1. **JavaScript**

## 章节目标

* 掌握脚本的基本结构
* 掌握JavaScript的执行原理
* 掌握JavaScript的核心语法
* 掌握JavaScript函数
* 掌握JavaScriptwindow对象
* 掌握Date对象
* 掌握定时函数

## 为什么要学习JavaScript

　JavaScript 通常被称为 JS，他发明的目的，就是作为浏览器的内置脚本语言，为网页开发者提供操控浏览器的能力，他可以让网页呈现出各种特殊效果，为用户提供友好的互动体验。

场景一：

在实际开发中，表单验证的应用场合比较常见，例如，网站中常见的会员注册，我们填写注册信息时，如果某项信息格式输入错误（例如用户名为空，密码长度位数不够等），表单页面将及时给出错误提示，这些错误在没有提交到服务器前，在客户端提前进行验证。这样，用户得到了即时的交互，同时呢，也减轻了网站服务器的压力，这是JavaScript最常用的场合。

场景二:

在实际开发中，会经常使用到层的切换特性，比如编写响应鼠标点击事件，创建动态页面特性，从而高效的控制页面的内容。它们可以在有限的页面空间展现更多的内容，从而增加客户端的体检，也可以使我们的网站更加有动感，有魅力，更加吸引更多的浏览者。

## 什么是JavaScript

JavaScript是一种基于对象(Object)和事件驱动(Event Driven)并具有安全性能的脚本语言。使用它的目的是与HTML超文本标记语言、Java 脚本语言（Java小程序）一起实现在一个Web页面中链接多个对象，与Web客户交互作用。从而可以开发客户端的应用程序等。它是通过嵌入或调入在标准的HTML语言中实现的。它的出现弥补了HTML语言的缺陷，它是Java与HTML折衷的选择。

**发展史（了解）**

**Nombas 和 ScriptEase**

大概在 1992 年，一家称作 Nombas 的公司开发了一种叫做 C 减减（C-minus-minus，简称 Cmm）的嵌入式脚本语言。Cmm 背后的理念很简单：一个足够强大可以替代宏操作（macro）的脚本语言，同时保持与 C （和 C ++）足够的相似性，以便开发人员能很快学会。这个脚本语言捆绑在一个叫做 CEnvi 的共享软件中，它首次向开发人员展示了这种语言的威力。

Nombas 最终把 Cmm 的名字改成了 ScriptEase，原因是后面的部分（mm）听起来过于消极，同时字母 C “令人害怕”。

现在 ScriptEase 已经成为了 Nombas 产品背后的主要驱动力。

**Netscape 发明了 JavaScript**

当 Netscape Navigator 崭露头角时，Nombas 开发了一个可以嵌入网页中的 CEnvi 的版本。这些早期的试验被称为 Espresso Page（浓咖啡般的页面），它们代表了第一个在万维网上使用的客户端语言。而 Nombas 丝毫没有料到它的理念将会成为万维网的一块重要基石。

当网上冲浪越来越流行时，对于开发客户端脚本的需求也逐渐增大。此时，大部分因特网用户还仅仅通过 28.8 kbit/s 的调制解调器连接到网络，即便这时网页已经不断地变得更大和更复杂。而更加加剧用户痛苦的是，仅仅为了简单的表单有效性验证，就要与服务器进行多次地往返交互。设想一下，用户填完一个表单，点击提交按钮，等待了 30 秒的处理后，看到的却是一条告诉你忘记填写一个必要的字段。

那时正处于技术革新最前沿的 Netscape，开始认真考虑开发一种客户端脚本语言来解决简单的处理问题。

当时工作于 Netscape 的 Brendan Eich，开始着手为即将在 1995 年发行的 Netscape Navigator 2.0 开发一个称之为 LiveScript 的脚本语言，当时的目的是在浏览器和服务器（本来要叫它 LiveWire）端使用它。Netscape 与 Sun 及时完成 LiveScript 实现。

就在 Netscape Navigator 2.0 即将正式发布前，Netscape 将其更名为 JavaScript，目的是为了利用 Java 这个因特网时髦词汇。Netscape 的赌注最终得到回报，JavaScript 从此变成了因特网的必备组件。

**三足鼎立**

因为 JavaScript 1.0 如此成功，Netscape 在 Netscape Navigator 3.0 中发布了 1.1 版。恰巧那个时候，微软决定进军浏览器，发布了 IE 3.0 并搭载了一个 JavaScript 的克隆版，叫做 JScript（这样命名是为了避免与 Netscape 潜在的许可纠纷）。微软步入 Web 浏览器领域的这重要一步虽然令其声名狼藉，但也成为 JavaScript 语言发展过程中的重要一步。

在微软进入后，有 3 种不同的 JavaScript 版本同时存在：Netscape Navigator 3.0 中的 JavaScript、IE 中的 JScript 以及 CEnvi 中的 ScriptEase。与 C 和其他编程语言不同的是，JavaScript 并没有一个标准来统一其语法或特性，而这 3 种不同的版本恰恰突出了这个问题。随着业界担心的增加，这个语言的标准化显然已经势在必行。

**标准化**

1997 年，JavaScript 1.1 作为一个草案提交给欧洲计算机制造商协会（ECMA）。第 39 技术委员会（TC39）被委派来“标准化一个通用、跨平台、中立于厂商的脚本语言的语法和语义”。由来自 Netscape、Sun、微软、Borland 和其他一些对脚本编程感兴趣的公司的程序员组成的 TC39 锤炼出了 ECMA-262，该标准定义了名为 ECMAScript 的全新脚本语言。

在接下来的几年里，国际标准化组织及国际电工委员会（ISO/IEC）也采纳 ECMAScript 作为标准（ISO/IEC-16262）。从此，Web 浏览器就开始努力（虽然有着不同的程度的成功和失败）将 ECMAScript 作为 JavaScript 实现的基础。

**什么是脚本语言？**

* 脚本语言介于HTML和C,C++,Java,C#等编程语言之间
* 脚本语言与编程语言有相似地方，其函数与编程语言类似，也有变量。与编程语言之间最大的区别是编程语言的语法和规则更为严格和复杂一些。
* 脚本语言是一种解释性语言，例如Python、VBScript、JavaScript等等，它不像c\c++等可以编译成二进制代码，以可执行文件的形式存在。
* 脚本语言一般都有相应的脚本引擎来解释执行，JavaScript的脚本引擎就是浏览器。

### JavaScript的组成

JS 由三部分组成：

* ECMAScript：也叫解释器，充当翻译角色，这是 JS 的核心部分。
* DOM：文档对象模型（Document Object Model）。DOM 赋予了 JS 操作 HTML 的能力，即 document 操作。
* BOM：浏览器对象模型（Browser Object Model）。BOM 赋予了 JS 操作浏览器的能力，即 window 操作。

### 脚本的基本结构

通常JavaScript代码是用<script>标签嵌入HTML文档中，可以将多个脚本嵌入到一个文档中，只需将每个脚本都封装在<script>标签中即可。浏览器在遇到<script>标签时，将逐行读取内容，直到遇到</script>结束标签为止。然后，浏览器将检查JavaScript语句的语法，如果有任何错误，就会在警告框中显示；如果没有错误，浏览器将编译并执行语句。

* + - 1. **脚本的基本结构**

脚本的基本结构如下：

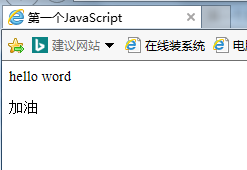
|  |
| --- |
| <script type=*"text/javascript"*>  JavaScript语句;  </script> |

* + - 1. **使用JavaScript脚本在页面上显示输出**

示例代码1：

|  |
| --- |
| <html xmlns=*"http://www.w3.org/1999/xhtml"*>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=utf-8"* />  <title>第一个JavaScript</title>  <script type=*"text/javascript"*>  document.write("hello word");  </script>  </head>  <body>  <p>加油</p>  </body>  </html> |

效果如下图：



其中 document.write()字段是标准的JavaScript命令，用来向页面输出内容，把 document.write()

命令输入到<script>与</script>之间后，浏览器就会把它当做一条JavaScript命令来执行，这样浏览器就会向页面写入hello word。

### 在网页中引用JavaScript的方式

JavaScript作为客户端程序，嵌入网页有以下三种方式：

* 使用<script>标签
* 使用外部JavaScript文件
* 直接在HTML标签中

1. **使用<script>标签**

直接使用<script>标签把JavaScript代码加入到HTML文档中，这是最常用的方法，大部分含有JavaScript的网页都采用这种方法，例如：

|  |
| --- |
| <html xmlns=*"http://www.w3.org/1999/xhtml"*>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=utf-8"* />  <title>第一个JavaScript</title>  <script type=*"text/javascript"*>  document.write("你好");  </script>  </head>  <body>  </body>  </html> |

另外需要注意：<script>...</script>的位置并不是固定的，可以包含在<head>...</head>或<body>...

