



# Python基础教程

## 第11章 正则表达式





正则表达式，又称正规表示法、常规表示法（英语：**Regular Expression**，在代码中常简写为**regex**、**regexp**或**RE**），计算机科学的一个概念。

正则表达式使用单个字符串来描述、匹配一系列符合某个句法规则的字符串。

在很多文本编辑器里，正则表达式通常被用来检索、替换符合某个模式的文本。





正则表达式是对字符串操作的一种逻辑公式，用事先定义好的一些特定字符以及这些特定字符的组合，组成一个“规则字符串”，这个“规则字符串”用来表达对字符串的一种过滤逻辑。





- 测试字符串内的模式（数据验证）

例如，可以测试输入字符串，以查看字符串内是否出现电话号码模式或信用卡号码模式。

- 替换文本

可以使用正则表达式来识别文档中的特定文本，完全删除该文本或者用其他文本替换它。

- 基于模式匹配从字符串中提取子字符串

可以查找文档内或输入域内特定的文本。





1. 灵活性、逻辑性和功能性非常的强；
2. 可以迅速地用极简单的方式达到字符串的复杂控制；
3. 对于刚接触的人来说，比较晦涩难懂。

正则表达式主要应用对象是文本，在各种文本编辑器场合都有应用：如编辑器**EditPlus**、**Microsoft Word**、**Visual Studio**等大型编辑器。





假设要提取一段招聘信息里面的联系电话，如：

```
>>> import re
>>> dataString = """以上所有岗位，工作地点：深圳南山区
根据不同级别，薪资水平不同。
货真价实的安全界绝对大牛，薪资可以更高。
以上岗位是业务持续扩展而产生的新领域里面的新岗位，个人在此的发展前景非常大，
欢迎大家投递！
在FeeBuf发表过文章或漏洞盒子白帽子加分。
联系人：徐先生 电话0755-86670332
简历投递邮箱：xuewei@sangfo.com
QQ:2746329146
公司网址：http://www.sangfor.com.cn
"""
```

```
>>> regex = "\d{3,4}-\d{6,8}"
>>> re.findall(regex, dataString)
['0755-86670332']
```





## 11.5 元字符-(1)

字符	说明
<b>\</b>	<p>将下一字符标记为特殊字符、文本、反向引用或八进制转义符。</p> <p>例如，“<b>n</b>”匹配字符“<b>n</b>”。</p> <p>“<b>\n</b>”匹配换行符。</p> <p>序列“<b>\\</b>”匹配“<b>\</b>”，“<b>\(</b>”匹配“<b>(</b>”。</p>
<b>^</b>	<p>匹配输入字符串开始的位置。如果设置了 <b>RegExp</b> 对象的 <b>Multiline</b> 属性，<b>^</b> 还会与“<b>\n</b>”或“<b>\r</b>”之后的位置匹配。</p>
<b>\$</b>	<p>匹配输入字符串结尾的位置。如果设置了 <b>RegExp</b> 对象的 <b>Multiline</b> 属性，<b>\$</b> 还会与“<b>\n</b>”或“<b>\r</b>”之前的位置匹配。</p>
<b>*</b>	<p><b>零次或多次</b>匹配前面的字符或子表达式。</p> <p>例如，<b>zo*</b> 匹配“<b>z</b>”和“<b>zoo</b>”。</p> <p><b>*</b> 等效于 <b>{0,}</b>。</p>





字符	说明
<b>+</b>	<p>一次或多次匹配前面的字符或子表达式。</p> <p>例如，“<b>zo+</b>”与“<b>zo</b>”和“<b>zoo</b>”匹配，但与“<b>z</b>”不匹配。</p> <p><b>+</b> 等效于 <b>{1,}</b>。</p>
<b>?</b>	<p>零次或一次匹配前面的字符或子表达式。</p> <p>例如，“<b>do(es)?</b>”匹配“<b>do</b>”或“<b>does</b>”。</p> <p><b>?</b> 等效于 <b>{0,1}</b>。</p> <p>注意：不能将空格插入逗号和数字之间。</p>







