正则表达式数字:精通 JavaScript中的正则表达式手机整理 推荐

疯狂代码 http://CrazyCoder.cn/ ĵ:http:/CrazyCoder.cn/DeveloperUtil/Article77373.html

正则表达式可以:

•测试串某个模式例如可以对个输入串进行测试看在该串是否存在个电话号码模式或个信用卡号码模式这称为数据有效性验证

•替换文本可以在文档中使用个正则表达式来标识特定文字然后可以全部将其删除或者替换为别文字

•根据模式匹配从串中提取个子串可以用来在文本或输入字段中查找特定文字

正则表达式语法

个正则表达式就是由普通(例如 a 到 z)以及特殊(称为元)组成文字模式该模式描述在查找文字主体时待匹配个或多个串正则表达式作为个模板将某个模式和所搜索串进行匹配

创建正则表达式

Js代码

复制代码 代码如下:

```
var re =  RegExp;//RegExp是个对象,和A.gif' />样
//但这样没有任何效果,需要将正则表达式内容作为串传递进去
re = RegExp("a");//最简单正则表达式,将匹配字母a
re= RegExp("a","i");//第 2个参数,表示匹配时不分大小写
```

RegExp构造第个参数为正则表达式文本内容,而第个参数则为可选项标志.标志可以组合使用

•g (全文查找)

•i (忽略大小写)

•m (多行查找)

Js代码

复制代码 代码如下:

```
var re =  RegExp("a","gi");//匹配所有a或A
```

正则表达式还有另种正则表达式字面量声明方式

Js代码

复制代码 代码如下:

```
var re = /a/gi;
```

和正则表达式相关思路方法和属性
正则表达式对象思路方法
•test,返回个 Boolean 值它指出在被查找串中是否存在模式如果存在则返回 true否则就返回 false
•exec,用正则表达式模式在串中运行查找并返回包<script type="text/javascript"
src="http://www.javaeye.com/javascripts/tinymce/themes/advanced/langs/zh.js"></script><script
type="text/javascript"
src="http://www.javaeye.com/javascripts/tinymce/plugins/javaeye/langs/zh.js"></script>含该查找结果个
•compile,把正则表达式编译为内部格式从而执行得更快
正则表达式对象属性
•source,返回正则表达式模式文本复本只读
•lastIndex,返回位置它是被查找串中下次成功匹配开始位置
•$1...$9,返回 9个在模式匹配期间找到、最近保存部分只读
•input ($_),返回执行规范标准表述查找串只读
•lastMatch ($&),返回任何正则表达式搜索过程中最后匹配只读
•lastParen ($+),如果有话返回任何正则表达式查找过程中最后括子匹配只读
•leftContext ($`),返回被查找串中从串开始位置到最后匹配的前位置的间只读
•rightContext ($'),返回被搜索串中从最后个匹配位置开始到串结尾的间只读
String对象些和正则表达式相关思路方法
•match,找到个或多个正则表达式匹配
•replace,替换和正则表达式匹配子串
•search,检索和正则表达式相匹配值
•split,把串分割为串
测试正则表达式是如何工作!
复制代码 代码如下:

```
//test思路方法,测试串,符合模式时返回true,否则返回false
var re = /he/;//最简单正则表达式,将匹配he这个单词
var str = "he";
alert(re.test(str));//true
str = "we";
alert(re.test(str));//false
str = "HE";
alert(re.test(str));//false,大写,如果要大小写都匹配可以指定i标志(i是ignoreCase或-insensitive表示)
```

```
re = /he/i;
alert(re.test(str));//true
str = "Certainly!He loves her!";
alert(re.test(str));//true,只要包含he(HE)就符合,如果要只是he或HE,不能有其它,则可使用^和$
re = /^he/i;//脱(^)代表开始位置
alert(re.test(str));//false,he不在str最开始
str = "He is a good boy!";
alert(re.test(str));//true,He是开始位置,还需要使用$
re = /^he$/i;//$表示结束位置
alert(re.test(str));//false
str = "He";
alert(re.test(str));//true
//当然,这样不能发现正则表达式有多强大,我们完全可以在上面例子中使用或indexOf
re = /\s/;// \s匹配任何空白包括空格、制表符、换页符等等
str= "user Name";//用户名包含空格
alert(re.test(str));//true
str = "user Name";//用户名包含制表符
alert(re.test(str));//true
re=/^[a-z]/i;//匹配指定范围内任意,这里将匹配英文字母,不区分大小写
str="variableName";//变量名必须以字母开头
alert(re.test(str));//true
str="123abc";
alert(re.test(str));//false
```

