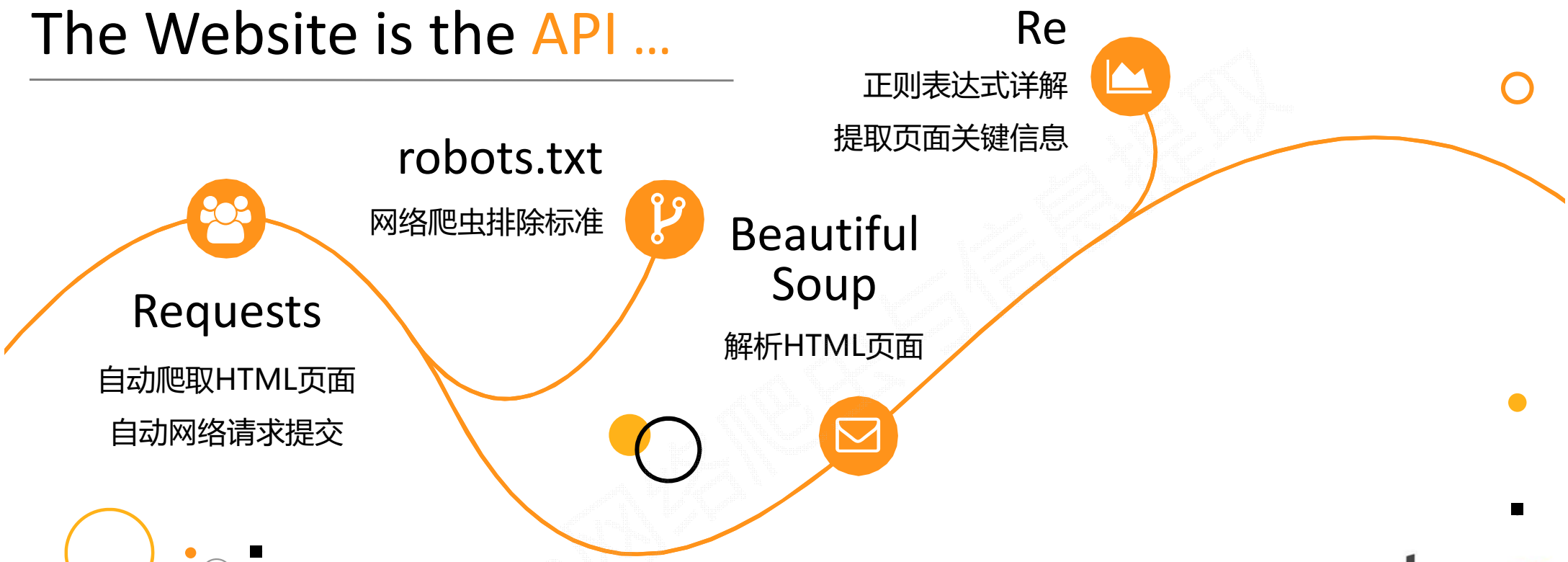
Re(正则表达式)库入门

WS07

嵩天

www.python123.org

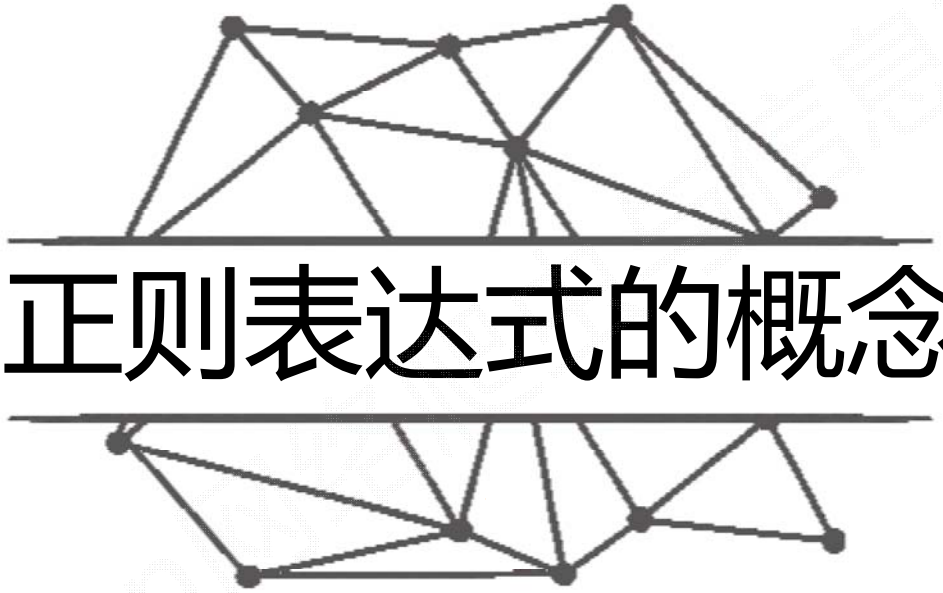


掌握定向网络数据爬取和网页解析的基本能力

python
弹指之间 · 享受创新

Python网络爬虫与信息提取

O4X -Tian



正则表达式的概念

正则表达式

regular expression, regex, RE

正则表达式是用来简洁表达一组字符串的表达式

```
'PN'
'PYN'
'PYTN'
'PYTHN'
'PYTHON'
```

正则表达式：

```
P(Y|YT|YTH|YTHO)?N
```

一组字符串

简洁表达

正则表达式

regular expression, regex, RE

使用正则表达式的优势是什么？

简洁

一行胜千言　　一行就是特征(模式)

正则表达式

regular expression, regex, RE

'PY'

'PYY'

'PYYY'

'PYYYY'

……

'PYYYY……'

正则表达式：

PY+

一组字符串（无穷个）

无穷字符串组的简洁表达

正则表达式

