Python编程基础教程

智普教育

www.jeapedu.com

March 19, 2014

目 录

第 13 章. re正则模块

13.1 正则表达式简介

正则表达式并不是Python的一部分。正则表达式是用于处理字符串的强大工具，拥有自己独特的语法以及一个独立的处理引擎，效率上可能不如str自带的方法，但功能十分强大。得益于这一点，在提供了正则表达式的语言里，正则表达式的语法都是一样的，区别只在于不同的编程语言实现支持的语法数量不同；但不用担心，不被支持的语法通常是不常用的部分。如果已经在其他语言里使用过正则表达式，只需要简单看一看就可以上手了。就个人而言，主要用它来做一些复杂字符串分析，提取想要的信息，学习原则：够用就行，需要的时候在深入。[1]

13.1.1 正则表达式的定义

正则表达式就是用某种模式去匹配一类字符串的公式，主要用来描述字符串匹配的工具。

匹配

在正则表达式中，匹配是最常用的一个词语，它描述了正则表达式动作结果。给定一段文本或字符串，使用正则表达式从文本或字符串中查找出符合正则表达式的字符串。有可能文本或字符存在不止一个部分满足给定的正则表达式，这时每一个这样的部分被称为一个匹配。

13.1.2 正则表达式的应用

正则表达式在程序设计语言中存在着广泛的应用，特别是用来处理字符串。如匹配字符串、查找字符串、替换字符串等。可以说，正则表达式是一段文本或一个公式，它是用来描述用某种模式去匹配一类字符串的公式，并且该公式具有一定的模式。正则表达式就是用某种模式去匹配一类字符串的公式，主要用来描述字符串匹配的工具。

正则表达式常见的应用如下:

- 验证字符串，即验证给定的字符串或子字符串是否符合指定特征，譬如验证是否是合法的邮件地址、验证是否为合法的HTTP地址等。
- 查找字符串，从给定的文本中查找符合指定特征的字符串，比查找固定字符串更加灵活方便。

[1] http://www.cnblogs.com/huxi/archive/2010/07/04/1771073.html

- 替换字符串，即把给定的字符串中的符合指定特征的子字符串替换为其他字符串，比普通的替换更强大。
- 提取字符串，即从给定的字符串中提取符合指定特征的子字符串。

*13.2 Python*里使用正则

那么我们实际在 Python中是如何使用正则呢？ Python里有个re模块，提供了一个正则表达式引擎的接口。

13.2.1 通过正则对象进行正则匹配

可以首先将正则表达式(字符串)首先编译(re.compile 函数)成re 模块的实例对象，再用这个正则表达式实例对象调用各种方法函数来进行正则匹配。

一旦有了已经编译了的正则表达式的实例对象，就可以借助实例对象调用match、search、findall、split 等方法函数对某字符串进行正则匹配了。

下面实例展示一下如何通过Python的re模块来使用正则表达式进行字符串匹配。

在Python里使用正则表达式步骤比较简单，首先是写一个正则表示式(字符串形式)，接着把这个正则表达式编译成正则对象，最后通过这个正则对象调用方法函数去匹配某个字符串从而得到想要的匹配结果。

Listing 13.1: hello_regular_expression.py

```
1  # 本例是在字符串 s 里查找出以 e 字母开始的长度为 2 的所有子串
2  # 通俗点儿: 找出 s 里以字母 e 开始第二字符随意字母的子串。
3  import re
4  # 要查找的字符串
5  s = "hello www.jeapedu.com"
6  print '--------------------------'
7  # 正则表达式: 以 e 开始第二个为字符
8  regexp = "e\w"
9  # 编译生成正则对象
10 pat = re.compile(regexp)
11 # 通过正则对象调用正则方法函数findall
12 result = pat.findall(s)
13 # 打印结果
14 print result
15 print '--------------------------'
```

运行结果如下所示:

```
--------------------------
['el', 'ea', 'ed']
--------------------------
```

13.2.2 通过re模块调用使用正则

上例也可以不编译regexp正则表达式而采用直通过re模块调用findall函数来进行正则匹配操作。

Listing 13.2: Hello_re.py

```
1 import re
2 s = "hello www.jeapedu.com"
3 print '--------------------------'
4 regexp = "\wo"
5 result = re.findall(regexp, s)
6 print result
7 print '--------------------------'
```

