.close()

# jieba.load\_userdict("customDict.txt")#必须通过python进行utf-8编码写入才能成功

jieba.load\_userdict("customDict.txt")#必须通过python进行utf-8编码写入才能成功

# 使用结巴中文分词

analyzer = jieba.analyse.ChineseAnalyzer()

# 创建schema, stored为True表示能够被检索

schema = Schema(title=TEXT(stored=True, analyzer=analyzer), path=ID(stored=False),

content=TEXT(stored=True, analyzer=analyzer))

# 存储schema信息至'indexdir'目录下

indexdir = 'indexdir/'

if not os.path.exists(indexdir):

os.mkdir(indexdir)

ix = create\_in(indexdir, schema)

### whoosh search 说明文档

<https://whoosh.readthedocs.io/en/latest/searching.html>

#### 融合文档提高速度

So, having a few segments is more efficient than rewriting the entire index every time you add some documents. But searching multiple segments does slow down searching somewhat, and the more segments you have, the slower it gets. So Whoosh has an algorithm that runs when you call commit()that looks for small segments it can merge together to make fewer, bigger segments.

To prevent Whoosh from merging segments during a commit, use the merge keyword argument:

writer**.**commit(merge**=**False)

To merge all segments together, optimizing the index into a single segment, use the optimizekeyword argument:

writer**.**commit(optimize**=**True)

### whoosh+jieba测试CODE

# coding=utf-8

import os

from whoosh.index import create\_in

from whoosh.fields import \*

# from jieba.analyse import ChineseAnalyzer

import jieba

import jieba.analyse

import json

#导入自定义词典

# jieba.load\_userdict(u"customDict.txt")#x显示ValueError: dictionary file customDict.txt must be utf-8

customDictOrNot = 1

if (customDictOrNot):

fhand = open("customDict.txt", 'w', encoding = 'utf-8')

fhand.write("人王尼玛")

fhand.close()

jieba.load\_userdict("customDict.txt")#必须通过python进行utf-8编码写入才能成功

# 使用结巴中文分词

analyzer = jieba.analyse.ChineseAnalyzer()

# 创建schema, stored为True表示能够被检索

schema = Schema(title=TEXT(stored=True, analyzer=analyzer), path=ID(stored=False),

content=TEXT(stored=True, analyzer=analyzer))

# 存储schema信息至'indexdir'目录下

indexdir = 'indexdir/'

if not os.path.exists(indexdir):

os.mkdir(indexdir)

ix = create\_in(indexdir, schema)

# 按照schema定义信息，增加需要建立索引的文档

# 注意：字符串格式需要为unicode格式

writer = ix.writer()

writer.add\_document(title=u"第一篇文档", path=u"/a",

content=u"这是我们增加的第一篇文档。证监会主管批准审核，主持人王尼玛讲话")

writer.add\_document(title=u"第二篇文档", path=u"/b",

content=u"第二篇文档也很interesting！银监会不批准审核，主席习近平")

writer.commit()

# 创建一个检索器

searcher = ix.searcher()

# 检索标题中出现'文档'的文档

# results = searcher.find("title", u"文档")

# results = searcher.find("content", u"银监会")#银监会是属于词库的

# results = searcher.find("content", u"习近平")#习近平是属于词库的

results = searcher.find("content", u"人王尼玛")#属于自定义词，直接找不到的，必须添加自定义词典

# 检索出来的第一个结果，数据格式为dict{'title':.., 'content':...}

firstdoc = results[0].fields()

# python2中，需要使用json来打印包含unicode的dict内容

jsondoc = json.dumps(firstdoc, ensure\_ascii=False)

print(jsondoc) # 打印出检索出的文档全部内容

print(results[0].highlights("title")) # 高亮标题中的检索词

print(results[0].score) # bm25分数

### CODE按页分割文本排序后进行全文检索并输出结果

# coding=utf-8

import os

from whoosh.index import create\_in

from whoosh.fields import \*

# from jieba.analyse import ChineseAnalyzer

import jieba

import jieba.analyse

import json

import re

def segParagraph(content, spliterOption = 1):

'''spliterOption = 1按段分割，2为按页分割'''

if (spliterOption ==1 ):

spliter = r'\n------------\n'

else:#

spliter = r'Page\s[\d]+'

# return content.split(content, spliter)#split不能用re

return re.split(spliter, content)

#导入自定义词典

# jieba.load\_userdict(u"customDict.txt")#x显示ValueError: dictionary file customDict.txt must be utf-8

def paragraphParser(contentPath, searchKey, pathName):

