# 正则表达式

## 正则表达式概述

正则表达式，又称正规表示式、正规表示法、正规表达式、规则表达式、常规表示法（英语：Regular Expression，在代码中常简写为regex、regexp或RE），是计算机科学的一个概念。正则表达式使用单个字符串来描述、匹配一系列匹配某个句法规则的字符串。在很多文本编辑器里，正则表达式通常被用来检索、替换那些匹配某个模式的文本。

Regular Expression的“Regular”一般被译为“正则”、“正规”、“常规”。此处的“Regular”即是“规则”、“规律”的意思，Regular Expression即“描述某种规则的表达式”之意。

## re模块操作

在Python中需要通过正则表达式对字符串进行匹配的时候，可以使用一个模块，名字为re

### re模块的使用过程

#coding=utf-8

# 导入re模块

import re

# 使用match方法进行匹配操作

result = re.match(正则表达式,要匹配的字符串)

# 如果上一步匹配到数据的话，可以使用group方法来提取数据

result.group()

re.match是用来进行正则匹配检查的方法，若字符串匹配正则表达式，则match方法返回匹配对象（Match Object），否则返回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。

### re模块示例

(匹配以taobao开头的语句)

#coding=utf-8

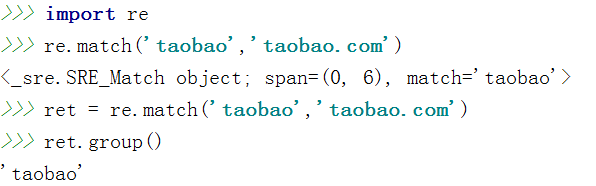
import re

result = re.match("taobao","taobao.com")

result.group()

运行结果为：

taobao



说明

* re.match() 能够匹配出以xxx开头的字符串

## 表示字符

在上一小节中，了解到通过re模块能够完成使用正则表达式来匹配字符串

本小节，将要讲解正则表达式的单字符匹配



中文

匹配中文字符的正则表达式： [\u4e00-\u9fa5]

示例1： .

import re

ret = re.match(".","a")

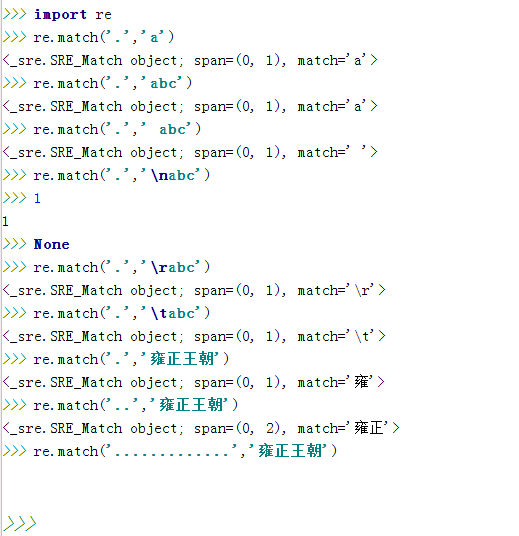
ret.group()

ret = re.match(".","b")

ret.group()

ret = re.match(".","M")

ret.group()



示例2：

#coding=utf-8

import re

# 如果hello的首字符小写，那么正则表达式需要小写的h

ret = re.match("h","hello Python")

ret.group()

# 如果hello的首字符大写，那么正则表达式需要大写的H

ret = re.match("H","Hello Python")

ret.group()

# 大小写h都可以的情况

ret = re.match("[hH]","hello Python")

ret.group()

ret = re.match("[hH]","Hello Python")

ret.group()

# 匹配0到9第一种写法

ret = re.match("[0123456789]","7Hello Python")

ret.group()

# 匹配0到9第二种写法

ret = re.match("[0-9]","7Hello Python")

ret.group()



示例3：

#coding=utf-8

import re

# 普通的匹配方式

ret = re.match("嫦娥1号","嫦娥1号发射成功")

print ret.group()

ret = re.match("嫦娥2号","嫦娥2号发射成功")

print ret.group()

ret = re.match("嫦娥3号","嫦娥3号发射成功")

print ret.group()

# 使用\d进行匹配

ret = re.match("嫦娥\d号","嫦娥1号发射成功")

