### 正则表达式

#### 1.正则表达式

正则表达式是使用模式匹配文本字符串的表达式。

使用正则表达式可以使代码更简洁、更优雅。

使用场景：

1. 操作HTML节点中的字符串。
2. 使用CSS选择器表达式定位部分选择器。
3. 判断一个元素是否具有指定的类名（class）。
4. 输入校验。

#### 2.正则形式

创建正则表达式的方式有两种：

1. 使用正则表达式字面量。
2. 通过创建RegExp对象的实例。

    // 使用字面量创建正则表达式。 /正则表达式/匹配模式

    const pattern = /test/

    // 创建RegExp对象的实例

    const pattern = new RegExp('test')

使用字面量的方式更加简单，使用构造函数创建更加灵活。

匹配模式的标记如下：

1. g：全局模式。查找符合模式字符串的全部内容。
2. i：不区分大小写。查找匹配时忽略patter和字符串的大小写。
3. m：允许多行匹配。对获取textarea元素的值很有用。
4. y：粘连匹配。从最后一个匹配的位置开始。
5. u：Unicode模式，启用Unicode匹配。

#### 3.术语和操作符

|  |  |  |
| --- | --- | --- |
| 精确匹配 | /test/ | 必须完全匹配test的字符串，且的关系。 |
| 匹配字符集 | /[abc]/ | 匹配abc中的任意一个字符，互斥的关系。 |
| /[^abc]/ | 匹配abc以外的字符，非的关系。 |
| /[a-m]/ | 匹配按字母顺序a-m之间所有字符，或的关系。 |
| 转义 | /\/ | 反斜线对其后面的字符进行转义，使其匹配本身的含义。 |
| 起止符号 | /^test/ | 确保匹配字符串的开始，匹配test出现在字符串的开头。 |
| /$test/ | 确保匹配字符串的结束，匹配test出现在字符串的结尾。 |
| /^test$/ | 同时使用表示匹配整个字符串。 |
| 重复出现 | /t?est/ | 指定可选字符出现0次或1次。匹配test或est |
| /t+est/ | 指定字符出现1次或多次。匹配test或ttest |
| /t\*est/ | 指定字符出现0次、1次或多次。 |
| /a{4}/ | 指定重复次数。匹配四个连续的a，aaaa |
| /a{4,10}/ | 指定重复次数的范围。匹配4-10个连续的字符a。 |
| /a{4，}/ | 指定开放区间。匹配4或更多个连续的a。 |
| 分组 | /(ab)+/ | 匹配一个或多个连续的ab。 |
| 或操作符 | /a|b/ | 匹配a或者b。 |
| 预定义字符集 | \d | 匹配任意十进制数字，等价于[0-9]。 |
| \D | 匹配除了十进制数字外的任意字符，等价于[^0-9]。 |
| \s | 匹配任意空白符。 |
| \S | 匹配除任意空白符。 |
| \w | 匹配任何字母、数字和下划线，等价于[A-Za-z0-9]。 |
| \W | 匹配除了字母、数字和下划线之外的字符，等价于[^A-Za-z0-9]。 |
| . | 匹配除了换行字符。 |
|  |  |  |

匹配所有内容可以用[\s\S]或者[\d\D]。

#### 3.正则进阶—捕获

捕获就是带匹配的字符串，贼在正则表达式中，使用圆括号定义捕获。放在圆括号内就变成了一个整体（原子组），组内就是一个整体，组编号从左到右为\1\2……

捕获匹配的片段，首先判断字符串是否匹配模式，如果匹配成功，通过match方法返回捕获到的值，若未匹配成功，则返回null。

每个字符串可以使用match函数，match函数的传入参数是正则表达式，返回值是匹配到的全部字符串以及全部捕获。使用replace函数可以对固定字符串进行替换。

    let s = `

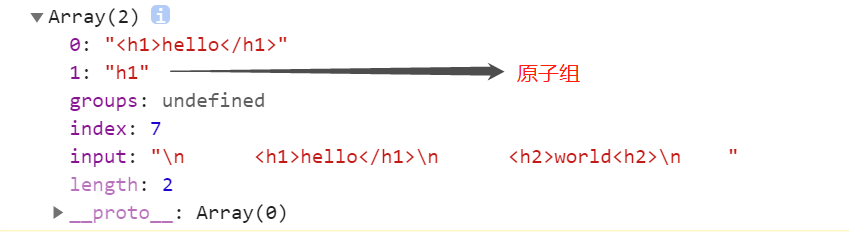
      <h1>hello</h1>

      <h2>world<h2>

    `

    let reg = /<(h[1-6])>[\s\S]\*<\/\1>/i

    console.dir(s.match(reg));



对捕获结果进行引用的方式有两种：

1. 自身匹配。
2. 替换字符串。可以使用字符串的replace进行替换，对替代字符串内获取捕获，不使用反向引用可以使用$1、$2、$3等标记捕获序号。

    //  使用replace对替代字符串内获取捕获。

    let res = `

    <h1>hello</h1>

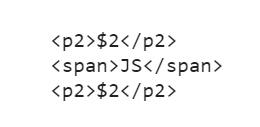
    <span>JS</span>

    <h2>world</h2>

    `

    let reg = /<(h[1-6])>[\s\S]\*<\/\1>/gi

    console.log(res.replace(reg,`<p2>$2</p2>`));



当匹配比较复杂的时候，replace不仅支持替换值，而且支持替换函数作为参数，当第二个参数是函数时，对每一个匹配到的值都会调用一遍。

括号从左往右数，有几个右括号就是几。

    let res = `

    <h1>hello</h1>

    <span>JS</span>

    <h2>world</h2>

    `

    let reg = /<(h[1-6])>(\w([\s\S]+))+<\/\1>/gi

    res.replace(reg,(p0,p1,p2,p3) => {

      // console.log(p0)     // <h1>hello</h1>  <h2>world</h2>

      // console.log(p1)     // h1 h2

      // console.log(p2)     // hello world

      // console.log(p3)     // ello orld

      return `<p>${p2}</p>`

    })

    console.log(res)

