# JavaScript基础入门

Javascript语言诞生主要是完成页面的数据验证。因此它运行在客户端，需要运行浏览器来解析执行JavaScript代码。js是Netscape网景公司的产品，最早取名为LiveScript，为了吸引更多java程序员，更名为javascript。js是弱数据类型，Java是强数据类型。JavaScript只需解析就可以执行，而java需要先编译成字节码文件，再执行。

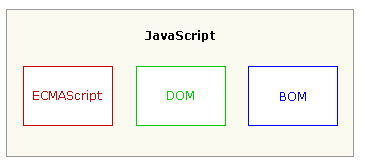
JavaScript特点

交互性（它可以做的就是信息的动态交互）

安全性（不允许直接访问本地硬盘）

跨平台性（只要是可以解释Js的浏览器都可以执行，和平台无关）

**JavaScript组成**



ECMAScript 核心部分 ,定义js的语法规范

DOM: document Object Model 文档对象模型 , 主要是用来管理页面的

BOM: Browser Object Model 浏览器对象模型, 前进,后退,页面刷新, 地址栏, 历史记录, 屏幕宽高

**JavaScript输出**

- alert() 直接弹框

- document.write() 向页面输出

- console.log() 向控制台输出

- innerHTML: 向页面输出

## JavaScript和HTML结合

**通过标签方式**

在HTML文档的head或body标签中使用script标签来写入JavaScript代码

|  |
| --- |
| <head>  <script type="text/javascript">  //使用script标签来定义js代码  alert("success\_head")  </script> </head> |

|  |
| --- |
| <body> <script type="text/javascript">  //使用script标签来定义js代码  alert("success\_body") </script> </body> |

**通过文件方式**

在HTML文档的head标签中使用script标签来导入单独的JavaScript文件，一个script标签只能完成一项功能，不能即用于写入JavaScript代码，又用于导入JavaScript文件。

在HTML页面中，如果一个js文件需要使用到另一个js文件中的内容，则需要先导入另一个js文件。

|  |
| --- |
| <script type="text/javascript" src="Learning.js"></script> |

## JavaScript基础

### 变量

**JavaScript的变量类型**

数值类型： number

字符串类型： string//JavaScript中字符串可以用双引号或者单引号包裹

对象类型： object

布尔类型： boolean（所有非空字符串都是true）

函数类型： function

可以通过typeof运算符查看变量类型（所有引用类型都是Object），通过instanceof运算符解决typeof对象类型判断问题。

**JavaScript的特殊的变量值**

undefined 未定义 所有变量在没有赋初始值的时候，的默认值是undefined

null 空值 访问的对象不存在

NAN 非数字非数值。 not a number

**JavaScript变量范围**

全局变量：在<script>标签中间定义的变量，在该标签内乃至整个页面都有效。

局部变量：在函数体内定义的变量。

**JavaScript定义变量的格式**

var 变量名（ = 值）;

因为JavaScript是若数据类型的编程语言，因此其所有变量的定义都是使用关键字var定义。

let 变量名（ = 值）;

|  |
| --- |
| <script type="text/javascript">  alert("success");  var a;  alert(a); // undefined  a = 12;  //typeof()是js中提供的一个函数，它可以取变量的数据类型  alert(typeof(a) );// number  a = "abc";  alert(typeof (a));// string  var b = 12;  // 当两个变量在做算术运算，但却无法计算，最终就会得到NAN。  var c = b \* a;  alert(c);  alert(b + a); </script> |

### 运算符

#### 关系运算

JavaScript中的关系运算基本和Java相同，只有“==”和“===”不同。“==”表示简单的字面值的比较，而“===”不仅比较字面值，还会比较变量的数据类型。

Javascript中小数和整数都是数字类型，所以除法中没有整数，会出现小数。

字符串与数字相加，是字符串链接，如果相减，字符串直接转换成数字再相减。

Boolean类型可以进行运算，false就是0或者null，非0非空是true，默认用1显示。

|  |
| --- |
| <script type="text/javascript">  var a = "12";  var b = 12;  // js语言在做等于比较的时候，是简单的做字面值的比较。  alert( a == b );//true  // 全等于是除了比较字面值之外，还会比较两个变量的数据类型  alert( a === b );//false </script> |

#### 逻辑运算

且运算 &&

或运算 ||

取反运算 !

在JavaScript中所有的变量都可以转换为逻辑值来使用，其中0、null、undefined和“”（空串）都被认为是false。&&和||也具有短路的特性。

|  |
| --- |
| <script type="text/javascript">  var a = "abc";  var b = true;  var d = false;  var c = null;  // && 与运算两种情况  // 第一种：当表达式全为真的时候。返回最后一个表达式的值。  alert( b && a ); // abc  // 第二种：当表达式中，有一个为假的时候。返回第一个为假的表达式的值  alert( a && d ); // false  // || 或运算两种情况  // 第一种情况：当表达式全为假时，返回最后一个表达式的值  alert( d || c ); // null  // 第二种情况：只要有一个表达式为真。就会把回第一个为真的表达式的值  alert( a || d ); // abc </script> |

### 数组

JavaScript中的数组定义方式为：

var arr = [];表示定义一个空的数组

var arr = [12,“abc”，true];表示定义一个数组的同时赋初始值

var arr = new Array(5);表示定义一个长度为5的数组

var arr = new Array(1,2,4);表示定义数组包含3个元素

JavaScript数组的长度属性为length，其长度是可变的，可以存放不同数据类型的数据。

|  |
| --- |
| <script type="text/javascript">  let arr = [];  alert( arr.length ); // 0 长度  arr[0] = 12;  alert( arr[0] ); // 12;  alert( arr.length ); // 1  arr[2] = "abc";  alert( arr[2] ); // abc  // 在js数组中，只要是通过数组下标给数组元素赋值，则js会自动的根据赋值的最大下标做扩容操作  alert( arr.length ); // 因此此时arr数组的长度扩容为了3  alert( arr[1] );// 因为只给下标为0和2的元素赋值，因此下标为1的元素值为undefined  // 遍历  for (let i = 0 ; i < arr.length; i++) {  alert(arr[i]);  } </script> |