字符

说明

**{n}**

**n** 是非负整数，正好匹配 **n** 次。

例如，“**o{2}**”与“**Bob**”中的“**o**”不匹配，但与“**food**”中的两个“**o**”匹配。

**{n,}**

**n** 是非负整数。至少匹配 **n** 次。

例如，“**o{2,}**”不匹配“**Bob**”中的“**o**”，而匹配“**fooooood**”中的所有 **o**。

“**o{1,}**”等效于“**o+**”。“**o{0,}**”等效于“**o\***”。

**{n,m}**

**m** 和 **n** 是非负整数，其中 **n ≤ m**。匹配至少 **n** 次，至多 **m** 次。

例如，“**o{1,3}**”匹配“**fooooood**”中的头三个 **o**。

“**o{0,1}**”等效于“**o?**”。

注意：不能将空格插入逗号和数字之间。





字符	说明
?	<p>当此字符紧随任何其他限定符（*、+、?、{n}、{n,}、{n,m}）之后时，匹配模式是“非贪心的”。</p> <p>“非贪心的”模式匹配搜索到的、尽可能短的字符串，而默认的“贪心的”模式匹配搜索到的、尽可能长的字符串。</p> <p>例如，在字符串“oooo”中，“o+?”只匹配单个“o”，而“o+”匹配所有“o”。</p>





字符	说明
.	匹配除“\n”之外的任何单个字符。 若要匹配包括“\n”在内的任意字符，请使用诸如“[\s\S]”之类的模式。
(pattern)	匹配 pattern 并捕获该匹配的子表达式。 可以使用 \$0...\$9 属性从结果“匹配”集合中检索捕获的匹配。若要匹配括号字符 (), 请使用 “\(" 或者 “\)”。





字符	说明
<b>(?:pattern)</b>	<p>匹配 <b>pattern</b> 但不捕获该匹配的子表达式。</p> <p>它是一个<b>非</b>捕获匹配，不存储供以后使用的匹配。</p> <p>这对于用 “<b>or</b>” 字符 (<b> </b>) 组合模式部件的情况很有用。</p> <p>例如，'<b>industr(?:y ies)</b>' 是比 '<b>industry industries</b>' 更经济的表达式。</p>
<b>(?=pattern)</b>	<p>执行正向预测先行搜索的子表达式，该表达式匹配处于匹配 <b>pattern</b> 的字符串的起始点的字符串。</p> <p>它是一个非捕获匹配，即不能捕获供以后使用的匹配。</p> <p>例如，'<b>Windows (?:=95 98 NT 2000)</b>' 匹配 “<b>Windows 2000</b>” 中的 “<b>Windows</b>”，但不匹配 “<b>Windows 3.1</b>” 中的 “<b>Windows</b>”。预测先行不占用字符，即发生匹配后，下一匹配的搜索紧随上一匹配之后，而不是在组成预测先行的字符后。</p>





字符	说明
<b>(?!pattern)</b>	<p>执行反向预测先行搜索的子表达式，该表达式匹配不处于匹配 <b>pattern</b> 的字符串的起始点的搜索字符串。</p> <p>它是一个非捕获匹配，即不能捕获供以后使用的匹配。</p> <p>例如，‘<b>Windows (?!95 98 NT 2000)</b>’ 匹配 “<b>Windows 3.1</b>” 中的 “<b>Windows</b>”，但不匹配 “<b>Windows 2000</b>” 中的 “<b>Windows</b>”。</p> <p>预测先行不占用字符，即发生匹配后，下一匹配的搜索紧随上一匹配之后，而不是在组成预测先行的字符后。</p>
<b>x y</b>	<p>匹配 <b>x</b> 或 <b>y</b>。例如，‘<b>z food</b>’ 匹配 “<b>z</b>”或 “<b>food</b>”。‘<b>(z f)ood</b>’ 匹配 “<b>zood</b>” 或 “<b>food</b>”。</p>





字符	说明
<b>[xyz]</b>	字符集。匹配包含的任一字符。 例如，“ <b>[abc]</b> ”匹配“ <b>plain</b> ”中的“ <b>a</b> ”。
<b>[^xyz]</b>	反向字符集。匹配未包含的任何字符。 例如，“ <b>[^abc]</b> ”匹配“ <b>plain</b> ”中的“ <b>p</b> ”。
<b>[a-z]</b>	字符范围。匹配指定范围内的任何字符。 例如，“ <b>[a-z]</b> ”匹配“ <b>a</b> ”到“ <b>z</b> ”范围内的任何小写字母。
<b>[^a-z]</b>	反向范围字符。匹配不在指定的范围内的任何字符。 例如，“ <b>[^a-z]</b> ”匹配任何不在“ <b>a</b> ”到“ <b>z</b> ”范围内的任何字符。