regular expression, regex, RE

'PY'开头

后续存在不多于10个字符

后续字符不能是'P'或'Y'

'PYABC' √

'PYKXYZ' X

正则表达式：

PY[^PY]{0,10}

一组具有某些特点的字符串

（可以枚举，但很繁琐）

某种特征字符串组的简洁表达

正则表达式

regular expression, regex, RE

正则表达式是用来简洁表达一组字符串的表达式

正则表达式是一种通用的字符串表达框架

进一步

正则表达式是一种针对字符串表达“简洁”和“特征”思想的工具

正则表达式可以用来判断某字符串的特征归属

正则表达式

regular expression, regex, RE

正则表达式在文本处理中十分常用：

表达文本类型的特征（病毒、入侵等）

同时查找或替换一组字符串

匹配字符串的全部或部分　　最主要应用在字符串匹配中

……

正则表达式的使用

正则表达式：

P(Y|YT|YTH|YTHO)?N

regex='P(Y|YT|YTH|YTHO)?N'

'PN'

'PYN'

'PYTN'

'PYTHN'

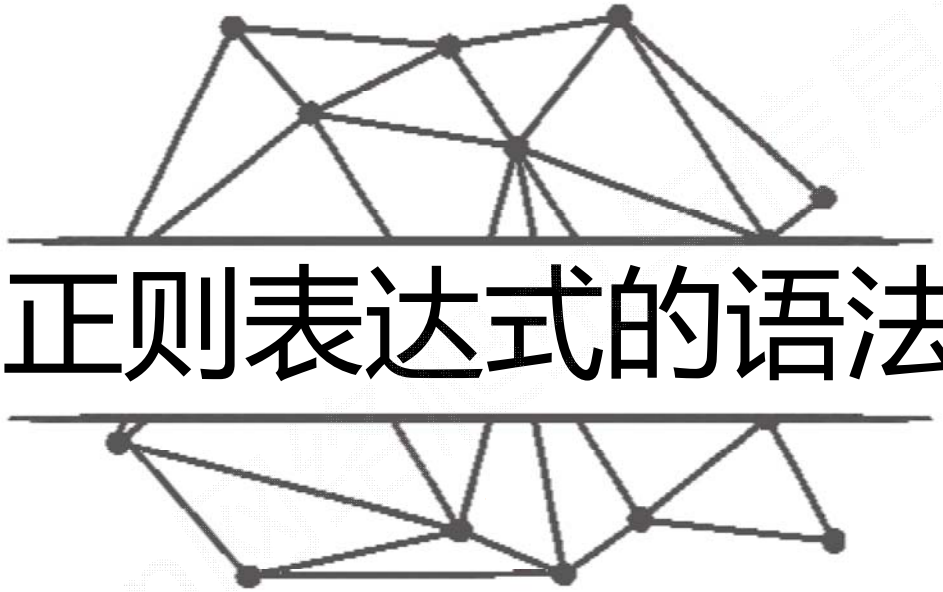
'PYTHON'

