# 正则化表达

python中re模块提供了正则表达式的功能，常用的有四个方法(match、search、findall)都可以用于匹配字符串。

Match

匹配字符串：

**re.match()必须从字符串开头匹配！**match方法尝试从字符串的起始位置匹配一个模式，如果不是起始位置匹配成功的话，match()就返回none。主要参数如下：

re.match(pattern, string)

# pattern 匹配的正则表达式

# string 要匹配的字符串

例子：

In[1]: import re

a = re.match('test','testasdtest')

print(a) #返回一个匹配对象

print(a.group()) #返回test，获取不到则报错

print(a.span()) #返回匹配结果的位置，左闭右开区间

print(re.match('test','atestasdtest')) #返回None

<\_sre.SRE\_Match object; span=(0, 4), match='test'>

test

(0, 4)

None

从例子中我们可以看出，re.match()方法返回一个匹配的对象，而不是匹配的内容。如果需要返回内容则需要调用group()。通过调用span()可以获得匹配结果的位置。而如果从起始位置开始没有匹配成功，即便其他部分包含需要匹配的内容，re.match()也会返回None。

单字符匹配

以下字符，都匹配单个字符数据。且开头（从字符串0位置开始）没匹配到，即使字符串其他部分包含需要匹配的内容，.match也会返回none

. 匹配任意一个字符

使用几个点号就代表几个字符

In[2]: import re

a = re.match('..','testasdtest')

print(a.group()) #输出te

b = re.match('ab.','testasdtest')

print(b) #返回none，因为表达式是以固定的ab开头然后跟上通配符. 所以必须要先匹配上ab才会往后进行匹配

te

None

\d 匹配数字

一个\d代表一个数字。开头没匹配到，即使字符串其他部分包含需要匹配的内容，.match也会返回none

In[3]: import re

a = re.match('\d\d','23es12testasdtest')

print(a)

b = re.match('\d\d\d','23es12testasdtest')

print(b) #要求匹配三个数字，匹配不到返回none

c = re.match('\d','es12testasdtest')

print(c) #起始位置没有匹配成功，一样返回none

<\_sre.SRE\_Match object; span=(0, 2), match='23'>

None

None

\D 匹配非数字

开头没匹配到，即使字符串其他部分包含需要匹配的内容，.match也会返回none

In[4]: import re

a = re.match('\D','23es12testasdtest')

print(a) #开头为数字所以返回none

b = re.match('\D\D','\*es12testasdtest')

print(b) #返回\*e

None

<\_sre.SRE\_Match object; span=(0, 2), match='\*e'>

\s 匹配特殊字符，如空白，空格，tab等

In[5]: import re

print(re.match('\s',' 23es 12testasdtest')) #匹配空格

print(re.match('\s',' 23es 12testasdtest')) #匹配tab

print(re.match('\s','\r23es 12testasdtest')) #匹配\r换行

print(re.match('\s','23es 12testasdtest')) #返回none

<\_sre.SRE\_Match object; span=(0, 1), match=' '>

<\_sre.SRE\_Match object; span=(0, 1), match=' '>

<\_sre.SRE\_Match object; span=(0, 1), match='\r'>

None

\S 匹配非空白

In[6]: import re

print(re.match('\S',' 23es 12testasdtest')) #返回none

print(re.match('\S','\r23es 12testasdtest')) #none

print(re.match('\S','23es 12testasdtest'))