**数组对象的方法**

concat() 连接两个或更多的数组，并返回结果。

join() 把数组的所有元素放入一个字符串。元素通过指定的分隔符进行分隔。

pop() 删除并返回数组的最后一个元素

push() 向数组的末尾添加一个或更多元素，并返回新的长度。

sort() 对数组的元素进行排序

### 常用对象

#### Date对象

创建Date对象：var date = new Date()

**常用方法**

getDate() 从 Date 对象返回一个月中的某一天 (1 ~ 31)

getDay() 从 Date 对象返回一周中的某一天 (0 ~ 6)

getMonth() 从 Date 对象返回月份 (0 ~ 11)

getFullYear() 从 Date 对象以四位数字返回年份

getTime() 返回 1970 年 1 月 1 日至今的毫秒数

parse() 返回1970年1月1日午夜到指定日期（字符串）的毫秒数

toLocaleString() 根据本地时间格式，把 Date 对象转换为字符串。

toLocaleDateString() 根据本地时间格式，把 Date 对象的日期部分转换为字符串

#### Math对象

Math对象中全部是静态方法，所以不用new就可以直接调用。

**常用方法**

ceil(x) 对一个数进行上舍入。

floor(x) 对一个数进行下舍入。

round(x) 把一个数四舍五入为最接近的整数

pow(x,y) 返回 x 的 y 次幂

random() 返回 0 ~ 1 之间的随机数

#### RegExp对象

正则表达式对象的写法有如下两种

(1)var reg = new RegExp(“[0-9]{5}”);

(2)var reg = /^[0-9]{5}$/;

常用方法

exec() 如果采用(1)方式，字符串中包含匹配的内容会返回结果。如果采用(2)方式会返回null

test() 如果采用(1)方式，字符串中包含匹配的结果会返回true。如果采用(2)方式会返回false

### 函数

如果要定义带有返回值的函数的话，只需要在函数体内直接使用return语句返回即可。对于JavaScript中函数的调用根据定义格式的不同而略微不同，分别为：函数名（参数列表）或变量名（参数列表）。

JavaScript中不存在重载形式；每个函数中都存在一个数组arguments用于存储参数列表；调用方法时如果没有写()，则会把函数对象的引用传递给接收的变量，打印出函数体的内容。

#### 函数定义方式

**普通方式定义函数**

function 函数名（参数列表）{

函数体;

}

|  |
| --- |
| <script type="text/javascript">  // 定义一个无参函数  function fun1(){  alert("fun1被调用了");  }  fun1();  // 定义有参函数  function fun2( a , b ){  alert("fun2函数被调用了。a=" + a + ",b="+b);  }  fun2(12,"abc");  // 带有返回值的函数定义  function sum(num1,num2){  let result = num1 + num2;  return result;  }  alert( sum(49,51) ); </script> |

**动态函数**

动态函数是通过js的内置对象Function，通过new Function(参数1，参数2)来创建动态函数。

|  |
| --- |
| <script type="text/javascript">  var add = new Function("x,y","" +  "var sum;" +  "sum=x\*y;" +  "return sum;")  var sum = add(4,5);  console.log(sum); </script> |

**匿名函数**

var 变量名 = function（参数列表）{

函数体;

}

|  |
| --- |
| <script type="text/javascript">  var fun1 = function(a){  alert("fun1被调用了. a => " + a);  return a;  }  alert( fun1("abc") ); </script> |

**函数的隐形参数**

函数中的隐形参数arguments不需要定义，但是可以之间使用，其使用方式和数组相同。

|  |
| --- |
| <script type="text/javascript">  function fun(a){  // 隐形参数，可以直接用来获取全部实参的值。  alert( arguments.length );  alert( arguments[0] );  alert( arguments[1] );  alert( arguments[2] );  alert("函数fun被调用了。a=>" + a);  }  fun("abc",true,12);  // 写一个函数计算传入的所有number类型的参数的值。相加的和并返回  function sum(num1,num2) {  var result = 0;  for (var i = 0; i < arguments.length; i++) {  // 检查是number类型才相加  if ( typeof(arguments[i]) == "number" ) {  result += arguments[i];  }  }  return result;  }  alert( sum(10,30,"abc",50,100,60) ); </script> |

#### 常用函数

##### global全局函数

encodeURI / decodeURI 编解码URI，进行url跳转时可以整体使用encodeURI

encodeURIComponent / decodeURIComponent 编解码URI组件，传递参数时需要使用encodeURIComponent

escape / unescape 对字符串进行unicode编码

escape不编码字符有69个：\*，+，-，.，/，@，\_，0-9，a-z，A-Z

encodeURI不编码字符有82个：!，#，$，&，'，(，)，\*，+，,，-，.，/，:，;，=，?，@，\_，~，0-9，a-z，A-Z

encodeURIComponent不编码字符有71个：!， '，(，)，\*，-，.，\_，~，0-9，a-z，A-Z

isNaN 检查某个值是否为数字

parseInt / parseFloat 解析字符串为整数 / 浮点数

eval 计算javascript 字符串，并把它作为脚本代码来执行

### 自定义对象

对象实例的访问

变量名.属性名/函数名()；

#### Object形式的自定义对象

**格式**

var 变量名 = new Object(); //创建一个对象实例

变量名.属性名1 = 值; //添加一个属性

变量名.属性名1 = 值; //添加另一个属性

变量名.函数名 = function(){}; //添加一个函数

|  |
| --- |
| <script type="text/javascript">  var obj = new Object();  obj.name = '华仔';  obj.age = 18;  obj.fun = function(){  alert( "姓名：" + this.name + " , 年龄：" + this.age );  }  obj.fun(); </script> |