编译

p=re.compile(regex)

特征（p）

编译：将符合正则表达式语法的字符串转换成正则表达式特征



正则表达式的语法

正则表达式的语法

P(Y|YT|YTH|YTHO)?N

正则表达式语法由字符和操作符构成

正则表达式的常用操作符(1)

操作符	说明	实例
.	表示任何单个字符	
[ ]	字符集，对单个字符给出取值范围	[abc]表示a、b、c，[a-z]表示a到z单个字符
[^]	非字符集，对单个字符给出排除范围	[^abc]表示非a或b或c的单个字符
*	前一个字符0次或无限次扩展	abc* 表示 ab、abc、abcc、abccc等
+	前一个字符1次或无限次扩展	abc+ 表示 abc、abcc、abccc等
?	前一个字符0次或1次扩展	abc? 表示 ab、abc
|	左右表达式任意一个	abc|def 表示 abc、def

正则表达式的常用操作符(2)

操作符	说明	实例
{m}	扩展前一个字符m次	ab{2}c表示abbc
{m,n}	扩展前一个字符m至n次（含n）	ab{1,2}c表示abc、abbc
^	匹配字符串开头	^abc表示abc且在一个字符串的开头
$	匹配字符串结尾	abc$表示abc且在一个字符串的结尾
( )	分组标记，内部只能使用 | 操作符	(abc)表示abc，(abc|def)表示abc、def
\d	数字，等价于[0-9]	
\w	单词字符，等价于[A-Za-z0-9_]	

正则表达式语法实例

正则表达式	对应字符串
P(Y|YT|YTH|YTHO)?N	'PN'、'PYN'、'PYTN'、'PYTHN'、'PYTHON'
PYTHON+	'PYTHON'、'PYTHONN'、'PYTHONNN' …
PY[TH]ON	'PYTON'、'PYHON'
PY[^TH]?ON	'PYON'、'PYaON'、'PYbON'、'PYcON'…
PY{:3}N	'PN'、'PYN'、'PYYN'、'PYYYN'…

经典正则表达式实例

`^[A-Za-z]+$`	由26个字母组成的字符串
`^[A-Za-z0-9]+$`	由26个字母和数字组成的字符串
`^-?\d+$`	整数形式的字符串
`^[0-9]*[1-9][0-9]*$`	正整数形式的字符串
`[1-9]\d{5}`	中国境内邮政编码，6位
`[\u4e00-\u9fa5]`	匹配中文字符
`\d{3}-\d{8}|\d{4}-\d{7}`	国内电话号码，010-68913536