# customDictOrNot = 0

# if (customDictOrNot):

# fhand = open("customDict.txt", 'w', encoding = 'utf-8')

# fhand.write("人王尼玛")

# fhand.close()

# jieba.load\_userdict("customDict.txt")#必须通过python进行utf-8编码写入才能成功

# 使用结巴中文分词

analyzer = jieba.analyse.ChineseAnalyzer()

# 创建schema, stored为True表示能够被检索

schema = Schema(title=TEXT(stored=True, analyzer=analyzer), path=ID(stored=False),

content=TEXT(stored=True, analyzer=analyzer))

# 存储schema信息至'indexdir'目录下

indexdir = 'indexdir/'

if not os.path.exists(indexdir):

os.mkdir(indexdir)

ix = create\_in(indexdir, schema)

fhand = open(contentPath, 'r', encoding='utf-8')

pContent = fhand.read()

fhand.close()

#文章进行分段

docs = segParagraph(pContent, 1)

counter = 1

writer = ix.writer()

for doc in docs:

# 按照schema定义信息，增加需要建立索引的文档

# 注意：字符串格式需要为unicode格式

writer.add\_document(title=str(counter), path=pathName, content=doc)

counter += 1

writer.commit()

# 创建一个检索器

searcher = ix.searcher()

# 检索标题中出现'文档'的文档

# results = searcher.find("title", u"文档")

# results = searcher.find("content", u"银监会")#银监会是属于词库的

# results = searcher.find("content", u"习近平")#习近平是属于词库的

results = searcher.find("content", searchKey)#属于自定义词，直接找不到的，必须添加自定义词典

# 检索出来的第一个结果，数据格式为dict{'title':.., 'content':...}

printSearch(results)

def printSearch(results):

#http://whoosh.readthedocs.io/en/latest/releases/1\_0.html?highlight=print

for hit in results:

print(hit["title"])

# test = hit.highlights("content")

# print(test)

print(hit["content"])

# print(type(test))#属于str类

# firstdoc = results[0].fields()

# python2中，需要使用json来打印包含unicode的dict内容

# jsondoc = json.dumps(firstdoc, ensure\_ascii=False)

# print(jsondoc) # 打印出检索出的文档全部内容

# print(results[0].highlights("title")) # 高亮标题中的检索词

# print(results[0].score) # bm25分数

#### 遍历字典添加到search的文档中

for (dIndex, dContent) in docs.items():

# 按照schema定义信息，增加需要建立索引的文档

# 注意：字符串格式需要为unicode格式

writer.add\_document(title=dIndex, path=pathName, content=dContent)

writer.commit()

(2) os.environ.items() 返回一个 tuple 的 list：[(key1, value1), (key2, value2), ...]。for 循环对这个 list 进行遍历。第一轮，它将 key1 赋给 k ，value1 赋给 v，所以 k = USERPROFILE，v = C:\Documents and Settings\mpilgrim。第二轮，k 得到第二个键字 OS，v 得到相应的值 Windows\_NT。

#### 关于python中re模块split方法的使用

已有 2708 次阅读 2014-3-12 11:30 |系统分类:[科研笔记](http://blog.sciencenet.cn/home.php?mod=space&do=blog&view=all&uid=314114&catid=1)

今天在写一段小代码的时候需要用到re.split()方法，在使用的过程中发现了一个以前不知道的新用法，发现这个用法还是挺实用的，就把它记录下来：

>>> m = re.split( ‘\d+’ , ’dkjj23jjjj44’)

>>>m

[‘dkjj’ , ‘jjjj’ , ‘’ ]

>>> m = re.split( ‘(\d+)’ , ’dkjj23jjjj44’)             #匹配部分加上（）

>>>m

[‘dkjj’ , ‘23’ , ‘jjjj’ , ‘44’ , ‘’ ]

>>> m = re.split( ‘\d+’ , ’dkjj23jjjj44as’)            #匹配部分加上了（）

>>>m

[‘dkjj’ , ‘jjjj’ , ‘as’ ]

>>> m = re.split( ‘(\d+)’ , ’dkjj23jjjj44as’)

>>>m

[‘dkjj’ , ‘23’ , ‘jjjj’ , ‘44’ , ‘as’ ]

在匹配部分加上（）之后所切出的结果是不同的，没有（）的没有保留所匹配的项，但是有（）的却能够保留了匹配的项，这个在某些需要保留匹配部分的使用过程是非常重要的。

#### \*\*非常全的关于re的解释

<http://www.cnblogs.com/huxi/archive/2010/07/04/1771073.html>

#### re.split返回None type 需要删除none

def fixPage(joinedContent):