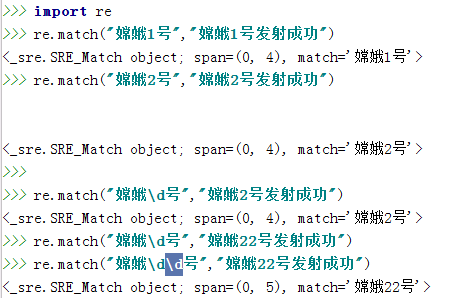
print ret.group()

ret = re.match("嫦娥\d号","嫦娥2号发射成功")

print ret.group()

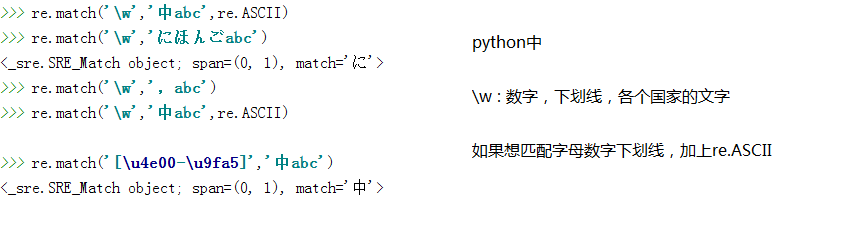
ret = re.match("嫦娥\d号","嫦娥3号发射成功")

print ret.group()





解决\w的问题



说明

* 其他的匹配符参见后面章节的讲解

## 原始字符串

>>> mm = "c:\\a\\b\\c"

>>> mm

'c:\\a\\b\\c'

>>> print(mm)

c:\a\b\c

>>> print(mm)

c:\a\b\c

>>> re.match("c:\\\\",mm).group()

'c:\\'

>>> ret = re.match("c:\\\\",mm).group()

>>> print(ret)

c:\

>>> ret = re.match("c:\\\\a",mm).group()

>>> print(ret)

c:\a

>>> ret = re.match(r"c:\\a",mm).group()

>>> print(ret)

c:\a

>>> ret = re.match(r"c:\a",mm).group()

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

AttributeError: 'NoneType' object has no attribute 'group'

>>>

说明

Python中字符串前面加上 r 表示原生字符串，

与大多数编程语言相同，正则表达式里使用"\"作为转义字符，这就可能造成反斜杠困扰。假如你需要匹配文本中的字符"\"，那么使用编程语言表示的正则表达式里将需要4个反斜杠"\\"：前两个和后两个分别用于在编程语言里转义成反斜杠，转换成两个反斜杠后再在正则表达式里转义成一个反斜杠。

Python里的原生字符串很好地解决了这个问题，有了原始字符串，你再也不用担心是不是漏写了反斜杠，写出来的表达式也更直观。

>>> ret = re.match(r"c:\\a",mm).group()

>>> print(ret)

c:\a

## 表示数量

匹配多个字符的相关格式



示例1：\*

需求：匹配出，一个字符串第一个字母为大写字符，后面都是小写字母并且这些小写字母可有可无

#coding=utf-8

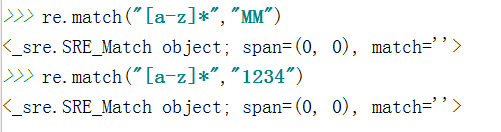
import re

ret = re.match("[A-Z][a-z]\*","Mm")

ret.group()

ret = re.match("[A-Z][a-z]\*","Aabcdef")

ret.group()



示例2：+

需求：匹配出，变量名是否有效

#coding=utf-8

import re

ret = re.match("[a-zA-Z\_]+[\w\_]\*","name1")

ret.group()

ret = re.match("[a-zA-Z\_]+[\w\_]\*","\_name")

ret.group()

ret = re.match("[a-zA-Z\_]+[\w\_]\*","2\_name")

ret.group()

示例3：?