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

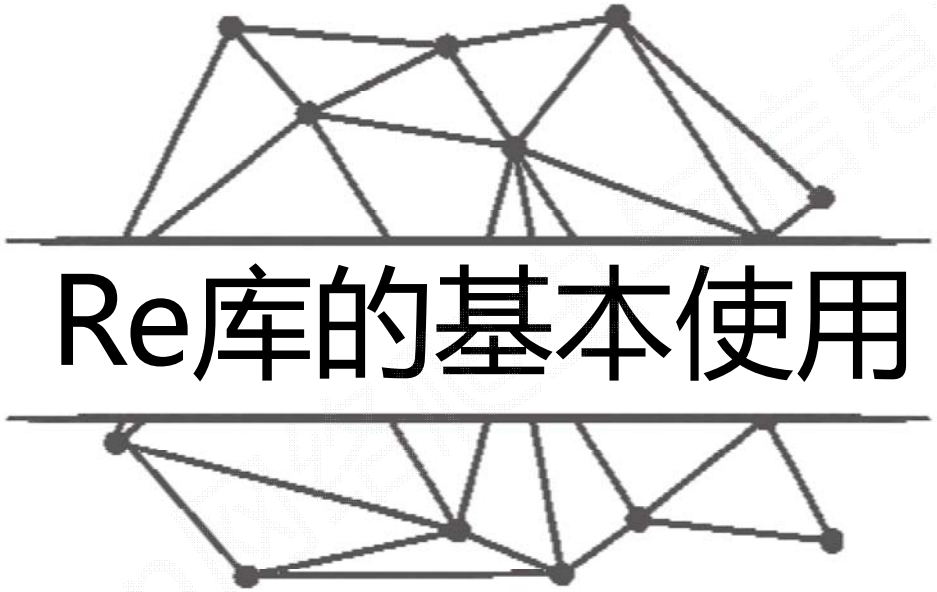
0-99： `[1-9]?\d`

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库的基本使用

Re库介绍

Re库是Python的标准库，主要用于字符串匹配

调用方式：

```
import re
```

Python Library

正则表达式的表示类型

raw string类型（原生字符串类型）

re库采用raw string类型表示正则表达式，表示为：

```
r'text'
```

例如：

```
r'[1-9]\d{5}'
r'\d{3}-\d{8}|\d{4}-\d{7}'
```

raw string是不包含对转义符再次转义的字符串

正则表达式的表示类型

re库也可以采用string类型表示正则表达式，但更繁琐

例如：

```
'[1-9]\\d{5}'
'\\d{3}-\\d{8}|\\d{4}-\\d{7}'
```

建议：当正则表达式包含转义符时，使用raw string

Re库主要功能函数

函数	说明
re.search()	在一个字符串中搜索匹配正则表达式的第一个位置，返回match对象
re.match()	从一个字符串的开始位置起匹配正则表达式，返回match对象
re.findall()	搜索字符串，以列表类型返回全部能匹配的子串
re.split()	将一个字符串按照正则表达式匹配结果进行分割，返回列表类型
re.finditer()	搜索字符串，返回一个匹配结果的迭代类型，每个迭代元素是match对象
re.sub()	在一个字符串中替换所有匹配正则表达式的子串，返回替换后的字符串

```
re.search(pattern, string, flags=0)
```

在一个字符串中搜索匹配正则表达式的第一个位置
返回match对象

- `pattern` ： 正则表达式的字符串或原生字符串表示
- `string` ： 待匹配字符串
- `flags` ： 正则表达式使用时的控制标记

```
re.search(pattern, string, flags=0)
```

- **flags** ：正则表达式使用时的控制标记

常用标记	说明
re.I re.IGNORECASE	忽略正则表达式的大小写，[A-Z]能够匹配小写字符
re.M re.MULTILINE	正则表达式中的^操作符能够将给定字符串的每行当作匹配开始
re.S re.DOTALL	正则表达式中的.操作符能够匹配所有字符，默认匹配除换行外的所有字符

```
re.search(pattern, string, flags=0)
```

```
>>> import re
>>> match = re.search(r'[1-9]\d{5}', 'BIT 100081')
>>> if match:
        print(match.group(0))


100081
>>>
```

Python编程快速上手

```
re.match(pattern, string, flags=0)
```

从一个字符串的开始位置起匹配正则表达式
返回match对象

- `pattern` ： 正则表达式的字符串或原生字符串表示
- `string` ： 待匹配字符串
- `flags` ： 正则表达式使用时的控制标记

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

```
>>> import re
>>> match = re.match(r'[1-9]\d{5}', 'BIT 100081')
>>> if match:
        match.group(0)


>>> match.group(0)
Traceback (most recent call last):
  File "<pyshell#14>", line 1, in <module>
    match.group(0)
AttributeError: 'NoneType' object has no attribute 'group'
>>> match = re.match(r'[1-9]\d{5}', '100081 BIT')
>>> if match:
        match.group(0)


'100081'
>>>
```

```
re.findall(pattern, string, flags=0)
```