字符	说明
<b>\b</b>	匹配一个字边界，即字与空格间的位置。 例如，“ <b>er\b</b> ”匹配“ <b>never</b> ”中“ <b>er</b> ”，但不匹配“ <b>verb</b> ”中的“ <b>er</b> ”。
<b>\B</b>	非字边界匹配。“ <b>er\B</b> ”匹配“ <b>verb</b> ”中的“ <b>er</b> ”，但不匹配“ <b>never</b> ”中“ <b>er</b> ”。
<b>\cx</b>	匹配 <b>x</b> 指示的控制字符， <b>x</b> 的值必须在 <b>A-Z</b> 或 <b>a-z</b> 之间。 例如， <b>\cM</b> 匹配 <b>Control-M</b> 或 <b>回车符</b> 。 如果不是这样，则假定 <b>c</b> 就是“ <b>c</b> ”字符本身。
<b>\d</b>	数字字符匹配。等效于 <b>[0-9]</b> 。





字符	说明
<b>\D</b>	非数字字符匹配。等效于 <b>[^0-9]</b> 。
<b>\f</b>	换页符匹配。等效于 <b>\x0c</b> 和 <b>\cL</b> 。
<b>\n</b>	换行符匹配。等效于 <b>\x0a</b> 和 <b>\cJ</b> 。
<b>\r</b>	匹配一个回车符。等效于 <b>\x0d</b> 和 <b>\cM</b> 。
<b>\s</b>	匹配任何空白字符，包括空格、制表符、换页符等。 与 <b>[\f\n\r\t\v]</b> 等效。
<b>\S</b>	匹配任何非空白字符。与 <b>[^\f\n\r\t\v]</b> 等效。
<b>\t</b>	制表符匹配。与 <b>\x09</b> 和 <b>\cI</b> 等效。







字符	说明
<b>\v</b>	垂直制表符匹配。与 <b>\x0b</b> 和 <b>\cK</b> 等效。
<b>\w</b>	匹配任何字类字符，包括下划线。 与 “[A-Za-z0-9_]” 等效。
<b>\W</b>	与任何非单词字符匹配。与 “[^A-Za-z0-9_]” 等效。





字符	说明
<b>\xn</b>	<p>匹配 <b>n</b>，此处的 <b>n</b> 是一个十六进制转义码。</p> <p>十六进制转义码必须正好是两位数长。</p> <p>例如，“<b>\x41</b>”匹配“<b>A</b>”。</p> <p>“<b>\x041</b>”与“<b>\x04</b>” &amp; “<b>1</b>”等效。</p> <p>允许在正则表达式中使用 <b>ASCII</b> 代码。</p>
<b>\num</b>	<p>匹配 <b>num</b>，此处的 <b>num</b> 是一个正整数。</p> <p>到捕获匹配的反向引用。</p> <p>例如，“<b>(.)\1</b>”匹配两个连续的相同字符。</p>





字符	说明
<b>\n</b>	标识一个八进制转义码或反向引用。 如果 <b>\n</b> 前面至少有 <b>n</b> 个捕获子表达式, 那么 <b>n</b> 是反向引用。 如果 <b>n</b> 是八进制数 (0-7), 那么 <b>n</b> 是八进制转义码。
<b>\nm</b>	标识一个八进制转义码或反向引用。 如果 <b>\nm</b> 前面至少有 <b>nm</b> 个捕获子表达式, 那么 <b>nm</b> 是反向引用。 如果 <b>\nm</b> 前面至少有 <b>n</b> 个捕获, 则 <b>n</b> 是反向引用, 后面跟有字符 <b>m</b> 。 如果两种前面的情况都不存在, 则 <b>\nm</b> 匹配八进制值 <b>nm</b> , 其中 <b>n</b> 和 <b>m</b> 是八进制数字 (0-7)。
<b>\nml</b>	当 <b>n</b> 是八进制数 (0-3), <b>m</b> 和 <b>l</b> 是八进制数 (0-7) 时, 匹配八进制转义码 <b>nml</b> 。
<b>\un</b>	匹配 <b>n</b> , 其中 <b>n</b> 是以四位十六进制数表示的 <b>Unicode</b> 字符。 例如, <b>\u00A9</b> 匹配版权符号 (©)。





