正则表达式

1. 动机

【1】 文本处理已经成为计算机的常见工作之一

【2】 对文本内容的搜索，定位，提取逻辑比较复杂

【3】 为了快速方便的解决上述问题产生了正则表达式技术

2. 定义 ：即文本的高级匹配模式，提供搜索，替换等功能。其本质是由一

系列字符和特殊符号构成的字符串，这个字符串即正则表达式。

3. 特点：\* 方便文本处理

\* 支持语言众多

\* 使用灵活，变化多样

4. python--》re模块

re.findall(pattern,string)

功能：提取所有正则表达式匹配内容

参数：pattern 正则表达式

string 目标字符串

返回值： 提取到的内容列表

5. 元字符使用

【1】 普通字符

匹配规则 ： 每个字符匹配对应字符

In [6]: re.findall('ab','abcdeabca')

Out[6]: ['ab', 'ab']

【2】 或

元字符 ： |

匹配规则： 匹配|两侧任意一个正则表达式

In [8]: re.findall('ab|ef','abcdefghab')

Out[8]: ['ab', 'ef', 'ab']

【3】 匹配单个字符

元字符： .

匹配规则：匹配除换行外的任意一个字符

In [11]: re.findall('H.Y','H1Y,HAY,H噶Y')

Out[11]: ['H1Y', 'HAY', 'H噶Y']

【4】匹配重复

元字符 ： \*

匹配规则：匹配前面的字符重复0次或者多次

In [18]: re.findall('Ha\*','Haaab')

Out[18]: ['Haaa']

【5】 匹配重复

元字符 ： +

匹配规则：匹配前面的字符出现1次或多次

In [19]: re.findall('Ha+','Haaab')

Out[19]: ['Haaa']

【6】 匹配重复

元字符：？

匹配规则：重复前面的字符出现0次或1次

In [29]: re.findall('H.?','Hi,Hey. How are you H')

Out[29]: ['Hi', 'He', 'Ho', 'H']

【7】匹配重复

元字符：{n}

匹配规则： 匹配前面字符出现n 次

In [30]: re.findall('H.{2}','Hi,Hey. How are you H')

Out[30]: ['Hi,', 'Hey', 'How']

【8】 匹配重复

元字符： {m,n}

匹配规则： 重复前面的字符出现m—n次

In [32]: re.findall('fo{2,4}','foo,food,fooooooo')

Out[32]: ['foo', 'foo', 'foooo']

【9】 匹配开始位置

元字符 ： ^

匹配规则： 匹配目标字符串的开始位置

In [40]: re.findall('^Jame','Jame,xxxxx')

Out[40]: ['Jame']

【10】 匹配结束位置

元字符： $

匹配规则：匹配目标字符串的结束位置

In [43]: re.findall('Jame$','Hi,Jame')

Out[43]: ['Jame']

【11】匹配字符集

元字符： [字符集]

匹配规则：匹配字符集中任意一个字符

写法：[abc!#%\_] [a-z] [A-Z] [0-9] [\_a-z0-9]

In [45]: re.findall('[A-Z][\_a-z]\*',"This is A Test,Joy")

Out[45]: ['This', 'A', 'Test', 'Joy']

【12】 字符集取反

元字符 ： [^…]

匹配规则：匹配除了字符集中的任意一个字符

In [50]: re.findall('[^ ,]+',"This is A Test,Joy")

Out[50]: ['This', 'is', 'A', 'Test', 'Joy']

【13】 匹配任意(非)数字字符

元字符 ： \d \D

匹配规则：\d 🡪任意数字字符 [0-9]

\D 🡪任意非数字字符 [^0-9]

In [52]: re.findall('1\d{10}',"13665668588")

Out[52]: ['13665668588']

【14】 匹配任意(非)普通字符

元字符： \w \W

匹配规则： \w 🡪 任意普通字符

\W 🡪 任意非普通字符

普通字符： 数字 字母 下划线 普通的汉字

In [54]: re.findall('\w+',"PORT#1035,Error 44%下降")

Out[54]: ['PORT', '1035', 'Error', '44', '下降']

【15】匹配（非）空字符

元字符： \s \S

匹配规则： \s 🡪 匹配任意空字符

\S 🡪 匹配任意非空字符

空字符： 空格 \r \n \t \v \f

In [57]: re.findall('\w+\s+\w+',"helle world")

Out[57]: ['helle world']

【16】 匹配字符串位置

元字符： \A \Z

匹配规则： \A 匹配字符串开头位置 ^

\Z 匹配字符串结尾位置 $

【17】匹配(非)单词边界位置

元字符： \b \B

匹配规则： \b 🡪 匹配单词边界位置

\B 🡪 非单词边界位置

单词边界：普通字符和非普通字符的交接为单词边界

In [62]: re.findall(r'\Bis\b',"This is a test")

Out[62]: ['is']

总结：匹配单个字符 . […] [^…] \d \D \w \W \s \S

匹配重复 \* + ? {n} {m,n}

匹配位置 ^ $ \A \Z \b \B

其他 | () \

6. 正则表达式转义

特殊符号 ： . \* + ？ $ ^ [] () | \

正则表达式需要匹配特殊字符时会使用转义

e.g. 匹配字符 . 用 \.

