**day16笔记**

正则表达式：称为正则

什么是正则？

它也是一个字符串数据，主要用来检测其它字符串数据中的信息/内容；

我们在之后的学习和开发过程中会经常使用它；领域：web方向、爬虫方向(更广)

爬虫：

如果我们想要从某个网页上抓取数据，可能在这个过程中时不时需要对趴取的数据进行筛选/过滤；

此时可以用到正则；

还可以用来进行验证：邮箱、手机号、qq号、密码、url = 网站地址、ip

【总结】：

对于程序员而言：

定义的正则一般情况下都是看不懂的(复杂正则)；

但是我们需要对于简单正则自己能够定义/看懂

【归纳】：世界上有3样东西我们人类是看不懂的：

1).医生的处方

2).道士的符

3).程序员的正则

【强调】：

正则不是python语言独有的技术，python语言直到1.5版本才将正则表达式完成的整理/纳入进re模块中，

我们只需要导入re模块，然后就可以使用其中所有和正则相关的函数和属性了

**1.re模块中最常用的4个函数：**

**函数：match(regex,string,[flags=0])**

参数：

regex：就是正则表达式(内部定义了一套验证规则)

string：需要被验证的字符串数据

flags：模式/标志位，默认情况下(不显示定义) --> 不开启任何的模式

功能：

将string数据从头开始尝试匹配 【注意】：如果开头就不匹配，那么直接返回None值

如果匹配成功，那么就会返回给程序一个match对象

如果开头就不匹配，那么直接返回None值

flags的取值：

**re.I：忽略大小写**

match对象有5个常用的函数：

group()：返回匹配成功的数据值(理解：原串中的某部分子串信息)

start()：返回匹配成功的数据的起始索引

end()： 返回匹配成功的数据的结束索引

span()： 返回一个元祖对象，有两个元素组成；

第一个元素记录了匹配成功的数据的起始索引

第二个元素记录了匹配成功的数据的结束索引

groups()：返回所有子组的信息，以元祖的形式返回；如果没有进行正则的分组，即返回一个空元祖对象

【注意事项】：

1).正则表达式返回的索引值需要满足含头不含尾的特点

2).正则表达式验证的数据内容严格区分大小写

3).我们之后在定义正则表达式的时候，在它的第一个引号前面，都显示的追加一个r，无脑行为...（转义字符）

**import** re  
mo = re.match(**r'Www'**,**'www.baidu.com'**)  
print(mo,type(mo))  
print(re.match(**r'www'**,**'www.sina.com'**))  
print(re.match(**r'www'**,**'www.sina.com'**).group())  
print(re.match(**r'www'**,**'www.sina.com'**).start())  
print(re.match(**r'www'**,**'www.sina.com'**).end())  
print(re.match(**r'www'**,**'www.sina.com'**).span())  
print(re.match(**r'www'**,**'www.sina.com'**).groups())  
print(re.match(**r'WWW'**,**'www.sina.com'**))  
print(re.match(**r'WWW'**,**'www.sina.com'**,flags=re.I))

*#得到 None <class 'NoneType'>*

*<\_sre.SRE\_Match object; span=(0, 3), match='www'>*

*www*

*0*

*3*

*(0, 3)*

*()*

*None*

*<\_sre.SRE\_Match object; span=(0, 3), match='www'>*

第二种方式实现正则对数据的校验，如下：

**compile(regex,[flags=0])：函数**执行完毕以后返回给程序一个Pattern对象(理解：对象的内部封装了一套regex和flags)，可以再通过Pattern对象调用其match函数(此时的match函数只需要传递一个参数：string即可)

【注意】：

以上的函数中涉及的参数：regex、flags、string都和re中的match一样理解

pat = re.compile(**r'www'**,flags=re.I)  
print(pat,type(pat))  
print(pat.match(**'www.baidu.com'**))  
re.compile.match(**r'www'**，**'www.baidu.com'**,flags=re.I）\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