字符	说明
(...)	<p>被括起来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。</p> <p>分组表达式作为一个整体，可以后接数量词。</p> <p>表达式中的 仅在该数组中有效。(abc){2}或a(123 456){2}。</p>
(?P<name>...)	<p>分组，命名一个名字为name的组，匹配规则符合后面的表达式 (... )。 (?P&lt;id&gt;abc){2}</p>
\<number>	<p>引用编号为number的分组匹配到的字符串。 (\d)abc\1</p>
(?P=name)	<p>引用别名为&lt;name&gt;的分组匹配到的字符串。</p> <p>(?P&lt;id&gt;\d)abc(?P=id)</p>





很多不同的语言都实现了正则表达式，**PHP、JavaScript、Perl、Java、Ruby**等等，还有各种程序：**awk、grep**等。

它们的正则表达式语法都是基本一致的，掌握之后，可以直接在其他语言中使用相同的正则语法。

**Python默认正则表达式模块为re模块。**





- **re.RegexObject**

- **re.compile()** 返回 **RegexObject** 对象。

- **re.MatchObject**

- **group()** 返回被 **RE** 匹配的字符串。

- **start()** 返回匹配开始的位置

- **end()** 返回匹配结束的位置

- **span()** 返回一个元组包含匹配 (**开始,结束**) 的位置





## 11.8 MatchObject - (1)

匹配的对象总是有一个布尔型的值**True**，因此可以测试比如使用**match**是否有匹配的值（或者没有匹配，返回**None**）。

**MatchObject**包含的方法：

方法/属性	作用
<b>group()</b>	返回被 <b>RE</b> 匹配的字符串
<b>start()</b>	返回匹配开始的位置
<b>end()</b>	返回匹配结束的位置
<b>span()</b>	返回一个元组包含匹配 (开始,结束) 的位置
<b>groups</b>	返回包含所有子分组的元组





## 11.8 MatchObject - (2)

```
>>> import re
>>> matchObj = re.match("(\w+)(\s+)(\w+)", "Isaac Newton, physicist")
>>> matchObj.group()
'Isaac Newton'
>>> matchObj.group(0)
'Isaac Newton'
>>> matchObj.group(1)
'Isaac'
>>> matchObj.group(2)
''
>>> matchObj.group(3)
'Newton'
>>> matchObj.groups()
('Isaac', ' ', 'Newton')
>>> matchObj.start()
0
>>> matchObj.end()
12
>>> matchObj.span()
(0, 12)
```





## 11.9 RegexpObject - Pattern

**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。

**Pattern**不能直接实例化，必须使用**re.compile()**进行构造。

**Pattern**提供了几个可读属性用于获取表达式的相关信息：

- **pattern**: 编译时用的表达式字符串。
- **flags**: 编译时用的匹配模式，数字形式。
- **groups**: 表达式中分组的数量。
- **groupindex**: 以表达式中有别名的组的别名为键、以该组对应的编号为值的字典，没有别名的组不包含在内。





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

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，取值可以使用按位或运算符'|'表示同时生效，比如**re.I | re.M**。

另外，可以在**regex**字符串中指定模式，比如  
**re.compile('pattern', re.I | re.M)**与**re.compile('(?im)pattern')**是等价的。





## ● *re.compile(strPattern[, flag])*

- **re.I(re.IGNORECASE)**: 忽略大小写（括号内是完整写法，下同）
- **M(MULTILINE)**: 多行模式，改变'^'和'\$'的行为
- **S(DOTALL)**: 点任意匹配模式，改变'.'的行为
- **L(LOCALE)**: 使预定字符类 `\w \W \b \B \s \S` 取决于当前区域设定
- **U(UNICODE)**: 使预定字符类 `\w \W \b \B \s \S \d \D` 取决于unicode定义的字符属性
- **X(VERBOSE)**: 详细模式。这个模式下正则表达式可以是多行，忽略空白字符，并可以加入注释。



```
import re
pattern = re.compile(r'\d+') # 用于匹配至少一个数字
m = pattern.match('one12twothree34four') # 查找头部, 没有匹配
print(m)

m = pattern.match('one12twothree34four', 2, 10) # 从'e'的位置开始匹配, 没有匹配
print(m)

m = pattern.match('one12twothree34four', 3, 10) # 从'1'的位置开始匹配, 正好匹配
print(m) # 返回一个 Match 对象
```

```
print(m.group(0)) # 可省略 0
```

```
print(m.start(0)) # 可省略 0
```

```
print(m.end(0)) # 可省略 0
```

```
print(m.span(0)) # 可省略 0
```

