日志标签 'Page Analysis'

单文档与多文档新闻摘要

2013年6月3日

1、自动文摘技术

文档自动摘要技术的研究比较成熟，在年前Yahoo 已经收购了自动新闻摘要应用Summly，又使得自动摘要重新火了一把。在当前市场的比较热门的摘要有Summly、Clipped等国外使用较为广泛的app，最近国内的今日头条在新闻客户端上也加上了摘要内容。即刻新闻移动客户端在新的版本计划中也加上了摘要的内容，这里对新闻的单文档和多文档摘要进行了调研分析。主要概述的论文：http://www.cs.cmu.edu/~nasmith/LS2/das-martins.07.pdf

一般来讲，文档自动摘要分为Extractive和Abstractive两种方法，任务可以分为单文档与多文档摘要两种。

目前大多数的自动摘要都是以Extractive进行的，只是抽取的方法有所改进。有通过原文的句子进行抽取分析、语料库进行机器学习、建立语义网模型等抽取方法。

基于Abstractive的方法也有很多人进行研究，基本思路是预先设计好一个模板，以及需要填充的字段。比如某新闻的发生时间、人物、地点等，利用计算机自动的在原文本中定位有关的信息片段，最后将这些片段填充到对应的模板的位置上。该方法能够产生较高质量的摘要，但是应用领域较为狭窄，模板不能统一。

2、单文档摘要技术

一个较为简单的Extractive的单文档摘要生成方法如下所示：

1. 对输入的文档进行分词处理，将一些停用词等与无关的词进行过滤
2. 提取每一个句子里面的实词（专有名词、动词等），对于所有的实词进行计算相似度和同义词分析
3. 对词义不同的词，计算其TF/IDF信息
4. 计算每个句子在整篇文章中的权重、句子在段落中的权重、段落与段落直接的相似度
5. 最后计算出每个句子在文档中的重要度，得到粗略的文摘
6. 对句子与句子直接进行重复检查
7. 最后得到整篇文摘的摘要

3、多文档摘要技术

多文档摘要的抽取一般有如下几类：

- 与单文档类似的方法
  - 通过计算词频、句子位置、主题词等抽取文档的重要内容，在通过多文档之间的相关性进行内容选择和过滤。
- 采用Abstractive的方法
  - 从多个文档中提取信息填充到预先定制好的模板系统中。
- 多文档集合判断的方法
  - 通过计算多文档的主题，作为该多文档的质心。判断文档中的句子与文档质心的距离，从而判断哪些句子比较重要。然后通过对句子安装相似程度进行聚类，最后从不同的类别中选取摘要句，从而减少摘要的冗余。

4、Summly的实现（猜测）

搜索：

搜索

近期文章

单文档与多文档新闻摘要
Level DB中的BloomFliter及Murmur Hash算法
采用可跳跃Trie树进行地理位置的多模匹配识别
单读单写无锁队列与单写多读无锁队列
Readability的Python实现

近期评论

文章归档

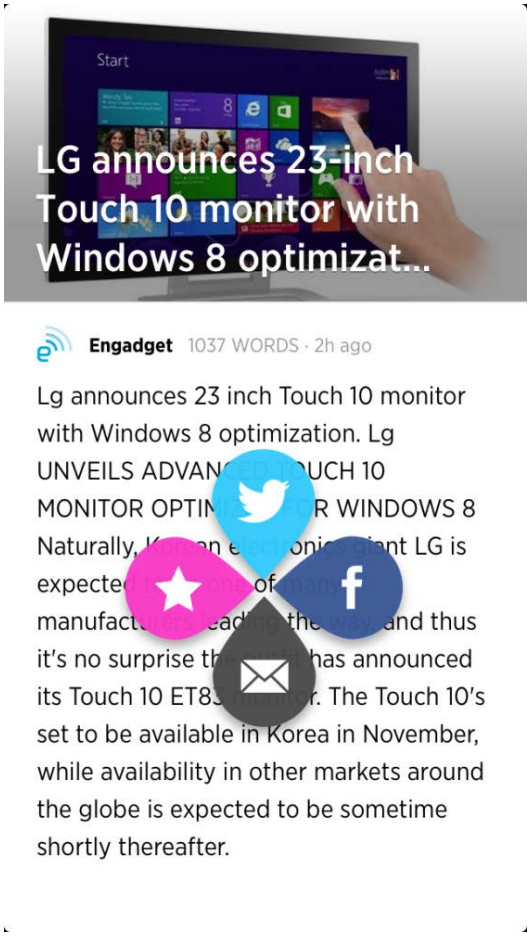
2013年六月
2013年三月
2012年十二月
2012年十月
2012年九月
2012年八月
2012年六月
2012年四月
2011年十月
2011年七月
2011年三月
2011年二月
2010年十月
2010年五月
2010年四月
2009年八月
2009年七月
2009年三月
2008年五月
2008年四月
2007年十月

