Python程序设计

第五讲 批量数据类型与操作 正则表达式



张华 WHU

正则表达式是什么

■引例

- * 字符串是编程时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。

正则表达式是什么

- 正则表达式(Regular Expression)是一种用来匹配字符串的强有力的工具。
 - **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,就认为它"匹配"了,否则,该字符串就是不合法的。
 - *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。



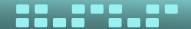
- 因为正则表达式也是用字符串表示的,所以,要首先了解如何用字符来描述字符。
- * 在正则表达式中,如果直接给出字符,就是精确匹配。
- *用\d可以匹配一个数字,\w可以匹配一个字母或数字,所以:
 - ▶'00\d'可以匹配'007', 但无法匹配'00A'
 - ▶ '\d\d\d'可以匹配'010'
 - ▶ '\w\w\d'可以匹配'py3'
- * 可以匹配任意字符,所以:
 - ▶'py.'可以匹配'pyc'、'pyo'、'py!'等等



- * 要匹配变长的字符, 在正则表达式中,
 - ▶用*表示任意个字符(包括0个),
 - ▶用+表示至少一个字符,
 - ▶用?表示0个或1个字符,
 - ▶用{n}表示n个字符,
 - ▶用{n,m}表示n-m个字符。



- *来看一个复杂的例子: \d{3}\s+\d{3,8}。
 - ▶\d{3}表示匹配3个数字,例如'010';
 - ▶\s可以匹配一个空格(也包括Tab等空白符),所以\s+表示至少有一个空格,例如匹配'', ''等;
 - ▶\d{3,8}表示3-8个数字,例如'1234567'。
- *综合起来,上面的正则表达式可以匹配以任意个空格隔开的带区号的电话号码。



■ 创建正则表达式的转义字符

- 如果要匹配'010-12345'这样的号码呢?
 - ▶由于'-'是特殊字符,在正则表达式中,要用'\'转义,
 - ▶所以,上面的正则是\d{3}\-\d{3,8}。
- *但是,仍然无法匹配'010 12345'。因为带有空格,所以 需要更复杂的匹配方式。



- *要做更精确地匹配,可以用[]表示范围,比如:
 - ▶[0-9a-zA-Z_]可以匹配一个数字、字母或者下划线;
 - ▶[0-9a-zA-Z_]+可以匹配至少由一个数字、字母或者下划线组成的字符串,比如'a100', '0_Z', 'Py3000'等等;
 - ▶ [a-zA-Z_][0-9a-zA-Z_]*可以匹配由字母或下划线开头,后接任意 个由一个数字、字母或者下划线组成的字符串,也就是Python合 法的标识符;
 - ▶[a-zA-Z_][0-9a-zA-Z_]{0, 19}更精确地限制了变量的长度是1-20 个字符(前面1个字符+后面最多19个字符)。



- * A|B可以匹配A或B,所以(P|p)ython可以匹配'Python'或者'python'。
- * ^表示行的开头, ^\d表示必须以数字开头。
- *\$表示行的结束,\d\$表示必须以数字结束。
- *py也可以匹配'python',但是加上^py\$就变成了整行匹配,就只能匹配'py'了。



■ 经典正则表达式

$$[1-9]\d{5}$$

$$\d{3}-\d{8}\|\d{4}-\d{7}$$

由26个字母组成的字符串

由26个字母和数字组成的字符串

整数形式的字符串

正整数形式的字符串

中国境内邮政编码,6位

匹配中文字符

国内电话号码,010-68913536

■ 经典正则表达式

匹配IP地址的正则表达式

IP地址字符串形式的正则表达式(IP地址分4段,每段0-255)

\d+.\d+.\d+.\d+ 或 \d{1,3}.\d{1,3}.\d{1,3}.\d{1,3}

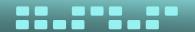
精确写法 0-99: [1-9]?\d

100-199: 1\d{2}

200-249: 2[0-4]\d

250-255: 25[0-5]

 $(([1-9]?\d|1\d\{2\}|2[0-4]\d|25[0-5]).)\{3\}([1-9]?\d|1\d\{2\}|2[0-4]\d|25[0-5])$



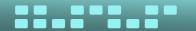
re库

- Python的re标准库包含所有的正则表达式功能,主要用于字符串匹配。
 - * 先看看如何判断正则表达式是否匹配:

```
>>> import re
>>> re.match(r'^\d{3}\-\d{3,8}$', '010-12345')
<_sre.SRE_Match object; span=(0, 9), match='010-12345'>
>>> re.match(r'^\d{3}\-\d{3,8}$', '010 12345')
```

* match()方法判断是否匹配,如果匹配成功,返回一个 Match对象,否则返回None。常见的判断方法就是:

```
test = '用户输入的字符串'
if re.match(r'正则表达式', test):
    print('ok')
else:
    print('failed')
```



■ 切分字符串

*用正则表达式切分字符串比用固定的字符更灵活:

```
>>> 'a b c'.split(' ')
['a', 'b', '', 'c']
```

```
>>> re.split(r'\s+', 'a b c')
['a', 'b', 'c']
```

▶加入,和;

```
>>> re.split(r'[\s\,\;]+', 'a,b;; c d')
['a', 'b', 'c', 'd']
```

■ 查找字符串

*用正则表达式在目标字符串中查找所有符合规则的字符串。 匹配成功,就返回一个列表;如果没有符合规则的字符串, 就返回None。

```
>>> r = re.findall('as\d+', 'as1,as2,as3,as11,as12')
>>> r
['as1', 'as2', 'as3', 'as11', 'as12']
```

■分组

- *除了简单地判断是否匹配之外,正则表达式还有提取子串的强大功能。用()表示的就是要提取的分组(Group)。
- * 比如: ^(\d{3})-(\d{3,8})\$分别定义了两个组,可以直接从 匹配的字符串中提取出区号和本地号码。

```
>>> m = re.match(r'^(\d{3})-(\d{3,8})$', '010-12345')
>>> m
<_sre.SRE_Match object; span=(0, 9), match='010-12345'>
>>> m.group(0)
'010-12345'
>>> m.group(1)
'010'
>>> m.group(2)
'12345'
```

■贪婪匹配

- * 正则匹配默认是贪婪匹配,也就是匹配尽可能多的字符。
- *举例, 匹配出数字后面的**0**:

```
>>> re.match(r'^(\d+)(0*)$', '102300').groups() ('102300', '')
```

- ▶由于\d+采用贪婪匹配,直接把后面的0全部匹配了,结果0*只能匹配空字符串了。
- ▶必须让\d+采用非贪婪匹配(也就是尽可能少匹配),才能把后面的0匹配出来,加个?就可以让\d+采用非贪婪匹配:

```
>>> re.match(r'^(\d+?)(0*)$', '102300').groups()
('1023', '00')
```



■编译

◆如果一个正则表达式要重复使用几千次,出于效率的考虑,可以预编译该正则表达式,接下来重复使用时就不需要编译这个步骤了,直接匹配:

```
>>> import re
# 编译:
>>> re_telephone = re.compile(r'^(\d{3})-(\d{3,8})$')
# 使用:
>>> re_telephone.match('010-12345').groups()
('010', '12345')
>>> re_telephone.match('010-8086').groups()
('010', '8086')
```

▶编译后生成正则表达式对象,由于该对象自己包含了正则表达式, 所以调用对应的方法时不用给出正则字符串。