当然,仅仅知道了串是否匹配模式还不够,我们还需要知道哪些匹配了模式
复制代码 代码如下:

```
var osVersion = "Ubuntu 8";//其中8表示系统主版本号
var re = /^[a-z]+\s+\d+$/i; //+号表示至少要出现1次,\s表示空白,\d表示个数字
alert(re.test(osVersion));//true,但我们想知道主版本号
//另个思路方法exec,返回个,第个元素为完整匹配内容
re=/^[a-z]+\s+\d+$/i;
arr = re.exec(osVersion);
alert(arr[0]);//将osVersion完整输出,整个串刚好匹配re
//我只需要取出数字
```

```
re=/\d+/;
var arr = re.exec(osVersion);
alert(arr[0]);//8
```

更复杂使用方法,使用子匹配
复制代码 代码如下:

```
//exec返回第1到n元素中包含是匹配中出现任意个子匹配
re=/^[a-z]+\s+(\d+)$/i;//用来创建子匹配
arr =re.exec(osVersion);
alert(arr[0]);//整个osVersion,也就是正则表达式完整匹配
alert(arr[1]);//8,第个子匹配,事实也可以这样取出主版本号
alert(arr.length);//2
osVersion = "Ubuntu 8.10";//取出主版本号和次版本号
re = /^[a-z]+\s+(\d+)\.(\d+)$/i;//.是正则表达式元的,若要用它字面意义须转义
arr = re.exec(osVersion);
alert(arr[0]);//完整osVersion
alert(arr[1]);//8
alert(arr[2]);//10
```

注意,当串不匹配re时,exec思路方法将返回null
String对象些和正则表达式有关思路方法
点击运行可以看到效果: //replace思路方法,用于替换串 var str ="some money";
alert(str.replace("some","much"));//much money //replace第个参数可以为正则表达式 var re = /\s/;//空白 alert(str.replace(re,"%"));//some%money //在不知道串中有多少空白时,正则表达式极为方便 str ="some some \tsome\t\f"; re = /\s+/; alert(str.replace(re,"#"));//但这样只会将第次出现堆空白替换掉 //个正则表达式只能进行次匹配,\s+匹配了第个空格后就退出了 re = /\s+/g;//g,全局标志,将使正则表达式匹配整个串 alert(str.replace(re,"@"));//some@some@some@ //另个和的相似是split var str = "a-bd-c"; var arr = str.split("-");//返回["a","bd","c"] //如果str是用户输入,他可能输入a-bd-c也可能输入a bd c或a_bd_c,但不会是abdc(这样就说他输错了) str = "a_db-c";//用户以他喜欢方式加分隔符s re=/[^a-z]/i;//前面我们说^表示开始,但在里它表示个负集 //匹配任何不在指定范围内任意,这里将匹配除字母处所有 arr = str.split(re);//仍返回["a","bd","c"]; //在串中查找时我们常用indexOf,和的对应用于正则查找思路方法是search str = "My age is 18.Golden age!";//年龄不是定,我们用indexOf不能查找它位置 re = /\d+/; alert(str.search(re));//返回查找到串开始下标10 //注意,查找本身就是出现第次就立即返回,所以无需在

search时使用g标志 //下面代码虽然不出错,但g标志是多余 re=/\d+/g; alert(str.search(re));//仍然是10
[Ctrl+A 全选 提示:你可先修改部分代码,再按运行]

注意,当search思路方法没有找到匹配时,将返回-1
类似于exec思路方法,String对象match思路方法也用于将串和正则表达式进行匹配并返回结果
复制代码 代码如下:

```
var str = "My name is CJ.Hello everyone!";
var re = /[A-Z]/;//匹配所有大写字母
var arr = str.match(re);//返回
alert(arr);//中只会包含个M,我们没有使用全局匹配
re = /[A-Z]/g;
arr = str.match(re);
alert(arr);//M,C,J,H
//从串中抽取单词
re = /\b[a-z]\b/i;//\b表示单词边界
str = "one two three four";
alert(str.match(re));//one,two,three,four
```

