**正则表达式**

## 正则表达式（字符串处理神器）

正则表达式，又称规则表达式。（英语：Regular Expression，在代码中常简写为regex、regexp或RE），计算机科学的一个概念。**正则表达式通常被用来检索、替换那些符合某个模式(规则)的文本。**许多程序设计语言都支持利用正则表达式进行字符串操作。例如，在[Perl](https://baike.baidu.com/item/Perl" \t "https://baike.baidu.com/item/%E6%AD%A3%E5%88%99%E8%A1%A8%E8%BE%BE%E5%BC%8F/_blank)中就内建了一个功能强大的正则表达式引擎。正则表达式这个概念最初是由[Unix](https://baike.baidu.com/item/Unix" \t "https://baike.baidu.com/item/%E6%AD%A3%E5%88%99%E8%A1%A8%E8%BE%BE%E5%BC%8F/_blank)中的工具软件（例如sed和[grep](https://baike.baidu.com/item/grep/5997841" \t "https://baike.baidu.com/item/%E6%AD%A3%E5%88%99%E8%A1%A8%E8%BE%BE%E5%BC%8F/_blank)）普及开的。正则表达式通常缩写成“regex”，[单数](https://baike.baidu.com/item/%E5%8D%95%E6%95%B0/1658633" \t "https://baike.baidu.com/item/%E6%AD%A3%E5%88%99%E8%A1%A8%E8%BE%BE%E5%BC%8F/_blank)有regexp、regex，[复数](https://baike.baidu.com/item/%E5%A4%8D%E6%95%B0/254365" \t "https://baike.baidu.com/item/%E6%AD%A3%E5%88%99%E8%A1%A8%E8%BE%BE%E5%BC%8F/_blank)有regexps、regexes、regexen。

