2-19正则表达式

1. 课堂案例

课堂讲解

一、上节回顾

无刷新分页

二、学习目标

正则表达式符号

三、教学过程描述

1、正则表达式

1-1、什么是正则？

正则表达式(Regular Expression)是一种文本模式，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"）。

1-2、语法

正则表达式(regular expression)描述了一种字符串匹配的模式（pattern），可以用来检查一个串是否含有某种子串、将匹配的子串替换或者从某个串中取出符合某个条件的子串等。

构造正则表达式的方法和创建数学表达式的方法一样。也就是用多种元字符与运算符可以将小的表达式结合在一起来创建更大的表达式。正则表达式的组件可以是单个的字符、字符集合、字符范围、字符间的选择或者所有这些组件的任意组合。

正则表达式是由普通字符（例如字符 a 到 z）以及特殊字符（称为"元字符"）组成的文字模式。模式描述在搜索文本时要匹配的一个或多个字符串。正则表达式作为一个模板，将某个字符模式与所搜索的字符串进行匹配。

1-2-1、普通字符

普通字符包括没有显式指定为元字符的所有可打印和不可打印字符。这包括所有大写和小写字母、所有数字、所有标点符号和一些其他符号。

1-2-2、非打印字符

非打印字符也可以是正则表达式的组成部分。下表列出了表示非打印字符的转义序列：

字符	描述
\cx	匹配由x指明的控制字符。例如， \cM 匹配一个 Control-M 或回车符。x 的值必须为 A-Z 或 a-z 之一。否则，将 c 视为一个原义的 'c' 字符。
\f	匹配一个换页符。等价于 \x0c 和 \cL。
\n	匹配一个换行符。等价于 \x0a 和 \cJ。
\r	匹配一个回车符。等价于 \x0d 和 \cM。
\s	匹配任何空白字符，包括空格、制表符、换页符等等。等价于 [\f\n\r\t\v]。
\S	匹配任何非空白字符。等价于 [^ \f\n\r\t\v]。
\t	匹配一个制表符。等价于 \x09 和 \cI。
\v	匹配一个垂直制表符。等价于 \x0b 和 \cK。

1-2-3、特殊字符

所谓特殊字符，就是一些有特殊含义的字符，如： runoo*b 中的 *，简单的说就是表示任何字符串的意思。如果要查找字符串中的 * 符号，则需要对 * 进行转义，即在其前加一个 \: runo*ob 匹配 runo*ob。

许多元字符要求在试图匹配它们时特别对待。若要匹配这些特殊字符，必须首先使字符"转义"，即，将反斜杠字符\ 放在它们前面。下表列出了正则表达式中的特殊字符：

特别字符	描述
$	匹配输入字符串的结尾位置。如果设置了 RegExp 对象的 Multiline 属性，则 $ 也匹配 '\n' 或 '\r'。要匹配 $ 字符本身，请使用 \$。
( )	标记一个子表达式的开始和结束位置。子表达式可以获取供以后使用。要匹配这些字符，请使用 \(和 \)。
*	匹配前面的子表达式零次或多次。要匹配 * 字符，请使用 *。
+	匹配前面的子表达式一次或多次。要匹配 + 字符，请使用 \+。
.	匹配除换行符 \n 之外的任何单字符。要匹配 . ，请使用 \. 。
[	标记一个中括号表达式的开始。要匹配 [，请使用 \[。
?	匹配前面的子表达式零次或一次，或指明一个非贪婪限定符。要匹配 ? 字符，请使用 \?。
\	将下一个字符标记为或特殊字符、或原义字符、或向后引用、或八进制转义符。例如， 'n' 匹配字符 'n'。'\n' 匹配换行符。序列 '\\' 匹配 "\"，而 '\(' 则匹配 "("。
^	匹配输入字符串的开始位置，除非在方括号表达式中使用，此时它表示不接受该字符集合。要匹配 ^ 字符本身，请使用 \^。
{	标记限定符表达式的开始。要匹配 { ，请使用 \{。
|	指明两项之间的一个选择。要匹配 | ，请使用 \\|。

1-2-4、限定符

限定符用来指定正则表达式的一个给定组件必须要出现多少次才能满足匹配。有 * 或 + 或 ? 或 {n} 或 {n,} 或 {n,m} 共6种。

正则表达式的限定符有:

字符	描述
*	匹配前面的子表达式零次或多次。例如，zo* 能匹配 "z" 以及 "zoo"。* 等价于{0,}。
+	匹配前面的子表达式一次或多次。例如，'zo+' 能匹配 "zo" 以及 "zoo"，但不能匹配 "z"。+ 等价于{1,}。
?	匹配前面的子表达式零次或一次。例如，"do(es)?" 可以匹配 "do" 或 "does" 中的"do" 。? 等价于{0,1}。
{n}	n 是一个非负整数。匹配确定的 n 次。例如，'o{2}' 不能匹配 "Bob" 中的 'o'，但是能匹配 "food" 中的两个 o。
{n,}	n 是一个非负整数。至少匹配n 次。例如，'o{2,}' 不能匹配 "Bob" 中的 'o'，但能匹配 "foooood" 中的所有 o。'o{1,}' 等价于 'o+'。'o{0,}' 则等价于 'o*'。
{n,m}	m 和 n 均为非负整数，其中n <= m。最少匹配 n 次且最多匹配 m 次。例如，"o{1,3}" 将匹配 "fooooood" 中的前三个 o。'o{0,1}' 等价于 'o?'。请注意在逗号和两个数之间不能有空格。

1、量词

{m,n}:m到n个

*:任意多个

+:一个到多个

？:0或一个

2、贪婪

贪婪模式在整个表达式匹配成功的前提下，尽可能多的匹配

3、非贪婪

非贪婪模式在整个表达式匹配成功的前提下，尽可能少的匹配。非贪婪模式只被部分NFA引擎所支持

4、定位符

定位符使您能够将正则表达式固定到行首或行尾。它们还使您能够创建这样的正则表达式，这些正则表达式出现在一个单词内、在一个单词的开头或者一个单词的结尾。

定位符用来描述字符串或单词的边界，^ 和 $ 分别指字符串的开始与结束，span class="marked">\b 描述单词的前或后边界，span class="marked">\B 表示非单词边界。

正则表达式的限定符有:

字符	描述
^	匹配输入字符串开始的位置。如果设置了 RegExp 对象的 Multiline 属性，^ 还会与 \n 或 \r 之后的位置匹配。
$	匹配输入字符串结尾的位置。如果设置了 RegExp 对象的 Multiline 属性，$ 还会与 \n 或 \r 之前的位置匹配。
\b	匹配一个字边界，即字与空格间的位置。
\B	非字边界匹配。

注意:不能将限定符与定位点一起使用。由于在紧靠换行或者字边界的前面或后面不能有一个以上位置，因此不允许诸如 ^* 之类的表达式。

若要匹配一行文本开始处的文本，请在正则表达式的开始使用 ^ 字符。不要将 ^ 的这种用法与中括号表达式内的用法混淆。

若要匹配一行文本的结束处的文本，请在正则表达式的结束处使用 $ 字符。

5、后向引用

使用小括号指定一个子表达式后，匹配这个子表达式的文本(也就是此分组捕获的内容)可以在表达式或其它程序中作进一步的处理。默认情况下，每个分组会自动拥有一个组号，规则是:从左向右，以分组的左括号为标志，第一个出现的分组的组号为1，第二个为2，以此类推。

呃……其实,组号分配还不像我刚说得那么简单:

分组0对应整个正则表达式

实际上组号分配过程是要从左向右扫描两遍的:第一遍只给未命名组分配，第二遍只给命名组分配－－因此所有命名组的组号都大于未命名的组号

你可以使用(?:exp)这样的语法来剥夺一个分组对组号分配的参与权.