None

None

<\_sre.SRE\_Match object; span=(0, 1), match='2'>

\w 匹配单词、字符，如大小写字母，数字，\_ 下划线

In[7]: import re

print(re.match('\w','23es 12testasdtest')) #返回none

print(re.match('\w\w\w','aA\_3es 12testasdtest')) #返回none

print(re.match('\w\w\w','\n12testasdtest')) #返回none

<\_sre.SRE\_Match object; span=(0, 1), match='2'>

<\_sre.SRE\_Match object; span=(0, 3), match='aA\_'>

None

\W 匹配非单词字符

In[8]: import re

print(re.match('\W','23es 12testasdtest')) #返回none

print(re.match('\W',' 23es 12testasdtest')) #匹配空格

None

<\_sre.SRE\_Match object; span=(0, 1), match=' '>

[ ] 匹配[ ]中列举的字符

只允许出现[ ]中列举的字符

In[9]: import re

print(re.match('12[234]','232s12testasdtest')) #因为开头的12没匹配上，所以直接返回none

print(re.match('12[234]','1232s12testasdtest')) #返回123

None

<\_sre.SRE\_Match object; span=(0, 3), match='123'>

[^2345] 不匹配2345中的任意一个

In[10]: import re

print(re.match('12[^234]','232s12testasdtest')) #因为开头的12没匹配上，所以直接返回none

print(re.match('12[^234]','1232s12testasdtest')) #返回none

print(re.match('12[^234]','1252s12testasdtest')) #返回125

None

None

<\_sre.SRE\_Match object; span=(0, 3), match='125'>

[a-z3-5] 匹配a-z或者3-5中的字符

In[11]: import re

print(re.match('12[1-3a-c]','1232b12testasdtest')) #123

print(re.match('12[1-3a-c]','12b2b12testasdtest')) #12b

print(re.match('12[1-3a-c]','12s2b12testasdtest')) #返回none

<\_sre.SRE\_Match object; span=(0, 3), match='123'>

<\_sre.SRE\_Match object; span=(0, 3), match='12b'>

None

表示数量

像上面写的那些都是匹配单个字符，如果我们要匹配多个字符的话，只能重复写匹配符。这样显然是不人性化的，所以我们还需要学习表达数量的字符

\* 出现0次或无数次

In[12]: import re

a = re.match('..','testasdtest')

print(a.group()) #输出te

a = re.match('.\*','testasdtest')

