第15章正则表达式

Windows系统的文件搜索功能中提供了 \*和？ 两个特殊字符，分别用于匹配多个字符和单个字 符，例如，在搜索框中输入“％txt”，搜索结果中 会包含所有符合“以.txt结尾”这种模式的文件， 如图154所示。

而若在搜索框中输入“?.txt”，搜索结果中只 会包含符合“以.txt结尾，但.txt之前只有一个字符” 这种模式的文件。

因为文件名的构成相对简单，所以简单的模

在没有索引的位置搜索可能较慢：E:队谙单击以添加到素引...

E：'i.腾讯

txt

图15』\*1xt搜索结果

式便可满足针对文件名的操作，但计算机中要处理的文本内容，如用户信息、网页新闻等，相 比文件名更为复杂，这就要求程序中应能实现更加丰富的模式匹配功能。由此，正则表达式应 运而生。

起源与爨屣

正则表达式（Regu血Exp^ssion）起源于两位神经生理方面的科学家一一美国新泽西州的 Warren McCulloch和岀生在美国底特律的Walter Pitts,他们研究出的一种用数学方式来描述神 经网络的新方法，创造性地将神经系统中的神经元描述成了小而简单的自动控制元。

1951年，数学科学家Stephen Kleene在他们工作的基础上，发表了一篇题为《神经网事件 的表示法》的论文，该论文中利用称为正则集合的数学符号来描述此模型，引入了正则表达式 的概念。之后的一段时间，人们逐渐将正则表达式应用于包括生物、数学以及计算机科学等的 各个领域。

在计算机领域中，正则表达式最先应用于搜索算法的一些早期研究，UNIX之父Kg Thompson将其引入编辑器QED,然后是UNIX上的编辑器园，并最终引入gmp。自此之后， 正则表达式被广泛地应用到了各种UNIX或类UNIX工具中。

随着正则表达式的广泛应用，正则表达式逐渐从模糊而深奥的数学概念，发展成为计算机 各类工具和软件包应用中的主要功能，亦有越来越多的编程语言，包括PHP、C#、Java等实现 了对正则表达式的支持,并提供了与其操作相关的接口 , Python也不例外。

15O2 IE螂表达式胸定鬼

正则表示“规则的、极好的”，正则表达式实际上就是规定了一组文本模式匹配规则的符 号语言，一条正则表达式也称为一个模式，使用这些模式，用户可匹配指定文本中与表达式模 式相同的字符串。

正则表达式也可理解为描述字符串结构模式的形式化表达“方法”或“思想”，它与一些 特定工具搭配使用，可以实现包括添加、删除、分离、插入、选择等各种文本分析功能。

字"类

正则表达式由丰富的符号组成，形式多变，可匹配多种模式的字符身其中的字符大体上分 为两类,一类为普通字符,另一类为元字符。

1位気1聳鑑拿綺

普通字符包括数字、字母和下画线字符，由普通字符组成的正则表达式是最简单的正则表 达式，可匹配表达式自身，示例如下？

（1 ） itheima：可匹配目标文本中的字符串"itheima”。

（2） \_Class：可匹配目标文本中的字符串。

（3） 0x4688：可匹配目标文本中的字符串“0x4688”。

15.3.2屍寥簷 .

元字符指在正则表达式中具有特殊含义的专用字符,可以用来规定其前导字符（位于元字 符之前的字符）在目标对象中出现的模式。正则表达式中的元字符一般由特殊字符和符号组成， 常用的元字符与其功能如表154所示。

表15瑚 常用元字符及其功能

|  |  |
| --- | --- |
| 元字符 | 说 明 |
|  | 匹配任何一个字符（除换行符外） |
| A | 脱字符，匹配行的开始 |
| $ | 美元符,匹配行的结束 |
| I | 连接多个可选元素,匹配表达式中出现的任意子项 |
| [] | 字符组，匹配其中的出现的任意一个字符 |
| - | 连字符，表示范围，如等价于"，2, 3, 4, 5” |
| ? | 匹配其前导元素0次或1次 |
|  | 匹配其前导元素0次或多次 |

