# JS用法

## 放入<head>或者<body>

<https://blog.csdn.net/lumeier/article/details/46398009>

<https://blog.csdn.net/Yy921117/article/details/51982159>

* 通常的做法是把函数放入 <head> 部分中，或者放在页面底部。这样就可以把它们安置到同一处位置，不会干扰页面的内容。
* 为什么我们经常看到有很多的人把js脚本放到head里面都不担心出问题？因为通常把javascript放在head里的话，一般都会绑定一个监听，当全部的html文档解析完之后，再执行代码。

## 外部JS

有时候你可能想在几个页面中运行同样的脚本程序， 而不需在各个页面中重复的写这些代码。这时你就要用到外部脚本。你可以把脚本写在一个外部文件中，保存在扩展名为 .js的文件中。例：

<script src="myScript.js"></script>

# JS注释

* 单行：//
* 多行：/\*…\*/

# JS输出

## window.alert()

使用 window.alert() 写入警告框

window.alert(5 + 6);

## document.write()

使用 document.write() 写入 HTML 输出

document.write(5 + 6);

## innerHTML

使用 innerHTML 写入 HTML 元素

document.getElementById("demo").innerHTML = 5 + 6;

## console.log()

使用 console.log()写入浏览器控制台

通过 F12 来激活浏览器控制台，并在菜单中选择“控制台”。

console.log(5 + 6);

# JS关键字

| 关键字 | 描述 |
| --- | --- |
| break | 用于跳出循环。 |
| catch | 语句块，在 try 语句块执行出错时执行 catch 语句块。 |
| continue | 跳过循环中的一个迭代。 |
| do ... while | 执行一个语句块，在条件语句为 true 时继续执行该语句块。 |
| for | 在条件语句为 true 时，可以将代码块执行指定的次数。 |
| for ... in | 用于遍历数组或者对象的属性（对数组或者对象的属性进行循环操作）。 |
| function | 定义一个函数 |
| if ... else | 用于基于不同的条件来执行不同的动作。 |
| return | 退出函数 |
| switch | 用于基于不同的条件来执行不同的动作。 |
| throw | 抛出（生成）错误 。 |
| try | 实现错误处理，与 catch 一同使用。 |
| var | 声明一个变量。 |
| while | 当条件语句为 true 时，执行语句块。 |

# JS运算符

其中位运算符处理 32 位数，运算中的任何数值运算数都会被转换为 32 位的数。结果会被转换回 64位JavaScript 数。

| 运算符 | 描述 | 例子 | 备注 |
| --- | --- | --- | --- |
| === | 等值等型 |  |  |
| !== | 不等值或不等型 |  |  |
| typeof | 返回变量的类型 | NaN 的数据类型是 number  数组(Array)的数据类型是object  日期(Date)的数据类型为 object  null 的数据类型是 object  未定义变量的数据类是undefined  new的数据类型是object | 始终返回字符串 |
| instanceof | 返回true，如果对象是对象类型的实例 |  |  |
| ^ | 异或 | 0101 ^ 0001  0100 | 计算一个数值的二进制补码，需要经过下列 3 个步骤： (1) 求这个数值绝对值的二进制码（例如，要求-18 的二进制补码，先求 18 的二进制码）； (2) 求二进制反码，即将 0 替换为 1，将 1 替换为 0； (3) 得到的二进制反码加 1。  位运算要对结果求补码。 |
| << | 零填充左位移  通过从右推入零向左位移，并使最左边的位脱落 | 1101 << 1  1010 |  |
| >> | 有符号右位移  通过从左推入最左位的拷贝来向右位移，并使最右边的位脱落 | 1101 >> 1  1110 |  |
| >>> | 零填充右位移  通过从左推入零向右位移，并使最右边的位脱落 | 1101 >>> 1  0110 |  |
| \*\* | 幂 | 5 \*\* 2  25 |  |

