JavaScript进阶

---RegExp正则表达式

---Error及异常处理

河北师范大学软件学院

Software College of Hebei Normal University

JavaScript进阶

---RegExp正则表达式

河北师范大学软件学院
Software College of Hebei Normal University

内容提纲

- **正则表达式简介及正则对象**
- **RegExp及String相关的正则方法**
- **正则表达式应用案例**

正则表达式（Regular Expression）简介

•什么是正则表达式

- 正则表达式是对字符串和特殊字符操作的一种逻辑公式，是对字符串执行模式匹配的工具
- 正则表达式通常被用来检索、替换那些符合某个模式(规则)的文本
- JS中正则表达式是一个描述字符模式的对象，可以通过字面量、RegExp构造器来生成

```
//正则对象的创建方式一 通过字面量直接创建
var reg1 = /[bcf]at/gi;

//正则对象的创建方式二 通过RegExp构造函数来实例化正则对象
var reg2 = new RegExp(/[bcf]at/,"gi");
var reg3 = new RegExp("[bcf]at","gi");

console.log("a fAt bat ,a faT cat".match(reg1));//同reg2 reg3
▶ (4) ["fAt", "bat", "faT", "cat"]
```


参见实例Demo1 创建JS正则表达式对象的两种方式

正则表达式（regular expression）简介

- 正则的语法概述和修饰符（g全局,i忽略大小写,m包含换行）

```
var regExp = /a?b/gi;
var matchResult = "xxabcaabbbxyz".match(regExp);
console.log(matchResult);
```

```
▶ (4) ["ab", "ab", "b", "b"]
```

正则表达式在线分析工具 https://regexper.com/

- 用正则对象来匹配字符串，也可以调用字符串方法来匹配

```
console.log("a2b3c4d".replace(/[2-3]/,"X"));//aXb3c4d
console.log("a2b3c4d".replace(/[2-3]/g,"X"));//aXbXc4d
```

方式一 字符串方法

```
var regExp = /a/gi;
console.log(regExp.test("ab"));//true
console.log(regExp.test("ab"));//false 为什么?
console.log(regExp.test("ab"));//true
console.log(regExp.test("ab"));//false 为什么?
```

方式二 正则对象方法

参见实例Demo2 正则的语法和修饰符 正则测试方法

正则对象相关字符

- 正则表达式的元字符 及\相关字符

元字符：1 3 5 a b c 等

转义字符：\t 、\v、\n、\r、\0、\f、\cX

与\相关的预定义特殊字符：\d、\D、\w、\W、\s、\S、\b 、\B（注意大小写的含义）

\	匹配将依照下列规则： 在非特殊字符之前的反斜杠表示下一个字符是特殊的，不能从字面上解释。例如，没有前面'\'的'b'通常匹配小写'b'，无论它们出现在哪里。如果加了'\',这个字符变成了一个特殊意义的字符，意思是匹配一个字符边界。 反斜杠也可以将其后的特殊字符，转义为字面量。例如，模式 /a*/ 代表会匹配 0 个或者多个 a。相反，模式 /a*/ 将 '*' 的特殊性移除，从而可以匹配像 "a*" 这样的字符串。 使用 new RegExp("pattern") 的时候不要忘记将 \ 进行转义，因为 \ 在字符串里面也是一个转义字符。

https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Guide/Regular_Expressions#note

参见实例Demo3 特殊字符实例

正则表达式定义及作用

- 正则表达式特殊字符 一 （字符类）

[ ]代表字符类，如[abc]代表abc中的任意一个字符，可以配合范围符号-如[a-c3-9]

[^]代表字符类取反，如[^abc]代表非abc中的任意一个字符

一个 - 代表范围，如[a-z] 、[a-z0-9A-Z]

一个 . 代表一个除了回车和换行符之外的所有字符 等效于[^\r\n]，（注意与*的区别和含义）

- 正则表达式特殊字符 二 （边界相关）

边界字符 ^ $ \b \B （注意^代表的意义与在[ ]中代表的意义不同）

- 正则表达式特殊字符 三 （量词）

？出现0次或1次（最多1次） +出现1次或多次（至少1次） *出现0次或多次（任意次）

{n} 出现n次 {n,m} 出现n到m次 {n,}出现至少n次

参见实例Demo4 Part1到Part3 字符类 边界相关 量词

正则表达式定义及作用

- 正则表达式贪婪模式与非贪婪模式

思考：“12345678”.replace(/\d{3,6}/,'X');返回多少?

