

python 高级

1.正则表达式概述

思考

场景 1: 在一个文件中，查找出 **wangdao** 开头的语句

测试文件

```
wangdao hello python
wangdao c++
cskaoyan ios
cskaoyan php
```

场景 2: 在一个文件中，找到含有 **wangdao** 的语句

测试文件

```
hello wangdao python
www.wangdao.cn c++
cskaoyan ios
cskaoyan php
```

场景 3: 在一个文件中，找到邮箱为 **163** 或者 **126** 的所有邮件地址

2.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

1. 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()
```

2. re 模块示例(匹配以 wangdao 开头的语句)

```
#coding=utf-8
```

```
import re
```

```
result = re.match("wangdao","wangdao.cn")
```

```
print(result.group())
```

运行结果为：

wangdao

3. 说明

- re.match() 能够匹配出以 xxx 开头的字符串

3.匹配单个字符

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

字符	功能
.	匹配任意 1 个字符（除了\n）
[]	匹配[]中列举的字符
\d	匹配数字，即 0-9 dicimal
\D	匹配非数字，即不是数字
\s	匹配空白，即 空格，tab 键 space
\S	匹配非空白
\w	匹配单词字符，即 a-z、A-Z、0-9、_（汉字） word
\W	匹配非单词字符

示例 1: .

```
#coding=utf-8
```

```
import re
```

```
ret = re.match(".", "M")  
print(ret.group())
```

```
ret = re.match("t.o", "too")  
print(ret.group())
```

```
ret = re.match("t.o", "two")  
print(ret.group())
```

运行结果:

```
M  
too  
two
```

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")
```

```
print(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")
print(ret.group())
```

大小写 h 都可以的情况

```
ret = re.match("[hH]", "hello Python")
print(ret.group())
ret = re.match("[hH]", "Hello Python")
print(ret.group())
ret = re.match("[hH]ello Python", "Hello Python")
print(ret.group())
```

匹配 0 到 9 第一种写法

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

匹配 0 到 9 第二种写法

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

```
ret = re.match("[0-35-9]Hello Python", "7Hello Python")
print(ret.group())
```

下面这个正则不能够匹配到数字 4，因此 ret 为 None

```
ret = re.match("[0-35-9]Hello Python", "4Hello Python")
# print(ret.group())
```

运行结果：

```
h
H
h
H
Hello Python
7Hello Python
7Hello Python
7Hello Python
```

示例 3: \d

```
#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())
```

运行结果:

```
嫦娥 1 号
嫦娥 2 号
嫦娥 3 号
嫦娥 1 号
嫦娥 2 号
嫦娥 3 号
```

说明

- 其他的匹配符参见后面章节的讲解

4.匹配多个字符

匹配多个字符的相关格式

字符	功能
*	匹配前一个字符出现 0 次或者无限次，即可有可无
+	匹配前一个字符出现 1 次或者无限次，即至少有 1 次
?	匹配前一个字符出现 1 次或者 0 次，即要么有 1 次，要么没有
{m}	匹配前一个字符出现 m 次
{m,n}	匹配前一个字符出现从 m 到 n 次

示例 1: *

需求：匹配出，一个字符串第一个字母为大小字符，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]*", "M")
print(ret.group())
```

```
ret = re.match("[A-Z][a-z]*", "MnnM")
print(ret.group())
```

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

运行结果：

```
M
Mnn
Aabcdef
```

示例 2: +

需求：匹配出，变量名是否有效

```
#coding=utf-8
import re
```

```
names = ["name1", "_name", "2_name", "__name__"]
```

```
for name in names:
    ret = re.match("[a-zA-Z_]+[\w]*", name)
```

```
if ret:
    print("变量名 %s 符合要求" % ret.group())
else:
    print("变量名 %s 非法" % name)
```

运行结果:

```
变量名 name1 符合要求
变量名 _name 符合要求
变量名 2_name 非法
变量名 __name__ 符合要求
```

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")
print(ret.group())

ret = re.match("[1-9]?\\d", "33")
print(ret.group())

ret = re.match("[1-9]?\\d", "09")
print(ret.group())
```

运行结果:

```
7
33
0 # 这个结果并不是想要的, 利用$才能解决
```

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")
print(ret.group())

