# 第二十章 python匹配

**本章所讲内容：**

**20.1 匹配介绍**

**20.2 Python re语法**

## 20.1 匹配介绍

### 20.1.1 正则表达式介绍

**在编程中，字符串是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：**

**创建一个匹配Email的正则表达式；**

**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。**

### 20.1.2 正则表达式过程

**首先：依次拿出表达式和文本中的字符比较，  
其次：如果每一个字符都能匹配，则匹配成功；一旦有匹配不成功的字符则匹配失败。  
最后：如果表达式中有量词或边界，这个过程会稍微有一些不同。**

## 20.2 Python re

### 20.2.1 语法

**在学习之前，我们了解在Python的string前面加上‘r’， 是为了告诉编译器这个string是个raw string，不要转义。**

**例如，\n 在raw string中，是两个字符，\和n， 而不会转意为换行符。由于正则表达式和 \ 会有冲突，因此，当一个字符串使用了正则表达式后，最好在前面加上'r'。**

**首先导入re内置模块：import re**

**#匹配自身**

**data = re.findall(r'abc','abc')  
print(data)**

**['abc']**

**#\为转义字符，将后面的字符改变原有的语义**

**data = re.findall(r'a\.c','a.c')  
print(data)**

**['a.c']**

**# . 匹配任意字符**

**data = re.findall(r'a.c','a.c')  
print(data)**

**['abc']**

**#字符集，里面的字符是可以看成b or c or d**

**#[A-Z]表示单词范围A,B…Z；[a-z]表示小写单词范围a,b…z;[0-9]范围0-9；**

**data = re.findall(r'a[bcd]e','ace')  
print(data)**

**['ace']**

**# \d 匹配数字字符[0-9]**

**data = re.findall(r"a\dc",'a2c')  
print(data)**

**['a2c']**

**# \D 匹配非数字字符**

**data = re.findall(r'a\Dc',"abc")  
print(data)**

**['abc']**

**# \s 匹配空字符，或者是\n\t\f\v(换行，制表符)**

**data = re.findall(r'a\sc',"abc")  
print(data)**

**['a c']**

**# \S 匹配非空字符相当于[^\s]**

**data = re.findall(r'a\Sc',"abc")  
print(data)**

**['abc']**

**# \w 匹配单词字符[A-Za-z0-9\_]**

**data = re.findall(r'a\wc','abc')  
print(data)**

**data = re.findall(r'a\wc','accc')  
print(data)**

**# \W 非单词字符相当于[^\w]**

**data = re.findall(r'a\Wc','a c')  
print(data)**

**['a c']**

**# \* 匹配前一个字符0或无限次**

**data = re.findall(r'abc\*','abccc')  
print(data)**

**['abccc']**

**# + 匹配前一个字符1次或无限次**

**data = re.findall(r'abc+','abccc')  
print(data)**

**['abccc']**

**#{m} 匹配前一个字符的m次**

**data = re.findall(r'ab{2}','abbc')  
print(data)**

**#{m,n}匹配前一个字符的m次到n次**

**data = re.findall(r'ab{1,2}','abbc')  
print(data)**

**['abb']**

**data = re.findall(r'ab{1,2}','abc')  
print(data)**

**['ab']**

**# ^ 匹配字符串开头**

**data = re.findall(r'^abc','abcdef')  
print(data)**

**['abc']**

**# $ 匹配字符串结尾**

**data = re.findall(r'abc$','abcdef')  
print(data)**

**[]**

**data = re.findall(r'^abc$','abcdef')  
print(data)**

**[]**

**data = re.findall(r'def$','abcdef')  
print(data)**

**['def']**

**# \A仅匹配字符串开头**

**data = re.findall(r'\Aabc','abc')  
print(data)**

**['abc']**

**data = re.findall(r'\Aabc','abcdef')  
print(data)**

**['abc']**

**# \Z 仅匹配字符串结尾**

**data = re.findall(r'def\Z','abcdef')  
print(data)**

**['def']**

**#\b 单词边界，单词边界就是单词和符号之间的边界**

**data = re.findall(r'a\b#bc','a#bc')  
print(data)**