*#得到 re.compile('www', re.IGNORECASE) <class '\_sre.SRE\_Pattern'>*

*<\_sre.SRE\_Match object; span=(0, 3), match='www'>*

**函数：search(regex,string,[flags=0])**

参数：和match一样理解

功能：

从头开始尝试匹配，如果开头就匹配不成功，不会返回None值，会继续尝试往后匹配；

一旦匹配成功了，就直接返回一个match对象，后续就算还存在可以匹配成功的子串数据，

也不会再匹配了(直接无视)；如果直到最后都匹配不成功，返回一个None值；

【注意】：

由于search函数调用返回的是match对象，所以仍然可以调用5个常用的函数

print(re.match(**r'www'**,**'hahawww.baidu.com!!www.qfedu.comhehe'**))  
print(re.search(**r'www'**,**'hahawww.baidu.com!!www.qfedu.comhehe'**))  
print(re.search(**r'www'**,**'hahawww.baidu.com!!www.qfedu.comhehe'**).group())  
print(re.search(**r'www'**,**'hahawww.baidu.com!!www.qfedu.comhehe'**).start())  
print(re.search(**r'www'**,**'hahawww.baidu.com!!www.qfedu.comhehe'**).end())  
print(re.search(**r'www'**,**'hahawww.baidu.com!!www.qfedu.comhehe'**).span())  
print(re.search(**r'www'**,**'hahawww.baidu.com!!www.qfedu.comhehe'**).groups())  
print(re.search(**r'Www'**,**'hahawww.baidu.com!!www.qfedu.comhehe'**))  
print(re.search(**r'Www'**,**'hahawww.baidu.com!!www.qfedu.comhehe'**,flags=re.I))

*# None  
# <\_sre.SRE\_Match object; span=(4, 7), match='www'>  
# www  
# 4  
# 7  
# (4, 7)  
# ()  
# None  
# <\_sre.SRE\_Match object; span=(4, 7), match='www'>*

**compile()配合search()使用：**pat = re.compile(**r'www'**)  
mo = pat.search(**'Wwww.sina.com!!www.baidu.com!!www'**)  
print(mo)  
  
*#<\_sre.SRE\_Match object; span=(1, 4), match='www'>*

**函数：findall(regex,string,[flags=0])：**

参数：和match、search一样理解

功能：

将所有匹配成功的子数据(子串)，都存入到**列表中**返回；

如果一个都匹配不成功，那么返回一个**空列表**

lt = re.findall(**r'www'**,**'www.sina.com!!www.baidu.com'**)  
lt1 = re.findall(**r'WWW'**,**'www.sina.com!!www.baidu.com!!WWW'**,flags=re.I)  
print(lt,type(lt))  
print(lt1,type(lt1))  
  
*# ['www', 'www'] <class 'list'>  
# ['www', 'www', 'WWW'] <class 'list'>*

**compile()和findall()一起使用：**pat = re.compile(**r'www'**,flags=re.I)  
lt = pat.findall(**'www.sina.com!!www.baidu.com!!WWW'**)  
print(lt)