|  |  |
| --- | --- |
| 续表 | |
| 元字符 | 说明 |
| + | 匹配其前导元素1次或多次 |
| (n}/{m,n} | 匹配其前导元素n次/匹配其前导元素m~n次 |
| 0 | 在模式中划分出子模式，并保存子模式的匹配结果 |
| 15.3.3 '  正则表达式中预定义了一些字符集，使用元字符和普通字符同样可以表示这些字符集，但 使用字符集表示更为简洁。预定义字符集如表15・2所示。  a 15-2预定义字符集 | |
| 元字符 | 说 明 |
| \w | 匹配下画线或任何字母（a〜z,A〜Z）与数字（0-9）,等价于［0-9a-zA~Z\_］ |
| \s | 匹配任意的空白字符，等价于［＜ 空格＞\t\r\nVl\vj |
| \d | 匹配任意数字，等价于［0-9］ |
| \b ‘ | 匹配单词的边界 |
| \W | 与8相反，匹配特殊字符 |
| \S | 与\s相反，匹配任意非空白字符的字符，等价于［A\s］ |
| \D | 与\d相反，匹配任意非数字的字符，等价于［Nd］ |
| \B | 与\b相反，匹配不出现在单词边界的元素 |
| \A | 仅匹配字符自开头，等价于八 |
| \Z | 仅匹配字符串结尾，等价于$ |
| , I5o4点字简 「 | |

点字符是正则表达式中最简单的元字符，可匹配包括字母、数字、下画线、空白符（除 换行符\n）等任意的单个字符，其用法如下所示。

（1） Im：匹配由字母J，字母m结尾，中间为任意一个字符的字符串，匹配结果可以是 Jam、Jom、J#m、Jim、J2m 等

（2） ,匹配任意两个字符，匹配结果可以是12、2n、@#等等

（3） m 匹配以任意字符开头，以m结尾的字符串，匹配结果可以是Im、@m、xm、\_m、\tai等。

位置題配

文本文件中的内容往往是多行的，每行又可能由多个单词组成，在这种文本中，我们可能 需要从行头、行尾获取字符串，也可能匹配一个单词狩或处于单词中的字符串，因此需要使用 一些可标识位置的元字符。

15.5J緡襲緝屢

脱字符e和美元符分别用于匹配行头和行尾，若想匹配处于串开头的"cat" 9可 使用表达式UAcatn ；若想匹配处于串结尾的"cat",则可使用表达式％龍$”。表达式及其 可匹配到的内容示例如下所示:

(1 )人cat：只能匹配行首出现的cat,例如categoryo

(2 ) cat$：只能匹配行尾出现的cat,例如concato

(3 ) Acat$:匹配只有cat的行。

1. cat：可匹配到行中任意位置出现的caf。
2. A$：匹配空行。

需要说明的是，以(1 )中的模式 如妒为例，虽然该模式会匹配到以字符串“cat”为首的行， 但在理解时，应理解为“匹配以字符c开头，第二、第三个字符依次为a和t的行”。这种理 解方式更符合正则表达式的思维，这种思维在正则表达式的学习和使用中非常重要，读者应熟 练掌握。

1SS.2 「

元字符“\b”用于匹配出现在单词边界的模式，它既可出现在元素之前，亦可出现在元素 之后，示例如下：

(1 ) \bcat：匹配处于单词开头的cato

(2 ) cat\b:匹配处于单词结尾的cato

(3 ) \bcat\b:匹配单词 cato

元字符与“\b”相反，用来匹配出现在单词中间的模式，它同样可出现在元素之前或之后， 示例如下:

(1 ) \Bcat：匹配不以cat开头的单词中的cat。

(2 ) cat\B:匹配不以cat结尾的单词中的cat。

(3 ) \Bcat\B:匹配处于单词中间的cato

6多选结构

T可将多个不同的子表达式进行逻辑连接，可简单地将T理解为逻辑运算符中的“或” 运算符，匹配结果为与任意一个子表达式模式相同的字符串。示例如下：

1. a|b|c|d：匹配字符a、b、c、d中的任意一个。

(2 ) cat|dog：匹配 cat 或 dogo

*(3* ) c|itheima：匹配字符c或字符串itheimao

字简11

正则表达式中使用一对方括号“［广标记字符组，字符组的功能是匹配其中的任意一个字符, 它也有“或”的含义，但与多选结构中的T不同，T既能匹配单个字符，亦能匹配字符串，