RegExp对象例子些属性
Js代码
复制代码 代码如下:

```
var re = /[a-z]/i;
alert(re.source);//将[a-z]串输出
//请注意,直接alert(re)会将正则表达式连同前向斜线和标志输出,这是re.toString思路方法定义
```

每个RegExp对象例子具有lastIndex属性,它是被查找串中下次成功匹配开始位置,默认值是-1 lastIndex 属性被 RegExp 对象 exec 和 test 思路方法修改.并且它是可写.
复制代码 代码如下:

```
var re = /[A-Z]/;
//exec思路方法执行后,修改了relastIndex属性,
var str = "Hello,World!!!";
```

```
var arr = re.exec(str);
alert(re.lastIndex);//0,没有设置全局标志
re = /[A-Z]/g;
arr = re.exec(str);
alert(re.lastIndex);//1
arr = re.exec(str);
alert(re.lastIndex);//7
```

当匹配失败(后面没有匹配)或lastIndex值大于串长度时再执行exec等思路方法会将lastIndex设为0(开始位置)
复制代码 代码如下:

```
var re = /[A-Z]/;
var str = "Hello,World!!!";
re.lastIndex = 120;
var arr = re.exec(str);
alert(re.lastIndex);//0
```

RegExp对象静态属性
复制代码 代码如下:

```
//input 最后用于匹配串(传递给test,exec思路方法串)
var re = /[A-Z]/;
var str = "Hello,World!!!";
var arr = re.exec(str);
alert(RegExp.input);//Hello,World!!!
re.exec("tempstr");
alert(RegExp.input);//仍然是Hello,World!!!,tempstr不匹配
//lastMatch 最后匹配
re = /[a-z]/g;
str = "hi";
re.test(str);
alert(RegExp.lastMatch);//h
re.test(str);
alert(RegExp["$&"]);//i ,$&是lastMatch短名字但由于它不是合法变量名所以要
```

```
//lastParen 最后匹配分组
re = /[a-z](\d+)/gi;
str = "Class1 Class2 Class3";
re.test(str);
alert(RegExp.lastParen);//1
re.test(str);
alert(RegExp["$+"]);//2
//leftContext 返回被查找串中从串开始位置到最后匹配的前位置的间
//rigthContext 返回被搜索串中从最后个匹配位置开始到串结尾的间
re = /[A-Z]/g;
str = "123ABC456";
re.test(str);
alert(RegExp.leftContext);//123
alert(RegExp.rightContext);//BC456
re.test(str);
alert(RegExp["$`"]);//123A
alert(RegExp["$'"]);//C456
```

multiline属性返回正则表达式是否使用多行模式,这个属性不针对某个正则表达式例子而是针对所有正则表达式并且这个属性可写.(IE和Opera不支持这个属性)
复制代码 代码如下:

```
alert(RegExp.multiline);
//IEOpera不支持这个属性所以最好还是单独指定
var re = /\w+/m;
alert(re.multiline);
alert(RegExp["$*"]);//RegExp对象静态属性不会给RegExp某个对象例子指定了m标志而改变
RegExp.multiline = true;//这将打开所有正则表达式例子多行匹配模式
alert(RegExp.multiline);
```

使用元注意事项:元是正则表达式部分当我们要匹配正则表达式本身时必须对这些元转义.下面是正则表达式用到所有元
([{ \ ^ $ |) ? * + .
复制代码 代码如下:

```
var str = "?";
var re = /?/;
alert(re.test(str));//出错？是元必须转义
re = /\?/;
alert(re.test(str));//true
```

使用RegExp构造和使用正则表达式字面量创建正则表达式注意点
复制代码 代码如下:

```
var str = "\?";
alert(str);//只会输出?
var re = /\?/;//将匹配?
alert(re.test(str));//true
re =  RegExp("\?");//出错,这相当于re = /\?/
re =  RegExp("\\?");//正确将匹配？
alert(re.test(str));//true
```