默认为贪婪模式（即尽可能多的匹配），在量词后加？可设置为非贪婪模式

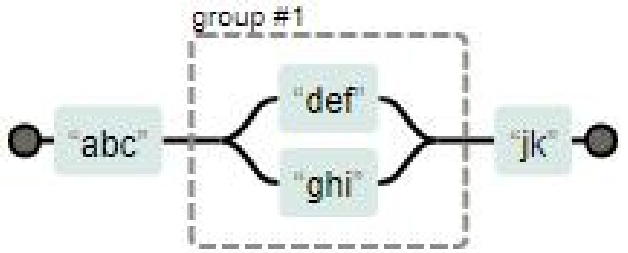
- 正则表达式的分组

思考：匹配Name连续出现3次的正则，/Name{3}/，这样可以么?

使用小括号来进行分组，如：/(Name){3}/g

或|、分组中的或|

/abcdef|ghijk/



/abc(def|ghi)jk/

参见实例Demo5 贪婪模式与非贪婪模式 正则的分组

正则表达式定义及作用

- 正则表达式的分组的反向引用

思考：如何将2017-10-23转成10/23/2017

```
"2017-10-23".replace(/(\d{4})-(\d{2})-(\d{2})/,"$2/$3/$1");
"10/23/2017"
```

- 正则表达式的分组的忽略分组

在分组内加上？：即可

```
"2017-10-23".replace(/(?:\d{4})-(\d{2})-(\d{2})/,"$2/$3/$1");
"23/$3/10"
```


参见实例Demo6 正则的分组的反向引用 和 忽略分组

内容提纲

- 正则表达式简介及正则对象
- RegExp及String相关的正则方法
- 正则表达式应用案例

RegExp及String相关的正则方法

- 正则表达式对象的属性（源自RegExp.prototype）
  - global 默认为false
  - ignore case 默认为false
  - multiline 默认为false
  - lastIndex 表示当前匹配内容的最后一个字符的下一个位置
  - sourse 正则表达式文本字符串

```
var reg1 = /\w/;
var reg2 = /\w/gi;
console.log(reg1.global,reg1.ignoreCase,reg1.lastIndex,reg1.source);
console.log(reg2.global,reg2.ignoreCase,reg2.lastIndex,reg2.source);
```

```
false false 0 "\w"
```

```
true true 0 "\w"
```


参见实例Demo7 正则表达式对象的属性

RegExp及String相关的正则方法

- 正则表达式RegExp原型方法（test）

```
var regExp = /a/gi;
console.log(regExp.test("ab"));//true
console.log(regExp.test("ab"));//false 为什么？
```

- 正则表达式RegExp原型方法（exec），可获得详细信息

```
var execExp2 = /\d{1,2}(\d)(\d)/g;
var ts = "12s342dsfsf233s";
console.log(execExp2.exec(ts),execExp2.lastIndex);//lastIndex为 6
console.log(execExp2.exec(ts),execExp2.lastIndex);//lastIndex为 14
```

```
▶ (3) ["342", "4", "2", index: 3, input: "12s342dsfsf233s"] 6
▶ (3) ["233", "3", "3", index: 11, input: "12s342dsfsf233s"] 14
```


参见实例Demo8 正则表达式相关的原型方法

RegExp及String相关的正则方法

- 字符串与正则相关的原型方法（String.prototype.search）

```
console.log("a1b2c3d4".search(/1/));//返回index 1
console.log("a1b2c3d4".search(/f/));//返回index -1 没找到
console.log("a1b2c3d4".search(/\d/g));//返回index 1 忽略全局
console.log("a1b2c3d4".search(/\d\w/g));//返回index 1 忽略全局
```

- 字符串与正则相关的原型方法（String.prototype.match）

```
console.log("a1b2c".match(/1/));//[ '1', index: 1, input: 'a1b2c' ]
console.log("a1b2c".match(/f/));//null
console.log("a1b2c".match(/\d/));//[ '1', index: 1, input: 'a1b2c' ]
console.log("a1b2c".match(/\d/g));//[ '1', '2' ]
```


参见实例Demo9 字符串与正则相关的原型方法

RegExp及String相关的正则方法

- 字符串与正则相关的原型方法（String.prototype.replace）

```
console.log("a,b,c,d".replace(",","X"));//aXb,c,d
console.log("a,b,c,d".replace(/,/g,"X"));//aXbXcXd
console.log("a2b3c4d".replace(/[2-3]/,"X"));//aXb3c4d
console.log("a2b3c4d".replace(/[2-3]/g,"X"));//aXbXc4d
```

