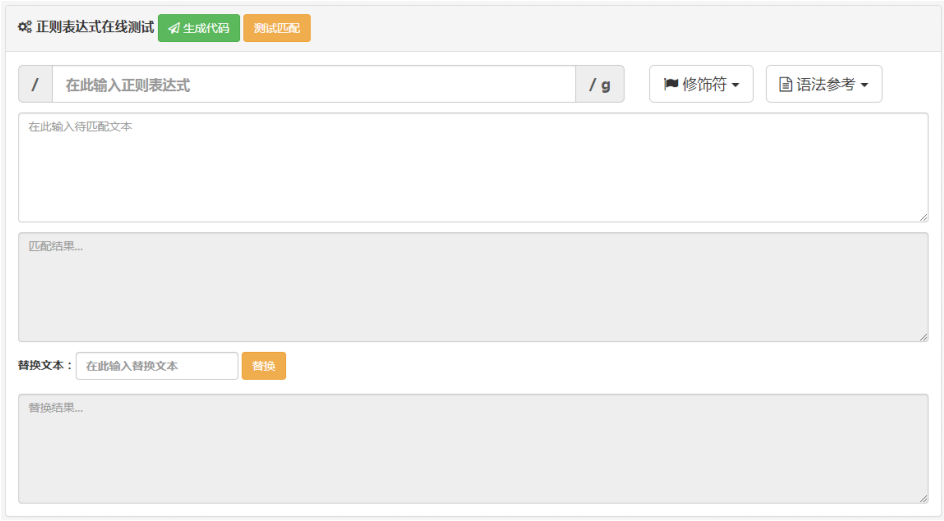
一、写在前面

- 在学习和使用正则表达式的时候，重要的并不是知道多少个特殊字符，而是会不会运用它们去解决实际问题
- **基本用途：搜索和替换**
- 正则表达式是一些用来匹配和处理文本的字符串
- 语法是正则表达式最容易掌握的部分，真正的挑战是学会如何运用语法把实际问题分解为一系列正则表达式并最终解决
- 在编写正则表达式的时候，同一个问题往往会有多种解决方案
- 验证某个模式能不能获得预期的匹配结果并不困难，但如何验证它不会匹配到不想到的东西可就没那么简单了
- **正则表达式在线测试网页**

> https://c.runoob.com/front-end/854/



二、语法

1. 匹配单个字符

- 正则表达式是区分字母大小写的
- 如果要忽略值的大小写，需要使用 `re.I`
- `.` 可以匹配任何一个字符、字母、数字、甚至是 . 本身
- 在同一个正则表达式里允许使用多个 . 字符，既可以连续出现，也可以间隔出现在模式的不同位置
- 使用 `\` 对特殊字符（元字符）进行转义，`\` 本身也是一个元字符

2. 匹配一组字符

- 使用 `[]` 定义字符集合
- 在定义一个字符区间的时候，一定要避免让这个区间的尾字符小于它的首字符（根据 ASCII 字符的大小）
- 在字符集合以外的地方，`-` 只是一个普通字符，只能与 `-` 本身相匹配，`-` 字符不需要做转义
- `^` 对一个字符集合进行取非匹配
- `^` 作用于给定字符集合里的所有字符和字符区间，而不是仅限于紧跟在 `^` 后面的那一个字符或字符区间

3. 使用元字符

元字符	说明
[\b]	回退（删除）一个字符
\f	换页符
\n	**换行符**
\r	**回车符**
\t	**制表符**

元字符	说明
\v	垂直制表符
\d	**任何一个数字字符，等价于[0-9]**
\D	**任何一个非数字字符**
\w	**等价于[a-zA-Z0-9_]**
\W	**任何一个非字母数字或非下划线字符**
\s	**等价于[\f\n\r\t\v]**
\S	**任何一个非空白字符**
\E	结束 \L 或 \U 转换
\l	把下一个字符转换为小写
\L	把 \L 到 \E 之间的字符全部转换为小写
\u	把下一个字符转换为大写
\U	把 \U 到 \E 之间的字符全部转换为大写
(?=)	**正向前查找**
(?!)	**负向前查找**
(?<=)	**正向后查找**
(?<!)	**负向后查找**
(?m)	分行模式匹配

- `\r\n` 是 Windows 所使用的文本行结束标签
- 在 Linux 上匹配空白行只使用 `\n\n` 即可
- 同时适用于 Windows 和 Linux 的正则表达式应该包含一个可选的 `\r` 和一个必须被匹配的 `\n`
- 十六进制（逢16进1）数值要用前缀 `\x` 来给出
- 八进制（逢8进1）数值要用前缀 `\0` 来给出
- `|` 表示逻辑 `或` 操作符