</body>中任何地方。

但是使用这种方式适用于JavaScript代码较少，并且网站中的每个页面使用的JavaScript代码均不相同的情况。在实际工作中，有时候会希望在若干个页面中运行JavaScript实现相同的页面效果，并且在每个页面中写不同的脚本，对于这样的问题，通常使用外部JavaScript文件。

**2.使用外部JavaScript外部文件**

外部JavaScript文件是将JavaScript写入一个外部文件中，以\*.js为后缀保存这个文件，然后把\*.js文件指定给<script>标签中的“src”属性，这样就可以使用这个外部文件了，如下代码：

|  |
| --- |
| <html xmlns=*"http://www.w3.org/1999/xhtml"*>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=utf-8"* />  <title>第一个JavaScript</title>  <script type=*"text/javascript"* src=*"js/hello.js"*></script>  </head>  <body>  <p>加油</p>  </body>  </html> |

Hello.js就是JavaScript外部文件，src属性表示指定外部JavaScript文件路径。

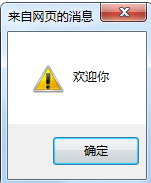
**3.直接在HTML标签中**

有时需要在页面加入很简单的JavaScript代码实现一个简单的页面效果，例如单击按钮时弹出一个对话框，这样通常会在按钮事件中加入JavaScript处理程序，如下面例子：

关键代码：

|  |
| --- |
| <input name=*"btn"* type=*"button"* value=*"弹出消息"* onclick="javascript:alert('欢迎你');" /> |

当点击“弹出消息”按钮时，弹出提示对话框，如下图：



## JavaScript核心语法

大家看前面的示例中源代码时，就会发现与我们学过的Java语言有很多相似的地方。实际上Java像学习过的Java一样，也是一门编程语言，也包含变量的声明、赋值、运算符号、逻辑控制语句等基本语法。

### 变量的声明和赋值

JavaScript是一种弱类型语言，没有明确的数据类型，也就是说，在声明变量时，不需要指定变量的类型，变量的类型由赋给变量的值决定。这一点不像Java那样，在声明变量的同时需要指定变量的数据 类型。

在JavaScript中，变量是使用关键字var声明的。具体语法格式如下：

|  |
| --- |
| **var** 合法的变量名; |

其中，var是声明变量所使用的关键字，“合法的变量名”是遵循JavaScript中变量名规则的变量名。JavaScript中变量名与Java变量的命名规则相同。可以在声明变量的同时为变量赋值，也可以在稍后的脚本中为变量赋值。

变量声明的同时为变量赋值，例如：

|  |
| --- |
| **var** a=20; //声明变量a的同时，将数值20赋给变量a  **var** x,y,z=10; //同时声明多个变量,各个变量之间用逗号隔开 |

特别注意：**JavaScript区分大小写，所以大小写不同的变量名表示不同的变量。**另外，由于JavaScript是一种弱类型语言，所以允许不声明变量而直接使用，系统将会自动声明该变量，例如：

|  |
| --- |
| b=66;//没有声明变量b，直接使 |

**但是这种方法很容易出错，也很难查找排除，不建议使用。**

### 数据类型

虽然JavaScript是一种弱类型语言，在声明变量时不需要声明变量的数据类型，而是由赋给变量的值决定，但有时也需要检测变量的具体数据类型，在JavaScript中，提供了常用的基本数据类型，如下：

* undefined（未定义类型）
* null（空类型）
* number（数值类型）
* string(字符串类型)
* boolean（布尔类型）

**1.typeof运算符**

ECMAScript提供了typeof运算符来判断一个值或变量究竟属于哪种数据类型。语法为：

|  |
| --- |
| **typeof**（变量或者值）; |

其返回结果有以下几种：

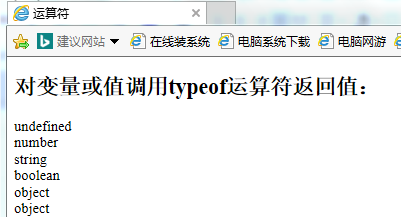
* Undefined：如果变量是undefined型的。
* Number：如果变量是number型的。
* string：如果变量是string型的。
* boolean：如果变量是boolean型的。
* Object:如果变量是null型，或者变量是一种引用类型，例如对象，函数。

如下例子。

实例2：

|  |
| --- |
| <html xmlns=*"http://www.w3.org/1999/xhtml"*>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=utf-8"* />  <title>运算符</title>  <script type=*"text/javascript"*>  document.write("<h2>对变量或值调用typeof运算符返回值：</h2>");  **var** a,b=10,name="zhansgan";  **var** date=**new** Date();  document.write(**typeof**(a)+"<br>");  document.write(**typeof**(b)+"<br>");  document.write(**typeof**(name)+"<br>");  document.write(**typeof**(**true**)+"<br>");  document.write(**typeof**(**null**)+"<br>");  document.write(**typeof**(date)+"<br>");  </script>  </head>  <body>  </body>  </html> |

运行结果如下图：



**2.undefined类型**

Undefined类型只有一个值，即undefined。当声明的变量未初始化时，该变量的默认值是undefined。

|  |
| --- |
| **var** a; |

这行代码声明了变量a，且变量没有初始值，显示为undefined。

1. **null类型**

只有一个值的类型是null，是一个表示“什么都没有”的占位符，可以用来检测某个变量是否被赋值。值undefined实际上是值null派生来的，因此ECMAScript把它们定义为相等的。

|  |
| --- |
| alert(**null**==undefined);//返回值为true |

尽管这两个值相等，但它们的含义不同，undefined是声明了变量但未对该量赋值，null则表示对该变量赋予了一个空值。

**4.number类型**

ECMA-262中定义的最特殊的类型是number类型，这种类型即可以表示32位的整数，还可以表示64位的浮点数。如下列代码：

|  |
| --- |
| **var** num=23;//定义了一个整数  **var** num1=23.0;//定义了一个浮点数 |

整数也可以表示为八进制或十六进制，八进制首数字必须是0，其后的数字可以是任何八进制数字（0-7），十六进制首数字也必须是0，后面是任意的十六进制数字和字母（0-9和A-F），如下代码：

|  |
| --- |
| **var** num2=070;//等于十进制的56  **var** num3=0x1f;//等于十进制的31 |

对于非常大或非常小的数，可以用科学计数法表示浮点数，也就是number类型，另外一个特殊值NaN表示非数它是number类型，如下：

|  |
| --- |
| **typeof**(NaN);//返回值为numbe |

1. **string类型**

在JavaScript中字符串是一组被引号括起来的文本，例如：

|  |
| --- |
| **var** str ="好好学习";//定义了一个字符串 |

与Java不同的是，JavaScript不对“字符”和“字符串”加以区别，如以下例子，也是定义了一个字符串。

|  |
| --- |
| **var** str1 ="a";//定义了一个字符的字符串 |

**6.boolean类型**

Boolean类型是JavaScript最常用的类型之一，它只有两个值true和false.

### 运算符

与Java中运算符一样，在JavaScript中常用的运算符分为算数运算符、比较运算符、逻辑运算符和赋值运算符。

**算术运算符：**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **运算符** | | **说明** | | **示例** | |
| + | | 如果操作数都是数字时执行加法运算，如果其中的操作数有字符串时，会执行连接字符串的作用 | | A = 5 + 8 //结果是13  A ="5"+ 8 //结果是"58" | |
| - | | 减法 | | A = 8 – 5 | |
| \* | | 乘法 | | A = 8 \* 5 | |
| / | | 除法 | | A = 20 / 5 | |
| % | | 取余 - 相除之后的余数 | | 10 % 3 = 1 | |
| ++ | | 一元递增。此运算符只计算一个操作数，将操作数的值加1。返回的值取决于++运算符是位于操作数之前还是位于操作数之后。 | | ++x返回递增后的x值  x++返回递增前的x值 | |
| - - | | 一元递减。此运算符只计算一个操作数。返回的值取决于--运算符是位于操作数之前还是位于操作数之后 | | --x返回递减后的x值  x--返回递减前的x值 | |
| - | | 一元求反。此运算符返回操作数的相反数 | | 如果a等于5，则-a =-5 | |

**比较运算符：**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **运算符** | | **说明** | | **示例** | |
| = = | | 等于。如果两个操作数相等，则返回True | | a = = b | |
| != | | 不等于。如果两个操作数不等，则返回True | | Var2 != 5 | |
| > | | 大于。如果左操作数大于右操作数，则返回True | | Var1 > var2 | |
| >= | | 大于或等于。如果左操作数大于或等于右操作数，则返回True | | Var1 >= 5  Var1 >= var2 | |
| < | | 小于。如果左操作数小于右操作数，则返回True | | Var2 < var1 | |
| <= | | 小于或等于。如果左操作数小于或等于右操作数，则返True | | Var2 <= 4  Var2 <= var1 | |

**逻辑运算符：**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **运算符** | | **例子** | | **说明** | |
| And(&&) | | expr1 && expr2 | | 逻辑与 | |
| Or (||) | | expr1 || expr2 | | 逻辑或 | |
| Not (!) | | !expr | | 逻辑非 | |

**字符串运算符：**

|  |
| --- |
| **var** x = "yellow";  **var** y = "green";  **var** z = x + y + "white"; //则z为yellowgreenwhite  **var** w = y + 9; //则w为green9 |

### 逻辑控制语句

在JavaScript中，逻辑控制语句用于控制程序的执行顺序。同Java一样，逻辑语句也分为两类，条件结构和循环结构。

* + - 1. **条件结构**

与Java一样，条件结构分为if结构和switch结构。

1. If结构

基本语法：

|  |
| --- |
| **if**(表达式){  JavaScript语句1;  }**else**  {  JavaScript语句2;  } |

例如，当i大于10时，提示“大于10”，等于10时，提示“等于10”，否小于10；

|  |
| --- |
| <html xmlns=*"http://www.w3.org/1999/xhtml"*>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=utf-8"* />  <title>if语句</title>  <script type=*"text/javascript"*>  **var** i=10;  **if**(i>10){  alert('大于10');  }**else** **if**(i=10)  {  alert('等于10');  }**else**  {  alert('小于10');  }  </script>  </head>  <body>  </body>  </html> |

1. Switch结构

基本语法如下：

|  |
| --- |
| **switch**(表达式){  **case** 值1:  JavaScript语句1;  **break**;  **case** 值2:  JavaScript语句2;  **break**;  .....  **default**:  JavaScript语句3;  **break**;  } |

与Java中的switch语句一样，switch语句和if语句都是进行条件判断的，当判断条件多于三个时，使用switch语句可以使程序的结构更加清晰。Case表示条件判断，关键字break会使代码跳出switch语句，如果没有关键字break，代码会继续进入下一个case。

与Java中的switch语句不同的是，在JavaScript中，switch语句可以用于字符串，例如：

|  |
| --- |
| <html xmlns=*"http://www.w3.org/1999/xhtml"*>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=utf-8"* />  <title>switch语句</title>  <script type=*"text/javascript"*>  **var** week="星期一";  **switch**(week){  **case** "星期一":  alert('新的一周开始了');  **break**;  **case**"星期五":  alert('明天就可以休息了');  **break**;  **default**:  alert('努力工作');  **break**;  }  </script>  </head>  <body>  </body>  </html> |

* + - 1. **循环语句**

同Java一样，JavaScript中也分为for循环，while循环，do-while循环

1. For循环

基本语法：

|  |
| --- |
| **for**(初始;条件;增量或增减){  JavaScript语句;  } |

其中，初始化参数告诉循环的开始值，必须赋予变量初值；条件是用于判断循环终止时的条件，若条件满足，则继续执行循环体重的语句，否则跳出循环；增量或减量是定义循环控制在每次循环时怎么变化的，在三个条件之间，必须使用分号(;)隔开。

