RegExp

> ! 本文介绍RegExp相关的知识点，包括正则表达式的创建、匹配规则等内容。

RegExp简单介绍

正则表达式 Regular Expression 是描述字符模式的对象，在JavaScript语言中提供了内置的 RegExp 来处理正则。正则表达式能够进行强大的 模式匹配 和 文本检索与替换 功能，在前端开发中往往有大量的表单数据校验的工作，使用正则表达式可以减少数据校验的工作量。

在JavaScript语言中，除 RegExp 内置构造函数外，字符串操作中也有很多操作涉及到正则，包括字符串的 `macth()` 、 `search()` 、 `split()` 以及 `replace()` 等方法都接受正则表达式作为参数，而这也为我们操作字符串提供了更强大的功能。

正则表达式的创建

正则表达式的创建支持两种形式，一种是直接字面量方式创建，一种是使用 RegExp 构造函数方式创建，两种创建方式得到的正则实例是等价的。

① 字面量

字面量创建正则的语法 `var reg = / pattern / flags;` 其中 pattern(模式) 部分可以是任何简单或复杂的正则表达式，而 flags(参数) 部分支持三种匹配模式。

```HTML
g: global               表示全局匹配
i: ignoreCase           表示忽略大小写
m: multiline            表示多行匹配,影响^，$的匹配结果
```

JavaScript语言在字面量创建正则的语法中，模式部分可以直接写特定的字符串文本，也可以使用正则元字符，需要注意的是因为这些元字符在正则表达式中都有一种或者多种特殊的用途，因此如果想要匹配字符串中包含的这些字符，那么就必须对它们进行转义操作。

```
/*01-字面量的方式创建*/
var reg1 = /文顶顶/g;                    /*匹配文顶顶文本，全局匹配*/
var reg2 = /[a-zA-Z0-9]\d{3}/g;          /*匹配以字母和数字开头后面跟3个数字的文本，全局匹配*/
var reg3 = /^javaScript/g;               /*匹配以JavaScript开头的文本，全局匹配*/
var reg4 = /^javaScript/gm;              /*匹配以JavaScript开头或作为行首的文本，全局多行匹配*/
var reg5 = /[xm]xia/i;                   /*匹配xxia或者mxia文本，不区分大小写*/

/*.是正则中的元字符,代表除了换行外的所有字符*/
var reg6 = /.com/gi;                     /*匹配所有以com结尾的4个字符，不区分大小写*/
var reg7 = /\.com/gi;                    /*匹配所有.com文本，不区分大小写*/
```

```
/*02-正则测试*/
console.log(reg1.test("你好，文顶顶！"));    //true
console.log(reg2.test("a123"));           //true
console.log(reg2.test("5123"));           //false
console.log(reg2.test("-123"));           //false

var str = "java\njavaScript";
console.log(reg3.test(str));              //false
console.log(str.match(reg3));             //null
console.log(str.match(reg4));             //["javaScript"]
console.log(reg5.test("xxiao"));          //true
console.log(reg5.test("MxiAo"));          //true
console.log(reg5.test("xiongXxiao"));     //true
console.log(reg5.test("xiaoxiao"));       //false

console.log(reg6.test("baiducom"));       //true


console.log(reg7.test("baiducom"));       //false
console.log(reg7.test("wendingding.com"));//true
```

② 构造函数

RegExp 构造函数创建正则实例的语法 var reg = new RegExp(pattern , flags) ;其中 pattern(模式) 部分可以是任何简单或复杂的正则表达式(直接写字符串规则即可)，而 flags(参数) 部分和字面量方式一样也支持三种匹配模式。

```
/*01-构造函数创建正则表达式*/
var reg1 = new RegExp("文顶顶","g");             /*匹配文顶顶文本，全局匹配*/
var reg2 = new RegExp("[a-zA-Z0-9]\\d{3}","g") /*匹配字母和数字开头后跟3个数字的文本全局匹配*/
var reg3 = new RegExp("^javaScript","g");  /*匹配以JavaScript开头的文本，全局匹配*/
var reg4 = new RegExp("^javaScript","gm"); /*匹配以JavaScript开头或作为行首的文本全局多行匹配*/
var reg5 = new RegExp("[xm]xia","i");      /*匹配xxia或者mxia文本，不区分大小写*/

/*.是正则中的元字符,代表除了换行外的所有字符*/
var reg6 = new RegExp(".com","gi");        /*匹配所有以com结尾的4个字符，不区分大小写*/
var reg7 = new RegExp("\\.com","gi");      /*匹配所有.com文本，不区分大小写*/


/*02-正则测试*/
console.log(reg1.test("hello，文顶顶！"));  //true
console.log(reg2.test("a123"));           //true
console.log(reg2.test("5123"));           //false
console.log(reg2.test("-123"));           //false

var str = "java\njavaScript";
console.log(reg3.test(str));              //false
console.log(str.match(reg3));             //null
console.log(str.match(reg4));             //["javaScript"]
console.log(reg5.test("xxiao"));          //true
console.log(reg5.test("MxiAo"));          //true
console.log(reg5.test("xiongXxiao"));     //true
console.log(reg5.test("xiaoxiao"));       //false

console.log(reg6.test("baiducom"));       //true
```

```
console.log(reg7.test("baiducom"));       //false
console.log(reg7.test("wendingding.com"));//true
```