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

运行结果:

12a3g4
1ad12f23s34455ff66

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

5.匹配开头结尾

字符 功能

- ^ 匹配字符串开头
- \$ 匹配字符串结尾

示例 1: \$

需求: 匹配 163.com 的邮箱地址

```
#coding=utf-8
```

```
import re
```

```
email_list = ["xiaowang@163.com", "xiaowang@163.comheihei", ".com.xiaowang@qq.com"]
```

```
for email in email_list:
    ret = re.match("[\w]{4,20}@163\.com", email)
    if ret:
        print("%s 是符合规定的邮件地址,匹配后的结果是:%s" % (email, ret.group()))
    else:
        print("%s 不符合要求" % email)
```

运行结果:

```
xiaowang@163.com 是符合规定的邮件地址,匹配后的结果是:xiaowang@163.com
xiaowang@163.comheihei 是符合规定的邮件地址,匹配后的结果是:xiaowang@163.com
.com.xiaowang@qq.com 不符合要求
```

完善后

```
email_list = ["xiaowang@163.com", "xiaowang@163.comheihei", ".com.xiaowang@qq.com"]
```

```
for email in email_list:
    ret = re.match("[\w]{4,20}@163\.com$", email)
    if ret:
        print("%s 是符合规定的邮件地址,匹配后的结果是:%s" % (email, ret.group()))
    else:
        print("%s 不符合要求" % email)
```

运行结果:

xiaoWang@163.com 是符合规定的邮件地址,匹配后的结果是:xiaoWang@163.com
xiaoWang@163.comheihei 不符合要求
.com.xiaowang@qq.com 不符合要求

6. 匹配分组

字符	功能
	匹配左右任意一个表达式
(ab)	将括号中字符作为一个分组
\num	引用分组 num 匹配到的字符串
(?P<name>)	分组起别名
(?P=name)	引用别名为 name 分组匹配到的字符串

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")  
print(ret.group()) # 8
```

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

```
# 不正确的情况
```

```
ret = re.match("[1-9]?\d", "08")  
print(ret.group()) # 0
```

```
# 修正之后的
```

```
ret = re.match("[1-9]?\d$", "08")  
if ret:  
    print(ret.group())  
else:  
    print("不在 0-100 之间")
```

```
# 添加|
```

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

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

```
ret = re.match("[1-9]?\d$|100", "08")  
# print(ret.group()) # 不是 0-100 之间
```

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

示例 2: ()

需求: 匹配出 163、126、qq 邮箱

```
#coding=utf-8
```

```
import re
```

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

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

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

```
ret = re.match("\\w{4,20}@(163|126|qq)\\.com", "test@gmail.com")
if ret:
    print(ret.group())
else:
    print("不是 163、126、qq 邮箱") # 不是 163、126、qq 邮箱
```

不是以 4、7 结尾的手机号码(11 位)

```
import re
```

```
tels = ["13100001234", "18912344321", "10086", "18800007777"]
```

```
for tel in tels:
    ret = re.match("1\\d{9}[0-35-68-9]$", tel)
    if ret:
        print(ret.group())
    else:
        print("%s 不是想要的手机号" % tel)
```

提取区号和电话号码

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

(**[^-]***) 代表没有遇到小横杠-就一直进行匹配, 一直匹配下去

```
>>> 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>")  
print(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>")  
print(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>")  
print(ret.group())
```

```
# 因为 2 对<>中的数据不一致, 所以没有匹配出来
```

```
test_label = "<html>hh</htmlbalabala>"  
ret = re.match(r"<([a-zA-Z]*)>\w*</\1>", test_label)  
if ret:  
    print(ret.group())  
else:  
    print("%s 这是一对不正确的标签" % test_label)
```

运行结果:

```
<html>hh</html>  
<html>hh</htmlbalabala>  
<html>hh</html>  
<html>hh</htmlbalabala> 这是一对不正确的标签
```

示例 4: \number

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

```
#coding=utf-8
```

```
import re
```

```
labels = ["<html><h1>www.cskaoyan.com</h1></html>", "<html><h1>www.cskaoyan.com</h2></html>"]
```