后向引用用于重复搜索前面某个分组匹配的文本。例如，\1代表分组1匹配的文本。

1-3、元字符

字符	描述
\	将下一个字符标记为一个特殊字符、或一个原义字符、或一个 向后引用、或一个八进制转义符。例如，'n' 匹配字符 "n"。'\n' 匹配一个换行符。序列 '\\' 匹配 "\" 而 "\(" 则匹配 "("。
^	匹配输入字符串的开始位置。如果设置了 RegExp 对象的 Multiline 属性，^ 也匹配 '\n' 或 '\r' 之后的位置。
$	匹配输入字符串的结束位置。如果设置了RegExp 对象的 Multiline 属性，$ 也匹配 '\n' 或 '\r' 之前的位置。
*	匹配前面的子表达式零次或多次。例如，zo* 能匹配 "z" 以及 "zoo"。* 等价于{0,}。
+	匹配前面的子表达式一次或多次。例如，'zo+' 能匹配 "zo" 以及 "zoo"，但不能匹配 "z"。+ 等价于{1,}。
?	匹配前面的子表达式零次或一次。例如，"do(es)?" 可以匹配 "do" 或 "does" 中的"do" 。? 等价于{0,1}。
{n}	n 是一个非负整数。匹配确定的 n 次。例如，'o{2}' 不能匹配 "Bob" 中的 'o'，但是能匹配 "food" 中的两个 o。
{n,}	n 是一个非负整数。至少匹配n 次。例如，'o{2,}' 不能匹配 "Bob" 中的 'o'，但能匹配 "foooood" 中的所有 o。'o{1,}' 等价于 'o+'。'o{0,}' 则等价于 'o*'。
{n,m}	m 和 n 均为非负整数，其中n <= m。最少匹配 n 次且最多匹配 m 次。例如，"o{1,3}" 将匹配 "fooooood" 中的前三个 o。'o{0,1}' 等价于 'o?'。请注意在逗号和两个数之间不能有空格。
?	当该字符紧跟在任何一个其他限制符 (*, +, ?, {n}, {n,}, {n,m}) 后面时，匹配模式是非贪婪的。非贪婪模式尽可能少的匹配所搜索的字符串，而默认的贪婪模式则尽可能多的匹配所搜索的字符串。例如，对于字符串 "oooo"，'o+?' 将匹配单个 "o"，而 'o+' 将匹配所有 'o'。
.	匹配除 "\n" 之外的任何单个字符。要匹配包括 '\n' 在内的任何字符，请使用像"**(.|\n)**"的模式。
(pattern)	匹配 pattern 并获取这一匹配。所获取的匹配可以从产生的 Matches 集合得到，在VBScript 中使用 SubMatches 集合，在JScript 中则使用 $0...$9 属性。要匹配圆括号字符，请使用 '\(' 或 '\)'。
(?:pattern)	匹配 pattern 但不获取匹配结果，也就是说这是一个非获取匹配，不进行存储供以后使用。这在使用 "或" 字符 (|) 来组合一个模式的各个部分是很有用。例如， 'industr(?:y|ies) 就是一个比 'industry|industries' 更简略的表达式。
(?=pattern)	正向预查，在任何匹配 pattern 的字符串开始处匹配查找字符串。这是一个非获取匹配，也就是说，该匹配不需要获取供以后使用。例如，'Windows (?=95|98|NT|2000)' 能匹配 "Windows 2000" 中的 "Windows" ，但不能匹配 "Windows 3.1" 中的 "Windows"。预查不消耗字符，也就是说，在一个匹配发生后，在最后一次匹配之后立即开始下一次匹配的搜索，而不是从包含预查的字符之后开始。
(?!pattern)	负向预查，在任何不匹配 pattern 的字符串开始处匹配查找字符串。这是一个非获取匹配，也就是说，该匹配不需要获取供以后使用。例如'Windows (?!95|98|NT|2000)' 能匹配 "Windows 3.1" 中的 "Windows"，但不能匹配 "Windows 2000" 中的 "Windows"。预查不消耗字符，也就是说，在一个匹配发生后，在最后一次匹配之后立即开始下一次匹配的搜索，而不是从包含预查的字符之后开始。
x|y	匹配 x 或 y。例如，'z|food' 能匹配 "z" 或 "food"。'(z|f)ood' 则匹配 "zood" 或 "food"。
[xyz]	字符集合。匹配所包含的任意一个字符。例如， '[abc]' 可以匹配 "plain" 中的 'a'。
[^xyz]	负值字符集合。匹配未包含的任意字符。例如， '[^abc]' 可以匹配 "plain" 中的'p'、'l'、'i'、'n'。
[a-z]	字符范围。匹配指定范围内的任意字符。例如，'[a-z]' 可以匹配 'a' 到 'z' 范围内的任意小写字母字符。
[^a-z]	负值字符范围。匹配任何不在指定范围内的任意字符。例如，'[^a-z]' 可以匹配任何不在 'a' 到 'z' 范围内的任意字符。
\b	匹配一个单词边界，也就是指单词和空格间的位置。例如， 'er\b' 可以匹配"never" 中的 'er'，但不能匹配 "verb" 中的 'er'。
\B	匹配非单词边界。'er\B' 能匹配 "verb" 中的 'er'，但不能匹配 "never" 中的 'er'。
\cx	匹配由 x 指明的控制字符。例如， \cM 匹配一个 Control-M 或回车符。x 的值必须为 A-Z 或 a-z 之一。否则，将 c 视为一个原义的 'c' 字符。
\d	匹配一个数字字符。等价于 [0-9]。
\D	匹配一个非数字字符。等价于 [^0-9]。
\f	匹配一个换页符。等价于 \x0c 和 \cL。
\n	匹配一个换行符。等价于 \x0a 和 \cJ。
\r	匹配一个回车符。等价于 \x0d 和 \cM。
\s	匹配任何空白字符，包括空格、制表符、换页符等等。等价于 [\f\n\r\t\v]。
\S	匹配任何非空白字符。等价于 [^ \f\n\r\t\v]。
\t	匹配一个制表符。等价于 \x09 和 \cI。
\v	匹配一个垂直制表符。等价于 \x0b 和 \cK。
\w	匹配包括下划线的任何单词字符。等价于'[A-Za-z0-9_]'。
\W	匹配任何非单词字符。等价于 '[^A-Za-z0-9_]'。
\xn	匹配 n，其中 n 为十六进制转义值。十六进制转义值必须为确定的两个数字长。例如，'\x41' 匹配 "A"。'\x041' 则等价于 '\x04' & "1"。正则表达式中可以使用 ASCII 编码。
\num	匹配 num，其中 num 是一个正整数。对所获取的匹配的引用。例如，'(.)\1' 匹配两个连续的相同字符。
\n	标识一个八进制转义值或一个向后引用。如果 \n 之前至少 n 个获取的子表达式，则 n 为向后引用。否则，如果 n 为八进制数字 (0-7)，则 n 为一个八进制转义值。
\nm	标识一个八进制转义值或一个向后引用。如果 \nm 之前至少有 nm 个获得子表达式，则 nm 为向后引用。如果 \nm 之前至少有 n 个获取，则 n 为一个后跟文字 m 的向后引用。如果前面的条件都不满足，若 n 和 m 均为八进制数字 (0-7)，则 \nm 将匹配八进制转义值 nm。
\nml	如果 n 为八进制数字 (0-3)，且 m 和 l 均为八进制数字 (0-7)，则匹配八进制转义值 nml。
\un	匹配 n，其中 n 是一个用四个十六进制数字表示的 Unicode 字符。例如， \u00A9 匹配版权符号 (?)。