但“口”只能匹配单个字符。

字符组的用法如下所示：

1. arg[vs]：匹配以字符串發 开头，以字符v或s结尾的字符串，匹配结果可能是雎v 或 argso
2. [cC]hina：匹配以字符c或C开头，以hina结尾的字符串，匹配结果可以是china或 China。

(3 ) [z!\*?]:匹配z、!、\*、?中的任意一个。

字符组外的普通字符从前到后依次匹配，如在表达式“arg[vs]”中，字符组外的字符a、『、 g的匹配方式是：先匹配以，再匹配“之后匹配g;而字符组中所有普通字符都是同级的，没 有先后顺序，匹配结果至多会选择字符组中的一个字符，换言之，字符组外的字符固定依次占 取一个位置，但字符组中的字符，只有本次匹配成功的字符会占取一个位置。

1S.7J矍掌簿

连字符 r 一般在字符组中使用，表示一个范围，如字符组“卬刃]”表示匹配一位数字， 字符组表示匹配一位大写字母，字符组“Rzf表示匹配一位小写字母。

当数字、字符等可能出现在字符串中的同一个位置时，字符组中允许使用多重范围匹配， 如字符组“[0-9a-zA-Z]”表示匹配数字或字母字符,字符组“。2%町‘表示匹配数字0、1、2 或者小写字母a、b、c、d。

需要注意的是，七’只有出现在字符组内部，且不处于起始位置时才是元字符，否则它 只是一个普通字符，不再表示一个范围。如在表达式“於xt[f]nT的匹配结果可能为texMn. text:m和text.m；表达式utest\_fbrkn的匹配结果只能为字符串test\_fork或None (匹配失败时)° 另外，七‘只能连接有明确范围的字符，否则将会出错，例如使用正则表达式“text[:Tm”匹 配字符串text-m^ text:m^ text.m时都出错。

15.7.2難験醴寥籥飆

脱字符 小”在字符组外使用时用于匹配处于行头的文本，但在字符组中使用时，5”表 示的含义为“排除”，用于匹配一个字符组中未列出的字符。

示例如下:

(1 ) qws[Aun]d：匹配字符串“qws°(T中第四个字符不为u或n的文本，匹配结果可以是 "qwsad”、"qwsed”，但不会是 “qwsud” 或 “qwsnd”。

1. [A1^4]：匹配除数字卜4之外的其他字符，匹配结果可以是5、a、〜，但不会是1、2、 3、4。

气艮乙登 晁事籥塲掌筏貪 1 ' "

除连字符七”和脱字符“5夕卜，字符组中也可能出现其他的元字符。需要说明的是，大 部分元字符出现在字符组中作为匹配项之一时，其特殊功能会被剥离，退化为一个普通字符， 即只表示匹配该字符本身，示例如下：

(1 ) r[a.b]s：匹配以『开头歹以s结尾，中间字符为％”七” ％”其中之一的3个字符串， 结果可能为ras^ r.s或而。

(2)r[-c.f]S：匹配以『开头，以&结尾，中间字符不为“c” " “f”的任意3个字符串， 结果可能为ras. rbs,但不可能为res、玷、rfs。

(3 ) [.\*$?]:匹配j "?”中的任意一个字符，匹配结果为七”

或 “?” 。

15o8 BMW

正则表达式中提供了一些匹配模式9可用于限定其前导元素的匹配次数。

15.8J圆遴樓豌

在英文中，月份可使用全拼，或取其前三个字母表示，如六月可写为June或Jun；七月可 写为July或Jul。我们发现，当表示月份的单词只有4个字母时，第4个字母变得可有可无。 换言之须当匹配这两个月份时，若前3个字符匹配成功后，第4个字符匹配成功，或不存在都 应该获得肯定的匹配结果。

此时，可以使用元字符“?"实现月份的匹配。使用元字符时，其前导元素将被匹配 0次或1次,示例如下:

1. June?:匹配元字符“?”前的字符粘” 0次或1次，匹配结果可以是Jim或Jtme。
2. July?:匹配元字符“?”前的字符“y”。次或1次，匹配结果可以是Jul或July。

1⑤思卷蠶虞滇就

正则表达式中提供了 3种方式来限定前导元素的重复模式，这三种方式涉及的符号分别为 “+”和“{广，它们表示的含义分别如下：

(1 )气匹配其前导元素0次或多次。

1. +：匹配其前导元素1次或多次。
2. {)： {n}表示匹配其前导元素11次；(m9n)表示匹配其前导元素m〜n次。

这三种重复模式的用法分别如下所示。

(1) ht\*p:匹配字符％”零次或多次，匹配结果可以是hp、htp、http、htttp等。

(2 ) ht+p:匹配字符"t” 一次或多次，匹配结果可以是htp、http、htttp,但不可能 是hp。

1. ht{2}p：匹配字符"” 2次，匹配结果为http。

(4 ) ht{2,4)p：匹配字符2〜4次,匹配结果可以是http、htttp与littttpo

5 “?” “ + ”等除了以元字符的方式作为匹配模式存在，也可能单纯地作为普通

字符进行匹配，例如在匹配算式“3+5”时，"+”不表示重复匹配数字，而只是一个普通的算

294*、*Python SB®®:期霉攣 Pythoo

术运算符，在这种情况下，若要使用元字符，需要对其进行转义。

pythoii中以“\”作为转义字符，当其与元字符连用时，元字符不再表示匹配模式，而只是 一个普通字符。示例如下：

1. 3\+5：匹配 3+5。
2. 3\\*5：匹配 3\*5。

(3 ) userWW：匹配 user\\o

正则表达式中的预定义字符集由反斜杠和字母组成，因此，当模式中出现反斜杠与普通字 符时，可能会构成预定义字符集，例如，指定模式u\daten 9本意为匹配u\daten这5个字符, 但“\d”会被视为预定义字符集，并匹配一个任意数字，这显然是不符合预期的。为此，我们 需要在u\date"之前添加转义符，将其写为“\\date”。

若单纯的考虑正则表达式的表示方法，"Wdate"已能够表示匹配字符串u\daten ,然而， 在python中使用正则表达式时，正则表达式会被视为一个字符串类型的参数进行传递，又因 pythg中的特殊字符也使用进行转义，所以，若要正确地在python中使用由字符串表示 的正则表达式“WdafR” 9需要将其中的两个“\”分别进行转义，即写为u\\\\daten *。*

综上所述，若要匹配含有预定义字符集的字符串，需要经过以下操作：

.(1 ) \date:获取要匹配的字符串。

(2 ) Wdate:基于正则表达式,对“\d”进行转义。

(3 ) \W\date:基于python字符串，对正则表达式中的进行转义。

这显然是极为麻烦的，且容易出错。因此，用户在使用正则表达式时，应尽量使用原始字 符串，即以七‘…’”的形式进行参数传递。

子n

在1581小节的月份匹配中，大多数月份的单词长度都超过4个字符，也就是说，在匹配 时，将会有多个字符同时被视为可选项。这种情况下，我们可以为正则表达式中的字符添加一 对圆括号“()”，将圆括号中包含的内容界定为子组(也称为子模式)，之后再使用元字符“?” 对其进行选择匹配，示例如下：

(1 ) Jan(uary)?:匹配子组“nary” 0次或1次，匹配结果为Jan或Januaryo

(2)Feb(ruary)?：匹配子组"ruary" 0次或1次，匹配结果为Feb或February。

(3 ) Mar(ch)?：匹配子组“ch” 0次或1次,匹配结果为March或Mar。

15J0J 一 「

为模式划分子组的意义有两个：

1. 降低多选结构中可选分支的体积,例如多选结构uthe|thread|think",可整合为 uth(e|read|ink)n 。
2. 方便获取子模式的匹配结果。

在模式匹配之后，匹配结果会被返回给函数调用者，当一个模式相对复杂时，用户可能需

要查看匹配过程产生的结果，若利用圆括号将模式划分为多个子模式，用户可通过下标来获取子 模式的匹配结果。例如，调用Hud疝()函数通过模式u((hello|hi) (world|itheima))n (注意：两个子 组之间有\_空格)对字符串"hello itheima"进行匹配后，其匹配结果"[fhello itheima7hello7ithep mW)]”会被返回，该结果为一个列表，列表中的内容是一个元组，元组的每一个元素对应模式的 一个匹配结果，其中第一个元素为整个模式的匹配结果，其后依次为各个子模式的匹配结果。

假设使用对象t接收匹配结果，则模式与其两个子模式匹配结果的获取方式依次为：㈣0]、 t[0][l], t[0][2],获取的结果依次为：'hello itheima5 . ' hello5 . ? itheima5 *。*