**['a#bc']**

**#相当于[^\b]**

**data = re.findall(r'a\Bbc','abc')  
print(data)**

**['abc']**

**# | 代表左右表达式中任意匹配一个，它总是先尝试匹配左边的表达式，一旦成功会跳过匹配右面的表达式**

**#如果 | 没有被包括在（）中，则它的范围是整个表达式**

**data = re.findall(r'abc|def','adc')  
print(data)**

**[]**

**data = re.findall(r'abc|def','abc')  
print(data)**

**['abc']**

**data = re.findall(r'abc|def','def')  
print(data)**

**['def']**

**data = re.findall(r'abc|def','adcdef')  
print(data)**

**['def']**

**data = re.findall(r'abc|def','abcdef')  
print(data)**

**['abc', 'def']**

**data = re.findall(r'abc|def','''abcdef''')  
print(data)**

**['abc', 'def']**

**#被包括的表达式将作为分组，从表达式左边开始每遇到一个分组的左括号，编号+1  
#另外分组表达式中作为一个整体，可以后接数量词。**

**#表达式中的 | 仅在该组中有效**

**data = re.findall(r'(abc){2}','abcabc')  
print(data)**

**['abc']**

**#注：这与匹配优先级有关，如果取消格式就在（）表达式中加入？:**

**data = re.findall(r'(?:abc){2}','abcabc')  
print(data)**

**data = re.findall(r'a(123|466)c','a466c')  
print(data)**

**['466']**

**>>>**

**匹配手机号：**

**分析：手机号有十一位数字，第一位通常为1，第二位第三位也有也有固定的。**

1. **^1(3[0-9]|6[189]|8[6689])+[0-9]{8}$**
2. **^(1|+861)(3[0-9]|6[189]|8[689])[0-9]{8}$**
3. **^1[0-9]{10}$**

**匹配邮箱地址**

**26个大小写英文字母表示为a-zA-Z**

**数字表示为0-9**

**下划线表示为\_ ,但是不允许首字母出现**

**中划线表示为-，但是不允许首字母出现**

**由于名称是由若干个字母、数字、下划线和中划线组成，所以需要用到+表示多次出现**

**根据以上条件得出邮件名称表达式：[a-zA-Z0-9\_-]+**

**得出：**

**^[a-zA-Z0-9\_-]+@[a-zA-Z0-9\_-]+[\.][a-zA-Z0-9\_-]+$**

**有的邮箱在第一阶段中有一个 “.”， 如果这个第一段有个 . 怎么办？**

**^[a-zA-Z0-9-\_\.]+@[a-zA-Z0-9\_-]+[\.][a-zA-Z0-9\_-]+$**

**20.2.2 匹配模式**

**1. re.I(re.IGNORECASE): 忽略大小写（括号内是完整写法,下同）；**

** 2. re.M(MULTILINE): 多行模式，改变'^'和'$'的行为；**

**3. re.S(DOTALL): 点任意匹配模式，改变'.'的行为。**

**re.I 表示忽略大小写。**

**案例：**

**data = re.findall(r'Hello','hello,world',re.I)  
print(data)**

**['hello']**

**>>> re.findall(r'Hello','hello,world')**

**[]**

**>>>**

**Re.M 多行模式，改变‘^’和‘$’的行为**

**案例：**

**s = 'i am langzi\nyou are qingren\nshe is xiaosan'  
print(s)**

**i am langzi**

**you are qingren**

**she is xiaosan**

**data = re.findall(r'\w+$',s,re.M)  
print(data)**

**['langzi', 'qingren', 'xiaosan']**

**在Python的正则表达式中，有一个参数为re.S。它表示“.”（不包含外侧双引号）的作用扩展到整个字符串，包括“\n”。看如下代码：**

**import re  
a = '''asdfhellopass:  
 123  
 worldaf  
 '''  
b = re.findall('hello(.\*)world',a)  
c = re.findall('hello(.\*)world',a,re.S)  
print('b is ' , b)  
print('c is ' , c)**

**运行结果如下：**

**b is []**

**c is ['pass:\n\t123\n\t']**

**正则表达式中，“.”的作用是匹配除“\n”以外的任何字符，也就是说，它是在一行中进行匹配。这里的“行”是以“\n”进行区分的。a字符串有每行的末尾有一个“\n”，不过它不可见。**

**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，不会跨行。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，将“\n”当做一个普通的字符加入到这个字符串中，在整体中进行匹配。**

