23. 休养生息-常用模块-03-re

本节主要内容

1. 正则表达式
2. re模块的使用
3. 一堆练习

一. 正则表达式

首先, 我们在网页上进行注册或者登陆的时候经常能看到一些格式上的错误提示. 比如: 你在注册百度账号的时候https://passport.baidu.com/v2/?reg&regType=1&tpl=wk 输入用户名随意的输入系统会提示你. 你的账号过长或者不允许使用中文等等操作. 那这种操作如果使用我们现有的知识点是可以完成的. 但是完成的效果并不好. 写起来也不容易. 尤其是对邮箱的匹配. 电话号码的匹配. 那正则表达式就是专门来处理类似问题的一种表达式. 英文全称: Regular Expression. 简称 regex或者re. 但你要知道我们在使用python的re模块之前. 我们首先要对正则有一定的了解和认识. 就像我们使用time模块之前. 我们已经对时间有了一定的认识.

正则表达式是对字符串操作的一种逻辑公式. 我们一般使用正则表达式对字符串进行匹配和过滤. 使用正则的优缺点:

优点: 灵活, 功能性强, 逻辑性强.

缺点: 上手难. 一旦上手, 会爱上这个东西

工具: 各大文本编辑器一般都有正则匹配功能. 我们也可以去 *http://tool.chinaz.com/regex/*进行在线测试.

正则表达式由普通字符和元字符组成. 普通字符包含大小写字母, 数字. 在匹配普通字符的时候我们直接写就可以了. 比如"abc" 匹配的就是"abc". 我们如果用python也可以实现相同的效果. 所以普通字符没什么好说的. 重点在元字符上.

元字符: 元字符才是正则表达式的灵魂. 元字符中的内容太多了, 在这里我们只介绍一些常用的.

1. 字符组

   字符组很简单用[]括起来. 在[]中出现的内容会被匹配. 例如:[abc] 匹配a或b或c

   如果字符组中的内容过多还可以使用- , 例如: [a-z] 匹配a到z之间的所有字母 [0-9] 匹配所有阿拉伯数字

   思考: [a-zA-Z0-9]匹配的是什么?

2. 简单元字符

   基本的元字符. 这个东西网上一搜一大堆. 但是常用的就那么几个:

```
.       匹配除换行符以外的任意字符
\w      匹配字母或数字或下划线
\s      匹配任意的空白符
```

```
\d      匹配数字
\n      匹配一个换行符
\t      匹配一个制表符
\b      匹配一个单词的结尾
^       匹配字符串的开始
$       匹配字符串的结尾
\W      匹配非字母或数字或下划线
\D      匹配非数字
\S      匹配非空白符
a|b     匹配字符a或字符b
()      匹配括号内的表达式，也表示一个组
[...]       匹配字符组中的字符
[^...]      匹配除了字符组中字符的所有字符
```

3. 量词

我们到目前匹配的所有内容都是单一文字符号. 那如何一次性匹配很多个字符呢, 我们要用到量词

```
*       重复零次或更多次
+       重复一次或更多次
?       重复零次或一次
{n}     重复n次
{n,}      重复n次或更多次
{n,m}      重复n到m次
```

4. 惰性匹配和贪婪匹配

在量词中的*, +,{} 都属于贪婪匹配. 就是尽可能多的匹配到结果.

```
str: 麻花藤昨天让英雄联盟关服了
reg: 麻花藤.*

此时匹配的是整句话
```

在使用.*后面如果加了? 则是尽可能的少匹配. 表示惰性匹配

```
str: 麻花藤昨天让英雄联盟关服了
reg: 麻花藤.*?

此时匹配的是 麻花藤

str: <div>胡辣汤</div>
reg: <.*>
结果: <div>胡辣汤</div>


str: <div>胡辣汤</div>
reg: <.*?>
结果:
```

```
    <div>
    </div>

str: <div>胡辣汤</div>
reg: <(div|/div*)?>
结果:
    <div>
    </div>
```

.*?x的特殊含义 找到下一个x为止.

```
str: abcdefgxhijklmn
reg: .*?x
结果:abcdefgx
```

5. 分组

在正则中使用()进行分组. 比如. 我们要匹配一个相对复杂的身份证号. 身份证号分成两种. 老的身份证号有15位. 新的身份证号有18位. 并且新的身份证号结尾有可能是x.

```
给出以下正则:
^[1-9]\d{13,16}[0-9x]$
^[1-9]\d{14}(\d{2}[0-9x])?$
^([1-9]\d{16}[0-9x]|[1-9]\d{14})$
```

6. 转义

在正则表达式中, 有很多有特殊意义的是元字符, 比如\n和\s等,如果要在正则中匹配正常的"\n"而不是"换行符"就需要对"\"进行转义, 变成'\\'.在python中, 无论是正则表达式, 还是待匹配的内容, 都是以字符串的形式出现的, 在字符串中\也有特殊的含义, 本身还需要转义. 所以如果匹配一次"\n", 字符串中要写成'\\n', 那么正则里就要写成"\\\\n",这样就太麻烦了. 这个时候我们就用到了r'\n'这个概念, 此时的正则是r'\\n'就可以了.