搜索字符串，以列表类型返回全部能匹配的子串

- `pattern` ： 正则表达式的字符串或原生字符串表示
- `string` ： 待匹配字符串
- `flags` ： 正则表达式使用时的控制标记

```
re.findall(pattern, string, flags=0)
```

```
>>> import re
>>> ls = re.findall(r'[1-9]\d{5}', 'BIT100081 TSU100084')
>>> ls
['100081', '100084']
>>>
```

Python语言程序设计

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

将一个字符串按照正则表达式匹配结果进行分割
返回列表类型

- `pattern` ： 正则表达式的字符串或原生字符串表示
- `string` ： 待匹配字符串
- `maxsplit`: 最大分割数，剩余部分作为最后一个元素输出
- `flags` ： 正则表达式使用时的控制标记

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

```
>>> import re
>>> re.split(r'[1-9]\d{5}', 'BIT100081 TSU100084')
['BIT', ' TSU', '']
>>> re.split(r'[1-9]\d{5}', 'BIT100081 TSU100084', maxsplit=1)
['BIT', ' TSU100084']
>>>
```

Python编程从入门到实践

```
re.finditer(pattern, string, flags=0)
```

搜索字符串，返回一个匹配结果的迭代类型，每个迭代元素是match对象

- `pattern` ： 正则表达式的字符串或原生字符串表示
- `string` ： 待匹配字符串
- `flags` ： 正则表达式使用时的控制标记

re.finditer(pattern, string, flags=0)

```
>>> import re
>>> for m in re.finditer(r'[1-9]\d{5}', 'BIT100081 TSU100084'):
        if m:
            print(m.group(0))


100081
100084
>>>
```

Python语言程序设计

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

在一个字符串中替换所有匹配正则表达式的子串
返回替换后的字符串

- `pattern` ： 正则表达式的字符串或原生字符串表示
- `repl` ： 替换匹配字符串的字符串
- `string` ： 待匹配字符串
- `count` ： 匹配的最大替换次数
- `flags` ： 正则表达式使用时的控制标记

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

```
>>> import re
>>> re.sub(r'[1-9]\d{5}', ':zipcode', 'BIT100081 TSU100084')
'BIT:zipcode TSU:zipcode'
>>>
```

Python语言程序设计

Re库主要功能函数

函数	说明
re.search()	在一个字符串中搜索匹配正则表达式的第一个位置，返回match对象
re.match()	从一个字符串的开始位置起匹配正则表达式，返回match对象
re.findall()	搜索字符串，以列表类型返回全部能匹配的子串
re.split()	将一个字符串按照正则表达式匹配结果进行分割，返回列表类型
re.finditer()	搜索字符串，返回一个匹配结果的迭代类型，每个迭代元素是match对象
re.sub()	在一个字符串中替换所有匹配正则表达式的子串，返回替换后的字符串

Re库的另一种等价用法

```
>>> rst = re.search(r'[1-9]\d{5}', 'BIT 100081')
```

函数式用法：一次性操作

```
>>> pat = re.compile(r'[1-9]\d{5}')
>>> rst = pat.search('BIT 100081')
```