运行结果如下所示:

```
--------------------------
['lo', 'co']
--------------------------
```

13.3 正则表达式

正则表达式由字符、预定义字符集、量词、边界元素等构成，上边示例中的"e\w"和"\wo"中的\w 就是预定义字符。借助这些元素可以构造出各种复杂的正则表达式从而可以从某字符串里提取出特定的子串来。

13.3.1 一般字符

使用普通的字符构造的正则表达式，可以从字符串里找出所有的符合正则的字串来，举例说明如下:

Listing 13.3: re_chars.py

```
1 #coding:utf-8
2 import re
3 s = "www.jeapedu.com" * 3
4 # 全词匹配，看看 s 里是否有"jeapedu"
5 regexp = "jeapedu"
6 pat = re.compile(regexp)
7 result = pat.findall(s)
8 print '----------------------------------------'
9 print result
10 print '----------------------------------------'
11 regexp = "w.j"
```

```
12 pat = re.compile(regexp)
13 result = pat.findall(s)
14 print result
15 print '----------------------------------------'
```

运行结果如下所示:

```
----------------------------------------
['jeapedu', 'jeapedu', 'jeapedu']
----------------------------------------
['w.j', 'w.j', 'w.j']
----------------------------------------
```

13.3.2 字符集

用方括号括起来的一些字符的集合称之为字符集，方括号在表达式里仅占一位，这位上可以是方括号里任意一个字符满足，即认为匹配成功，举例说明如下:

Listing 13.4: re_charsetsScope1.py

```
1 #coding:utf-8
2 import re
3 s = "hello www.jeapedu.com"
4 # s 里有 el 或者 ea 么?
5 regexp = "e[la]"
6 # 生成正则表达式对象
7 pat = re.compile(regexp)
8 result = pat.findall(s)
9 print '----------------------------------------'
10 print result
11 print '----------------------------------------'
```

运行结果如下所示:

```
----------------------------------------
['el', 'ea']
----------------------------------------
```

从运行结果可以看出子串"ed"没有被找到，为何？因为正则表达式方块里没有d这个字符。

方括号里可以用减号表示一个范围，起点到终点，就不用一一列出所有字符了，多个范围之间可用用逗号间隔，^放在最前表示排除括号内的所有字符，而一些特殊符号]、-最好放在字符符号前边，否则得使用转义了。

Listing 13.5: re_charsetsScope2.py

```
#coding : utf-8
import re
s = "as1b ab2 ab3e bb4d ab68e ab7ot ccb04a xbbc2"
regexp = "b[0-9a-z]"
pat = re.compile(regexp)
result = pat.findall(s)
print '----------------------------------------'
print result
print '----------------------------------------'
s = "call jeapedu 153-1153-1358 or 010-62231977"
# 第一个方括号 - 、0 、3 任意可选
# 第二个方块好 0 、1 、2 、3 、4 、5 、6 任意一个均认为匹配
# 结果是长度为 2 的子串
regexp = "[-03][0-6]"
pat = re.compile(regexp)
result = pat.findall(s)
print result
print '----------------------------------------'
```

运行结果如下所示:

```
----------------------------------------
['b2', 'b3', 'bb', 'b6', 'b7', 'b0', 'bb']
----------------------------------------
['-1', '-1', '35', '01', '-6', '31']
----------------------------------------
```

13.3.3 预定义字符集

在正则表达式里可以用\d匹配任意数字(等价[0-9])、\w任意字符(等价[a-zA-Z0-9])，\D匹配任意非数字(等价[^\d])、\W任意非字符(等价[^\w])，\s匹配空白字符(等价于[空格\t\n\f\v])。

Listing 13.6: re_predefinedCharsets.py

```
#coding : utf-8
import re
print '----------------------------------------'
s = "call jeapedu 153-1153-1358 or 010-62231977"
regexp = "\d\d\d\D\d\d\d\d"
pat = re.compile(regexp)
result = pat.findall(s)
```

```
print result
print '----------------------------------------'
# 字符开始中间是空格最后是数字的子串
regexp = "\w\s\d"
pat = re.compile(regexp)
result = pat.findall(s)
print result
print '----------------------------------------'
```

运行结果如下所示:

```
----------------------------------------
['153-1153', '010-6223']
----------------------------------------
['u 1', 'r 0']
----------------------------------------
```

13.3.4 量词

上边代码多个\d写起来太费劲了吧，能不能少些几个就可以把完整的电话提取出来呢?