print(a.group()) #全部输出

te

testasdtest

In[13]: import re

print(re.match('a\*','aatestasdtest')) #匹配跟随在字母a后面的所有a字符

print(re.match('\d\*','23aatestasdtest')) #匹配前面为数字的字符

print(re.match('a\d\*','ad23aatestasdtest')) #输出a, 因为\*也可以代表0次

<\_sre.SRE\_Match object; span=(0, 2), match='aa'>

<\_sre.SRE\_Match object; span=(0, 2), match='23'>

<\_sre.SRE\_Match object; span=(0, 1), match='a'>

+ 至少出现一次

In[14]: import re

print(re.match('a+','aaatestasdtest')) #匹配前面为字母a的字符，且a至少有1一个

print(re.match('a+','atestasdtest')) #a

print(re.match('a+','caaatestasdtest')) #none

<\_sre.SRE\_Match object; span=(0, 3), match='aaa'>

<\_sre.SRE\_Match object; span=(0, 1), match='a'>

None

? 1次或则0次

In[15]: import re

print(re.match('a?','abatestasdtest')) #匹配a出现0次或者1次数

print(re.match('a?','batestasdtest')) #输出空，因为a可以为0次

print(re.match('a?','aaatestasdtest')) #a出现0次或者1次，输出1个a

<\_sre.SRE\_Match object; span=(0, 1), match='a'>

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

<\_sre.SRE\_Match object; span=(0, 1), match='a'>

{m}指定出现m次

In[16]: import re

print(re.match('to{3}','toooooabatestasdtest')) #匹配t以及跟随在后面的三个ooo

print(re.match('to{3}','tooabatestasdtest')) #只有两个0,返回none

<\_sre.SRE\_Match object; span=(0, 6), match='tooooo'>

None

{m,n} 指定从m-n次的范围

In[18]: import re

print(re.match('to{3,4}','toooabatestasdtest')) #刚好有三个ooo，成功匹配

print(re.match('to{3,4}','tooabatestasdtest')) #只有两个o,返回none

print(re.match('to{3,4}','toooooabatestasdtest')) #提取最多四个o

<\_sre.SRE\_Match object; span=(0, 4), match='tooo'>

None

<\_sre.SRE\_Match object; span=(0, 5), match='toooo'>

匹配边界

$ 匹配结尾字符

定义整个字符串必须以指定字符串结尾

In[19]: import re

print(re.match('.\*d$','2testaabcd')) #字符串必须以d结尾

print(re.match('.\*c','2testaabcd')) #字符串不是以c结尾，返回none

<\_sre.SRE\_Match object; span=(0, 10), match='2testaabcd'>

<\_sre.SRE\_Match object; span=(0, 9), match='2testaabc'>

^ 匹配开头字符

定义整个字符串必须以指定字符开头

In[20]: import re

print(re.match('^2','2stoooabatestas')) #规定必须以2开头，否则none

print(re.match('^2s','2stoooabatestas')) #必须以2s开头

<\_sre.SRE\_Match object; span=(0, 1), match='2'>

<\_sre.SRE\_Match object; span=(0, 2), match='2s'>

\b 匹配一个单词的边界

\b：表示字母数字与非字母数字的边界，非字母数字与字母数字的边界。即下面ve的右边不能有字母和数字

In[21]: import re

print(re.match(r'.\*ve\b','ve.2testaabcd')) #因为在python中\代表转义，所以前面加上r消除转义

print(re.match(r'.\*ve\b','ve2testaabcd'))

<\_sre.SRE\_Match object; span=(0, 2), match='ve'>

None

\B 匹配非单词边界

In[22]: import re

print(re.match(r'.\*ve\B','2testaavebcdve')) #ve的右边需要有字母或者数字

print(re.match(r'.\*ve\B','2testaave3bcdve'))

<\_sre.SRE\_Match object; span=(0, 9), match='2testaave'>

<\_sre.SRE\_Match object; span=(0, 9), match='2testaave'>

匹配分组

| 匹配左右任意一个表达式

只要|两边任意一个表达式符合要求就行

In[23]: import re

print(re.match(r'\d[1-9]|\D[a-z]','2233')) #匹配|两边任意一个表达式

print(re.match(r'\d[1-9]|\D[a-z]','as'))

<\_sre.SRE\_Match object; span=(0, 2), match='22'>

<\_sre.SRE\_Match object; span=(0, 2), match='as'>

(ab) 将括号中字符作为一个分组

()中的内容会作为一个元组字符装在元组中

In[24]: import re

a = re.match(r'<h1>(.\*)<h1>','<h1>你好啊<h1>')

print(a.group()) #输出匹配的字符

print(a.groups()) #会将()中的内容会作为一个元组字符装在元组中

print('`````````````')

b = re.match(r'<h1>(.\*)(<h1>)','<h1>你好啊<h1>')