'''进一步融合自然段，防止出现分隔符隔断语义，可以容忍分段数的下降'''

spliter = r'''([\d]+)\n------------\n|------------\n([\d,]{2,4})'''

results = re.split(spliter, joinedContent)

#这里居然会返回none type.只能取出了

ans = ''.join([x for x in results if x is not None])

return ans

## 正则表达式的详细内容

字符串是编程时涉及到的最多的一种数据结构，对字符串进行操作的需求几乎无处不在。比如判断一个字符串是否是合法的Email地址，虽然可以编程提取@前后的子串，再分别判断是否是单词和域名，但这样做不但麻烦，而且代码难以复用。

正则表达式是一种用来匹配字符串的强有力的武器。它的设计思想是用一种描述性的语言来给字符串定义一个规则，凡是符合规则的字符串，我们就认为它“匹配”了，否则，该字符串就是不合法的。

所以我们判断一个字符串是否是合法的Email的方法是：

1. 创建一个匹配Email的正则表达式；
2. 用该正则表达式去匹配用户的输入来判断是否合法。

因为正则表达式也是用字符串表示的，所以，我们要首先了解如何用字符来描述字符。

在正则表达式中，如果直接给出字符，就是精确匹配。用\d可以匹配一个数字，\w可以匹配一个字母或数字，所以：

* '00\d'可以匹配'007'，但无法匹配'00A'；
* '\d\d\d'可以匹配'010'；
* '\w\w\d'可以匹配'py3'；

.可以匹配任意字符，所以：

* 'py.'可以匹配'pyc'、'pyo'、'py!'等等。

要匹配变长的字符，在正则表达式中，用\*表示任意个字符（包括0个），用+表示至少一个字符，用?表示0个或1个字符，用{n}表示n个字符，用{n,m}表示n-m个字符：

来看一个复杂的例子：\d{3}\s+\d{3,8}。

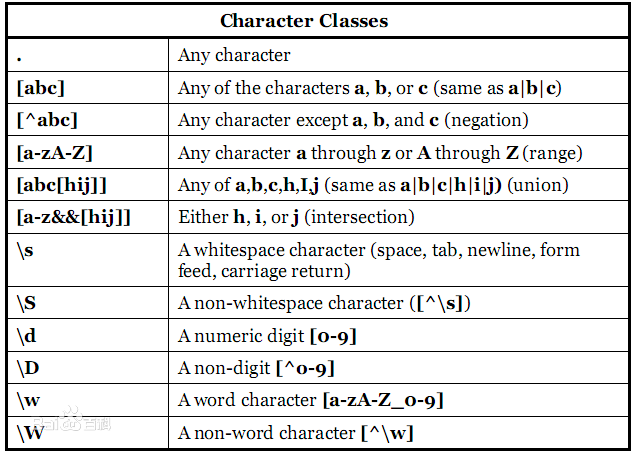
我们来从左到右解读一下：

1. \d{3}表示匹配3个数字，例如'010'；
2. \s可以匹配一个空格（也包括Tab等空白符），所以\s+表示至少有一个空格，例如匹配' '，' '等；
3. \d{3,8}表示3-8个数字，例如'1234567'。

综合起来，上面的正则表达式可以匹配以任意个空格隔开的带区号的电话号码。

如果要匹配'010-12345'这样的号码呢？由于'-'是特殊字符，在正则表达式中，要用'\'转义，所以，上面的正则是\d{3}\-\d{3,8}。

但是，仍然无法匹配'010 - 12345'，因为带有空格。所以我们需要更复杂的匹配方式。



### 进阶

要做更精确地匹配，可以用[]表示范围，比如：

* [0-9a-zA-Z\\_]可以匹配一个数字、字母或者下划线；
* [0-9a-zA-Z\\_]+可以匹配至少由一个数字、字母或者下划线组成的字符串，比如'a100'，'0\_Z'，'Py3000'等等；
* [a-zA-Z\\_][0-9a-zA-Z\\_]\*可以匹配由字母或下划线开头，后接任意个由一个数字、字母或者下划线组成的字符串，也就是Python合法的变量；
* [a-zA-Z\\_][0-9a-zA-Z\\_]{0, 19}更精确地限制了变量的长度是1-20个字符（前面1个字符+后面最多19个字符）。