**1 正则表达式用途**

1). 字符串匹配

2). 字符串查找

3). 字符串替换

**2 基础知识**

|  |  |
| --- | --- |
| **元字符** | **描述** |
| **\** | 将下一个字符标记符、或一个向后引用、或一个八进制转义符。例如，“\\n”匹配\n。“\n”匹配换行符。序列“\\”匹配“\”而“\(”则匹配“(”。即相当于多种编程语言中都有的“转义字符”的概念。 |
| **^** | 匹配输入字行首。如果设置了RegExp对象的Multiline属性，^也匹配“\n”或“\r”之后的位置。 |
| **$** | 匹配输入行尾。如果设置了RegExp对象的Multiline属性，$也匹配“\n”或“\r”之前的位置。 |
| **\*** | 匹配前面的子表达式任意次。例如，zo\*能匹配“z”，也能匹配“zo”以及“zoo”。\*等价于{0,}。 |
| **+** | 匹配前面的子表达式一次或多次(大于等于1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”。?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为一组，后三个o为一组。“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地匹配所搜索的字符串，而默认的贪婪模式则尽可能多地匹配所搜索的字符串。例如，对于字符串“oooo”，“o+”将尽可能多地匹配“o”，得到结果[“oooo”]，而“o+?”将尽可能少地匹配“o”，得到结果 ['o', 'o', 'o', 'o'] |
| **.点** | 匹配除“\n”和"\r"之外的任何单个字符。要匹配包括“\n”和"\r"在内的任何字符，请使用像“[\s\S]”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)** | 非获取匹配，匹配pattern但不获取匹配结果，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时很有用。例如“industr(?:y|ies)”就是一个比“industry|industries”更简略的表达式。 |
| **(?=pattern)** | 非获取匹配，正向肯定预查，在任何匹配pattern的字符串开始处匹配查找字符串，该匹配不需要获取供以后使用。例如，“Windows(?=95|98|NT|2000)”能匹配“Windows2000”中的“Windows”，但不能匹配“Windows3.1”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)** | 非获取匹配，正向否定预查，在任何不匹配pattern的字符串开始处匹配查找字符串，该匹配不需要获取供以后使用。例如“Windows(?!95|98|NT|2000)”能匹配“Windows3.1”中的“Windows”，但不能匹配“Windows2000”中的“Windows”。 |
| **(?<=pattern)** | 非获取匹配，反向肯定预查，与正向肯定预查类似，只是方向相反。例如，“(?<=95|98|NT|2000)Windows”能匹配“2000Windows”中的“Windows”，但不能匹配“3.1Windows”中的“Windows”。  “(?<=95|98|NT|2000)Windows”目前在python3.6中re模块测试会报错，用“|”连接的字符串长度必须一样，这里“95|98|NT”的长度都是2，“2000”的长度是4，会报错。 |
| **(?<!pattern)** | 非获取匹配，反向否定预查，与正向否定预查类似，只是方向相反。例如“(?<!95|98|NT|2000)Windows”能匹配“3.1Windows”中的“Windows”，但不能匹配“2000Windows”中的“Windows”。这个地方不正确，有问题  此处用或任意一项都不能超过2位，如“(?<!95|98|NT|20)Windows正确，“(?<!95|980|NT|20)Windows 报错，若是单独使用则无限制，如(?<!2000)Windows 正确匹配。  同上，这里在python3.6中re模块中字符串长度要一致，并不是一定为2，比如“(?<!1995|1998|NTNT|2000)Windows”也是可以的。 |
| **x|y** | 匹配x或y。例如，“z|food”能匹配“z”或“food”(此处请谨慎)。“[zf]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“plin”。 |
| **[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。  注意:只有连字符在字符组内部时,并且出现在两个字符之间时,才能表示字符的范围; 如果出字符组的开头,则只能表示连字符本身. |
| **[^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（即正则表达式的“匹配”有两种概念，一种是匹配字符，一种是匹配位置，这里的\b就是匹配位置的）。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]。grep 要加上-P，perl正则支持 |
| **\D** | 匹配一个非数字字符。等价于[^0-9]。grep要加上-P，perl正则支持 |
| **\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** | 匹配任何不可见字符，包括空格、制表符、换页符等等。等价于[ \f\n\r\t\v]。 |
| **\S** | 匹配任何可见字符。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。类似但不等价于“[A-Za-z0-9\_]”，这里的"单词"字符使用Unicode字符集。 |
| **\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（0-7），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（&copy;）。 |
| **\p{P}** | 小写 p 是 property 的意思，表示 Unicode 属性，用于 Unicode 正表达式的前缀。中括号内的“P”表示Unicode 字符集七个字符属性之一：标点字符。  其他六个属性：  L：字母；  M：标记符号（一般不会单独出现）；  Z：分隔符（比如空格、换行等）；  S：符号（比如数学符号、货币符号等）；  N：数字（比如阿拉伯数字、罗马数字等）；  C：其他字符。  \*注：此语法部分语言不支持，例：javascript。 |
| **\<**  **\>** | 匹配词（word）的开始（\<）和结束（\>）。例如正则表达式\<the\>能够匹配字符串"for the wise"中的"the"，但是不能匹配字符串"otherwise"中的"the"。注意：这个元字符不是所有的软件都支持的。 |
| **( )** | 将( 和 ) 之间的表达式定义为“组”（group），并且将匹配这个表达式的字符保存到一个临时区域（一个正则表达式中最多可以保存9个），它们可以用 \1 到\9 的符号来引用。 |
| **|** | 将两个匹配条件进行逻辑“或”（Or）运算。例如正则表达式(him|her) 匹配"it belongs to him"和"it belongs to her"，但是不能匹配"it belongs to them."。注意：这个元字符不是所有的软件都支持的。 |

## 二、在java中使用

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class Test {

/\*\*\*

\* 抓取网页的Email

\*/

public static void test5() {

try {

//先从网页保存一个页面，从读取文件内容

BufferedReader bufferedReader = new BufferedReader(new FileReader("D:\\桌面\\text\\html.html"));

String line = "";

while ((line = bufferedReader.readLine()) != null) {

catchEmail(line);

}

} catch (IOException e) {

e.printStackTrace();

}

}

private static void catchEmail(String content) {

String reg = "[\\w[.-]]+@[\\w[.-]]+\\.[\\w]+";

Pattern p = Pattern.compile(reg);

Matcher m = p.matcher(content);

while (m.find()) {

print(m.group());

}

}

/\*\*\*

\* 分组

\*/

public static void test4() {

//用()分组表达式，分组的组号规则，整个是第0组，然后从左往右依次遇到“)”就算一组，逐渐+1

Pattern p = Pattern.compile("(\\d{3,5})([a-z]{2})");//整个第0组，3到5位的数字1组，两个小写字母2组，这里共3组,

