正则

- 正则表达式，又名 “规则表达式”
- 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
- 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

```
var reg = /\d+/
var str1 = '123'
var str2 = 'abc'
console.log(reg.test(str1)) // true
console.log(reg.test(str2)) // false
```

  - 上面的变量 `reg` 就是定制好的规则
  - 检测 `str1` 这个字符串的时候，符合规则
  - 检测 `str2` 这个字符串的时候，不符合规则

创建一个正则表达式

- 想制定 “规则”，必须要按照人家要求的方式来制定
- 把一些字母和符号写在 `//` 中间的东西，叫做正则表达式，比如 `/abcdefg/`
- 创建正则表达式有两个方式 **字面量** 和 **构造函数创建**

字面量创建

```
// 下面就是字面量创建一个正则表达式
var reg = /abcdefg/
```

- 这个正则表达式就可以去检测字符串了

构造函数创建

```
// 下面就是构造函数创建一个正则表达式
var reg = new RegExp('abcdefg')
console.log(reg) // /abcdefg/
```

- 使用构造函数方式创建的和字面量创建的，得到的结果一样

正则表达式里面的符号

- 知道了怎么创建一个正则表达式以后，我们就来详细的说一下正则表达式里面涉及到的一些符号了

元字符

- `.` ： 匹配非换行的任意字符
- `\` ： 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/…）
- `\S` ： 匹配非空白字符
- `\d` ： 匹配数字
- `\D` ： 匹配非数字
- `\w` ： 匹配数字字母下划线
- `\W` ： 匹配非数字字母下划线

- 有了元字符我们就可以简单的制定一些规则了

```
var reg = /\s/
var str = 'a b'
var str2 = 'ab'
console.log(reg.test(str)) // true
console.log(reg.test(str2)) // false
var reg = /\d/
var str = 'abc1'
var str2 = 'abc'
console.log(reg.test(str)) // true
console.log(reg.test(str2)) // false
var reg = /\w/
var str = 'a1'
var str2 = '#@$'
console.log(reg.test(str)) // true
console.log(reg.test(str2)) // false
```

限定符

- `*` ： 前一个内容重复至少 0 次，也就是可以出现 **0 ～ 正无穷** 次
- `+` ： 前一个内容重复至少 1 次，也就是可以出现 **1 ～ 正无穷** 次
- `?` ： 前一个内容重复 0 或者 1 次，也就是可以出现 **0 ～ 1** 次
- `{n}` ： 前一个内容重复 n 次，也就是必须出现 **n** 次
- `{n,}` ： 前一个内容至少出现 n 次，也就是出现 **n ～ 正无穷** 次
- `{n,m}` ： 前一个内容至少出现 n 次至多出现 m 次，也就是出现 **n ～ m** 次

- 限定符是配合元字符使用的

```
// 下面正则表示验证数字出现 0 ～ 正无穷次都可以
var reg = /\d*/
var str = 'abc'
var str2 = 'abc1'
var str3 = 'abc123'
console.log(reg.test(str)) // true
console.log(reg.test(str2)) // true
console.log(reg.test(str3)) // true
// 下面正则表示验证数字出现 1 ～ 正无穷次都可以
var reg = /\d+/
var str = 'abc'
var str2 = 'abc1'
var str3 = 'abc123'
console.log(reg.test(str)) // false
console.log(reg.test(str2)) // true
console.log(reg.test(str3)) // true
// 下面正则表示验证数字出现 0 ~ 1 次都可以
var reg = /\d?/
var str = 'abc'
var str2 = 'abc1'
console.log(reg.test(str)) // true
console.log(reg.test(str2)) // true
// 下面正则表示验证数字必须出现 3 次
var reg = /\d{3}/
var str = 'abc'
var str2 = 'abc1'
var str3 = 'abc123'
console.log(reg.test(str)) // false
console.log(reg.test(str2)) // false
console.log(reg.test(str3)) // true
// 下面正则表示验证数字出现 3 ～ 正无穷次
var reg = /\d{3,}/
var str = 'abc'
var str2 = 'abc1'
var str3 = 'abc123'
var str4 = 'abcd1234567'
console.log(reg.test(str)) // false
console.log(reg.test(str2)) // false
console.log(reg.test(str3)) // true
```

```
console.log(reg.test(str4)) // true
// 下面正则表示验证数字只能出现 3 ～ 5 次
var reg = /\d{3,5}/
var str = 'abc'
var str2 = 'abc1'
var str3 = 'abc123'
var str4 = 'abc12345'
console.log(reg.test(str)) // false
console.log(reg.test(str2)) // false
console.log(reg.test(str3)) // true
console.log(reg.test(str4)) // true
```