#### {}形式的自定义对象

var 变量名 = { //创建空对象实例

属性名1:值, //添加一个属性，多个属性中间用逗号隔开

属性名2:值, //添加另一个属性

函数名:function(){}; //添加一个方法

}

|  |
| --- |
| <script type="text/javascript">  let obj = {  name : "国哥",  age : 18,  fun : function(){  alert("姓名：" + this.name + " , 年龄：" + this.age);  }  };  obj.fun(); </script> |

### 定时器

- setInterval : 每隔多少毫秒执行一次函数，直到clearInterval()被调用或窗口被关闭

- setTimeout: 多少毫秒之后执行一次函数，只执行一次；如果要多次执行则可以调用自身

- clearInterval

- clearTimeout

## JavaScript事件

事件就是电脑输入设备与页面中元素进行交互的响应。

### 事件的注册

#### 静态注册

静态注册事件是指通过标签的事件属性直接赋于事件响应后的代码，如果同时由静态注册和动态注册，则以静态注册为优先，而不执行动态注册。

#### 动态注册

动态注册事件是指，先通过js代码获取标签对象。然后再通过标签对象.事件名=function(){}。

window.onload = function(){

// 1 获取标签对象

// 2 通过标签对象.事件名 = function(){}

}

### 事件的种类

#### onload加载完成事件

页面加载完成之后可以做一些初始化工作

|  |
| --- |
| <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Insert title here</title>  <script type="text/javascript">  function onloadFun(){  alert("静态注册的onload函数");  }  // 动态注册onload事件  window.onload = function(){  alert("动态注册onload事件");  onloadFun();  }  </script> </head> <!-- 静态注册onload事件<body onload="onloadFun()"> --> <body> 页面出来了 </body> |

#### onunload

#### onclick单击事件

常用于按钮的点击响应效果

|  |
| --- |
| <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Insert title here</title>  <script type="text/javascript">  function onclickFun() {  alert("静态注册onclick事件");  }  // 动态注册第二个按钮的onclick事件  window.onload = function () {  //1 通过代码获取标签对象  /\* document是js语言中提供的一个现成的对象。  document表示整个html文档  getElementById() 通过id获取标签对象 \*/  let btnObj = document.getElementById("btn01");  // alert(btnObj); // [object HTMLButtonElement]  // 2 通过标签对象.事件名 = function(){}  btnObj.onclick = function () {  alert("动态注册onclick事件");  }  }  </script> </head> <body>  <!--静态注册onclick事件-->  <button onclick="onclickFun()">按钮1</button>  <!--动态注册onclick事件-->  <button id="btn01">按钮2</button> </body> |

#### ondblclick

#### onblur失去焦点事件

常用于输入框失去焦点后验证其内容是否合法。

|  |
| --- |
| <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Insert title here</title>  <script type="text/javascript">  function onblurFun(){  // console 是javaScript语言提供的一个叫控制台对象  // log()往浏览器的控制台打印输出  console.log("静态注册onblur事件");  }  // 动态注册onblur事件  window.onload = function(){  //1 通过js代码获取标签对象  let passObj = document.getElementById("password");  //2 通过标签对象.事件名 = function(){}  passObj.onblur = function(){  console.log("动态注册onblur事件");  }  }   </script> </head> <body> <!-- 静态注册失去焦点事件onblur --> 用户名：<input type="text" onblur="onblurFun()"/><br/> <!-- 动态注册失去焦点事件onblur--> 密码：<input id="password" type="password"/><br/> </body> |

#### onfocus获得焦点事件

#### onchange内容发生改变事件

常用于输入框和下拉列表中内容发生改变时响应。

|  |
| --- |
| <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Insert title here</title>  <script type="text/javascript">  function onchangeFun() {  console.log("静态注册onchange事件");  }  window.onload = function () {  var selObj = document.getElementById("sel01");  selObj.onchange = function () {  console.log("动态注册onchange事件");  }  }  </script> </head> <body> 请选择你心中的女神： <select onchange="onchangeFun()">  <option>--选择女神--</option>  <option>苍老师</option>  <option>柳岩</option>  <option>迪丽热巴</option> </select><br/> 请选择你心中的男神： <select id="sel01">  <option>--选择男神--</option>  <option>华仔</option>  <option>国哥</option>  <option>飞龙哥</option> </select> </body> |

#### onselect

#### onsubmit表单提交事件

常用于表单提交前验证是否有表单项不合法的情况，如果有就阻止表单提交。

|  |
| --- |
| <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Insert title here</title>  <script type="text/javascript">  function onsubmitFun() {  alert("静态注册onsubmit事件====并验证表单项。发现有不合法");  return false;  }  window.onload = function () { let formObj = document.getElementById("form01");  formObj.onsubmit = function () {  alert("动态注册onsubmit事件 === 开始检查表单，发现有表单项不合法，阻止提交");  return false;  }  }  </script> </head> <body> <form action="http://www.baidu.com" onsubmit="return onsubmitFun();">  <input type="submit" value="静态onsubmit"/> </form> <form id="form01" action="http://www.baidu.com">  <input type="submit" value="动态onsubmit"/> </form> </body> |

