**正则表达式**

1. **概念和简介**

正则表达式是对字符串(包括普通字符(例如:a到z之间的字母)和特殊字符(称为”元字符”)操作的一种逻辑公式,就是使用事先定义好的一些特殊字符、及这些特定字符的组合,组成一个”规则字符串” ,这个”规则字符串”用来表达对字符串一种过滤规则,正则表达式是一种文本模式,模式描述在搜索文本时要匹配的一个或多个字符串.

1. **目的**

给定一个正则表达式和另外一个字符串,我们可以达到如下的目的:

1. 给定的字符串是否符合正则表达式的过滤逻辑(称作”匹配”). ex:电子邮箱是否合法
2. 可以通过正则表达式,从字符串中获取我们想要的内容
3. **特点**

正则表达式的特点:

1. 灵活性、逻辑性和功能性非常强
2. 可以迅速地用极其简单的方式达到字符串的复杂控制
3. 对于刚接触的人来说,稍显晦涩难懂
4. 其主要的应用对象是文本
5. **符号**

正则表达式由一些普通的字符和一些元字符组成.普通字符包括大小英文字母和数字,而元字符则具有特殊的含义,稍后予以解释.

在最简单的情况下,一个正则表达式看上去就是一个普通的字符串. ex: 正则表达式” testing ”中没有包含任何元字符,它可以匹配” testing ”和” testing123 ”等字符串,但是不能匹配” Testing ”,匹配后返回的结果就正则表达式本身: ” testing ”.

要想真正用好正则表达式,正确理解元字符是最重要的事情.下表列出了常用的元字符及对它们的一个简短描述.

**常见正则表达式符号和特殊字符**

|  |  |  |
| --- | --- | --- |
| **表示法** | **描述** | **正则表达式示例** |
| **符号** |  | |
| literal | 匹配文本字符串的字面值literal | foo |
| re1|re2 | 匹配正则表达式re1或re2 | foo|bar |
| . | 匹配除’\n’外的任意字符 | b.b |
| ^ | 匹配字符串起始部分 | ^Dear |
| $ | 匹配字符串终止部分 | /bin/\*sh$ |
| \* | 匹配0次或多次前面出现的正则子表达式 | [A-Za-z0-9]\* |
| + | 匹配1次或多次前面出现的正则子表达式 | [a-z]+\.com |
| ? | 1、匹配0次或1次前面出现的正则子表达式  2、当该字符紧跟在任何一个其他限制符(\*, +, ?, {n}, {n, m})后面时,匹配模式是非贪婪的.非贪婪模式尽可能少地匹配所搜索的字符串,而默认的贪婪模式则尽可能多地匹配所搜索的字符串 | goo? |
| {n} | 匹配n次前面出现的子表达式,n是一个非负整数 | [0-9]{3} |
| {n,} | 至少匹配n次前面出现的子表达式,n是一个非负整数 | [0-9]{5,} |
| {n, m} | 匹配n到m次前面出现的子表达式(最少n次,最多m次),n<=m且均为发非负整数 | [0-9]{3,10} |
| […] | 匹配来自字符集的任意单一字符 | [auiou] |
| [..x-y..] | 匹配x ~ y范围中的任意单一字符 | [0-9],[A-Za-z] |
| [^…] | 不匹配此字符集中出现的任何一个字符,包括某一范围的字符(如果在此字符集中出现) | [^auiou], [^A-Za-z0-9] |
| (\*|+|?|{})? | 用于匹配上面频繁出现/重复出现符号的非贪婪模式版本(\*, +, ?, {}) | .\*?[a-z] |
| (…) | 匹配封闭的正则表达式,然后另存为子组 | ([0-9]{3})?, f(oo|u)bar |
| **特殊字符** |  |  |
| \ | 将下一个字符标记为正常的字符、或一个向后的引号、或一个八进制转义符.ex: ’\\n’匹配’ \n ’(换行符)、’\\’匹配’\’、而’\(‘匹配’(‘ .相当于多种编程语言中的转义字符 | \\n |
| \d | 匹配任何十进制数字,与[0-9]一致(\D与\d相反,不匹配任何数字) | data\d+.txt |
| \w | 匹配任何字母数字字符,与[A-Za-z0-9]相同(\W与之相反) | [A-Za-z\_]\w+ |
| \s | 匹配任何空格字符,与[\n\t\r\v\f]相同(\S与之相反) | of\sthe |
| \b | 匹配任何单词边界(\B与之相反) | \bThe\b |
| \N | 匹配以保存的子组N(参见上面的(…)) | Price:\16 |
| \c | 逐字匹配任何特殊字符c(即:仅按照字面意义匹配,不匹配特殊含义) | \., \\, \\* |
| \A(\Z) | 匹配字符串的起始(结束)(另见上面介绍的^和$) | \ADear |
| **扩展表示法** |  | |
| (?P<name>…) | 像一个仅由name标识而不是数字ID标识的正则分组匹配 | (?P<data>\d{3,6}) |

1. **速记理解技巧**

**（1）等价**

等价就是等同于的意思,表示同样的功能,用不同的符号来书写.

? , \* , + , \d , \w都是等价字符:

? 等价于匹配长度{0,1}

\* 等价于匹配长度{0,}

+ 等价于匹配长度{1,}

\d等价于[0-9]

\D等价于[^0-9]

\w等价于[A-Za-z\_0-9]

\W等价于[^A-Za-z\_0-9]

**（2）常用运算符于表达式**

^ 开始

( ) 域段

[] 包含,默认是一个字符长度

[^] 不包含,默认是一个字符长度

{n, m} 匹配长度

. 任何单个字符(\.)

| 或

\ 转义

$ 结尾

[A-Z] 26个大写字母

[a-z] 26个小写字母

[0-9] 0至9的数字

[A-Za-z0-9] 26个大写字母、26个小写字母和0至9数字

, 分割

**（3）分割语法**

[A,H,T,W] 包含A或H或T或W字母

[a,h,t,w] 包含a或h或t或w字母

[0,3,6,8] 包含0或3或6或8数字

**（4）语法与释义**

基础语法”^([]{})([]{})([]{})$”

正则字符串 = “开始([包含内容]{长度})([包含内容]{长度})([包含内容]{长度})结束”

?, \*, +, \d, \w这些都是简写的,完全可以用[]和{}代替,在(?:)(?=) (?!)(?<=)(?<!)(?i) (\*?)(+?)这种特殊情况下除外,初学者可以忽略?, \*, +, \d, \w这些简写标识符,学会了基础的使用再按自己的表达去等价替换

1. **Python的re模块**

re.match() 从头开始匹配

re.search() 匹配包含

re.findall() 把所有匹配到的字符放到列表中，最后返回一个列表

re.splitall() 以匹配到的字符当做列表分隔符

re.sub() 匹配字符并替换

re.fullmatch() 全部匹配

re.compile() 将正则表达式进行编译，以供后续使用

re.split() 使用正则方法对字符串进行分割