既然双重转义这么不友好所以还是用正则表达式字面量声明方式
如何在正则表达式中使用特殊？
复制代码 代码如下:

```
//ASCII方式用十 6进制数来表示特殊
var re = /^\x43\x4A$/;//将匹配CJ
alert(re.test("CJ"));//true
//也可使用 8进制方式
re = /^\103\112$/;//将匹配CJ
alert(re.test("CJ"));//true
//还可以使用Unicode编码
re =/^\u0043\u004A$/;//使用 Unicode必须使用u开头接着是编码 4位16进制表现形式
alert(re.test("CJ"));
```

另处还有些其它预定义特殊如下表所示:

描述
\n 换行符
\r 回车符
\t 制表符
\f 换页符(Tab)
\cX 和X对应控制
\b 退格符(BackSpace)
\v 垂直制表符
\0 空("")
类 ---〉简单类反向类范围类组合类预定义类
复制代码 代码如下:

```
//简单类
var re = /[abc123]/;//将匹配abc123这6个中个
//负向类
re = /[^abc]/;//将匹配除abc的外个
//范围类
re = /[a-b]/;//将匹配小写a-b 26个字母
re = /[^0-9]/;//将匹配除0-9 10个的处个
//组合类
re = /[a-b0-9A-Z_]/;//将匹配字母数字和下划线
```

下面是正则表达式中预定义类
代码 等同于 匹配
. IE下[^\n]其它[^\n\r] 匹配除换行符的外任何个
\d [0-9] 匹配数字
\D [^0-9] 匹配非数字
\s [\n\r\t\f\x0B] 匹配个空白
\S [^ \n\r\t\f\x0B] 匹配个非空白
\w [a-zA-Z0-9_] 匹配字母数字和下划线
\W [^a-zA-Z0-9_] 匹配除字母数字下划线的外
量词(下表量词单个出现时皆是贪婪量词)
代码 描述
* 匹配前面子表达式零次或多次例如zo* 能匹配 "z" 以及 "zoo" * 等价于{0,}
+ 匹配前面子表达式次或多次例如'zo+' 能匹配 "zo" 以及 "zoo"但不能匹配 "z"+ 等价于 {1,}

? 匹配前面子表达式零次或次例如"do(es)?" 可以匹配 "do" 或 "does" 中"do" ? 等价于 {0,1}
{n} n 是个非负整数匹配确定 n 次例如'o{2}' 不能匹配 "Bob" 中 'o'但是能匹配 "food" 中两个 o
{n,} n 是个非负整数至少匹配n 次例如'o{2,}' 不能匹配 "Bob" 中 'o'但能匹配 "foooood" 中所有 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 次刘 "o{1,3}" 将匹配 "fooooood" 中前3个 o'o{0,1}' 等价于 'o?'请注意在逗号和两个数的间不能有空格
贪婪量词和惰性量词
•用贪婪量词进行匹配时它首先会将整会串当成个匹配如果匹配话就退出如果不匹配就截去最后个进行匹配如果不匹配继续将最后个截去进行匹配直到有匹配为止直到现在我们遇到量词都是贪婪量词
•用贪婪量词进行匹配时它首先将第个当成个匹配如果成功则退出如果失败则测试前两个依些增加直到遇到合适匹配为止
惰性量词仅仅在贪婪量词后面加个"?"而已,如"a+"是贪婪匹配,"a+?"则是惰性
复制代码 代码如下:

```
var str = "abc";
var re = /\w+/;//将匹配abc
re = /\w+?/;//将匹配a
```

多行模式
复制代码 代码如下:

```
var re = /[a-z]$/;
var str = "ab\ncdef";
alert(str.replace(re,"#"));//ab\ncde#
re =/[a-z]$/m;
alert(str.replace(re,"#"));//a#\ncde#
```

分组和非捕获性分组
复制代码 代码如下:

```
re = /abc{2}/;//将匹配abcc
re = /(abc){2}/;//将匹配abcabc
//上面分组都是捕获性分组
str = "abcabc ###";
```

```
arr = re.exec(str);
alert(arr[1]);//abc
//非捕获性分组 (?:)
re = /(?:abc){2}/;
arr = re.exec(str);
alert(arr[1]);//und
```