1. While循环

基本语法：

|  |
| --- |
| **while**(条件){  JavaScript语句;  } |

其中，当条件为真时，就执行JavaScript语句，相反，退出循环。

1. Do-while循环语句

具体语法：

|  |
| --- |
| **do**{  JavaScript语句;  }**while**(条件); |

表示反复执行JavaScript语句，直到条件为假才退出循环。

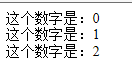
1. 中断循环

同Java一样，在JavaScript中，也有两种特殊的语句可以用于循环内部，用来终止循环：break和continue。

* Break：可以立即退出整个循环 ，例如：

|  |
| --- |
| <script type=*"text/javascript"*>  **var** i=0;  **for**(i=0;i<=5;i++){  **if**(i==3){  **break**;  }  document.write("这个数字是："+i+"<br/>");  }  </script> |

运行结果如下图：

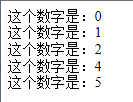


Continue只是退出当前循环，根据控件表达式还允许继续进行下一次循环。

例如：

|  |
| --- |
| <script type=*"text/javascript"*>  **var** i=0;  **for**(i=0;i<=5;i++){  **if**(i==3){  //break;  **continue**;  }  document.write("这个数字是："+i+"<br/>");  }  </script> |

运行结果如下图：



### 数组

**问题：**

王大爷有6只乌龟，它们的体重分别是3kg,5kg,1kg,3.4kg,2kg,50kg。请问这六只乌龟的总体重是多少？平均体重是多少？请你用现在掌握的技术变一个程序解决。用什么最方便呢？答案是数组。

在实际开发中，总面临大量数据存储的问题。JavaScript语言不想C那样合适用于数据结构的设计，因此需要系统内部提供针对大量数据存储的工具，数组因此而产生。JavaScript数据的目标是能组织存储各种各样的数据，并且访问方式和其他语言一样，特点是能混合存储类型不相同的数据。

数组的概念：JavaScript数组是将多个数据对象编码存储、提供一致的存取方式的集合。与其他程序语言不同的是，由于JS的弱检查特性，因此在JS数组中可以同时存储不同类型的变量，比如你可以把数字、字符串、字符、对象等内容放在同一个数组中。

* + - 1. **创建数组**

不同于JavaScript的其他变量，在使用数组之前，必须首先声明数组。下面的例子创建了一个有4个人元素的数组：

|  |
| --- |
| scores=**new** Array(4); |

如果想给数组赋值的话，要在中括号中使用一个索引。索引从0开始，因为在本例中数组的索引是0-3。如以下例子：

|  |
| --- |
| scores=**new** Array(4);  scores[0]=39;  scores[1]=40;  scores[2]=41;  scores[3]=42;  document.write("scores："+scores+"<br/>"); |

可以声明一个数组并且同时指定元素的值。如下：

|  |
| --- |
| scores1=**new** Array(39,40,41,42);  document.write("scores1："+scores1+"<br/>"); |

也可以使用简略的语法声明数组并赋值。

|  |
| --- |
| scores2=[39,40,41,42];  document.write("scores2："+scores2+"<br/>"); |

如果不使用关键字new，应使用中括号。

* + - 1. **理解数组的长度**

像字符串一样，数组有长度属性。它告诉数组中元素的个数。如果你在建立数组时指定长度，这个值就成为长度属性的值。如以下例子，将输出数字30；

|  |
| --- |
| scores4=**new** Array(30);  document.write("scores4数组长度为:"+scores4.length);//获取数组scores4的长度 |

在声明一个数组时可以不指定长度，并且可以在以后改变数组的长度，只要给元素赋值或者改变长度属性就可以了。如下代码：

|  |
| --- |
| test=**new** Array();  test[0]=39;  test[5]=40;  document.write("test数组长度为:"+test.length);//获取数组scores4的长 |

由于最大的索引数是5，因此数组的长度属性是6。

* + - 1. **存取数组元素**

可以使用与赋值时相同的符号来读取数组的内容。如下，将显示数组scores数组的前两个元素的值：

|  |
| --- |
| scores=**new** Array(4);  scores[0]=39;  scores[1]=40;  scores[2]=41;  scores[3]=42;  score="scores:"+scores[0]+","+scores[1];  document.write(score); |

* + - 1. **分离字符串**

JavaScript包括了一个名为split的字符串方法，它可以将一个字符串分成几个部分。使用这个方法时，要指定需要分离的字符串和以哪个字符为标志分离字符串：

|  |
| --- |
| str="A B C";  pars=str.split(" ");  document.write("pars[0]:"+pars[0]+"<br/>pars[1]:"+pars[1]+"<br/>pars[2]:"+pars[2]); |

Split方法将str字符串在每个空格处分开，这样就分成了三个字符串。这个字符串存储在一个名为parts的字符串数组中。上例语句执行后，parts数组的元素内容如下：

* parts[0]=”A”
* parts[1]=”B”
* parts[2]=”C”

JavaScript还提供了另一个数组操作方法—join，它的作用与split相反。下面的语句将parts数组合并成一个字符串：

|  |
| --- |
| Fullname=pars.join(" ");//合并字符串  document.write("Fullname:"+Fullname); |

括号内的数值指定了隔开数组各个部分的字符。本例中，使用的是空格，合并后的字符串就是A B C

,如果不指定一个字符，默认使用逗号。

* + - 1. **字符串数组的排序**

JavaScript还提供了一个数组的排序方法—sort()，它可以按字母顺序将数组进行排序。例如：

|  |
| --- |
| **var** arr1 = ["b","d","a","r","c"];  arr1.sort();  alert(arr1); |

arr1.sort()将arr1数组进行排序，并将结果存储在一个新的数组arr1中。

JavaScript还提供了另一个数组排序方法—reverse，它的作用与sort相反。对数组元素倒序排序：

|  |
| --- |
| **var** arr1 = ["b","d","a","r","c"];  arr1.reverse(); //倒叙排序 是按照集合的位置进行倒叙的  alert(arr1); |

* + - 1. **数值数组排序**

因为sort排序的方法是按字母顺序的，所以它不能用于数值数组。例如，如果一个数组包括了数值4、10、30、200，排序结果将是10、30、200、4——一点都不对。幸运的是，有一个解决方案：可以指定一个函数作为排序的参数，这个函数用于数字的比较。如下代码：

这个例子定义了一个简单的函数num，它将两个数相减。当你把这个函数用于排序方法中时，数组的正确地排列为：4、30、40、200

|  |
| --- |
| **function** num(a,b){  **return** a-b;  }  nums=**new** Array(30,40,200,4);  sortnums=nums.sort(num);  alert(sortnums); |

如果a在b的前面，JavaScript的比较函数返回一个负数，如果两者相同，返回0，如果a在b后面返回一个正数，这就是a-b可用来排列数字的原因。

* + - 1. **数组元素的添加**

JavaScript提供了数组添加方法：push()、unshift()、splice()；

push()：将一个或多个新元素添加到数组结尾，并返回数组新长度；如下示例：

|  |
| --- |
| **var** arr = [1,2] ;  **var** result = arr.push(1,2,**true**); //push方法向数组中追加元素(返回值为新数组的长度)  alert(arr);  alert(result); |

push()向数组arr结尾添加了三个元素：1，2，true，返回新数组长度为5；

unshift()：将一个或多个新元素添加到数组开始，数组中的元素自动后移，返回数组新长度。如下示例：

|  |
| --- |
| **var** arr1 = [1,2,3, **true**];  **var** r2 = arr.unshift(10,**false**); //从头部插入多个元素(返回新数组的长度)  alert(arr1);  alert(r2); |

unshift()向数组arr1开始添加了三个元素：10，false，返回新数组长度为6；

splice()将一个或多个新元素插入到数组的指定位置，插入位置的元素自动后移，返回""。

语法为：

|  |
| --- |
| arrayObject.splice(index,howmany,element1,.....,elementX) |

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | | **描述** | |
| index | | 必需。规定从何处添加/删除元素。  该参数是开始插入和（或）删除的数组元素的下标，必须是数字。 | |
| howmany | | 必需。规定应该删除多少元素。必须是数字，但可以是 "0"。  如果未规定此参数，则删除从 index 开始到原数组结尾的所有元素。 |
| element1 | | 可选。规定要添加到数组的新元素。从 index 所指的下标处开始插入。 | |
| elementX | | 可选。可向数组添加若干元素。 | |

例如，我们创建一个新数组，并向其添加一个元素。

|  |
| --- |
| **var** arr =[1,2,3,4,5];  document.write(arr + "<br />")  arr.splice(2,0,6); //1,2,6,3,4,5  document.write(arr + "<br />") |

* + - 1. **数组元素的删除**

JavaScript提供了删除数组元素方法：pop()、shift()、splice()。

pop()：移除最后一个元素并返回该元素值。如下例子：

|  |
| --- |
| **var** arr = [1,2,**false**] ;  **var** obj = arr.pop();//从数组的尾部移除一个元素(返回值为移除的元素)  alert(arr);  alert(obj); |

pop()移除数组arr最后一个元素：false，返回移除元素false。

shift()：移除最前一个元素并返回该元素值，数组中元素自动前移。如下例子：

|  |
| --- |
| **var** arr1 = [1,2,3, **true**];  **var** r1 = arr.shift();//从头部移除一个元素(返回移除的元素)  alert(arr1);  alert(r1); |

shift()移除数组arr头部元素，返回了移除元素1

arrayObj.splice(deletePos,deleteCount)：删除从指定位置deletePos开始的指定数量deleteCount的元素，数组形式返回所移除的元素。

例如：删除位于 arr2 2 的元素，并添加一个新元素来替代被删除的元素：

|  |
| --- |
| **var** arr2 = [1,2,3, **true**];  document.write(arr2 + "<br />")  arr2.splice(2,1,"a") //1,2,a,true  document.write(arr2); |

* + - 1. **数组的截取和合并**

JavaScript提供了数组截取方法slice()和数组合并方法concat()。

截取方法语法为：

|  |
| --- |
| slice(start, [end]); |

start：必需。规定从何处开始选取。如果是负数，那么它规定从数组尾部开始算起的位置。也就是说，-1 指最后一个元素，-2 指倒数第二个元素，以此类推。

end：可选。规定从何处结束选取。该参数是数组片断结束处的数组下标。如果没有指定该参数，那么切分的数组包含从 start 到数组结束的所有元素。如果这个参数是负数，那么它规定的是从数组尾部开始算起的元素。