运算符优先级（从高到低）：

| 优先级 | 运算符 | 描述 | 例子 |
| --- | --- | --- | --- |
| 1 | ( ) | 表达式分组 | (3 + 4) |
| 2 | . | 成员 | person.name |
| 2 | [] | 成员 | person["name"] |
| 2 | () | 函数调用 | myFunction() |
| 2 | new | 创建 | new Date() |
| 3 | ++ | 后缀递增 | i++ |
| 3 | -- | 后缀递减 | i-- |
| 4 | ++ | 前缀递增 | ++i |
| 4 | -- | 前缀递减 | --i |
| 4 | ! | 逻辑否 | !(x == y) |
| 4 | typeof | 类型 | typeof x |
| 5 | \*\* | 求幂 (ES7) | 10 \*\* 2 |
| 6 | \* | 乘 | 10 \* 5 |
| 6 | / | 除 | 10 / 5 |
| 6 | % | 模数除法 | 10 % 5 |
| 7 | + | 加 | 10 + 5 |
| 7 | - | 减 | 10 - 5 |
| 8 | << | 左位移 | x << 2 |
| 8 | >> | 右位移 | x >> 2 |
| 8 | >>> | 右位移（无符号） | x >>> 2 |
| 9 | < | 小于 | x < y |
| 9 | <= | 小于或等于 | x <= y |
| 9 | > | 大于 | x > y |
| 9 | >= | 大于或等于 | x >= y |
| 9 | in | 对象中的属性 | "PI" in Math |
| 9 | instanceof | 对象的实例 | instanceof Array |
| 10 | == | 相等 | x == y |
| 10 | === | 严格相等 | x === y |
| 10 | != | 不相等 | x != y |
| 10 | !== | 严格不相等 | x !== y |
| 11 | & | 按位与 | x & y |
| 12 | ^ | 按位 XOR | x ^ y |
| 13 | | | 按位或 | x | y |
| 14 | && | 逻辑与 | x && y |
| 15 | || | 逻辑否 | x || y |
| 16 | ? : | 条件 | ? "Yes" : "No" |
| 17 | = | 赋值 | x = y |
| 17 | += | 赋值 | x += y |
| 17 | -= | 赋值 | x -= y |
| 17 | \*= | 赋值 | x \*= y |
| 17 | %= | 赋值 | x %= y |
| 17 | <<= | 赋值 | x <<= y |
| 17 | >>= | 赋值 | x >>= y |
| 17 | >>>= | 赋值 | x >>>= y |
| 17 | &= | 赋值 | x &= y |
| 17 | ^= | 赋值 | x ^= y |
| 17 | |= | 赋值 | x |= y |
| 18 | yield | 暂停函数 | yield x |
| 19 | , | 逗号 | 7 , 8 |

# JS数据类型

number、string、boolean、Object、undefined、null、function、Array、Date

在 JavaScript 中有 5 种不同的数据类型：

* string：与C++不同，C++string属于自定义类型，JS string属于内置基础类型之一；
* number
* boolean
* object
* function

3 种对象类型：

* Object
* Date
* Array

2 个不包含任何值的数据类型：

* null
* undefined

## JS原始数据类型

string、number、boolean、undefined

## JS复杂数据类型

function、Object、Date、数组Array和null都解释为Object

## JavaScript 拥有动态类型

这意味着相同变量可用作不同类型：

var x; // 现在 x 是 undefined

x = 7; // 现在 x 是数值

x = "Bill"; // 现在 x 是字符串值

## JavaScript 对象书写

JavaScript 对象用花括号来书写，对象属性是 *name*:*value* 对，由逗号分隔。

var person = {firstName:"Bill", lastName:"Gates", age:62, eyeColor:"blue"};

## 数组类型Array

数组在JS中被解释为对象，typeof 返回 object

## Undefined

任何变量均可通过设置为undefined来清空，清空后其类型也将是undefined。

Person = undefined;

## null

为“noting”，不过在JS中被解释为对象，typeof 也返回object，也可以使用null清空对象,清空后其类型为Object：

var person = null;

常用来主动释放一个变量引用的对象，表示一个变量不再指向任何对象地址。

## undefined与null的区别

typeof undefined // undefined

typeof null // object

null === undefined // false

null == undefined // true

## 使用new

使用new存放在堆中，尽量不要使用new创建对象。它会拖慢执行速度，并可能产生其他副作用。

new出来的数据类型都是对象：

var x = "John";

//返回”String”

typeof x;

var y = new String("John");

//返回”object”

typeof y;

var x = "John";

var y = new String("John");

// 结果为 false，因为 x 是字符串，y 是对象

x === y;