这里可以使用量词，来表示**前词**重复次数。用花括号将重复次数(可重复至少m到至多n 次)括起来即可。

Listing 13.7: re_repeat.py

```
#coding : utf-8
import re

s = "call jeapedu 153-1153-1358 or 010-62231977 qq
    1941847311"
# 等价于 "\d\d\d\D\d\d\d\d\D\d\d\d\d"
regexp = "\d{3}\D\d{4}\D\d{4}"
result = re.findall(regexp, s)
print '----------------------------------------'
print result
print '----------------------------------------'
# 连续8 个到11 个数字组成的字串
regexp = "\d{8, 11}"
result = re.findall(regexp, s)
print result
print '----------------------------------------'
```

运行结果如下所示:

```
----------------------------------------
['153-1153-1358']
----------------------------------------
['62231977', '1941847311']
----------------------------------------
```

用星号*表示的次数是**前词**可重复0次到无限次。

Listing 13.8: re_repeatStart.py

```
1  #coding : utf-8
2  import re
3
4  print '----------------------------------------'
5  s = "jeab jeapedu jeappython jeappprogramme"
6  # 字母 p 可不出现或者连续多个 p
7  regexp = "jeap*"
8  pat = re.compile(regexp)
9  result = pat.findall(s)
10 print result
11 print '----------------------------------------'
```

运行结果如下所示:

```
----------------------------------------
['jea', 'jeap', 'jeapp', 'jeappp']
----------------------------------------
```

用加号+表示的次数是**前词**可重复1次到无限次。

Listing 13.9: re_repeatPlus.py

```
1  #coding : utf-8
2  import re
3
4  print '----------------------------------------'
5  s = "jeab jeapedu jeappython jeappprogramme"
6  # p 至少有一个
7  regexp = "jeap+"
8  pat = re.compile(regexp)
9  result = pat.findall(s)
10 print result
11 print '----------------------------------------'
```

运行结果如下所示:

```
----------------------------------------
['jeap', 'jeapp', 'jeappp']
----------------------------------------
```

用问号？表示0次或者**前词**可重复1次。

Listing 13.10: re_repeatQuestion.py

```
1 #coding : utf-8
2 import re
3 print '----------------------------------------'
4 s = "jeab jeapedu jeappython jeappprogramme"
5 # 可以没有，有的话只能一个p
6 regexp = "jeap?"
7 pat = re.compile(regexp)
8 result = pat.findall(s)
9 print result
10 print '----------------------------------------'
```

运行结果如下所示:

```
----------------------------------------
['jea', 'jeap', 'jeap', 'jeap']
----------------------------------------
```

量词的贪婪模式与非贪婪模式

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"。

Listing 13.11: re_repeatGreed.py

```
1 s = "abbbbc"
2 print '---ab*----------------------------------'
3 regexp = "ab*"
4 pat = re.compile(regexp)
5 result = pat.findall(s)
6 print result
7 print '---ab*?---------------------------------'
8 # b* 是说 b 最少可不出现。？选不出现 b
9 regexp = "ab*?"
```

```
10 pat = re.compile(regexp)
11 result = pat.findall(s)
12 print result
13 print '---ab+-----------------------------------'
14 regexp = "ab+"
15 pat = re.compile(regexp)
16 result = pat.findall(s)
17 print result
18 print '---ab+?----------------------------------'
19 # b+ 要求字母 b 至少出现一次。? 选最少一次 b 字母
20 regexp = "ab+?"
21 pat = re.compile(regexp)
22 result = pat.findall(s)
23 print result
24 print '-----------------------------------------'
```

运行结果如下所示:

```
---ab*-------------------------------------
['abbbb']
---ab*?------------------------------------
['a']
---ab+-------------------------------------
['abbbb']
---ab+?------------------------------------
['ab']
-----------------------------------------
```

13.3.5 边界限定词

有的时候希望匹配以某字符开始或者以某字符结尾的单词，这时候可以使用正则里的预定义好的边界限定词来完成。常用的边界限定词有以下几个:

限定以某字符开始或结尾的限定词的单词有\b。

Listing 13.12: re_EscapeCodes.py

```
1 #coding : utf-8
2 import re
3 print '-----------------------------------------'
4 s = "welcome to jeapedu.com to learn python has eat?"
5 # 以 e 开始的单词
6 regexp = r"\bea"
7 pat = re.compile(regexp)
```

```
result = pat.findall(s)
print result
print '----------------------------------------'
regexp = r"\w*\Wcom\b"
pat = re.compile(regexp)
result = pat.findall(s)
print result
print '----------------------------------------'
```

运行结果如下所示:

```
----------------------------------------
['ea']
----------------------------------------
['jeapedu.com']
----------------------------------------
```

匹配字符串头部、尾部的限定词\A、^、\Z、$。

Listing 13.13: re_Anchoring.py

```
#coding : utf-8
import re
s = "welcome to www.jeapedu.com to learn python has eat
    ?"
print '---1----------------------------------\n\n\n'
regexp = r"\w+\W$"
pat = re.compile(regexp)
result = pat.findall(s)
print result
print '---2----------------------------------'
regexp = r"\A\w+"
pat = re.compile(regexp)
result = pat.findall(s)
print result
print '---3----------------------------------'
s = '''call jeapedu 153-1153-1358 or 010-62231977
    or qq 1941847311
    23aaa welcome to www.jeapedu to learn Python!
    Python is good!'''
print s
regexp = r"\w+\W$"
# compile 的第二参数可以指定匹配模式, 're.M支持多行匹配'
```

```
22 # 多行模式，改变和'^''$'的行为'
23 pat = re.compile(regexp, re.M)
24 result = pat.findall(s)
25 print result
26 print '----------------------------------------'
```

运行结果如下所示:

```
---1------------------------------------
['eat?']
---2------------------------------------
['welcome']
---3------------------------------------
['Python!', 'good!']
----------------------------------------
```

13.3.6 逻辑与分组

逻辑

选择逻辑'|'前后的正则，一般先查'|'前的正则是否匹配，如果前边的不匹配再查查看'|' 后的是否匹配?

Listing 13.14: re_Selected.py

```
1 #coding : utf-8
2 import re
3 print '----------------------------------------'
4 s = '''jeapedu.com or
5         quanzhan.org or
6         http://uliweb.clkg.org/tutorial or
7         dou.bz/35ODPo'''
8 regexp = r"\w+\Wcom|\w+\Worg"
9 pat = re.compile(regexp,re.M)
10 result = pat.findall(s)
11 print result
12 print '----------------------------------------'
```

运行结果如下所示:

```
['jeapedu.com', 'quanzhan.org', 'clkg.org']
```

分组

Python正则表达式里的分组比较复杂，它的意思是将一些正则表达式元素组合在一起用圆括号括起来，括号外部可能还会有其他正则表达式元素，故分组也成子模式，即模式里的模式。含分组的正则表达式匹配结果输出时，只输出括号内匹配子串，而括号外部匹配内容不会输出。

可以这样去理解，正则首先用真个正则表达式去匹配字符串，如果这个大的正则能够匹配，那么输出这个大的正则里的分组能够匹配的子串作为这个大的正则的结果。

Listing 13.15: re_Groups.py

```
1  #coding : utf-8
2  import re
3  print '----------------------------------------'
4  s = "abc" * 2 + "abd" * 2 + "abcd" * 2
5  # 看看 s 是否含有"abcabc有则输出""abc而不是""abcabc"
6  regexp = "(abc){2}"
7  #regexp = "(abc)"
8  pat = re.compile(regexp,re.M)
9  result = pat.findall(s)
10 print result
11 print '----------------------------------------'
```

运行结果如下所示:

```
----------------------------------------
['abc']
----------------------------------------
```

再看一个例子，可能会更好的理解逻辑和分组。本例是提取域名后缀。

Listing 13.16: re_SelectGroups.py

```
1  #coding : utf-8
2  import re
3  print '----------------------------------------aa--'
4  s = '''www.jeapedu.com ios.jeapedu.org
5          video.jeapedu.gov lesson.jeapdu.com.cn
6          welcome incoming forgive egovation
7      '''
8  # 为何第 3 行的'com、''gov、''org没提取出来？'
9  # 为何没在输出结果里？'.'
10 regexp = r"\W(com|org|gov)"
11 # 're.M支持多行匹配的模式字'
```

```
12 pat = re.compile(regexp, re.M)
13 result = pat.findall(s)
14 print result
15 print '----------------------------------------'
```