*# ['www', 'www', 'WWW']*

2.正则中元字符的使用：

**匹配单个字符(数字、英文、其它...)**

符号位：

**[]：表示一字符位**

**[9876123450]：表示一位，取值范围：[0,9]中间的任何一位值**

**[0-9]：表示一位，取值范围：[0,9]中间的任何一位值**

**\d：表示一位，取值范围：[0,9]中间的任何一位值**

**\D：对以上的\d进行取反,意味着：匹配除了数字字符以外的所有字符**

**[24680]：表示一位，取值范围：2、4、6、8、0中的任何一个值**

**[abcdefg]：表示一位，取值范围：a、b、c、d、e、f、g中的任何一个值**

**[a-z]：表示一位，取值范围：[a,z]中的任何一个值**

**[A-Z]：表示一位，取值范围：[A,Z]中的任何一个值**

**[0-9a-zA-Z\_]：表示一位，取值范围：0~9、a~z、A~Z以及\_中的任何一个值**

**\w：表示一位，取值范围：0~9、a~z、A~Z以及\_中的任何一个值**

**\W：对\w进行取反操作，意味着：匹配除了0~9、a~z、A~Z以及\_中的其它所有字符**

**.： 匹配除了换行符以外的所有字符**

**代码演示：**

reg = **r'[12345]'**print(re.match(reg,**'4'**))  
print(re.match(reg,**'42'**))  
print(re.match(reg,**'842'**))  
*# <\_sre.SRE\_Match object; span=(0, 1), match='4'>  
# <\_sre.SRE\_Match object; span=(0, 1), match='4'>  
# None*reg = **r'[0-9]'** *#两者作用一样reg = r'\d'*print(re.match(reg,**'842'**))  
*#<\_sre.SRE\_Match object; span=(0, 1), match='8'>*reg = **r'\d\d'**print(re.match(reg,**'842'**))  
*#<\_sre.SRE\_Match object; span=(0, 2), match='84'>*reg = **r'\D\d'**print(re.match(reg,**'a842'**))  
*#<\_sre.SRE\_Match object; span=(0, 2), match='a8'>*reg = **r'\w\d[a-z][0-9]'** *#一共要匹配四位数据*print(re.match(reg,**'\_6a842'**))  
print(re.findall(reg,**'\_6a842'**))  
print(re.findall(reg,**'\_6a842c7'**))  
*#<\_sre.SRE\_Match object; span=(0, 4), match='\_6a8'>  
# ['\_6a8']  
# ['\_6a8', '42c7']*

reg1 = **r'.'**print(re.match(reg1,**'\_6a842'**))  
*#<\_sre.SRE\_Match object; span=(0, 1), match='\_'>*reg2 = **r'..'**print(re.match(reg2,**'\_6a842'**))  
print(re.match(reg2,**'\n\_6a842'**))  
print(re.search(reg2,**'\n\_6a842'**))  
print(re.findall(reg2,**'\n\_6a842'**))  
*# <\_sre.SRE\_Match object; span=(0, 2), match='\_6'>  
# None  
# <\_sre.SRE\_Match object; span=(1, 3), match='\_6'>  
# ['\_6', 'a8', '42']*

**匹配锚字符(边界字符)：**

**^：**从字符串数据的头部开始匹配，在开启了**多行模式的情况下(re.M)**，它可以尝试匹配每一行的头部数据

**$：**从字符串数据的尾部开始匹配，在开启了多行模式的情况下(re.M)，它可以尝试匹配每一行的尾部数据

**\A：**从字符串数据的头部开始匹配，在开启了多行模式的情况下(re.M)，它没有多行的概念，还是匹配第一行的头部数据

**\Z：**从字符串数据的尾部开始匹配，在开启了多行模式的情况下(re.M)，它没有多行的概念，还是匹配最后一行的尾部数据

print(re.search(**r'^www'**,**'hahawww.qfedu.comhehe'**)) *#search改成match效果一样*print(re.search(**r'^haha'**,**'hahawww.qfedu.comhehe'**)) *#search改成match效果一样*print(re.search(**r'hehe$'**,**'hahawww.qfedu.comhehe'**))  
print(re.search(**r'.com$'**,**'hahawww.qfedu.comhehe'**))  
*# None  
# <\_sre.SRE\_Match object; span=(0, 4), match='haha'>  
# <\_sre.SRE\_Match object; span=(17, 21), match='hehe'>  
# None*print(re.findall(**r'hehe'**,**'hahawww.heheqfedu.comhehe'**))  
print(re.findall(**r'hehe$'**,**'hahawww.heheqfedu.comhehe'**))  
*# ['hehe', 'hehe']  
# ['hehe']*print(**'8'**\*60)  
print(re.search(**r'\Awww'**,**'hahawww.qfedu.comhehe'**))  
print(re.search(**r'\Ahaha'**,**'hahawww.qfedu.comhehe'**))  
print(re.search(**r'hehe\Z'**,**'hahawww.qfedu.comhehe'**))  
print(re.search(**r'.com\Z'**,**'hahawww.qfedu.comhehe'**))  
*# None  
# <\_sre.SRE\_Match object; span=(0, 4), match='haha'>  
# <\_sre.SRE\_Match object; span=(17, 21), match='hehe'>  
# None*

**演示匹配边界字符首尾的区别：**

str1 = **'''laoguo is a good man  
laoguo is a nice man  
laoguo is a cool man  
laoguo is real man'''**print(re.findall(**r'^laoguo'**,str1,flags=re.M))  
print(re.findall(**r'\Alaoguo'**,str1,flags=re.M))  
print(re.findall(**r'man$'**,str1,flags=re.M))  
print(re.findall(**r'man\Z'**,str1,flags=re.M))  
*# ['laoguo', 'laoguo', 'laoguo', 'laoguo']  
# ['laoguo']  
# ['man', 'man', 'man', 'man']  
# ['man']*