分类目录

Projects
Rush
学习笔记
未分类
比赛
生活
算法
编程

功能

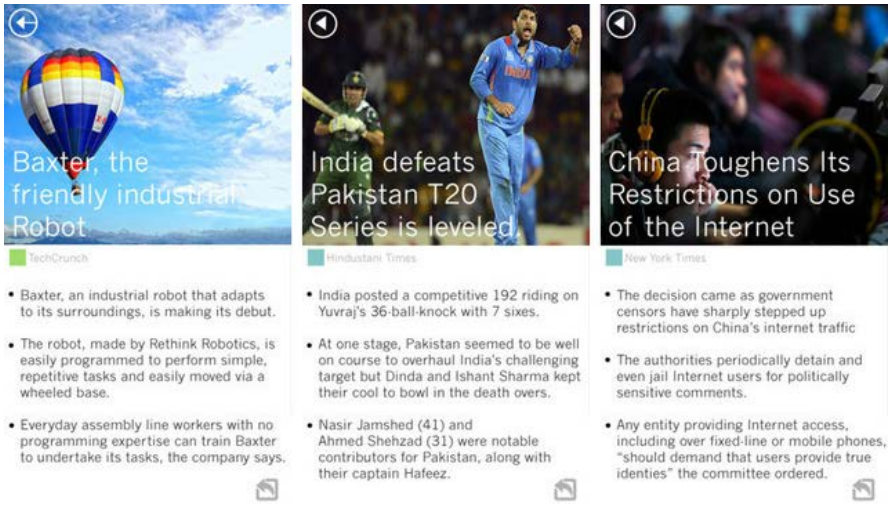
登录
文章RSS
评论RSS
WordPress.org



通过多次使用Summly，发现其是采用聚类分析后的新闻文档，然后选取与该topic中概括性（与topic的主题词、文档标题进行相似度计算）最大的段落作为摘要。

缺陷：　很多时候选取的摘要部分是第一短内容。有的时候会出现与原文毫无关系的一段话，尤其是在体育新闻中。

5、Clipped的实现



关于*Clipped*的自然语言处理原理，坦登告诉我们，*Clipped*能根据语法分析文本，并能识别哪种句子结构包含了最重要的信息。该算法通过分词标注器来分析一个句子，并能确定某一区块的信息与其他部分信息的依赖关系。通过统计和关键字的组合分析，*Clipped*能够对信息块的重要度进行排名，并选出那些关联系数最高的句子。然后*Clipped*根据分析结果生成内容摘要，最后还会重新读取它自动生成的摘要进行分析，以确保选定的信息内容是合理的。然后，该算法才会将最终结果呈现给用户阅读。

Clipped的主要思路还是单文档的Extractive来选取最为重要的三句话返回给用户。

缺陷：对于某些专业论文计算的比较好，但是普通的新闻有的时候很不连贯，无法概括全部的含义。

6、今日头条的实现（猜测）

今日头条的新闻很多都来源于微博，猜测是思路是根据新闻聚类将微博内容和新闻同时进行聚类（粒度较小），这样所有相关的微博和新闻都会到一个topic中，然后在选取摘要的时候，选取相似的微博内容作为摘要。当微博摘要不存在的时候，往往选取的是第一段内容。