4. 重复匹配

- `+` 匹配一个或多个字符（或字符集合）
- `*` 匹配零个或多个字符（或字符集合）
- `?` 匹配零个或一个字符（或字符集合）
- 如果打算同时使用 `[]` 和 `?` ，千万记得应该把 `?` 放在字符集合的外面
- 重复次数要用 `{}` 来给出，把数值写在里面，重复的次数可以是0
- `{m,n}` 为匹配次数设定一个最小值和最大值
- `{m,}` 为匹配次数设定一个最小值

- `{,n}` 为匹配次数设定一个最大值
- 懒惰型元字符的写法，只要给贪婪型元字符加上一个 `?` 后缀即可

5. 位置匹配

- 位置匹配解决在什么地方进行字符串匹配操作的问题
- 由限定符 `\b` 指定单词的边界，用来匹配一个单词的开头和结尾
- `\b` 匹配且只匹配一个位置，不匹配任何字符
- 想表明不匹配一个单词边界，使用 `\B`
- `^` 定义字符串开头，`$` 定义字符串结尾
- `^\s*` 匹配一个字符串的开头位置和随后的零个或多个空白字符
- `\s*$` 匹配一个字符串的结尾位置零个或多个空白字符
- 如果要使用分行匹配模式，需要使用 `re.S`

6. 子表达式

- 使用 `()` 来把子表达式当做一个独立元素
- 子表达式允许多重嵌套，但在实际工作中一个遵循适可而止的原则
- 绝大多数嵌套子表达式都没有看上去那么复杂
- 把必须匹配的情况考虑周全并写出一个匹配结果符合预期的正则表达式很容易，但把不需要匹配的情况也考虑周全并确保它们都将被排除在匹配结果之外往往要困难得多

7. 回溯引用（前后一致匹配）

- 回溯引用允许正则表达式模式引用前面的匹配结果
- 回溯引用指的是模式的后半部分引用在前半部分中定义的子表达式
- 可以把回溯引用想象成变量
- Java 和 Python 将返回一个包含 `group` 的数组匹配对象
- 回溯引用只能用来引用模式里的子表达式
- 回溯引用匹配通常从 `1` 开始计数（ `\1` ）
- 如果子表达式的相对位置发生了变化，整个模式也许就不能再完成原来的工作
- **查找时，使用** `\n` **；替换时，使用** `$n`
- 在对文本进行重新排版时，把文本分解成多个子表达式的做法往往非常有用，可以对文本的排版效果做出更精准的控制
- 需要用到 `re.finditer(pattern,string,flags)`

```
list_temp = []
for item in re.finditer(r'(?<=<h([1-6])>).*?(?=</h\1>)','<h1>一级标题</h1><h2>
二级标题</h2><h3>这里是错误</h4>'):
    list_temp.append(item.group())

list_temp

# ['一级标题', '二级标题']
```

8. 前后查找

- 在同一个搜索模式里可以使用多个前后查找表达式
- 向前查找指定了一个必须匹配但不在结果中返回的模式，实际就是一个以 `?=` 开头的子表达式
- 向后查找指定了一个必须匹配但不在结果中返回的模式，实际就是一个以 `?<=` 结尾的子表达式
- 向前查找模式的长度是可变的，可以包含 `\` 和 `+` 之类的元字符；向后查找模式只能是固定长度
- 前后查找必须用 `!` 替换掉 `=` 来取非

三、在 Python 中使用

1. re 库

（1）re.findall(pattern,string,flags)

- 返回的结果是列表，如果没有匹配到结果，返回空列表

```
import re

result = re.findall('工号是(.*?)，','我的名字是张三，工号是110110，职业是法外狂徒；他的名字是李四，工号是120120，职业是医生')

# ['110110', '120120']
```

（2）re.search(pattern,string,flags)

- 要得到匹配结果，需要通过 `group()` 方法获取值，如果没有匹配到结果，返回 `None`
- 只有在 `group()` 的参数为 1 时，才会返回子表达式中的结果
- `group()` 的参数最大不能超过正则表达式里子表达式的个数

```
import re

result = re.search('工号是(.*?)，','我的名字是张三，工号是110110，职业是法外狂徒；他的名字是李四，工号是120120，职业是医生').group(1)

# '110110'
```

（3）re.sub(pattern,replace,string,count) / re.subn(pattern,replace,string,count)

- 可以指定替换次数，不指定则默认替换全部

```
import re

result1 = re.sub('\d+','119119','我的名字是张三，工号是110110，职业是法外狂徒；他的名字是李四，工号是120120，职业是医生')

result2 = re.subn('\d+','119119','我的名字是张三，工号是110110，职业是法外狂徒；他的名字是李四，工号是120120，职业是医生',1)

print(result1)
print(result2[0])


# 我的名字是张三，工号是119119，职业是法外狂徒；他的名字是李四，工号是119119，职业是医生
# 我的名字是张三，工号是119119，职业是法外狂徒；他的名字是李四，工号是120120，职业是医生
```

（4） re.finditer(pattern,string,flags)

```
import re

for item in re.finditer('\d+','我的名字是张三，工号是110110，职业是法外狂徒；他的名字是李四，工号是120120，职业是医生'):
    print(item.group())

# 110110
# 120120
```