# JS对象

## 对象定义

var person =

{

firstName:"Joh

n",lastName:"Doe",

age:50,

eyeColor:"blue"

};

## 定义方法

var person = {

firstName: "John",

lastName : "Doe",

id : 5566,

//也可以简写为function(){…}

fullName : function()

{

return this.firstName + " " + this.lastName;

}

};

## 访问属性和方法

### 访问属性

person.first;

person[“first”];

### 访问方法

//返回function() { return this.firstName + " " + this.lastName; }

person.fullName;

//返回函数返回值John Doe

person.fullName();

# JS函数

实例：

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>测试实例</title>

<script>

function myFunction()

{

alert("Hello World!");

}

</script>

</head>

<body>

<button onclick="myFunction()">点我</button>

</body>

</html>

## JavaScript 函数语法

函数就是包裹在花括号中的代码块，前面使用了关键词 function：function functionname(argument1,argument2…)  
{

    // 执行代码  
}

<button onclick="myFunction('Harry Potter','Wizard')">点击这里</button>

<script>

function myFunction(name,job)

{

alert("Welcome " + name + ", the " + job);

}

</script>

## 带有返回值的函数

通过使用 return 语句就可以实现，在使用 return 语句时，函数会停止执行，并返回指定的值。

<p id="demo"></p>

<script>

function myFunction(a,b)

{

return a\*b;

}

document.getElementById("demo").innerHTML=myFunction(4,3);

</script>

## 局部 JavaScript 变量

在 JavaScript 函数内部声明的变量（使用 var）是*局部*变量，所以只能在函数内部访问它。（该变量的作用域是局部的）。

您可以在不同的函数中使用名称相同的局部变量，因为只有声明过该变量的函数才能识别出该变量。

只要函数运行完毕，本地变量就会被删除。

## 全局 JavaScript 变量

在函数外声明的变量是*全局*变量，网页上的所有脚本和函数都能访问它。

## JavaScript 变量的生存期

* JavaScript 变量的生命期从它们被声明的时间开始。
* 局部变量会在函数运行以后被删除。
* 全局变量会在页面关闭后被删除。

## 向未声明的 JavaScript 变量分配值

如果您把值赋给尚未声明的变量，该变量将被自动作为 window 的一个属性。

//将声明 window 的一个属性 carname。

carname="Volvo";

非严格模式下给未声明变量赋值创建的全局变量，是全局对象的可配置属性，可以删除delete。

# JS作用域

## JavaScript 局部作用域

变量在函数内声明，变量为局部作用域。

局部变量：只能在函数内部访问。

## JavaScript 全局变量

变量在函数外定义，即为全局变量。全局变量有全局作用域: 网页中所有脚本和函数均可使用。

如果变量在函数内没有声明（没有使用 var 关键字），该变量为全局变量。如果你的变量没有声明，它将自动成为全局变量。

以下实例中 carName 在函数内，但是为全局变量：

<script>

myFunction();

document.getElementById("demo").innerHTML =

"我可以显示 " + carName;

function myFunction()

{

carName = "Volvo";

}

</script>

## JavaScript 变量生命周期

JavaScript 变量生命周期在它声明时初始化。

局部变量在函数执行完毕后销毁。

全局变量在页面关闭后销毁。

## HTML 中的全局变量

在 HTML 中, 所有全局变量都会成为window成员变量。例：

<script>

myFunction();

document.getElementById("demo").innerHTML =

"我可以显示 " + window.carName;

function myFunction()

{

carName = "Volvo";

}

</script>

# JS事件

HTML 事件是发生在 HTML 元素上的事情。

当在 HTML 页面中使用 JavaScript 时， JavaScript 可以触发这些事件。

## HTML 事件

HTML 事件可以是浏览器行为，也可以是用户行为。

以下是 HTML 事件的实例：

* HTML 页面完成加载
* HTML input 字段改变时
* HTML 按钮被点击

通常，当事件发生时，你可以做些事情。

在事件触发时 JavaScript 可以执行一些代码。

HTML 元素中可以添加事件属性，使用 JavaScript 代码来添加 HTML 元素。

单引号:

<*some-HTML-element* *some-event*='JavaScript 代码'>

双引号:

<*some-HTML-element* *some-event*="JavaScript 代码">

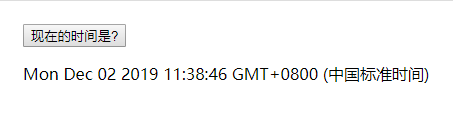
事件修改HTML元素：

<body>

<button onclick="getElementById('demo').innerHTML=Date()">现在的时间是?</button>

<p id="demo"></p>

</body>

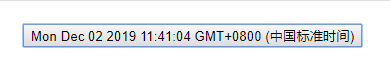


事件修改自身元素，使用this.innerHTML

<body>

<button onclick="this.innerHTML=Date()">现在的时间是?</button>

</body>



JavaScript代码通常是几行代码。比较常见的是通过事件属性来调用JS函数：

<body>

<p>点击按钮执行 <em>displayDate()</em> 函数.</p>

<button onclick="displayDate()">点这里</button>

<script>

function displayDate(){

document.getElementById("demo").innerHTML=Date();

}

</script>

<p id="demo"></p>

</body>

## 常见的HTML事件

下面是一些常见的HTML事件的列表:

| 事件 | 描述 |
| --- | --- |
| onchange | HTML 元素改变 |
| onclick | 用户点击 HTML 元素 |
| onmouseover | 用户在一个HTML元素上移动鼠标 |
| onmouseout | 用户从一个HTML元素上移开鼠标 |
| onkeydown | 用户按下键盘按键 |
| onload | 浏览器已完成页面的加载 |

HTML DOM事件：

<https://www.runoob.com/jsref/dom-obj-event.html>

## JavaScript 可以做什么

* 事件可以用于处理表单验证，用户输入，用户行为及浏览器动作:
* 页面加载时触发事件
* 页面关闭时触发事件
* 用户点击按钮执行动作
* 验证用户输入内容的合法性
* 等等 ...

可以使用多种方法来执行 JavaScript 事件代码：

* HTML 事件属性可以直接执行 JavaScript 代码
* HTML 事件属性可以调用 JavaScript 函数
* 你可以为 HTML 元素指定自己的事件处理程序
* 你可以阻止事件的发生。
* 等等 ...

# JS字符串

## 字符串书写

字符串可用单引号或者双引号，假如引号需要嵌套，外层内层引号不能相同，单引号包含双引号，或者双引号包含单引号，或者使用转义符:”\”

## 转义字符表

| 代码 | 输出 |
| --- | --- |
| \' | 单引号 |
| \" | 双引号 |
| \\ | 反斜杠 |
| \n | 换行 |
| \r | 回车 |
| \t | tab(制表符) |
| \b | 退格符 |
| \f | 换页符 |

## 字符串长度

可以使用内置属性length来计算字符串的长度：

var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

var sln = txt.length;

## 字符串属性和方法

原始值字符串(双引号字符串)，如 "John", 没有属性和方法(因为他们不是对象)。

//str1可以使用String的属性和方法，因为其为String类型对象

var str1 = “John”;

//str2不能使用String的属性和方法，因为其类型为Object

var str2 = new String(“John”);

String类型属性和方法（包括HTML包装方法）：

<https://www.runoob.com/jsref/jsref-obj-string.html>

## 数值和字符串相加

JavaScript将把数值视为字符串，不同的次序产生不同的结果：

var x = 911 + 7 + "Porsche"; //x = 918Porsche

var x = "Porsche" + 911 + 7; //x = Porsche9117

# JS语句

## 条件语句

与C++类似：

* if(){…}
* if(){…}else(…)
* if(){…}else if(){…}…else{…}

## switch

与C++类似。

## for循环

与C++类似，语句1、2、3均可以忽略：

for (语句 1; 语句 2; 语句 3)

{  
    //被执行的代码块

}

## for/In 循环

JavaScript for/in 语句循环遍历对象的属性：

var person={fname:"John",lname:"Doe",age:25};

// x 为属性名

for (x in person)

{

txt=txt + person[x];

}

## while

与C++类似。

## do/while

与C++类似。

## break

与C++类似，额外用法：

JS可以对代码块进行标记，break可通过标记用来跳出该代码块，例：

cars=["BMW","Volvo","Saab","Ford"];

list:

{

document.write(cars[0] + "<br>");

document.write(cars[1] + "<br>");

document.write(cars[2] + "<br>");