- 字符串与正则相关的原型方法（String.prototype.split）

```
console.log("a,b,c,d".split(","));
console.log("a2b3c4d".split(/\d/));
```

```
▶ (4) ["a", "b", "c", "d"]
▶ (4) ["a", "b", "c", "d"]
```


参见实例Demo9 字符串与正则相关的原型方法

内容提纲

- 正则表达式简介及正则对象
- RegExp及String相关的正则方法
- **正则表达式应用案例**

RegExp及String相关的正则方法

- 常用正则表达式

/\w+((-\w+)|(\.\w+))*\@[A-Za-z0-9]+((\.|-)[A-Za-z0-9]+)*\.[A-Za-z0-9]+/ （邮箱）

/^[A-Za-z0-9_-]+$/ （密码）

/((?:(?:25[0-5]|2[0-4]\d|[01]?\d?\d)\.){3}(?:25[0-5]|2[0-4]\d|[01]?\d?\d))/ （IP地址）

/(.*)\.(rar|zip|7zip|tgz)$/ （压缩格式）

/(.*)\.(jpg|bmp|gif|ico|pcx|jpeg|tif|png|raw|tga)$/ （图片判断）

/^#[a-fA-F0-9]{6}$/ （颜色值）

/^[A-Za-z0-9_\-\u4e00-\u9fa5]+$/ （用户名）

/0?(13|14|15|18)[0-9]{9}/ （手机号）

/^[A-Za-z0-9_()（）\-\u4e00-\u9fa5]+$/ （公司名称）

- 集成开发环境中的应用

参考链接：
http://www1.qdfuns.com/tools.php?mod=regex

参见实例Demo10 字符串与正则相关的原型方法

Thank You!

作业

- 阅读《深入理解JavaScript》第19章 正则表达式

JavaScript进阶

---Error及异常处理

河北师范大学软件学院
Software College of Hebei Normal University

内容提纲

- **JS异常处理**
- **Error对象及其子对象**

JS异常处理

- JS中异常处理概述及异常处理的语法

```
try{
    //try_statements 可能出现错误部分
    console.log("try_statements");
    throw "Some Error";//可以抛出异常
}
catch(e){ //catch和finally至少有一个
    //catch_statements 捕获处理异常
    console.log("catch_statements",e);
}
finally{ //catch和finally至少有一个
    //finally_statements 最终处理
    console.log("finally_statements");
}
```

无论是否捕获到异常，finally都会执行

参见实例Demo11 异常处理语法

JS异常处理

- JS中异常处理的案例

```
window.onload = function () {
    window.Foo = function () {
        var inputValue = document.getElementById("inputID").value;
        try{
            var n = parseInt(inputValue);
            var a= new Array(n);//定义一个数组 传3试试、再传-5试试
            for(var i=0;i<n;i++){a[i] = i;}
        }catch(e){
            alert(e.name+e.message);
        }
        finally {
            document.getElementById("labelID").innerHTML = a;
        }
    };
};
```


参见实例LS中的Demo12 和 index.html

内容提纲

- JS异常处理
- Error对象及其子对象

JS中的错误以及Error对象

JS中的错误概述

- 当 JavaScript 引擎执行 JavaScript 代码时，会发生各种错误
- 可能是语法错误、或是由于浏览器差异产生的错误、或是来自服务器或用户导致的错误
- 有些错误是可以控制和避免的，有些是不可控的（比如来自用户输入等第三方的操作）

JS中对错误的处理

- 优化代码避免可控错误，对不可控错误需要使用异常处理来进行处理，避免程序直接崩溃

Error对象

- 当运行时错误产生时，会抛出一个错误对象，可以对此对象进行捕获和处理
- 也可以通过Error的构造器new一个错误对象，当检测到异常时或不满足逻辑时，手动抛出错误对象
- 所有错误对象的基础原型是Error.prototype，默认的name属性为“Error”，message属性为“”

参见实例Demo13 JS中的错误及Error对象

Error对象

- Error的子类
  - ReferenceError 引用错误、RangeError 范围错误、TypeError 类型错误
  - URIError 资源定位错误、EvalError 与eval( )有关的错误、其他错误

```
//Part 3333333333 类型错误 TypeError
try{
    var a;a.aa();
    //var a= new 123; //在chrome中测试
}catch(e){
    console.log(e.name,e.message);
}
finally {
    console.log("finally");//有无异常该句都会执行
}
```


参见实例Demo14 其他类Error

Thank You!