RegExp基本使用

字符串方法

```HTML
str.search()
    返回第一次匹配时所在的索引值,如果匹配不到则返回-1
str.match()
    - 默认匹配字符串，返回一个数组
        + 0:所匹配的字符
        + index:匹配第一个字符所在的索引
        + input:对字符串的引用
    - 全局匹配(g)，返回一个匹配所有字符串数组
    - 如果匹配不到则返回null
str.replace()  利用正则匹配来替换字符串
str.split()    利用正则匹配来切割字符串
```

```
/*01-replace方法的基本使用*/
/*01-1 清空字符串前面和后面的N个空格(实现字符串trim方法功能)*/
//var reg1   = /^\s+|\s+$/g;
var reg1     = new RegExp("^\\s+|\\s+$","g")
var result   = " trim test   ".replace(reg1,"");
console.log(result);                            //trim test

/*01-2 处理字符串中的敏感词*/
var  world = "华为荣耀";
var  reg2  = new RegExp(world,"g");
var  str   = "华为公司今天宣布旗下手机华为荣耀正式上市，华为荣耀价格感人只卖998";
console.log(str.replace(reg2, "****"));
/*华为公司今天宣布旗下手机****正式上市，****价格感人只卖998*/

/*02-match方法的基本使用*/
/*02-1 默认匹配*/
console.log(str.match(/华为荣耀/));
//["华为荣耀", index: 13, input: "华为公司今天宣布旗下手机华为荣耀正式上市，华为荣耀价格感人只卖998"
//groups: undefined]


/*02-2 全局匹配*/
console.log(str.match(/华为荣耀/g));  /*["华为荣耀", "华为荣耀"]*/

/*02-3 匹配失败*/
console.log(str.match(/苹果/));      /*null*/

/*03-search方法的基本使用*/
console.log(str.search(/华为荣耀/));  /*13*/
console.log(str.search(/苹果/));     /*-1*/

/*04-split方法的基本使用*/
var str1 = "2019-05-20";
var str2 = "熊大  ,熊二 ，光头强，   毛毛，  吉吉>凯特";
console.log(str1.split("-"));        //["2019", "05", "20"]
console.log(str2.split(","));        //["熊大", "熊二，光头强，   毛毛，  吉吉>凯特"]
```

```
console.log(str2.split(/\s*[,，>]\s*/g));//["熊大", "熊二", "光头强", "毛毛", "吉吉", "凯特"]
```

RegExp的核心成员

```HTML
regExp.test() 测试正则表达式用test方法,返回布尔值
    - 格式：正则表达式.test(字符串)
    - 用<正则表达式>测试<字符串>是否匹配,返回true/false
regExp.exec() 测试正则表达式exec方法
    - 格式：/xx/.exec(字符串)
global          是否应用g
ignoreCase      是否忽略大小写模式
multiline       是否应用多行匹配模式
source          包含正则表达式文本的字符串
lastIndex       整数，如果正则中应用了g全局匹配，则保存下一次开始检索的位置，在exec和test方法中会被用到
```

```
var reg = /小青蛙/gi;
console.log(reg.global);       //true
console.log(reg.ignoreCase);   //true
console.log(reg.lastIndex);    //0
console.log(reg.multiline);    //false
console.log(reg.source);       //小青蛙
```

RegExp匹配规则

```HTML
001 所有字母和数字都是按照字面量进行匹配,和字符串匹配等效 如/good/gi
002 字符类（只记小写字母即可）
    -  . ： 除换行以外的字符
    - \w ： 代表数字或字母或下划线

    - \W ： 非数字字母和下划线字符
    - \d ： 数字
    - \D ： 非数字
    - \s ： 代表一个空格
    - \S ： 空格以外的字符
    注意：以上所有字符类都只是匹配“一个”字符

003 边界处理
    - \b ： 匹配一个单词边界，也就是指单词和空格间的位置
    - \B ： 匹配非单词边界。

004 特殊符号
    ^  $  .  *  +  ?  =  !  :  |  \  /  ()  []  {}
    - []: 代表任意“单个字符” ,里面的内容表示“或”的关系
        + -: 代表范围
        + !: 代表非

    - (): 表示分组（n是以最左边括号出现的顺序排列）
        + $1: 表示第一个分组
        + $n: 表示第n个分组（不能写在正则表达式里）
        + \n: 在正则分组后面使用，表示对第n个分组的引用(一定要写在正则表达式里)
        建议：编写的正则分组数量越少越好
```

```
- |:  表示或者

- 锚点定位
    + ^: 表示以什么开头
    + $: 表示以什么结尾

- 表示数量，对前一个字符计数，
    + *: 代表0个或0个以上  {0,}
    + +: 代表1个或1个以上  {1,}
    + ?: 代表0个或1个      {0,1}
    + {}:
        \d{5}: 匹配5个数字
        \d{5,10}: 匹配5个到10个数字
        \d{5,}: 匹配5个或5个以上的数字
    说明:
        1）数量词*,+,{5,}，会尽可能多的去匹配结果（贪婪）
        2）在后面加一个?表示尽可能少的去匹配结果（非贪婪）
            google,goooogle ==> /go+/
```