//跳出list

break list;

document.write(cars[3] + "<br>");

document.write(cars[4] + "<br>");

document.write(cars[5] + "<br>");

}

## continue

与C++类似。

# JS类型转换

## constructor 属性

**constructor**属性返回所有JavaScript变量的构造函数,类型为function。

//返回function String() { [native code] }

"John".constructor;

//返回function Number(){ [nativecode] }

(3.14).constructor;

//返回function Boolean() { [native code] }

false.constructor;

//返回function Array() { [native code] }

[1,2,3,4].constructor;

//返回function Object() { [native code] }

{name:'John', age:34}.constructor;

//返回function Date() { [native code] }

new Date().constructor;

//返回 function String() { [native code] }

思考：Date()返回字符串，因此相当于“字符串”.constructor

new Date()构造了Date类型对象且返回Date对象

Date().constructor;

//返回function Function() { [native code] }

function () {}.constructor;

可以使用 constructor 属性来查看对象是否为数组 (包含字符串 "Array"),同样此方法也可以查看是否为Date:

function isArray(myArray)

{

//先将构造函数转为String类型，调用String类型方法indexOf()通

//过函数返回下标位置判断构造函数的字符串中是否包含”Array”

return myArray.constructor.toString().indexOf("Array") > -1;

}

## JavaScript 类型转换方式

JavaScript 变量可以转换为新变量或其他数据类型：

* 通过使用 JavaScript 函数，包括全局方法和类成员方法
* 通过 JavaScript 自身自动转换，即隐式转换

## 将数字转换为字符串

* 全局方法**String()**可以将数字转换为字符串：

该方法可用于任何类型的数字，字母，变量，表达式：

String(x)          // 将变量 x 转换为字符串并返回

String(123)        // 将数字 123 转换为字符串并返回

String(100 + 23)   // 将数字表达式转换为字符串并返回

* Number 方法**toString()**也是有同样的效果：

x.toString()

(123).toString()

(100 + 23).toString()

## 将布尔值转换为字符串

* 全局方法**String()**可以将布尔值转换为字符串。

String(false)        // 返回 "false"

String(true)         // 返回 "true"

* Boolean 方法**toString()**也有相同的效果：

false.toString()     // 返回 "false"

true.toString()      // 返回 "true"

## 将日期转换为字符串

* Date() 函数本身返回字符串

//返回Thu Jul 17 2014 15:38:19 GMT+0200 (W. Europe Daylight Time)

Date()

* 全局方法 String() 可以将日期对象转换为字符串。

//返回Thu Jul 17 2014 15:38:19 GMT+0200 (W. Europe Daylight Time)

String(new Date())

* Date 方法**toString()**也有相同的效果。

obj = new Date()

//返回Thu Jul 17 2014 15:38:19 GMT+0200 (W. Europe Daylight Time)

obj.toString()

Date()方法：

<https://www.runoob.com/js/js-type-conversion.html>

## 将字符串转换为数字

全局方法**Number()**可以将字符串转换为数字。

Number("3.14")     // 返回 3.14

Number(" ")       // 返回 0

Number("")       // 返回 0

Number("99 88")    // 返回 NaN

其他全局方法：

| 方法 | 描述 |
| --- | --- |
| parseFloat() | 解析一个字符串，并返回一个浮点数。 |
| parseInt() | 解析一个字符串，并返回一个整数。 |

## 变量使用一元运算符+转为数字

**Operator +**可用于将变量（字符串、日期等）转换为数字，调用Number(),如果变量不能转换，它仍然会是一个数字，但值为 NaN (不是一个数字)：

var y = "5";      // y 是一个字符串

var x = + y;       // x 是一个数字

var y = "John";    // y 是一个字符串

var x = + y;       // x 是一个数字 (NaN)

## 将布尔值转换为数字

全局方法**Number()**可将布尔值转换为数字。

## 将日期转换为数字

全局方法Number()可将日期转换为数字，返回1970年1月1日至今的毫秒数，同Date类方法getTime()。

// Number(Date()) 不行，返回NaN

d = new Date();

