

海量数据爬虫系统

南京大学自然语言处理组

version 1.0.0

2013 年 11 月 4 号

目 录

前 言	1
1 戴老师负责部分	2
1.1 微博用户云图展示	2
1.1.1 云图	2
1.2 政府应用	2
1.3 经济应用 1	2
1.4 经济应用 2	2
1.5 民生应用	2
1.6 结束语	3
2 邹远航负责部分	4
2.1 新浪微博模拟登陆实现	4
2.1.1 登陆过程	4
2.1.1.1 模拟登陆新浪微博, 保存 cookie	4
2.2 新浪微博评论提取实现	6
2.2.1 base62 编码和十进制数字的相互转换	6
2.2.2 mid 至 url 转换	7
2.3 怎样使用 python 操作 mysql 数据库	8
2.3.1 安装 mysqlldb	8
2.3.2 mysql 相关信息	8
2.3.3 怎样使用 MySQLdb	8
2.4 服务器信息	9
2.5 mongodb 集群部署	9
3 吕萧负责部分	11
3.1 引言	11
3.2 从微博搜索页面 (s.weibo.com) 获取数据	11
3.2.1 文件一览表	12
3.2.2 抓取搜索内容	13
3.2.3 从抓取的页面中提取信息	14
3.2.4 将提取出的信息存储到 mongodb	14

3.3	数据分析部分的 Lasso 相关词分析	14
3.3.1	文件一览表	14
3.3.2	(Global View) 生成词典, 统计词频等信息生成词矩阵	14
3.3.3	(Local View) 根据上一步生成的词典, 对关键词生成 index	14
3.3.4	调用 lasso 代码, 生成相关词	14
4	黄家君负责部分	15
4.1	引言	15
4.2	存储设计	15
4.2.1	WeiboUser	15
4.2.2	UserInfo	16
4.2.3	Follow	16
4.2.4	MicroBlog	17
4.3	Webpage Parsing	18
4.3.1	Profile Parser	18
4.3.1.1	URL Pattern	18
4.3.1.2	页面元素定位	18
4.3.2	Follow Parser	18
4.3.2.1	URL Pattern	19
4.3.2.2	页面元素定位	19
4.3.3	MicroBlog Parser	19
4.3.3.1	URL Pattern	19
4.3.3.2	页面元素定位	20
5	程川负责部分	22
5.1	引言	22
5.2	关于该部分代码实现的一些说明	22
5.2.1	所需的 python 依赖包	22
5.2.2	注释说明	22
5.3	框架类图及说明	22
5.3.1	类图	22
5.3.2	实现说明	23
5.3.2.1	负责模拟登录的模块: loginer.py	23
5.3.2.2	负责数据库维护的模块: storage_manager.py	23

5.3.2.3	负责日志写入的模块: <code>my_log.py</code>	23
5.3.2.4	主要的调度模块: <code>weibo_scheduler.py</code>	24
5.3.3	高级微博搜索的公用模块: <code>advance_weibo_frame.py</code>	25
5.3.3.1	<code>URLWrapper</code>	25
5.3.3.2	<code>PageParser</code>	25
5.3.3.3	<code>WeiboCrawler</code>	27
5.3.3.4	<code>DataStorer</code>	28
5.3.4	与具体任务相关的模块: <code>advance_keyword_hot_weibo.py</code> 、 <code>advance_keyword_real_weibo.py</code>	28
5.3.5	异常类模块: <code>my_exceptions.py</code>	28
5.3.6	用户指定的关键字维护模块: <code>keyword_processor.py</code>	28
5.3.7	任务无关模块（即对所有任务来说都一样的模块，不涉及类型的差异）	28
5.3.8	添加任务示例	28
5.4	存储设计	30
5.4.1	微博	30
5.4.2	评论	30
5.4.3	日志	31
5.4.4	关键词	31
6	论坛的实时爬取	33
6.1	功能描述	33
6.2	存储设计	33
6.3	<code>pipelines</code>	34
6.4	<code>Downloader Middleware</code>	35
6.4.1	<code>RandomUserAgentMiddleware</code>	35
6.4.2	<code>The404Middleware</code>	35
6.4.3	<code>MyRedirectedDownloaderMiddleware</code>	35
6.5	启动多个 <code>Spider</code>	35
6.6	杂项: <code>settings</code>	35
	致 谢	37
	参考文献	37

插图目录

1.1	云图	2
2.1	get-content	4
2.2	post-data	5
2.3	mid,url 转换图	7
4.1	Profile 页面	19
5.1	类关系图	23
5.2	高级关键字搜索单条微博结构图	27

前 言

海量数据爬虫系统，正在做...

1 戴老师负责部分

1.1 微博用户云图展示

offline demo, 抓取若干大 V 用户 weibo 数据, 或者官媒发布的 weibo 数据, 构建云图

1.1.1 云图



图 1.1 云图

1.2 政府应用

抓取 11 月份, 搜索包含”北京”、”上海”、”南京”的 weibo 数据。去除 stop words 之后按照词频去除前 10 的词汇。每个词汇, 通过特征提取的方法, 将对应一个当地的事件 (由若干的关键词汇构成)。通过构造时序图, 分析描述该事件的关键词的演变过程。

1.3 经济应用 1

抓取 11 月 8 日 -11 月 12 日, 关于”京东”, ”天猫”, ”易购”等关键词, 通过 weibo 的数量, 评论的数量, 转发的数量, 粉丝的差异性, 内容的聚类等等, 进行系列的分析。

1.4 经济应用 2

抓取特定股票的 weibo

1.5 民生应用

关注雾霾、PM2.5、污染等.....

1.6 结束语

[对本文工作进行总结]

2 邹远航负责部分

2.1 新浪微博模拟登陆实现

由于模拟登陆微博是爬虫爬取微博信息的第一步，所以在这里先介绍一下怎么模拟登陆微博（以下以新浪微博为例）

2.1.1 登陆过程

登陆过程分两步：

- (1) 模拟登陆新浪微博，保存 cookie
- (2) 利用 cookie 信息直接访问微博 url

第一步是讨论的重点。要想更好的模拟这一步，一个好的网络监视工具是必须的，可以使用 Firefox 的 Firebug 或者 Firefox 的 httpfox 或者 Chrome（Chrome 的网络交互报文可以到 network 部分查看）

下面的例子使用的是 Firefox 浏览器以及 httpfox 插件

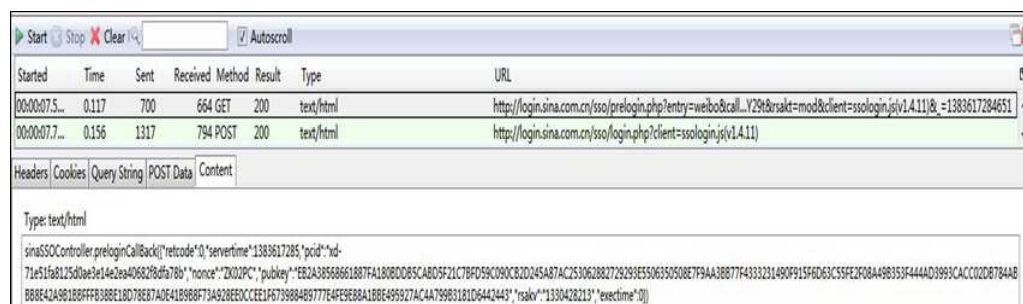
2.1.1.1 模拟登陆新浪微博，保存 cookie

打开<http://weibo.com/>（如果以前设置过让浏览器自动记住密码并自动登陆，需要清除 cookie），通过网络监视工具会发现在用户登录过程中，浏览器与服务器有三次交互，分别如下

1) Get 请求

在用户名一栏输入微博账号后，当你的焦点离开用户名输入框，浏览器会自动向下面链接发送一个 Get 请求，

[http://login.sina.com.cn/sso/prelogin.php?entry=weibo&callback=sinaSSOController.preloginCallBack&su=&rsakt=mod&checkpin=1&client=ssologin.js\(v1.4.11\)](http://login.sina.com.cn/sso/prelogin.php?entry=weibo&callback=sinaSSOController.preloginCallBack&su=&rsakt=mod&checkpin=1&client=ssologin.js(v1.4.11))