缺陷：依赖新浪微博，时效性、领域性要求很强，即便是新浪微博的内容，质量也并不太高。

发表在 编程 没有评论 »
Tags: Page Analysis Python 摘要

Readability的Python实现

2012年10月15日

http://yanghao.org/tools/readability 做了一个将javascript实现的readability，采用python进行了实现。具体代码见https://github.com/kingwkb/readability

在使用的过程中主要有一些问题：

1、版权信息容易被抽取到,很多中文网站的版权区的class name等信息不一定可以枚举到
2、评论信息等很难被权重干掉，容易抽错误
3、多个div的页面不能够抽取完成
4、table表示的正文内容抽取不完全
5、标题抽取未计算<title>标签内的标题与里面<h1>等的相似度，标题抽取的准确率不高。

针对这些问题，我做了一些修改。

代码如下所示：

```
# -*- coding: utf-8 -*from __future__ import division
import os
import sys
import urllib
import urllib2
import urlparse
import re
import HTMLParser
import math
import urlparse
import posixpath
import time
import HTMLParser
import chardet
from BeautifulSoup import BeautifulSoup
#from bs4 import BeautifulSoupclass Readability:regexps = {
'unlikelyCandidates':
re.compile("combx|comment|community|disqus|extra|foot|header|menu|"
"remark|rss|shoutbox|sidebar|sponsor|ad-break|agegate|"
"pagination|pager|popup|tweet|twitter|login|Submit",re.I),
'okMaybeItsACandidate': re.compile("and|article|body|column|main|shadow", re.I),
'positive':
re.compile("article|body|content|entry|hentry|main|page|pagination|post|text|"
"blog|story",re.I),
'negative':
re.compile("combx|comment|com|contact|foot|footer|footnote|masthead|media|"
"meta|outbrain|promo|related|scroll|shoutbox|sidebar|sponsor|Submit|submit"
"shopping|tags|tool|widget|time|source", re.I),
'extraneous': re.compile("print|archive|comment|discuss|e[\-]?
mail|share|reply|all|login|"
"sign|single",re.I),
'divToPElements': re.compile("<(blockquote|dl|div|ol|ul|iframe)",re.I),
'replaceBrs': re.compile("(<br[^>]*>[ \n\r\t]*){2,}",re.I),
'replaceFonts': re.compile("<(/?)font[^>]*>",re.I),
'trim': re.compile("^\s+|\s+$",re.I),
'normalize': re.compile("\s{2,}",re.I),
'killBreaks': re.compile("(<br\s*/?>(\s| ?)*)+",re.I),
```

```
'videos': re.compile("http://(www\.)?(youtube|vimeo)\.com",re.I),
'skipFootnoteLink': re.compile("^\s*((\[|\【)?[a-z0-9]{1,2}(\]|\】)?
|^|edit|citation needed)\s*$",re.I),
'nextLink': re.compile(u"(下一页|下页|更多|【\d*】\s*|next|<|«)",re.I),
#// Match: next, continue, >, >>, » but not >|, »| as those usually mean last.
'prevLink': re.compile(u"(上一页|上页|^\d+\s*$|prev|earl|old|new|<|«)",re.I),
'comment': re.compile("<!-(.|\\s)*?->"),
}
chinese_punctuation = [u'，',u'。',u'！',u'、',u'？']
chinese_copyright = [u'版权与责任说明',u'凡本网注明',u'版权说明',u'免责声明',u'版权
属于',u'可能已被删除',
u'声明：',u'版 权 所 有',u'声明:','Copyright',u"版权申明：",
u"独家稿件声明",u"依法追究责任",u"保留追究其法律责任的权利"]
chinese_time_or_source = re.compile(u"(发表时间:|发表时间：|发布时间:|发步时
间：|201[0-9]年\d{1,2}月\d{1,2}日\s*\d{2}:\d{2}|201[0-9][\-\./][0-9]{1,2}[\-\./]
[0-9]{1,2}\s*\d{2}:\d{2}(:\d{2})?|来源：|来源:|发布会员：|当前位置：)",re.I)

chinese_others = [u"相关",u"热点",u"推荐",u"专题",u"更多",u"相关新闻",u"相关链接",
u"相关报道",u"相关阅读",u"相关文章",u"今日推荐",u"上一页",
u"上一篇",u"下一页",u"下一篇",u"延伸阅读",u"责编：",
u"本文被浏览",u"收藏",u"复制网址",u'复制成功',
u"分享到",u"loading…",u"加载中",u"开通微博",u"发布会员：",
'—','第1页',u'播主:',u" 播主：",u"网友评论",
u"邮箱：",u"传真：",u"电话：",u"字号",u"&gt;",
u"联系方式：",u"加入收藏",u"出错了",u"未经授权"]

def __init__(self, input, url):
"""
        url = "http://yanghao.org/blog/"
        htmlcode = urllib2.urlopen(url).read().decode('utf-8')

        readability = Readability(htmlcode, url)

        print readability.title
        print readability.content
        """
self.candidates = {}

self.input = input
self.url = url
self.input = self.regexps['comment'].sub("",self.input)
self.input = self.regexps['replaceBrs'].sub("</p><p>",self.input)
self.input = self.regexps['replaceFonts'].sub("<\g<1>span>",self.input)

self.html = BeautifulSoup(self.input)
#        print self.html.originalEncoding
#        print self.html
self.removeDisplayNone()
self.removeScript()
self.removeStyle()
self.removeLink()

self.title = self.getArticleTitle()
self.content = self.grabArticle()

def removeDisplayNone(self):
for elem in self.html.findAll(attrs={'style':'display:none'}):
elem.extract()

