正则表达式概述

- 概念
    - 正则表达式是对字符串操作的一种逻辑公式，就是用事先定义好的一些特定字符、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（可以用来做检索，截取或者替换操作）
- 简介
    - 正则表达式是对字符串（包括普通字符（例如，a 到 z 之间的字母）和特殊字符（称为“元字符”））操作的一种逻辑公式，就是用事先定义好的一些特定字符、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。正则表达式是一种文本模式，模式描述在搜索文本时要匹配的一个或多个字符串
- 作用
    - 1.给定的字符串是否符合正则表达式的过滤逻辑（称作“匹配”）
    - 2.可以通过正则表达式，从字符串中获取我们想要的特定部分。
    - 3.还可以对目标字符串进行替换操作

在 Python 中需要通过正则表达式对字符串进行匹配的时候， 可以一个模块， 名字为 re

re 模块之 match 的基本使用：

- 语法：
    - result = re.match(正则表达式,要匹配的字符串)
- 意义：
    - re.match 是用来进行正则匹配检查的方法，如果字符串开头的 0 个或多个字符匹配正则表达式模式，则返回相应的 match 对象。如果字符串不匹配模式，返回 None（注意不是空字符串""）
    - 匹配对象 Macth Object 具有 group()方法， 用来返回字符串的匹配部分,具有 span()方法。返回匹配字符串的位置（元组存储开始，结束位置），具有 start(),end()方法，存储匹配数据的开始和结束位置。(也可以通过对象的 dir(对象查看对象的方法))
    - 注意：如果想在目标字符串的任意位置查找，需要使用 search

- 案例：(mathch vs search)

```
import re
pattern = 'Hello'
str1 = 'HelloWorld'
v = re.match(pattern,str1)
print(type(v))
print(v)
# print(dir(v))
```

```
#使用 group 可以获取匹配到的数据
# print(v.group())
```

```

# 表示字符

- 语法格式:

  字符 | 功能
  ---|---
  . | 匹配任意一个字符（除了\n）
  [] | 匹配列表中的字符
  \d | 匹配数字，即 0-9
  \D | 匹配非数字
  \s | 匹配空白、即空格（\n,\t）
  \S | 匹配非空格
  \w | 匹配单词字符，即 a-z,A-Z,0-9,_
  \W | 匹配非单词字符
- 使用示例 1

  ```
  import re
  v = re.match('.','a')
  v = re.match('.','1')
  v = re.match('.','_')
  #返回 None
  v = re.match('.','\n')
  print(v)
  v = re.match('\d','1')
  print(v)
  v = re.match('\D','a1')
  print(v)
  v = re.match('\s',' ')
  v = re.match('\s','\n')
  v = re.match('\s','\t')
  print(v)
  #非空格
  v = re.match('\S',' ')
  print(v)
  v = re.match('\w','a')
  v = re.match('\w','A')
  v = re.match('\w','1')
  v = re.match('\w','_')
  print(v)
```
```

```
v = re.match('\W','a')
v = re.match('\W','A')
v = re.match('\W','1')
v = re.match('\W','_')
print(v)

v = re.match('\w\W','1a')
print(v)

```

- 使用示例 2：‘[]’的使用

```
#手机号匹配问题
v = re.match('\d\d\d\d\d\d\d\d\d\d\d','13312341234')
v = re.match('1[35789]\d\d\d\d\d\d\d\d\d','12312341234')
#\d == [0-9]
#\D == [^0-9]
#\w == [a-zA-Z0-9_]
#\W == [^a-zA-Z0-9_]
print(v)
```

表示数量（匹配多个字符）

- 语法

字符	功能
*	匹配前一个字符出现 0 次或者无限次（可有可无）
+	匹配前一个字符出现 1 次或者无限次(至少有 1 次)
？	匹配前一个字符串出现 1 次或者 0 次(要么 1 次要么没有)
{m}	匹配前一个字符出现 m 次
{m,}	匹配前一个字符至少出现 m 次
{m,n}	匹配前一个字符出现 m 到 n 次