Started	Time	Sent	Received	Method	Result	Type	URL
00:00:07.5...	0.117	700	664	GET	200	text/html	http://login.sina.com.cn/sso/prelogin.php?entry=weibo&callback=sinaSSOController.preloginCallBack&su=&rsakt=mod&checkpin=1&client=ssologin.js(v1.4.11)&_=1383617284651
00:00:07.7...	0.156	1317	794	POST	200	text/html	http://login.sina.com.cn/sso/login.php?client=ssologin.js(v1.4.11)

Headers	Cookies	Query String	POST Data	Content
Type: text/html sinaSSOController.preloginCallBack({"retcode":0,"servertime":1383617285,"nonce":"71e51fa8125d0ae3e14e2ea40682f8d7a78b","pubkey":"ER2A38568661887FA180BDD85CABD5F21C78FD59C090CB2D245A87AC251062882729299E5506350508E7F94A38877F4333231490F915F6D63C55FE2F08A498353F444AD3993CACCC02D8784AB888E42A9B188FF8388E18D78E87A0E418988F73A928E0CCE1F673988A8977E4FE9E88A188E46927AC4A79983181D6442443","rsakv":"1330428213","exectime":0})				

图 2.1 get-content

得到的数据中有“servertime”和“nonce”和“rsakv”的值，这 3 个值是后面需要的，其

它值，目前是没用的

2) POST 请求

该部分用来提交用户信息给服务器并由服务器判断用户信息是否正确，从而判断是否登陆成功，通过 `httpfox`，可以观察 POST 报文的相关信息

(1) POST 信息的截图：

POST 的 POST-DATA 信息如下：

Headers	Cookies	Query String	POST Data	Content
Type: application/x-www-form-urlencoded				
Parameter	Value			
entry	weibo			
gateway	1			
from				
savestate	0			
useticket	1			
pagerefer				
vsnf	1			
su	eXVhbmhhbmcbmp1JTQwZ21haWwuY29t			
service	miniblog			
servertime	1383617285			
nonce	ZK02PC			
pwencode	rsa2			
rsakv	1330428213			
sp	0fee7b9bcd5d06569b66d77183292e2c7ecee4fe258772eda4c2a5bfd8c921cae106c311b2201278271462926d0aa80898cabd49dac4ec			
encoding	UTF-8			
prelt	113			
url	http://weibo.com/ajaxlogin.php?framelogin=1&callback=parent.sinaSSOController.feedBackUrlCallBack			
returntype	META			

图 2.2 post-data

(2) POST 数据分析

”su” 是加密后的 username，”sp” 是加密后的 password，”servertime”、” nonce” 和”rsakv” 是上一步 GET 请求返回的 JSON 串中的内容, 其他参数不变

su 是 username 经过 BASE64 计算得来的: `su = base64.encodestring(urllib.quote(username)[:-1]`

sp 的加密算法可能会经常变动, 目前新浪采用的是 RSA 算法（可以通过查看 js 文件获取加密算）

如果以上各步进展的比较顺利，会得到如下响应：

```
location.replace("http://weibo.com/ajaxlogin.php?framelogin=1&callback=parent.sinaSSOController.f
&retcode=101&reason=%B5%C7%C2%BC%C3%FB%BB%F2%C3%DC%C2%EB%B4%ED
%CE%F3");
```

这是一条自动跳转语句，里面包含了准备跳转的 URL，如果正确登陆，则这个 URL 中的 `retcode` 的值为 0，否则登陆错误，仔细检查上面的步骤

3) 跳转到对应微博页面

上面已经得到待跳转的 url，只需请求这个 URL 即可。需要注意，要想服务器知道

你是否登陆，这一步的请求需要用到前面操作的 cookie 信息（因为第二步中登陆成功与否的信息会自动记录在 cookie 中）

所以我们在整个程序的一开始要打开 cookie，保证以后的操作都能正确的使用 cookie. 对于 python 的用户，不要定制 header 了，默认情况下 urllib2 会自动打包 cookie 信息

2.2 新浪微博评论提取实现

使用新浪微博 API 时，有时需要微博的 url，但是如 statuses/public_timeline 等接口中取得的微博 status 的字段中并没有。不过，status 中包含了一个 mid 字段，通过 mid，我们实际上是可以计算出 url 的。在计算的过程中需要使用 base62 编码。

base62 编码, 是十进制和 62 位进制的互换。对于 62 进制，从 0 数到 9 以后，10 用小写字母 a 表示，接着数完 26 个字母，到 z 为 35，然后 36 为大写字母 A，一直到 61 为大写字母 Z。

2.2.1 base62 编码和十进制数字的相互转换

代码实现如下：

```
ALPHABET = "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"

def base62_encode(num, alphabet=ALPHABET):
    """Encode a number in Base X

    `num`: The number to encode
    `alphabet`: The alphabet to use for encoding
    """
    if (num == 0):
        return alphabet[0]
    arr = []
    base = len(alphabet)
    while num:
        rem = num % base
        num = num // base
        arr.append(alphabet[rem])
    arr.reverse()
    return ''.join(arr)
```

```
def base62_decode(string , alphabet=ALPHABET):
    """Decode a Base X encoded string into the number

    Arguments:
    - `string`: The encoded string
    - `alphabet`: The alphabet to use for encoding
    """
    base = len(alphabet)
    strlen = len(string)
    num = 0

    idx = 0
    for char in string:
        power = (strlen - (idx + 1))
        num += alphabet.index(char) * (base ** power)
        idx += 1

    return num
```

2.2.2 mid 至 url 转换

新浪微博 ur 形如: <http://weibo.com/2991905905/z579Hz9Wr>, 中间的数字是用户的 uid, 重要的是后面的字符串 **z579Hz9Wr**。

对于一个 mid, 从后向前每 7 位一组, 用 base62 编码来 encode, 拼起来即可。需要注意的是, 每 7 个一组的数字, 除了开头一组, 如果得到的 62 进制数字不足 4 位, 需要补足 0。

1) mid, url 转换图



图 2.3 mid,url 转换图

2) 代码实现

```
def mid_to_url(midint):
```

```
midint = str(midint)[::-1]
size = len(midint) / 7 if len(midint)
result = []
for i in range(size):
    s = midint[i * 7: (i + 1) * 7][::-1]
    s = base62_encode(int(s))
    s_len = len(s)
    if i < size - 1 and len(s) < 4:
        s = '0' * (4 - s_len) + s
    result.append(s)
    result.reverse()
return ''.join(result)
```

2.3 怎样使用 python 操作 mysql 数据库

这里通过 MySQLdb 这个 mysql 数据库的 python 接口来操作数据库.

2.3.1 安装 mysqlldb

mysqlldb 下载链接<https://pypi.python.org/pypi/MySQL-python>, 下载后解压缩参见 INSTALL 文件, 来进行安装

mysqlldb 使用说明<http://mysql-python.sourceforge.net/MySQLdb.html>

2.3.2 mysql 相关信息

在 mysql 数据库中新建了两个数据库, weibo 和 weibosearch. 可以使用用户名 njunlp, 密码 njunlp 进行访问, 访问方法见下一节

在这两个数据库里分别建立了数据表 [crawl_log](#), 字段参考 [xialiang](#) 给的文档。

2.3.3 怎样使用 MySQLdb

这里以远程主机 114.212.189.55, 用户名 njunlp, 密码 njunlp 来访问数据库 weibo

```
import MySQLdb as mdb
import sys

#how to connect database named weibo in mysql server
con = mdb.connect('114.212.189.55', 'njunlp', 'njunlp', 'weibo');
```

```
with con:
```

```
    cur = con.cursor()
    #execute SQL to create table and write information in table
    cur.execute("CREATE TABLE IF NOT EXISTS \
        examples(Id INT PRIMARY KEY AUTO_INCREMENT, Name VARCHAR(25))")
    cur.execute("INSERT INTO examples(Name) VALUES('Jack London')")
    cur.execute("INSERT INTO examples(Name) VALUES('Honore de Balzac')")
    cur.execute("INSERT INTO examples(Name) VALUES('Lion Feuchtwanger')")
```