Number(d)          // 返回 1404568027739

## 自动转换为字符串

* 当你尝试输出一个对象或一个变量时 JavaScript 会自动调用变量的 toString() 方法，例如：document.getElementById("demo").innerHTML = myVar;时：

myVar = {name:"Fjohn"} // toString 转换为 "[object Object]"

myVar = [1,2,3,4]    // toString 转换为 "1,2,3,4"

myVar = new Date()  // toString 转换为 "Fri Jul 18 2014 09:08:55 GMT+0200"

* 数字和布尔值也经常相互转换：

myVar = 123    // toString 转换为 "123"

myVar = true     // toString 转换为 "true"

myVar = false    // toString 转换为 "false"

## 类型转换表

使用不同的数值转换为数字(Number), 字符串(String),布尔值(Boolean):

| 原始值 | 转换为数字 | 转换为字符串 | 转换为布尔值 |
| --- | --- | --- | --- |
| false | 0 | "false" | false |
| true | 1 | "true" | true |
| 0 | 0 | "0" | false |
| 1 | 1 | "1" | true |
| "0" | 0 | "0" | true |
| "000" | 0 | "000" | true |
| "1" | 1 | "1" | true |
| NaN | NaN | "NaN" | false |
| Infinity | Infinity | "Infinity" | true |
| -Infinity | -Infinity | "-Infinity" | true |
| "" | 0 | "" | false |
| "20" | 20 | "20" | true |
| "Runoob" | NaN | "Runoob" | true |
| [ ] | 0 | "" | true |
| [20] | 20 | "20" | true |
| [10,20] | NaN | "10,20" | true |
| ["Runoob"] | NaN | "Runoob" | true |
| ["Runoob","Google"] | NaN | "Runoob,Google" | true |
| function(){} | NaN | "function(){}" | true |
| { } | NaN | "[object Object]" | true |
| null | 0 | "null" | false |
| undefined | NaN | "undefined" | false |

# JS正则表达式

正则表达式是由一个字符序列形成的搜索模式。

当你在文本中搜索数据时，你可以用搜索模式来描述你要查询的内容。

正则表达式可以是一个简单的字符，或一个更复杂的模式。

正则表达式可用于所有文本搜索和文本替换的操作。

## 语法

关于正则表达式的用法需额外学习，这里只做简单介绍。

/正则表达式主体/修饰符(可选)

实例：

var patt = /runoob/i

实例解析：

/runoob/i 是一个正则表达式。

runoob 是一个正则表达式主体 (用于检索)。

i 是一个修饰符 (搜索不区分大小写)。

## 使用字符串方法

在 JavaScript 中，支持正则表达式的 String 对象的方法: search()、replace()、match()和split()。

### search() 方法

search() 方法 用于检索字符串中指定的子字符串，或检索与正则表达式相匹配的子字符串，并返回子串的起始位置。

#### search() 方法使用正则表达式

var str = "Visit Runoob!";

//输出为6

var n = str.search(/Runoob/i);

#### search() 方法使用字符串

search 方法使用字符串作为参数实际上参数会转换为正则表达式/Runoob/：

var str = "Visit Runoob!";

//输出为6

var n = str.search("Runoob");

### match()方法

match() 方法用于检索字符串中指定的子字符串，或找到一个或多个正则表达式的匹配，返回数组或null。

#### match() 方法使用正则表达式

var str = "The best things in life are free"

//返回e,e,e,e,e,e

document.write(str.match(/e/g));

#### match() 方法使用字符串

同样也会把参数先转换为正则表达式/e/：

var str = "The best things in life are free"

//返回e

document.write(str.match(“e”));

### replace() 方法

replace() 方法 用于在字符串中用一些字符替换另一些字符，或替换一个与正则表达式匹配的子串，并返回替换后的字符串。

#### replace() 方法使用正则表达式

var str = "Visit Microsoft!";

var txt = str.replace(/microsoft/i,"Runoob");

//输出Runoob Microsoft!

document.getElementById("demo").innerHTML = txt;

#### replace() 方法使用字符串

同样也会把参数先转换为正则表达式/ Microsoft/：

var str = "Visit Microsoft!";

var txt = str.replace("Microsoft ","Runoob");

//输出Runoob Microsoft!

document.getElementById("demo").innerHTML = txt;

## 正则表达式修饰符

| 修饰符 | 描述 |
| --- | --- |
| i | 执行对大小写不敏感的匹配。 |
| g | 执行全局匹配（查找所有匹配而非在找到第一个匹配后停止）。 |
| m | 执行多行匹配。 |

