正则表达式 Regular Expression

目 录

1. Overview(正则表达式概述)

1.1. Regular Expresson(什么是正则表达式)

正则表达式: Regular Expression, 是一些由特殊的字符和符号组成的字符串, 主要用来进行高级的文本匹配、搜索、替换等功能操作。

不论是系统级别的软件(类似操作系统)，或者应用程序软件中，对于字符串的操作都是最为频繁和复杂的，正则表达式(Regular Expresson)的出现，对于通过程序的方式自动的批量的进行字符串的操作提供了可行性。

REMARK:正则表达式的由来
在很久很久以前，美国新泽西州和底特律的两位伟大的人类学家(神经科专家和精神科专家，当然他们也是优秀的数学家)Warren McCulloch、Walter Pitts 研究怎么通过数学的方式来表达人类神经网络认知世界的过程，将神经系统中的神经元描述成数学中的自动控制大院，并将实验数据和假设进行了猜解和整合形成了文档资料，也算是最初的基于理论层次的指导思想

另一个哈特福德市的数学家 Stephen Kleene 在他们研究的基础上，通过添加数学算法，将神经网络和数学的正则集合符号来描述这一过程，并发表论文《神经网事件的表示法》，引入了正则表达式的概念

在很多年之后，Ken Thompson 将这个成果应用在了计算机的操作系统中用于提升系统对于字符的处理效率，并取得斐然的成果，Unix 就是在这样的基础上如虎添翼的，Ken Thompson 就是大名鼎鼎的 Unix 之父。

正则表达式是一个独立的技术，在实际操作过程中被很多软件、编程语言等所支持
PYTHON 中就提供了 re 模块对于正则表达式进行了非常良好的技术支持，正则操作处理和字符串在 python 中也是有着非常不错的处理效率的。

REMARK:正则表达式也是一个字符串，但是它之所以可以对目标字符串进行匹配搜索查询，是因为正则表达式字符串可以被编译成一个用于匹配的模式对象。
编译正则表达式，就需要用到正则表达式引擎(类似运行 python 代码需要 python 解释器)，目前比较主流的引擎：DFA,NFA,POSIX NFA

1.2. Quick Start(第一个正则表达式)

我们通过定义一个简单的正则表达式，然后在目标字符串中进行查询匹配的操作，完成第一个程序的编写，同时对于正则表达式又一个初步的了解和认知。

```
'''
AUTHOR: DAMU 牟文斌
VERSION: V1.0.000
DESC: TODO
'''
# 引入正则表达式模块
import re

# 目标字符串
s = 'food foo tonight'
```

```
# 正则表达式
r = r'foo'

# 在目标字符串中查询符合正则表达式的字符
res = re.findall(r, s)

# 打印结果 查询到两个数据#['foo', 'foo']
print(res)
```

2. Syntax of Regex(正则表达式操作语法)

2.1. (Base Syntax)基本语法

正则表达式的核心就是**匹配**，匹配用到的也是字符串，对于正则表达式的理解就需要对正则表达式的语法有一个比较良好的熟悉度。下面针对正则表达式的常见操作，分成三个部分进行了简要描述，对正则表达式语法先有一个全面的了解。

2.1.1. 基本正则语法

匹配字符符号	描述	案例
literal	匹配文本字面值	foo
re1|re2	匹配正则表达式 re1 或者 re2	foo|bar
^	匹配目标字符串开头位置	^Dear
$	匹配目标字符串结束位置	py$
.	匹配任意一个字符(\n 除外)	h.llo
?	匹配任意一个字符出现 0 次/1 次	foo?
+	匹配任意一个字符出现 1 次/n 次	foo+
*	匹配任意一个字符出现 0 次/n 次	foo*
{m}	匹配任意一个字符按 m 此	foo{m}
{m,}	匹配任意一个字符出现至少 m 次	foo{m,}
{m,n}	匹配任意一个字符出现 m~n 次	foo{2,5}
[0-9]	匹配任意一个 0~9 之间的数字	[0-9]{11}
[^0-9]	匹配任意一个非数字字符	
[3-6]	匹配任意一个 3~6 之间的数字	
[0-10]	匹配 00 或者 10 两个数字	
[a-z]	匹配任意一个小写字母	
[A-Z]	匹配任意一个大写字母	
[a-zA-Z]	匹配任意一个字母	
[a-zA-Z0-9_]	匹配任意字母/数字/下划线	