```
for label in labels:
    ret = re.match(r"<(\w*)><(\w*)>.*</\2></\1>", label)
    if ret:
        print("%s 是符合要求的标签" % ret.group())
    else:
        print("%s 不符合要求" % label)
```

运行结果:

```
<html><h1>www.cskaoyan.com</h1></html> 是符合要求的标签
<html><h1>www.cskaoyan.com</h2></html> 不符合要求
```

示例 5: (?P<name>)(?P=name)

需求: 匹配出<html><h1>www.cskaoyan.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.cskaoyan.com</h1></html>")
print(ret.group())

ret =
re.match(r"<(P<name1>\w*)><(P<name2>\w*)>.*</(P=name2)></(P=name1)>
",
        "<html><h1>www.cskaoyan.com</h2></html>")
if ret:
    print("%s 是符合要求的标签" % ret.group())
else:
    print("%s 不符合要求" % label)
```

注意: (?P<name>)和(?P=name)中的字母 p 大写

7.re 模块的高级用法

search

需求：匹配出文章阅读的次数

```
#coding=utf-8
import re

ret = re.search(r"\d+", "阅读次数为 9999")
ret.group()
```

运行结果：

'9999'

findall

需求：统计出 python、c、c++相应文章阅读的次数

```
#coding=utf-8
import re

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

运行结果：

['9999', '7890', '12345']

Sub

```
import re
```

```
s = 'hello world, now is 2020/7/20 18:48, 现在是 2020 年 7 月 20 日 18 时 48 分。'
ret_s = re.sub(r'年|月', r'/', s)
ret_s = re.sub(r'日', r' ', ret_s)
ret_s = re.sub(r'时|分', r':', ret_s)
```

```
print(ret_s)
# findall 有问题
com = re.compile(r'\d{4}/[01]?[0-9]/[1-3]?[0-9]\s(0[0-9]|1[0-9]|2[0-4])\[0-5][0-9]')
ret = com.findall(ret_s)
print(ret)
```

改为下面没问题，在分组前面加?:

```
com = re.compile(r'\d{4}/[01]?[0-9]/[1-3]?[0-9]\s(?:0[0-9]|1[0-9]|2[0-4])\[0-5][0-9]')
```

```
ret = com.findall(ret_s)
print(ret)
```

```
# search 没问题
ret1 = re.search(r'\d{4}/[01]?[0-9]/[1-3]?[0-9]\s(0[0-9]|1[0-9]|2[0-4])\[0-5][0-9]', ret_s)
print(ret1.group())
```

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:

```
#coding=utf-8
import re

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

运行结果:

```
python = 998
```

方法 2:

```
#coding=utf-8
import re

def add(temp):
    strNum = temp.group()
    num = int(strNum) + 1
```



```
return str(num)
```