def removeScript(self):
for elem in self.html.findAll("script"):
elem.extract()

def removeStyle(self):
for elem in self.html.findAll("style"):
elem.extract()

def removeLink(self):
for elem in self.html.findAll("link"):
elem.extract()

def grabArticle(self):

for elem in self.html.findAll(True):
```

```
#print elem.get("id")
unlikelyMatchString = elem.get('id','')+elem.get('class','')

if self.regexps['unlikelyCandidates'].search(unlikelyMatchString) and \
not self.regexps['okMaybeItsACandidate'].search(unlikelyMatchString) and \
elem.name != 'body':
elem.extract()
continue
#                pass
if elem.name == 'div' or elem.name == 'textarea' or elem.name == 'td' or
elem.name == 'P':
#print elem.get('id')
s = elem.renderContents(encoding=None)
if not self.regexps['divToPElements'].search(s):
elem.name = 'p'

for node in self.html.findAll('p'):

parentNode = node.parent
grandParentNode = parentNode.parent
innerText = node.text

#            print '=================='
#            print node
#            print '_______'
#            print parentNode

if not parentNode or len(innerText) < 20:
continue

parentHash = hash(str(parentNode))
grandParentHash = hash(str(grandParentNode))

if parentHash not in self.candidates:
self.candidates[parentHash] = self.initializeNode(parentNode)

if grandParentNode and grandParentHash not in self.candidates:
self.candidates[grandParentHash] = self.initializeNode(grandParentNode)

contentScore = 1
for punctuation in self.chinese_punctuation:
contentScore += innerText.count(punctuation)*5
if contentScore == 1:
contentScore -= min(math.floor(len(innerText) / 30), 3)

for c in self.chinese_copyright:
contentScore -= innerText.count(c)*1000

for c in self.chinese_others:
contentScore -= innerText.count(c)*100

if self.chinese_time_or_source.search(innerText):
contentScore -= len(innerText)*100
#if contentScore < -200:
#    node.extract()
contentScore +=  min(math.floor(len(innerText) / 100), 3)

self.candidates[parentHash]['score'] += contentScore

#            print '======================'
#            print self.candidates[parentHash]['score']
#            print self.candidates[parentHash]['node']
#            print '________'
#            print node

if grandParentNode:
self.candidates[grandParentHash]['score'] += contentScore / 2

topCandidate = None

for key in self.candidates:
#            print '======================'
#            print self.candidates[key]['score']
#            print self.candidates[key]['node']

self.candidates[key]['score'] = self.candidates[key]['score'] * \
```

```
(1 - self.getLinkDensity(self.candidates[key]['node']))

if not topCandidate or self.candidates[key]['score'] > topCandidate['score']:
topCandidate = self.candidates[key]

content = ''

if topCandidate:
content = topCandidate['node']
#            print content
content = self.cleanArticle(content)

contentScore = 0
for punctuation in self.chinese_punctuation:
contentScore += content.count(punctuation)*5
parser = HTMLParser.HTMLParser()
content=parser.unescape(content)
if contentScore == 0:
return ''
if len(content) < 30:
return ''
return content

def cleanArticle(self, content):

self.cleanStyle(content)
self.clean(content, 'h1')
self.clean(content, 'object')
self.cleanConditionally(content, "form")

if len(content.findAll('h2')) == 1:
self.clean(content, 'h2')

self.clean(content, 'iframe')
self.cleanNextLink(content)
self.cleanConditionally(content, "table")
self.cleanConditionally(content, "ul")
self.cleanConditionally(content, "div")

self.fixImagesPath(content)

content = content.text
#content = content.renderContents(encoding=None)

#content = self.regexps['killBreaks'].sub("<br />", content)
content = re.compile("( )+",re.I).sub(" ", content)
return content

def clean(self,e ,tag):

targetList = e.findAll(tag)
isEmbed = 0
if tag =='object' or tag == 'embed':
isEmbed = 1