A|B可以匹配A或B，所以(P|p)ython可以匹配'Python'或者'python'。

^表示行的开头，^\d表示必须以数字开头。

$表示行的结束，\d$表示必须以数字结束。

你可能注意到了，py也可以匹配'python'，但是加上^py$就变成了整行匹配，就只能匹配'py'了。

### re模块

有了准备知识，我们就可以在Python中使用正则表达式了。Python提供re模块，包含所有正则表达式的功能。由于Python的字符串本身也用\转义，所以要特别注意：

s = 'ABC\\-001' *# Python的字符串*

*# 对应的正则表达式字符串变成：*

*# 'ABC\-001'*

因此我们强烈建议使用Python的r前缀，就不用考虑转义的问题了：

s = r'ABC\-001' *# Python的字符串*

*# 对应的正则表达式字符串不变：*

*# 'ABC\-001'*

先看看如何判断正则表达式是否匹配：

>>> **import** re

>>> re.match(r'^\d{3}\-\d{3,8}$', '010-12345')

<\_sre.SRE\_Match object; span=(0, 9), match='010-12345'>

>>> re.match(r'^\d{3}\-\d{3,8}$', '010 12345')

>>>

match()方法判断是否匹配，如果匹配成功，返回一个Match对象，否则返回None。常见的判断方法就是：

test = '用户输入的字符串'

**if** re.match(r'正则表达式', test):

print('ok')

**else**:

print('failed')

### re模块常用函数

#### re.match

　　re.match 尝试从字符串的开始匹配一个模式，如：下面的例子匹配第一个单词。

import re

text = "JGood is a handsome boy, he is cool, clever, and so on..."

m = re.match(r"(\w+)\s", text)

if m:

print m.group(0), '\n', m.group(1)

else:

print 'not match'

import re text = "JGood is a handsome boy, he is cool, clever, and so on..." m = re.match(r"(\w+)\s", text) if m: print m.group(0), '\n', m.group(1) else: print 'not match'

re.match的函数原型为：re.match(pattern, string, flags)

第一个参数是正则表达式，这里为"(\w+)\s"，如果匹配成功，则返回一个Match，否则返回一个None；

第二个参数表示要匹配的字符串；

第三个参数是标致位，用于控制正则表达式的匹配方式，如：是否区分大小写，多行匹配等等。

#### Match 和 search

match（）函数只检测RE是不是在string的开始位置匹配，

 search()会扫描整个string查找匹配；

也就是说match（）只有在0位置匹配成功的话才有返回，

如果不是开始位置匹配成功的话，match()就返回none。

#### re.search

　　re.search函数会在字符串内查找模式匹配,只到找到第一个匹配然后返回，如果字符串没有匹配，则返回None。

import re

text = "JGood is a handsome boy, he is cool, clever, and so on..."

m = re.search(r'\shan(ds)ome\s', text)

if m:

print m.group(0), m.group(1)

else:

print 'not search'

import re text = "JGood is a handsome boy, he is cool, clever, and so on..." m = re.search(r'\shan(ds)ome\s', text) if m: print m.group(0), m.group(1) else: print 'not search'

re.search的函数原型为： re.search(pattern, string, flags)

每个参数的含意与re.match一样。

re.match与re.search的区别：re.match只匹配字符串的开始，如果字符串开始不符合正则表达式，则匹配失败，函数返回None；而re.search匹配整个字符串，直到找到一个匹配。

#### re.sub

　　re.sub用于替换字符串中的匹配项。下面一个例子将字符串中的空格 ' ' 替换成 '-' :

import re

text = "JGood is a handsome boy, he is cool, clever, and so on..."

print re.sub(r'\s+', '-', text)

import re text = "JGood is a handsome boy, he is cool, clever, and so on..." print re.sub(r'\s+', '-', text)

re.sub的函数原型为：re.sub(pattern, repl, string, count)

其中第二个函数是替换后的字符串；本例中为'-'

第四个参数指替换个数。默认为0，表示每个匹配项都替换。

re.sub还允许使用函数对匹配项的替换进行复杂的处理。如：re.sub(r'\s', lambda m: '[' + m.group(0) + ']', text, 0)；将字符串中的空格' '替换为'[ ]'。

#### re.split

　　可以使用re.split来分割字符串，如：re.split(r'\s+', text)；将字符串按空格分割成一个单词列表。

#### re.findall

　　re.findall可以获取字符串中所有匹配的字符串。如：re.findall(r'\w\*oo\w\*', text)；获取字符串中，包含'oo'的所有单词。

#### re.compile

　　可以把正则表达式编译成一个正则表达式对象。可以把那些经常使用的正则表达式编译成正则表达式对象，这样可以提高一定的效率。下面是一个正则表达式对象的一个例子：

import re

text = "JGood is a handsome boy, he is cool, clever, and so on..."