2.4 服务器信息

四台服务器，访问方式如下

(1) ssh root@114.212.190.201 -p15711

(1) password:36ErwT

(2) ssh root@114.212.190.201 -p11856

(1) password:57DzYL

(3) ssh root@114.212.190.201 -p18351

(1) password:Daixinyu@123

(4) ssh root@114.212.190.201 -p14266

(1) password:Daixinyu@N1

2.5 mongodb 集群部署

这里使用三个节点，组成一个集群，其中一个作为主节点，其余作为从节点

主节点作为读写节点，其余节点不具有读写能力，从节点负责复制数据。

主节点的数据会异步的复制到次节点，这样在主节点发生故障时候，可以从从节点恢复数据。

这个集群实现了自动故障转移，当某个主节点发生故障的时候，会自动的转移到此节点上。

主节点信息是172.16.84.3

次节点信息是172.16.146.2

仲裁节点信息是172.16.146.3

我们需要连接的节点是主节点，访问主节点跟访问普通的节点一样

配置过程：

(1) 首先在每台机器上安装 mongodb，参考<http://docs.mongodb.org/manual/tutorial/install-mongodb-on-ubuntu/>

(2) 在每台机器上编辑文件 `/etc/mongodb.conf`，以主节点为例，填写下面内容

- (1) `port = 27017`
- (2) `bind_ip = 172.16.84.3`
- (3) `fork = true`
- (4) `replSet = nju`

(3) 在每台机器上执行下面的命令

- (1) `sudo mongod -config /etc/mongodb.conf`

(4) 依次在主节点执行下面的命令

- (1) `mongo`
- (2) `rs.initiate()`
- (3) `rs.add("172.16.146.2")`
- (4) `rs.add("172.16.146.3")`

(5) 使用下面的命令查看集群状态

- (1) `rs.conf()`

3 吕萧负责部分

3.1 引言

吕萧负责的部分包括新浪微博搜索部分的数据获取，和数据分析部分的 Lasso 相关词分析

- 新浪微博搜索部分的数据获取，大致分为三个部分
 - (1) 根据用户输入的关键词、给定的时间段，爬取新浪微博搜索页面的内容
 - (2) 从爬取的页面中提取 tweets、评论等信息
 - (3) 将提取出的信息存储到 mongodb
- 数据分析部分的 Lasso 相关词分析，工作流程大致分为一下几个部分
 - (1) (Global View) 生成词典，统计词频等信息生成词矩阵
 - (2) (Local View) 根据上一步生成的词典，对关键词生成 index
 - (3) 调用 lasso 代码，生成相关词

以下几个小节将会具体解释各个功能的实现

3.2 从微博搜索页面 (s.weibo.com) 获取数据

爬虫部分的数据流如下：

- (1) 用关键词 (KEY) 生成一个 `crawl_keyword()` 类的对象 `s`

$$s = \text{crawl_keyword}(\text{KEY})$$

,

`crawl_keyword` 类在 `search.py` 中定义。

- (2) 调用 `crawl_keyword` 类中的 `CommonSearch()` 或者 `AdvSearch()` 方法，爬取相关的页面，例如

$$s.\text{AdvSearch}(2014, 3, 1, 0, 2014, 3, 20, 23)$$

,

爬取以 **KEY** 为关键字的，**2014 年 3 月 1 日 0 点到 2014 年 3 月 20 日 23 点 59 分** 之间的微博。

- (3) 在 `CommonSearch()` 或者 `AdvSearch()` 方法中, 爬取一个页面 `P` 后, 将调用 `storage_mongoengine.py` 中的 `store()` 函数

$$store(P)$$

,

将爬取的页面内容处理后存起来。

- (4) 存储模块中的 `store()` 函数处理页面 `P` 的时候, 首先调用 `parser` 解析页面, 提取微博内容、时间等关键信息, 然后将提取出的信息写入 `mongodb`. 例如,

$$tweets = page2tweets(P)$$

调用 `page2tweets()` 函数将页面中的多条微博提取出来, 放到 `tweets` 列表中;

$$mid = midoftweet(tweet)$$

调用 `midoftweet()` 函数, 提取一条微博的 `mid`。

其中 `page2tweets()`、`midoftweet()` 等函数属于 `parser` 模块, 在 `searchparser_json.py` 文件中定义。

3.2.1 文件一览表

微博搜索页面爬取的代码在 `weibosearch` 文件夹中, 包括以下文件:

- `base62.py`, 其中的 `mid2url` 函数根据微博 `id(mid)` 生成这条微博的 `http` 链接的后缀, 在爬取某一条微博的评论 (`comments.py`) 时用到
- `comments.py`, 用于爬取指定 `mid` 的微博下的评论数目和内容
- `conf.py`, 用于读取配置文件 `weibosearch.yaml` 中的配置信息
- `cookie_outofdate.dat`, 一个过期的 `cookie` 文件, 留待考察。登陆时使用的 `cookie` 文件在一定条件下会过期, 导致登陆失败。发生登陆失败时, 现在的处理是删除过期的 `cookie` 文件, 重试。

- **cookie_outofdate.html**, **cookie** 过期时爬到的页面。根据这个页面里特有的“**r'http://weibo.com/sso/login.php?ssosavestate'**”字符串来判断爬取当前页面是否遇到了 **cookie** 过期的问题
- **errorlog.py**, 定义错误类型, 定义 **errorlog** 类, **errorlog** 类的方法在出错时用于将对应的错误写入 **log** 文件
- **login.py**, 用于新浪微博用户登陆, 登陆过程中如果没有 **cookie** 文件, 会对当前每个 **username** 创建一个; 已有的话, 会直接使用现有的 **cookie** 文件。几乎所有的页面爬取都需要在用户登陆后进行
- **search.py**, 根据用户输入的关键字和时间区间, 爬取相关的页面, 调用 **storage** 模块, 将相关信息存入数据库
- **searchparser_lxml.py**, 用 **lxml** 和 **xpath** 工具写的搜索页面的 **paser**, 用于提取微博内容, **uid**, **mid** 等信息
- **storage_mongoengine.py**, 存储模块, 被 **search.py** 调用。存储前调用 **parser** 提取信息, 再将相关信息存入 **mongodb**
- **weibo_login_cookies.dat**, 在登陆过程中生成的 **cookie** 文件
- **weibosearch.yaml**, 用户 **id**、密码, 数据库名字, **ip** 等配置信息

3.2.2 抓取搜索内容

抓取新浪微博搜索页面的功能主要在 **search.py** 文件中实现。

search.py 文件中定义了 **crawl_keyword** 类, 其中

- **CommonSearch()** 方法用于一般的关键字搜索;
- **AdvSearch()** 方法用于指定时间段的关键字搜索 (只支持当月);
- **_howmanypgs()** 方法用于根据搜索结果的第一页的内容确定需要爬多少个页面;
- **urlencoding()** 方法用于根据关键字生成相应的 **url** 链接;
- **getpage()** 方法用于根据 **url** 获取页面。

创建 `crawl_keyword` 类时，在初始化中需要给定 `keyword` 和数据库名字的后缀。爬取得到的相关结果会存储到指定后缀的数据库集合中。

使用 `crawl_keyword` 类时，调用 `CommonSearch()` 或 `AdvSearch()`，会爬取相关页面，并由 `CommonSearch()` 或 `AdvSearch()` 调用 `storage` 模块。`storage` 模块先调用 `parser` 提取有用的信息，然后存储信息到 `mongodb`。

3.2.3 从抓取的页面中提取信息

新浪微博搜索页面的 `parser`，主要在 `searchparser_json.py` 中实现，用到 `lxml` 库和 `xpath` 方法。

`parser` 中实现了以下一些方法用于从爬取的页面提取相关信息，包括：

- `page2tweets()`, 从一个页面中提取多条 `tweets`;
- `midoftweet()`, 从一条 `tweet` 中提取它的 `mid`;
- `uidoftweet()`, 从一条 `tweet` 中提取 `po` 它的用户的 `uid`;
- `cntnt_tweet()`, 从一条 `tweet` 中提取它的文本内容;
- `weibois_forward()`, 判断一条 `tweet` 是否是转发, 返回 `False` 表示这条 `tweet` 是原创的;
- `ref_forword()`, 从一条转发的微博中找到它的原微博链接;
- `cntnt_forward()`, 从一条转发的微博中找到它的原微博内容;
- `info()`, 从一条微博中找到它的时间戳, 被赞次数, 被转发数, 评论数。返回值为一个长为 4 的列表, `[time, n_like, n_forward, n_comment]`