print(b.groups()) #有两括号就分为两个元组元素

print(b.group(0)) #group中默认是0

print(b.group(1)) #你好啊

print(b.group(2)) #h1

<h1>你好啊<h1>

('你好啊',)

`````````````

('你好啊', '<h1>')

<h1>你好啊<h1>

你好啊

<h1>

search

和match差不多用法，从字符串中进行搜索

In[25]: import re

print(re.match(r'\d\d','123test123test'))

print(re.search(r'\d\d','123test123test'))

<\_sre.SRE\_Match object; span=(0, 2), match='12'>

<\_sre.SRE\_Match object; span=(0, 2), match='12'>

findall

从字面意思上就可以看到，findall是寻找所有能匹配到的字符，并以列表的方式返回

In[26]: import re

print(re.search(r'test','123test123test'))

print(re.findall(r'test','123test123test')) #以列表的方式返回

<\_sre.SRE\_Match object; span=(3, 7), match='test'>

['test', 'test']

re.s

findall中另外一个属性re.S

在字符串a中，包含换行符\n，在这种情况下

如果不使用re.S参数，则只在每一行内进行匹配，如果一行没有，就换下一行重新开始。

而使用re.S参数以后，正则表达式会将这个字符串作为一个整体，在整体中进行匹配。

如下要寻找test.\*123的数据，因为test和123在不同的行，如果没加re.s的话，他会在每一个进行匹配查找而不是将字符串作为一个整体进行查找

In[27]: import re

a = """aaatestaa

aaaa123"""

print(re.findall(r'test.\*123',a))

print(re.findall(r'test.\*123',a,re.S))

[]

['testaa \naaaa123']

sub

查找字符串中所有相匹配的数据进行替换

sub(要替换的数据，替换成什么，要替换的数据所在的数据)

In[28]: import re

print(re.sub('php','python','php是世界上最好的语言——php'))

#输出 "python是世界上最好的语言——python"

python是世界上最好的语言——python

split

对字符串进行分割，并返回一个列表

In[29]: import re

s = "itcase,java:php-php3;html"

print(re.split(r",",s)) #以,号进行分割

print(re.split(r",|:|-|;",s)) #以,或者：或者-或者;进行分割

print(re.split(r",|:|-|%",s)) #找不到的分隔符就忽略

['itcase', 'java:php-php3;html']

['itcase', 'java', 'php', 'php3', 'html']

['itcase', 'java', 'php', 'php3;html']

贪婪与非贪婪

python里的数量词默认是贪婪的，总是尝试尽可能的匹配更多的字符。python中使用?号关闭贪婪模式

如

In[30]: import re

print(re.match(r"aa\d+","aa2323")) #会尽可能多的去匹配\d

print(re.match(r"aa\d+?","aa2323")) #尽可能少的去匹配\d

import re

s = "this is a number 234-235-22-423"

# 1.贪婪模式

resule = re.match(r"(.+)(\d+-\d+-\d+-\d)",s) #我们本想数字和字母拆解成两个分组

print(resule.groups()) #('this is a number 23', '4-235-22-4')但我们发现输出的结果中23的数字竟然被弄到前面去了

#因为+它会尽可能多的进行匹配，\d，只需要一个4就能满足，所以前面就尽可能多的匹配

# 2.关闭贪婪模式

#在数量词后面加上 ?，进入非贪婪模式，尽可能少的进行匹配

result = re.match(r"(.+?)(\d+-\d+-\d+-\d)",s)

print(result.groups()) #('this is a number ', '234-235-22-4')

('this is a number 23', '4-235-22-4')

('this is a number ', '234-235-22-4')

案例

匹配手机号

要求，手机号为11位，必须以1开头，且第二个数字为35678其种一个

import re

result = re.match(r'1[35678]\d{9}','13111111111')

print(result.group()) #匹配成功

result = re.match(r'1[35678]\d{9}','12111111111')

print(result) #none，第二位为2

result = re.match(r'1[35678]\d{9}','121111111112')

print(result) #none，有12位

提取网页源码中所有的文字

如下，将其中的所有文字提取出来，去掉标签。思路就是运用sub方法，将标签替换为空

s = """<div>

<p>岗位职责:</p>

<p>完成推荐算法、数据统计、接口、后台等服务器端相关工作</p>

<p><br></p>

<P>必备要求:</p>

<p>良好的自我驱动力和职业素养，工作积极主动、结果导向</p>

<p>&nbsp;<br></p>

<p>技术要求:</p>

<p>1、一年以上 Python开发经验，掌握面向对象分析和设计，了解设计模式</p>

<p>2、掌握HTTP协议，熟悉NVC、MVVM等概念以及相关wEB开发框架</p>

<p>3、掌握关系数据库开发设计，掌握SQL，熟练使用 MySQL/PostgresQL中的一种<br></p>

<p>4、掌握NoSQL、MQ，熟练使用对应技术解决方案</p>

