# 正则表达式入门

正则表达式(Regular Expression)其实就是一门工具，**目的**是为了字符串模式匹配，从而实现搜索和替换功能。它是一种用来描述规则的表达式。

1. **语法规则**
   1. **基础组成**

正则表达式是由**字符**和**元字符**组成，**字符**通常是指数字、英文字母，而**元字符**也称为特殊字符，用于表达特殊语义的字符，如表示开始和结束符、重复出现的次数和范围等等。

* + 1. **普通字符**
       1. 单个字符

|  |  |
| --- | --- |
| **表达式** | **含义** |
| /a/ | 匹配字母a |
| /1/ | 匹配数字1 |

* + - 1. 多个字符

|  |  |
| --- | --- |
| **表达式** | **含义** |
| /aa/ | 匹配两个字符：aa |
| /11111/ | 匹配5个数字：11111 |

* + - 1. 字符集合区间

正则中

* ”[]”用于表示集合，即从”[]”中的字符任选一个;
* ”[^]”表示集合中取反，即不取”[]”中出现的任意字符；
* “-”表示区间范围；

|  |  |
| --- | --- |
| **表达式** | **含义** |
| /[135]/ | 从数字1、3、5中匹配任意1个 |
| /[12345]/ | 从数字1、2、3、4、5中匹配任意1个 |
| /[1-5]/ | 从数字1-5中匹配任意1个 |
| /[acdeg]/ | 从字母a、c、d、e、g中匹配任意1个 |
| /[a-e]/ | 从字母a-e中匹配任意1个 |
| /[^135ac]/ | 匹配1、3、5、a、c以外的任意1个字符 |

结合以上三种方式，尝试匹配出5个数字，表达式如下：

|  |
| --- |
| **方式1：**  Regex=/[0123456789] [0123456789] [0123456789] [0123456789] [0123456789]/  **方式2：**  Regex=/[0-9][0-9][0-9][0-9][0-9]/ |

* + 1. **元字符（特殊字符）**

经历了用普通字符方式进行匹配”5个数字“的繁琐写法后，咱们再用元字符展现下能力：regex=/\d{5}/

元字符按使用场景课细分为如下种类：

1. **修饰符**

|  |  |  |
| --- | --- | --- |
| **元字符** | **含义** | **示例** |
| g | 全局匹配 | “Hi a,hi b,hi c”  Regex=/hi/   * “Hi a,hi b,hi c”   Regex=/hi/g   * “Hi a,hi b,hi c”   Regex=/hi/i   * “Hi a,hi b,hi c”   Regex=/hi/gi   * “Hi a,hi b,hi c” |
| i | 忽略大小写 |
| gi | 全局匹配并且忽略大小写 |

1. **边界符**

|  |  |  |
| --- | --- | --- |
| **元字符** | **含义** | **示例** |
| ^ | 匹配字符串的开始 | Regex=/^abc/  仅匹配以”abc”开头的字符”abc” |
| $ | 匹配字符串的结束 | Regex=/^de$/  仅匹配以”de”结尾的字符”de” |
| \b | 匹配字符的开始或结束，每个单词的前后都各有一个\b符号，是看不见的，不同于空格等空白符\s | “His name is Jack”   1. Regex=/\bis\b/  * “His name is Jack”   仅匹配单词is而非His中的is.   1. Regex=/\Bis/  * “His name is Jack”  1. Regex=/\Bis\B/  * 啥都匹配不到 |
| \B | 匹配字符的非开始及非结束 |

1. **预定义符**

预定义符，主要为了书写方便（一个字母替代一坨表达式），具体如下表.

|  |  |
| --- | --- |
| **元字符** | **含义** |
| \d | 匹配任意一个数字，  等价于[0-9] |
| \D | 匹配任意一个非数字，  等价于[^0-9] |
| \w | 匹配任意一个字母、数字、下划线  等价于[a-zA-Z0-9\_] |
| \W | 匹配任意一个非字母、数字、下划线  等价于[^a-zA-Z0-9\_] |
| \s | 匹配任意一个空白符(空格、制表符、回车、换行等)，  等价于 [ \f\n\r\t\v] |
| \S | 匹配任意一个非空白符(空格、制表符、回车、换行等)，等价于 [ ^\f\n\r\t\v] |
| . | “点”匹配除\r\n（回车换行符）以外的任意一个字符,  等价于[^\r\n] |

1. **频次符**

频次符应用于匹配到的字符重复出现的次数.

|  |  |
| --- | --- |
| **元字符** | **含义** |
| \* | 匹配0次或无数次 |
| ? | 匹配0次或1次 |
| + | 匹配1次或无数次 |
| {n} | 匹配出现n次 |
| {min, max} | 至少出现min次但不超过max次 |
| {min,} | 至少出现min次 |
| {0, max} | 最多出现max次 |

1. **其他特殊符**

|  |  |
| --- | --- |
| **元字符** | **含义** |
| \ | 转义字符，用于匹配特殊字符.  如：  匹配反斜杠，regex=\\  匹配\*，regex=\\* |
| | | 表示或；  如：  Regex=\[2|3]\  即匹配2或3 |

* 1. **分组与反向引用**

通过上述《基础组成》部分的学习，尝试下如何匹配出如下字符串中重复出现的字符或单词（aa、22）

Str1=”abc aa 22 235”

发现目前已学的语法无法实现此需求，因此正则表达式中引入了更高级的用法——“子表达式”，通过**分组**以及**反向引用**，实现所有逻辑正确的匹配查找。

* + 1. **分组**

所有出现在(和)之间的正则表达式被分为一组，每一个组成为子表达式。

示例：

如何匹配出”abcc abcabc”

Regex= /(abc){2}/

假设Regex= /abc{2}/，则只能匹配到”abcc abcabc”

* + 1. **反向引用**

指的是模式的后面部分引用前面已经匹配到的子字符串。你可以把它想象成是变量，回溯引用的语法像\1,\2,….,其中\1表示引用的第一个子表达式，\2表示引用的第二个子表达式，以此类推。而\0则表示整个表达式。

Str1=”abc aa 22 235”

Regex=/(\w)\1/

1. 验证工具
   1. 正则表达式图形化

<https://regexper.com/>

以图形化方式直观查看正则表达式的组成.

* 1. 匹配验证

<https://rubular.com/>

在线测试正则表达式是否正确。

1. 实战

参考：

<https://www.cnblogs.com/Allen-rg/p/9323506.html>

<https://blog.csdn.net/weixin_37536446/article/details/81053172>

<https://baijiahao.baidu.com/s?id=1588848792548192879&wfr=spider&for=pc>

<https://www.runoob.com/regexp/regexp-intro.html>

<https://www.cnblogs.com/zery/p/3438845.html>

<https://cloud.tencent.com/developer/news/96975>