Python 3.7.4 Shell

File Edit Shell Debug Options Window Help

Python 3.7.4 (tags/v3.7.4:e09359112e, Jul 8 2019, 19:29:22)

n32

Type "help", "copyright", "credits" or "license()" for more i

>>>

===== RESTART: D:\桌面\2018-12-

=====

None

None

<re.Match object; span=(3, 5), match='12'>

12

3

5

(3, 5)



# 11.9 RegexObject - Pattern

```
import re
```

```
p = re.compile(r'(\w+) (\w+)(?P<sign>.*)', re.DOTALL)
```

```
print("p.pattern:", p.pattern)
```

```
print("p.flags:", p.flags)
```

```
print("p.groups:", p.groups)
```

```
print("p.groupindex:", p.groupindex)
```

Python 3.7.4 Shell

File Edit Shell Debug Options Window Help

Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  
n32

Type "help", "copyright", "credits" or "li  
>>>

===== RESTART:

=====

p.pattern: (\w+) (\w+)(?P<sign>.\*)

p.flags: 48

p.groups: 3

p.groupindex: {'sign': 3}





- *re.search(pattern, string, flags=0)*

**Scan through string** looking for a location where the regular expression pattern produces a match, and return a corresponding **MatchObject** instance. Return **None** if no position in the string matches the pattern; note that this is different from finding a zero-length match at some point in the string.

```
>>> import re
>>> test = 'abcdef'
>>> matchobj = re.search('c',test)
>>> matchobj
<re.Match object; span=(2, 3), match='c'>
```





- *re.match(pattern, string, flags=0)*

If zero or more characters at the beginning of string match the regular expression pattern, return a corresponding **MatchObject** instance. Return **None** if the string does not match the pattern; note that this is different from a zero-length match.

```
>>> import re
>>> test = 'abcdef'
>>> matchobj = re.search('c',test)
>>> matchobj
<re.Match object; span=(2, 3), match='c'>
>>> matchobj = re.match('c',test)
>>> matchobj
>>> matchobj = re.match('a',test)
>>> matchobj
<re.Match object; span=(0, 1), match='a'>
```



```
import re
m = re.match(r'(\w+) (\w+)(?P<sign>.*)', 'hello world!')

print("m.string:", m.string)
print("m.re:", m.re)
print("m.pos:", m.pos)
print("m.endpos:", m.endpos)
print("m.lastindex:", m.lastindex)
print("m.lastgroup:", m.lastgroup)

print("m.group(1,2):", m.group(1, 2))
print("m.groups():", m.groups())
print("m.groupdict():", m.groupdict())
print("m.start(2):", m.start(2))
print("m.end(2):", m.end(2))
print("m.span(2):", m.span(2))
print("m.expand(r'\2 \1\3'):", m.expand(r'\2 \1\3'))
```

Python 3.7.4 Shell

File Edit Shell Debug Options Window Help

Python 3.7.4 (tags/v3.7.4:e09359112e, Jul 8 2019)  
n32

Type "help", "copyright", "credits" or "license()"

>>>

===== RESTART: D:\桌

=====

m.string: hello world!

m.re: re.compile('(\w+) (\w+)(?P<sign>.\*)')

m.pos: 0

m.endpos: 12

m.lastindex: 3

m.lastgroup: sign

m.group(1,2): ('hello', 'world')

m.groups(): ('hello', 'world', '!')

m.groupdict(): {'sign': '!'}

m.start(2): 6

m.end(2): 11

m.span(2): (6, 11)

m.expand(r'\2 \1\3'): world hello!







## 11.12 search与match的区别

这两个函数在正则表达式中经常使用，但是最容易被混淆，如Python说明文档中说的：

Python offers two different primitive operations based on regular expressions: **match** checks for a match **only at the beginning of the string**, while **search** checks for a match **anywhere in the string** (this is what Perl does by default).

**match**只在开头检查是否匹配；

**search**会扫描整个字符串检查匹配。





*re.split(pattern, string, maxsplit=0, flags=0)*

使用正则表达式分割字符串，返回值是使用正则表达式分割之后的字符串列表。

如果**捕获括号**在 **RE** 中使用，那么它们的内容也会作为结果列表的一部分返回。

如果 **maxsplit** 非零，那么最多只能分出 **maxsplit** 个分片。





```
import re
print (re.split('\W+', 'Words,words,words.'))
print (re.split('(\W+)', 'Words,words,words.'))
print (re.split('\W+', 'Words,words,words.', 1))
print (re.split('[a-f]+', '0a3B9', flags = re.IGNORECASE))
```

Python 3.7.4 Shell

File Edit Shell Debug Options Window Help

Python 3.7.4 (tags/v3.7.4:e09359112e, Jul 8 2019, 19:29:22)

n32

Type "help", "copyright", "credits" or "license()" for more

>>>

===== RESTART: D:\桌面\2018-1

=====

['Words', 'words', 'words', '']

['Words', ',', 'words', ',', 'words', '.', '']

['Words', 'words,words.']