（5）re.split(pattern,string,maxsplit)

- 返回字符串被分割后的列表，可以指定最大分割次数，不指定则全部分割

```
import re

result1 = re.split('\d+','我的名字是张三，工号是110110，职业是法外狂徒；他的名字是李四，工号是120120，职业是医生')

result2 = re.split('\d+','我的名字是张三，工号是110110，职业是法外狂徒；他的名字是李四，工号是120120，职业是医生',maxsplit=1)

print(result1)
print(result2)

# ['我的名字是张三，工号是', '，职业是法外狂徒；他的名字是李四，工号是', '，职业是医生']
# ['我的名字是张三，工号是', '，职业是法外狂徒；他的名字是李四，工号是120120，职业是医生']
```

2. pandas 库

- 构造数据

```
import pandas as pd

df = pd.DataFrame(data=['纤美(CICI)','江中猴姑','PENTAL SOFTLY','25°', '卡士(CLASSY·KISS)'],
                  columns=['品牌全名'])
df
```

	品牌全名
0	纤美(CICI)
1	江中猴姑
2	PENTAL SOFTLY
3	25°
4	卡士(CLASSY·KISS)

- 自定义函数

```
def chinese_name(x):
    result = re.findall('[一-龟]',x)
    return ''.join(result)

df['中文品牌名_自定义函数'] = df['品牌全名'].apply(chinese_name)
df
```

	品牌全名	中文品牌名_自定义函数
0	纤美(CICI)	纤美
1	江中猴姑	江中猴姑
2	PENTAL SOFTLY	
3	25°	
4	卡士(CLASSY·KISS)	卡士

- Series.str.findall(pattern, flags)

```
df['中文品牌名_findall'] = df['品牌全名'].str.findall('[一-龟]').apply(lambda x:''.join(x))
df
```

	品牌全名	中文品牌名_自定义函数	中文品牌名_findall
0	纤美(CICI)	纤美	纤美
1	江中猴姑	江中猴姑	江中猴姑
2	PENTAL SOFTLY		
3	25°		
4	卡士(CLASSY·KISS)	卡士	卡士

- Series.str.contains(pattern,regex=True,flags)

```
df['英文品牌名_contains'] = df['品牌全名'].str.contains('[a-zA-Z]',regex=True)
df
```

	品牌全名	中文品牌名_自定义函数	中文品牌名_findall	英文品牌名_contains
0	纤美(CICI)	纤美	纤美	True
1	江中猴姑	江中猴姑	江中猴姑	False
2	PENTAL SOFTLY			True
3	25°			False
4	卡士(CLASSY·KISS)	卡士	卡士	True

- Series.str.replace(pattern,replace)

```
df['英文品牌名_match'] = df['品牌全名'].str.match('[a-zA-Z]')
df
```

	品牌全名	中文品牌名_自定义函数	中文品牌名_findall	英文品牌名_contains	英文品牌名_match
0	纤美(CICI)	纤美	纤美	True	False
1	江中猴姑	江中猴姑	江中猴姑	False	False
2	PENTAL SOFTLY			True	True
3	25°			False	False
4	卡士(CLASSY·KISS)	卡士	卡士	True	False

- Series.str.extract(pattern, flags,expand) / Series.str.extractall(pattern, flags)

```
df['品牌名_replace'] = df['品牌全名'].str.replace('[0-9]','XX')
df
```

	品牌全名	中文品牌名_自定义函数	中文品牌名_findall	英文品牌名_contains	英文品牌名_match	品牌名_replace
0	纤美(CICI)	纤美	纤美	True	False	纤美(CICI)
1	江中猴姑	江中猴姑	江中猴姑	False	False	江中猴姑
2	PENTAL SOFTLY			True	True	PENTAL SOFTLY
3	25°			False	False	XXXX°
4	卡士(CLASSY·KISS)	卡士	卡士	True	False	卡士(CLASSY·KISS)

- Series.str.match(pattern, flags).str[0]

```
df['品牌名_extract'] = df['品牌全名'].str.extract(r'([a-zA-Z].*)',expand=True)
df['品牌名_extract'] = df['品牌名_extract'].str.replace('\)','')
df
```

	品牌全名	中文品牌名_自定义函数	中文品牌名_findall	英文品牌名_contains	英文品牌名_match	品牌名_replace	品牌名_extract
0	纤美(CICI)	纤美	纤美	True	False	纤美(CICI)	CICI
1	江中猴姑	江中猴姑	江中猴姑	False	False	江中猴姑	NaN
2	PENTAL SOFTLY			True	True	PENTAL SOFTLY	PENTAL SOFTLY
3	25°			False	False	XXXX°	NaN
4	卡士(CLASSY·KISS)	卡士	卡士	True	False	卡士(CLASSY·KISS)	CLASSY·KISS