面向对象用法：编译后的多次操作

```
regex = re.compile(pattern, flags=0)
```

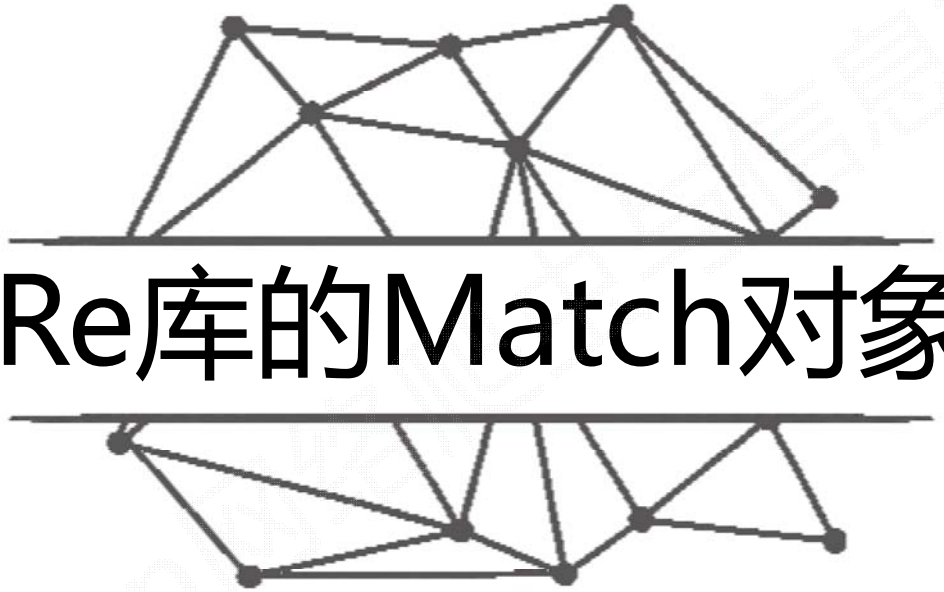
将正则表达式的字符串形式编译成正则表达式对象

- **pattern** ： 正则表达式的字符串或原生字符串表示
- **flags** ： 正则表达式使用时的控制标记

```
>>> regex = re.compile(r'[1-9]\d{5}')
```

Re库的另一种等价用法

函数	说明
regex.search()	在一个字符串中搜索匹配正则表达式的第一个位置，返回match对象
regex.match()	从一个字符串的开始位置起匹配正则表达式，返回match对象
regex.findall()	搜索字符串，以列表类型返回全部能匹配的子串
regex.split()	将一个字符串按照正则表达式匹配结果进行分割，返回列表类型
regex.finditer()	搜索字符串，返回一个匹配结果的迭代类型，每个迭代元素是match对象
regex.sub()	在一个字符串中替换所有匹配正则表达式的子串，返回替换后的字符串



Re库的Match对象

Match对象介绍

Match对象是一次匹配的结果，包含匹配的很多信息

```
>>> match = re.search(r'[1-9]\d{5}', 'BIT 100081')
>>> if match:
        print(match.group(0))
>>> type(match)
<class '_sre.SRE_Match'>
```

Match对象的属性

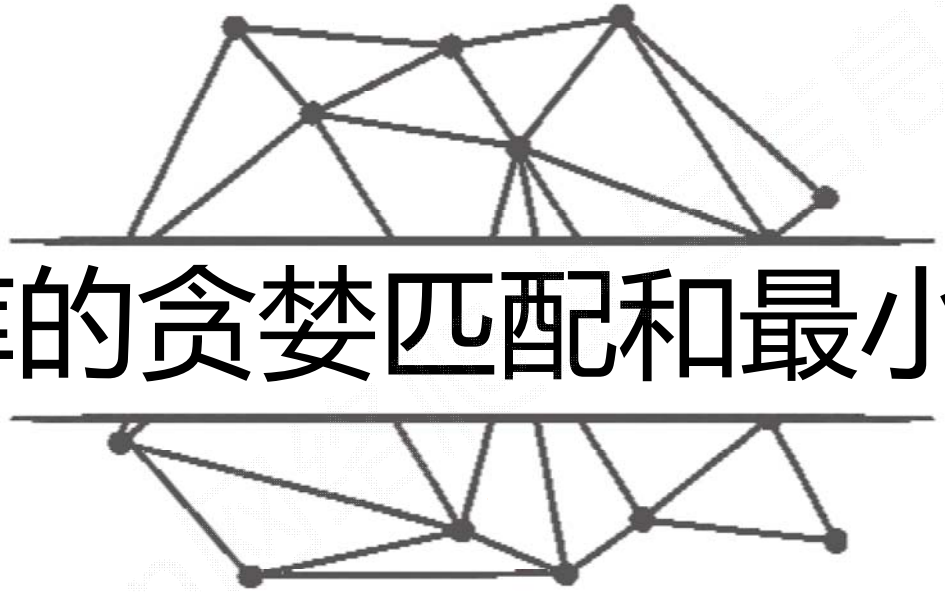
属性	说明
.string	待匹配的文本
.re	匹配时使用的patter对象（正则表达式）
.pos	正则表达式搜索文本的开始位置
.endpos	正则表达式搜索文本的结束位置