(..)	匹配分组正则表达式	(0-9){3}-(0-9){4}

2.1.2. 正则表达式：元字符

符号	描述	示例
\d	匹配任意一个[0-9]的数字	data**\d+**.txt
\D	匹配任意一个非数字字符	info_**\D+**.log
\s	匹配任意一个空白字符[\n\t\r\v\f]	<a**\s+**href=
\S	匹配任意一个非空白字符	\S{6}
\w	匹配任意一个字母/数字/下划线	\w{8,18}
\W	匹配任意一个非字母/数字/下划线	
\b	匹配任意一个单词的边界	\bfood\b
\	转义字符，可以用于匹配正则中用到的字符	\\, \., *, \+,...

2.1.3. 正则表达式：零宽断言

符号	描述	示例
(?reg)	特殊标记参数	
(?:reg)	匹配不用保存的分组	
(?P<name>reg)	匹配到分组结果命名 name	
(?P=name)	匹配(?P<name>reg)前的文本	
(?#comment)	注释	
(?=reg)	正向前视断言(零宽断言)	
(?!reg)	负向前视断言(零宽断言)	
(?<=reg)	正向后视断言(零宽断言)	
(?<!reg)	负向后视断言(零宽断言)	
(?(id/name)Yre1/Nre2)	条件断言分组	

2.2.(Example for Regexp)正则语法案例

2.2.1. 基本正则语法

```
'''
AUTHOR: DAMU 牟文斌
VERSION: V1.0.000
DESC: TODO
'''
import re
```

```
s = '''Are you new to Django or to programming? This is the place to start!
From scratch: Overview | Installation
Tutorial: Part 1: Requests and responses | Part 2: Models and the admin site
| Part 3:
Views and templates | Part 4: Forms and generic views | Part 5: Testing |
Part 6:
Static files | Part 7: Customizing the admin site
Advanced Tutorials: How to write reusable apps | Writing your first patch
for Django
'''

# 基本语法：字符匹配
reg1 = r'to'

print(re.findall(reg1, s)) # ['to', 'to', 'to', 'to', 'to', 'to', 'to']

# 基本语法：或者匹配
reg2 = r'Django|Part'

print(re.findall(reg2, s)) # ['Django', 'Part', 'Part', 'Part', 'Part',
'Part', 'Part', 'Part', 'Django']

# 基本语法：开头匹配
reg3 = r'^Are'

print(re.findall(reg3, s)) # ['Are']

# 基本语法：结尾匹配
reg4 = r'Django$'

print(re.findall(reg4, s)) # ['Django']

# 基本语法：任意字符匹配
reg5 = r'Ho.'

print(re.findall(reg5, s)) # ['How']

# 基本语法：范围匹配?
reg6 = r'pr?'

print(re.findall(reg6, s)) # ['pr', 'p', 'p', 'p', 'p', 'p', 'p']

# 基本语法：范围匹配+
```

```
reg7 = r'pr+'

print(re.findall(reg7, s)) # ['pr']

# 基本语法：范围匹配*
reg8 = r'pr*'

print(re.findall(reg8, s)) # ['pr', 'p', 'p', 'p', 'p', 'p', 'p']

# 基本语法：范围匹配{m}
reg9 = r'l{2}'

print(re.findall(reg9, s))  # ['ll']

# 基本语法：范围匹配{m,n}
reg10 = r'l{1,3}'

print(re.findall(reg10, s)) # ['l', 'll', 'l', 'l', 'l', 'l', 'l', 'l']

# 基本语法：范围匹配[0-9]
reg11 = r'[0-9]+'

print(re.findall(reg11, s)) # ['1', '2', '3', '4', '5', '6', '7']

# 基本语法：范围匹配[a-z]
reg12 = r'[a-z]{10}'

print(re.findall(reg12, s)) # ['programmin', 'nstallatio', 'ustomizing']

# 基本语法：范围匹配[a-zA-Z0-9_]
reg13 = r'[0-9a-zA-Z_]{10}'

print(re.findall(reg13, s)) # ['programmin', 'Installati', 'Customizin']
```