regex = re.compile(r'\w\*oo\w\*')

print regex.findall(text) #查找所有包含'oo'的单词

print regex.sub(lambda m: '[' + m.group(0) + ']', text) #将字符串中含有'oo'的单词用[]括起来。

import re text = "JGood is a handsome boy, he is cool, clever, and so on..." regex = re.compile(r'\w\*oo\w\*') print regex.findall(text) #查找所有包含'oo'的单词 print regex.sub(lambda m: '[' + m.group(0) + ']', text) #将字符串中含有'oo'的单词用[]括起来。

　　更详细的内容，可以参考Python手册。

### 切分字符串

用正则表达式切分字符串比用固定的字符更灵活，请看正常的切分代码：

>>> 'a b c'.split(' ')

['a', 'b', '', '', 'c']

嗯，无法识别连续的空格，用正则表达式试试：

>>> re.split(r'\s+', 'a b c')

['a', 'b', 'c']

无论多少个空格都可以正常分割。加入,试试：

>>> re.split(r'[\s\,]+', 'a,b, c d')

['a', 'b', 'c', 'd']

再加入;试试：

>>> re.split(r'[\s\,\;]+', 'a,b;; c d')

['a', 'b', 'c', 'd']

如果用户输入了一组标签，下次记得用正则表达式来把不规范的输入转化成正确的数组。

### 分组

除了简单地判断是否匹配之外，正则表达式还有提取子串的强大功能。用()表示的就是要提取的分组（Group）。比如：

^(\d{3})-(\d{3,8})$分别定义了两个组，可以直接从匹配的字符串中提取出区号和本地号码：

>>> m = re.match(r'^(\d{3})-(\d{3,8})$', '010-12345')

>>> m

<\_sre.SRE\_Match object; span=(0, 9), match='010-12345'>

>>> m.group(0)

'010-12345'

>>> m.group(1)

'010'

>>> m.group(2)

'12345'

如果正则表达式中定义了组，就可以在Match对象上用group()方法提取出子串来。

注意到group(0)永远是原始字符串，group(1)、group(2)……表示第1、2、……个子串。

提取子串非常有用。来看一个更凶残的例子：

>>> t = '19:05:30'

>>> m = re.match(r'^(0[0-9]|1[0-9]|2[0-3]|[0-9])\:(0[0-9]|1[0-9]|2[0-9]|3[0-9]|4[0-9]|5[0-9]|[0-9])\:(0[0-9]|1[0-9]|2[0-9]|3[0-9]|4[0-9]|5[0-9]|[0-9])$', t)

>>> m.groups()

('19', '05', '30')

这个正则表达式可以直接识别合法的时间。但是有些时候，用正则表达式也无法做到完全验证，比如识别日期：

'^(0[1-9]|1[0-2]|[0-9])-(0[1-9]|1[0-9]|2[0-9]|3[0-1]|[0-9])$'

对于'2-30'，'4-31'这样的非法日期，用正则还是识别不了，或者说写出来非常困难，这时就需要程序配合识别了。

### python 贪婪 非贪婪 匹配

<http://www.51itong.net/python-8705.html>

一般的匹配会在遇到换行符久停止

### 贪婪匹配

最后需要特别指出的是，正则匹配默认是贪婪匹配，也就是匹配尽可能多的字符。举例如下，匹配出数字后面的0：

>>> re.match(r'^(\d+)(0\*)$', '102300').groups()

('102300', '')

由于\d+采用贪婪匹配，直接把后面的0全部匹配了，结果0\*只能匹配空字符串了。

必须让\d+采用非贪婪匹配（也就是尽可能少匹配），才能把后面的0匹配出来，加个?就可以让\d+采用非贪婪匹配：

>>> re.match(r'^(\d+?)(0\*)$', '102300').groups()

('1023', '00')

### 多行匹配和多行开头匹配

<http://blog.csdn.net/my2010sam/article/details/10062939>

urlList = re.findall(patternUrl, content, re.S)

能匹配

<xx>

.

x.

.

.

</xx>

一大段内容，再用 result.splitlines()

再去得到结果的每行组成的list

如果是多行同时有相同的模式，无需re.s re.m 直接fingdall，就会在每一行上找，因为一遇到分隔符匹配就停止了。

例子：

import xlrd

import re

fhand = open('downloadFile.txt', 'r', encoding = 'utf-8')

fcontent = fhand.read()

fhand.close()

# print(fcontent)

patternLines = r'\./pdfStores/(merge.+?)\n'

nameLists = re.findall(patternLines, fcontent)

fhand = open('downloadFile.txt', 'a', encoding = 'utf-8')

fhand.write('\ncorrected:\n')

fhand.write('\n'.join(nameLists))