#### onreset

#### onmousemove

#### onmouseout

#### onmouseover

#### onmousedown

#### onmouseup

#### onkeydown

#### onkeyup

#### onkeypress

## Bom

### Window对象

Window 对象表示浏览器中打开的窗口，如果文档包含框架（frame 或 iframe 标签），浏览器会为 HTML 文档创建一个 window 对象，并为每个框架创建一个额外的 window 对象。

window.frames 返回窗口中所有命名的框架

parent是父窗口（如果窗口是顶级窗口，那么parent==self==top）

top是最顶级父窗口（有的窗口中套了好几层frameset或者iframe）

self是当前窗口（等价window）

opener是用open方法打开当前窗口的那个窗口

**常用方法**

Js window.showModalDialog()方的使用简介

      a.url: 弹出窗口指向哪一个页面  
      b.varArgIn: 向弹出页面传递的参数  
      写法如下:  
         window.showModalDialog("openner3.htm",tableObj);  
2.在父页面中,如何制定所需传递的常数  
      在Js中,定义一个类是很简单的(类的写法,不过感觉像是个HashMap),定义方法如下  
      //定义一个超级对象  
      var tableObj = new Object();  
      //下面的写法就是自定义属性了,"tableObj."后面的属性名可以随便写,感觉有点像key值  
      //"="后面的是属性值,也是什么都行,感觉有点像value值  
      tableObj.table = window.document.getElementById("myTable");  
      tableObj.userName = window.document.getElementById("txtUserName").value;  
      //模拟调用一下  
      var tempObj = tableObj;  
      //在这里,"."后的属性名与之前对象定义的属性名一直就Ok了,感觉像通过key值取value值  
      //"tempObj.table.tagName"之所以这样写是因为"tempObj.table"是指向了id为"myTable"的一个Table对象,所以可以调用这个对象的"tagName"属性来查看其是什么元素  
      alert(tempObj.table.tagName);  
      alert(tempObj.userName);  
3.在子页面中如何接收来自父页面的参数  
      调用下面的方法即可:  
      var comeFromFatherPageObj = window.dialogArguments;  
4.子页面向父页面传递返回值(当子页面即将关闭时)  
      写法如下:  
      //首先在父页面调用window.showModalDialog()方法的时候做些操作,如下:  
      objReturnValue = window.showModalDialog("openner3.htm",tableObj);  
      //然后再子页面,即将关闭的时候加一句话,如下:  
      window.returnValue = "asdad";//可以是任何东西  
      window.close();

window.alert(String)

Boolean window.confirm(String)

String window.prompt(String)

## Dom

### 概述

DOM 全称是**Document Object Model**文档对象模型，可以将文档中的标签，属性，文本，转换成为对象来管理。

**文档**：标记型文档

**对象**：封装了属性和行为的实例，可以直接被调用。

**模型**：所有的标记型文档都具有一些共性特征的一个体现。

用来将标记型文档封装成对象，并将标记型文档中的所有内容（标签、文本、属性）都封装成对象。封装成对象的目的是为了更方便的操作这些文档及其文档中的所有内容。因为对象包含属性和行为。标记型文档包含标签、属性、标签中封装的数据。只要是标记型文档，DOM这种技术都可以对其进行操作。常见的标记型文档包括：HTML XML。

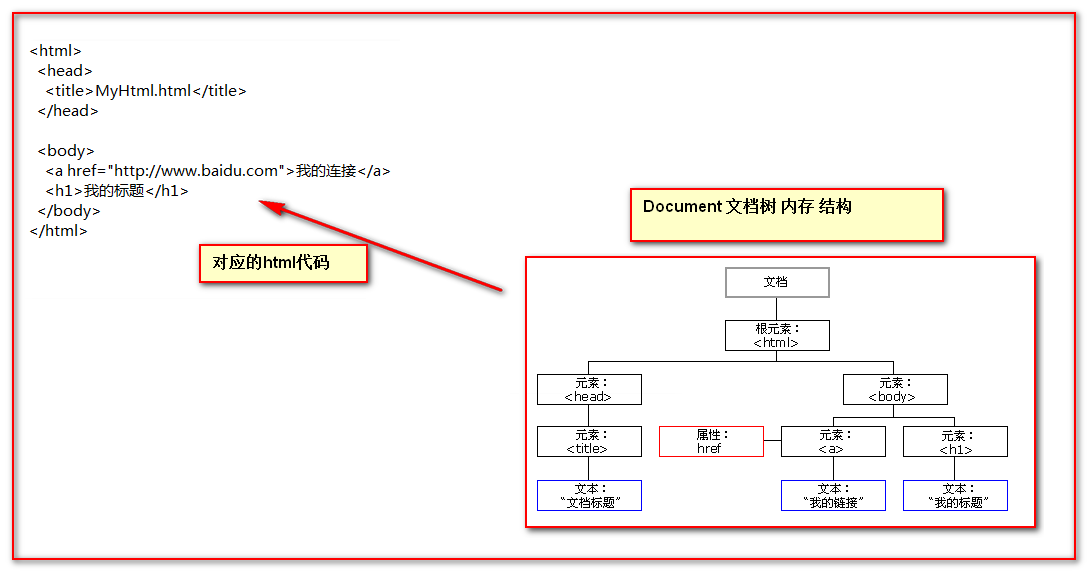
DOM要操作标记型文档必须先进行解析。

**DOM模型有三种：**

DOM level 1：将html文档封装成对象。

DOM level 2：在level 1的基础上添加新的功能，例如：对于事件和css样式的支持。

DOM level 3：支持xml1.0的一些新特性。



**DHTML**

动态的HTML，它不是一门语言，是多项技术综合体的简称。包括html，css，dom，javascript。

这四种语言的职责：

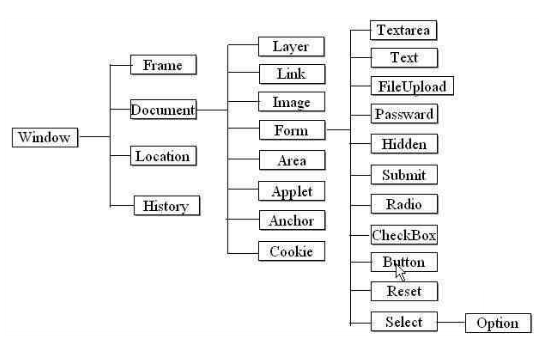
Html：负责提供标签，封装数据，这样便于操作数据。

Css：负责提供样式，对标签中的数据进行样式定义。

Dom：负责将标签及其内容解析，封装成对象，对象中具有属性和行为。

Javascript：负责提供程序设计语言，对页面中的对象进行逻辑操作。

#### Bom和HTML DOM关系图



### Dom对象

#### Document对象

Document管理了所有的HTML文档内容；是一种树结构的文档，有层级关系；会将所有的标签对象化；可以通过document访问所有的标签对象。每个载入浏览器的 HTML 文档都会成为 Document 对象。