1-4、运算符优先级

正则表达式从左到右进行计算，并遵循优先级顺序，这与算术表达式非常类似。

相同优先级的从左到右进行运算，不同优先级的运算先高后低。下表从最高到最低说明了各种正则表达式运算符的优先级顺序：

运算符	描述
\	转义符
(), (?:), (?=), []	圆括号和方括号
*, +, ?, {n}, {n,}, {n,m}	限定符
^, $, \任何元字符、任何字符	定位点和序列（即：位置和顺序）
|	替换，"或"操作 字符具有高于替换运算符的优先级，使得"m|food"匹配"m"或"food"。若要匹配"mood"或"food"，请使用括号创建子表达式，从而产生"(m|f)ood"。

1-5、模式修正法

模式修正符号	功能描述
i	在和正则匹配是不区分大小写
m	将字符串视为多行。默认的正则开始“^”和结束“$”将目标字条串作为一单一的一“行”字符（甚至其中包括换行符也是如此）。如果在修饰符中加上“m”，那么开始和结束将会指点字符串的每一行的开头就是“^”结束就是“$”。
s	如果设定了这个修正符，那么，被匹配的字符串将视为一行来看，包括换行符，换行符将被视为普通字符串。
x	忽略空白，除非进行转义的不被忽略。
e	只用在preg_replace()函数中，在替换字符串中逆向引用做正常的替换，将其(即“替换字符串”)作为PHP代码求值，并用其结果来替换所搜索的字符串。
A	如果使用这个修饰符，那么表达式必须是匹配的字符串中的开头部分。比如说”/a/A”匹配”abcd”。
D	模式中的$字符权匹配目标字符的结尾。没有此选项时，如果最后一个字符是换行符的话，美元符号也会匹配此字符之前。如果设定了修正符m则忽略此项。
E	与”m”相反，如果使用这个修饰符，那么”$”将匹配绝对字符串的结尾，而不是换行符前面，默认就打开了这个模式。
U	贪婪模式，和问号的作用差不多，最大限度的匹配就是贪婪模式。