注意：该方法并不会修改数组，而是返回一个子数组。

例如，创建一个新数组，然后显示从其中选取的元素：

|  |
| --- |
| **var** arr = [1,2,3, **true**];  document.write(arr + "<br />");  document.write(arr.slice(1,2) + "<br />");//输出2  document.write(arr); |

concat() 方法用于连接两个或多个数组。该方法不会改变现有的数组，而仅仅会返回被连接数组的一个副本。

语法为：

|  |
| --- |
| arrayObject.concat(arrayX,arrayX,......,arrayX); |

arrayX:必需。该参数可以是具体的值，也可以是数组对象。可以是任意多个。

例如：创建两个数组，然后使用 concat() 把它们连接起来，代码如下：

|  |
| --- |
| **var** arr1 =[1,2,3];  **var** arr2 =[4,5,6];  document.write(arr1.concat(arr2));//1,2,3,4,5,6 |

### JavaScript语法规则

JavaScript是一个简单的语言，但必须认真地按照它的规则来使用—规则就是定义如何使用该语言。

* + - 1. **区分大小写**

几乎所有JavaScript的内容都是区分大小写的：小写字母和大写字母是不能互相替换的。几个基本的规则如下：

* JavaScript的关键字，例如for和if，永远是小写。
* 内置对象，例如Math和Date是以大写字母开头的。
* DOM对象的名称通常是小写，但其方法经常是大小写混合的。通常第一个字母不是大写，例如getElementById。
  + - 1. **变量、对象和函数名称**

当定义自己使用的变量、对象或函数时，可以选择它们的名称。名称可以包括大写字母、小写字母、数字和下划线。名字必须以一个字母或下划线开头。

可以选择在变量名中使用大写字母还是小写字母，但必须记住JavaScript是区分大小写的：score、Score和SCORE将被认为是三个不同的变量。要保证每次使用同一个变量时都是同样的名称。

* + - 1. **保留字**

变量名称的另一个规则是它们不能使用保留字。保留字包括JavaScript语言的组成部分，（例如if和for）、DOM对象的名称（例如window和document）和内置对象的名称（例如Math和Date）。

* + - 1. **空格**

JavaScript是忽略空格的。程序中可以在一行中包括空格和制表符，也可以包括空白行，这些不会造成错误。空格通常是为了使JavaScript程序更具可读性。

### 注释

JavaScript语言注释与Java语言的注释一样，分为单行注释和多行注释两种。

* 单行注释以//开始，以行末结束，例如：

|  |
| --- |
| b=66;//没有声明变量b，直接使用 |

* 多行注释以/\*开始，以\*/结束。例如：

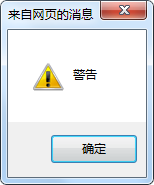
|  |
| --- |
| <script type=*"text/javascript"*>  /\*使用for循环运行document.write("这个数字是："+i+"<br/>");5次  使用document.write在页面输出i\*/  **var** i=0;  **for**(i=0;i<=5;i++){  document.write("这个数字是："+i+"<br/>");  }  </script> |

### 常用的输入、输出

在网页上经常会弹出一些信息框，例如注册时弹出某些文本框中必须输入值的信息框或弹出一个等待用户输入数据的对话框，这样的输入或输出在JavaScript中称为警告对话框和提示对话框。

* + - 1. **警告（alert）**

alert()方法会创建一个特殊的小窗口，该窗口带有一个字符串和一个“确定”按钮，如下图：



alert()方法的基本语法格式为：

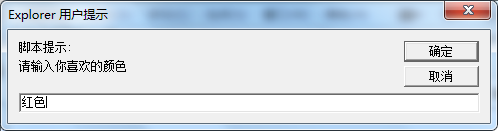
|  |
| --- |
| alert('提示信息'); |

该方法将弹出一个警告对话框，其内容可以是一个变量的值，也可是一个表达式的值。如下：

|  |
| --- |
| **var** username="小米";  **var** str="我的名字叫小米";  alert('hello');  alert('我的名字叫'+username);  alert(str); |

* + - 1. **提示（prompt）**

prompt()方法会弹出一个提示对话框，等待用户输入一行数据，如下图：



prompt()方法的基本格式语法为：

|  |
| --- |
| prompt('提示信息','请输入框的默认值'); |

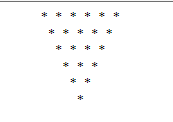
该方法的返回值可以被引用或存储到变量中，例如：

|  |
| --- |
| **var** color=prompt("请输入你喜欢的颜色","红色"); |

prompt()方法的第一个参数显示在对话框上，通常是一些提示信息，第二参数出现在用户输入的文本框中，且被选中，作为默认使用，第二个参数也可以省略。

* + - 1. **课上小练习**

1. 根据prompt（）方法输入的次数，连续在页面上输入hello word，并提示输出次数。
2. 根据用户输入的整数在页面打印倒三角。



### 程序调试

程序调试是JavaScript重点一个重要环节，我们可以使用alert（）方法进行调试。

模拟一个错误程序：

在课上练习的基础上修改下代码，把i>0改成i<0;执行完成后发现什么效果也没有。

这时要怎么找到咱们哪一块写错了呢？可以使用alert（）进行调试。

首先使用alert（）方法检查t的值是否与输入值一致，增加alert（）方法的代码如下所示：

|  |
| --- |
| <script type=*"text/javascript"*>  **var** t=prompt("请输入一个整数","");  alert(t);  **for**(**var** i=t;i<0;i--){  **for**(**var** j=0;j<i;j++){  document.write("\*&nbsp;&nbsp;");  }  document.write("<br/>");  }  </script> |

此时运行程序，在输入框中输入4，即可弹出图1所示的提示框，说明t的值是存在的，然后把alert(t)改成i查看是否存在的代码，即alert(i),这样弹出图2所示的提示框，说明此时i未被赋值。

图1 图2

把alert(i)放到for循环后面，查看是否会被赋值，代码如下：

|  |
| --- |
| <script type=*"text/javascript"*>  **var** t=prompt("请输入一个整数","");  **for**(**var** i=t;i<0;i--){  alert(i);  **for**(**var** j=0;j<i;j++){  document.write("\*&nbsp;&nbsp;");  }  document.write("<br/>");  }  </script> |

再次运行程序，没有任何警告框弹出，说明这个for循环的条件不被满足，检查这句代码，我们发现当t=4时，后面又出现i<0的条件，这是两个矛盾的条件，当然不会成立，我们把把i<0改成i>0;就正确了，然后重新运行程序，发现出现了我们想要的效果。

## JavaScript常用语法—函数

在JavaScript中函数类似于Java中的方法，是执行特定任务的语句块。但是JavaScript中的函数使用更简单，不用定义函数属于哪个类，更不需要像Java中使用“对象名.方法名()”的方式调用函数，可以直接调用函数名称来使用函数。在JavaScript中函数有如下特点：

* JavaScript中的函数是完成特定功能并可以随时随地地运行的语句，比如简易的计算器、表单验证等。
* 函数不仅能在一个或多个HTML页面中被多次调用，而且能够在不同网站中应用，从而提高代码的复用率。
* 函数为程序设计人员带来了很多方便，通常在进行一个复杂的程序设计时，总是根据所要完成的功能，将程序划分为一些相对独立的部分，每一部分编写一个函数，从而使各部分充分独立，任务单一，使程序结构清晰，易读，易懂，易重用，易维护。
* JavaScript函数可以封装那些在程序中可能要多次用到的模块，并可以作为事件驱动的结果而被调用，从而实现一个函数与事件驱动相关联。
* 可以将值传递给函数，函数也可以返回一个值。

JavaScript中函数有两种，一种是JavaScript自带的系统函数，一种是用户自行创建的自定义函数。

### 常用系统函数

ECMScript提供了两种把非数字的原始值转成数字的函数，即parseInt()和parseFloat()，另外还提供了一个检查是否是非数字的函数isNaN().

* + - 1. **parseInt()**

parseInt()函数可解析一个字符串，并返回一个整数，语法格式为：

|  |
| --- |
| parseInt("字符串"); |

在判断字符串是否是数字前，parseInt()和parseFloat()都会分析该字符串。

parseInt()函数首先查看位置0处的字符，判断它是否是个有效数字，如果不是则返回NaN,不在继续执行其他操作。但如果该字符是有效数字，该函数将查看位置1处的字符，进行同样的测试，这一过程将持续到发现非有效数字的字符为止，此时parseInt将该字符串之前的字符转换成数字，例如：

|  |
| --- |
| **var** num1=parseInt("78.89");//返回值为78  **var** num2=parseInt("4567cc");//返回值为4567  **var** num3=parseInt("this36");//返回值为NaN |

下面通过一个例子来掌握parseInt()函数的用法。使用prompt()方法获取两个数，使用parseInt()函数把这两个数转换为整数，然后相加并输出在页面，实例代码如下：

|  |
| --- |
| <script type=*"text/javascript"*>  **var** op1=prompt("请输入第一个数：",",");  **var** op2=prompt("请输入第二个数：",",");  **var** p1=parseInt(op1);  **var** p2=parseInt(op2);  **var** result=p1+p2;  document.write(p1+"+"+p2+"="+result);  </script> |

运行结果如下图：



* + - 1. **parseFloat()**

parseFloat()函数可解析一个字符串，并返回一个浮点数，语法格式为：

|  |
| --- |
| parseFloat("字符串"); |

parseFloat()函数和parseInt()函数处理方式相同。从位置0开始查看每个字符串，直到找到第一个非有效的字符为止，然后把该字符之间的字符串转换为浮点数。

对于这个函数来说，第一个出现的小数和点是有效字符，如果两个小数点，那么第二个小数点被看作是无效的，例如：

|  |
| --- |
| **var** num1=parseFloat("4567cc");//返回值为4567  **var** num2=parseFloat("45.58");//返回值为45.58  **var** num3=parseFloat("45.58.25");//返回值为45.58  **var** num4=parseFloat("this36");//返回值为NaN  alert(num1);  alert(num2);  alert(num3);  alert(num4); |