## 正则表达式模式

### 表达式[]/()

|  |  |
| --- | --- |
| 表达式 | 描述 |
| [abc] | 查找方括号之间的任何字符。 |
| [0-9] | 查找任何从 0 至 9 的数字。 |
| (x|y) | 查找任何以 | 分隔的选项。 |

### 元字符

|  |  |
| --- | --- |
| 元字符 | 描述 |
| \d | 查找数字。 |
| \s | 查找空白字符。 |
| \b | 匹配单词边界。 |
| \uxxxx | 查找以十六进制数 xxxx 规定的 Unicode 字符。 |

### 量词

|  |  |
| --- | --- |
| 量词 | 描述 |
| n+ | 匹配任何包含至少一个 n 的字符串。 |
| n\* | 匹配任何包含零个或多个 n 的字符串。 |
| n? | 匹配任何包含零个或一个 n 的字符串。 |

## 使用RegExp对象

RegExp对象介绍：

<https://www.runoob.com/jsref/jsref-obj-regexp.html>

### 使用test()成员方法

test() 方法是一个正则表达式方法。

test() 方法用于检测一个字符串是否匹配某个模式，如果字符串中含有匹配的文本，则返回 true，否则返回 false。

以下实例用于搜索字符串中的字符 "e"：

var patt = /e/;

//返回true

//两行可以合并为一行：/e/.test("The best things in life are free!")

patt.test("The best things in life are free!");

### 使用exec()成员方法

exec() 方法是一个正则表达式方法。

exec() 方法用于检索字符串中的正则表达式的匹配。

该函数返回一个数组，其中存放匹配的结果。如果未找到匹配，则返回值为 null。

以下实例用于搜索字符串中的字母 "e":

//返回e，由于返回数组，所以返回的e的类型是object

//注意：与match()相比，返回的数组不是各匹配项子字符串，参见：

// <https://www.cnblogs.com/chunyangji/p/5953418.html>

/e/.exec("The best things in life are free!");

# JS错误

* try 语句测试代码块的错误。
* catch 语句处理错误。
* throw 语句创建自定义错误。
* finally 语句在 try 和 catch 语句之后，无论是否有触发异常，该语句都会执行。

## 基本语法

//异常的抛出

try {

…

throw

}

//异常的捕获与处理

catch(err) {

…

}

finally {

//结束处理

…

}

## throw 语句

throw exception;

异常可以是 JavaScript 字符串、数字、逻辑值或对象。

# 变量提升

* JavaScript 中，函数及变量的声明都将被提升到函数的最顶部。
* JavaScript 中，变量可以在使用后声明，也就是变量可以先使用再声明。
* JavaScript 只有声明的变量会提升，初始化的不会。

# 使用误区

## 比较运算符常见错误

var x = 10;

var y = "10";

// true

if (x == y)

var x = 10;

var y = "10";

// false

if (x === y)

## 加法与连接注意事项

数字与字符串相+，不论前后数字自动转为字符串，同时+从左至右结合律：

10 + “a” + 1 //10a1

10 + 1 + “a” //11a

## JavaScript 字符串分行

字符串断行需要使用反斜杠(\)，例：

var x = "Hello \

World!";

## return 语句使用注意事项

在 JavaScript 中，分号是可选的 。

由于 return 是一个完整的语句，所以 JavaScript 将关闭 return 语句。

function myFunction(a) {

var

power = 10;

// 分号结束，返回 undefined

return;

a \* power;

}

## 数组中使用名字来索引

在 JavaScript 中, 对象 使用 名字作为索引。

如果你使用名字作为索引，当访问数组时，JavaScript 会把数组重新定义为标准对象。

var person = [];

person["firstName"] = "John";

person["lastName"] = "Doe";

person["age"] = 46;

var x = person.length; // person.length 返回 0

var y = person[0]; // person[0] 返回 undefined

person.age //返回46 或person[“age”] = person.age

## 程序块作用域

在每个代码块中 JavaScript 不会创建一个新的作用域，一般各个代码块的作用域都是全局的。

for (var i = 0; i < 10; i++) {

var ii = 1;

}

//代码块外面可以访问 i 和 ii

return ii;