String s = "1212ss-453bb-223ccc-23fdfd-deefeffe324";

Matcher m = p.matcher(s);

while (m.find()) {

print(m.group(0));//匹配第0组

print(m.group(1));

print(m.group(2));

}

}

/\*\*\*

\* 字符串替换

\*/

public static void test3() {

Pattern pattern = Pattern.compile("java", Pattern.CASE\_INSENSITIVE);//忽略大小写

Matcher matcher = pattern.matcher("Java java JAva JaVa I love Java youHate JAVA asasfd dda");

StringBuffer sb = new StringBuffer();

int i = 0;

while (matcher.find()) {//把匹配到的字符串，第技术

i++;

if (i % 2 == 0) {

matcher.appendReplacement(sb, "JAVA");

} else {

matcher.appendReplacement(sb, "java");

}

print(matcher.group());

}

matcher.appendTail(sb);//把尾部没有匹配的字符串加上

print(sb);

//print(matcher.replaceAll("JAVA"));//匹配到的全部替换为答谢的java

}

/\*\*\*

\* 得到匹配的字符串

\*/

public static void test2() {

Pattern pattern = Pattern.compile("\\d{3,5}");

String s = "123-453456-3435-0";

Matcher matcher = pattern.matcher(s);

print(matcher.matches());//否有有匹配的

matcher.reset();//执行matches()方法之后再需要执行reset()方法，find()方法能正确匹配整个字符串

while (matcher.find()) {

print(matcher.group());

}

}

/\*\*\*

\* 匹配反斜杠和Email

\*/

public static void test1() {

print("\\".matches("\\\\"));//Java中匹配一个反斜杠,需要四个\\\\

print(" \n".matches("^[\\s&&[^\\n]]\*\\n$"));//空白行匹配:开头是个空白符，非换行符，任意多个或一个，结尾是换行符

print("1df@164.comm".matches("[\\w[.-]]+@[\\w[.-]]+\\.[\\w]+"));//匹配Email

}

public static void print(Object obj) {

System.out.println(obj);

}

}

## 三、常用的正则表达式

**1 简单常用汇总**

^\d+$　　//匹配非负整数（正整数 + 0）

^[0-9][1-9][0-9]$　　//匹配正整数

^((-\d+)|(0+))$　　//匹配非正整数（负整数 + 0）

^-[0-9][1-9][0-9]$　　//匹配负整数

^-?\d+$　　　　//匹配整数

^\d+(.\d+)?$　　//匹配非负浮点数（正浮点数 + 0）

^(([0-9]+.[0-9][1-9][0-9])|([0-9][1-9][0-9].[0-9]+)|([0-9][1-9][0-9]))$　　//匹配正浮点数

^((-\d+(.\d+)?)|(0+(.0+)?))$　　//匹配非正浮点数（负浮点数 + 0）

^(-(([0-9]+.[0-9][1-9][0-9])|([0-9][1-9][0-9].[0-9]+)|([0-9][1-9][0-9])))$　　//匹配负浮点数

^(-?\d+)(.\d+)?$　　//匹配浮点数

^[A-Za-z]+$　　//匹配由26个英文字母组成的字符串

^[A-Z]+$　　//匹配由26个英文字母的大写组成的字符串

^[a-z]+$　　//匹配由26个英文字母的小写组成的字符串

^[A-Za-z0-9]+$　　//匹配由数字和26个英文字母组成的字符串

^\w+$　　//匹配由数字、26个英文字母或者下划线组成的字符串

^[\w-]+(.[\w-]+)\*@[\w-]+(.[\w-]+)+$　　　　//匹配email地址

匹配中文字符的正则表达式： [\u4e00-\u9fa5]

匹配双字节字符(包括汉字在内)：[^\x00-\xff]

匹配空行的正则表达式：\n[\s| ]\*\r

匹配HTML标记的正则表达式：/<(.)>.<\/\1>|<(.\*) \/>/

匹配首尾空格的正则表达式：(^\s\*)|(\s\*$)

