day14 正则

正则概述

正则是用于检验对应的字符串的一种特殊表达式。一般用于用户格式验证。

正则对象声明

使用//来声明

```
var regexp = /a/ig
console.log(regexp)
```

使用new关键字声明

```
//第一个传入的是对应的正则表达式 第二个传入为当前匹配模式
var regexp = new RegExp('a','g')
console.log(regexp)
```

正则的匹配修饰符

- g 全局
- i 不区分大小写
- m 换行
- s 单个匹配

正则对象的方法及属性

方法

- test 验证是否匹配
- exec 返回匹配的数组

```
//正则对象的方法 boolean
console.log(regexp.test('hello')) //false
console.log(regexp.test('apple')) //true
//exec 执行 返回的是一个数组 类于match
console.log(regexp.exec('ooA'))
```

属性

- dotAll 是否到匹配修饰符s
- flags 匹配修饰符
- global 是否全局查找
- ignoreCase 是否不区分大小写
- lastIndex 下一次匹配的开始索引

```
//属性
console.log(regexp.dotAll)//是否到匹配修饰符s
console.log(regexp.flags) //匹配修饰符
console.log(regexp.global) //是否全局查找
console.log(regexp.ignoreCase) //是否不区分大小写
console.log(regexp.lastIndex) //下一次匹配的开始索引
```

正则表达式中的元字符（特殊字符）

- **^ 表示开头**
- **$ 表示结尾**

```
var regexp = /^ab$/  //以a开头 以b结尾
console.log(regexp.test('ab')) //true
console.log(regexp.test('a1b')) //false
```

- **[] 表示里面的任意一个元素**

```
//[] 表示里面的任意一个元素
var reg = /^[abc]a$/ //只匹配aa ba ca
console.log(reg.test('ab')) //false
var reg = /^[abc][ab]$/ //只匹配aa ab ba bb ca cb
console.log(reg.test('ab')) //true
```

- **{} 表示个数**

  {n} 表示n个

  {n,m} 表示n到m个 （必要条件 m>n）

  {n,} 表示n到无穷个

```
//{} 表示个数
var reg = /[abc]{2}/ //里面包含  /[abc]{2}/ == /[abc][abc]/
console.log(reg.test('ab'))//true
//{2,3} 俩个到三个
var reg = /^[abc]{2,3}$/
console.log(reg.test('abc'))//true
//{2,} 俩个到无穷个
var reg = /[a]{2,}/
console.log(reg.test('aaab'))//true
```

- **() 表示分组**

```
//() 表示分组
var reg = /(ab){2}/ //abab
console.log(reg.test('abaa')) //false
```

- **数字的表示方式**

  \d 表示数字 \D表示非数字

  [0-9] 表示数字

```
//\d表示数字 [0-9]
console.log(/\d/.test('1a'))//true
console.log(/[0-9]/.test('1a'))//true
//\D表示非数字 大小的表示相反
console.log(/\D/.test('123')) //false
```

- **字母表示方式**

```
//表示字母 [A-Z]大写字母 [a-z] 小写字母  [A-z] 大写字母和小写字母
console.log(/[A-Z]/.test('abc')) //false
console.log(/[A-z]/.test('abc')) //true
console.log(/[a-z]/.test('abc')) //true
```

- **\w 表示数字字母下滑线 \W 非数字字母下滑线**

```
// \w表示数字 字母及下滑线 \W表示非数字 字母 下滑线
console.log(/\w/.test('_')) //true
console.log(/\w/.test('1')) //true
console.log(/\w/.test('a')) //true
console.log(/\w/.test('-')) //false
console.log(/\W/.test('-')) //true
console.log(/\W/.test('_')) //false
```

- **\s 表示空白字符 \S非空白字符**

```
//\s表示空白字符 \S表示非空白字符
console.log(/\s/.test("  ")) //true
console.log(/\S/.test("  ")) //false
console.log(/\S/.test("a")) //true
```

- `+` **表示1个到多个 相当于{1,}**
- `?` **表示0个到1个 相当于{0,1}**
- `*` **表示0个到多个 相当于{0,}**

```
// +表示0到多个 ?表示0个到一个 *表示0个到多个
console.log(/^([\d\w]*)+$/.test('19w8101397198347')) //true
console.log(/^([\d\w]*)+$/.test('?_+1'))//false
console.log(/^([\d\w]*)+$/.test(''))//true
```

- **| 或者符号**

```
//| 或者 | 或符号建议和分组一起使用
console.log(/^([a]{2})|([b]{3})$/.test('aa'))//true
console.log(/^([a]{2})|([b]{3})$/.test('bbb'))//true
console.log(/^([a]{2})|([b]{3})$/.test('abb'))//false
```

- **.表示任意字符**

```
//. 任意字符串
console.log(/^.$/.test('1')) //true
console.log(/^.$/.test('b'))//true
console.log(/^.$/.test('?'))//true
console.log(/^.$/.test(' '))//true
```

正则的转义

转义的概念就是将我们的元字符变为普通字符串

- 使用[] 进行转义 只能对应的普通的元字符 (? * . +等)
- 使用\转义字符 可以对任意进行转义

```
//转义 第一种方式将对应的非修饰的字母 不带反斜杠开头的元字符 使用[]把他包起来
console.log(/[?]/.test('?'))
console.log(/[.]/.test('.'))//true
console.log(/[.]/.test('a'))//false
console.log(/[\d]/.test('\d')) //false
console.log(/[\w]/.test('\w')) //false
//第二种使用转义字符 \
console.log(/\[\]/.test('[]'))
console.log(/\./.test('.'))
```

支持正则的字符串的方法

- match 匹配
- search 查找
- replace 替换
- split 切割