DAY15 正则

2023年5月11日 17:07

正则的概述以及创建

一.正则表达式概述

由我们自己来书写“规则”，专门用来检测 字符串 是否符合“规则”使用的

字符串规则对象

可以对字符串进行检索，替换，匹配

我们使用一些特殊的字符或者符号定义一个“规则公式”，然后用我们定义好的“规则公式”去检测字符串是不是合格

二.创建正则表达式

1.构造函数创建

let reg = new RegExp(正则规则,修饰符);// reg:实例对象，正则对象 RegExp:构造函数

参1：正则规则，可以是字符串或者变量

参2：修饰符(i:忽略大小写 g:全局匹配 m:多行匹配)，修饰符可以组合使用的,可选的

```
// let reg = new RegExp('hello', 'i');
```

2.字面量创建

let reg = /hello/i; //等同于new RegExp('hello', 'i');

注意：选择字面量创建正则，规则一定是字符串格式(无需添加引号)，这里的规则不允许使用变量的。

正则对象的方法

1.test

test是用来检测字符串是否符合我们正则的标准,查看字符串中是否存在正则规则匹配的字符串。

语法： 正则.test(字符串)

返回值： boolean

```
// let reg = /hello/i;
// console.log(reg.test('abchelloabc'));//true
// console.log(reg.test('abcheelloabc'));//false
// console.log(reg.test('abchElloabc'));//true
// console.log(reg.test('healloheblloheclo'));//false
```

2.exec

把字符串中符合正则要求的第一项以及一些其他信息，以数组的形式返回

语法： 正则.exec(字符串)

全局和exec没有关系

```
  // let reg = /a/;
  // console.log(reg.exec('cancancan'));//['a', index: 1, input: 'cancancan',
groups: undefined]
```

正则表达式里面的符号

1.[]：表示区间范围中的一个字符。

[0-9]:表示数字0-9中的一个

[a-z]:表示字母a-z中的一个

[A-Z]:表示字母A-Z中的一个

[5-9]:表示数字5-9中的一个

[049]:表示数字049中的一个

[0-9a-zA-Z]:表示数字字母中的一个

中括号里面可以设置^,表示取反

[^0-9]:表示非数字

```
//    var reg = /[^049]/;
//    console.log(reg.test('4'));//false
//    console.log(reg.test('dasfd9safdasf'));//true
```

2.行首匹配(^)和行尾匹配($)

边界符是限定字符串的开始和结束的

从行首开始一直到行尾进行匹配，实现恒等匹配

```
// let reg = /^hello$/;
// console.log(reg.test('hello'));//true
// console.log(reg.test('hellohello'));//false
// console.log(reg.test('helloo'));//false
```

3.{}:匹配数量

a{3} : 匹配3个a

a{3,5} : 匹配至少3个，最多5个a

a{3,} : 匹配至少3个

```
// let reg = /^hello{3,6}$/;
// console.log(reg.test('helloo'));//false
// console.log(reg.test('hellooo'));
// console.log(reg.test('helloooo'));
// console.log(reg.test('hellooooo'));
// console.log(reg.test('helloooooo'));
// console.log(reg.test('hellooooooo'));//false
```

4.\ ： 转译符号，把有意义的符号转换成没有意义的字符，把没有意义的字符转换成有意义的符号

\s ： 匹配空白字符（空格 / 制表符 /...）

```
// let reg = /\s/;
// console.log(reg.test('abc'));//false
// console.log(reg.test('a bc'));//true
```

\S ： 匹配非空白字符

```
// console.log(/^\S$/.test('a'));//true
// console.log(/^\S$/.test(' '));//false
```

\d ： 匹配数字[0-9]

```
// let reg = /^\d$/;
// console.log(reg.test('1'));
```

\D ： 匹配非数字

\w ： 匹配数字字母下划线[0-9a-zA-Z_]

\W ： 匹配非数字字母下划线

```
// var reg = /^\w\w\w$/;
// console.log(reg.test('abc'));
// console.log(reg.test('111'));
// console.log(reg.test('AAA'));
```