### 编译

当我们在Python中使用正则表达式时，re模块内部会干两件事情：

1. 编译正则表达式，如果正则表达式的字符串本身不合法，会报错；
2. 用编译后的正则表达式去匹配字符串。

如果一个正则表达式要重复使用几千次，出于效率的考虑，我们可以预编译该正则表达式，接下来重复使用时就不需要编译这个步骤了，直接匹配：

>>> **import** re

*# 编译:*

>>> re\_telephone = re.compile(r'^(\d{3})-(\d{3,8})$')

*# 使用：*

>>> re\_telephone.match('010-12345').groups()

('010', '12345')

>>> re\_telephone.match('010-8086').groups()

('010', '8086')

编译后生成Regular Expression对象，由于该对象自己包含了正则表达式，所以调用对应的方法时不用给出正则字符串。

### 小结

正则表达式非常强大，要在短短的一节里讲完是不可能的。要讲清楚正则的所有内容，可以写一本厚厚的书了。如果你经常遇到正则表达式的问题，你可能需要一本正则表达式的参考书。

### 练习

请尝试写一个验证Email地址的正则表达式。版本一应该可以验证出类似的Email：

someone@gmail.com

bill.gates@microsoft.com

版本二可以验证并提取出带名字的Email地址：

<Tom Paris> tom@voyager.org

#### Findall 模式匹配

ans = re.findall('([A-X].+?)\n', string);

print(ans)

ans = re.findall(r'([A-X].+?)\n', string);

print(ans)

ans = re.findall('[A-X].+?\n', string);

print(ans)

for res in ans:

print(res.strip());

### 小结

Python的分布式进程接口简单，封装良好，适合需要把繁重任务分布到多台机器的环境下。

注意Queue的作用是用来传递任务和接收结果，每个任务的描述数据量要尽量小。比如发送一个处理日志文件的任务，就不要发送几百兆的日志文件本身，而是发送日志文件存放的完整路径，由Worker进程再去共享的磁盘上读取文件。

### 斜杠匹配问题

raw string is recommended.

Backslash has a specific usage in matching to indicate special forms or to allow special characters to be used without invoking their special meaning, so do . ^ $ \* + ? { } [ ] \ | ( )

Therefore when we match a literal backslash or literal star, we should precede them with a backslash in raw string.

Now we have the raw string. If we don’t want use raw string. We need to turn it into regular string. We just need to replace all the backslash in raw string with double backslash.

| **Regular String** | **Raw string** |
| --- | --- |
| "ab\*" | r"ab\*" |
| "\\\\section" | r"\\section" |
| "\\w+\\s+\\1" | r"\w+\s+\1" |

#### Example:

All these patterns are matching to href=\"\/watch?v=rqf7Za7Zxac\

pattern = r'href=\\"\\(/watch\?v=\w+?\\)';

#pattern = r'href=\\"\\(/watch\?v=[\w]+?\\)'; #[\w] and \w are the same

#pattern = r'href=\\"\\(/watch\?v=[a-zA-Z0-9]+?\\)';

pattern\_reg = 'href=\\\\"\\\\(/watch\\?v=\\w+?\\\\)'

vedio = re.findall(pattern, html\_content);

vedio = set(vedio)

print(vedio)

得到不重复的匹配结果，但是这些结果每个最后都有两个斜杠???

{'/watch?v=YM2a9hlu2T0\\"', 这很奇怪啊

### 匹配中文

[\u4e00-\u9fa5]+

### 匹配后原位置替换

import re

p = re.compile(r'(\w+) (\w+)')

s = 'i say, hello world!'

print(p.sub(r'\2 \1', s))

def func(m):

'''替换匹配到的整个pattern内容'''

print(m.group(0))//匹配的整个内容

print(m.group(1))//匹配pattern中第一个（）内容

print(m.group(2))

return 'sdafdasf'+m.group(1)+'xxxx'

print(p.sub(func, s))

### output ###

### 反转并且改为大写

[**Python**](http://lib.csdn.net/base/python)在对找到的匹配项进行适当处理后，再替换掉原来那个匹配的项。

**[python]** [view plain](http://blog.csdn.net/joeblackzqq/article/details/7360004) [copy](http://blog.csdn.net/joeblackzqq/article/details/7360004)

1. #!/usr/bin/python
2. # coding=GBK
4. **import** re
6. # 对m作适当处理后返回结果
7. **def** fun(m):
8. **print**("in: %s" %m.group(0))
9. ret = m.group(0).upper()[::-1]
10. **return** ret
12. src = "what [can] I do for can you[can] come on"
13. pat = "(?<=