**常用属性**

all[] 提供对文档中所有 HTML 元素的访问 FF不支持

forms[]返回对文档中所有 Form 对象引用

body 提供对 <body> 元素的直接访问

innerHTML - 节点（元素）的文本值

parentNode - 节点（元素）的父节点

childNodes - 节点（元素）的子节点

attributes - 节点（元素）的属性节点

**常用方法**

getElementById()返回对拥有指定 id 的第一个对象的引用

getElementsByName()返回带有指定名称的对象集合

getElementsByTagName()返回带有指定标签名的对象集合

write()向文档写 HTML 表达式 或 JavaScript 代码

#### Element对象

**操纵 Element 对象的属性**

获取属性：getAttitude(name)方法。

设置属性：setAttribute(name,value)方法。

删除属性：removeAttribute(name)方法。

**在 Element 对象中查找 Element 对象**

在Element对象的范围内，可以用来查找其他节点的唯一有效方法就是getElementsByTagName()方法。而该方法返回的是一个集合

### Dom对象的方法

doucment.对象三个查询方法，使用顺序如下：

有id的时候，优先使用id查询

没有id的时候，优先使用name查询。

即没id，又没name最后使用标签名查询。

以上三个查询方法，都必须要在页面加载完成之后执行才有效。

#### 通过id

**document.getElementById(elementId)**通过标签的id属性查找标签dom对象，elementId是标签的id属性值。

|  |
| --- |
| <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Insert title here</title>  <script type="text/javascript">  // 当用户点击【验证】按钮之后，获取输入框里的文本内容。验证其是否合法。  // 验证的规则如下：用户名必须由字母，数字。下划线组成,并且长度是5到12位。  function checkUsername() {  // 要操作一个标签，要先得到这个标签对象  var usernameObj = document.getElementById("username");  // 获取用户名输入框中的内容  var usernameText = usernameObj.value;  // 检查用户名是否合法。需要使用到正则表达式。  var patt = /^\w{5,12}$/;  // 正则表达式对象有一个方法叫test(string ) ：boolean  // 返回true，说明合法。返回false，说明不合法。  var spanObj = document.getElementById("usernameSpan");  // innerHTML表示起始标签和结束标签中的内容  // 这个属性可读，可写  if (patt.test(usernameText)) {  spanObj.innerHTML = "";  alert("用户名合法"); // spanObj.innerHTML = '<img alt="" src="right.png" width="15" height="15"/>';   } else {  spanObj.innerHTML = "用户名不合法";  alert("用户名不合法"); // spanObj.innerHTML = '<img alt="" src="wrong.png" width="15" height="15"/>';  }  }  </script> </head> <body>  用户名：<input type="text" id="username" value="wzg"/>  <span id="usernameSpan" style="color: red;"></span><br/>  <button onclick="checkUsername()">验证</button> </body> |

#### 通过name

**document.getElementsByName(elementName)**通过标签的name属性查找标签dom对象，elementName标签的name属性值。

|  |
| --- |
| <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Insert title here</title>  <script type="text/javascript">  function checkAll() {  // getElementsByName 通过name属性值查找标签对象集合返回  // 查找返回的集合操作跟数组一样  // 集合中的元素顺序刚好是这些标签在页面中从上到下的顺序  var hobbies = document.getElementsByName("hobby");  // 复选框标签对象。checked属性表示当前的选中状态  // true表示选中，false表示不选中  // checked属性可读，可写  for (var i = 0; i < hobbies.length; i++) {  hobbies[i].checked = true;  }  }  function checkNo() {  var hobbies = document.getElementsByName("hobby");  for (var i = 0; i < hobbies.length; i++) {  hobbies[i].checked = false;  }  }  function checkReverse() {  var hobbies = document.getElementsByName("hobby");  for (var i = 0; i < hobbies.length; i++) { // if ( hobbies[i].checked ) { // hobbies[i].checked = false; // } else { // hobbies[i].checked = true; // }  hobbies[i].checked = !hobbies[i].checked;  }  }  </script> </head> <body>兴趣爱好：<br/>  <input type="checkbox" name="hobby" value="java" checked="checked"/>java  <input type="checkbox" name="hobby" value="cpp"/>C++  <input type="checkbox" name="hobby" value="php"/>php  <input type="checkbox" name="hobby" value="python"/>python<br/>  <button onclick="checkAll()">全选</button>  <button onclick="checkNo()">全不选</button>  <button onclick="checkReverse()">反选</button> </body> |

#### 通过tagName

**document.getElementsByTagName(tagname)**通过标签名查找标签dom对象。tagname是标签名。

|  |
| --- |
| <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Insert title here</title>  <script type="text/javascript">  // 全选  function checkAll(){  // getElementsByTagName方法是根据标签名来查找标签对象集合返回  // 这个集合操作跟数组一样  // 这个集合中元素的顺序刚好是标签在html页面中从上到下的顺序  var inputs = document.getElementsByTagName("input");  for (var i = 0; i < inputs.length; i++) {  inputs[i].checked = true;  }  }  </script> </head> <body>兴趣爱好：<br/> <input type="checkbox" value="java" checked="checked"/>java <input type="checkbox" value="cpp"/>C++ <input type="checkbox" value="php"/>php <input type="checkbox" value="python"/>python<br/> <button onclick="checkAll()">全选</button> </body> |

#### 创建标签对象

**document.createElement(tagName)**方法，通过给定的标签名，创建一个标签对象。tagName是要创建的标签名。如果同时调用了innerHTML和innerText属性添加赋值，则后添加的值会覆盖掉先添加的值。