### 20.2.2 相关注释

**1、数量词的贪婪模式与非贪婪模式**

**正则表达式通常用于在文本中查找匹配的字符串。Python里数量词默认是贪婪的（在少数语言里也可能是默认非贪婪），总是尝试匹配尽可能多的字符；非贪婪的则相反，总是尝试匹配尽可能少的字符。例如：正则表达式”ab\*”如果用于查找”abbbc”，将找到”abbb”。而如果使用非贪婪的数量词”ab\*?”，将找到”a”。注：我们一般使用非贪婪模式来提取**

**贪婪匹配**

**import re  
test\_str = '''  
<thead>  
 <tr>  
 <th class = 'ok'>Month</th>  
 <th>Savings</th>  
 </tr>  
 </thead>  
'''  
print(re.findall(r"<tr>(.\*)</th>",test\_str,re.S))**

**运行结果如下：**

**["\n <th class = 'ok'>Month</th>\n <th>Savings"]**

**你会发现一只会匹配到</html>前面的所有内容,会尽可能多的去匹配**

**非贪婪匹配**

**import re  
test\_str = '''  
<thead>  
 <tr>  
 <th class = 'ok'>Month</th>  
 <th>Savings</th>  
 </tr>  
 </thead>  
'''  
print(re.findall(r"<tr>(.\*?)</th>",test\_str,re.S))**

**运行结果如下：**

**["\n <th class = 'ok'>Month"]**

**只匹配到了</th>标签前面的，尽可能少的去匹配**

**2、反斜杠问题**

**与大多数编程语言相同，正则表达式里使用”\”作为转义字符，这就可能造成反斜杠困扰。假如你需要匹配文本中的字符”\”，那么使用编程语言表示的正则表达式里将需要4个反斜杠”\\\\”：前两个和后两个分别用于在编程语言里转义成反斜杠，转换成两个反斜杠后再在正则表达式里转义成一个反斜杠。 Python里的原生字符串很好地解决了这个问题，这个例子中的正则表达式可以使用r”\\”表示。同样，匹配一个数字的”\\d”可以写成r”\d”。有了原生字符串，妈妈也不用担心是不是漏写了反斜杠，写出来的表达式也更直观。**

**import re  
temp = 'abc123'  
  
print(re.findall('\\d',temp))  
  
print(re.findall(r'\d',temp))**

**运行结果如下：**

**['1', '2', '3']**

**['1', '2', '3']**

**print(re.findall('^www\..\*?m$','www.baidu.com'))  
print(re.findall('^www\..\*?m$','www.m'))**

**运行结果如下：**

**['www.baidu.com']**

**['www.m']**

### 20.2.4 匹配函数

**Python通过re模块提供对正则表达式的支持。使用re的一般步骤是先将正则表达式的字符串形式编译为Pattern实例，然后使用Pattern实例处理文本并获得匹配结果。我们可以通过re.compile()方法实现。**

1. **compile()**

**re.compile(strPattern[, flag])**

**这个方法是Pattern类的工厂方法，用于将字符串形式的正则表达式编译为Pattern对象。**

**第二个参数flag是匹配模式。**

**import re  
temp = '''first line  
second line  
third line  
'''  
*#将正在表达式编译为pattern对象*pat = re.compile('.+',re.S)  
*#使用match匹配文本，获取匹配结果，我发匹配时返回None*result = pat.match(temp)  
*#判断是否有结果，如果有结果进行判断操作*if result:  
 *#获取分组信息* print(result.group())**

**2、match()**

**re.match(pattern, string[, flags])**

**这个方法将会从string（我们要匹配的字符串）的开头开始，尝试匹配pattern，一直向后匹配，如果遇到无法匹配的字符，立即返回None，如果匹配未结束已经到达string的末尾，也会返回None。两个结果均表示匹配失败，否则匹配pattern成功，同时匹配终止，不再对string向后匹配。**

**import re  
temp = '123 hello world'  
ret = re.match(r'\d+',temp)  
if ret:  
 print(ret.group())**

**3、search**

**re.search(pattern, string[, flags])**

**search方法与match方法极其类似，区别在于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，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。同样，search方法的返回对象同match()返回对象的方法和属性一样。**