2、PHP正则表达式函数

2-1、preg_match

执行匹配正则表达式

preg_match (string $pattern , string $subject [, array &$matches [, int $flags = 0 [, int $offset = 0]]])

说明

参数说明：

$pattern: 要搜索的模式，字符串形式。

$subject: 输入字符串。

$matches: 如果提供了参数matches，它将被填充为搜索结果。
$matches[0]将包含完整模式匹配到的文本，$matches[1]
将包含第一个捕获子组匹配到的文本，以此类推。
$flags：flags 可以被设置为以下标记值：
PREG_OFFSET_CAPTURE:
如果传递了这个标记，对于每一个出现的匹配返回时会附加字符串偏移量(相对于目标字符串的)。
注意：这会改变填充到matches参数的数组，使其每个元素成为一个由
第0个元素是匹配到的字符串，第1个元素是该匹配字符串 在目标字符串subject中的偏移量。
offset: 通常，搜索从目标字符串的开始位置开始。可选参数 offset 用于
指定从目标字符串的某个未知开始搜索(单位是字节)。
返回值
返回 pattern 的匹配次数。它的值将是 0 次（不匹配）或 1 次，因为 preg_match() 在第一次匹配后
将会停止搜索。preg_match_all() 不同于此，它会一直搜索subject 直到到达结尾。
如果发生错误preg_match()返回 FALSE。

2-2、preg_match_all

执行一个全局正则表达式匹配
说明
preg_match_all (string $pattern , string $subject [, array &$matches [, int $flags = PREG_PATTERN_ORDER [, int $offset = 0]]])
参数
$pattern: 要搜索的模式，字符串形式。
$subject: 输入字符串。
$matches: 如果提供了参数matches，它将被填充为搜索结果。
$matches[0]将包含完整模式匹配到的文本，$matches[1]
将包含第一个捕获子组匹配到的文本，以此类推。
$flags：flags 可以被设置为以下标记值：
PREG_OFFSET_CAPTURE:
如果传递了这个标记，对于每一个出现的匹配返回时会附加字符串偏移量(相对于目标字符串的)。
注意：这会改变填充到matches参数的数组，使其每个元素成为一个由
第0个元素是匹配到的字符串，第1个元素是该匹配字符串 在目标字符串subject中的偏移量。
offset: 通常，搜索从目标字符串的开始位置开始。可选参数 offset 用于
指定从目标字符串的某个未知开始搜索(单位是字节)。
返回值
返回 pattern 的匹配次数。它的值将是 0 次（不匹配）或 1 次，因为 preg_match() 在第一次匹配后
将会停止搜索。preg_match_all() 不同于此，它会一直搜索subject 直到到达结尾。
如果发生错误preg_match()返回 FALSE。

2-3、preg_quote

转义正则表达式字符

说明

preg_quote (string $str [, string $delimiter = NULL])

参数说明:

$str: 输入字符串。

$delimiter: 如果指定了可选参数 delimiter，它也会被转义。这通常用于 转义 PCRE 函数使用的分隔符。/ 是最通用的分隔符。

四、小结

1、贪婪模式和非贪婪模式的就？的区分

2、PHP执行正则表达式（PCRE）函数

2. 课堂练习

1、用贪婪模式和非贪婪模式匹配一个字符:'<td><img src="a.jpg"></td><td><img src="b.jpg"></td><td><img src="v.jpg"></td>

获取图片信息:Array ([0] => a.jpg [1] => b.jpg [2] => v.jpg)

3. 课后练习

1、写一个手机验证正则

2、邮箱验证正则

4. 资料扩展

1、preg_filter

执行一个正则表达式搜索和替换

preg_filter()等价于preg_replace() 除了它仅仅返回(可能经过转化)与目标匹配的结果.

这个函数怎样工作的更详细信息请阅读 preg_replace()文档.

说明

preg_filter (mixed $pattern , mixed $replacement , mixed $subject [, int $limit = -1 [, int &$count]])

返回值¶

如果subject是一个数组，返回一个数组，其他情况返回一个字符串。

如果没有找到匹配或者发生了错误，当subject是数组时返回一个空数组，其他情况返回NULL。

2、preg_grep

返回匹配模式的数组条目

说明

preg_grep (string $pattern , array $input [, int $flags = 0])

参数、、

pattern

要搜索的模式, 字符串形式.

input

输入数组.

flags

如果设置为PREG_GREP_INVERT, 这个函数返回输入数组中与 给定模式pattern不匹配的元素组成的数组.