3.2.4 将提取出的信息存储到 `mongodb`

3.3 数据分析部分的 Lasso 相关词分析

数据分析部分的代码在 `data-analysis` 文件夹中。

3.3.1 文件一览表

3.3.2 (Global View) 生成词典, 统计词频等信息生成词矩阵

3.3.3 (Local View) 根据上一步生成的词典, 对关键词生成 `index`

3.3.4 调用 `lasso` 代码, 生成相关词

4 黄家君负责部分

4.1 引言

本人负责针对给定微博用户的微博爬取实现，即，给定一个微博用户的 UID，实现爬取该用户的个人资料 (Profile)，粉丝 (followers)，关注 (followees) 以及其所有可见微博。

子项目目录为/weibo。

4.2 存储设计

使用 MongoDB 实现结构化存储。

在 Python 中使用 MongoDB，可以用 PyMongo 或者 MongoEngine，我选用了后者。利用 MongoEngine 可以像操作 Python 对象一样来操作 MongoDB 的 Document。

在本子项目中，需要存储的主要有两个，一是微博用户 (WeiboUser) 的信息，二是每条微博 (MicroBlog) 的信息。相关的代码在/weibo/storage.py 中。以下介绍这两种数据结构的组成。

4.2.1 WeiboUser

WeiboUser 类的代码如下：

```
class Weibo(Document):
    uid = StringField(required=True)
    last_update = DateTimeField()
    last_mid = StringField()

    info = EmbeddedDocumentField(UserInfo)
    followees = ListField(EmbeddedDocumentField(Follow))
    followers = ListField(EmbeddedDocumentField(Follow))
```

其中，uid 是用户的唯一标识，last_update 是用户最后一次更新微博的时间，last_mid 是用户最新的那条微博的 id。info 是用户的个人信息，是 UserInfo 类的实例，UserInfo 类的数据结构下面将会给出。followees 和 followers 分别是关注列表和粉丝列表，列表的元素都是 Follow 类的实例，Follow 类的结构同样在下面给出。

4.2.2 UserInfo

```
class UserInfo(EmbeddedDocument):
    nickname = StringField()
    location = StringField()
    sex = StringField()
    birth = StringField()
    blog = URLField()
    site = URLField()
    intro = StringField()

    email = EmailField()
    qq = StringField()
    msn = StringField()

    edu = ListField(EmbeddedDocumentField(EduInfo))
    work = ListField(EmbeddedDocumentField(WorkInfo))
    tags = ListField(StringField())

    n_followees = IntField()
    n_followers = IntField()
    n_mblogs = IntField()

    domain = StringField()
```

个人资料 (UserInfo) 包括基本信息，联系方式，教育信息 (edu)，工作信息 (work)，标签 (tags)，和一些动态的信息。

其中，基本信息包括：昵称 (nickname)，所在地 (location)，性别 (sex)，生日 (birth)，博客 (blog)，个人主页 (site)，简介 (intro)。

联系方式包括：email，QQ 和 MSN。教育信息 (edu) 和工作信息 (work) 都是一个 list，list 里的元素类型另外分别定义。tags 是标签，也是 list。

n_followees，n_followers，n_mblogs 是一些动态信息，分别是关注数，粉丝数和已发微博数。而 domain 的含义不明确，获取这个信息是为了爬取微博时需要。

4.2.3 Follow

```
class Follow(EmbeddedDocument):
    uid = StringField()
    nickname = StringField()
```

记录了每个粉丝或者关注的 uid 和昵称。

4.2.4 MicroBlog

MicroBlog 类用来记录一条微博，所有字段如下：

```
class MicroBlog(Document):
    mid = StringField(required=True)
    uid = StringField(required=True)
    content = StringField()
    created_time = DateTimeField()
    geo = EmbeddedDocumentField(Geo)

    is_forward = BooleanField()
    ori_mblog = EmbeddedDocumentField(oriMblog)

    n_likes = IntField()
    likes = ListField(EmbeddedDocumentField(Like))
    n_forwards = IntField()
    forwards = ListField(EmbeddedDocumentField(ForwardOrComment))
    n_comments = IntField()
    comments = ListField(EmbeddedDocumentField(ForwardOrComment))
```

其中，mid 是一条微博的唯一唯一标识；uid 是发布这条微博的微博用户的 uid；content 是微博文本内容，需要注意的是，如果这条微博是 A 转发 B 的，那么 content 记录的是 A 转发时加上的内容，而不是原微博的内容。created_time 是微博发布的时间；geo 记录了地理信息，其具体内容在下面给出。is_forward 用来标记一条微博是否是转发别人的，如果是，则为 True 并且原微博的信息将用 ori_mblog 来记录，否则为 False 并且 ori_mblog 为 None。

n_likes, n_forward 和 n_comments 分别记录该条微博被“赞”，转发和评论了多少次。likes 是点赞的人的列表，forwards 和 coments 是转发或评论的列表。

地理位置信息 Geo 类:

```
class Geo(EmbeddedDocument):
    longitue = FloatField()
    latitude = FloatField()
    location = StringField()
```

原微博信息 OriMblog 类:

```
class OriMblog(EmbeddedDocument):
    uid = StringField()
    mid = StringField()
    content = StringField()
```

其中, uid 是原微博的发布者, mid 是原微博的 mid, content 是原微博的文本内容。

4.3 Webpage Parsing

本部分介绍 Webpage Parser 的相关内容。

每一种结构不同的网页都需要用一个解析器来解析, 并把我们需要的内容从中抽取出来。页面结构相同的页面有着满足同样 pattern 的 url。目前使用 BeautifulSoup 来解析, 主要使用 html tag 和 css class 来定位页面元素。下面对于每个 Parser, 将分别介绍它对应的 url pattern 和页面元素的定位信息。

4.3.1 Profile Parser

Profile Parser 用来爬取用户信息。

4.3.1.1 URL Pattern

`http://weibo.com/{uid}/info`

其中 {uid} 表示由 10 位数字表示的用户的 uid。

4.3.1.2 页面元素定位

如上图所示, 该用户的关注数、粉丝数和微博数是 node-type 分别为 follo、fans 和 weibo 的 标签。每一个被红框都是一个 css class 为”pf_item clearfix”的 <div>。上图所示的页面中并没有包含工作信息, 实际上, 工作信息也差不多是这样的 <dic>。另外, 橙色的标签信息都是一个上述 <div> 中的 <a>。

4.3.2 Follow Parser

由于粉丝的页面和关注的页面是一样的结构, 所以它们共用一个 Parser。



图 4.1 Profile 页面

4.3.2.1 URL Pattern

- (1) 关注 `http://weibo.com/{uid}/follow?page={page}`
`{page}` 最大为 10, 10 页后的粉丝或者关注就获取不到了。
- (2) 粉丝 `http://weibo.com/{uid}/follow?relate=fans&page={page}`

4.3.2.2 页面元素定位

Follow 页面比较简单, 只要找出其中所有的粉丝/关注就可以了。每一个 Follow 都是一个 class 为 `clearfix S_line5` 的 ``。

4.3.3 MicroBlog Parser

MicroBlog Parser 用来抽取每一条微博。

4.3.3.1 URL Pattern

- (1) 普通用户 `http://weibo.com/weibo?page={page}`
- (2) media 用户 `http://weibo.com/mblog?page={page}`

- (3) ajax 加载部分我们在新浪微博上浏览一个用户的微博时，当滚动条滚到页面底部时，页面会加载更多的微博到此页上，这个动作在每一页会执行两次。这部分微博是不包含在原页面的 html 代码里的，是额外加载的，需要配置特殊的 url 来获取：

<http://weibo.com/p/aj/mblog/mbloglist?>

param

其中，

param

是一组 url 参数，具体为：

- (1) max_id: 上一段中的最后一条微博的 mid
- (2) end_id: 本页的第一条微博的 mid
- (3) page: 页号
- (4) prepage: 也是页号
- (5) count: 15
- (6) feed_type: 0
- (7) __rnd: 当前时间，可以用 `str(int(time.time()*1000))` 配置
- (8) id: domain + uid, 其中，domain 的具体含义不明，但在 profile parsing 的阶段获取到，uid 为用户的 uid
- (9) pagebar: 第一次 ajax 加载时为 0，第二次为 1
- (10) domain: domain
- (11) script_uri: `/uid/weibo`

4.3.3.2 页面元素定位

每一页中有很多条微博，每条微博的所有信息都分别包含在一个 class 为 "WB_feed_type SW_fun S_line2" 的 <div> 里。

在每一个上述的 里各个字段的定位如下：

- (1) mid <div>.mid
- (2) uid <div>.uid
- (3) content <div>.<div class="WB_text">

0

- (4) created_time <div>.