**2 正则表达式用例**

1、^\S+[a-z A-Z]$ 不能为空 不能有空格 只能是英文字母

2、\S{6,} 不能为空 六位以上

3、^\d+$ 不能有空格 只能是数字

4、(.\*)(.jpg|.bmp)$ 只能是jpg和bmp格式

5、^\d{4}-\d{1,2}-\d{1,2}$ 只能是2004-10-22格式

6、^0$ 至少选一项

7、^0{2,}$ 至少选两项

8、^[\s|\S]{20,}$ 不能为空 二十字以上

9、^+?a-z0-9\*@([a-z0-9]+(.|-))+[a-z]{2,6}$邮件

10、\w+([-+.]\w+)@\w+([-.]\w+).\w+([-.]\w+)([,;]\s\w+([-+.]\w+)@\w+([-.]\w+).\w+([-.]\w+)) 输入多个地址用逗号或空格分隔邮件

11、^(([0-9]+))?[0-9]{7,8}$电话号码7位或8位或前面有区号例如（022）87341628

12、^[a-z A-Z 0-9 ]+@[a-z A-Z 0-9 ]+(.[a-z A-Z 0-9 ]+)+(,[a-z A-Z 0-9 ]+@[a-z A-Z 0-9 ]+(.[a-z A-Z 0-9 ]+)+)\*$

只能是字母、数字、下划线；必须有@和.同时格式要规范 邮件

13 ^\w+@\w+(.\w+)+(,\w+@\w+(.\w+)+)\*$上面表达式也可以写成这样子，更精练。

14 ^\w+((-\w+)|(.\w+))\@\w+((.|-)\w+).\w+$ [/size]

3 1.验证用户名和密码：（"^[a-zA-Z]\w{5,15}$"）正确格式："[A-Z][a-z]\_[0-9]"组成,并且第一个字必须为字母6~16位；

2.验证电话号码：（"^(\d{3,4}-)\d{7,8}$"）正确格式：xxx/xxxx-xxxxxxx/xxxxxxxx；

3.验证手机号码："^1[3|4|5|7|8][0-9]{9}$"；

4.验证身份证号（15位）："\d{14}[[0-9],0-9xX]"，（18位）："\d{17}[[0-9],0-9xX]"；

5.验证Email地址：("^\w+([-+.]\w+)\*@\w+([-.]\w+)\*\.\w+([-.]\w+)\*$")；

6.只能输入由数字和26个英文字母组成的字符串：("^[A-Za-z0-9]+$")；

7.整数或者小数：^[0-9]+([.][0-9]+){0,1}$

8.只能输入数字："^[0-9]\*$"。

9.只能输入n位的数字："^\d{n}$"。

10.只能输入至少n位的数字："^\d{n,}$"。

11.只能输入m~n位的数字："^\d{m,n}$"。

12.只能输入零和非零开头的数字："^(0|[1-9][0-9]\*)$"。

13.只能输入有两位小数的正实数："^[0-9]+(\.[0-9]{2})?$"。

14.只能输入有1~3位小数的正实数："^[0-9]+(\.[0-9]{1,3})?$"。

15.只能输入非零的正整数："^\+?[1-9][0-9]\*$"。

16.只能输入非零的负整数："^\-[1-9][0-9]\*$"。

17.只能输入长度为3的字符："^.{3}$"。

18.只能输入由26个英文字母组成的字符串："^[A-Za-z]+$"。

19.只能输入由26个大写英文字母组成的字符串："^[A-Z]+$"。

20.只能输入由26个小写英文字母组成的字符串："^[a-z]+$"。

21.验证是否含有^%&',;=?$\"等字符："[%&',;=?$\\^]+"。

22.只能输入汉字："^[\u4e00-\u9fa5]{0,}$"。

23.验证URL："^http://([\w-]+\.)+[\w-]+(/[\w-./?%&=]\*)?$"。

24.验证一年的12个月："^(0?[1-9]|1[0-2])$"正确格式为："01"～"09"和"10"～"12"。

25.验证一个月的31天："^((0?[1-9])|((1|2)[0-9])|30|31)$"正确格式为；"01"～"09"、"10"～"29"和“30”~“31”。

26.获取日期正则表达式：\\d{4}[年|\-|\.]\d{\1-\12}[月|\-|\.]\d{\1-\31}日?