15J0D2

子组在正则表达式的使用中非常常见，但在有些情况下，我们或许需要对正则表达式进行 分组，但并不需要获取子组的匹配结果。在这种情况下，可在子组的左括号之后添加字符串“?:”， 如此,子组的匹配结果不再被捕获,例如将模式“((hello|hi) (world|itheima))n改为u((?:hello|hi) (world|itheima))n ,整个模式的匹配结果则为 “[('hello itheima?辻heima')]”。

15oH re«

python中对正则表达式的支持由筮模块提供，该模块主要包含一些函数接口和Patteni对象， 用户可通过函数接口和操作Pattern对象的方法，对模式与模式匹配结果进行操作。re模块中常 用的函数及方法如表15・3所示。

表俘3 re模块函数及方法

|  |  |
| --- | --- |
| 函 数 | 说 明 |
| compile() | 对正则表达式进行编译，并返回一个Paitem对象 |
| match() | 从头匹配，匹配成功返回匹配对象，失败返回None |
| search() | 从任意位置开始匹配，匹配成功返回匹配对象，否则返回None |
| spiitQ | 将目标对象使用正则对象分割，成功返回匹配对象(是一个列表)，可指定最大分割次数 |
| findall() | 在目标对象中从左至右查找与正则对象匹配的所有非重叠子串，将这些子串组成一个列表并返回 |
| finditei'O | 功能与findallO相同，但返回的是迭代器对象iterator |
| sub() | 搜索目标对象中与正则对象匹配的子串，使用指定字符串替换,并返回替换后的对象 |
| subn() | 搜索目标对象中与正则对象匹配的子串，使用指定字符串替换，返回替换后的对象和替换次数 |
| group。 | 返回全部匹配对象 |
| groups() | 以元组形式返回所有子组的匹配结果，若匹配失败或正则表达式中不包含子组，返回空元组 |

其中compile()是re模块的函数，返回值为一个正则对象；gn)iip()和groups()是匹配对象 的方法；其余的是正则对象的方法，这些方法大多在孙模块中也有对应的函数实现，因此用户 可通过如下两种方式对文本进行模式匹配。

(1)定义一个正则对象，通过“对象•方法”的方式实现。

(2 )通过"re.函数”的方式实现。

下面将对R模块中的内容进行讲解。

:15o 12颓编译

转模块的compileQ函数可以对正则表达式进行预编译，该函数调用成功将会返回一个正 则表达式对象，函数的声明如下：

compile (pattern^ flags=0)

其中参数pattern为一个正则表达式，参数Hags是可选标识符，默认值为0,亦可使用 如表15・4所示的取值。

表15Y脂gs常用取值

|  |  |
| --- | --- |
| flags | 说 明 |
| re.I | 使匹配对大小写不敏感 |
| re.L | 做本地化识别(locale-aware)匹配，使预定义字符集\w、\W、\b、旧、\s、\5取决于当前区域设定 |
| re.M | 多行匹配，影响人和$ |
| re.S | 使.匹配所有字符，包括换行符 |
| re.U | 根据Unicode字符集解析字符 |
| re.A | 根据ASCII字符集解析字符 |
| re.X | 允许使用更灵活的格式(可以是多行、忽略空白字符、可加入注释)书写正则表达式，以便表达式更易理解 |
| Hags支持使用单一选项，也支持使用T'连接表15N中的多个选项。 compileQ函数的用法如下所示。  >>> import re  >>> regex = re.compile(r1\w+o\w+\*) | |

此时我们获取到了一个名为regex的正则对象，通过该对象可以匹配符合“以一个或多个 下画线、任意数字或字母开头,之后紧跟一个字母0,以一个或多个下画线、任意数字或字母结尾” 这种模式的文本。

假设现有一字符串aAnd slowly read,and dream of the soft lookJ，,通过regex对象匹配该字 符串的方法以及匹配结果如下所示。

»> words = \* And slowly readz and dream of the soft look,

>>> regex.findall (words)

[\* slowly', \* soft1# ,look \*]