需求：匹配出，0到99之间的数字

#coding=utf-8

import re

ret = re.match("[1-9]?[0-9]","7")

ret.group()

ret = re.match("[1-9]?[0-9]","33")

ret.group()

ret = re.match("[1-9]?[0-9]","09")

ret.group()

示例4：{m}

需求：匹配出，8到20位的密码，可以是大小写英文字母、数字、下划线

#coding=utf-8

import re

ret = re.match("[a-zA-Z0-9\_]{6}","12a3g45678")

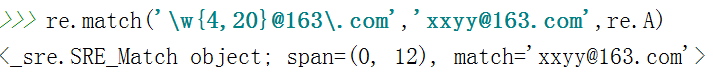
ret.group()

ret = re.match("[a-zA-Z0-9\_]{8,20}","1ad12f23s34455ff66")

ret.group()

练一练

题目1：匹配出163的邮箱地址，且@符号之前有4到20位，例如hello@163.com



## 表示边界



示例1：$

需求：匹配163.com的邮箱地址

#coding=utf-8

import re

# 正确的地址

ret = re.match("[\w]{4,20}@163\.com", "xiaoWang@163.com")

ret.group()

# 不正确的地址

ret = re.match("[\w]{4,20}@163\.com", "xiaoWang@163.comheihei")

ret.group()

# 通过$来确定末尾

ret = re.match("[\w]{4,20}@163\.com$", "xiaoWang@163.comheihei")

ret.group()

示例2: \b

>>> re.match(r".\*\bver\b", "ho ver abc").group()

'ho ver'

>>> re.match(r".\*\bver\b", "ho verabc").group()

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

AttributeError: 'NoneType' object has no attribute 'group'

>>> re.match(r".\*\bver\b", "hover abc").group()

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

AttributeError: 'NoneType' object has no attribute 'group'

>>>

示例3：\B

>>> re.match(r".\*\Bver\B", "hoverabc").group()

'hover'

>>> re.match(r".\*\Bver\B", "ho verabc").group()

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

AttributeError: 'NoneType' object has no attribute 'group'

>>> re.match(r".\*\Bver\B", "hover abc").group()

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

AttributeError: 'NoneType' object has no attribute 'group'

>>> re.match(r".\*\Bver\B", "ho ver abc").group()

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

AttributeError: 'NoneType' object has no attribute 'group'

## 匹配分组



示例1：|

需求：匹配出0-100之间的数字

#coding=utf-8

import re

ret = re.match("[1-9]?\d","8")

ret.group()

ret = re.match("[1-9]?\d","78")

ret.group()

# 不正确的情况

ret = re.match("[1-9]?\d","08")

ret.group()

# 修正之后的

ret = re.match("[1-9]?\d$","08")

ret.group()

# 添加|

ret = re.match("[1-9]?\d$|100","8")

ret.group()

ret = re.match("[1-9]?\d$|100","78")

ret.group()

ret = re.match("[1-9]?\d$|100","08")

ret.group()

ret = re.match("[1-9]?\d$|100","100")

ret.group()

示例2：( )

需求：匹配出163、126、qq邮箱之间的数字

#coding=utf-8

import re

ret = re.match("\w{4,20}@163\.com", "test@163.com")

ret.group()

ret = re.match("\w{4,20}@(163|126|qq)\.com", "test@126.com")

ret.group()

ret = re.match("\w{4,20}@(163|126|qq)\.com", "test@qq.com")

ret.group()

ret = re.match("\w{4,20}@(163|126|qq)\.com", "test@gmail.com")

ret.group()

练习：

>>> ret = re.match("([^-]\*)-(\d+)","010-12345678")

>>> ret.group()

'010-12345678'

>>> ret.group(1)

'010'

>>> ret.group(2)

'12345678'

示例3：\

需求：匹配出<html>hh</html>

#coding=utf-8

import re

# 能够完成对正确的字符串的匹配

ret = re.match("<[a-zA-Z]\*>\w\*</[a-zA-Z]\*>", "<html>hh</html>")

ret.group()

# 如果遇到非正常的html格式字符串，匹配出错

ret = re.match("<[a-zA-Z]\*>\w\*</[a-zA-Z]\*>", "<html>hh</htmlbalabala>")

ret.group()

# 正确的理解思路：如果在第一对<>中是什么，按理说在后面的那对<>中就应该是什么

# 通过引用分组中匹配到的数据即可，但是要注意是元字符串，即类似 r""这种格式

ret = re.match(r"<([a-zA-Z]\*)>\w\*</\1>", "<html>hh</html>")

ret.group()

# 因为2对<>中的数据不一致，所以没有匹配出来

ret = re.match(r"<([a-zA-Z]\*)>\w\*</\1>", "<html>hh</htmlbalabala>")

ret.group()