- (5) n_likes, n_forwards, n_comments <div>.<div class="WB_handle">
- (6) geo <div>.<div class="map_data">
- (7) is_forward <div>.is_forward
- (8) original mblog ori_uid: <div>. ori_mid: <div>.omid ori_content:
<div>.<div class="WB_text">

-1

5 程川负责部分

5.1 引言

5.2 关于该部分代码实现的一些说明

5.2.1 所需的 python 依赖包

- (1) rsa
- (2) lxml
- (3) yaml
- (4) mysql-python
- (5) dbutils
- (6) mongoengine

5.2.2 注释说明

注释中的特殊意义

注释	意义
#need to log	未来可能需要写入日志数据库中
#need to process	此时考虑可能不完备，未来需要改变的地方
#need to refactor	此处需要重构
#need to faster	此处可能要加速
#need to configure	此处常量可能需要写到配置文件里
#rely on third	依赖于别人的实现
#for exception handle	此处代码是为了异常和日志处理
#HTML relevant	此处代码的处理与网页的 HTML 结构相关

5.3 框架类图及说明

5.3.1 类图

类图如图 (5.1)

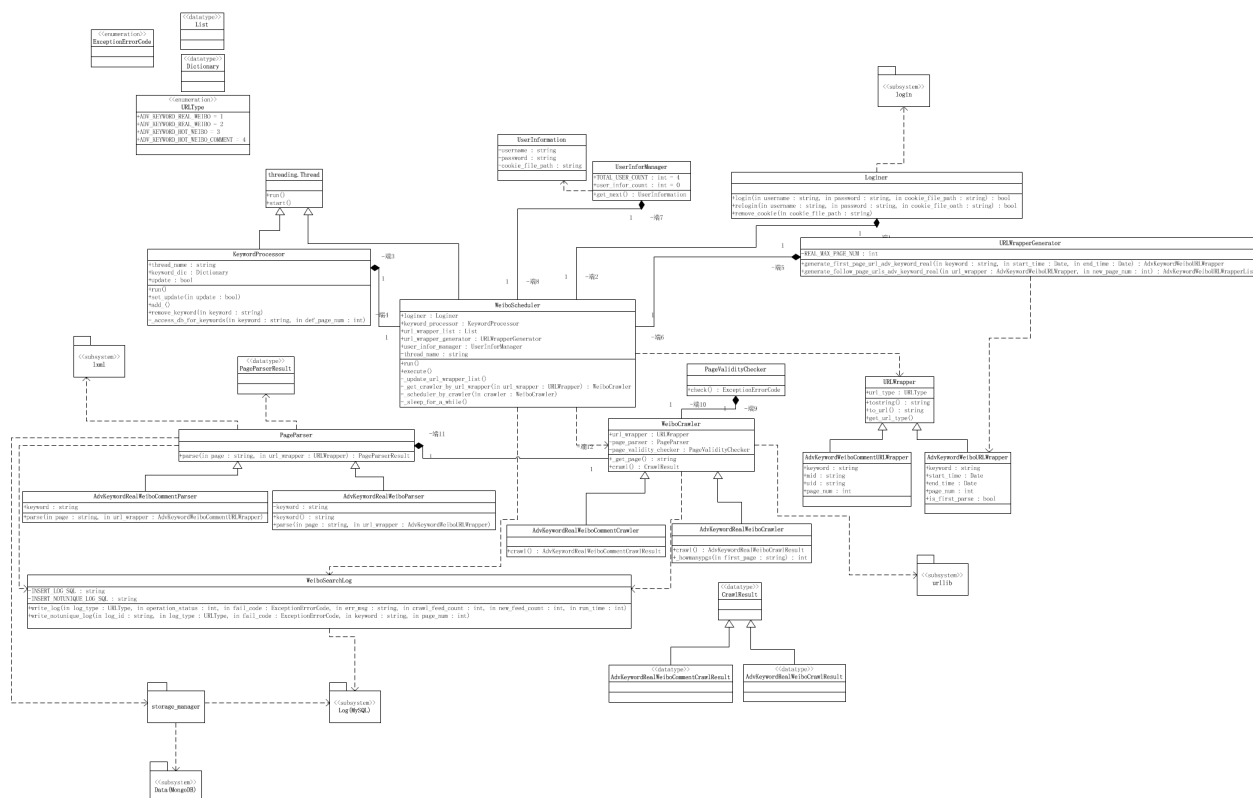


图 5.1 类关系图

5.3.2 实现说明

5.3.2.1 负责模拟登录的模块：loginer.py

该模块是对第3方登录模块 `login.py` 的封装，它主要提供一个函数 `login(username, password, cookie file)`，这个函数负责完成登录操作，需传入的参数如名字所示。

5.3.2.2 负责数据库维护的模块: storage manager.py

该模块主要负责数据存储的数据库的创建、数据表的创建、提供 `mongodb` 的连接接口等。需要注意的是，在对 `MySQL` 数据库进行访问时为了提高效率，此处使用了数据库连接池。而具体的数据表见 (5.4) 节。

5.3.2.3 负责日志写入的模块: my_log.py

该模块提供数据库日志信息的写入接口，其主要提供两个函数：

- ```
(1) write_log(log_type, operation_status, fail_code, err_msg, crawl_feed_count=0,
 new feed count=0, run time=0)
```

参数说明:

`log_type`: 这页微博所属 URL 的类型，具体有哪些类型见表 (5.8)

`operation_status`: 表示这页微博的爬取是成功 (1) 还是失败 (0)

`fail_code`: 如果这页微博爬取失败，则返回其失败原因，具体意义见 (5.3.5) 节

`err_msg`: 如果这页微博爬取失败，则记录其一些失败的说明

`crawl_feed_count`: 这页微博中保护的微博条数

`new_feed_count`: 这页微博的成功爬取并新增到数据库中的微博条数

`run_time`: 这页微博爬取所消耗的总的时间（包括休眠时间）

(2) `write_notunique_log(log_id, log_type, fail_code, keyword, page_num)`

参数说明：

`log_id`: 这条微博的 mid

`log_type`: 这页微博所属 URL 的类型，具体有哪些类型见表 (5.8)

`fail_code`: 其失败原因，具体意义见 (5.3.5) 节

`keyword`: 这条微博所属的关键词

`page_num`: 这条微博所属的微博页数

#### 5.3.2.4 主要的调度模块：weibo\_scheduler.py

该模块是执行爬虫程序的主要调度模块，该模块中的 `WeiboScheduler` 类维护一个名为 `url_wrapper_list` 的 URL 列表。其核心函数是 `execute()`，其核心操作时执行一个主循环，其中循环地每次从 URL 列表中取出一个 URL，根据其类型生成对应的 `parser` 和 `crawler`，将该 `url_wrapper` 和 `parser` 以参数传递给该 `crawler`，再执行 `crawler` 对应的 `crawl`，完成这页微博的爬取、解析和向数据库的存储操作。

需要注意的是，由于微博爬取的两次爬取之间 25 秒时间间隔的限制，所以每爬取一页微博后程序需要休眠 25 秒（平均意义下），这个休眠操作在整个程序中仅在这个循环中进行（表现为 `_sleep_for_a_while()` 函数），其它地方不涉及。具体的休眠时间间隔由参数 `GET_INTERVAL_SECONDS` 控制。

此外，由于可能涉及到 `cookie_file` 的过时问题，所以若在执行一个 `crawler` 的 `crawl()` 函数时可能会得到 `cookie_file` 过时等爬取的页面不合法的返回值，此时就再进行若干次登录和爬取尝试，具体尝试的最大次数由参数 `MAX_RELOGIN_TRIAL_NUMBER` 控制。