其中findall()是m模块定义的方法，用于获取目标文本中所有符合条件的内容。观察以上 匹配结果，字符串中符合条件的内容被放在同一个列表中并返回。

四配与搜紊

re模块中的match()函数和search()函数用于匹配和搜索目标文本中与正则表达式模式相 同的内容。

15J3-1 mat©h()

match。是re模块的函数，也是操作Pattern对象的方法，它的功能是检测目标文本的开始

位置是否符合指定模式，若匹配成功则返回一个匹配对象，否则返回None。imtch()函数的声 明如下：

match (pattern^ string^ flags=O)

其中参数pattern为一个正则表达式，参数曲ing是待匹配的目标文本，参数Hags是可选 标识符。

下面以两个字符串为例，分别对match()函数与match方法的用法进行讲解。

>» word\_one = !And slowly read,and dream of the soft look\*

>» word\_two = v slowly read,and dream of the soft lookv

(1) match()函数

以模式"\w+o\w+”匹配目标对象si、s2, match()函数的用法如下所示。

>» import re

>>> re.match(r\*\w+o\w+1fword\_one)

»> re .match (r 1 \w+o\w+ ' r word\_two)

<\_sre . SRE\_Match object; span= (0, 6) , match= 1 slowly1 > # 匹配信息

结合匹配结果进行分析：目标文本wgd\_og开始位置的单词字符串And不符合模式 "\w+o\w+"，match()函数返回None,没有打印结果；目标文本word\_two的开始位置的单词 slowly符合模式"\w+o\w+" 9解释器打印match。函数的匹配信息，其匹配信息中包含两项， 其中spaii表示匹配对象在目标文本中岀现的位置，match表示匹配对象本身的内容。

Pattern对象也可以作为match。函数的pattern参数，因此可使用如下方式为match。函数 传参：

>>>regex = re.compile(r \*\w+o\w+ \*)

»> re .match (regex7 word\_two)

<\_sre.SRE\_Match object; span= (0f 6)f match=\* slowly \*>

使用这种方式，可在一定程度上提升程序的效率，但因缓存空间有限，用户还应酌情考虑 是否先对正则表达式进行预编译。

(2 ) match 方法

用户亦可使用“对象•方法"的形式调用match(),以15J2节中获取的正则对象:regex为例, match方法的使用方式如下所示。

»> regex«match (word\_one)

>>> regex.match(word\_two)

<\_sre o SRE\_Match object; span= (0 f

*#传入参数为*目标文本word\_one

#传入参数为目标文本word\_two

6)' match= \* slowly'>

对比两种形式，可知match()函数与match方法可实现相同的功能。

其他方法和函数的用法与match类似，之后只以函数的功能与用法为例进行 讲解。

15J3a2 s@arch()

虽然也有需要匹配文本开头内容的情况，但大部分情况下，需要匹配的是出现在文本任意 位置的字符串，这项功能由於模块中的search()函数实现，若调用search()函数匹配成功会返 回一个匹配对象,否则返回None。search()函数的声明如下：

search (pattern, string^ flags=0)

其中参数pattern为一个正则表达式，参数string是待匹配的目标文本，参数flags是可选 标识符。

下面通过对比来展示search()函数的功能。

>>> import re

>>> word = \* slowly1

>>> re . match ( ' ow \* f word) #match ()函数返回 None

>>> re . search ( ! ow \* , word) # 调用 search ()函数进行匹配

<\_sre • SRE\_\_Match obj ect; span= (2 f 4) f match= ? ow , > #search ()函数匹配结果 观察以上函数执行结果：match。函数匹配结果为空，这是因为字符串"slowly”的开头不 符合模式"ow" ; search()函数成功匹配到符合模式"ow"的内容，并成功返回。

15o 14四配瑚象

使用match()函数和search()函数进行正则匹配时，返回的不是单一的匹配结果，而是如 下形式的字符串：

<\_sre . SRE\_Match object; span= (2, 4) , match= 1 ow1 > #search ()函数匹配结果

该字符串表明返回结果是一个匹配对象，其中主要包含两项内容，分别为span和match, spaii表示本次获取的匹配对象在原目标文本中所处的位置，目标文本的下标从0开始；match 表示匹配对象的内容。

span属性是一个元组,元组中有两个元素，第一个元素表示匹配对象在目标文本中的开始 位置，第二个元素表示匹配对象在目标文本中的结束位置。如上所示的字符串中，匹配对象“ow” 在原目标文本中的起始位置为2,结束位置为4o