)(can)(?=

)"

1. #print(re.search(pat, src).group(1))
2. #result = re.sub(pat,lambda m:m.group(1).upper()[::-1], src)
4. # 使用lambda
5. result1 = re.sub(pat, **lambda** m:m.group(0).upper()[::-1], src)
6. **print**("result1: %s\n" %result1)
8. # 在re.sub中使用函数
9. result2 = re.sub(pat, fun, src)
10. **print**("result2: %s" %result2)

[http://static.blog.csdn.net/images/save_snippets.png](javascript:;)

运行输出：

**[plain]** [view plain](http://blog.csdn.net/joeblackzqq/article/details/7360004) [copy](http://blog.csdn.net/joeblackzqq/article/details/7360004)

1. [zcm@python #112]$./del.py
2. result1: what [NAC] I do for can you[NAC] come on
4. in: can
5. in: can
6. result2: what [NAC] I do for can you[NAC] come on
7. [zcm@python #113]$

看到了吗，**所有匹配"[can]"的项都被“转换成大写并逆顺”了。**

## 改用re.search检索全文

map()传入的第一个参数是f，即函数对象本身。由于结果r是一个Iterator，Iterator是惰性序列，因此通过list()函数让它把整个序列都计算出来并返回一个list。

你可能会想，不需要map()函数，写一个循环，也可以计算出结果：

具体看<https://my.oschina.net/zyzzy/blog/115096>

### Map函数

def abc(a, b, c):

return a\*10000 + b\*100 + c

list1 = [11,22,33]

list2 = [44,55,66]

list3 = [77,88,99]

r = map(abc,list1,list2,list3)

print(list(r))

### CODE re.search检索

def parallelRe(contentPath, spliter, query):

docs = getList(contentPath, spliter)

results = map(reSearch(query), docs)

print(sum(results))

def reSearch(pattern):

def f(doc):

if (re.search(pattern, doc)):

return 1

else:

return 0

return f

if \_\_name\_\_ == '\_\_main\_\_':

pathName = u"merge0"#不知道有什么用，每个add\_document多一个标签？？？

contentPath = './txtAndXls/merge0\_.txt'

spliter = r'------------\n'

querys = [ r"股东大会", r"日"]

for query in querys:

parallelRe(contentPath, spliter, query)

### CODE re.searchControl

from dir.searchingTool import \*

import re

# test = {"1":"2、2016年5月20日，顺丰控股股东大会通过决议，批准本次重大资产重组的\n"}

# string = test["1"]

#

# searchKey = u"股东大会"

# pathName = u"a"

# parseResults1 = paragraphParser(test, searchKey, pathName)

# print(result2strDict(parseResults1))

# print(parseResults1)

#

# reResult = re.search(r'股东大会', string)

# print(reResult)

def parallelRe(docs, docTitle, spliter, query):

results = list(map(reSearch(query), docs))

resultDict = {}

for index in range(len(results)):

if (results[index]):

resultDict[docTitle[index]] = docs[index]

# print(resultDict)

return resultDict

# resultDict = {}

# for

def loopReSearch(contentPath, spliter, querys):

docs = getList(contentPath, spliter)

docTitle = list(range(len(docs)))

for query in querys:

resultDict = parallelRe(docs, docTitle, spliter, query)

docs = []

docTitle = []

for (t, s) in resultDict.items():

docs.append(s)

docTitle.append(t)

# printDict(resultDict)

print(query, ":", len(docs))

printDict(resultDict)

def reSearch(pattern):

'''re.search找到则返回1否返回0'''

def f(doc):

if (re.search(pattern, doc, re.S)):

# print(doc)

return 1

else:

return 0

return f

if \_\_name\_\_ == '\_\_main\_\_':

pathName = u"merge0"#不知道有什么用，每个add\_document多一个标签？？？

contentPath = './txtAndXls/merge0\_.txt'

spliter = r'------------\n'

querys = [ r'股东大会', r'日', ]

for query in querys:

loopReSearch(contentPath, spliter, querys)

### 取出页码的分割器，多种分隔符匹配

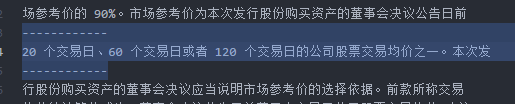
spliter = r'''Page\s[\d]+------------\n.+?\n------------[\n]+|

Page\s[\d]+------------\n|1-[\d]+\n------------\n|Page\s[\d]+------------\n.+?\n'''

Re.split(spliter, text)