Match对象的方法

方法	说明
.group(0)	获得匹配后的字符串
.start()	匹配字符串在原始字符串的开始位置
.end()	匹配字符串在原始字符串的结束位置
.span()	返回(.start(), .end())

Match对象实例

```
>>> import re
>>> m = re.search(r'[1-9]\d{5}', "BIT100081 TSU100084")
>>> m.string
'BIT100081 TSU100084'
>>> m.re
re.compile('[1-9]\\d{5}')
>>> m.pos
0
>>> m.endpos
19
>>> m.group(0)
'100081'
>>> m.start()
3
>>> m.end()
9
>>> m.span()
(3, 9)
```



Re库的贪婪匹配和最小匹配

实例

```
>>> match = re.search(r'PY.*N', 'PYANBNCNDN')
>>> match.group(0)
```

同时匹配长短不同的多项，返回哪一个呢？

贪婪匹配

```
>>> match = re.search(r'PY.*N', 'PYANBNCNDN')
>>> match.group(0)
'PYANBNCNDN'
```

Re库默认采用贪婪匹配，即输出匹配最长的子串

最小匹配

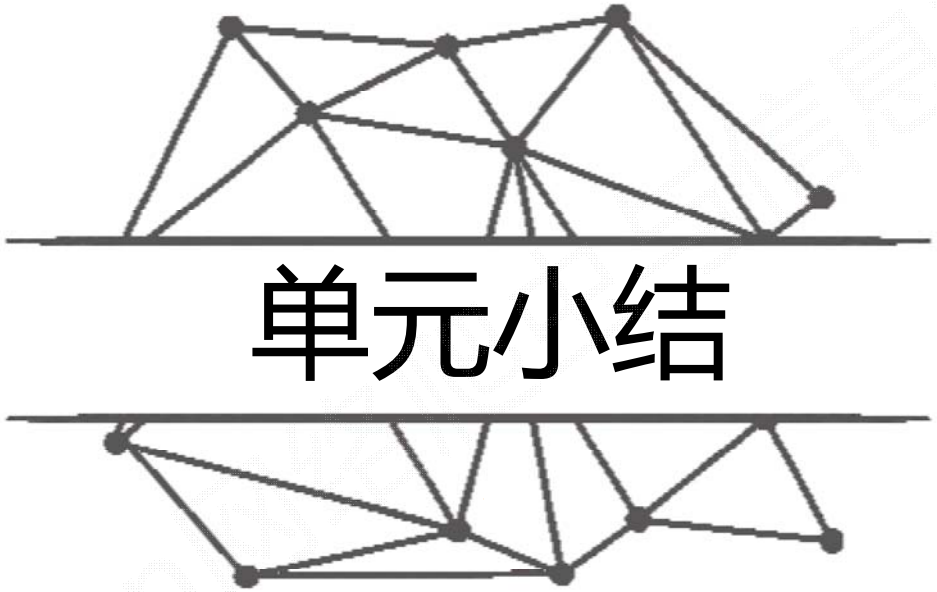
如何输出最短的子串呢？

```
>>> match = re.search(r'PY.*?N', 'PYANBNCNDN')
>>> match.group(0)
'PYAN'
```

最小匹配操作符

操作符	说明
*?	前一个字符0次或无限次扩展，最小匹配
+?	前一个字符1次或无限次扩展，最小匹配
??	前一个字符0次或1次扩展，最小匹配
{m,n}?	扩展前一个字符m至n次（含n），最小匹配

只要长度输出可能不同的，都可以通过在操作符后增加?变成最小匹配



单元小结

Re库(正则表达式)入门

正则表达式是用来简洁表达一组字符串的表达式

```
r'\d{3}-\d{8}|\d{4}-\d{7}'
```

```
re.search()
re.match()
re.findall()
re.split()
re.finditer()
re.sub()
```

=

```
regex=re.compile()
```

+

```
regex.search()
regex.match()
regex.findall()
regex.split()
regex.finditer()
regex.sub()
```