- 代码验证

```
import re
pattern = '\d*'
#注意，这时候表示数字可有可无，如果没有的话，则匹配''
v = re.match(pattern,'abc123')
print(v)
print('11111111111111')
pattern = '\d+'
v = re.match(pattern,'abc123')
print(v)
v = re.match(pattern,'123abc123')
```

```
print(v)
pattern = '\d?'
v = re.match(pattern,'123abc')
print(v)
pattern = '\d{3}'
v = re.match(pattern,'1234abc')
print(v)
pattern = '\d{3,}'
v = re.match(pattern,'1234abc')
print(v)

pattern = '\d{3,6}'
v = re.match(pattern,'1235674abc')
print(v)
```

- 使用示例 1 *
  - 匹配出一个字符串首字母为大写字符，后边都是小写字符，这些小写字母可有可无
    ```
    pattern = '[A-Z][a-z]*'
    v = re.match(pattern,'Hello')
    print(v)
    ```
- 使用示例 2 +
  - 匹配出有效的变量名
    ```
    #有效的变量名　开头为字母
    pattern = '[a-zA-Z_]+[\w_]*'
    v = re.match(pattern,'1name123')
    print(v)
    ```
- 使用示例 3 ?
  - 匹配出 1-99 之间的数字
    ```
    pattern = '[1-9][0-9]?'
    v = re.match(pattern,'09')
    print(v)
    v = re.match(pattern,'33')
    print(v)
    v = re.match(pattern,'7')
    print(v)
    ```
- 使用示例 4 {m}
  - 匹配出一个随机密码 8-20 位以内
    ```
    ```

```
pattern = '[a-zA-Z0-9_]{8,20}'
v = re.match(pattern,'dafadf22432adfag')
print(v)
```

原始字符串

- 概述:
  - Python 中字符串前边加上 r 表示原生字符串
- 示例:
  - 字符串中的使用
    - 字符串 s = '\n123' 与 s = r'\n123'
  - 正则中使用

```
s = '\n123'
print(s)
s = r'\n123'
print(s)
# s = '\\n123'
# print(s)
# pattern = '\\\\n\d{3,}'
pattern = r'\\n\d{3,}'
v = re.match(pattern,s)
print(v)
```

表示边界

- 语法及意义:

字符	功能
^	匹配字符串开头
$	匹配字符串结尾
\b	匹配一个单词的边界
\B	匹配非单词的边界

- 案例:
  - 使用示例 1 匹配 QQ 邮箱 &

```
#匹配 qq 邮箱,     5-10 位
pattern = '[\d]{5,10}@qq.com'
#必须限制结尾的
# pattern = '[1-9]\d{4,9}@qq.com$'
#正确的地址
v = re.match(pattern,'12345@qq.com')
#未限制结尾的前提下使用不正确的地址
# v = re.match(pattern,'12345@qq.comabc')
```

```
print(v)
```

- 使用示例 2 \b 匹配单词边界

```
pattern = r'.*\bab'
#ab 左边界的情况
v = re.match(pattern,'123 abr')
print(v)

pattern = r'.*ab\b'
#ab 为右边界的情况
v = re.match(pattern,'wab')
print(v)
```

- 使用示例 3 \B 匹配非单词边界

```
#ab 不为左边界
pattern = r'.*\Bab'
v = re.match(pattern,'123 abr')
print(v)
#ab 不为右边界
pattern = r'.*ab\B'
v = re.match(pattern,'wab')
print(v)
```

匹配分组

- 语法

字符	功能
|	匹配左右任意一个表达式
(ab)	将括号中的字符作为一个分组
\num	引用分组 num 匹配到的字符串
(?p<name>)	分别起组名
(?p=name)	引用别名为 name 分组匹配到的字符串

- 示例 1 |的使用
  - 匹配 0-100 之间所有的数字

```
pattern = '[1-9]?\d$|100$'
v = re.match(pattern,'0')
print(v)
```

```
v = re.match(pattern,'10')
print(v)
v = re.match(pattern,'100')
print(v)
v = re.match(pattern,'99')
print(v)
v = re.match(pattern,'200')
print(v)

