后盾人 人人做后盾

www.houdunren.com

正则表达式

后盾人 2011-2018

什么是正则表达式

一个用来描述或者匹配一系列符合某个语法的字符串的语言。

在很多文本编辑器或其他工具里，正则表达式通常被用来检索、替换或拆分那些符合某个模式的文本内容。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

应用场合

数据验证、文本替换、内容检索、过滤内容

可以理解为：执行字符串函数无法完成的特殊的匹配、拆分、替换功能

正则表达式

正则表达式的创建

正则表达式是一种验证规则，是以对象的形式存在的

1.通过构造函数创建

```
reg=new RegExp(“正则表达式”,”模式修正符”)
var reg = new RegExp("houdun");
var stat = reg.test("houdunwang");
alert(stat);
```

2.通过字面量方式创建

```
var reg = /houdun/i;
var stat = reg.test("houdunwang");
alert(stat);
```

通常将正则表达式字符串放在 /RegExp/ 中间//称为定界符

正则对象.test(str)

返回一个布尔值，它指出在被查找的字符串中是否存在符合正则规则要求的内容

test方法

正则对象.exec() 用正则去匹配字符串，成功返回数组，失败返回假

返回的数组包含特殊属性：

input -被匹配的字符串

index -子串位置

如果采用g修饰符

如果设置了g，那么exec执行之后会更新正则表达式的lastIndex属性，表示本次匹配后，所匹配字符串的下一个字符的索引，下一次再用这个正则表达式匹配字符串的时候就会从上次的lastIndex属性开始匹配。

exec方法

原子是正则表达式中的最小的元素，包括英文、标点符号等

\d 匹配任意一个数字 [0-9]

\D 与除了数字以外的任何一个字符匹配 [^0-9]

\w 与任意一个英文字母,数字或下划线匹配 [a-z0-9A-Z_]

\W 除了字母,数字或下划线外与任何一个字符匹配 [^a-z0-9A-Z_]

\s 与任意一个空白字符匹配 [\n\f\r\t\v]

\S 与除了空白符外任意一个字符匹配 [^\n\f\r\t\v]

原子

在正则表达式中有一些特殊字符代表特殊意义叫元字符。

. 除换行符以外的任何一个字符

| 或的意思，匹配其中一项就代表匹配

元字符

[ ]　　只匹配其中的一个原子

[^]　　只匹配"除了"其中字符的任意一个原子

[0-9]　匹配0-9任何一个数字

[a-z]　匹配小写a-z任何一个字母

[A-Z]　匹配大写A-Z任何一个字母

原子表

可以使用一些元字符，重复表示一些原子或元字符

*	重复零次或更多次
+	重复一次或更多次
?	重复零次或一次
{n}	重复n次
{n,}	重复n次或更多次
{n,m}	重复n到m次

量词

正则匹配是贪婪的，禁止贪婪用

*?	重复任意次，但尽可能少重复
+?	重复1次或更多次，但尽可能少重复
??	重复0次或1次，但尽可能少重复
{n,m}?	重复n到m次，但尽可能少重复
{n,}?	重复n次以上，但尽可能少重复

贪婪和吝啬

字符边界

^　　匹配字符串的开始

$　　匹配字符串的结束，忽略换行符

边界匹配

i	不区分大小写字母的匹配
m	将字符串视为多行，修饰^与$
g	全局匹配，找到所有匹配项

模式修正符

用小括号可以将正则规则中的部分内容进行分组，分组后可以用\1\2这种形式进行调用

```
var reg = /(\d{3})b\1/;
var result1 = reg.test('a123b123c');
var result2 = reg.test('a123b456c');
console.log(result1);//true
console.log(result2);//false
```

分组

非捕获性分组工作模式下分组(?:)会作为匹配校验，并出现在匹配结果字符里面，但不作为子匹配返回。

```
var str = '000aaa111';
var reg = /(?:[a-z]+)(?:\d+)/;
var result = reg.exec(str);
console.log(result);
```

非捕获性分组

前瞻分为正向前瞻和反(负)向前瞻，正向前瞻(?=表达式)表示后面要有什么，反向前瞻(?!=表达式)表示后面不能有什么。

前瞻分组会作为匹配校验，但不出现在匹配结果字符里面，而且不作为子匹配返回。

前瞻 = 先行断言

(?=) 正向前瞻 = 正向零宽先行断言

(?!) 反向前瞻 = 负向前瞻 = 负向零宽先行断言

前瞻

```
//正向前瞻，匹配.jpg后缀文件名
var str = '123.jpg,456.gif,houdun.jpg';
var partern = /\w+(?=\.jpg)/g; //正向前瞻匹配
 //['123', 'abc']  返回结果正确，没有匹配456.gif
console.log(str.match(partern));

//反向前瞻，匹配3个及以上的a，而且后面不能有000的字符
var str = 'aaa000 aaaa111 aaaaaaa222';
var partern = /a{3,}(?!000)/g; //反向前瞻匹配
//['aaaa', 'aaaaaaa']  返回结果正确，没有匹配aaa000
console.log(str.match(partern));
```

前瞻

- **字符串对象.replace（正则或字符串,替换新内容）**
- 支持全局g修饰符，如果模式不是全局，当匹配到一个以后将不会继续匹配，反之则会继续往下匹配。

函数

字符串对象.split() 方法

拆分字符串，参数可以为字符串或正则表达式

函数

1.写一个去除字符串两边空格的函数

2.写一个去除字符串当中所有空格的函数

小练习