评注：可用来匹配大多数年月日信息。

27.匹配双[字节](https://baike.baidu.com/item/%E5%AD%97%E8%8A%82" \t "https://baike.baidu.com/item/%E6%AD%A3%E5%88%99%E8%A1%A8%E8%BE%BE%E5%BC%8F/_blank)[字符](https://baike.baidu.com/item/%E5%AD%97%E7%AC%A6" \t "https://baike.baidu.com/item/%E6%AD%A3%E5%88%99%E8%A1%A8%E8%BE%BE%E5%BC%8F/_blank)(包括汉字在内)：[^\x00-\xff]

评注：可以用来计算字符串的长度（一个双字节字符长度计2，ASCII字符计1）

28.匹配空白行的正则表达式：\n\s\*\r

评注：可以用来删除空白行

29.匹配HTML标记的正则表达式：<(\S\*?)[^>]\*>.\*?</>|<.\*? />

评注：网上流传的版本太糟糕，上面这个也仅仅能匹配部分，对于复杂的嵌套标记依旧无能为力

30.匹配首尾空白[字符](https://baike.baidu.com/item/%E5%AD%97%E7%AC%A6" \t "https://baike.baidu.com/item/%E6%AD%A3%E5%88%99%E8%A1%A8%E8%BE%BE%E5%BC%8F/_blank)的正则表达式：^\s\*|\s\*$

评注：可以用来删除行首行尾的空白字符(包括空格、[制表符](https://baike.baidu.com/item/%E5%88%B6%E8%A1%A8%E7%AC%A6" \t "https://baike.baidu.com/item/%E6%AD%A3%E5%88%99%E8%A1%A8%E8%BE%BE%E5%BC%8F/_blank)、换页符等等)，非常有用的表达式

31.匹配网址URL的正则表达式：[a-zA-z]+://[^\s]\*

评注：网上流传的版本功能很有限，上面这个基本可以满足需求

32.匹配帐号是否合法(字母开头，允许5-16[字节](https://baike.baidu.com/item/%E5%AD%97%E8%8A%82" \t "https://baike.baidu.com/item/%E6%AD%A3%E5%88%99%E8%A1%A8%E8%BE%BE%E5%BC%8F/_blank)，允许字母数字下划线)：^[a-zA-Z][a-zA-Z0-9\_]{4,15}$

评注：表单验证时很实用

33.匹配腾讯QQ号：[1-9][0-9]{4,}

评注：腾讯QQ号从10 000 开始

34.匹配中国邮政编码：[1-9]\\d{5}(?!\d)

评注：中国邮政编码为6位数字

35.匹配ip地址：([1-9]{1,3}\.){3}[1-9]。

评注：提取ip地址时有用

**3 复杂常用汇总**

只能输入数字："^[0-9]\*$"。

只能输入n位的数字："^\d{n}$"。

只能输入至少n位的数字："^\d{n,}$"。

只能输入m~n位的数字：。"^\d{m,n}$"

只能输入零和非零开头的数字："^(0|[1-9][0-9]\*)$"。

只能输入有两位小数的正实数："^[0-9]+(.[0-9]{2})?$"。

只能输入有1~3位小数的正实数："^[0-9]+(.[0-9]{1,3})?$"。

只能输入非零的正整数："^\+?[1-9][0-9]\*$"。

只能输入非零的负整数："^\-[1-9][]0-9"\*$。

只能输入长度为3的字符："^.{3}$"。

只能输入由26个英文字母组成的字符串："^[A-Za-z]+$"。

只能输入由26个大写英文字母组成的字符串："^[A-Z]+$"。

只能输入由26个小写英文字母组成的字符串："^[a-z]+$"。

只能输入由数字和26个英文字母组成的字符串："^[A-Za-z0-9]+$"。

只能输入由数字、26个英文字母或者下划线组成的字符串："^\w+$"。

验证用户密码："^[a-zA-Z]\w{5,17}$"正确格式为：以字母开头，长度在6~18之间，只能包含字符、数字和下划线。

验证是否含有^%&',;=?$\"等字符："[^%&',;=?$\x22]+"。

只能输入汉字："^[\u4e00-\u9fa5]{0,}$"