```

- 示例 2 ()的使用
  - 匹配座机号码
    ```
    #匹配一个固定电话号码  010-66668888
    pattern = r'(\d+)-(\d{5,8}$)'
    v = re.match(pattern,'010-66668888')
    print(v)
    print(v.group())
    print(v.group(1))
    print(v.group(2))
    print(v.groups())
    print(v.groups()[0])
    print(v.groups()[1])
    ```
- 示例 3 \num 的使用
  - 匹配出网页标签内的数据
  ```
  #
  s = '<html><title>我是标题</title></html>'
  #优化前
  # pattern = r'<.+><.+>.+</.+></.+>'
  #优化后  可以使用分组  \2  表示引用第 2 个分组  \1 表示引用第 1 个分组
  pattern = r'<(.+)><(.+)>.+</\2></\1>'
  v = re.match(pattern,s)
  print(v)

  ```
- 示例 4 ?P<要起的别名> (?P=起好的别名)
  ```
  s = '<html><h1>我是一号字体</h1></html>'
  # pattern = r'<(.+)><(.+)>.+</\2></\1>'
  #如果分组比较多的话，数起来比较麻烦，可以使用起别名的方法?P<要起的名字>  以及使用别名(?P=之前起的别名)
  pattern = r'<(?P<key1>.+)><(?P<key2>.+)>.+</(?P=key2)></(?P=key1)>'
```

```
v = re.match(pattern,s)
print(v)
```

re 模块的高级用法

- search
  - 作用:
    - 扫描字符串，查找正则表达式模式产生匹配的第一个位置，并返回相应的匹配对象。如果字符串中没有与模式匹配的位置，则返回 None;
  - 用法:

```
import re
v = re.match('\d+','阅读次数为 9999 次')
print(v)
v1 = re.search('\d+','阅读次数为 999 次')
print(v1)
```

- findall
  - 作用
    - 从左到右扫描字符串，并按照找到的顺序返回匹配。如果模式中有一个或多个组，返回组列表
  - 用法

```
list1 = re.findall(r'\d+',"阅读次数 C:129 Python:999 C++:99")
print(list1)
```

- sub
  - 作用:
    - 返回通过替换 repl 替换字符串中最左边不重叠的模式出现而得到的字符串。如果没有找到模式，则返回字符串不变
  - 用法 1: 直接替换

```
#将目标字符串中所有的字符'c'替换成'a'
s = re.sub(r'a','c',"adfjalkdfkdasf")
print(s)
```

  - 用法 2: 使用函数(可以运算)替换

```
```

```
def replace(result):
print(type(result))
print(type(result.group()))
print(result.group())
r = int(result.group())+ 1
return str(r)
#将目标字符串中所有的阅读次数+1
v = re.sub(r'\d+',replace,"阅读次数 C:129 Python:999 C++:99")
print(v)
```

- split
  - 作用:
    - 通过指定模式拆分字符串
  - 用法 1:
    - 按指定的格式拆分字符串

```
v = re.split(r',|-|:','Tom:HelloWorld,james-bond')
print(v)

```

贪婪模式和非贪婪

- 什么是贪婪模式?
  - Python 里数量词默认是贪婪的， 总是尝试匹配尽可能多的字符
- 什么是非贪婪
  - 与贪婪相反，总是尝试匹配尽可能少的字符，可以使用"*","?","+","{m,n}"后面加上？，使贪婪变成非贪婪
- 使用示例 1:

```
#贪婪模式，.+中的'.'会尽量多的匹配
# v = re.match(r'(.+)(\d+-\d+-\d+)','This is my tel:133-1234-1234')
v = re.match(r'(.+?)(\d+-\d+-\d+)','This is my tel:133-1234-1234')
print(v.group(1))
print(v.group(2))
```

- 使用示例 2

```
#贪婪模式
v= re.match(r'abc(\d+)','abc123')
print(v.group(1))
#非贪婪模式
v= re.match(r'abc(\d+?)','abc123')
```

```
print(v.group(1))
```