|  |  |
| --- | --- |
| **字符** | **含义** |
| . | 表示匹配除了换行符外的任何字符 注：通过设置 re.DOTALL 标志可以使 . 匹配任何字符（包含换行符） |
| | | A | B，表示匹配正则表达式 A 或者 B |
| ^ | 1. （脱字符）匹配输入字符串的开始位置 2. 如果设置了 re.MULTILINE 标志，^ 也匹配换行符之后的位置 |
| $ | 1. 匹配输入字符串的结束位置 2. 如果设置了 re.MULTILINE 标志，$ 也匹配换行符之前的位置 |
| \ | 1. 将一个普通字符变成特殊字符，例如 \d 表示匹配所有十进制数字 2. 解除元字符的特殊功能，例如 \. 表示匹配点号本身 3. 引用序号对应的子组所匹配的字符串 4. 详见下方列举 |
| [...] | 字符类，匹配所包含的任意一个字符 注1：连字符 - 如果出现在字符串中间表示字符范围描述；如果如果出现在首位则仅作为普通字符 注2：特殊字符仅有反斜线 \ 保持特殊含义，用于转义字符。其它特殊字符如 \*、+、? 等均作为普通字符匹配 注3：脱字符 ^ 如果出现在首位则表示匹配不包含其中的任意字符；如果 ^ 出现在字符串中间就仅作为普通字符匹配 |
| {M,N} | M 和 N 均为非负整数，其中 M <= N，表示前边的 RE 匹配 M ~ N 次 注1：{M,} 表示至少匹配 M 次 注2：{,N} 等价于 {0,N} 注3：{N} 表示需要匹配 N 次 |
| \* | 匹配前面的子表达式零次或多次，等价于 {0,} |
| + | 匹配前面的子表达式一次或多次，等价于 {1,} |
| ? | 匹配前面的子表达式零次或一次，等价于 {0,1} |
| \*?, +?, ?? | 默认情况下 \*、+ 和 ? 的匹配模式是贪婪模式（即会尽可能多地匹配符合规则的字符串）；\*?、+? 和 ?? 表示启用对应的非贪婪模式。 *举个栗子：对于字符串 "FishCCC"，正则表达式 FishC+ 会匹配整个字符串，而 FishC+? 则匹配 "FishC"。* |
| {M,N}? | 同上，启用非贪婪模式，即只匹配 M 次 |
| (...) | 匹配圆括号中的正则表达式，或者指定一个子组的开始和结束位置 注：子组的内容可以在匹配之后被 \数字 再次引用 *举个栗子： (\w+) \1 可以字符串 "FishC FishC.com" 中的 "FishC FishC"（注意有空格）* |
| (?...) | (? 开头的表示为正则表达式的扩展语法（下边这些是 Python 支持的所有扩展语法） |
| (?aiLmsux) | 1. (? 后可以紧跟着 'a'，'i'，'L'，'m'，'s'，'u'，'x' 中的一个或多个字符，只能在正则表达式的开头使用 2. 每一个字符对应一种匹配标志：re-A（只匹配 ASCII 字符），re-I（忽略大小写），re-L（区域设置），re-M（多行模式）, re-S（. 匹配任何符号），re-X（详细表达式），包含这些字符将会影响整个正则表达式的规则 3. 当你不想通过 re.compile() 设置正则表达式标志，这种方法就非常有用啦 注意，由于 (?x) 决定正则表达式如何被解析，所以它应该总是被放在最前边（最多允许前边有空白符）。如果 (?x) 的前边是非空白字符，那么 (?x) 就发挥不了作用了。 |
| (?:...) | 非捕获组，即该子组匹配的字符串无法从后边获取 |
| (?P<name>...) | 命名组，通过组的名字（name）即可访问到子组匹配的字符串 |
| (?P=name) | 反向引用一个命名组，它匹配指定命名组匹配的任何内容 |
| (?#...) | 注释，括号中的内容将被忽略 |
| (?=...) | 前向肯定断言。如果当前包含的正则表达式（这里以 ... 表示）在当前位置成功匹配，则代表成功，否则失败。一旦该部分正则表达式被匹配引擎尝试过，就不会继续进行匹配了；剩下的模式在此断言开始的地方继续尝试。 *举个栗子：love(?=FishC) 只匹配后边紧跟着 "FishC" 的字符串 "love"* |
| (?!...) | 前向否定断言。这跟前向肯定断言相反（不匹配则表示成功，匹配表示失败）。 *举个栗子：FishC(?!\.com) 只匹配后边不是 ".com" 的字符串 "FishC"* |