验证Email地址："^\w+([-+.]\w+)\*@\w+([-.]\w+)\*\.\w+([-.]\w+)\*$"。

验证InternetURL："^http://%28[/\w-]+\.)+[\w-]+(/[\w-./?%&=]\*)?$"。

验证电话号码："^(\(\d{3,4}-)|\d{3.4}-)?\d{7,8}$"正确格式为："XXX-XXXXXXX"、"XXXX-XXXXXXXX"、"XXX-XXXXXXX"、"XXX-XXXXXXXX"、"XXXXXXX"和"XXXXXXXX"。

验证身份证号（15位或18位数字）："^\d{15}|\d{18}$"。

验证一年的12个月："^(0?[1-9]|1[0-2])$"正确格式为："01"～"09"和"1"～"12"。

验证一个月的31天："^((0?[1-9])|((1|2)[0-9])|30|31)$"正确格式为；"01"～"09"和"1"～"31"。

利用正则表达式限制网页表单里的文本框输入内容：

用正则表达式限制只能输入中文：onkeyup="value=value.replace(/[^\u4E00-\u9FA5]/g,'')" onbeforepaste="clipboardData.setData('text',clipboardData.getData('text').replace(/[^\u4E00-\u9FA5]/g,''))"

用正则表达式限制只能输入全角字符： onkeyup="value=value.replace(/[^\uFF00-\uFFFF]/g,'')" onbeforepaste="clipboardData.setData('text',clipboardData.getData('text').replace(/[^\uFF00-\uFFFF]/g,''))"

用正则表达式限制只能输入数字：onkeyup="value=value.replace(/[^\d]/g,'') "onbeforepaste="clipboardData.setData('text',clipboardData.getData('text').replace(/[^\d]/g,''))"

用正则表达式限制只能输入数字和英文：onkeyup="value=value.replace(/[\W]/g,'') "onbeforepaste="clipboardData.setData('text',clipboardData.getData('text').replace(/[^\d]/g,''))"

得用正则表达式从URL地址中提取文件名的javascript程序，如下结果为page1

## 四、Js正则表达式

**1 基本知识**

search（‘str’）搜索位置，当没有找到的时候，返回值为-1.

substring（2,5）获得第二到第五个位置的字符串（不包括结束位置五的字符！）

substring（2）从第二个位置开始往后取。

charAt（）获取某个位置的字符串。

split（‘-’）切分，将字符串以‘-’符号切分成数组。

match( )将所有符合要求的提取出来。

**2 正则表达式有两种表示方式**

（1）js风格——var re =new RegExp(‘a’，‘i’)

（2）perl风格——/a/i

元字符（方括号）：

[abc]pc——（abc,bpc,cpc）

[a-z],[0-9]都是范围

[^a-z]除了字母

[^a-z0-9]除了字母和数字

**3 正则表达式常用的转义字符**

i 为忽略大小写 g 为找到全部

.(点)——任意字符，但是一般很少使用，杀伤力太大。

\d——数字

\w——英文，数字和下划线

\s——空白字符

\D——[^1-9]非数字

\W——[^a-z0-9]非英文、数字或下划线

\d——代表d字母。

**4 量词**

{n}：正好出现n次。比如\d{8},则为出现8次数字，[1-9]\d{7}则为第一位为1-9，后面七位为数字。

{n，m}最少出现n次，最多出现m次。

{n，}最少n次，最多不限。

？={0,1} 可有可无。例如区号，(0\d{2,3}-)?[1-9]\d{7}

**var str = “13 ，24 ，5678”**

**str.match(re)**

re=/\d/g 结果为1，3，2，4，5，6，7，8因为只找了一位数字。

re=/\d\d/g结果为13,24，56,78，是找了两位数字

re=/\d+/g结果为13 ，24 ，5678，加号 代表若干，即多少都可以。

**5 replace方法可以用来进行替换**

**str.replace(/a/g,”\*”)** 即所有的a都替换成星号。

**6 过滤HTML标签**

re=/[^<>]+/g 意思为 在尖括号内除了<>之外。

test方法校验：

re.test(oTxt.value)

在test方法中，只要有一部分而不是全部符合 ，其返回值就为true。

所以解决办法就是在正则表达式首尾加上^和，例如：邮箱校验：re=\w+@[a−z0−9]+.[a−z]