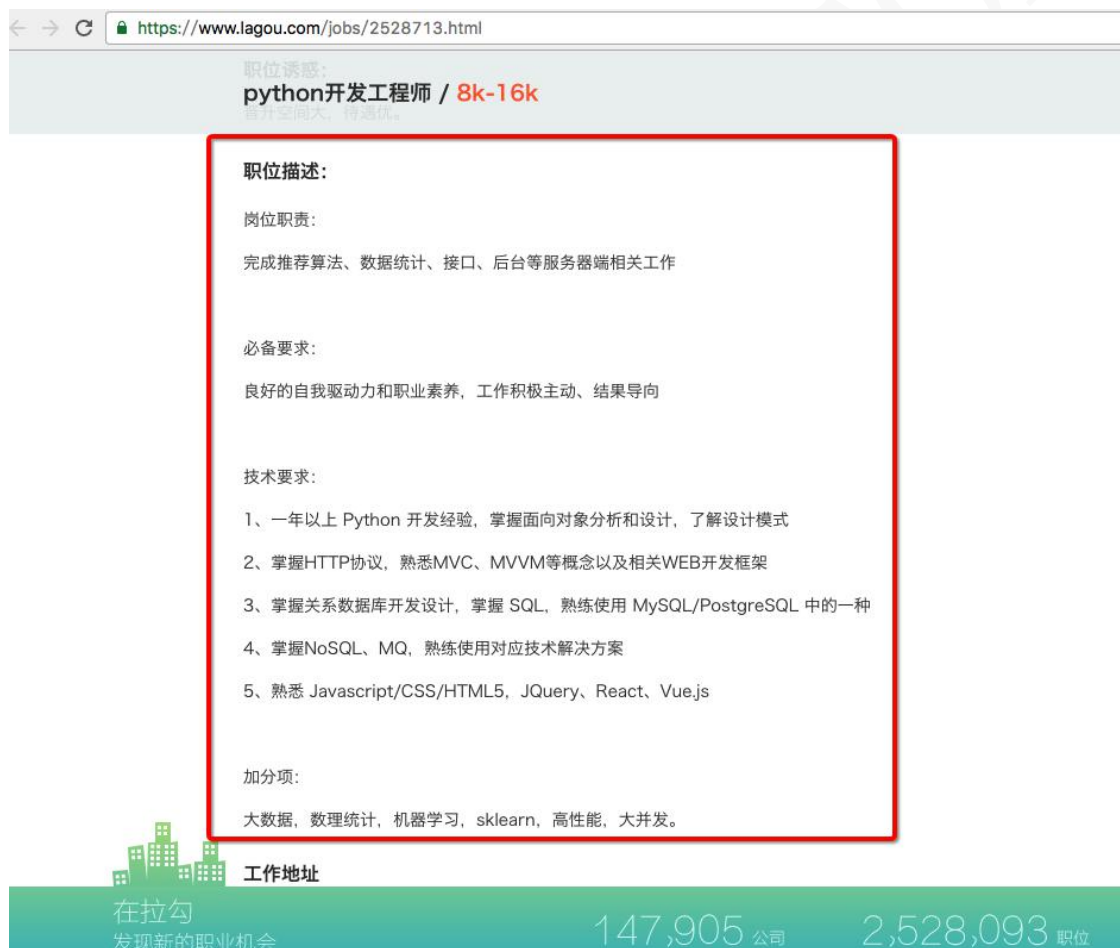
```
ret = re.sub(r"\d+", add, "python = 997")  
print(ret)
```

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

运行结果:

```
python = 998  
python = 100
```

练习



职位诱惑:
python开发工程师 / 8k-16k
薪资范围: 8k-16k

职位描述:

岗位职责:

完成推荐算法、数据统计、接口、后台等服务器端相关工作

必备要求:

良好的自我驱动力和职业素养, 工作积极主动、结果导向

技术要求:

- 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
- 2、掌握HTTP协议, 熟悉MVC、MVVM等概念以及相关WEB开发框架
- 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
- 4、掌握NoSQL、MQ, 熟练使用对应技术解决方案
- 5、熟悉 Javascript/CSS/HTML5, JQuery, React, Vue.js

加分项:

大数据, 数理统计, 机器学习, sklearn, 高性能, 大并发。

工作地址

在拉勾 发现新的职业机会

147,905 公司 2,528,093 职位

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>
</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>
</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
</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>
</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>

参考答案:

```
re.sub(r"<[^>]*>|&nbsp;|\n", "", test_str)
```

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)
```

运行结果:

```
['info', 'xiaoZhang', '33', 'shandong']
```

8.python 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。

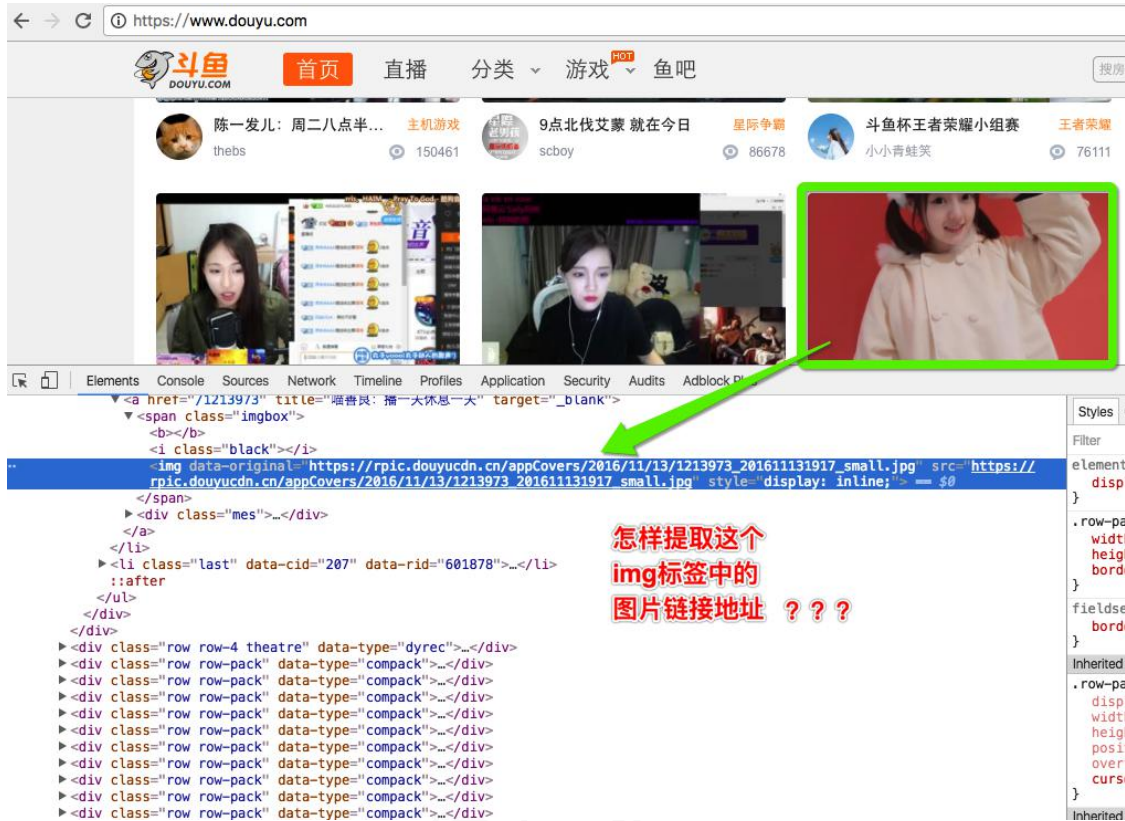
```
>>> s="This is a number 234-235-22-423"
>>> r=re.match(".*(\d+-\d+-\d+-\d+)",s)
>>> r.group(1)
'4-235-22-423'
>>> r=re.match(".*?(\d+-\d+-\d+-\d+)",s)
>>> r.group(1)
'234-235-22-423'
>>>
```

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，“`.*`”会从字符串的起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'
>>>
```

练一练



Elements Console Sources Network Timeline Profiles Application Security Audits AdBlock Plus

```

<a href="/1213973/" title="喂兽良：播一天休息一天" target="_blank">
  <span class="imgbox">
    <b></b>
    <i class="black"></i>
    
  </span>
  <div class="mes"></div>
</a>
<li class="last" data-cid="207" data-rid="601878"></li>
::after
</ul>
</div>
</div>
<div class="row row-4 theatre" data-type="dyrec"></div>
<div class="row row-pack" data-type="compact"></div>
<div class="row row-pack" data-type="compact"></div>
<div class="row row-pack" data-type="compact"></div>
<div class="row row-pack" data-type="compact"></div>
<div class="row row-pack" data-type="compact"></div>
<div class="row row-pack" data-type="compact"></div>
<div class="row row-pack" data-type="compact"></div>
<div class="row row-pack" data-type="compact"></div>
<div class="row row-pack" data-type="compact"></div>
</div>

```

怎样提取这个
img标签中的
图片链接地址 ???

字符串为:

```

```

请提取 url 地址

参考答案

```
re.search(r"https://.*?\.(jpg)", test_str)
```

如何更进一步呢，比如不要最后文件名字，或者只需要主域名

```
import re
```

```
test_str=''
```

```
str1=re.search(r"https://.*?\.(jpg)", test_str)

str2=str1.group()
print(str2)

str3=re.search(r"https://.*\/",str2)

print(str3.group())

str4=re.search(r"https://.*?\/",str2)

print(str4.group())
```

运行结果为:

```
https://rpic.douyucdn.cn/appCovers/2016/11/13/1213973_201611131917_small.
jpg
https://rpic.douyucdn.cn/appCovers/2016/11/13/
https://rpic.douyucdn.cn/
```

9.r 的作用

```
>>> mm = "c:\\a\\b\\c"
>>> 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，有了原生字符串，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
```

10