### 因为日期和金钱显示大小不同导致分隔符出现的修复匹配

spliter = r'''([^。]\n)------------\n(.\*[\d,]{1,4}[\s]\*[年月日元].\*\n)|(.\*[\d,]{1,4}[\s]\*[年月日元].\*\n)------------\n'''

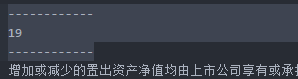


在atom搜索器中使用

------------

.+[年月日元].+

------------



最后：

spliter = r'''------------\n(.\*[\d,]{1,10}[\s]\*[^、].\*\n)|(\n.\*[\d,]{1,10}[\s]\*[^、].\*)\n------------'''

### 总结

前期pdf转html后，txt没有考虑到数字字体大小变化的特点

. 匹配符遇到换行会中断，其他不会

^[\d]{1,2}、.{1,20} 会匹配下图是因为没有在后面加上$

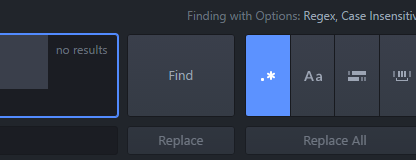


### Atom的正则匹配检索

------------

^[\d].+

------------



### 如何统计数量

用Python怎么统计一个列表的元素种类和各个种类的个数？

比如一个列表numbers=["cc","cc","ct","ct","ac"]。他的元素种类有3个（"cc","ct","ac"）,每个种类的个数分别为2，2，1

Kyoya斯MH62014-07-14

其他回答

res = {}

for i in numbers:

res[i] = res.get(i, 0) + 1

for (k, v) in res.items():

print(k,v)

## NLP

### 详细ntlk库资料汇总

<http://blog.csdn.net/huyoo/article/details/12188573>

### NTLK应用例子

<http://stackoverflow.com/questions/26662618/python-nltk-interpret-a-fixed-pattern-of-sentence-and-tokenize-it>

### ntlk+ 中文分词器

http://www.52nlp.cn/python自然语言处理实践-在nltk中使用斯坦福中文分词器

### [Python 使用nltk发掘文本实体间的联系](http://blog.csdn.net/guaguastd/article/details/43147525)

http://blog.csdn.net/guaguastd/article/details/43147525

### ntlk高级入门

http://www.ibm.com/developerworks/cn/linux/l-cpnltk/index.html

### 优秀大数据GitHub项目一览

https://www.shiyanlou.com/questions/3067

### python 自然语言处理笔记

<http://www.cnblogs.com/yuxc/category/307122.html>

### 学习nlp的建议

<https://www.quora.com/How-do-I-learn-Natural-Language-Processing>

### NLP优秀博客

介绍了基本的操作方法，但是是针对英文

<http://www.pythontip.com/blog/post/10012/>

说到了中文处理分词

http://www.52nlp.cn/python自然语言处理实践-在nltk中使用斯坦福中文分词器

## 开始NLP

### 安装

改用python3.4，按着

<http://www.nltk.org/>

指导下载后

Import nltk

Nltk.download()

下载所有的model

#### 中文分词器的下载

<http://nlp.stanford.edu/software/segmenter.shtml>

下载页：

http://nlp.stanford.edu/software/stanford-segmenter-2015-12-09.zip

### 测试

#### 中文：

from nltk.tokenize.stanford\_segmenter import StanfordSegmenter

segmenter = StanfordSegmenter(

path\_to\_jar="stanford-segmenter-3.6.0.jar",

path\_to\_slf4j = "slf4j-api.jar",

path\_to\_sihan\_corpora\_dict="./data",

path\_to\_model="./data/pku.gz",

path\_to\_dict="./data/dict-chris6.ser.gz")

sentence = u"这是斯坦福中文分词器测试"

segmenter.segment(sentence)

# >>> u'\u8fd9 \u662f \u65af\u5766\u798f \u4e2d\u6587 \u5206\u8bcd\u5668 \u6d4b\u8bd5\n'

segmenter.segment\_file("test.simp.utf8")

#### 英文：

import nltk

text = 'i am a good boy.you are a bad girl'

sens = nltk.sent\_tokenize(text)

print(sens)

words = []

for sent in sens:

words.append(nltk.word\_tokenize(sent))

for line in words:

print(line)

tags = []

for tokens in words:

tags.append(nltk.pos\_tag(tokens))

for tag in tags:

ners = nltk.ne\_chunk(tag)

print(ners)

### 利用jieba分析句子提取关键词

for (index, string) in dict.items():

tags = jieba.analyse.extract\_tags(string, topNum)

print(index, '\n', string, '\n关键词\n', tags)

## 新的pdf表格提取器

<https://github.com/ashima/pdf-table-extract>