好模块中提供了一些与匹配对象相关的方法，用于获取匹配结果中的各项数据，具体如 表15-5所示。

表佑・5匹配对象常用方法

|  |  |
| --- | --- |
| 函数’ | 说明 |
| group([num]) | 获取匹配的字符串，或获取第num个子组的匹配结果 |
| start() | 获取匹配对象的开始位置 |
| end() | 获取匹配对象的结束位置 |
| spanQ | 获取表示匹配对象位置的元组 |

以searchQ函数的匹配结果为例，表15-5中方法的用法如下所示。

>>> import re

>>> word = s hello itheima \*

»> match\_result = re . search (r ' \whe\w' z word)

>>> match\_result # 匹配结果

<\_sre.SRE\_Match object; span=(7, 11)z match=!thei ? >

>>> match\_result. group () # 匹配对象

5 thei'

>>> match\_result o start() 7

#起始位置，

#结束位置

#(起始位置*，*结束位置)

»> match\_result. end ()

11

>>> match\_result.span() ⑺11)

当正则表达式中包含子组时，pythoii解释器会将每个子组的匹配结果临时存储到缓冲区中, 若用户想获取和子组的匹配结果，可使用匹配对象的方法group(),示例如下：

>>> words = re.search("(h) (e)"7 shello heooo')

»> words . group (1) #获取第一个子组的匹配结果

，h星

此外，匹配对象还有一个名为groups()的方法，使用该方法可以获取一个包含所有子组匹 配结果的元组，示例如下：

>>> words = re.search("(h) (e)n r 1 hello heooo5)

>>> words o groups()

(!h\ »e?)

若正则表达式中不包含子组,则groups()方法返回一个空元组。

15o!5全曳匹配

match()函数只检测文本开头的内容是否符合指定模式;search()函数只会返回文本中第一 个符合指定模式的匹配对象，即便其中有多个符合条件的内容存在。相对这种操作，匹配目标 文本中符合条件的全部匹配项的操作更为常见。

跆模块中提供了 find疽1()、finditerQ函数，用于匹配目标对象中所有符合指定模式的字 符串。

15J5J fmdaiK)

findall()函数可以获取目标文本中所有与指定正则表达式模式相同的字符串，并将所有匹 配结果以列表的形式返回。findall()函数的声明如下：

findall (pattern^ string^ flags=0)

其中参数pattern为一个正则表达式，参数\*ing是待匹配的目标文本，参数Hags是可选 标识符。

以字符串 u And slowly read,and dream of the soft look'5 为例,findall()函数的用法如下所示。

>>> import re

>>> words = "And slowly read^ and dream of the soft look"

>>> match\_result = re .findall (r 1 \wo\w1 r word)

>>> match\_result

[slow\ » sof ? z 'loo'] # 匹配结果

比较字符串words与以上所示的findall()函数调用结果，可知words中所有符合模式"\wo\w”

的字符串被返回。

因为findaHO函数的返回结果是一个列表，所以用户可以使用操作列表的方式，对返回结 果进行操作，例如要获取其中第二个匹配项，则可使用如下方式获取：

>>> match\_result[1]

1 sof 1

由match\_result[l]的输出结果“sof"可知，成功获取了匹配结果中的第二个匹配对象。

15J5-2

finditer()函数同样用于获取目标文本中所有与指定正则表达式模式相同的字符串，但该函 数会将所有匹配结果放在一个迭代器中，用户需以迭代的形式获取匹配结果。finditeK)函数的 声明如下：

finditer (pattern^ string^ flags=0)

其中参数pattern为一个正则表达式，参数string是待匹配的目标文本，参数flags是可选 标识符。

finiterQ函数的用法如下所示。

>>> import re

>>> words = "And slowly readf and dream of the soft lookn

>>> result = re sfinditer (r s \wo\w? r words)

>>> result

<callable\_iterato]： object at 0x0056C330> # 迭代器 result 的地址信息

>>> for m\_ob j in result: #迭代获取result中的匹配对象

。。“ print(m\_obj)

<\_sre » SRE\_Match object; span= (5z 8)z match=1 low5 >

<\_sre•SRE\_Match object; span= (33, 36)f match=,sof\*>

<\_sre.SRE\_Match object; span= (38, 41)r match=9 loo 5>

15a16檢素替換

关模块中提供的sub()^ subn()函数用于替换目标文本中的匹配项，这两个函数的声明分别 如下所示。

sub(pattern, repl, string, count=0, flags=0)

subn (pattern, repl, string, count=0, flags=0)