2.2.2. 元字符语法

```
'''
AUTHOR: DAMU 牟文斌
VERSION: V1.0.000
DESC: TODO
'''
import re

# 目标字符串
```

```
s = """
Django Community 11445 people, 164 countries, 3854 packages and projects.
"""

# 元字符: \d
reg1 = r'\d+'

print(re.findall(reg1, s)) # ['11445', '164', '3854']

# 元字符: \s
reg2 = r'\s+'

print(re.findall(reg2, s)) # ['\n', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
' ', ' \n']

# 元字符: \b
reg3 = r'\ban.'

print(re.findall(reg3, s)) # ['and']

# 元字符:\w
reg4 = r'\w{5}'

print(re.findall(reg4, s)) # ['Djang', 'Commu', '11445', 'peopl', 'count',
'packa', 'proje']
```

2.2.3. 分组查询语法


```
'''
AUTHOR: DAMU 牟文斌
VERSION: V1.0.000
DESC: TODO
'''
import re

s = '''
<img src="./images/1.jpg"/>
<img src="./images/2.jpg"/>
<img src="./images/3.jpg"/>
<img src="./images/4.jpg"/>
<img src="./images/5.jpg"/>
'''
```

```
# 正则表达式中添加分组语法
reg = r'<img\s+src="(.*)"/>'

res = re.finditer(reg, s)
for r in res:
    print(r.group())
    print(r.group(1))

--- result
<img src="./images/1.jpg"/>
./images/1.jpg
<img src="./images/2.jpg"/>
./images/2.jpg
<img src="./images/3.jpg"/>
./images/3.jpg
<img src="./images/4.jpg"/>
./images/4.jpg
<img src="./images/5.jpg"/>
./images/5.jpg
```

2.2.4. 零宽断言语法

```
'''
AUTHOR: DAMU 牟文斌
VERSION: V1.0.000
DESC: TODO
'''
import re

# 目标字符串
s = 'aahelloworldbbhellojerryjerryup'

# 零宽断言：正向前视断言
reg1 = r'.{2}hello(?=world)'
res = re.finditer(reg1, s)
for r in res:
    print(r.group()) # aahello

# 零宽断言：负向前视断言
reg2 = r'.{2}hello(?!world)'
res = re.finditer(reg2, s)
for r in res:
    print(r.group()) # bbhello
```

```python
# 零宽断言：正向后视断言
reg3 = r'(?<=hello)jerry.{2}'
res = re.finditer(reg3, s)
for r in res:
    print(r.group()) # jerryje

# 零宽断言：负向后视断言
reg3 = r'(?<!hello)jerry.{2}'
res = re.finditer(reg3, s)
for r in res:
    print(r.group()) # jerryup
```

2.3. 贪婪匹配 & 懒惰匹配

贪婪模式和懒惰模式(非贪婪)是在正则操作过程中，需要严格注意的一个问题，如果操作不当的话很容易在匹配过程中得到非正常数据或者垃圾数据。

2.3.1. 什么是贪婪和非贪婪模式

贪婪模式：在正则表达式匹配成功的前提下，尽可能多的匹配结果数据
非贪婪模式：在正则表达式匹配成功的前提下，尽可能少的匹配结果数据

2.3.2. 贪婪和非贪婪模式


```python
'''
AUTHOR: DAMU 牟文斌
VERSION: V1.0.000
DESC: TODO
'''
import re

target_str = "<div>hello</div><p>world</p><div>regular expression</div>"

# 正则表达式：贪婪模式
reg = re.compile(r"<div>.*</div>")
# 查询数据
res = reg.search(target_str)
print(res.group())
# <div>hello</div><p>world</p><div>regular expression</div>
```

```
# # 正则表达式：非贪婪模式
reg = re.compile(r"<div>.*?</div>")
# 查询数据
res = reg.search(target_str)
print(res.group())
# <div>hello</div>
```