因为body标签经常被使用，因此JavaScript单独为其创建了对象，然后直接调用即可。

|  |
| --- |
| <script type="text/javascript">  // 需要通过代码方式手动创建一个标签对象：<div>国哥好帅！</div>。  // 然后把这个标签添加到窗口中显示  window.onload = function () {  // createElement 方法创建一个标签对象  var divObj = document.createElement("div"); // <div></div>  // 往标签中间加内容 // divObj.innerHTML = "国哥好帅！"; // <div>国哥好帅！</div>  // 通过代码把一个字符串创建成为节点对象  var textNode = document.createTextNode("国哥好帅！");  divObj.appendChild(textNode); //<div>国哥好帅！</div>  // 把div标签添加到body标签中  document.body.appendChild(divObj);  }  </script> |

### Node节点的属性和方法

节点可以简单的理解为标签对象

|  |  |  |
| --- | --- | --- |
| nodeName | String | 节点的名字；根据节点的类型而定义  如果节点是元素节点，nodeName返回这个元素的名称  如果是属性节点，nodeName返回这个属性的名称  如果是文本节点，nodeName返回一个内容为#text 的字符串 |
| nodeValue | String | 节点的值；根据节点的类型而定义  如果给定节点是一个属性节点，返回值是这个属性的值  如果给定节点是一个文本节点，返回值是这个文本节点内容  如果给定节点是一个元素节点，返回值是 null |
| nodeType | Number | 节点的类型常量值之一  Node.ELEMENT\_NODE ---1 -- 元素节点  Node.ATTRIBUTE\_NODE ---2 -- 属性节点  Node.TEXT\_NODE ---3 -- 文本节点 |
| ownerDocument | Document | 指向这个节点所属的文档 |
| firstChild | Node | 指向在childNodes列表中的第一个节点 |
| lastChild | Node | 指向在childNodes列表中的最后一个节点 |
| childNodes | NodeList | 所有子节点的列表 |
| parentNode | Node | 返回一个给定节点的父节点。  parentNode 属性返回的节点永远是一个元素节点，因为只有元素节点才有可能包含子节点。  document 节点的没有父节点。 |
| previousSibling | Node | 指向前一个兄弟节点；如果这个节点就是第一个兄弟节  点，那么该值为null |
| nextSibling | Node | 指向后一个兄弟节点；如果这个节点就是最后一个兄弟节  点，那么该值为null |
| **hasChildNodes**() | Boolean | 当childNodes包含一个或多个节点时，返回真 |
| attributes | NamedNodeMap | 包含了代表一个元素的特性的Attr对象；仅用于Element节点 |
| **appendChild**(*node*) | Node | 将*node*添加到childNodes的末尾 |
| **removeChild**(*node*) | Node | 从childNodes中删除*node* |
| **replaceChild** (*newnode*, *oldnode*) | Node | 将childNodes中的*oldnode*替换成*newnode* |
| **insertBefore** (newnode, refnode) | Node | 在childNodes中的refnode之前插入newnode |
| **cloneNode**(boolean) |  | 参数boolean表示判断是否复制子节点 |

#### 属性

**className**

用于获取或设置标签的class属性值

**innerHTML**

属性，表示获取/设置起始标签和结束标签中的内容，可以用来读，写某给定元素里的 HTML 内容。多与div或span标签配合使用

**innerText**

属性，表示获取/设置起始标签和结束标签中的文本

|  |
| --- |
| <head>  <meta charset="UTF-8">  <title>dom查询</title>  <link rel="stylesheet" type="text/css" href="style/css.css" />  <script type="text/javascript">  window.onload = function(){  //1.查找#bj节点  document.getElementById("btn01").onclick = function(){  console.log(document.getElementById("bj").innerText);  };  //2.查找所有li节点  let btn02Ele = document.getElementById("btn02");  btn02Ele.onclick = function(){  let arr = document.getElementsByTagName("li");  for (let i = 0; i < arr.length; i++) {  console.log(arr[i].innerText);  }  };  //3.查找name=gender的所有节点  let btn03Ele = document.getElementById("btn03");  btn03Ele.onclick = function(){  let arr = document.getElementsByName("gender");  for (let i = 0; i < arr.length; i++) {  console.log(arr[i].value);  }  };  //4.查找#city下所有li节点  let btn04Ele = document.getElementById("btn04");  btn04Ele.onclick = function(){  //1 先查找city标签对象  let city = document.getElementById("city");  //2 查找它之下的li子节点  let lis = city.getElementsByTagName("li");  for (let i = 0; i < lis.length; i++) {  console.log(lis[i].innerHTML);  }  };  //5.返回#city的所有子节点  let btn05Ele = document.getElementById("btn05");  btn05Ele.onclick = function(){  //1 先查找city标签对象  let city = document.getElementById("city");  //2 通过childNodes属性获得  let lis = city.childNodes;  for (let i = 0; i < lis.length; i++) {  console.log(lis[i].innerHTML);  }  };  //6.返回#phone的第一个子节点  let btn06Ele = document.getElementById("btn06");  btn06Ele.onclick = function(){  let firstChild = document.getElementById("phone").firstChild;  console.log(firstChild.innerHTML);  };  //7.返回#bj的父节点  let btn07Ele = document.getElementById("btn07");  btn07Ele.onclick = function(){  let parent = document.getElementById("bj").parentNode;  console.log(parent)  };  //8.返回#android的前一个兄弟节点  let btn08Ele = document.getElementById("btn08");  btn08Ele.onclick = function(){  let broNode = document.getElementById("android").previousSibling;  console.log(broNode.innerHTML);  };  //9.读取#username的value属性值  let btn09Ele = document.getElementById("btn09");  btn09Ele.onclick = function(){  let username = document.getElementById("username").value;  console.log(username);  };  //10.设置#username的value属性值  let btn10Ele = document.getElementById("btn10");  btn10Ele.onclick = function(){  let username = document.getElementById("username").value;  console.log(username)  };  //11.返回#bj的文本值  let btn11Ele = document.getElementById("btn11");  btn11Ele.onclick = function(){  alert( document.getElementById("city").innerHTML );//包含标签  alert( document.getElementById("city").innerText );//不包含标签  };  }; </script>  </head> <body> <div id="total">  <div class="inner">  <p>  你喜欢哪个城市?  </p>  <ul id="city">  <li id="bj">北京</li>  <li>上海</li>  <li>东京</li>  <li>首尔</li>  </ul>  <br>  <br>  <p>  你喜欢哪款单机游戏?  </p>  <ul id="game">  <li id="rl">红警</li>  <li>实况</li>  <li>极品飞车</li>  <li>魔兽</li>  </ul>  <br />  <br />  <p>  你手机的操作系统是?  </p>  <ul id="phone"><li>IOS</li><li id="android">Android</li><li>Windows Phone</li></ul>  </div>  <div class="inner">  gender:  <input type="radio" name="gender" value="male"/>  Male  <input type="radio" name="gender" value="female"/>  Female  <br>  <br>  name:  <input type="text" name="name" id="username" value="abcde"/>  </div> </div> <div id="btnList">  <div><button id="btn01">查找#bj节点</button></div>  <div><button id="btn02">查找所有li节点</button></div>  <div><button id="btn03">查找name=gender的所有节点</button></div>  <div><button id="btn04">查找#city下所有li节点</button></div>  <div><button id="btn05">返回#city的所有子节点</button></div>  <div><button id="btn06">返回#phone的第一个子节点</button></div>  <div><button id="btn07">返回#bj的父节点</button></div>  <div><button id="btn08">返回#android的前一个兄弟节点</button></div>  <div><button id="btn09">返回#username的value属性值</button></div>  <div><button id="btn10">设置#username的value属性值</button></div>  <div><button id="btn11">返回#bj的文本值</button></div> </div> </body>  /\*导入的css文件\*/  @CHARSET "UTF-8"; body {  width: 800px;  margin-left: auto;  margin-right: auto; } button {  width: 300px;  margin-bottom: 10px; } #btnList {  float: left; } #total {  width: 450px;  float: left; } ul {  list-style-type: none;  margin: 0px;  padding: 0px; } .inner li {  border-style: solid;  border-width: 1px;  padding: 5px;  margin: 5px;  background-color: #99ff99;  float: left; } .inner {  width: 400px;  border-style: solid;  border-width: 1px;  margin-bottom: 10px;  padding: 10px;  float: left; } |