**补充：**

**\b：尝试匹配边界(左侧、右侧)数据，如果一旦满足返回对象(match、list)**

**\B：先舍弃规定边界的数据，然后一定满足从左侧开始匹配数据，...**

*#\b相当于^ 和 $ 两者的作用*

print(re.search(**r'\bne'**,**'never'**))  
print(re.search(**r'ne\b'**,**'never'**))  
print(re.search(**r'er\b'**,**'neverer'**))  
*# <\_sre.SRE\_Match object; span=(0, 2), match='ne'>  
# None  
# <\_sre.SRE\_Match object; span=(5, 7), match='er'>*

*\B相当于除去边界之后再匹配*print(re.search(**r'er\B'**,**'never'**))  
print(re.search(**r'er\B'**,**'nerver'**))  
print(re.search(**r'er\B'**,**'ernerver'**))  
print(re.search(**r'\Ber'**,**'ernever'**))  
*# None  
# <\_sre.SRE\_Match object; span=(1, 3), match='er'>  
# <\_sre.SRE\_Match object; span=(0, 2), match='er'>  
# <\_sre.SRE\_Match object; span=(5, 7), match='er'>*

**匹配多个字符：**

以下的一些x,y,n等都是变量名

分类：

**1).模糊匹配：**

x?：表示0个或者1个 取值范围：[0,1]

x+：表示1个或者多个 取值范围：[1,无穷大)

x\*：表示0个或者多个 取值范围：[0,无穷大)

【注意事项】：

关于以上模糊匹配涉及到的三个符号(?、+、\*)都满足了正则中的贪婪匹配的行为/特点；

理解：在确保匹配的前提下，尽可能多的返回数据

思考：如何取消贪婪行为？

理解：最终希望得到的是，在确保匹配的前提下，尽可能少的返回数据

代码体现：在以后的3个符号后面都显示的再追加一个?即可

str1 = **"aaaaaaacccaaaaaaaaa"**print(re.search(**r'a?'**,str1).group())  
print(re.search(**r'a??'**,str1).group())  
print(re.search(**r'a+'**,str1).group())  
print(re.search(**r'a+?'**,str1).group())  
print(re.search(**r'a\*'**,str1).group())  
print(re.search(**r'a\*?'**,str1).group())  
*# a  
#  
# aaaaaaa  
# a  
# aaaaaaa  
#*  
print(re.findall(**r'a?'**,str1))  
print(re.findall(**r'a??'**,str1))  
print(re.findall(**r'a+'**,str1))  
print(re.findall(**r'a+?'**,str1))  
print(re.findall(**r'a\*'**,str1))  
print(re.findall(**r'a\*?'**,str1))  
*#只要有0的都会有结尾符号和匹配不上的空字符  
# ['a', 'a', 'a', 'a', 'a', 'a', 'a', '', '', '', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', '']  
# ['', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '']  
# ['aaaaaaa', 'aaaaaaaaa']  
# ['a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a']  
# ['aaaaaaa', '', '', '', 'aaaaaaaaa', '']  
# ['', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '']*

**#需求：尝试匹配的内容为：laoguo ... man**str2 = **'laoguo is a good man laoguo is a nice man laoguo is a super man'**regex1= **r'laoguo.\*man'** *#有贪婪行为，中间尽可能一次性多的返回数据*regex2 = **r'laoguo.\*?man'** *#取消其贪婪行为，每个数据里面数据尽可能少*lt1 = re.findall(regex1,str2)  
print(lt1)   
*#['laoguo is a good man laoguo is a nice man laoguo is a super man']*lt2 = re.findall(regex2,str2)print(lt2)  
*#['laoguo is a good man', 'laoguo is a nice man', 'laoguo is a super man']*

**2).精确匹配：**

n{x}： 将n匹配x次

n{x,}： 将n最少有x次，最多无穷大 范围：[x,无穷多)

n{x,y}： 将n最少有x次，最多有y次 范围：[x,y]