\n:字符串换行

简单来说就是在一些字符前加 “\” 使它具有其他意义,例如：\d \D \w \W \s \S

如果在一个正常字符前添加反斜杠，JS会忽略该反斜杠 ,就不再对其做特殊处理，当做普通字符使用。

```
// var reg = /^\m\m\m$/;  // 等同于/mmm/
// console.log(reg.test('mmm'));//true
// console.log(reg.test('mmn'));//false
// console.log('abc\ndef');
```

正则的其他符号

一.限定符

x*:匹配0个或者多个x

```
// let reg = /^x*$/;
// console.log(reg.test(''));
```

```
// console.log(reg.test('x'));
// console.log(reg.test('xxxxxxxx'));
```

x?:匹配0个或者1个x

```
// let reg = /^x?$/;
// console.log(reg.test(''));
// console.log(reg.test('x'));
// console.log(reg.test('xxxxxxxx'));//false
```

x+:匹配1个或者多个x

```
// let reg = /^x+$/;
// console.log(reg.test(''));//false
// console.log(reg.test('x'));
// console.log(reg.test('xxxxxxxx'));
```

二.限定一组元素()

```
// let reg = /^abc{3}$/;
// console.log(reg.test('abccc'));//true
// let reg1 = /^(abc){3}$/;
// console.log(reg1.test('abcabcabc'));//true
// console.log(reg1.test('abccc'));//false
```

三.| ： 或，正则里面的或 a|b 表示字母 a 或者 b 都可以

四.点(.) ： 匹配非换行的任意字符,匹配除了换行符(\n)之外的其他任意字符

```
// let reg = /^.com$/;
// console.log(reg.test('.com'));
// console.log(reg.test('acom'));
// console.log(reg.test('bcom'));
// console.log(reg.test('\ncom'));//false
// 必须匹配点com
// let reg = /^\.com$/
// console.log(reg.test('.com'));//true
// console.log(reg.test('acom'));//false
// console.log(reg.test('bcom'));//false
```

字符串对象的方法

一.replace

是将字符串中满足正则条件的字符串替换掉

语法： 字符串.replace(正则，要替换的字符串)

参数1：正则规则

参数2：可以是替换的字符串，也可以是一个函数。

返回值替换后的字符串

```
// 敏感词过滤
// let str = '这件衣服是你妈的，还是你妹的，我猜是你大爷的';
// let reg = /妈|妹|大爷/g;    //修饰符，表示全局匹配
// console.log(str.replace(reg, '**'));

// let str = '1a2b3c4d5e6f';
// // console.log(str.replace(/[a-z]/g, ''));//123456
// str.replace(/[a-z]/g, function (res) {
//    console.log(res);//a,b,c,d,e,f匹配的结果  replace函数做参数，做参数的函数又有一个参数，而这个参数就是匹配的结果。
// });
```

二.split 根据参数将字符串分割成数组(两个参数：分割符和分割后数组的长度)

```
// let str = '1a2b3c4d5e6';
// console.log(str.split(/[a-z]/));//['1', '2', '3', '4', '5', '6']
```

三.match找到字符串中符合正则条件的内容返回

```
// let str = '1a2b3c4d5e6';
// console.log(str.match(/[a-z]/g));//['a', 'b', 'c', 'd', 'e']
```

四.search是查找字符串中是否有满足正则条件的内容，返回的是索引位置

返回的值为-1，表示不符合正则条件

```
// var reg = /\d{3}/;//找三个数字
// var str = 'hello423'
// var str2 = 'hello'
// console.log(str.search(reg)) // 5
// console.log(str.search('423'));//5
// console.log(str2.search(reg)) // -1  没有找到
```

正则分组和$符的应用 - 补充

1.正则的属性 - lastIndex

获取匹配字符的下一个索引，这个属性配合全局匹配意义比较重要。

正则全局匹配存在一个移动的属性控制匹配关系

```
// let reg = /[a-z]/g;
// let str = '1a2b3';
// console.log(reg.test(str));//true  匹配的是a字符
// console.log(reg.lastIndex);//2(获取匹配字符的下一个索引)
```

```
// console.log(reg.test(str));//true  匹配的是b字符
// console.log(reg.lastIndex);//4(获取匹配字符的下一个索引)
// console.log(reg.test(str));//false 后面没有匹配的字符串，从头开始 将lastIndex重置为0
// console.log(reg.lastIndex);//0
```