通常在编程语言中，为了防止编程语言对字符串的转义，写正则表达式字符串用raw字符串，即前面加r

7. 贪婪模式和非贪婪模式

贪婪模式：正则表达式的重复匹配默认总是尽可能向后匹配更多的内容。 \* + ？ {m,n}

非贪婪（懒惰模式）：尽量少的向后匹配内容

贪婪 –》 非贪婪 \*？ +？ ?? {m,n}?

In [88]: re.findall(r'H.\*?\b',"Hi Hey,How are you")

Out[88]: ['Hi', 'Hey', 'How']

8． 正则表达式分组

【1】 定义：使用() 可以为正则表达式建立内部分组，子组为正则表达式一部分，可以视为内部整体。

【2】 注意事项

* 有子组的正则表达式，任然是整体匹配内容，子组需在整体能匹配到内容的前提下有作用
* 一个正则表达式中可以有多个子组，一般由外到内，由左到右计数为第一，第二，第三…..
* 子组不宜出现嵌套过多，或者重叠

【3】 作用

* 作为内部整体，可以改变某些元字符操作对象

re.search(r'\w+@\w+\.(com)?\.?(com|cn)',"Wang@163.com.cn").group()

'Wang@163.com.cn'

* 子组在匹配到内容的前提下，可以单独提取对应内容

re.search(r'(http|https|ftp|file)://\S+',"https://www.baidu.com").group(1)

'https'

【4】 子组命名

格式: (?P<name>pattern)

re.search(r'(?P<pig>ab)cdef',"abcdefgh").group('pig')

'ab'

作用：名字可以表达一定的含义，也可以通过名称更方便获取某个子组对应内容。

9. 正则表达式匹配原则

【正确性】 能够正确的匹配出目标字符串

【排他性】 除了要匹配的目标内容不会匹配出其他无用数据

【全面性】 全面考虑目标字串的规则情况，不遗漏

python re模块

regex = re.compile(pattern,flags=0)

功能：生成正则表达式对象

参数： pattern 正则表达式

flags 功能标志位

返回值： 返回正则表达式对象

re.findall(pattern,string,flags=0)

功能:使用正则匹配目标字符串所有内容

参数：pattern 正则

string 目标字符串

flags 标志位

regex.findall(string,pos,endpos)

功能:使用正则匹配目标字符串所有内容

参数： string 目标字符串

re.split(pattern,string,flags)

功能： 使用正则表达式匹配内容切割字符串

参数： pattern 正则

string 目标字符串

返回值：返回一个列表，为切割后的内容

In [117]: re.split(r'\s+',"hello world nihao china")

Out[117]: ['hello', 'world', 'nihao', 'china']

re.sub(pattern,replace,string,max,flags)

功能：使用指定字符串替换正则匹配到的内容

参数：replace 指定字符串

string 目标字符串

max 最多替换几处

返回值：替换后的字符串

In [119]: re.sub(r'\s+','##',"hello world nihao china",2)

Out[119]: 'hello##world##nihao china'

re.finditer(pattern,string,flags)

功能： 使用正则匹配目标字符串内容

参数： pattern正则

string 目标字符串

返回值 ： 迭代对象

re.match(pattern,string,flags)

功能：匹配目标字符串开头位置

参数：同findall

返回： 匹配到的内容对应的match对象

In [138]: re.match(r"\d+","88,hi")

Out[138]: <\_sre.SRE\_Match object; span=(0, 2), match='88'>

re.search(pattern,string,flags)

功能：匹配目标字符串第一个匹配内容

参数：同findall

返回： 匹配到的内容对应的match对象

In [141]: re.search(r"\d+","xxxx,75,43")

Out[141]: <\_sre.SRE\_Match object; span=(5, 7), match='75'>

re.fullmatch(pattern,string,flags)

功能：匹配目标字符串全部内容

参数：同findall

返回： 匹配到的内容对应的match对象

In [147]: re.fullmatch(r"\d+","1234567")

Out[147]: <\_sre.SRE\_Match object; span=(0, 7), match='1234567'>

compile对象的属性：

groups ： 表示有多少子组

groupindex ： 获取捕获组和组序号对应字典

pattern ： 正则表达式

flags ： 表示为值

match object 属性和方法

* 每个match对象代表一个匹配内容的信息

属性变量 ：pos 匹配目标字符串的开始位置

endpos 匹配目标字符串的结束位置

re 正则表达式

string 目标字符串

lastgroup 最后一组组名

lastindex 最后一组第几组

属性方法：start() 匹配到内容的起始位置

end() 匹配到内容的结束位置

span() 匹配到内容的起止位置

groups() 获取每个子组匹配到的内容

groupdict() 获取捕获组字典，组名为键对应内容为值

group(n)

功能 ： 获取match对象对应的内容

参数： 默认为0，表示获取匹配到的所有内容

如果为子组序列号或者名称则表示获取对应子组的匹配内容

flags的使用

作用：丰富正则表达式匹配规则，re调用匹配函数通常含有该参数

I == IGNORECASE 匹配时忽略字母大小写

In [177]: re.findall(r'hi',"hi,Hi",flags=re.I)

Out[177]: ['hi', 'Hi']

S == DOTALL 使 . 可以匹配换行

In [179]: re.findall(r'.+',"hello\nworld",re.S)

Out[179]: ['hello\nworld']

M == MULTILINE 使 ^ $可以匹配每一行的开头结尾位置

In [180]: s ='''hello

world'''

In [183]: re.findall(r'^world',s,re.M)

Out[183]: ['world']

同时使用多个flag ： flags = re.I | re.S