**PHP正则表达式的快速学习方法**

1、入门简介

简单的说，正则表达式是一种可以用于模式匹配和替换的强有力的工具。我们可以在几乎所有的基于UNIX系统的工具中找到正则表达式的身影，例如，vi编辑器，Perl或PHP脚本语言，以及awk或sed shell程序等。此外，象JavaScript这种客户端的脚本语言也提供了对正则表达式的支持。由此可见，正则表达式已经超出了某种语言或某个系统的局限，成为人们广为接受的概念和功能。  
正则表达式可以让用户通过使用一系列的特殊字符构建匹配模式，然后把匹配模式与数据文件、程序输入以及WEB页面的表单输入等目标对象进行比较，根据比较对象中是否包含匹配模式，执行相应的程序。  
　　举例来说，正则表达式的一个最为普遍的应用就是用于验证用户在线输入的邮件地址的格式是否正确。如果通过正则表达式验证用户邮件地址的格式正确，用户所填写的表单信息将会被正常处理；反之，如果用户输入的邮件地址与正则表达的模式不匹配，将会弹出提示信息，要求用户重新输入正确的邮件地址。由此可见正则表达式在WEB应用的逻辑判断中具有举足轻重的作用。

2、基本语法

在对正则表达式的功能和作用有了初步的了解之后，我们就来具体看一下正则表达式的语法格式。  
正则表达式的形式一般如下：  
/love/  
其中位于“/”定界符之间的部分就是将要在目标对象中进行匹配的模式。用户只要把希望查找匹配对象的模式内容放入“/”定界符之间即可。为了能够使用户更加灵活的定制模式内容，正则表达式提供了专门的“元字符”。所谓元字符就是指那些在正则表达式中具有特殊意义的专用字符，可以用来规定其前导字符（即位于元字符前面的字符）在目标对象中的出现模式。  
较为常用的元字符包括： “+”， “\*”，以及 “?”。其中，“+”元字符规定其前导字符必须在目标对象中连续出现一次或多次，“\*”元字符规定其前导字符必须在目标对象中出现零次或连续多次，而“?”元字符规定其前导对象必须在目标对象中连续出现零次或一次。  
下面，就让我们来看一下正则表达式元字符的具体应用。  
/fo+/  
因为上述正则表达式中包含“+”元字符，表示可以与目标对象中的 “fool”, “fo”, 或者 “football”等在字母f后面连续出现一个或多个字母o的字符串相匹配。  
/eg\*/  
因为上述正则表达式中包含“\*”元字符，表示可以与目标对象中的 “easy”, “ego”, 或者 “egg”等在字母e后面连续出现零个或多个字母g的字符串相匹配。  
/Wil?/  
因为上述正则表达式中包含“？”元字符，表示可以与目标对象中的 “Win”, 或者 “Wilson”,等在字母i后面连续出现零个或一个字母l的字符串相匹配。  
除了元字符之外，用户还可以精确指定模式在匹配对象中出现的频率。例如，  
/jim{2,6}/  
上述正则表达式规定字符m可以在匹配对象中连续出现2-6次，因此，上述正则表达式可以同jimmy或jimmmmmy等字符串相匹配。  
在对如何使用正则表达式有了初步了解之后，我们来看一下其它几个重要的元字符的使用方式。  
\s：用于匹配单个空格符，包括tab键和换行符；  
\S：用于匹配除单个空格符之外的所有字符；  
\d：用于匹配从0到9的数字；  
\w：用于匹配字母，数字或下划线字符；  
\W：用于匹配所有与\w不匹配的字符；  
. ：用于匹配除换行符之外的所有字符。  
（说明：我们可以把\s和\S以及\w和\W看作互为逆运算）  
下面，我们就通过实例看一下如何在正则表达式中使用上述元字符。  
/\s+/  
上述正则表达式可以用于匹配目标对象中的一个或多个空格字符。  
/\d000/  
如果我们手中有一份复杂的财务报表，那么我们可以通过上述正则表达式轻而易举的查找到所有总额达千元的款项。

除了我们以上所介绍的元字符之外，正则表达式中还具有另外一种较为独特的专用字符，即定位符。定位符用于规定匹配模式在目标对象中的出现位置。  
较为常用的定位符包括： “^”, “$”, “\b” 以及 “\B”。其中，“^”定位符规定匹配模式必须出现在目标字符串的开头，“$”定位符规定匹配模式必须出现在目标对象的结尾，\b定位符规定匹配模式必须出现在目标字符串的开头或结尾的两个边界之一，而“\B”定位符则规定匹配对象必须位于目标字符串的开头和结尾两个边界之内，即匹配对象既不能作为目标字符串的开头，也不能作为目标字符串的结尾。同样，我们也可以把“^”和“$”以及“\b”和“\B”看作是互为逆运算的两组定位符。举例来说：  
/^hell/  
因为上述正则表达式中包含“^”定位符，所以可以与目标对象中以 “hell”, “hello”或 “hellhound”开头的字符串相匹配。  
/ar$/  
因为上述正则表达式中包含“$”定位符，所以可以与目标对象中以 “car”, “bar”或 “ar” 结尾的字符串相匹配。