候选(也就是所说“或”)
复制代码 代码如下:

```
re = /^a|bc$/;//将匹配开始位置a或结束位置bc
str ="add";
alert(re.test(str));//true
re = /^(a|bc)$/;//将匹配a或bc
str ="bc";
alert(re.test(str));//true
```

当包含分组正则表达式进行过test,match,search这些思路方法的后每个分组都被放在个特殊地方以备将来使用
这些存储是分组中特殊值我们称的为反向引用
复制代码 代码如下:

```
var re = /(A?(B?(C?)))/;
/*上面正则表达式将依次产生 3个分组
(A?(B?(C?))) 最外面
(B?(C?))
(C?)*/
str = "ABC";
re.test(str);//反向引用被存储在RegExp对象静态属性$1—$9中
alert(RegExp.$1+"\n"+RegExp.$2+"\n"+RegExp.$3);
//反向引用也可以在正则表达式中使用\1 ,\2...这类形式使用
re = /\d+(\D)\d+\1\d+/;
str = "2008-1-1";
alert(re.test(str));//true
str = "2008-4_3";
```

```
alert(re.test(str));//false
```

使用反向引用可以要求串中某几个位置上必须相同.另外在replace这类思路方法中可用特殊序列来表示反向引用
Js代码
复制代码 代码如下:

```
re = /(\d)\s(\d)/;
str = "1234 5678";
alert(str.replace(re,"$2 $1"));//在这个里面$1表示第个分组1234,$2则表示5678
```

其它——〉正向前瞻,用来捕获出现在特定的前,只有当后面跟着某个特定才去捕获它和正向前瞻对应有负向前瞻它用匹配只有当后面不跟着某个特定时才去匹配它在执行前瞻和负向前瞻的类运算时正则表达式引擎会留意串后面部分然而却不移动index
复制代码 代码如下:

```
//正向前瞻
re = /([a-z]+(?=\d))/i;
//我们要匹配后面跟个数字单词然后将单词返回而不要返回数字
str = "abc every1 abc";
alert(re.test(str));//true
alert(RegExp.$1);//every
alert(re.lastIndex);//使用前瞻好处是,前瞻内容(?=\d)并不会当成次匹配下次匹配仍从它开始
//负向前瞻(?!)
re = /([a-z](?!\d))/;i
//将匹配后面不包含数字字母,并且不会返回(?!\d)中内容
str = "abc1 _disibledevent=>正则表达式电子邮箱地址有效性要求(我们姑且这样定义):用户名只能包含字母数字以及下划线最少位最多25位用户名后面紧跟@后面是域名域名名称要求只能包含字母数字和减号(-)并且不能以减号开头或结尾然后后面是域名后缀(可以有多个)域名后缀必须是点号连上2-4位英文字母
复制代码 代码如下:

var re = /^\w{1,15}(?:@(?!-))(?:(?:[a-z0-9-]*)(?:[a-z0-9](?!-))(?:\.(?!-)))+[a-z]{2,4}$/;
```

好像漏了些,比如,replace思路方法第 2个参数作为情况
恰巧上次在某君博客里看到个面试题,读取个文本文件中串,统计其中"9"出现次数,用JS写个也很简单(不包含打开文件代码,打开文件可以用ActiveXObject等浏览器提供对象弄)
复制代码 代码如下:

```
var str="ADF9DF9DF9",//那个文本文件中串;
re=/9/gi,//匹配9
counter=0;//计数器
str.replace(re,function  {
counter;//每出现次匹配,就被执行次,返回值用来替换原值
 "#";
});
//最后str 变成 ADF#DF#DF#"
```

至于传递给replace思路方法接收参数....楼下继续
你说那个问题:
$&是lastMatch短名字
还有,不知道为什么你正则表达式中没有对{}两个特殊转义
提到JS兼容性问题,这里要提是RegExp对象例子compile 思路方法
compile思路方法用来将正则表达式编译为内部格式以使其执行更快
Js代码
复制代码 代码如下:

```
var re =  RegExp;
re.compile("[0-9]\\n");//注意要对斜杠多转义次,compile思路方法返回值为re这个对象(编译后)
```

但经测试,这个思路方法在Safari及Chrome类似浏览器,compile思路方法始终返回und,不可用　　2009-11-30 19:26:24
　　疯狂代码 http://CrazyCoder.cn/