**import re  
temp = 'hello world 123'  
ret = re.search(r'\d+',temp)  
if ret:  
 print(ret.group())**

**执行结果（如果用match匹配的话，执行不成功，因为match是从开头开始查找的）**

**123**

**4、findall()**

**搜索string，以列表形式返回全部能匹配的子串。**

**import re  
temp = 'hello world 123 466'  
ret = re.findall(r'\d+',temp)  
print(ret)**

**执行结果（如果用search只能匹配到123）**

**['123', '466']**

**5、split()**

**类似于字符串的split(),将字符串按照匹配到的内容进行切分；**

**import re  
string = 'he123llo world'  
print(re.split(r'\d{3}',string))**

**运行结果如下：**

**['he', 'llo world']**

**6、sub()**

**类似于字符串的replace，将匹配到的内容转换为指定的字符**

**import re  
string = 'he123llo world'  
*#第一个匹配，第二个参数为填充，最后一个参数为字符串对象*print(re.sub(r'\d','0',string))**

**运行结果如下：**

**he000llo world**

**7、group 和groups**

**group([group1, ...])**

**返回Match对象的一个或多个子组。**

**如果单个参数，结果是一个单一的字符串 ；**

**如果有多个参数，其结果是参数每一项的元组。**

**如果没有参数， group1默认为零 （整场比赛返回）。**

**如果groupN参数为零，相应的返回值是整个匹配的字符串 ；**

**如果它是在具有包容性的范围 [1..99]，它是模式进行相应的括号组匹配的字符串。**

**如果组编号是负值或大于在模式中定义的组的数目，被引发[IndexError](http://python.usyiyi.cn/documents/python_278/library/exceptions.html" \l "exceptions.IndexError" \o "exceptions.IndexError)异常。**

**如果一组包含在模式不匹配的一部分中，相应的结果是没有的。**

**如果一组包含在模式匹配多次的一部分，则返回最后一次结果。**

**import re  
m = re.match(r"(\w+) (\w+)", "For Django while")  
print(m.group(0))**

**结果：For Django**

**print(m.group(1))**

**结果：For**

**print(m.group(2))**

**结果：Django**

**print(m.group(1,2))**

**结果：(For Django)**

**如果正则表达式使用（？P < 名称 >...) 语法， groupN参数也可能查明群体按他们的通讯组名称的字符串。如果字符串参数不用作模式中的组名称，被引发[IndexError](http://python.usyiyi.cn/documents/python_278/library/exceptions.html" \l "exceptions.IndexError" \o "exceptions.IndexError)异常。**

**import re  
m = re.match(r"(?P<first\_name>\w+) (?P<last\_name>\w+)", "Malcolm Reynolds")  
print(m.group('first\_name'))  
print(m.group('last\_name'))**

**Malcolm**

**Reynolds**

**通过它们的索引还可以获取到已命名的组：**

**print(m.group(1))  
print(m.group(2))**

**运行结果如下：**

**Malcolm**

**Reynolds**

**如果一组匹配多次，只有最后一个匹配可访问：**

**import re  
m = re.match(r"(..)+", "a1b2c3")  
print(m.group(1))**

**groups([default])**

**返回包含所有匹配到的子组的元组， 从1到模式中的所有组。如果没有参加匹配 ，它将默认为None。**

**例子：**

**m = re.match(r"(\d+)\.(\d+)", "24.1632")  
print(m.groups())**

**('24', '1632')**

**如果我们使小数点和一切在它以后可选，并不是所有的组可能会参加，这些group将默认为无，除非给出了默认参数：**

**m = re.match(r"(\d+)\.?(\d+)?", "24")  
print(m.groups())**

**('24', None)**

**#给定默认参数**

**print(m.groups('0'))**

**7、group和groups的区别**

**import re  
pattern = re.compile(r'(\w+) (\w+)')  
matche = pattern.match('hello world')  
print(matche.groups())  
print(matche.group())**

**运行结果如下：**

**hello**

**('hell', 'o')**

**总结：**

**20.1 匹配介绍**

**20.2 Python re语法**