**演示精确匹配**

str3 = **'aaaaaaacccaaaaaaaaaa'**print(re.findall(**r'a{5}'**,str3))  
print(re.findall(**r'a{3}'**,str3))  
print(re.findall(**r'a{3,}'**,str3)) *#会尽可能返回多的*print(re.findall(**r'a{3,}?'**,str3)) *#取消贪婪行为*print(re.findall(**r'a{8,10}'**,str3))  
print(re.findall(**r'a{8,10}?'**,str3))  
*# ['aaaaa', 'aaaaa', 'aaaaa']  
# ['aaa', 'aaa', 'aaa', 'aaa', 'aaa']  
# ['aaaaaaa', 'aaaaaaaaaa']  
# ['aaa', 'aaa', 'aaa', 'aaa', 'aaa']  
# ['aaaaaaaaaa']  
# ['aaaaaaaa']*

**不使用正则验证QQ号码：  
要求：1).只能都是数字字符  
 2).不能以0开头  
 3).长度：5~12位**qq = **'12345678'  
if** len(qq) < 5 **or** len(qq) > 12:  
 **raise** Exception(**'qq号码长度不合法...'**)  
**elif** qq[0] == **'0'**:  
 **raise** Exception(**'qq号码不能以0开头...'**)  
*#如果程序能够顺利执行到此处，说明以上两个验证没有问题，验证是否全是数字字符*num = int(qq)  
print(**'qq号码合法，号码为：%d'** %num)

**使用正则验证qq号码：  
import** re

qq = **'12345678'**regex = **r'^[1-9]\d{4,11}$'**pat = re.compile(regex)  
res = pat.match(qq)  
**if** res:  
 print(**'qq号码合法，号码为：%s'** %qq)  
**else**:  
 print(**'qq号码不合法...'**)

**3.正则中的分组：**

**分组：**

在正则中的定义体现就是使用(...)进行分组操作，理解为分了一个子组

好处：

1).如果正则表达式的逻辑比较复杂，意味着代码比较多，如果在适当的部分进行分组就可以提高代码的阅读性(更有层次感)

2).一旦进行了分组操作，在正则表达式的后半部分内容中很多可能需要使用到子组中的内容； 一旦引用了子组，那么这两部分的内容(值)就可以保持一致了

phone = **'62586666'**regex = **r'^[1-9]\d{3}(\d)\1{3}$'** *#'.....(..)..(..).(.)...\1'*

print(re.match(regex,phone))  
print(re.match(regex,phone).groups())  *#返回子组的内容*  
print(re.match(regex,phone).group()) *#和下面效果一样*  
print(re.match(regex,phone).group(0))  *#和上面效果一样*  
print(re.match(regex,phone).group(1)) *#第一个子组里面的元祖元素*  
*# <\_sre.SRE\_Match object; span=(0, 8), match='62586666'>  
# ('6',)  
# 62586666  
# 62586666  
# 6*

**以下的代码有问题：  
出现了下标越界的异常：IndexError  
原因是正则中只有1个子组，没有index=2这一组，所以报错了...***# print(re.match(regex,phone).group(2))*

phone = **'021-53437788'**regex = **r'^(?P<one>\d{3})(?P<two>-)(?P<three>\d{8})$'**

mo = re.search(regex,phone)  
print(mo.groups()) *#('021', '-', '53437788')*print(mo.group()) *#021-53437788*print(mo.group(0)) *#021-53437788*print(mo.group(1)) *#021*print(mo.group(2)) *#-*print(mo.group(3)) *#53437788*print(mo.group(**'one'**)) *#021*print(mo.group(**'two'**)) *#-*print(mo.group(**'three'**)) *#53437788*

**4.正则中的替换和切割操作**

**替换：**

sub(regex,repl,string,[count],[flags=0])：替换字符数据，返回字符串(已经被替换完成后的内容)

subn(regex,repl,string,[count],[flags=0])：替换字符数据，返回元祖对象，此对象有两个元素；

第一个元素记录了替换以后的字符串内容，

第二个元素记录了被替换的次数(count)