for target in targetList:
attributeValues = ""
for attribute in target.attrs:
attributeValues += target[attribute[0]]

if isEmbed and self.regexps['videos'].search(attributeValues):
continue

if isEmbed and
self.regexps['videos'].search(target.renderContents(encoding=None)):
continue
target.extract()

def cleanStyle(self, e):

for elem in e.findAll(True):
del elem['class']
del elem['id']
del elem['style']

def cleanNextLink(self, e):
tagsList = e.findAll('a')
```

```
for node in tagsList:
if self.regexps['nextLink'].search(node.text) or
self.regexps['prevLink'].search(node.text):
node.extract()

def cleanTimeAndSource(self, e):
tagsList = e.findAll('a')
for node in tagsList:
if self.regexps['nextLink'].search(node.text) or
self.regexps['prevLink'].search(node.text):
node.extract()

def cleanConditionally(self, e, tag):
tagsList = e.findAll(tag)

for node in tagsList:
weight = self.getClassWeight(node)
hashNode = hash(str(node))
if hashNode in self.candidates:
contentScore = self.candidates[hashNode]['score']
else:
contentScore = 0

if weight + contentScore < 0:
node.extract()
else:
p = len(node.findAll("p"))
img = len(node.findAll("img"))
li = len(node.findAll("li"))-100
input = len(node.findAll("input"))
embedCount = 0
embeds = node.findAll("embed")
for embed in embeds:
if not self.regexps['videos'].search(embed['src']):
embedCount += 1
linkDensity = self.getLinkDensity(node)
contentLength = len(node.text)
toRemove = False

if img > p:
toRemove = True
elif li > p and tag != "ul" and tag != "ol":
toRemove = True
elif input > math.floor(p/3):
toRemove = True
elif contentLength < 25 and (img==0 or img>2):
toRemove = True
elif weight < 25 and linkDensity > 0.2:
toRemove = True
elif weight >= 25 and linkDensity > 0.5:
toRemove = True
elif (embedCount == 1 and contentLength < 35) or embedCount > 1:
toRemove = True

if toRemove:
node.extract()

def getArticleTitle(self):
title = ''
try:
title = self.html.find('title').text
except:
pass

return title

def initializeNode(self, node):
contentScore = 0

if node.name == 'div':
contentScore += 5;
elif node.name == 'blockquote':
```

```
contentScore += 3;
elif node.name == 'form':
contentScore -= 3;
elif node.name == 'th':
contentScore -= 5;

contentScore += self.getClassWeight(node)

return {'score':contentScore, 'node': node}

def getClassWeight(self, node):
weight = 0
if 'class' in node:
if self.regexps['negative'].search(node['class']):
weight -= 25
if self.regexps['positive'].search(node['class']):
weight += 25

if 'id' in node:
if self.regexps['negative'].search(node['id']):
weight -= 25
if self.regexps['positive'].search(node['id']):
weight += 25

return weight

def getLinkDensity(self, node):
links = node.findAll('a')
textLength = len(node.text)

if textLength == 0:
return 0
linkLength = 0
for link in links:
linkLength += len(link.text)

return linkLength / textLength

def fixImagesPath(self, node):
imgs = node.findAll('img')
for img in imgs:
src = img.get('src',None)
if not src:
img.extract()
continue

if 'http://' != src[:7] and 'https://' != src[:8]:
newSrc = urlparse.urljoin(self.url, src)

newSrcArr = urlparse.urlparse(newSrc)
newPath = posixpath.normpath(newSrcArr[2])
newSrc = urlparse.urlunparse((newSrcArr.scheme, newSrcArr.netloc, newPath,
newSrcArr.params, newSrcArr.query, newSrcArr.fragment))
img['src'] = newSrc
def decode(s):
if type(s) == str:
code = chardet.detect(s)["encoding"]
if code == "GB2312":
s = s.decode("gb18030")
else:
s = s.decode(code)
return s
def main():
url= 'http://futures.jrj.com.cn/2012/07/03074513663141.shtml'
response = urllib2.urlopen(url)
html_str = response.read()

encode_str = decode(html_str).encode('utf-8')
#print encode_str
readability = Readability(encode_str,url)
print readability.content
if __name__ == '__main__':
start = time.time()
```

```
main()
#get_baidu_html()
end = time.time()
print 'Time Used: %r' % (end-start)
```