# ECMAScript6概述

ECMAScript 6.0（以下简称 ES6）是 JavaScript 语言的下一代标准，已经在 2015 年 6 月正式发布了。它的目标，是使得 JavaScript 语言可以用来编写复杂的大型应用程序，成为企业级开发语言。

**ECMAScript 和 JavaScript 的关系**

1996 年 11 月，JavaScript 的创造者 Netscape 公司，将 JavaScript 提交给标准化组织 ECMA，希望这种语言能够成为国际标准。次年，ECMA 发布 262 号标准文件（ECMA-262）的第一版，规定了浏览器脚本语言的标准，并将这种语言称为 ECMAScript，这个版本就是 1.0 版。

因此，ECMAScript 和 JavaScript 的关系是，前者是后者的规格，后者是前者的一种实现（另外的 ECMAScript 方言还有 Jscript 和 ActionScript）

**ES6 与 ECMAScript 2015 的关系**

2011 年，ECMAScript 5.1 版发布后，就开始制定 6.0 版了。因此，ES6 这个词的原意，就是指 JavaScript 语言的下一个版本。

ES6 的第一个版本，在 2015 年 6 月发布，正式名称是《ECMAScript 2015 标准》（简称 ES2015）。

2016 年 6 月，小幅修订的《ECMAScript 2016 标准》（简称 ES2016）如期发布，这个版本可以看作是 ES6.1 版，因为两者的差异非常小，基本上是同一个标准。根据计划，2017 年 6 月发布 ES2017 标准。

因此，ES6 既是一个历史名词，也是一个泛指，含义是 5.1 版以后的 JavaScript 的下一代标准，涵盖了 ES2015、ES2016、ES2017 等等，而 ES2015 则是正式名称，特指该年发布的正式版本的语言标准。本书中提到 ES6 的地方，一般是指 ES2015 标准，但有时也是泛指“下一代 JavaScript 语言”。

# ECMAScript6基本语法

ES标准中不包含 DOM 和 BOM的定义，只涵盖基本数据类型、关键字、语句、运算符、内建对象、内建函数等通用语法。

## let声明变量

|  |
| --- |
| {  var a = 0;// var 声明的变量没有局部作用域  let b = 1;// let 声明的变量有局部作用域  }  console.log(a);// 0  console.log(b);// ReferenceError: b is not defined |

|  |
| --- |
| var m = 1 // var 可以声明多次  var m = 2 // let 只能声明一次  let n = 3  let n = 4  console.log(m) // 2  console.log(n) // Identifier 'n' has already been declared |

|  |
| --- |
| console.log(x) //undefined  var x = 'apple' // var 会变量提升  console.log(y) //ReferenceError: y is not defined  let y = 'banana'// let 不存在变量提升 |

## const声明变量（只读变量）

|  |
| --- |
| const PI = '3.1415926' // const声明之后不允许改变  PI = 3 // TypeError: Assignment to constant variable  // 一但声明必须初始化，否则会报错  const MY\_AGE // SyntaxError: Missing initializer in const declaration |

## 解构赋值

解构赋值是对赋值运算符的扩展，一种针对数组或者对象进行模式匹配，然后对其中的变量进行赋值。在代码书写上简洁且易读，语义更加清晰明了；也方便了复杂对象中数据字段获取。

|  |
| --- |
| // 传统数组解构  let a = 1, b = 2, c = 3  console.log(a, b, c)  // ES6数组解构  let [x, y, z] = [1, 2, 3]  console.log(x, y, z) |

|  |
| --- |
| let user = {name: 'Helen', age: 18}  // 传统对象解构  let name1 = user.name  let age1 = user.age  console.log(name1, age1)  // ES6对象解构  let { name, age } = user//注意：解构的变量必须和user中的属性同名  console.log(name, age) |