返回值¶

返回使用input中key做索引的数组.

3、preg_last_error

返回最后一个PCRE正则执行产生的错误代码

说明

preg_last_error (void)

4、preg_replace_callback_array

函数执行一个正则表达式搜索并且使用一个回调进行替换。

说明

preg_replace_callback_array (array $patterns_and_callbacks , mixed $subject [, int $limit = -1 [, int &$count]])

参数说明:

$patterns_and_callbacks: 关联数组, key(模式) => value(回调函数)

$subject: 要查找的和替换的字符串或数组。

$limit: 可选, 每个模式最大的替换次数, 默认为 -1(无限制, 全部匹配完)。

$count: 可选, 指定替换的次数。

返回值

如果 subject 是一个数组返回数组, 否则返回字符串。发生错误则返回 NULL。

如果查找到了匹配, 返回替换后的目标字符串(或字符串数组), 其他情况 subject 将会无变化返回。

5、preg_replace_callback

执行一个正则表达式搜索并且使用一个回调进行替换

说明

mixed preg_replace_callback (mixed $pattern , callable $callback , mixed $subject [, int $limit = -1 [, int &$count]])

这个函数的行为除了 可以指定一个 callback 替代 replacement 进行替换 字符串的计算，其他方面等同于 preg_replace()。

参数说明：

$pattern: 要搜索的模式，可以使字符串或一个字符串数组。

$callback: 一个回调函数，在每次需要替换时调用，调用时函数得到的参数是从subject 中匹配到的结果。

$subject: 要搜索替换的目标字符串或字符串数组。

$limit: 可选，对于每个模式用于每个 subject 字符串的最大可替换次数。默认是-1（无限制）。

$count: 可选，为替换执行的次数。

返回值

如果subject是一个数组， preg_replace_callback()返回一个数组，其他情况返回字符串。 错误发生时返回 NULL。

如果查找到了匹配，返回替换后的目标字符串（或字符串数组），其他情况subject 将会无变化返回。

6、preg_replace

执行一个正则表达式的搜索和替换

说明

mixed preg_replace (mixed $pattern , mixed $replacement , mixed $subject [, int $limit = -1 [, int &$count]])

搜索subject中匹配pattern的部分，以replacement进行替换。

参数说明：

$pattern: 要搜索的模式，可以使字符串或一个字符串数组。

$replacement: 用于替换的字符串或字符串数组。

$subject: 要搜索替换的目标字符串或字符串数组。

$limit: 可选，对于每个模式用于每个 subject 字符串的最大可替换次数。默认是-1（无限制）。

$count: 可选，为替换执行的次数。

返回值

如果 subject 是一个数组，preg_replace() 返回一个数组，其他情况下返回一个字符串。

如果匹配被查找到，替换后的 subject 被返回，其他情况下 返回没有改变的 subject。如果发生错误，返回 NULL。

7、preg_split

通过一个正则表达式分隔字符串

说明

array preg_split (string $pattern , string $subject [, int $limit = -1 [, int $flags = 0]])

通过一个正则表达式分隔给定字符串.

参数

pattern

用于搜索的模式，字符串形式。

subject

输入字符串

limit

如果指定，将限制分隔得到的子串最多只有limit个，返回的最后一个 子串将包含所有剩余部分。limit值为-1，0或null时都代表"不限制"，作为php的标准，你可以使用null跳过对flags的设置。

flags

flags 可以是任何下面标记的组合(以位或运算 | 组合)：

PREG_SPLIT_NO_EMPTY

如果这个标记被设置，preg_split() 将进返回分隔后的非空部分。

PREG_SPLIT_DELIM_CAPTURE

如果这个标记设置了，用于分隔的模式中的括号表达式将被捕获并返回。

PREG_SPLIT_OFFSET_CAPTURE

如果这个标记被设置，对于每一个出现的匹配返回时将会附加字符串偏移量.

注意：这将会改变返回数组中的每一个元素，使其每个元素成为一个由第0个元素为分隔后的子串，第1个元素为该子串在subject 中的偏移量组成的数组。

返回值

返回一个使用 pattern 边界分隔 subject 后得到 的子串组成的数组，或者在失败时返回 FALSE。