其中参数pattern表示正则表达式；参数repl表示用于替换的字符串;参数string表示目标 文本；参数count表示替换的次数，默认值0表示替换所有的匹配项。

snb()函数与simb()函数的参数及功能相同，不同的是若调用成功，sub()函数会返回替换 后的字符串，subn()函数会返回包含替换结果和替换次数的元组。这两个函数的用法如下所示。

>>> import re

>» words = ,And slowly readzand dream of the soft look'

>» result\_one = re . sub (r1 \s 1 z ? ?, words) #sub ()函数的用法

>>> result\_one

\* And-slowly-read,and-dream-of-the-soft-look?

»> result\_two = re . subn (r 1 \s \ ' -1 f words) #subn ()函数的用法

>>> result\_two

('And-slowly-read,and-dream-of-the-soft-lookf, 7)

17曳H割

re模块中提供的split。函数,可使用与正则表达式模式相同的字符串分割指定文本。split() 函数的声明如下：

split (pattern^ string, maxsplit=0, flags=0)

其中参数pattern表7K正则表达式；参数string表示待分割的文本；参数maxsplit用于设定 分割的次数，其默认值为0,表示匹配指定模式并全部进行分割；参数朗sgs表示可选标识符。 split()函数调用成功后，分割出的子项会被保存到列表中并返回。

以字符串"And slowly read,and dream of the soft look”为例,split()函数的用法如下所示。

>>> import re

>>> words = 1 And slowly read,and dream of the soft look'

»> result = re . split (r ? \s \* f words) # 以''\s〃分割字符串 s

»> result #分割结果

['And? r , slowly!f !read7 and!f ? dream5 f f of 5 z \* the 5 f !soft1f \* look?]

观察分割结果，字符串s以空格分割的子项被存储到了列表之中。

15ol8贪礬匹配

正则表达式中有两种匹配方式：贪婪匹配和非贪婪匹配。所谓贪婪匹配，即在条件满足的 情况下，尽量多地进行匹配；反之若尽量少地进行匹配，则为非贪婪匹配。python中正则表达 式的默认匹配方式为贪婪匹配。

以字符串 u And slowly read?and dream of the soft look59 为例,假设使用正则表达式 uand\s.\*n 对该字符串进行匹配，示例如下：

>>> import re

>>> words = v And slowly read^ and dream of the soft lookf

>>> result = re.search(r \*and\s.\* 17 words)

则匹配结果如下：

>>> result.group ()

'and dream of the soft look \*

正则表达式的含义为：匹配以字符串and开头，之后紧接一个空格，空格后有 零个或多个字符的字符串。观察匹配结果，正则表达式中的匹配了从Fnd”到字符串 words结尾的所有字符，这样的匹配便是贪婪匹配。

贪婪匹配方式也称为匹配优先，即在可匹配可不匹配时，优先尝试匹配；非贪婪匹配方式 也称忽略优先，即在可匹配可不匹配时，优先尝试忽略。这两种匹配方式总是体现在重复匹配+”“{}”)默认为匹配优先，但当其与“?”搭配, "{}?”这些形式出现时，则为忽略优先。

即使用正则表达式％nd\s°\*?”进行匹配，示例如下：

slowly readz and dream of the

中，重复匹配中使用的元字符(“?”“\* 即以 “??，， “\*?，， %?，，

若使用非贪婪方式，

>>> import re

>>> words = , And

>>> result = re o search(r!and\s.\*? ? zwords)

则匹配结果如下：

>>> result o group()

!and \*

观察匹配结果，正则表达式中的‘爭”匹配了零个字符。 类似地,若使用“and\s.+”与“and\s.+?”匹配字符串s, >>> re.search(r \*and\s « + !zwords) .group()

!and dream of the soft look?

>>> re.search(r Jand\s.+? 5 z s) .group()

'and d!

观察以上匹配结果，可知在贪婪匹配方式中，表达式匹配了尽量多的字符；在非贪婪匹配 方式中,表达式仅匹配了一个字符。

soft look J

则匹配结果分别如下所示。

#贪婪匹配

#非贪婪匹配