边界符

- `^` ： 表示开头
- `$` ： 表示结尾

- 边界符是限定字符串的开始和结束的

```
// 下面表示从开头到结尾只能有数字，并且出现 3 ～ 5 次
var reg = /^\d{3,5}$/
var str = 'abc'
var str2 = 'abc123'
var str3 = '1'
var str4 = '1234567'
var str5 = '123'
var str6 = '12345'
console.log(reg.test(str)) // false
console.log(reg.test(str2)) // false
console.log(reg.test(str3)) // false
console.log(reg.test(str4)) // false
console.log(reg.test(str5)) // true
console.log(reg.test(str6)) // true
```

特殊符号

- `()` ： 限定一组元素
- `[]` ： 字符集合，表示写在 `[]` 里面的任意一个都行
- `[^]` ： 反字符集合，表示写在 `[^]` 里面之外的任意一个都行
- `-` ： 范围，比如 `a-z` 表示从字母 a 到字母 z 都可以
- `|` ： 或，正则里面的或 `a|b` 表示字母 a 或者 b 都可以

- 现在我们就可以把若干符号组合在一起使用了

```
// 下面是一个简单的邮箱验证
// 非_$开头，任意字符出现至少6次，一个@符号，(163|126|qq|sina)中的任意一个，一个点，(com|cn|net)中的任意一个
var reg = /^[^_$].{6,}@(163|126|qq|sina)\.(com|cn|net)$/
```

标示符

- `i` ： 表示忽略大小写
  - 这个 i 是写在正则的最后面的
  - `/\w/i`
  - 就是在正则匹配的时候不去区分大小写
- `g` ： 表示全局匹配
  - 这个 g 是写在正则的最后面的
  - `/\w/g`
  - 就是全局匹配字母数字下划线

正则表达式的方法

- 正则提供了一些方法给我们使用
- 用来检测和捕获字符串中的内容的

test

- `test` 是用来检测字符串是否符合我们正则的标准
- 语法： `正则.test(字符串)`
- 返回值： boolean

```
console.log(/\d+/.test('123')) // true
console.log(/\d+/.test('abc')) // false
```

exec

- `exec` 是把字符串中符合条件的内容捕获出来
- 语法： `正则.exec(字符串)`
- 返回值： 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

```
var reg = /\d{3}/
var str = 'hello123world456你好789'
var res = reg.exec(str)
console.log(res)
/*
 ["123", index: 5, input: "hello123world456你好789", groups: undefined]
 0: "123"
 groups: undefined
 index: 5
 input: "hello123world456你好789"
 length: 1
 __proto__: Array(0)
*/
```

- 数组第 0 项就是匹配到的字符串内容
- index 属性表示从字符串的索引几开始是匹配的到字符串

字符串的方法

- 字符串中有一些方法也是可以和正则一起使用的

search

- `search` 是查找字符串中是否有满足正则条件的内容
- 语法： `字符串.search(正则)`
- 返回值 ： 有的话返回开始索引，没有返回 -1

```
var reg = /\d{3}/
var str = 'hello123'
var str2 = 'hello'
console.log(str.search(reg)) // 5
console.log(str2.search(reg)) // -1
```

match

- `match` 找到字符串中符合正则条件的内容返回
- 语法： `字符串.match(正则)`
- 返回值 ：

- 没有标示符 g 的时候，是和 exec 方法一样
- 有标示符 g 的时候，是返回一个数组，里面是匹配到的每一项

```
var reg = /\d{3}/
var str = 'hello123world456'
var str2 = 'hello'
console.log(str.match(reg))
// ["123", index: 5, input: "hello123wor456", groups: undefined]
console.log(str2.match(reg)) // null
```

```
var reg = /\d{3}/g
var str = 'hello123world456'
var str2 = 'hello'
console.log(str.match(reg))
// ["123", "456"]
console.log(str2.match(reg)) // null
```

replace

- `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)`
- 返回值 ： 替换后的字符串

```
var reg = /\d{3}/
var str = 'hello123world456'
var str2 = 'hello'
console.log(str.replace(reg)) // hello666world456
console.log(str2.replace(reg)) // hello
```

```
var reg = /\d{3}/g
var str = 'hello123world456'
var str2 = 'hello'
console.log(str.replace(reg)) // hello666world666
console.log(str2.replace(reg)) // hello
```