发表在 算法 没有评论 »
Tags: Page Analysis Python Readability

readability源码分析

2012年10月12日

From: http://stblog.baidu-tech.com/?p=79

以readability为例进行分析(http://code.google.com/p/arc90labs-readability)，readability是一个js库，通过自动化提取算法对网页的dom树进行改写，并在浏览器上展现页面抽取后的网页。工作流程：

1．清除JavaScript，css等html标签

2．根据定义的规则集合过滤结点，根据id和class字符串特征进行过滤

3．结点权重计算

a) 结点类型调权

b)结点Id和Class特征调权

c)逗号分隔调权

d)文本长度调权

e) 权重向父母转移

f)链接文本比的权重调整

4．定位文本统领文本结点，兄弟结点考虑

ContentScore最大的结点作为文本统领结点Max；

对Max的兄弟结点进行尝试，contentScore满足一定阈值且满足一些强制要求者，加入到dom树中。

发表在 算法 没有评论 »
Tags: Page Analysis Readability

正文抽取方法总结

2012年8月15日

如果是做正文抽取的话，想要做到很精准的效果是难的（尤其是准确剔除掉正文周边内容），尤其是来自一些不正规的站点网页。网页信息抽取的方法有很多，比如从算法上分：基于模板的，基于信息量、基于视觉的、基于语义挖掘的、基于统计的。从HTML处理上分为：基于行块、基于DOM树。

- 1.一般方法，基于文本信息（行密度、行长度、链接文字比、文本中标点符号比例、图片、视频等），计算正文在源码哪些行上分布较多，取正文较多的行；有算法是根据行的正文密度来计算的，简单点说就是正文长度/标签数量。一般的程序编写，就是建立Dom树，把再计算Dom树上某个节点的评估函数。
  对于大多数新闻类网站，这个方法会工作得很好。但是需要考虑以下问题：1、文章长度比较短、信息比较分散的内容 2、版权信息或者网站说明的过滤。
- 2. 基于视觉的页面分割算法，是基于分块的算法的一种具体实现方式，这是微软亚洲研究院的一个算法。一是根据视觉来分块，二是根据视觉来进行块合并。基于视觉处理较复杂，需要用到CSS、Javascript等引擎，需要用浏览器内核库来处理HTML，性能可能不高。另外，这个算法的结果只是告诉大家网页大概可以分为多少块，每一块的位置、大小是什么，而哪块和哪块是正文还需要进一步计算。计算量和下载量都比较大。
- 3. 基于语义的正文抽取，根据锚文本和页面标题等不容易出错的信息去发现正文块，这类算法有效，但是仍有局限性。实际很少采用这种方案。
- 4. 基于统计的，基于分块和统计相结合的新闻正文抽取 和 基于同层网页相似性去除网页噪音。前者利用统计是找到同一网页里面的正文块，后者是链接同一路径下的不同网页的相似度去除噪音，两者是有区别的。基于统计，可以减少个别网页的差异带来的误差，提高准确度。