另外，为了达到用户能实时指定需要爬取的关键词的效果。该模块与模块 (5.3.6) 合作，采用多线程的方式（模块 (5.3.6) 为一线程，该模块为一线

程)。每当 URL 列表被循环访问一轮后，该线程去查看 KeywordProcessor 中的 update 是否被置为 true，如果被置为 true，则丢弃原来的 URL，再根据新的 URL 生成新的 URL 列表。该模块的参数：**GET\_INTERVAL\_SECONDS**, **MAX\_RELOGIN\_TRIAL\_NUMBER**

### 5.3.3 高级微博搜索的公用模块：advance\_weibo\_frame.py

改模库提供爬取任务需要实现的各个接口，分别为：

URLWrapper

PageParser

WeiboCrawler

DataStorer

#### 5.3.3.1 URLWrapper

该类维护一个 URL，其中需要存储的公共数据为 url\_type，其它数据由任务自定义。并提供两个公共函数 to\_url() 和 get\_url\_type()。

#### 5.3.3.2 PageParser

这个类需要实现的函数为 parse(page,url\_wrapper) 其中 page 为爬取到的网页，url\_wrapper 为 URLWrapper 类型的一个实例。

需要特别说明的是，这个接口很简单，但是在该模块中关于 Parser 有许多的实现，实际上这部分主要完成对实时和热门微博的爬取。而这两个任务的网页结构是一样的，故为了不重复这段代码，将这段代码维护在该模块中。这段代码中有多个 dictionary，分别是 ADV\_KEYWORD\_COMMENT\_CONSTANT\_DIC, ADV\_KEYWORD\_WEIBO\_CONSTANT\_DICT，它们分别保存着在解析微博评论和微博时各个数据字段在 HTML 树中的 DOM 路径信息。

需要注意的是，该函数要返回一些值，而具体返回的值视具体任务不同而不同，例如对微博解析返回 [crawl\_feed\_count, new\_feed\_count, weibo\_comment\_infor]，而对评论解析返回 [crawl\_feed\_count, new\_feed\_count] 这种差异需要在自己实现对应的 WeiboCrawler 时注意处理。

对于一条微博的网页结构如图 (5.2)。

单纯微博（不含评论）的获取操作说明，而评论的获取与此类似，此处不再赘述：

单纯的微博（不含评论）需获取的字段如表 (5.1):

| 字段                          | 在 HTML 中的路径                                      | 属性名                      |
|-----------------------------|--------------------------------------------------|--------------------------|
| 表征包含内部 HTML 的 script        | /descendant::script                              | html                     |
| 指示一个 script 是否包含内部 HTML 的标识 |                                                  | feed_list<br>W_linecolor |
| 抽取内部 HTML 文本的正则表达式          |                                                  | view({.*})               |
| 在内部 HTML 中的每条微博             | /descendant::dl[@class='feed_list W_linecolor '] |                          |
| 微博 ID(mid)                  | //dl[@class='feed_list W_linecolor ']            | mid                      |
| 用户 ID(uid)                  | //img[@usercontent]                              | usercontent              |
| 微博内容                        | //p[@node-type='feed_list_content']/em           |                          |
| 是否被转发                       | //dl[@class='feed_list W_linecolor ']            | isforward                |
| 原始 ID(forward_uid)          | //dt[@node-type='feed_list_forwardContent']/a    | usercontent              |
| 原始 reference                | //dt[@node-type='feed_list_forwardContent']/a    | href                     |
| 原始微博内容                      | //dt[@node-type='feed_list_forwardContent']/em   |                          |
| 创建时间 (路径相对于原始微博内容)          | //a[@class='date']                               | title                    |
| 以下统计信息的根路径                  | //p[@class='info W_linkb W_textb']               |                          |
| 微博喜欢人数 (路径相对于原始微博内容)        | //a[@action-type='feed_list_like']               | text()                   |
| 微博转发人数 (路径相对于原始微博内容)        | //a[@action-type='feed_list_forward']            | text()                   |
| 微博搜藏人数 (路径相对于原始微博内容)        | //a[@action-type='feed_list_favorite']           | text()                   |
| 微博评论数 (路径相对于原始微博内容)         | //a[@action-type='feed_list_comment']            | text()                   |

表 5.1 高级关键字微博搜索字段表

单条微博在 HTML 中的结构如图 (5.2)

单纯的微博（不含评论）的获取过程如下：

- (1) 先对外层 HTML 进行解析，获取某个 <script>...</script> 中的内层 HTML，注意此时若用 lxml 解析的话所用的 parser 要是 HTMLParser，但是在用 HTMLParser 进行解析时，会在原来的结构基础上默认加上 <html><body></body></html> 两个节点
- (2) 利用 lxml 将这部分文本转换为 DOM 树，由于这棵 DOM 树中有多条微博，故需对这个 DOM 树进行解析并将其转换成多个单条微博组成的



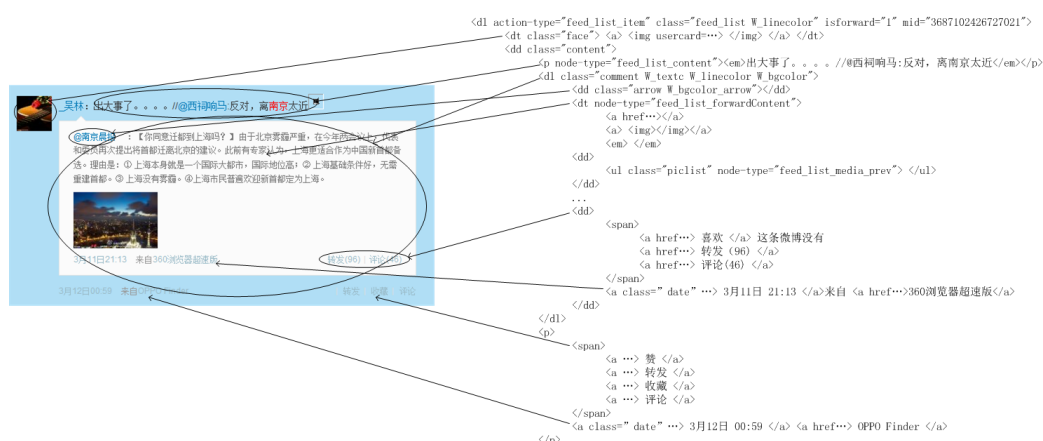


图 5.2 高级关键字搜索单条微博结构图

DOM 树列表，同样要注意在用 HTMLParser 进行解析时，会在原来的结构基础上默认加上 `<html><body></body></html>` 两个节点

- (3) 循环处理上一步产生的每棵 DOM 树：以其为参数，传入 `AdvKeyword-WeiboXMLTreeParser` 形成一个该类的示例，然后进行解析，并将解析后的结果存储到 (5.3.3.4) 节中其实现的对应的 `DataStorer` 中，然后存入数据库

需要注意的是：

- 获取微博 `content` 时，但要注意的，这个 `content` 中包含“# 为马航失联祈祷 #”这种东西
- 在解析过程中用的是 `lxml`，为了后面可能会跟着微博网页结构的变化而好改动，所有的 `<` 字段，在网页中的路径 `>` 存储在 `ADV_KEYWOR_CONSTANT_DICT` 中，需要注意的是，路径中的相对起始节点可能有所不同。
- 在解析过程中可能会由于页面获取失败或者页面结构调整导致的解析失败，此时需要将相应信息写入 `log` 数据库中，具体信息见 (5.3.5) 节

### 5.3.3.3 WeiboCrawler

该类要实现的是 `crawl(wait_time)` 函数，其中维护这该 `crawler` 对应的 `url_wrapper` 和 `page_parser`，该函数需要进行的核心操作是先请求网页，然后对网页的合法性进行判断，接着调用 `page_parser` 对网页进行解析，并存储下由 `page_parser` 可能生的新的信息（如微博爬取时产生的微博评论信息），然后整合这些信息再结合该微博是否为某个关键词的第一页以及存储的关于该关键字需要实时爬取多少页等信息，生成新的 `URLWrapper` 列表，并将这列表返回给 `WeiboScheduler`。