* + - 1. **IsNaN()**

isNaN()函数用于检查其参数是否是非数字，语法格式如下：

|  |
| --- |
| isNaN(x); |

如果x是特殊的非数字值，返回值就是true，否则返回false，例如：

|  |
| --- |
| **var** num1=isNaN("12.5");//返回值为false  **var** num2=isNaN("12.5s");//返回值为true  **var** num3=isNaN("45.8");//返回值为false  alert(num1);  alert(num2);  alert(num3); |

### 自定义函数

同Java语言一样，我们需要先定义函数，然后 才能调用函数，下面来详细介绍。

* + - 1. **创建函数**

在JavaScript中，自定义函数由关键字function，函数名，一组参数以及置于括号中的待执行的JavaScript语句组成。语法格式为：

|  |
| --- |
| **function** 函数名(参数1，参数2,...){  JavaScript语句;  } |

* 参数1，参数2等是传入函数的变量或值。
* “{”和“}”定义了函数的开始和结束。

函数中的参数是可选的，当传入参数时通常把函数称为有参函数，当没有传入参数时，称为无参函数，无参函数必须在其函数名后加括号。例如：

|  |
| --- |
| **function** 函数名(){  JavaScript语句;  } |

在JavaScript中，return语句用来规定从函数返回的值，因此需要返回某个值的函数必须使用return语句。

* + - 1. **调用函数**

要执行一个函数，必须先调用这个函数，当调用函数时，必须指定函数名及其后面的参数（如果有参数）。函数的调用一般和表单元素的事件结合使用，调用格式如下：

|  |
| --- |
| 事件名=“函数名()”; |

下面通过示例1和示例2来学习如何创建函数和调用函数。

示例1：

|  |
| --- |
| <html xmlns=*"http://www.w3.org/1999/xhtml"*>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=utf-8"* />  <title>示例1</title>  <script type=*"text/javascript"*>  **function** showHello(){  **for**(**var** i=0;i<5;i++){  document.write("<h2>Hello Word</h2>");  }  }  </script>  </head>  <body>  <input name=*"btn"* type=*"button"* value=*"显示5次Hello Word"* onclick="showHello()" />  </body>  </html> |

* showHello()是创建的无参函数。
* Onclick表示按钮单击事件，当点击按钮时调用函数showHello()。

在浏览器运行示例1如图1所示，单击按钮，调用showHello()无参函数，动态的在页面中循环输出5行“Hello Word”，如图2所示：

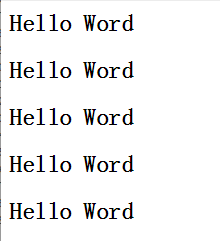
 

图1 图2

在示例1中使用的无参函数，运行一次页面只能输出5行“Hello Word”，如果需要根据用户的要求每次输入不同行数，该怎么办呢？有参函数可以实现这样的功能。

下面修改示例1，把函数showHello()修改成一个有参函数，使用prompt()提示用户每次输出“Hello Word”的行数，然后啊prompt()返回的值作为参数传递给函数showHello().

示例2：

|  |
| --- |
| <html xmlns=*"http://www.w3.org/1999/xhtml"*>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=utf-8"* />  <title>示例2</title>  <script type=*"text/javascript"*>  **function** showHello(count){  **for**(**var** i=0;i<count;i++){  document.write("<h2>Hello Word</h2>");  }  }  </script>  </head>  <body>  <input name=*"btn"* type=*"button"* value=*"请输入显示Hello Word的次数"* onclick="showHello(prompt('请输入显示Hello Word的次数',','))" />  </body>  </html> |

showHello(count)是有参函数，count表示传递的参数。

使用prompt('请输入显示Hello Word的次数',',')弹出窗口，把用户输入的值作为参数传递给函数showHello(count)。

在for循环中，参数count的值作为循环输出“Hello Word”的次数。

在浏览器运行示例2如图1所示，单击按钮，弹出提示用户输入显示Hello Word次数的窗口；用户输入值后，根据用户输入的值在页面上输出Hello Word，如图2所示：

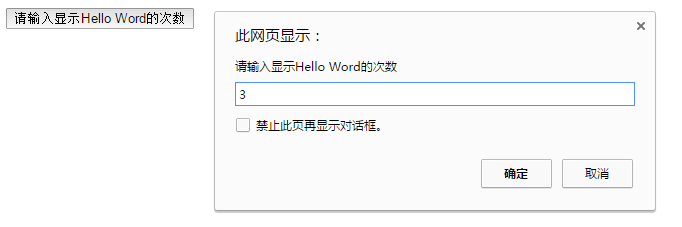
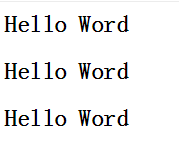
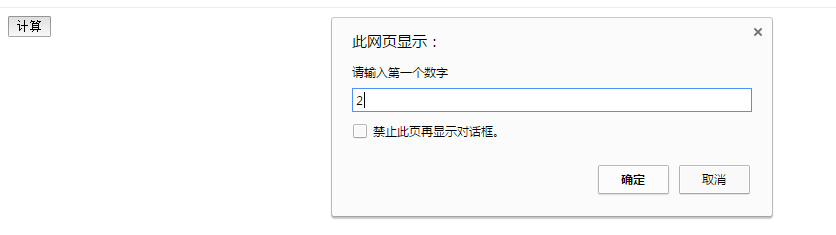
 

图1 图2

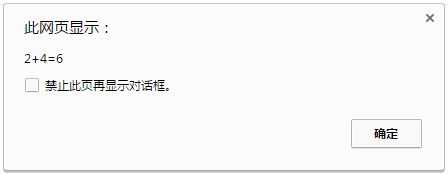
### 上机练习3

需求说明:

单击页面的按钮时，调用函数，使用prompt()方法获取两个变量的值和一个运算符，如下图所示：



使用两个运算的结果使用alert()方法显示出来，如下图：



使用switch判断获取运算符。

### 变量的作用域

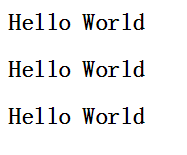
与Java中的变量一样，在JavaScript中，根据变量作用的范围不同，可分为全局变量和局部变量。

JavaScript中的全局变量与Java中的成员变量类似，是在所有函数之外的脚本中声明的变量，作用的范围是该变量定义后的所有语句，包括其后定义的函数中的代码，以及其后的<script>和</script>标签中的代码。

JavaScript中的局部变量与Java中局部变量类似，是在函数内声明的变量，只有在该函数中且位于该变量之后的代码可以使用这个变量，如果在之后的其他函数中声明了与这个局部变量同名的变量，则后声明的变量与这个局部变量毫无关系。

如以下例子

|  |
| --- |
| <html xmlns=*"http://www.w3.org/1999/xhtml"*>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=utf-8"* />  <title>变量的作用域</title>  <script type=*"text/javascript"*>  **var** num;  **function** showHello(){  **for**(**var** i=0;i<num;i++)  {  document.write("<h2>Hello World</h2>");  }  }  **function** counts(){  num=prompt("请输入显示HelloWorld的次数：","");  showHello();  }  </script>  </head>  <body>  <input name=*"btn"* type=*"button"* value=*"请输入显示HelloWorld的次数"* onclick="counts()"/>  </body>  </html> |

运行上面的例子，在prompt()弹出的输入框中输入3，点击确定按钮，运行结果如下图：  


从运行结果可以看出，一开始声明的num为全局变量，但在showHello()中声明的num为局部变量，它只作用于函数showHello()中。

## window对象

wndow对象是浏览器对象中的顶层对象，同时window对象也称为浏览器对象，浏览器打开HTML文档时，通常会创建一个window对象。然后，如果文档定义了一个或多个框架，浏览器将为原始文档创建一个window对象，同时为每一个框架另外创建一个window对象，下面来学习一下window对象常用属性、方法和事件。

### 常用的属性

window对象常用属性如下表：

|  |  |
| --- | --- |
| **名称** | **说明** |
| screen | 有关客户端的屏幕和显示性能的信息 |
| history | 有关客户访问过的URL的信息 |
| location | 有关当前URL的信息 |

### 常用的方法

wndow对象的常用方法如下表：

|  |  |
| --- | --- |
| **名称** | **说明** |
| prompt | 显示可提示用户输入的对话框 |
| alert | 显示带有一个提示信息和一个确定按钮的警示框 |
| confirm | 显示一个带有提示信息、确定和取消按钮的对话框 |
| close | 关闭浏览器窗口 |
| open | 打开一个新的浏览器窗口，加载给定 URL 所指定的文档 |
| setTimeout | 在指定的毫秒数后调用函数或计算表达式 |
| setInterval | 按照指定的周期（以毫秒计）来调用函数或表达式 |

在JavaScript中，方法的使用格式如下：

|  |
| --- |
| window.方法名(); |

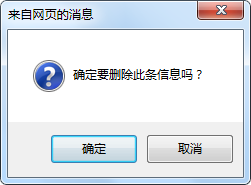
由于window对象表示当前窗口对象，是一个全局变量，因此可以把当前窗口对象的方法当作函数来使用。比如alert()，而不使用window.alert()。

1. **confirm()**

confrim()将弹出一个确认对话框，语法格式为：

|  |
| --- |
| window.confirm("对话框中显示的纯文本"); |

例如，confirm(“确定要删除此条信息吗？”)在页面弹出如下图所示：



在confrim()弹出的确认对话框中，有一条提示信息，一个“确定”按钮和一个“取消”按钮，如果用户点击确认按钮，则confrim()返回true,如果点击取消按钮，则confrim()返回false。

在用户单击“确定”按钮或“取消”按钮把对话框关闭之前，它将阻止用户对浏览器的所有操作。在调用confrim()时，将暂时对JavaScript代码的执行，在用户作出响应之前，不会执行下一条语句，如以下例子：

|  |
| --- |
| <html xmlns=*"http://www.w3.org/1999/xhtml"*>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=utf-8"* />  <title>confirm方法</title>  <script type=*"text/javascript"*>  **var** flag=confirm("确认要删除此条信息吗？");  **if**(flag==**true**){  alert("删除成功！");  }  **else**{  alert("你取消了删除");  }  </script>  </head>  <body>  </body>  </html> |