['0', '3', '9']





- *findall(string[, pos[, endpos]]) / re.findall(pattern, string[, flags])*

找到 **RE** 匹配的所有子串，并把它们作为一个列表返回。

从左向右扫描字符串，并把匹配的结果按找到的顺序存入列表；

如果正则表达式包含一个或者多个分组，那么将返回一个分组

列表，此时如果有多个分组，那么它是一个元组的列表。

```
>>> import re
>>> re.findall('\d+', 'abc12de34fg5 6')
['12', '34', '5', '6']
>>> re.findall('(\d+)(\w{2})', 'abc12de34fg5 6')
[('12', 'de'), ('34', 'fg')]
```





- *finditer(string[, pos[, endpos]]) / re.finditer(pattern, string[, flags])*

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。 分组列表，此时如果有多个分组，那么它是一个元组的列表。

```
>>> import re
>>> p = re.compile(r'\d+')
>>> for m in p.finditer('one1two2three3four4'):
    print(m.group())
```

1  
2  
3  
4





使用正则表达式进行字符串替换。找到 **RE** 匹配的所有子串，并将其用一个不同的字符串替换。

*re.sub(pattern, repl, string, count=0, flags=0)*

返回的字符串是在字符串中用 **RE** 最左边不重复的匹配来替换。如果模式没有发现，字符将被没有改变地返回。

可选参数 **count** 是模式匹配后替换的最大次数；**count** 必须是非负整数。缺省值是 **0** 表示替换所有的匹配。

```
>>> re.sub('\d+', '-Numbers-', 'abc12de34fg56')  
'abc-Numbers-de-Numbers-fg-Numbers-'
```





*subn(repl, string[, count]) / re.sub(pattern, repl, string[, count])*

返回 (**sub(repl, string[, count])**, 替换次数)。

```
import re

p = re.compile(r'(\w+) (\w+)')
s = 'i say, hello world!'

print(p.subn(r'\2 \1', s))

def func(m):
    return m.group(1).title() + ' ' + m.group(2).title()

print(p.subn(func, s))
```

-----  
(**'say i, world hello!', 2**)  
(**'I Say, Hello World!', 2**)  
\\





测试文本：

“以上所有岗位，工作地点：深圳南山区

根据不同级别，薪资水平不同。货真价实的安全界绝对大牛，

薪资可以更高。以上岗位是业务持续扩展而产生的新领域

里面的新岗位，个人在此的发展前景非常大，

欢迎大家投递！在FreeBuf发表过文章或漏洞盒子白帽子加分。

联系人：徐先生 电话0755-86670332

联系人：张小姐 电话0755-86670379

简历投递邮箱：xuwei@sangfor.com

QQ：2746329146

公司网址：<http://www.sangfor.com.cn>”





#提取电话号码

# 0 - 9 的个数为3到4个，加上“-”，后面再接上6到8位 0 - 9 数字

```
print("-----\n1.")
```

```
regex = '[0-9]{3,4}-[0-9]{6,8}'
```

```
tel = re.findall(regex, dataString)
```

```
print('\n'.join(tel).decode('utf8'))
```

#或者：使用 \d 来表示数字

```
print("-----\n2.")
```

```
regex = '\d{3,4}-\d{6,8}'
```

```
tel = re.findall(regex, dataString)
```

```
print('\n'.join(tel).decode('utf8'))
```

#提取网址 - 转义操作

```
print("-----\n3.")
```

```
regex = "http://[\\.\w]+"
```

```
site = re.findall(regex, dataString)
```

```
print("\n".join(site).decode("utf8"))
```

#联系人 - 可进行中文正则

```
print("-----\n3.")
```

```
regex = "联系人: .+"
```

```
commun= re.findall(regex, dataString)
```

```
print("\n".join(commun).decode("utf8"))
```





四川大學  
SICHUAN UNIVERSITY

熟练掌握正则表达式的用法！