/\bbom/  
因为上述正则表达式模式以“\b”定位符开头，所以可以与目标对象中以 “bomb”, 或 “bom”开头的字符串相匹配。  
/man\b/  
因为上述正则表达式模式以“\b”定位符结尾，所以可以与目标对象中以 “human”, “woman”或 “man”结尾的字符串相匹配。  
为了能够方便用户更加灵活的设定匹配模式，正则表达式允许使用者在匹配模式中指定某一个范围而不局限于具体的字符。例如：  
/[A-Z]/  
上述正则表达式将会与从A到Z范围内任何一个大写字母相匹配。  
/[a-z]/  
上述正则表达式将会与从a到z范围内任何一个小写字母相匹配。  
/[0-9]/  
上述正则表达式将会与从0到9范围内任何一个数字相匹配。  
/([a-z][A-Z][0-9])+/  
上述正则表达式将会与任何由字母和数字组成的字符串，如 “aB0” 等相匹配。这里需要提醒用户注意的一点就是可以在正则表达式中使用 “()” 把字符串组合在一起。“()”符号包含的内容必须同时出现在目标对象中。因此，上述正则表达式将无法与诸如 “abc”等的字符串匹配，因为“abc”中的最后一个字符为字母而非数字。  
如果我们希望在正则表达式中实现类似编程逻辑中的“或”运算，在多个不同的模式中任选一个进行匹配的话，可以使用管道符 “|”。例如：  
/to|too|2/  
上述正则表达式将会与目标对象中的 “to”, “too”, 或 “2” 相匹配。  
正则表达式中还有一个较为常用的运算符，即否定符 “[^]”。与我们前文所介绍的定位符 “^” 不同，否定符 “[^]”规定目标对象中不能存在模式中所规定的字符串。例如：  
/[^A-C]/  
上述字符串将会与目标对象中除A，B，和C之外的任何字符相匹配。一般来说，当“^”出现在 “[]”内时就被视做否定运算符；而当“^”位于“[]”之外，或没有“[]”时，则应当被视做定位符。  
最后，当用户需要在正则表达式的模式中加入元字符，并查找其匹配对象时，可以使用转义符“\”。例如：  
/Th\\*/  
上述正则表达式将会与目标对象中的“Th\*”而非“The”等相匹配。

3、使用实例

①PHP中可以使用ereg()函数进行模式匹配操作。ereg()函数的使用格式如下：

|  |
| --- |
| 以下为引用的内容： 　 ereg(pattern, string)  　　其中，pattern代表正则表达式的模式，而string则是执行查找替换操作的目标对象。同样是验证邮件地址，使用PHP编写的程序代码如下：  < ?php  　　 if (ereg(“^([a-zA-Z0-9\_-])+@([a-zA-Z0-9\_-])+(\.[a-zA-Z0-9\_-])+”,$email)){                echo “Your email address is correct!”;}  　　 else{                echo “Please try again!”;        }  ?> |

②JavaScript 1.2中带有一个功能强大的RegExp()对象，可以用来进行正则表达式的匹配操作。其中的test()方法可以检验目标对象中是否包含匹配模式，并相应的返回true或false。

我们可以使用JavaScript编写以下脚本，验证用户输入的邮件地址的有效性。

|  |
| --- |
| 以下为引用的内容： 　<html>  　　 <head>  　　　 <script language=\"Javascript1.2\">  　　　　 <!-- start hiding  　　　　 function verifyAddress(obj)  　　　　　{  　　　　　　var email = obj.email.value;  　　　　　　var pattern = /^([a-zA-Z0-9\_-])+@([a-zA-Z0-9\_-])+(\.[a-zA-Z0-9\_-])+/;  　　　　　　flag = pattern.test(email);  　　　　　　if(flag)  　　　　　　{  　　　　　　　alert(“Your email address is correct!”);  　　　　　　　return true;  　　　　　　}  　　　　　　else  　　　　　　　{  　　　　　　　　alert(“Please try again!”);  　　　　　　　　return false;  　　　　　　　 }  　　　　　 }  　　　　 // stop hiding -->  　　　 </script>  　　　</head>  　　 <body>  　　 <form onSubmit=\"return verifyAddress(this);\">  　　　<input name=\"email\" type=\"text\"/>  　　　<input type=\"submit\"/>  　　　</form>  　　</body>  　</html> |