参数：

regex：正则规则(字符串)

repl：替换成的字符串内容

string：原串数据

count：次数，默认全部替换

str1 = **'i love beijing beijing beijing so much'**regex = **r'(beijing)'**str2 = re.sub(regex,**'shanghai'**,str1)  
print(str2,type(str2))  
tp = re.subn(regex,**'shanghai'**,str1,2) *#2改成4也不会报错，就会全部替换*print(tp,type(tp))  
*# i love shanghai shanghai shanghai so much <class 'str'>  
# ('i love shanghai shanghai beijing so much', 2) <class 'tuple'>***需求：模拟让游戏世界变得和谐(使用正则)**regex = **r'CNM|MB|SB|NC|WQNMLGB|TMD|NND'**game = **'WQNMLGB!!连装备都不会出...小学生！！SB...'**s = re.sub(regex,**'\*\*\*'**,game)  
print(s)  
*# \*\*\*!!连装备都不会出...小学生！！\*\*\*...*

**切割：**

**split(regex,string)：返回的一个列表对象**str3 = **'i love shanghai so much'**regex = **r' +'** *#切割空格 +表示一个到多个 有贪婪行为，把空格全切了*lt = re.split(regex,str3)  
print(lt)  
*#['i', 'love', 'shanghai', 'so', 'much']*str4 = **'i love shanghai so much'**regex = **r' +?'** *#切割空格 +表示一个到多个 取消贪婪行为，能少切就少切*lt = re.split(regex,str4)  
print(lt)  
*#['i', '', '', '', '', '', '', 'love', '', '', '', '', '', '', '', 'shanghai', '', '', '', 'so', '', 'much']*

str4 = **'dasfas23141sfa123dsafas13dasdfa1231241\_\_\_\_3241234^&\*&(dsafa**

**214331asdfa'**regex = **r'\d+?'**lt = re.split(regex,str4)  
print(lt)  
*# ['dasfas', '', '', '', '', 'sfa', '', '', 'dsafas', '', 'dasdfa', '', '', '', '', '', '', '\_\_\_\_', '', '', '', '', '', '', '^&\*&(dsafa', '', '', '', '', '', 'asdfa']*regex1 = **r'\d+'**lt1 = re.split(regex1,str4)  
print(lt1)  
*# ['dasfas', 'sfa', 'dsafas', 'dasdfa', '\_\_\_\_', '^&\*&(dsafa', 'asdfa']*

**补充案例：练习和替换相关操作  
需求：将如下代码中的#替换为-  
考虑：  
1).一个#替换成为一个-  
2).一堆#替换成为一个-  
'''**str5 = **'dsaf######32141asf#####dsafa#########()!,.\_\_\_######21341##'**regex1 = **r'#'**str6 = re.sub(regex1,**'-'**,str5)  
print(str6)

*#dsaf------32141asf-----dsafa---------()!,.\_\_\_------21341--*

regex2 = **r'#+'**tp = re.subn(regex2,**'-'**,str5)  
print(tp)

*#('dsaf-32141asf-dsafa-()!,.\_\_\_-21341-', 5)*

**5.finditer()函数的使用：**

函数：finditer(regex,string,[flags=0])：

参数：和match、search、findall一样理解

功能：

将所有匹配成功的数据封装为一个一个的match对象，

然后以iterator(迭代器对象)返回

str1 = **'i hate japan japan japan so much'**regex = **'japan'**it = re.finditer(regex,str1)  
  
 **iterator(迭代器对象)不能直接使用内置函数len()执行查看其容量大小，会报错：TypeError  
但是我们可以先将其转换为容器对象(list、tuple...)可以被len()所执行了；***# print(len(it))*print(len(list(it))) *#3*

**使用遍历的思想去访问iterator中的元素  
  
for** i **in** it:  
 print(i)  
 print(type(i))  
 print(i.group())

*# <\_sre.SRE\_Match object; span=(7, 12), match='japan'>  
# <class '\_sre.SRE\_Match'>  
# japan  
# <\_sre.SRE\_Match object; span=(13, 18), match='japan'>  
# <class '\_sre.SRE\_Match'>  
# japan  
# <\_sre.SRE\_Match object; span=(19, 24), match='japan'>  
# <class '\_sre.SRE\_Match'>  
# japan*

**迭代器对象中的内容只能被使用一次，不可逆，否则报错：StopIteration***# print(next(it))* **思考：可不可以使用while循环来遍历iterator对象呢？  
while** 1:  
 **try**:  
 mo = next(it)  
 print(mo)  
 print(mo.group())  
 **except**:  
 **break**

#结果和for......in..一样