示例4：\number

需求：匹配出<html><h1>www.taobao.com</h1></html>

#coding=utf-8

import re

ret = re.match(r"<(\w\*)><(\w\*)>.\*</\2></\1>", "<html><h1>www.taobao.com</h1></html>")

ret.group()

ret = re.match(r"<(\w\*)><(\w\*)>.\*</\2></\1>", "<html><h1>www.taobao.com</h2></html>")

ret.group()

示例5：(?P<name>) (?P=name)

需求：匹配出<html><h1>www.taobao.com</h1></html>

#coding=utf-8

import re

ret = re.match(r"<(?P<name1>\w\*)><(?P<name2>\w\*)>.\*</(?P=name2)></(?P=name1)>", "<html><h1>www.taobao.com</h1></html>")

ret.group()

ret = re.match(r"<(?P<name1>\w\*)><(?P<name2>\w\*)>.\*</(?P=name2)></(?P=name1)>", "<html><h1>www.taobao.com</h2></html>")

ret.group()

注意：

(?P<name>)和(?P=name)中的字母p大写

## re模块的高级用法

### search

需求：匹配出文章阅读的次数

import re

ret = re.search(r"\d+", "阅读次数为 9999")

ret.group()

### findall

需求：统计出python、c、c++相应文章阅读的次数

import re

ret = re.findall(r"\d+", "python = 9999, c = 7890, c++ = 12345")

print(ret)

### sub 将匹配到的数据进行替换

需求：将匹配到的阅读次数加1

方法1：

import re

ret = re.sub(r"\d+", '998', "python = 997")

print ret

方法2：

import re

def add(temp):

strNum = temp.group()

num = int(strNum) + 1

return str(num)

ret = re.sub(r"\d+", add, "python = 997 java=100")

print ret

ret = re.sub(r"\d+", add, "python = 99")

print ret

练习



从下面的字符串中取出文本

<div>

<p>岗位职责：</p>

<p>完成推荐算法、数据统计、接口、后台等服务器端相关工作</p>

<p><br></p>

<p>必备要求：</p>

<p>良好的自我驱动力和职业素养，工作积极主动、结果导向</p>

<p>&nbsp;<br></p>

<p>技术要求：</p>

<p>1、一年以上 Python 开发经验，掌握面向对象分析和设计，了解设计模式</p>

<p>2、掌握HTTP协议，熟悉MVC、MVVM等概念以及相关WEB开发框架</p>

<p>3、掌握关系数据库开发设计，掌握 SQL，熟练使用 MySQL/PostgreSQL 中的一种<br></p>

<p>4、掌握NoSQL、MQ，熟练使用对应技术解决方案</p>

<p>5、熟悉 Javascript/CSS/HTML5，JQuery、React、Vue.js</p>

<p>&nbsp;<br></p>

<p>加分项：</p>

<p>大数据，数理统计，机器学习，sklearn，高性能，大并发。</p>

</div>

re.sub(r'</?\w\*>|\n|\s','',s)

### split

根据匹配进行切割字符串，并返回一个列表

需求：切割字符串“info:xiaoZhang 33 shandong”

#coding=utf-8

import re

ret = re.split(r":| ","info:xiaoZhang 33 shandong")

print(ret)

## 贪婪和非贪婪

Python里数量词默认是贪婪的（在少数语言里也可能是默认非贪婪），总是尝试匹配尽可能多的字符；

非贪婪则相反，总是尝试匹配尽可能少的字符。

在"\*","?","+","{m,n}"后面加上？，使贪婪变成非贪婪。

>>> s="This is a number 234-235-22-423"

>>> re.match(".+(\d+-\d+-\d+-\d+)",s)

>>> re.match(".+?(\d+-\d+-\d+-\d+?)",s)

正则表达式模式中使用到通配字，那它在从左到右的顺序求值时，会尽量“抓取”满足匹配最长字符串，在我们上面的例子里面，“.+”会从字符串的启始处抓取满足模式的最长字符，其中包括我们想得到的第一个整型字段的中的大部分，“\d+”只需一位字符就可以匹配，所以它匹配了数字“4”，而“.+”则匹配了从字符串起始到这个第一位数字4之前的所有字符。

解决方式：非贪婪操作符“？”，这个操作符可以用在"\*","+","?"的后面，要求正则匹配的越少越好。

>>> re.match(r"aa(\d+)","aa2343ddd").group(1)

'2343'