想必很多人都对正则表达式都头疼。今天，我以我的认识，加上网上一些文章，希望用常人都可以理解的表达方式。来和大家分享学习经验。

开篇，还是得说说 ^ 和 $ 他们是分别用来匹配字符串的开始和结束，以下分别举例说明：

“^The”：开头一定要有”The”字符串；  
“of despair$”：结尾一定要有”of despair” 的字符串；

那么，  
“^abc$”：就是要求以abc开头和以abc结尾的字符串，实际上是只有abc匹配；  
“notice”：匹配包含notice的字符串；

你可以看见如果你没有用我们提到的两个字符（最后一个例子），就是说模式（正则表达式）可以出现在被检验字符串的任何地方，你没有把他锁定到两边。

接着，说说 ‘\*’ ‘+’ 和 ‘?’  
他们用来表示一个字符可以出现的次数或者顺序，他们分别表示：  
“zero or more”相当于{0,}  
“one or more”相当于{1,}  
“zero or one.”相当于{0,1}

这里是一些例子：

“ab\*”：和ab{0,}同义，匹配以a开头，后面可以接0个或者N个b组成的字符串(”a”, “ab”, “abbb”, 等)；  
“ab+”：和ab{1,}同义，同上条一样，但最少要有一个b存在 (”ab” “abbb”等)；  
“ab?”：和ab{0,1}同义，可以没有或者只有一个b；  
“a?b+$”：匹配以一个或者0个a再加上一个以上的b结尾的字符串。

要点：’\*’ ‘+’ 和 ‘?’ 只管它前面那个字符。

你也可以在大括号里面限制字符出现的个数，比如：

“ab{2}”: 要求a后面一定要跟两个b（一个也不能少）(”abb”)；  
“ab{2,}”: 要求a后面一定要有两个或者两个以上b(如”abb” “abbbb” 等)；  
“ab{3,5}”: 要求a后面可以有2－5个b(”abbb”, “abbbb”, or “abbbbb”)。

现在我们把一定几个字符放到小括号里，比如：

“a(bc)\*”: 匹配 a 后面跟0个或者一个”bc”；  
“a(bc){1,5}”: 一个到5个 “bc”；

还有一个字符 ‘|’，相当于OR操作：

“hi|hello”: 匹配含有”hi” 或者 “hello” 的 字符串；  
“(b|cd)ef”: 匹配含有 “bef” 或者 “cdef”的字符串；  
“(a|b)\*c”: 匹配含有这样多个（包括0个）a或b，后面跟一个c的字符串；

一个点(’.’)可以代表所有的单一字符，不包括”\n”

如果，要匹配包括”\n”在内的所有单个字符，怎么办？

用’[\n.]’这种模式。

“a.[0-9]”: 一个a加一个字符再加一个0到9的数字；  
“^.{3}$”: 三个任意字符结尾。

中括号括住的内容只匹配一个单一的字符

“[ab]”: 匹配单个的 a 或者 b ( 和 “a│b” 一样)；  
“[a-d]”: 匹配’a’ 到’d’的单个字符 (和”a│b│c│d” 还有 “[abcd]”效果一样)；

一般我们都用[a-zA-Z]来指定字符为一个大小写英文：

“^[a-zA-Z]”: 匹配以大小写字母开头的字符串；  
“[0-9]%”: 匹配含有 形如 x％ 的字符串；  
“,[a-zA-Z0-9]$”: 匹配以逗号再加一个数字或字母结尾的字符串；

你也可以把你不想要得字符列在中括号里，你只需要在总括号里面使用’^’ 作为开头 “%[^a-zA-Z]%” 匹配含有两个百分号里面有一个非字母的字符串。

要点：^用在中括号开头的时候，就表示排除括号里的字符。

为了PHP能够解释，你必须在这些字符面前后加”，并且将一些字符转义。

不要忘记在中括号里面的字符是这条规路的例外—在中括号里面，所有的特殊字符，包括(”)，都将失去他们的特殊性质 “[\*\+?{}.]”匹配含有这些字符的字符串：

还有，正如regx的手册告诉我们：”如果列表里含有’]’，最好把它作为列表里的第一个字符(可能跟在’^’后面)。如果含有’-’，最好把它放在最前面或者最后面， or 或者一个范围的第二个结束点[a-d-0-9]中间的‘-’将有效。

看了上面的例子，你对{n,m}应该理解了吧。要注意的是，n和m都不能为负整数，而且n总是小于m。这样，才能 最少匹配n次且最多匹配m次。如”p{1,5}”将匹配 “pvpppppp”中的前五个p

下面说说以\开头的

\b 书上说他是用来匹配一个单词边界，就是…比如’ve\b’，可以匹配love里的ve而不匹配very里有ve

\B 正好和上面的\b相反。例子我就不举了

…..突然想起来….可以到 http://www.phpv.net/article.php/251 看看其它用\开头的语法

好，我们来做个应用：如何构建一个模式来匹配货币数量的输入。

构建一个匹配模式去检查输入的信息是否为一个表示money的数字。我们认为一个表示money的数量有四种方式：”10000.00″ 和 “10,000.00″，或者没有小数部分，”10000″ and “10,000″。现在让我们开始构建这个匹配模式：