需要注意的是，在这个函数中，对一页进行解析后，不管成功与否都要将其写入 log 数据库。如果页面不合法，还要向 WeiboScheduler 返回，从而帮助其判断针对该 URL 是否需要重新爬取。所以一般来讲 crawl 需要返回的信息是 [page\_is\_validity, new\_url\_list]，而传入的参数仅仅是为了写日志（日志中需要记录这个 URL 的处理的总的时间，包括休眠时间，而休眠操作发生在 WeiboScheduler 中，故需要以参数的形式传入）。

#### 5.3.3.4 DataStorer

该模块维护着与数据库中的表对应的类，仅仅是为了 mongoengine 使用的需要和数据的存储，此处不详细说明。

#### 5.3.4 与具体任务相关的模块:advance\_keyword\_hot\_weibo.py、advance\_keyword\_real\_weibo.py

这两个模块均为 advance\_weibo\_frame.py 中定义的接口的实现，参见 (5.3.3) 节。

#### 5.3.5 异常类模块: my\_exceptions.py

该模块维护四个在微博的爬取、写入数据库过程中会出现的异常和其对应的异常代码，这四个异常类分别为：AdvKeywordWeiboPageParseException、AdvKeywordPageGetException、AdvKeywordWeiboCommentPageParseException 和 OtherException，异常说明和其对应的代码如表 (5.2):

#### 5.3.6 用户指定的关键字维护模块: keyword\_processor.py

为了维护用户定义的关键字再考虑到与用 JAVA 实现的前端的交互，关键字列表被维护在一个 mongoDB 的数据表 (5.9) 中。为了能“实时”地将关键字的改变反映出来，在实现时采用多线程的方式，本模块的 KeywordProcessor 类实际上要维护一个线程，这个线程定时 (时间间隔为 KEYWORD\_DB\_SCAN\_INTERVAL) 地去访问数据表 (5.9)，如果发现关键字列表有改变，则获取新的列表，并将其中的列表更新标识 update 置为 true。

需要注意的是，该模块与 (5.3.2.4) 模块配合工作，具体配合方式见 (5.3.2.4)。

#### 5.3.7 任务无关模块（即对所有任务来说都一样的模块，不涉及类型的差异）

(1) main\_entry.py

#### 5.3.8 添加任务示例

下面以添加高级关键字热点搜索为例，说明整个添加过程：

| 异常类名: AdvKeywordWeiboPageParseException        |    |
|------------------------------------------------|----|
| 异常说明                                           | 代码 |
| "total tree construct"                         | 1  |
| "script find failed"                           | 2  |
| "content inside script find failed"            | 3  |
| "weibo trees construct"                        | 4  |
| "get mid element failed"                       | 5  |
| "get mid attribute failed"                     | 6  |
| "get uid element failed"                       | 7  |
| "get uid attribute failed"                     | 8  |
| "get content element failed"                   | 9  |
| "get is forward element failed"                | 10 |
| "get forward uid element failed"               | 11 |
| "get forward uid attribute failed"             | 12 |
| "get howmannypgs failed"                       | 13 |
| 异常类名: AdvKeywordPageGetException               |    |
| 异常说明                                           | 代码 |
| "urllib2 open failed"                          | 14 |
| "verification error"                           | 15 |
| "cookie outofdate error"                       | 16 |
| "pass certificate error"                       | 17 |
| "page not exist"                               | 18 |
| 异常类名: AdvKeywordWeiboCommentPageParseException |    |
| 异常说明                                           | 代码 |
| "total tree construct"                         | 19 |
| "script find failed"                           | 20 |
| "content inside script find failed"            | 21 |
| "comment trees construct"                      | 22 |
| "get cid element failed"                       | 23 |
| "get cid attribute failed"                     | 24 |
| "get attached mid element failed"              | 25 |
| "get attached mid attribute failed"            | 26 |
| "get content element failed"                   | 27 |
| "extract cid failed"                           | 28 |
| "get forward uid element failed"               | 29 |
| "get forward uid attribute failed"             | 30 |
| "get howmannypgs failed"                       | 31 |
| 异常类名: OtherException                           |    |
| 异常说明                                           | 代码 |
| "weibo store NotUniqueError"                   | 32 |
| "weibo store ValidationError"                  | 33 |
| "weibo store OperationError"                   | 34 |
| "weibo store OtherError"                       | 35 |
| "weibo comment store NotUniqueError"           | 36 |
| "weibo comment store ValidationError"          | 37 |
| "weibo comment store OperationError"           | 38 |
| "weibo comment store OtherError"               | 39 |

表 5.2 异常类和其中的异常与其代码的对应关系表

- (1) 新建 `advance_keyword_hot_weibo.py` 模块
- (2) 在该模块中实现对应的 `URLWrapper`
- (3) 在该模块中实现对应的 `PageParser`，这儿的 `parser` 实际上与实时搜索的一样，故此处封装了一下，调用实时的 `parse`
- (4) 在该模块中实现对应的 `WeiboCrawler`
- (5) 在该模块中实现对应的数据存储类和定义出与 `mongodb` 中对应给定数据表

## 5.4 存储设计

### 5.4.1 微博

如表 (5.3):

| 存储方式: MongoDB<br>表名: <code>single_weibo</code> |                     |    |     |                   |
|------------------------------------------------|---------------------|----|-----|-------------------|
| 字段名                                            | 类型                  | 主键 | 可为空 | 说明                |
| <code>mid</code>                               | <code>string</code> | Y  | N   | 微博 ID             |
| <code>uid</code>                               | <code>string</code> | N  | N   | 微博所属用户 ID         |
| <code>keyword</code>                           | <code>string</code> | N  | Y   | 本条微博所属的关键词        |
| <code>typeid</code>                            | <code>int</code>    | N  | N   | 微博的类型 ID          |
| <code>content</code>                           | <code>string</code> | N  | Y   | 本条微博的内容           |
| <code>is_forward</code>                        | <code>bool</code>   | N  | N   | 该条微博是否转发其它微博      |
| <code>forward_uid</code>                       | <code>string</code> | N  | Y   | 被这条微博所转发的微博的用户 ID |
| <code>original_ref</code>                      | <code>string</code> | N  | Y   | 微博所在的用户的主页        |
| <code>original_cntnt</code>                    | <code>string</code> | N  | N   | 被转发的微博的原始内容       |
| <code>n_like</code>                            | <code>int</code>    | N  | N   | 该微博点赞人数           |
| <code>n_forward</code>                         | <code>int</code>    | N  | N   | 该微博转发人数           |
| <code>n_favorite</code>                        | <code>int</code>    | N  | N   | 该微博收藏人数           |
| <code>n_comment</code>                         | <code>int</code>    | N  | N   | 该微博评论条数           |
| <code>create_time</code>                       | <code>date</code>   | N  | N   | 该微博的发布时间          |
| <code>url</code>                               | <code>string</code> | N  | N   | 该微博所在的 URL        |

表 5.3 微博相关数据字段表

| 类型 ID | 说明            |
|-------|---------------|
| 1     | 根据关键字搜索到的实时微博 |

表 5.4 微博数据库中微博类型及其说明

### 5.4.2 评论

如表 (5.5):

| 存储方式: MongoDB |        |    |     |             |
|---------------|--------|----|-----|-------------|
| 表名: comment   |        |    |     |             |
| 字段名           | 类型     | 主键 | 可为空 | 说明          |
| cid           | string | Y  | N   | 该评论的 ID     |
| attatched_mid | string | N  | N   | 该评论所属微博的 ID |
| content       | string | N  | N   | 该评论的内容      |
| create_time   | string | N  | N   | 该评论发布的时间    |

表 5.5 微博评论相关数据字段表

### 5.4.3 日志

如表 (5.6):

| 存储方式: MySQL             |          |    |     |                           |
|-------------------------|----------|----|-----|---------------------------|
| 表名: sinaweibosearch_log |          |    |     |                           |
| 字段名                     | 类型       | 主键 | 可为空 | 说明                        |
| log_id                  | long     | Y  | N   | 自动生成, 自增                  |
| type                    | int      | N  | N   | 当前处理的微博的类型                |
| status                  | int      | N  | N   | 当前日志表示的操作时成功 (1) 还是失败 (0) |
| crawl_feed_count        | int      | N  | N   | 抓取的微博数量                   |
| new_feed_count          | int      | N  | N   | 新增的微博数量                   |
| fail_code               | int      | N  | Y   | 如果失败, 其失败错误代码是什么          |
| err_msg                 | string   | N  | Y   | 如果失败, 其失败信息是什么            |
| write_time              | datetime | N  | N   | 本条日志的添加时间                 |
| run_time                | int      | N  | N   | 运行时间 (毫秒)                 |