排除上面的情况，直接将正则和匹配的字符串不存变量，这样每次都是从头开始

```
// console.log(/[a-z]/g.test('1a2b3'));//匹配的是a字符
// console.log(/[a-z]/g.test('1a2b3'));//匹配的是a字符
// console.log(/[a-z]/g.test('1a2b3'));//匹配的是a字符
```

2.正则的分组

匹配常见中文：/[\u4e00-\u9fa5]/

```
// let str = 'a你好bc很好de大家好fg';
// console.log(str.match(/[\u4e00-\u9fa5]/g));// ['你', '好', '很', '好', '大', '家', '好']
// console.log(str.match(/[\u4e00-\u9fa5]+/g));// ['你好', '很好', '大家好']
```

正则表达式里面添加()进行分组

非全局情况下可以自动捕获分组里面的值

```
// let reg = /([\u4e00-\u9fa5])([\u4e00-\u9fa5])([\u4e00-\u9fa5])/;
// let str = '大家好';
// console.log(str.match(reg));// ['大家好', '大', '家', '好', index: 0, input: '大家好', groups: undefined]
```

非捕获性分组 - ?:

```
// let reg = /(?:[\u4e00-\u9fa5])(?:[\u4e00-\u9fa5])(?:[\u4e00-\u9fa5])/;
// let str = '大家好';
// console.log(str.match(reg));
// ['大家好', index: 0, input: '大家好', groups: undefined]
```

3.正则分组，正则表达式里面约定了\1-\9分别可以表示第一个分组到第九个分组

要求\1匹配的内容和第一个分组匹配的内容一样的

要求\2匹配的内容和第二个分组匹配的内容一样的

```
// let str = '小小的花园里面挖呀挖呀挖'
// let reg = /([\u4e00-\u9fa5])([\u4e00-\u9fa5])\1\2/g;
// console.log(str.match(reg));//['挖呀挖呀']

// let str = '我希望大家好好学学，天天上上，平时要好好休休，对自己痛痛快快';
// let reg = /([\u4e00-\u9fa5])\1([\u4e00-\u9fa5])\2/g;
// console.log(str.match(reg));// ['好好学学', '天天上上', '好好休休', '痛痛快快']
```

通过正则对象下面的静态属性($1,$2,$3....)获取分组的值

注意：使用静态属性，正则必须执行

console.log(RegExp.$1);//第一个分组匹配的内容 痛

console.log(RegExp.$2);//第二个分组匹配的内容 快

4.正则表达式的贪婪和惰性(正则都是贪婪的)

如果在正则表达式表示匹配内容的地方添加一个?,正则就是惰性匹配

?的两种情况

1种直接放在某个字符后面，表示匹配0个或者1个

2种直接放在某个量词后面，表示惰性匹配

```
// let reg = /8.+8/
// let str = '8abcdefg8abcdefg8';
// console.log(str.replace(reg, '*'));//*
// let reg = /8.+?8/
// let str = '8abcdefg8abcdefg8';
// console.log(str.replace(reg, '*'));//*abcdefg8
```

扩展eval函数 - ES提供的函数

1.eval()函数就是一个js语言的执行器, 它能把其中的参数(字符串)按照JavaScript语法进行解析并执行

```
// eval('let a = 1; let b = 2; console.log(a+b)');//let a = 1; let b = 2; console.log(a + b)
```

2.注意eval函数在处理大括号”{}”时，会把它当成是一个语句块，只会执行该语句，并不会返回该值。

对象格式的字符串，不会当成一个整体去执行,产生错误

```
// console.log(eval('{ a: 1, b: 2, c: 3 }'));
```

3.正确的写法是在对象格式外加上小括号”()”，使其强制转换成对象表达式，并返回该对象值。

```
// console.log(eval('({ a: 1, b: 2, c: 3 })'));//{a: 1, b: 2, c: 3}
```

核心应用：使用eval将**字符串格式的对象**转换成**真正的对象**