运行结果如下所示:

```
----------------------------------------
['com', 'org', 'gov', 'com']
----------------------------------------
```

13.3.7 反向引用

当一个正则表达式被分组之后，每一个组将自动被赋予一个组号，该组号可以代表该组的表达式。其中，组号的编制规则为：从左到右、以分组的左括号"("为标志，第一个分组的组号为1，第二个分组的组号为2，以此类推。

反向引用提供**查找重复字符组**的方便的方法。它们可被认为是再次匹配同一个字符串的快捷指令。后向引用可以使用数字命名（即默认名称）的组号，也可以使用指定命名的组号。[2]

Listing 13.17: re_reference.py

```
1 #coding : utf-8
2 import re
3 # 如果 s = "1abc22，会有结果么？"
4 # '\1' 匹配的内容和 '\d' 得一模一样
5 s = "1abc12"
6 regexp = r'(\d)abc\1'
7 pat = re.compile(regexp)
8 result = pat.findall(s)
9 print result
10 print '----------------------------------------'
```

运行结果如下所示:

```
----------------------------------------
['1']
----------------------------------------
```

13.4 re模块

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（一

[2]http://blog.csdn.net/wlzhengzebiaodashi/article/details/2213225

个Match实例），最后使用Match实例获得信息，进行其他的操作。

13.4.1 Pattern对象及实例方法函数

Pattern对象是一个编译好的正则表达式，通过Pattern提供的一系列方法可以对文本进行匹配查找，一般得到Match对象。

findall函数

findall方法函数可以通过patternObject对象来调用，也可以用re直接调用。
用法如下:

```
resultList = patternObject.findall(string)
resultList = re.findall(regularExpressionString,string)
```

函数会在string搜索匹配子串，以列表形式返回全部能匹配的子串。其中pattern–Object对象需用compile 方法函数生成。

```
patternObject = re.compile(regularExpressionString)
```

search函数

这个方法用于查找字符串中可以匹配成功的子串。只到找到第一个匹配即返回，如果字符串没有匹配，则返回None。

```
search(string[, pos[, endpos]])
re.search(pattern, string[, flags])
```

从string的pos下标处起尝试匹配pattern，如果pattern结束时仍可匹配，则返回一个Match对象；若无法匹配，则将pos加1后重新尝试匹配；直到pos=endpos时仍无法匹配则返回None。

split函数

用法和作用与string里split函数基本一样。

Listing 13.18: re_split.py

```
1 #coding : utf-8
2 import re
3 p = re.compile(r'\W+')
4 print p.split('www.jeapedu.com')
```

运行结果如下所示:

```
['www', 'jeapedu', 'com']
```

finditer函数

finditer函数搜索string，返回一个顺序访问每一个匹配结果（Match对象）的迭代器。

```
finditer(string[, pos[, endpos]])
re.finditer(pattern, string[, flags]):
```

举例说明一下:

Listing 13.19: re_finditer.py

```
1  #coding : utf-8
2  import re
3  s = "www.jeapedu.com" * 3
4  regexp = r"\W(\w+)\W"
5  result = re.findall(regexp, s)
6  print result
7  regexp = r"\w{7}"
8  result = re.finditer(regexp, s)
9  for i in result:
10     print i.group()
```

运行结果如下所示:

```
jeapedu
jeapedu
jeapedu
```

13.4.2 Match对象及实例方法函数

Match对象是一次匹配的结果，包含了很多关于此次匹配的信息，可以使用Match提供的可读属性或方法来获取这些信息。[3]

Listing 13.20: re_matchObject.py

```
1  #coding : utf-8
2  import re
3
4  print '----------------------------------------'
5  s = 'hello jeapedu.com' * 3
6  regexp = 'jeapedu'
7  #regexp = 'hello'
8  pat = re.compile(regexp)
9  matobj = pat.match(s)
```

[3]http://www.cnblogs.com/huxi/archive/2010/07/04/1771073.html

```
10 if matobj:
11     # 使用'Match方法函数获得的分组信息'
12     print matobj.group()
13 else:
14     print 'not matched'
15 print '----------------------------------------'
```