在浏览器中运行上面的代码，如果单击“确定”按钮弹出如图1所示的对话框，如果点击“取消”按钮将弹出如图2所示对话框。

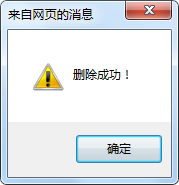
 

图1 图2

前面我们学习了prompt()方法和alert()方法的用法，现在我又学了confrim()，虽然它们都是在页面上弹出对话框，但是作用却不相同。

* alert()只有一个参数，仅显示警告的消息，无返回值，不能对脚本产生任何改变。
* confrim()有两个参数，是输入对话框，用来提示用户输入一些信息，点击“取消”按钮则返回null，单击“确定”按钮则返回用户输入的值。
* confrim()只有一个参数，是确认对话框，显示提示框消息，“确认”按钮和“取消”按钮，点击“确定”按钮返回true，点击“取消”按钮返回false，常用于if-else语句。

**2.open()**

在页面上弹出一个新的浏览器窗口，弹出窗口的语法如下：

|  |
| --- |
| window.open("弹出窗口的URL","窗口名称","窗口特征"); |

窗口的特征属性如下表：

|  |  |
| --- | --- |
| **名称** | **说明** |
| height、width | 窗口文档显示区的高度、宽度。以像素计。 |
| left、top | 窗口的x坐标、y坐标。以像素计 |
| toolbar=yes | no  |1 | 0 | 是否显示浏览器的工具栏。黙认是yes。 |
| scrollbars=yes | no  |1 | 0 | 是否显示滚动条。黙认是yes。 |
| location=yes | no  |1 | 0 | 是否显示地址地段。黙认是yes。 |
| status=yes | no  |1 | 0 | 是否添加状态栏。黙认是yes。 |
| menubar=yes | no  |1 | 0 | 是否显示菜单栏。黙认是yes。 |
| resizable=yes | no  |1 | 0 | 窗口是否可调节尺寸。黙认是yes。 |
| titlebar=yes | no  |1 | 0 | 是否显示标题栏。黙认是yes。 |
| fullscreen=yes | no  |1 | 0 | 是否使用全屏模式显示浏览器。黙认是no。处于全屏模式的窗口必须同时处于剧院模式。 |

如以下例子：

|  |
| --- |
| <html xmlns=*"http://www.w3.org/1999/xhtml"*>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=utf-8"* />  <title>open方法</title>  <script type=*"text/javascript"*>  **function** open\_win()  {  window.open("http://www.baidu.com","baidu","toolbar=yes, location=yes, status=no, menubar=yes, scrollbars=yes, resizable=no, width=400, height=400")//点击打开百度页面，设置显示浏览器的工具栏，显示地址栏，不添加状态栏，显示菜单栏，显示滚动条，窗口不能调节尺寸，高度、宽度分别为400像素  }  </script>  </head>  <body>  <form>  <input type=*"button"* value=*"打开窗口"* onclick="open\_win()">  </form>  </body>  </html> |

**3.close()方法**

close()方法用于关闭浏览器窗口，语法格式为：

|  |
| --- |
| window.close(); |

这里介绍了window对象的几种常用方法。

### 常用的事件

其实window对象有很多事件，不过常用的不多，比较常用的window对象事件如下表：

|  |  |
| --- | --- |
| **名称** | **说明** |
| onload | 一个页面或一幅图像完成加载 |
| onmouseover | 鼠标移到某元素之上 |
| onlick | 当用户单击某个对象时调用的事件句柄 |
| onkeydowm | 某个键盘按键被 按下 |
| onchange | 域的内容被改变 |

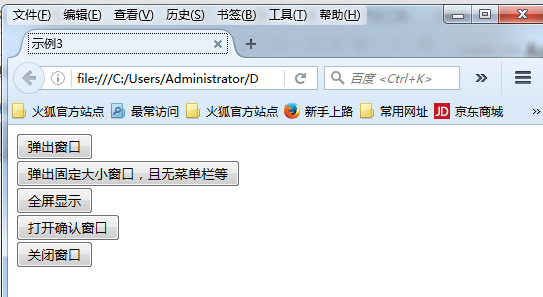
我们在网上冲浪时，通常打开一个页面就会有广告页面或网站信息声明页面等窗口弹出来，并且很多网站的页面中有可以对当前窗口进行关闭的按钮。维护网站的信息时，删除一条信息通常会弹出一个“确认是否删除”的提示框。在线看电影时，我们经常会通过全屏显示来观看，这些功能都可通过window对象来实现。下面通过一个例子来介绍最常用的window对象。

示例4：

|  |
| --- |
| <html xmlns=*"http://www.w3.org/1999/xhtml"*>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=utf-8"* />  <title>示例3</title>  <script type=*"text/javascript"*>  /\*弹出窗口\*/  **function** open\_adv(){  window.open("adv.html");  }  /\*弹出固定大小窗口，并且无菜单栏等\*/  **function** open\_fix\_adv(){  window.open("adv.html","","height=380,width=320,toolbar=0,scrollbars=0,location=0,status=0,menubar=0,resizable=0");  }  /\*全屏显示\*/  **function** fullscreen(){  window.open("adv.html","","fullscreen=yes");  }  /\*弹出确认消息框\*/  **function** confirm\_msg(){  **if**(confirm("你相信自己是最棒的吗？"))  { alert("有信心必定会赢，没信息一定会输！"); }  }  /\*关闭窗口\*/  **function** close\_plan(){  window.close();  }  </script>  </head>  <body>  <input name=*"open1"* type=*"button"* value=*"弹出窗口"* onclick="open\_adv()" /><br/>  <input name=*"open2"* type=*"button"* value=*"弹出固定大小窗口，且无菜单栏等"* onclick="open\_fix\_adv()"/><br/>  <input name=*"full"* type=*"button"* value=*"全屏显示"* onclick="fullscreen()"/><br/>  <input name=*"con"* type=*"button"* value=*"打开确认窗口"* onclick="confirm\_msg()"/><br/>  <input name=*"c"* type=*"button"* value=*"关闭窗口"* onclick="close\_plan()"/>  </body>  </html> |

这个例子主要实现了弹出窗口，全屏显示页面，打开确定窗口和关闭窗口的功能。

首先创建不同的函数实现各个功能，然后通过各个按钮的事件来调用对应的函数 ，实现弹出窗口,全屏显示等功能，运行效果如下图：



* 用户单击“弹出窗口”按钮时，调用open\_adv()函数，这个函数会调用window.open()方法打开新的窗口。由于open()方法只设定了打开窗口的页面，而没有对窗口名称和窗口特征进行设置，因此打开的窗口和通常大家在浏览器中打开的窗口一样。
* 用户单击“弹出固定大小窗口，且无菜单栏等”按钮时，则调用open\_fix\_adv()方法，但是此方法对弹出窗口的大小，是否有菜单栏、地址栏等进行了设置，即弹出的窗口大小固定，没有地址栏、菜单栏等。
* 单击“全屏显示”按钮，调用了fullscreen()函数。
* 单击“打开确认窗口”按钮，调用了confirm\_msg()函数，在这个函数中使用了if-else语句，并且把confirm()方法的返回值作为if-else语句的表达式进行判断，在confirm()弹出的确认对话框中，当点击“确定”按钮时则使用alert()方法弹出一个警告提示框，否则什么也不显示。
* 单击“关闭窗口”按钮，调用close\_plan()方法，直接关闭当前窗口。

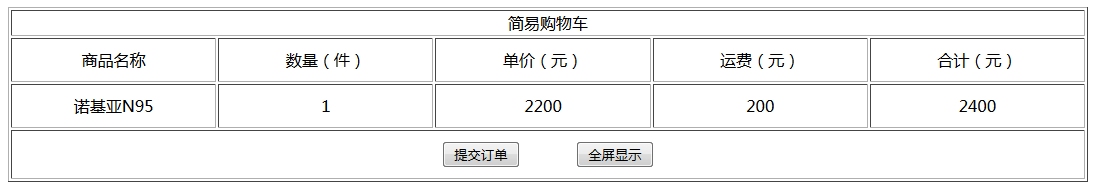
### 上机练习4

制作简单的购物车页面

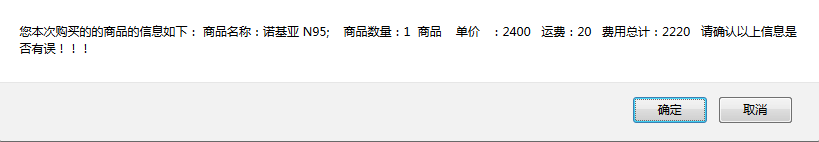
* 使用open()方法弹出窗口
* 使用alert()方法提示信息
* 使用confirm()方法进行信息确认

需求说明：

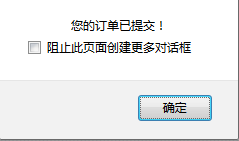
* 购物车页面有“提交订单”和“全屏显示”两个按钮，如下图所示。



* 点击“提交订单”，弹出购物车信息确认页面，如下图：



* 在确认信息页面点击“确定”按钮，提交订单成功。



* 点击全屏显示，购物车进行全屏显示。

## Date对象

时间是大家都很关注并在意的东西，所以如果我们在页面的导航栏或者其他位置显示当前的时间、星期、日期的话，会给浏览者提供很多遍历，同时也会为网页增加特色。

既然要在网页上显示时间，那么就涉及到日期对象Date。Date对象就是对系统的内置对象，在JavaScript中，系统的内置对象并不止Date一个，还有Array对象，String对象和Math对象等。

* Array:用于在单独的变量名中存储一系列的值
* String：用于支持对字符串的处理
* Math：使我们有能力执行常用的数学任务，它包含了若干个数字常量和函数
* Date：用于操作日期和时间

JavaScript中的对象与Java中的类非常相似，需要使用“new 对象名()”的方法创建一个实例，并且JavaScript中的对象都有一套属性和方法进行各种操作，

Date对象包含日期和时间两个信息，使用var和new两个关键字创建一个日期实例，语法如下：

|  |
| --- |
| **var** 日期实例=**new** Date(参数); |

* 日期实例是存储Date对象的变量。
* 可以省略参数，如果没有参数，则表示当前日期和时间，例如：

|  |
| --- |
| **var** today=**new** Date();//将当前日期和时间存储在变量today中 |

* 参数是字符串格式“MM DD,YYYY,hh:mm:ss”，表示日期和时间，例如：

|  |
| --- |
| **var** tdate=**new** Date("september 1,2016,14:58:12"); |

Date对象没有属性，但有大量用于设置，获取和操作日期的方法，从而实现在页面中显示不同类型的日期时间，Date对象有两个方法组来对日期进行操作，方法如下表所示：

|  |  |
| --- | --- |
| **Date方法分组** | **说 明** |
| setXxx | 这些方法用于设置时间和日期值 |
| getXxx | 这些方法用于获取时间和日期值 |