<p>5、熟悉 Javascript/cSS/HTML5，JQuery,React.Vue.js</p>

<p>&nbsp;<br></p>

<p>加分项:</p>

<p>大数据，数理统计，机器学习，sklearn，高性能，大并发。</p>

</div>"""

要提取出来最重要的就是关闭贪婪模式，

result = re.sub(r'<.\*?>|&nbsp','',s) #

print(result)

如果关闭贪婪模式，<xx>中的内容会尽可能少的匹配，只要能够满足后面的>就行，然后<>xxx<>中xxx内容也替换掉了

提取图片地址

import re

s = """<img data-original="https://img02.sogoucdn.com/app/a/100520024/36189693dc8db6bd7c0be389f8aaddbd.jpg" src="https://img02.sogoucdn.com/app/a/100520024/36189693dc8db6bd7c0be389f8aaddbd.jpg" width="250" height="375" .jpg>"""

result1 = re.search(r"src=\"https.\*.jpg\"",s)

print(result1.group())

result2 = re.search(r"src=\"(https.\*.jpg)\"",s) #我只是想将网址提取出来，所以httpxx加括号，这样我就可以把它单独提取出来，src则不会出来

print(result2.groups()[0])

从字符串中提取指定范围内的字符串

案例1

如想从"csrftoken=ZjfvZBcDMcVs7kzYqexJqtKiJXIDxcmSnXhGD1ObR2deuHzaU0FuCxSmh10fSmhf; expires=Thu, 29 Jun 2023 07:59:04 GMT; Max-Age=31449600; Path=/; SameSite=Lax”

里面提取出token值

import re

str = "csrftoken=ZjfvZBcDMcVs7kzYqexJqtKiJXIDxcmSnXhGD1ObR2deuHzaU0FuCxSmh10fSmhf; expires=Thu, 29 Jun 2023 07:59:04 GMT; Max-Age=31449600; Path=/; SameSite=Lax"

a = re.search(r'=(.\*?);', str)

print(a.group(1)) #输出匹配的字符

案例2

从响应包里提取关键词所在的行，可以在正则中添加变量

import re

key = "威胁"

a = """

<body data-spm="7663354">

<div data-spm="1998410538">

<div class="header">

<div class="container">

<div class="message">

很抱歉，由于您访问的URL有可能对网站造成安全威胁，您的访问被阻断。

<div>您的请求ID是: <strong>

781bad0a16702307419116917e43b3</strong></div>

</div>

</div>

</div>

"""

res = re.search(r'<.\*>(.\*?%s.\*?)<.\*?>'%(key),a,re.S)

print(res.group(1).replace("\n","").replace(" ",""))

案例3 匹配ip地址

简单的匹配一下ip地址

import re

str = "http://10.1.1.1/index.html"

try:

s = re.search("\d\*?\.\d\*?\.\d\*?\.\d\*",str)

print(s.group())

except:

print("未能匹配出ip地址")

————————————————

**以上内容全部来自于一位博主的文章，个人觉得非常详细，故拿来排版后作为学习笔记。**

**版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。**

**原文链接：https://blog.csdn.net/qq\_44159028/article/details/120575621**

另外，我额外的关注了一下删除字典中键值对的操作，总结下来有以下几种：

一、pop（key）有返回值

dic = {'a':1,'b':2,'c':3}

dic = {'a':1,'b':2,'c':3}

# 删除key值为'a'的元素，并赋值给变量e1

e1 = dic.pop('a')

print(e1)

# 如果key不存在，则可以设置返回值

e2 = dic.pop('m','404')

print(e2)

# 如果key不存在，不设置返回值就报错

e3 = dic.pop('m')

1

404

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

KeyError Traceback (most recent call last)

<ipython-input-8-ba999cec27ff> in <module>

7 print(e2)

8 # 如果key不存在，不设置返回值就报错

----> 9 e3 = dic.pop('m')

KeyError: 'm'

二、del dic(key)

dic = {'a':1,'b':2,'c':3}

del dic['a']

print(dic)

{'b': 2, 'c': 3}

三、popitem() 字典popitem()方法作用是：随机返回并删除字典中的一对键和值。

d = {'a':1,'b':2,'c':3}

# 第一次运行

d.popitem()

print(d)

# 第二次运行

d.popitem()

print(d)

# 第三次运行

d.popitem()

print(d)

{'a': 1, 'b': 2}

{'a': 1}

{}

四、 clear（） 一次性删除所有字典元素

d = {'a':1,'b':2,'c':3}

print(d)

# 删除所有元素，允许d为{}

d.clear()

print(d)

​

{'a': 1, 'b': 2, 'c': 3}

{}