正则表达式

正则表达式

正则表达式是一种用来匹配字符串的强有力的武器。它的设计思想是用一种描述性的语言来给字符串定义一个规则,凡是符合规则的字符串,我们就认为它“匹配”了,否则,该字符串就是不合法的。

基本模式

- 字面模式: 就是字面长量,就代表其本身
- . 匹配任何字符
- \w 匹配一个单词(字母) \W 匹配非字母
- \s 匹配空白 \S 匹配非空白字符
- \d 匹配数字
- ^ 开头 $ 结尾
- \ 转义字符

次数的匹配

次数的匹配，匹配其前面的字符出现的次数：

- * 0 次或多次
- + 一次或多次
- ? 零次或一次
- {n} 出现 n 次
- {m,n} 出现 m 到 n 次

中括号

- 中括号用于指向一个字符集合
- 中括号可以使用元字符
- 中括号中的. 表示其字面意思

[a-z] [A-Z] [0-9] [A-Za-z]

中括号

- [0-9a-zA-Z_] 可以匹配一个数字、字母或者下划线;
- [0-9a-zA-Z_]+ 可以匹配至少由一个数字、字母或者下划线组成的字符串;
- [a-zA-Z_][0-9a-zA-Z_]{0, 19} 更精确地限制了变量的长度是 1-20 个字符;
- A|B 可以匹配 A 或 B
- ^\d 表示必须以数字开头
- \d$ 表示必须以数字结束

思考

- 判断一个字符串是否是合法的 Email 的方法；
- 判断满足029-1234567这样要求的电话号码的方法;

re 模块

r = r'hello'

re.match(r, 'hello')

re.match(r, 'westos')

match() 方法判断是否匹配,如果匹配成功,返回一个 Match 对象,否则返回 None 。

分组

```
m = re.match(r'^(\d{3})-(\d{3,8})$', '010-12345')

m.group(0)

m.group(1)

m.group(2)
```

贪婪匹配

正则匹配默认是贪婪匹配,也就是匹配尽可能多的字符

```
>>> re.match(r'^(\d+)(0*)$', '102300').groups()
('102300', '')
```

贪婪匹配

正则匹配默认是贪婪匹配,也就是匹配尽可能多的字符

```
>>> re.match(r'^(\d+)(0*)$', '102300').groups()
('102300', '')
```

\d+ 采用贪婪匹配,直接把后面的 0 全部匹配了,结果 0* 只能匹配空字符串

贪婪匹配

正则匹配默认是贪婪匹配,也就是匹配尽可能多的字符

```
>>> re.match(r'^(\d+)(0*)$', '102300').groups()
('102300', '')
```

- \d+ 采用贪婪匹配,直接把后面的 0 全部匹配了,结果 0* 只能匹配空字符串
- 必须让 \d+ 采用非贪婪匹配(也就是尽可能少匹配),才能把后面的 0 匹配出来,加个？就可以让 \d+ 采用非贪婪匹配

编译

当我们在 Python 中使用正则表达式时,re 模块内部会干两件事情:

1. 编译正则表达式,如果正则表达式的字符串本身不合法,会报错;

2. 用编译后的正则表达式去匹配字符串。

重复使用几千次,出于效率的考虑,我们可以预编译该正则表达式。

编译

```
r = r'hello'

r_compile = r.compile(r)

r_compile .match()
```

练习

基础版:有一个日志文件access.log,统计访问前十的 IP 地址和访问次数。

升级版:有多个日志文件access.log,统计访问前十的 IP 地址和访问次数。

总结

- re.match(p,text) :p 为正则表达式模式, text 要查找的字符串,会返回一个 match 对象
- re.search(p,text) : 只要在 text 中匹配到了 p 就返回,只返回第一个匹配到的
- re.findall(p,text) :将能匹配上的全返回,会返回一个 list
- re.split(p,text) : 按照 p 匹配,并且以匹配到的字符为分隔符切割 text, 返回一个切割后的 list
- re.sub(p,s,text) : 替换,将 p 匹配到的字符替换为 s.
- pattern = re.compile(p) 先编译 p 模式,当正则表达式模式比较复杂的时候,会先编译,然后再使用

over !