表 5.6 微博评论相关数据字段表

如表 (5.7):

| 存储方式: MySQL                       |          |    |     |                  |
|-----------------------------------|----------|----|-----|------------------|
| 表名: sinaweibosearch_notunique_log |          |    |     |                  |
| 字段名                               | 类型       | 主键 | 可为空 | 说明               |
| id                                | string   | Y  | N   | 微博/评论 ID         |
| type                              | int      | N  | N   | 当前处理的微博类型        |
| keyword                           | string   | N  | N   | 相关的关键字           |
| page_num                          | int      | N  | N   | 微博/评论所在的页码       |
| fail_code                         | int      | N  | Y   | 如果失败, 其失败错误代码是什么 |
| write_time                        | datetime | N  | N   | 本条日志的添加时间        |

表 5.7 数据库重复 ID 记录日志字段表

### 5.4.4 关键词

如表 (5.9):

| 类型 ID | 说明              |
|-------|-----------------|
| 1     | 根据关键字搜索到的实时微博   |
| 2     | 根据关键字搜索到的实时微博评论 |
| 3     | 根据关键字搜索到的热点微博   |
| 4     | 根据关键字搜索到的热点微博评论 |

表 5.8 日志中微博类型及其说明

| 存储方式: MongoDB              |        |    |     |               |
|----------------------------|--------|----|-----|---------------|
| 表名: sinaweibo_keyword_real |        |    |     |               |
| 字段名                        | 类型     | 主键 | 可为空 | 说明            |
| keyword                    | string | N  | N   | 关键词           |
| def_page_num               | int    | N  | N   | 该关键词默认的实时爬取页数 |

表 5.9 微博评论相关数据字段表

## 6 论坛的实时爬取

作者：黄家君

### 6.1 功能描述

本部分工程名为 `realtime_bbs`，主要功能是实时爬取用户指定的论坛版面。目前已经实现的论坛有天涯论坛 ([bbs.tianya.cn](http://bbs.tianya.cn))，百度贴吧 ([tieba.baidu.com](http://tieba.baidu.com))，财富网股吧 (<http://guba.eastmoney.com/>)。如果需要爬其他的论坛，则需要编写相应的 Spider，这一点会在后文详述。

本子系统使用 scrapy 爬虫框架，scrapy 的文档参见 <http://doc.scrapy.org/en/latest/>。

### 6.2 存储设计

存储设计体现在 `items.py`。

```
class PostItem(Item):
 url = Field()
 title = Field()
 content = Field()
 time = Field()
 author = Field()
 n_click = Field() #int
 n_reply = Field() #int
 comments = Field() #list
 tags = Field() #list
```

一个 `PostItem` 对应着一个帖子页面，`url` 是这个帖子的 `url`，`str` 类型。`title` 为标题，`content` 是帖子的文本内容，`time` 为发贴时间，`author` 是发贴者，此 4 个字段都是 `unicode` 类型。`n_click` 和 `n_reply` 是点击量和回复量，`int` 类型。`comments` 是评论，是一个列表，每一条评论都是一段文本，也是 `unicode`。`tags` 是额外加上的标签，可以理解成敏感词或者关键词，比如用户希望监控和”警察“相关的帖子，那么爬到的帖子里如果标题、内容或者评论里如果出现了”警察“这个词，那么这个 `Item` 就会打上”警察“这个标签。

数据库使用的是 MongoDB，读取的时候注意要检查字段是否为空。

## Spiders

每个论坛需要有一个 Spider 来解析该论坛的页面。天涯论坛相对应的 Spider 类为 TianyaSpider，百度贴吧是 TiebaSpider，股吧是 EastmoneySpider。

Spider 类的结构大致都是一样的：

```
Spider:
 -name
 -allowed_domains
 -url_generator
 +start_requests()
 +parse_board(response)
 +parse_item(response)
```

其中，name 是用来标识一个 Spider 类的名字，str 类型。allowed\_domains 是相应论坛的域名，比如财富网股吧是“eastmoney.com”。url\_generator 负责从 MongoDB 中循环取出版块的 url，需要指定数据库的主机地址，端口号，数据库名和集合名，详见 url\_generator.URLGenerator。start\_requests 生成初始的 HttpRequest，基本做法是从 url\_generator 取第一个板块的 url，并指定 parse\_board 为回调函数 (callback)。parse\_board 是负责从版块中抽出版中帖子的 url，指定 parse\_item 作为 callback 生成一个 Request 的列表；然后，要从 url\_generator 取下一个板块的 url，加到 requests 列表中，最后返回 requests。

parse\_item 是各个 Spider 主要差异之处，它的功能是从 html 页面中抽取出 Item 来，因为不同网站的页面结构不一样，所以每个网站的 Spider 的 parse\_item 不一样。

## 6.3 pipelines

Spider 返回的 Item 会经过 pipelines，pipelines 可以是任何有 process\_item 方法的类，每个 pipeline（或者说 pipelines 的每一节）都可以对 Item 进行一些处理。激活 pipeline 需要在 settings 中设置，设置的方法和细节请参见 scrapy 的官方文档。

目前，我写了两个 pipeline，分别是 TaggingPipeline 和 RealtimeBbsPipeline，spiders 返回的 Item 依次流过这两个 pipeline。TaggingPipeline 负责给 Item 打上标签，如上一节（存储设计）中所述。而 RealtimeBbsPipeline 负责把 Item 存入



MongoDB，如果数据库中已经存在相同 url 的 item，则更新之（同一个帖子可能会被重复抓取）。

## 6.4 Downloader Middleware

Downloader Middleware 是连接 scrapy 引擎和 Downloader（下载器）的钩子。spiders 提交的 Request 首先会交给 Scheduler（调度者），然后 Scheduler 每次选出一个给 Downloader，这时 Request 会经过一些 Downloader Middleware，这些 Downloader Middleware 可以对经过的 Request 做一些处理。Downloader 向 web 请求得到的 Response 在交给 spider 之前，也会经过 Downloader Middleware，同样 Downloader Middleware 也可以对 Response 做一些处理。

目前总共有 3 个 Downloader Middleware。下面分别介绍。

### 6.4.1 RandomUserAgentMiddleware

RandomUserAgentMiddleware 的作用是给每个 Request 带上一个随机的 User Agent 信息，这样做的目的是在尽量避免爬虫被封禁。User Agent 列表在 settings.py 中。

### 6.4.2 The404Middleware

The404Middleware 的功能是过滤掉 404 页，因为 404 页不需要交给 spider 去解析。过滤掉 404 页面时需要写系统日志。

### 6.4.3 MyRedirectedDownloaderMiddleware

MyRedirectedDownloaderMiddleware 本意是想自定义地处理重定向，目前没有被激活，所以还是使用 scrapy 默认的重定向处理中间件。

## 6.5 启动多个 Spider

使用 scrapy 命令只能一次启动一个 spider，因此需要写一个脚本来同时启动多个爬虫，这就是 run\_multiple\_spiders.py，将来如果需要启动更多 spider，注意请在这个脚本中添加。

## 6.6 杂项: settings

关于 settings 的详细，请参见 scrapy 官方文档。此处点出一些值得注意的设置。

- (1) 设置宽度优先 scrapy 默认是深度优先策略，宽度优先需要特别设置，参见 settings.py 的 set to BFO 部分。
- (2) 禁用 scrapy 的去重避免重复请求同一个 url 对于一般的爬虫来说是必要的，但是我们需要对指定的版块重复爬取，所以要禁用 scrapy 默认提供的去重机制，方法是设置 DUPEFILTER\_CLASS = 'scrapy.dupefilter.BaseDupeFilter'。
- (3) Downloader 延时和 cookie 为了避免爬虫被封禁，我们应该避免过于频繁地向 web 服务器发出请求，这需要设置 DOWNLOAD\_DELAY。同样，禁用 COOKIE 也是为了防止被封，可以通过 COOKIE\_ENABLED=False。

## 致 谢

感谢各位同学的辛勤劳动