^[1-9][0-9]\*$

这是所变量必须以非0的数字开头。但这也意味着单一的”0″也不能通过测试。以下是解决的方法：

^(0|[1-9][0-9]\*)$

“只有0和不以0开头的数字与之匹配”，我们也可以允许一个负号在数字之前：

^(0|-?[1-9][0-9]\*)$

这就是：0或者一个以0开头且可能有一个负号在前面的数字。好了，现在让我们别那么严谨，允许以0开头。现在让我们放弃负号，因为我们在表示钱币的时候并不需要用到。我们现在指定模式用来匹配小数部分：

^[0-9]+(\.[0-9]+)?$

这暗示匹配的字符串必须最少以一个阿拉伯数字开头。但是注意，在上面模式中 “10.” 是不匹配的， 只有 “10″ 和 “10.2″ 才可以，你知道为什么吗？

^[0-9]+(\.[0-9]{2})?$

我们上面指定小数点后面必须有两位小数。如果你认为这样太苛刻，你可以改成：

^[0-9]+(\.[0-9]{1,2})?$

这将允许小数点后面有一到两个字符。现在我们加上用来增加可读性的逗号（每隔三位），我们可以这样表示：

^[0-9]{1,3}(,[0-9]{3})\*(\.[0-9]{1,2})?$

不要忘记’+’可以被’\*’替代如果你想允许空白字符串被输入话，也不要忘记反斜杆’\’在php字符串中可能会出现错误 (很普遍的错误)：

现在，我们已经可以确认字符串了，我们现在把所有逗号都去掉str\_replace(”,”, “”, $money)然后在把类型看成double然后我们就可以通过他做数学计算了。

再来一个：

构造检查email的正则表达式

在一个完整的email地址中有三个部分：

1. 用户名 (在 ‘@’ 左边的一切)  
2.’@’  
3. 服务器名(就是剩下那部分)

用户名可以含有大小写字母阿拉伯数字，句号(’.’)减号(’-’)and下划线’\_’)。服务器名字也是符合这个规则，当然下划线除外。

现在，用户名的开始和结束都不能是句点，服务器也是这样。还有你不能有两个连续的句点他们之间至少存在一个字符，好现在我们来看一下怎么为用户名写一个匹配模式：

^[\_a-zA-Z0-9-]+$

现在还不能允许句号的存在。我们把它加上：

^[\_a-zA-Z0-9-]+(\.[\_a-zA-Z0-9-]+)\*$

上面的意思就是说：以至少一个规范字符（除了.）开头，后面跟着0个或者多个以点开始的字符串。

简单化一点， 我们可以用eregi()取代ereg()、eregi()对大小写不敏感， 我们就不需要指定两个范围 “a-z” 和 “A-Z”只需要指定一个就可以了：

^[\_a-z0-9-]+(\.[\_a-z0-9-]+)\*$

后面的服务器名字也是一样，但要去掉下划线：

^[a-z0-9-]+(\.[a-z0-9-]+)\*$

好。现在只需要用”@”把两部分连接：

^[\_a-z0-9-]+(\.[\_a-z0-9-]+)\*@[a-z0-9-]+(\.[a-z0-9-]+)\*$

这就是完整的email认证匹配模式了，只需要调用：

eregi(”^[\_a-z0-9-]+(\.[\_a-z0-9-]+)\*@[a-z0-9-]+(\.[a-z0-9-]+)\*$”,$eamil)

就可以得到是否为email了

正则表达式的其他用法

提取字符串

ereg() and eregi() 有一个特性是允许用户通过正则表达式去提取字符串的一部分(具体用法你可以阅读手册)。比如说，我们想从 path/URL 提取文件名，下面的代码就是你需要：

ereg(”([^\\/]\*)$”, $pathOrUrl, $regs);  
echo $regs[1];

高级的代换

ereg\_replace() 和 eregi\_replace()也是非常有用的，假如我们想把所有的间隔负号都替换成逗号：

ereg\_replace(”[ \n\r\t]+”, “,”, trim($str));

最后，我把另一串检查EMAIL的正则表达式让看文章的你来分析一下：

“^[-!#$%&\’\*+\\./0-9=?A-Z^\_`a-z{|}~]+’.’@’.’[-!#$%&\’\*+\\/0-9=?A-Z^\_`a-z{|}~]+\.’.’[-!#$%&\’\*+\\./0-9=?A-Z^\_`a-z{|}~]+$”

如果能方便的读懂，那这篇文章的目的就达到了。