运行结果如下所示:

```
----------------------------------------
not matched
----------------------------------------
```

为何？因为match方法函数会从字符串的开始的地方开始匹配，由于字串s是以"hello"开始，而要匹配的正则子串是"jeapedu"，故match的返回值是None，所以打印了not matched。

match对象的属性

- string: 匹配时使用的文本。
- re: 匹配时使用的Pattern对象。
- pos: 文本中正则表达式开始搜索的索引。值与Pattern.match()和Pattern.seach()方法的同名参数相同。
- endpos: 文本中正则表达式结束搜索的索引。值与Pattern.match()和Pattern.seach()方法的同名参数相同。
- lastindex: 最后一个被捕获的分组在文本中的索引。如果没有被捕获的分组，将为None。
- lastgroup: 最后一个被捕获的分组的别名。如果这个分组没有别名或者没有被捕获的分组，将为None。

match对象的方法

- group([group1, ...]): 获得一个或多个分组截获的字符串；指定多个参数时将以元组形式返回。group1可以使用编号也可以使用别名；编号0代表整个匹配的子串；不填写参数时，返回group(0)；没有截获字符串的组返回None；截获了多次的组返回最后一次截获的子串。
- groups([default]): 以元组形式返回全部分组截获的字符串。相当于调用group(1,2,...last)。default表示没有截获字符串的组以这个值替代，默认为None。
- groupdict([default]): 返回以有别名的组的别名为键、以该组截获的子串为值的字典，没有别名的组不包含在内。default含义同上。
- start([group]): 返回指定的组截获的子串在string中的起始索引（子串第一个字符的索引）。group默认值为0。
- end([group]): 返回指定的组截获的子串在string中的结束索引（子串最后一个字符的索引+1）。group默认值为0。

- span([group]): 返回(start(group), end(group))。
- expand(template):

Listing 13.21: re_matchAttr.py

```
#coding : utf-8
import re
s = 'hello jeapedu.com'
regexp = 'jeap'
pat = re.compile(regexp)
matobj = pat.match(s)
if matobj:
    print matobj.group()
matobj = re.match(regexp, s)

s = 'hello jeapedu.com' * 3
regexp = r'(\w+)(\w+)'
pat = re.compile(regexp)
matobj = pat.match(s[6 : ])
if matobj:
    print 'match.group() ->', matobj.group()
else:
    print 'not matched!'
print "matobj.re:", matobj.re
print "matobj.pos:\t", matobj.pos
print "matobj.endpos:\t", matobj.endpos
print "matobj.lastindex\t:", matobj.lastindex
print "matobj.lastgroup\t:", matobj.lastgroup
print "matobj.group(1, 2):", matobj.group(1, 2)
print "matobj.groups():\t", matobj.groups()
print "matobj.groupdict():\t", matobj.groupdict()
print "matobj.start(2):\t", matobj.start(2)
print "matobj.end(2):\t", matobj.end(2)
print "matobj.span(2):\t", matobj.span(2)
```

运行结果如下所示:

```
--------------------------------------------------
match.group() -> jeapedu
matobj.re:  <_sre.SRE_Pattern object at 0x00AC6248>
matobj.pos:          0
matobj.endpos:       45
matobj.lastindex:    2
```

```
matobj.lastgroup:    None
matobj.group(1, 2):  ('jeaped', 'u')
matobj.groups():     ('jeaped', 'u')
matobj.groupdict():  {}
matobj.start(2):     6
matobj.end(2):       7
matobj.span(2):      (6, 7)
----------------------------------------------------
```

13.5 匹配电子邮件地址

最后来个小练习吧：匹配Email地址的正则表达式：

```
res = r'[\w\.-]+@[\w\.-]+\.\w{2,4}'
```

，这里的*表示可以匹配0次及以上, +表示可以匹配1次级以上。

```
1 import re
2 s = "Plz write a email to jeap@jeapedu.com or
     service@jeapedu.com, thanks!"
3 res = r'\w[\w\.-]+@[\w\.-]+\.\w{2,4}'
4 print re.findall(res, s)
```

输出结果如下所示：

```
['jeap@jeapedu.com', 'service@jeapedu.com']
```

当然这个正则还是不太准确，不太符合邮件网站对电子邮件名的规定，但能基本匹配出一般字符串里的邮件地址来，也就够用了。