Date对象方法的参数值及其对应的整数如下表所示：

|  |  |
| --- | --- |
| **值** | **整数** |
| Seconds 和 Minutes | 0 至 59 |
| Hours | 0 至 23 |
| Day | 0 至 6（星期几） |
| Date | 1 至 31（月份中的天数） |
| Months | 0 至 11（一月至十二月） |

Date对象的get分组的方法如下表所示：

|  |  |  |  |
| --- | --- | --- | --- |
| **方法** | | **描述** | |
| [getDate()](http://www.runoob.com/jsref/jsref-getdate.html) | | 从 Date 对象返回一个月中的某一天 (1 ~ 31)。 | |
| [getDay()](http://www.runoob.com/jsref/jsref-getday.html) | | 从 Date 对象返回一周中的某一天 (0 ~ 6)。 | |
| [getFullYear()](http://www.runoob.com/jsref/jsref-getfullyear.html) | | 从 Date 对象以四位数字返回年份。 | |
| [getHours()](http://www.runoob.com/jsref/jsref-gethours.html) | | 返回 Date 对象的小时 (0 ~ 23)。 | |
| [getMinutes()](http://www.runoob.com/jsref/jsref-getminutes.html) | | 返回 Date 对象的分钟 (0 ~ 59)。 | |
| [getMonth()](http://www.runoob.com/jsref/jsref-getmonth.html) | | 从 Date 对象返回月份 (0 ~ 11)。 | |
| [getSeconds()](http://www.runoob.com/jsref/jsref-getseconds.html) | | 返回 Date 对象的秒数 (0 ~ 59)。 | |
| [getTime()](http://www.runoob.com/jsref/jsref-gettime.html) | | 返回 1970 年 1 月 1 日至今的毫秒数。 | |

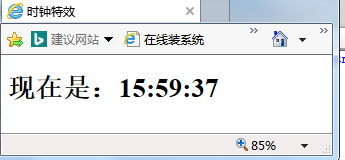
下面通过完成时钟特效的实例来掌握Date对象的方法，在时钟特效例子中只显示当前时间的小时、分和秒。

示例4：

|  |
| --- |
| <html xmlns=*"http://www.w3.org/1999/xhtml"*>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=utf-8"* />  <title>时钟特效</title>  <script type=*"text/javascript"*>  **function** disptime()  {  **var** today = **new** Date(); //获得当前时间  **var** hh = today.getHours(); //获得小时、分钟、秒  **var** mm = today.getMinutes();  **var** ss = today.getSeconds();  document.getElementById("myclock").innerHTML="<h1>现在是："+hh+":"+mm+":"+ss+"</h1>";  }  </script>  </head>  <body>  <body onload="disptime()">  <div id=*"myclock"*></div>  </body>  </html> |

* 首先创建一个日期实例today。
* 使用Date对象的getHours()方法和getMinutes()和getSeconds()方法获取当前时间的小时、分和秒。
* 把当前时间显示在id为myclock中。
* innerHTML属性设置或返回id为myclock的div元素的开始和结束标签之间的HTML。
* document.getElementById("myclock")表示引用id为myclock的div元素。

效果如下图：



遗憾的是示例4的事件不能动态显示，只能显示打开网页时的那一刻的时间，时间是静止的，不能动态更新，若要像电子表一样能不停的动态改变，应该怎么办呢？

## 定时函数

### 常用的定时函数

* + - 1. **setTimeout( )方法**

setTimeout( )用于在指定的毫秒后调用函数或计算表达式，语法格式如下：

|  |
| --- |
| setTimeout("调用的函数", "指定的时间后"); |

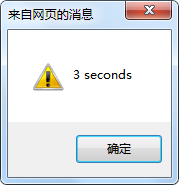
下面的例子使用了定时函数setTimeout( )，代码如下：

|  |
| --- |
| <html xmlns=*"http://www.w3.org/1999/xhtml"*>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=utf-8"* />  <title>定时函数的应用</title>  <script type=*"text/javascript"*>  **function** timer(){  **var** t=setTimeout("alert('3 seconds')",3000);  }  </script>  </head>  <body>  <input name=*"s"* type=*"button"* value=*"显示消息"* onclick="timer()"/>  </body>  </html> |

3000表示3000毫秒，即3秒

单击按钮调用timer()函数时，弹出一个警示框，由于使用了setTimeout()函数，因此调用timer()后，需要等待3秒钟，才能弹出警示框。

下图为在浏览器单击后按钮，等待3秒钟后的运行结果：



* + - 1. **setInterval()方法**

setInterval()方法可按照指定的周期来调用函数或计算表达式，语法格式如下：

|  |
| --- |
| setInterval("调用的函数", "指定的时间间隔"); |

setInterval()会不停地调用函数，直到窗口被关闭或其他方式强制停止。修改上面的例子把setTimeout()函数改成使用setInterval()函数，示例代码如下：

|  |
| --- |
| <html xmlns=*"http://www.w3.org/1999/xhtml"*>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=utf-8"* />  <title>定时函数的应用</title>  <script type=*"text/javascript"*>  **function** timer(){  **var** t=setInterval("alert('3 seconds')",3000);  }  </script>  </head>  <body>  <input name=*"s"* type=*"button"* value=*"显示消息"* onclick="timer()"/>  </body>  </html> |

在浏览器中运行，单击按钮“显示消息”，等待3秒钟后，弹出消息框，关闭此消息框后，间隔3秒钟后又会弹出此消息框，并且只要弹出的消息框关掉，3秒钟后就会再次弹出此消息框。

修改示例4时钟特效，增加setInterval()函数使时钟动起来，代码如下：

示例5：

|  |
| --- |
| <html xmlns=*"http://www.w3.org/1999/xhtml"*>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=utf-8"* />  <title>时钟特效</title>  <script type=*"text/javascript"*>  **function** disptime()  {  **var** today = **new** Date(); //获得当前时间  **var** hh = today.getHours(); //获得小时、分钟、秒  **var** mm = today.getMinutes();  **var** ss = today.getSeconds();  document.getElementById("myclock").innerHTML="<h1>现在是："+hh+":"+mm+":"+ss+"</h1>";  }  **var** myTime=setInterval("disptime()",1000);//使用setInterval()每间隔指定毫秒后调用disptime()  </script>  </head>  <body>  <body onload="disptime()">  <div id=*"myclock"*></div>  </body>  </html> |

在浏览器中运行，时钟已经动起来了，达到了真正的时钟特效。

## 课后总结

* JS 由三部分组成：
* ECMAScript：也叫解释器，充当翻译角色，这是 JS 的核心部分。
* DOM：文档对象模型（Document Object Model）。DOM 赋予了 JS 操作 HTML 的能力，即 document 操作。
* BOM：浏览器对象模型（Browser Object Model）。BOM 赋予了 JS 操作浏览器的能力，即 window 操作。
* 在HTML页面中引用js的三种方式：直接把JavaScript代码写在标签<script>和</script>之间；使用外部JavaScript文件或直接把简短的JavaScript代码写在HTML标签中。
* JavaScript的核心语法有变量的声明和赋值、数据类型、逻辑控制语句，注释，在JavaScript中代码区分大小写。并且建议每一句的末尾使用分号结束。
* 在JavaScript中常用的输入、输出是prompt()方法和alert()方法。
* 使用alert()方法调试程序。
* 函数分为系统函数和自定义函数；自定义函数需要先创建，然后才能调用。
* 自定义函数分为有参函数和无参函数。
* 使用window对象可以实现弹出窗口、关闭当前窗口、弹出页面消息框等效果。
* Date对象可以获得当前系统日期，时间。
* 使用定时函数与Date对象可以制作时钟特效。

## 课后作业

1. **描述prompt()、alert()和confrim()三者的区别，举例说明。**

2.当用户输入的值大于5时，在页面上输出一个倒三角形，例如输入8时，在页面上输出如图1所示的倒三角形，当输入值小于或等于5时，在页面上输出倒正三角形，例如输出4时，在页面上输出如图2所示的倒正三角形。

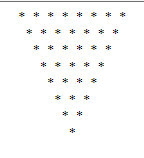
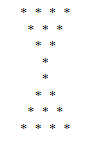
 

图1 图2

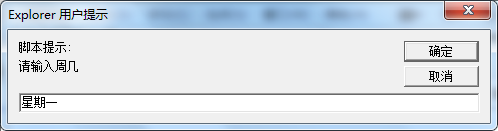
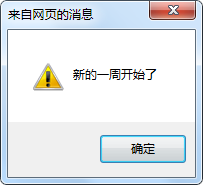
3.使用JavaScript脚本在页面上输出一个正方形，要求如下：

* 使用prompt()方法输入正方形的行数
* 在页面上输出正方形，如下图所示：

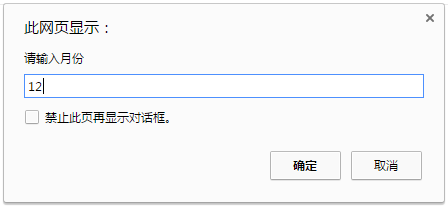
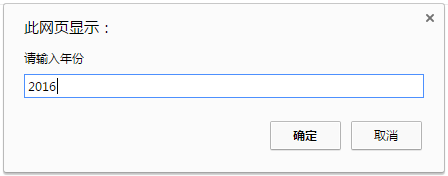


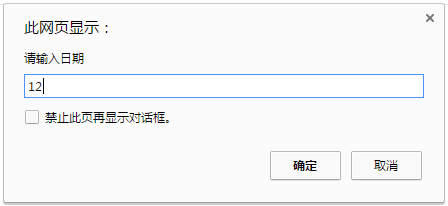
4.prompt()方法在页面中弹出提示，根据用户输入星期一至星期日的不同，弹出不同的信息提示框，要求如下：

* 输入“星期一”时，弹出“新的一周开始了”。
* 输入“星期二”，“星期三”，“星期四”时，弹出“努力工作”。
* 输入“星期五”时，弹出“明天就是周末了”。
* 输入其他内容弹出“放松的休息”。