3. RegExp for Python(python 中的正则)

3.1. re 正则模块

python 中通过标准库 re 模块对于正则表达式进行良好的技术支持。

3.1.1. 为什么使用正则表达式

正则表达式本身就是对于字符串的高效处理，尽管字符串本身也内置了一些操作函数，但是相对于较为复杂的需求，通过正则表达式的操作更加的灵活并且效率更高

正则表达式在程序中的主要应用，有以下几个方面：

- 匹配：测试一个字符串是否符合正则表达式语法，返回 True 或者 False
- 获取：从目标字符串中，提取符合正则表达式语法的字符数据
- 替换：在目标字符串中查询符合正则语法的字符，并替换成指定的字符串
- 分割：按照符合正则表达式的语法字符对目标字符串进行分割

3.1.2. 创建正则表达式对象的方式

python 中有两种方式可以创建正则表达式的匹配对象，两种方式使用都较为广泛，具体满足开发人员所在团队的开发规范即可

- 隐式创建：通过一个普通字符串，直接创建正则表达式
  要求：必须通过 re 模块的函数调用才能正常编译执行

```
import re

# 创建一个正则表达式
reg = r“\bfoo\b”

# 从目标字符串中提取数据
result = re.findall(reg, target_str)
```

- 显式创建：通过内建函数 compile()编译创建一个正则表达式对象

要求：可以通过调用该类型的方法，完成字符串的操作

```
import re

pattern = re.compile(“\\bfoo\\b”)

pattern.findall(target_str)
```

- 操作简要案例

```
'''
AUTHOR: DAMU 牟文斌
VERSION: V1.0.000
DESC: TODO
'''
import re

target_str = "helloworldhelloregexp"

# 1. 函数
reg = r"(?<=hello).{3}"
print(re.findall(reg, target_str)) # ['wor', 'reg']

# 2. 对象
reg2 = re.compile(r"(?=hello).{3}")
print(reg2.findall(target_str)) # ['wor', 'reg']
```

3.2. 常用方法

3.2.1. 常用方法

常用方法	描述
re.findall(s, start, end)	返回(指定位置)查询到结果内容的列表
re.finditer(s, start, end)	返回(指定位置)查询到结果匹配对象的生成器
re.search(s, start, end)	返回(指定位置)第一次查询到的匹配对象
re.match(s, start, end)	返回(指定位置)从第一个字符匹配到的结果对象
re.sub(s, r, c)	使用 r 替换字符串中所有匹配的数据，c 是替换次数
re.subn(s, r, c)	使用 r 替换字符串中所有匹配的数据，c 是替换次数
re.split(s, m)	使用正则表达式拆分字符串，m 是拆分次数

3.2.2. 简单示例

```
'''
AUTHOR: DAMU 牟文斌
VERSION: V1.0.000
DESC: TODO
'''
import re

target_str = "helloworldhellojerryhelloregularexpression"

# 定义正则表达式
reg = r"hello"

# findall()
print(re.findall(reg, target_str)) # ['hello', 'hello', 'hello']

# finditer()
res = re.finditer(reg, target_str)
for s in res:
    print(s.group())# hello hello hello

# search()
res = re.search(reg, target_str)
print(res) # <_sre.SRE_Match object; span=(0, 5), match='hello'>

# match()
res = re.match(reg, target_str)
print(res) # <_sre.SRE_Match object; span=(0, 5), match='hello'>

# sub
res = re.sub(reg, "**", target_str)
print(res) # **world**jerry**regularexpression

# subn
res = re.subn(reg, "**", target_str)
print(res) # ('**world**jerry**regularexpression', 3)

# split
res = re.split(reg, target_str)
print(res) # ['', 'world', 'jerry', 'regularexpression']
```