练习:

1. 匹配邮箱
2. 匹配手机号
3. 匹配生日. 日期格式(yyyy-MM-dd)
4. 匹配电话号码
5. 匹配IP

二. re模块

re模块是python提供的一套关于处理正则表达式的模块. 核心功能有四个:

1. findall 查找所有. 返回list

```
lst = re.findall("m", "mai le fo len, mai ni mei!")
print(lst)     # ['m', 'm', 'm']

lst = re.findall(r"\d+", "5点之前. 你要给我5000万")
```

```
print(lst)     # ['5', '5000']
```

2. search 会进行匹配. 但是如果匹配到了第一个结果. 就会返回这个结果. 如果匹配不上search返回的则是None

```
ret = re.search(r'\d', '5点之前. 你要给我5000万').group()
print(ret) # 5
```

3. match 只能从字符串的开头进行匹配

```
ret = re.match('a', 'abc').group()
print(ret)     # a
```

4. finditer 和findall差不多. 只不过这时返回的是迭代器

```
it = re.finditer("m", "mai le fo len, mai ni mei!")

for el in it:
    print(el.group()) # 依然需要分组
```

5. 其他操作

```
ret = re.split('[ab]', 'qwerafjbcd')  # 先按'a'分割得到'qwer'和'fjbcd',在对'qwer'和'fjbcd'分别按'b'分割
print(ret)  # ['qwer', 'fj', 'cd']

ret = re.sub(r"\d+", "_sb_", "alex250taibai250wusir250ritian38") # 把字符串中的数字换成__sb__
print(ret)     # alex_sb_taibai_sb_wusir_sb_ritian_sb_


ret = re.subn(r"\d+", "_sb_", "alex250taibai250wusir250ritian38") # 将数字替换成'__sb__', 返回元组(替换的结果,替换了多少次)
print(ret)     # ('alex_sb_taibai_sb_wusir_sb_ritian_sb_', 4)

obj = re.compile(r'\d{3}')  # 将正则表达式编译成为一个 正则表达式对象, 规则要匹配的是3个数字
ret = obj.search('abc123eeee') # 正则表达式对象调用search, 参数为待匹配的字符串
print(ret.group())  # 结果: 123

爬虫重点:
obj = re.compile(r'(?P<id>\d+)(?P<name>e+)')  # 从正则表达式匹配的内容每个组起名字
ret = obj.search('abc123eeee') # 搜索
print(ret.group())  # 结果: 123eeee
print(ret.group("id"))  # 结果: 123 # 获取id组的内容
print(ret.group("name")) # 结果: eeee # 获取name组的内容
```

6. 两个坑

注意: 在re模块中和我们在线测试工具中的结果可能是不一样的.

```
ret = re.findall('www.(baidu|oldboy).com', 'www.oldboy.com')
print(ret)  # ['oldboy']     这是因为findall会优先把匹配结果组里内容返回,如果想要匹配结果,取消权限即可

ret = re.findall('www.(?:baidu|oldboy).com', 'www.oldboy.com')
print(ret)  # ['www.oldboy.com']
```

split里也有一个坑

```
ret=re.split("\d+","eva3egon4yuan")
print(ret) #结果 ： ['eva', 'egon', 'yuan']

ret=re.split("(\d+)","eva3egon4yuan")
print(ret) #结果 ： ['eva', '3', 'egon', '4', 'yuan']

#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。
```

这种优先级的问题有时候会帮我们完成很多功能. 我们来看一个比较复杂的例子

```
import re
from urllib.request import urlopen
import ssl
# 干掉数字签名证书
ssl._create_default_https_context = ssl._create_unverified_context

def getPage(url):
    response = urlopen(url)
    return response.read().decode('utf-8')

def parsePage(s):
    ret = re.findall(
        '<div class="item">.*?<div class="pic">.*?<em .*?>(?P<id>\d+).*?<span class="title">(?P<title>.*?)</span>'
        '.*?<span class="rating_num" .*?>(?P<rating_num>.*?)</span>.*?<span>(?P<comment_num>.*?)评价</span>', s, re.S)
    return ret

def main(num):
    url = 'https://movie.douban.com/top250?start=%s&filter=' % num
    response_html = getPage(url)
    ret = parsePage(response_html)
    print(ret)

count = 0
for i in range(10):   # 10页
```

```
    main(count)
    count += 25
```

此时利用的就是分组之后. 匹配成功后获取到的是分组后的结果. (?P<id>\d+) 此时当前组所匹配的数据就会被分组到id组内. 此时程序可以改写成:

```
import ssl
import re
from urllib.request import urlopen

# 干掉数字签名证书
ssl._create_default_https_context = ssl._create_unverified_context


def getPage(url):
    response = urlopen(url)
    return response.read().decode('utf-8')

def parsePage(s):
    com = re.compile(
        '<div class="item">.*?<div class="pic">.*?<em .*?>(?P<id>\d+).*?
<span class="title">(?P<title>.*?)</span>'
        '.*?<span class="rating_num" .*?>(?P<rating_num>.*?)</span>.*?
<span>(?P<comment_num>.*?)评价</span>', re.S)

    ret = com.finditer(s)
    for i in ret:
        yield {
            "id": i.group("id"),
            "title": i.group("title"),
            "rating_num": i.group("rating_num"),
            "comment_num": i.group("comment_num"),
        }


def main(num):
    url = 'https://movie.douban.com/top250?start=%s&filter=' % num
    response_html = getPage(url)
    ret = parsePage(response_html)
    # print(ret)
    f = open("move_info7", "a", encoding="utf8")

    for obj in ret:
        print(obj)
        data = str(obj)
        f.write(data + "\n")

count = 0
for i in range(10):
    main(count)
```

```
    count += 25
```

正则表达式和re模块就说这么多. 如果要把正则所有的内容全部讲清楚讲明白, 至少要一周以上的时间. 对于我们日常使用而言. 上述知识点已经够用了. 如果碰到一些极端情况建议想办法分部处理. 先对字符串进行拆分. 然后再考虑用正则.