正则表达式

最简单的正则表达式

```
In : import re

In : re.match('test', 'test')
Out: <_sre.SRE_Match object; span=(0, 4), match='test'>

In : re.match('test', 'test123')
Out: <_sre.SRE_Match object; span=(0, 4), match='test'>

In : re.match('test', 'tes')
```

```
match(pattern, string, flags=0)  # re.match的函数签名
```

```
In : re.search('test', 'test123')
Out: <_sre.SRE_Match object; span=(0, 4), match='test'>
```

search/match的区别

```
In : re.search('test', '1test')
Out: <_sre.SRE_Match object; span=(1, 5), match='test'>

In : re.match('test', '1test')
```

```
search(pattern, string, flags=0)  # re.search的函数签名
```

元字符(metacharacters)

. 小圆点符号表示匹配除换行符以外的任意字符
\w 匹配字母或数字或下划线或汉字
\s 匹配任意的空白符，包括空格制表符换页符等
\d 匹配数字
\b 匹配单词的开始或结束

```
In : re.search('..', 'hi')
Out: <_sre.SRE_Match object; span=(0, 2), match='hi'>

In : re.search('\w\w\w\w', 'a1_哈')
Out: <_sre.SRE_Match object; span=(0, 4), match='a1_哈'>

In : re.search('\s\s\s', ' \t\r')
Out: <_sre.SRE_Match object; span=(0, 3), match=' \t\r'>

In : re.search('\d\d', '12')
Out: <_sre.SRE_Match object; span=(0, 2), match='12'>
```

匹配内容

```
In : match = re.search('\d', 'test1test')

In : match.group()
Out: '1'
```

元字符(二)

^ 匹配字符串的开始
$ 匹配字符串的结束

```
In : re.search('^test', 'test123')
Out: <_sre.SRE_Match object; span=(0, 4), match='test'>

In : re.search('^test', '123test123')

In : re.search('123$', 'test123')
Out: <_sre.SRE_Match object; span=(4, 7), match='123'>

In : re.search('123$', 'test123t')
```

元字符(三)

x|y 匹配 x 或 y
[xyz] 字符集合。匹配所包含的任意一个字符
[a-z] 字符范围。匹配指定范围内的任意字符

```
In : re.search('a|e', 'abcd')
Out: <_sre.SRE_Match object; span=(0, 1), match='a'>

In : re.search('[a12]', 'abcd')
Out: <_sre.SRE_Match object; span=(0, 1), match='a'>

In : re.search('[a-z]', 'abc')
Out: <_sre.SRE_Match object; span=(0, 1), match='a'>

In : re.search('[d-z]', 'abc')
```

重复

```
In : re.search('\d\d\d\d\d', '12345')
Out: <_sre.SRE_Match object; span=(0, 5), match='12345'>
```

```
? 匹配前面的子表达式零次或一次
+ 匹配前面的子表达式一次或多次
* 匹配前面的子表达式零次或多次
{n} 重复n次
{n,}    重复n次或更多次
{n,m}   重复n到m次
```

```
In : re.search('\d{5}', '12345')
Out: <_sre.SRE_Match object; span=(0, 5), match='12345'>

In : re.search('ca*t', 'cat')
Out: <_sre.SRE_Match object; span=(0, 3), match='cat'>

In : re.search('ca*t', 'cart')

In : re.search('ca*t', 'caat')
Out: <_sre.SRE_Match object; span=(0, 4), match='caat'>
```

反义

```
[^x]  匹配除了x以外的任意字符
[^abc]  匹配除了abc这几个字母以外的任意字符
\W  匹配任意不是字母，数字，下划线，汉字的字符。等价于 '[^A-Za-z0-9_]'
\S  匹配任意不是空白符的字符 等价于 [^ \f\n\r\t\v]
\D  匹配任意非数字的字符 [^0-9]
\B  匹配不是单词开头或结束的位置
```

贪婪与懒惰

```
In : re.search('a.*b', 'aabab').group()
Out: 'aabab'
```

懒惰限定符：

```
*?   重复任意次，但尽可能少重复
+?   重复1次或更多次，但尽可能少重复
??   重复0次或1次，但尽可能少重复
{n,m}?   重复n到m次，但尽可能少重复
{n,}?    重复n次以上，但尽可能少重复
```

```
In : re.search('a.*?b', 'aabab').group()
Out: 'aab'
```

编译标志

```
In : re.search('a.*b', 'aabab').group()
Out: 'aabab'
```

```
DOTALL, S    使 . 匹配包括换行在内的所有字符
IGNORECASE, I    使匹配对大小写不敏感
LOCALE, L    做本地化识别匹配
MULTILINE, M     多行匹配，影响 ^ 和 $
VERBOSE, X   详细状态
DEBUG 调试模式
```

```
In : re.search('.', '\n')

In : re.search('.', '\n', re.S)
Out: <_sre.SRE_Match object; span=(0, 1), match='\n'>

In : re.search('a.', 'A\n', re.S|re.I)
Out: <_sre.SRE_Match object; span=(0, 2), match='A\n'>
```

编译正则表达式

```
In : regex = re.compile(r'^\d{1,3}$')

In : regex.match('12')
Out: <_sre.SRE_Match object; span=(0, 2), match='12'>

In : regex.match('1234')
```

检索和替换

```
re.sub(pattern, repl, string, count=0, flags=0)  # re.sub签名
```

```
In : re.sub('\d+', '', 'test123')
Out: 'test'

In : re.sub('\d', '', 'test123')
Out: 'test'

In : re.sub('\d', '', 'test123', count=2)
Out: 'test3'
```

findall/finditer

```
In : re.findall('\d', '1a2b3c4d')
Out: ['1', '2', '3', '4']

In : for i in re.finditer('\d', '1a2b3c4d'):
...:     print(i)
...:
<_sre.SRE_Match object; span=(0, 1), match='1'>
<_sre.SRE_Match object; span=(2, 3), match='2'>
<_sre.SRE_Match object; span=(4, 5), match='3'>
<_sre.SRE_Match object; span=(6, 7), match='4'>
```

分组

```
In : m = re.match('(a)b', 'ab')

In : m.group(1)
Out: 'a'

In : m = re.match('([a-c]+).*(\w)', 'abcbde')

In : m.groups()
Out: ('abcb', 'e')

In : m.group(1), m.group(2), m.group(1, 2)
Out: ('abcb', 'e', ('abcb', 'e'))
```

命名分组

```
(?P<name>正则表达式) # 命名分组格式
```

```
In : pattern = '(?P<year>\d{4})-(?P<month>\d{2})-(?P<day>\d{2})'

In : m = re.match(pattern, '2018-03-23')

In : m.groupdict()
Out: {'day': '23', 'month': '03', 'year': '2018'}

In : m.group('year')
Out: '2018'
```

延伸阅读

1.http://deerchao.net/tutorials/regex/regex.htm
2.http://wiki.ubuntu.org.cn/Python%E6%AD%A3%E5%88%99%E8%A1%A8%E8%BE%BE%E5%BC%8F%E6%93%8D%E4%BD%9C%E6%8C%87%E5%8D%97