| (?<=...) | 后向肯定断言。跟前向肯定断言一样，只是方向相反。 *举个栗子：(?<=love)FishC 只匹配前边紧跟着 "love" 的字符串 "FishC"* |
| (?<!...) | 后向否定断言。跟前向肯定断言一样，只是方向相反。 *举个栗子：(?<!FishC)\.com 只匹配前边不是 "FishC" 的字符串 ".com"* |
| (?(id/name)yes-pattern|no-pattern) | 1. 如果子组的序号或名字存在的话，则尝试 yes-pattern 匹配模式；否则尝试 no-pattern 匹配模式 2. no-pattern 是可选的 *举个栗子：(<)?(\w+@\w+(?:\.\w+)+)(?(1)>|$) 是一个匹配邮件格式的正则表达式，可以匹配 <user@fishc.com> 和 'user@fishc.com'，但是不会匹配 '<user@fishc.com' 或 'user@fishc.com>'* |
| \ | 下边列举了由字符 '\' 和另一个字符组成的特殊含义。注意，'\' + 元字符的组合可以解除元字符的特殊功能 |
| \序号 | 1. 引用序号对应的子组所匹配的字符串，子组的序号从 1 开始计算 2. 如果序号是以 0 开头，或者 3 个数字的长度。那么不会被用于引用对应的子组，而是用于匹配八进制数字所表示的 ASCII 码值对应的字符 *举个栗子：.+) \1 会匹配 "FishC FishC" 或 "55 55"，但不会匹配 "FishCFishC"（注意，因为子组后边还有一个空格）* |
| \A | 匹配输入字符串的开始位置 |
| \Z | 匹配输入字符串的结束位置 |
| \b | 零宽断言，匹配一个单词边界，单词被定义为 Unidcode 的字母数字或下横线字符 *举个栗子：\bFishC\b 会匹配字符串 "love FishC"、FishC." 或 "(FishC)"* |
| \B | 零宽断言，匹配非单词边界，其实就是与 \b 相反 *举个栗子：py\B 会匹配字符串 "python"、"py3"  或 "py2"，但不会匹配 "py  "、"py." 或  "py!"* |
| \d | 1. 对于 Unicode（str 类型）模式：匹配任何一个数字，包括 [0-9] 和其他数字字符；如果开启了 re.ASCII 标志，就只匹配 [0-9] 2. 对于 8 位（bytes 类型）模式：匹配 [0-9] 中任何一个数字 |
| \D | 匹配任何非 Unicode 的数字，其实就是与 \d 相反；如果开启了 re.ASCII 标志，则相当于匹配 [^0-9] |
| \s | 1. 对于 Unicode（str 类型）模式：匹配 Unicode 中的空白字符（包括 [ \t\n\r\f\v] 以及其他空白字符）；如果开启了 re.ASCII 标志，就只匹配 [ \t\n\r\f\v] 2. 对于 8 位（bytes 类型）模式：匹配 ASCII 中定义的空白字符，即 [ \t\n\r\f\v] |
| \S | 匹配任何非 Unicode 中的空白字符，其实就是与 \s 相反；如果开启了 re.ASCII 标志，则相当于匹配 [^ \t\n\r\f\v] |
| \w | 1. 对于 Unicode（str 类型）模式：匹配任何 Unicode 的单词字符，基本上所有语言的字符都可以匹配，当然也包括数字和下横线；如果开启了 re.ASCII 标志，就只匹配 [a-zA-Z0-9\_] 2. 对于 8 位（bytes 类型）模式：匹配 ASCII 中定义的字母数字，即 [a-zA-Z0-9\_] |
| \W | 匹配任何非 Unicode 的单词字符，其实就是与 \w 相反；如果开启了 re.ASCII 标志，则相当于 [^a-zA-Z0-9\_] |
| 转义符号 | 正则表达式还支持大部分 Python 字符串的转义符号：\a，\b，\f，\n，\r，\t，\u，\U，\v，\x，\\ 注1：\b 通常用于匹配一个单词边界，只有在字符类中才表示“退格” 注2：\u 和 \U 只有在 Unicode 模式下才会被识别 注3：八进制转义（\数字）是有限制的，如果第一个数字是 0，或者如果有 3 个八进制数字，那么就被认为是八进制数；其他情况则被认为是子组引用；至于字符串，八进制转义总是最多只能是 3 个数字的长度 |