主要的开源程序：

Readability https://code.google.com/p/arc90labs-readability/ ， javascript语言

Readability C#、Python、Node.js、Objective-C 版代码
见http://blog.arc90.com/2009/06/20/readability-now-available-in-three-delicious-flavors/

cx-extractor 基于行块分布函数的通用网页正文抽取：线性时间、不建DOM树、与HTML标签无关 https://code.google.com/p/cx-extractor/

boilerpipe Boilerplate Removal and Fulltext Extraction from HTML pages
https://code.google.com/p/boilerpipe/

发表在 算法 没有评论 »
Tags: Page Analysis 正文抽取

Python 进行网页抓取（utf-8编码转换和基于webkit的抓取）

2012年6月29日

1、抓取与编码转换

对于抓取网页来说，python提供了urllib和urllib2来获取html源代码，也可以使用wget来下载页面。

一般抓取的网页都需要转换成UTF-8编码，很多中文网页采用GB2312、GBK等编码方式，尤其是GB2312的网页，其实里面含有非GB2312的字符，就会导致读取错误。

一般的抓取代码：
[gist]https://gist.github.com/xddaijun/6319585[/gist]

2、特殊网页的抓取

特殊网页：采用ajax进行传输的内容，通过javascript加载等直接读取HTML无法分析和抽取的。需要使用PhantomJS 进行处理得到准确的HTML。

PhantomJS 是一个基于webkit内核的javascript api，目前是是1.6版本。项目地址：https://code.google.com/p/phantomjs/

如这个网页：http://pic.news.sohu.com/911196/911317/group-362161.shtml#0，将里面的大图片采用javascript来进行下载，直接读取HTML获取不了大图。部分HTML代码如下：

```
<!--以上代码不要删除-->
                    </div>
                    <!--单图-->
                    <div id="picPlayerWrap" class="pic">
                        <a href="javascript://"><img src="http://images.sohu.com/ccc.gif" style="width:800px;height:480px;"/></a>
                    </div>


                    <!--过场-后台输出-->
```

这样就需要有基于webkit的爬虫来获取HTML代码，PhantomJS正好来可以完成该功能。

获取网页源代码的PhantomJS可以这么实现：
gethtml.js
[gist]https://gist.github.com/xddaijun/6319591[/gist]
然后运行

phantomjs gethtml.js http://pic.news.sohu.com/911196/911317/group-362161.shtml#0

即可以获取到经过JS运行过的HTML代码:

```
<!--以上代码不要删除-->
                    </div>
                    <!--单图-->
                    <div id="picPlayerWrap" class="pic">
                        <a href="http://m3.biz.itc.cn/pic/new/n/66/12/Img3971266_n.jpg"><img src="http://m3.biz.itc
.cn/pic/new/n/66/12/Img3971266_n.jpg" style="visibility: visible; height: 593px; width: 939px; opacity: 1; padding-top: 0px; displa
y: block; "></a>
                    </div>
                    <!--过场-后台输出-->
```

然后再将得到的HTML代码进行进一步的抽取，从而得到该页面的大图。

发表在 算法 没有评论 »
Tags: javascript Page Analysis Python webkit 抓取

Python 一句话抓取百度搜狗的热词

2012年6月19日

- 需要依赖 python, pyquery

[gist]https://gist.github.com/xddaijun/3115077[/gist]

发表在 编程 没有评论 »
Tags: Page Analysis 抓取 正文抽取