## 模板字符串

模板字符串相当于加强版的字符串，用反引号 `,除了作为普通字符串，还可以用来定义多行字符串，还可以在字符串中加入变量和表达式。

|  |
| --- |
| // 字符串插入变量和表达式。变量名写在 ${} 中，${} 中可以放入 JavaScript 表达式。  let name = 'Mike'  let age = 27  let info = `My Name is ${name},I am ${age+1} years old next year.`  console.log(info)// My Name is Mike,I am 28 years old next year. |

## 声明对象简写

|  |
| --- |
| const age = 12  const name = 'Amy'  // 传统声明对象  const person1 = {age: age, name: name}  console.log(person1)  // ES6声明对象  const person2 = {age, name}  console.log(person2) //{age: 12, name: 'Amy'} |

## 定义方法简写

|  |
| --- |
| // 传统定义方法  const person1 = {  sayHi:function(){  console.log('Hi')  }  }  person1.sayHi();//'Hi'  // ES6定义方法  const person2 = {  sayHi(){  console.log('Hi')  }  }  person2.sayHi() //'Hi' |

## 对象扩展运算符

拓展运算符（...）用于取出参数对象所有可遍历属性然后拷贝到当前对象。

|  |
| --- |
| let person = {name: 'Amy', age: 15}  // let someone = person //引用赋值  let someone = { ...person } //对拷拷贝  someone.name = 'Helen'  console.log(person) //{name: 'Amy', age: 15}  console.log(someone) //{name: 'Helen', age: 15} |

## 函数的默认参数

|  |
| --- |
| function showInfo(name, age = 17) {  console.log(name + "," + age)  }  // 只有在未传递参数，或者参数为 undefined 时，才会使用默认参数  // null 值被认为是有效的值传递。  showInfo("Amy", 18) // Amy,18  showInfo("Amy") // Amy,17  showInfo("Amy", undefined) // Amy,17  showInfo("Amy", null) // Amy, null |

## 箭头函数

箭头函数提供了一种更加简洁的函数书写方式。基本语法是：参数 => 函数体，箭头函数多用于匿名函数的定义。

|  |
| --- |
| let arr = ["10", "5", "40", "25", "1000"]  arr1 = arr.sort()  console.log(arr1)  //上面的代码没有按照数值的大小对数字进行排序，要实现这一点，就必须使用一个排序函数  arr2 = arr.sort(function(a,b){  return a - b  })  console.log(arr2)  // 当箭头函数没有参数或者有多个参数，要用 () 括起来。  // 当箭头函数函数体有多行语句，用 {} 包裹起来，表示代码块，  //当只有一行语句，并且需要返回结果时，可以省略 {} , 结果会自动返回。  var f3 = (a,b) => {  let result = a+b  return result  }  console.log(f3(6,2)) // 8  // 前面代码相当于：  var f4 = (a,b) => a+b |

# JavaScript模块规范

随着网站逐渐变成"互联网应用程序"，嵌入网页的Javascript代码越来越庞大，越来越复杂。

Javascript模块化编程，已经成为一个迫切的需求。理想情况下，开发者只需要实现核心的业务逻辑，其他都可以加载别人已经写好的模块。但是，Javascript不是一种模块化编程语言，它不支持"类"（class），包（package）等概念，也不支持"模块"（module）。

## CommonJS模块规范

每个文件就是一个模块，有自己的作用域。在一个文件里面定义的变量、函数、类，都是私有的，对其他文件不可见。

math.js

|  |
| --- |
| // 定义成员：  const sum = function(a,b){  return a + b  }  const subtract = function(a,b){  return a - b  }  const multiply = function(a,b){  return a \* b  }  const divide = function(a,b){  return a / b  }  // 导出成员：  // module.exports = {  // sum: sum,  // subtract: subtract,  // multiply: multiply,  // divide: divide  // }  //简写  module.exports = {  sum,  subtract,  multiply,  divide  } |

useMath.js

|  |
| --- |
| const m = require('./math.js')  console.log(m)  const result1 = m.sum(1,2)  const result2 = m.multiply(4,4)  console.log(result1,result2)  // 使用node useMath.js命令来运行该js代码 |

## ES6模块规范

### 方式一

1、创建导出模块src/userApi.js

|  |
| --- |
| export function getList() {  console.log('获取数据列表')  }  export function save() {  console.log('保存数据')  } |

2、创建导入模块src/userComponent.js

|  |
| --- |
| //只取需要的方法即可，多个方法用逗号分隔  import { getList, save } from './userApi.js'  getList()  save() |

这时程序还无法运行，因为ES6的模块化无法在Node.js中执行，需要用Babel编辑成ES5后再执行。

3、初始化项目，并配置.babelrc，再在项目中安装转码器，然后在package.json中自定义运行脚本，最后执行转码命令

npm init -y

{

"presets": ["es2015"],

"plugins": []

}

npm install --save-dev babel-preset-es2015

{

// ...

"scripts": {

"build": "babel src -d dist"

}

}

npm run build

4、运行程序

node dist/userComponent.js

### 方式二

1、创建导出模块src/userApi.js

|  |
| --- |
| export default {  getList() {  console.log('获取数据列表2')  },  save() {  console.log('保存数据2')  }  } |

2、创建导入模块src/userComponent.js

|  |
| --- |
| import user from "./userApi.js"  user.getList()  user.save() |

3、初始化项目，并配置.babelrc，再在项目中安装转码器，然后在package.json中自定义运行脚本，最后执行转码命令

npm init -y

{

"presets": ["es2015"],

"plugins": []

}

npm install --save-dev babel-preset-es2015

{

// ...

"scripts": {

"build": "babel src -d dist"

}

}

npm run build

4、运行程序

node dist/userComponent.js