>>> re.match(r"aa(\d+?)","aa2343ddd").group(1)

'2'

>>> re.match(r"aa(\d+)ddd","aa2343ddd").group(1)

'2343'

>>> re.match(r"aa(\d+?)ddd","aa2343ddd").group(1)

'2343'

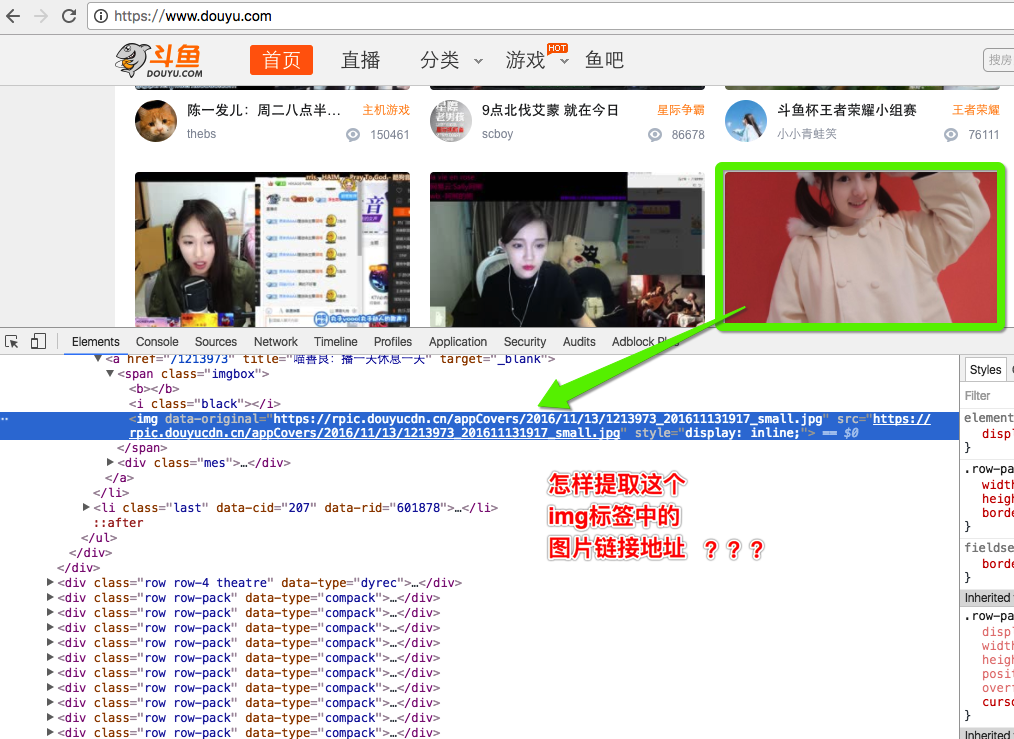
>>> re.match(r"(\d+)(.\*)","2343ddd").groups()

('2343', 'ddd')

>>> re.match(r"(\d+?)(.\*)","2343ddd").groups()

('2', '343ddd')

练一练



字符串为:

|  |
| --- |
| <img data-original="https://rpic.douyucdn.cn/appCovers/2016/11/13/1213973\_201611131917\_small.jpg" src="https://rpic.douyucdn.cn/appCovers/2016/11/13/1213973\_201611131917\_small.jpg" style="display: inline;"> |

请提取url地址

## 小练习

1. 匹配网址

有一批网址：

http://www.interoem.com/messageinfo.asp?id=35

http://3995503.com/class/class09/news\_show.asp?id=14

http://lib.wzmc.edu.cn/news/onews.asp?id=769

http://www.zy-ls.com/alfx.asp?newsid=377&id=6

http://www.fincm.com/newslist.asp?id=415

需要 正则后为：

http://www.interoem.com/

http://3995503.com/

http://lib.wzmc.edu.cn/

http://www.zy-ls.com/

<http://www.fincm.com/>

参考答案：

re.match(r'http.+\.[a-z]{2,3}/',str)

re.match(r'http://.\*?/',str)

re.sub(r'(http://.\*?/).\*',lambda x:x.group(1),str)

2. 找出单词

有一句英文如下：

hello world ha ha

查找所有的单词

|  |
| --- |
| re.split(r' +',’ hello world ha ha’)  re.findall(r' \b\w+\b',’ hello world ha ha’) |