5.prompt()方法在页面中弹出提示，根据用户输入年份，月份和天数的不同，计算出是输入年份的第几天（注意闰年和平年的计算）





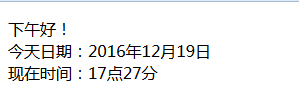
6.制作显示年、月、日，显示星期几，显示时钟特效，时钟为12小时进制

实现思路

* 使用getFullYear()获得当前年份
* 使用getMonth()+1获得当前月份
* 使用getDate()获得当前日期
* 根据getHours()获得的小时，使用if语句判断当前时间是否大于12
* 使用getDay()获取当前表示星期几的数字，然后使用switch设置当前星期几

7.根据Date对象获取当前的日期和时间，根据不同时间显示不同的问候语

* 如果当前时间小于12点（含），显示上午好
* 如果当前时间大于12点，小于18点（含），显示下午好
* 如果当前时间大于18点，显示晚上好



# 正则表达式及表单验证

## 章节目标

* 理解表单验证的意义
* 掌握常用的验证方法
* 掌握String对象的属性方法
* 掌握数字验证
* 掌握正则表达式

## 表单验证的意义



表单验证是web/手机浏览者与界面交互时重要的一个核心沟通流程。它的重要性远超过我们所认识到的，不相信我？试想一下你经常遇到的需要表单验证的情形:

* 注册/登录表单
* 购物车 - 支付表单
* 邮件表单

我们会经常与表单打交道，那么用户表单中的数据是需要提交到服务器的，那么用户数据的有效性是不是需要考虑到呢？也就是说，我们需要验证用户数据，那么在哪里验证呢？是服务器吗？如果用户数量小的话，完全是可以的。但假如用户数量太过于庞大呢？这样服务器压力无疑太大了！那需要怎么办呢？我们可以通过js进行表单验证！

## 常用的表单验证方法

### String对象方法

我们所获取的表单数据都是String类型的，那么使用String对象的属性和方法可以帮助我们进行验证：

* 属性：
* length：字符串长度
* 方法：
* indexOf方法放回String对象内第一次出现子字符串位置。如果没有找到子字符串，则返回-1。

格式：strObj.indexOf(substr[,startIndex])

说明： substr要在String对象中查找的子字符串。 startIndex该整数值指出在String对象内开始查找的索引。如果省略，则从字符串的开始处查找。

例如：

var str = "ABCDECDF";

str.indexOf("CD"，1); // 由1位置从左向右查找 123...

结果：2

* lastIndexOf方法返回String对象中字符串最后出现的位置。如果没有匹配到子字符串，则返回-1。

格式：strObj.lastIndexOf(substr[,startindex])

说明： substr要在String对象内查找的子字符串。 startindex该整数值指出在String对象内进行查找的开始索引位置。如果省略，则查找从字符串的末尾开始。

例如：

var str = "ABCDECDF";

str.lastIndexOf("CD",6); // 由6位置从右向左查找 ...456

结果：5

* charAt方法返回指定索引位置处的字符。如果超出有效范围的索引值返回空字符串。

strObj.charAt(index)

说明： index想得到的字符的基于零的索引。有效值是0与字符串长度减一之间的值。

例如：

var str = "ABC";

str.charAt(1);

结果：B

* charCodeAt方法返回一个整数，代表指定位置字符的Unicode编码。

strObj.charCodeAt(index)

说明：

index将被处理字符的从零开始计数的编号。有效值为0到字符串长度减1的数字。

如果指定位置没有字符，将返回NaN。

例如：

var str = "ABC";

str.charCodeAt(0);

结果：65

* substring方法返回位于String对象中指定位置的子字符串。

strObj.substring(start,end)

说明：

start指明子字符串的起始位置，该索引从0开始起算。

end指明子字符串的结束位置，该索引从0开始起算。

substring方法使用start和end两者中的较小值作为子字符串的起始点。如果start或end为NaN或者为负数，那么将其替换为0。

例如：

012345

var str = "ABCDEF";

str.substring(2,4); // 或 str.substring(4,2);

结果：CD

* slice方法返回字符串的片段。

strObj.slice(start[,end])

说明：

start下标从0开始的strObj指定部分其实索引。如果start为负，将它作为length+start处理，此处length为字符串的长度。

end小标从0开始的strObj指定部分结束索引。如果end为负，将它作为length+end处理，此处length为字符串的长度。

例如：

012345

var str = "ABCDEF";

str.slice(2,4);

结果：CD

* split(separator) 将一个字符串分割为子字符串，然后将结果作为字符串数组返回。

格式：strObj.split([separator[,limit]])

说明：

separator字符串或 正则表达式 对象，它标识了分隔字符串时使用的是一个还是多个字符。如果忽略该选项，返回包含整个字符串的单一元素数组。

limit该值用来限制返回数组中的元素个数。

例如：

var str = "AA BB CC DD EE FF";

alert(str.split(" "，3));

结果：

AA,BB,CC

* 以下是一个使用到String对象表单验证的例子：

|  |
| --- |
| <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  <html xmlns=*"http://www.w3.org/1999/xhtml"*>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=gb2312"* />  <title>百度注册页面</title>  <style type=*"text/css"*>  \*{padding:*0*; margin:*0*;}  **body**{  margin-top: *0px*;  margin-right: *auto*;  margin-bottom: *0px*;  margin-left: *auto*;  font-size:*12px*;  line-height:*30px*;  }  *.center*{  float:*none*;  width: *503px*;  clear:*both*;  margin-top: *0px*;  margin-right: *auto*;  margin-bottom: *0px*;  margin-left: *auto*;  }  *.inputs* {  border: *1px solid #333*;  width:*120px*;  float:*left*;  }  *.left*{width:*120px*;  text-align:*right*;  padding-right:*5px*;  }  *.red*{  color:*#ff0000*;  padding-left:*10px*;  font-size:*12px*;  }  </style>  <script type=*"text/javascript"*>  **function** $(ElementId){  **return** document.getElementById(ElementId);  }  **function** check(){  **var** user=$("user");  **var** userId=$("userId");  userId.innerHTML="";  **if**(user.value.length==0){  userId.innerHTML="用户名不能为空";  **return** **false**;  }  **if**(user.value.length<4 || user.value.length>12){  userId.innerHTML="用户名长度为在4―12字符";  **return** **false**;  }  **for**(**var** i=0;i<user.value.length;i++){  **var** **char**=user.value.toLowerCase().charAt(i);  **if**((!(**char**>=0 && **char**<=9)) && (!(**char**>='a'&& **char**<='z'))&&**char**!='\_'){  userId.innerHTML="用户名必须由字母、数字和下划线组成";  **return** **false**;  }  }  **var** pwd=$("pwd");  **var** pwdId=$("pwdId");  pwdId.innerHTML="";  **if**(pwd.value==""){  pwdId.innerHTML="密码不能为空";  **return** **false**;  }  **if**(pwd.value.length<6 || pwd.value.length>12){  pwdId.innerHTML="密码长度为在6―12字符";  **return** **false**;  }  **var** repwd=$("repwd");  **var** repwdId=$("repwdId");  repwdId.innerHTML="";  **if**(pwd.value!=repwd.value){  repwdId.innerHTML="两次输入的密码不一致";  **return** **false**;  }  **var** sexId=$("sexId");  sexId.innerHTML="";  **var** j=0;  **var** sex=document.getElementsByName("sex");  **for**(**var** i=0;i<sex.length;i++){  **if**(sex[i].checked==**true**){  j=1;  **break**;  }  }  **if**(j==0){  sexId.innerHTML="请选择性别";  **return** **false**;  }  **var** mail=$("email");  **var** emailId=$("emailId");  emailId.innerHTML="";  **if**(mail.value==""){//检测Email是否为空  emailId.innerHTML="电子邮件地址不能为空";  **return** **false**;  }  **if**(mail.value.indexOf("@")==-1){  emailId.innerHTML="电子邮件地址格式不正确\n必须包含@";  **return** **false**;  }  **if**(mail.value.indexOf(".")==-1){  emailId.innerHTML="电子邮件地址格式不正确\n必须包含.";  **return** **false**;  }  **return** **true**;  }  </script>  </head>  <body>  <div class=*"center"*><img src=*"images/header1.jpg"* /></div>  <div class=*"center"*>  <table width=*"100%"* border=*"0"* cellspacing=*"0"* cellpadding=*"0"*>  <form action=*"success.html"* method=*"post"* name=*"myform"* onsubmit="return check()">  <tr>  <td class=*"left"*>用户名：</td>  <td><input id=*"user"* type=*"text"* class=*"inputs"* onblur="check()"/><div id=*"userId"* class=*"red"*></div></td>  </tr>  <tr>  <td class=*"left"*>密码：</td>  <td><input id=*"pwd"* type=*"password"* class=*"inputs"* onblur="check()"/><div id=*"pwdId"* class=*"red"*></div></td>  <tr>  <td class=*"left"*>确认密码：</td>  <td><input id=*"repwd"* type=*"password"* onblur="check()" class=*"inputs"*/><div id=*"repwdId"* class=*"red"*></div></td>  </tr>  <tr>  <tr>  <td class=*"left"*>性别：</td>  <td><div style="float:*left*;"><input name=*"sex"* type=*"radio"* value=*"男"* />男 <input name=*"sex"* type=*"radio"* value=*"女"* />女</div>  <div id=*"sexId"* class=*"red"*></div></td>  </tr>  <tr>  <td class=*"left"*>电子邮件地址</td>  <td><input id=*"email"* type=*"text"* class=*"inputs"* onblur="check()"/><div id=*"emailId"* class=*"red"*></div></td>  </tr>  <tr>  <td class=*"left"*>出生日期：</td>  <td><select id=*"year"*>  <script type=*"text/javascript"*>  **for**(**var** i=1900;i<=2009;i++){  document.write("<option value="+i+">"+i+"</option>");  }  </script>  </select>年  <select id=*"month"*>  <option checked=*"checked"*>1</option>  <option checked=*"checked"*>2</option>  <option checked=*"checked"*>3</option>  <option checked=*"checked"*>4</option>  <option checked=*"checked"*>5</option>  <option checked=*"checked"*>6</option>  <option checked=*"checked"*>7</option>  <option checked=*"checked"*>8</option>  <option checked=*"checked"*>9</option>  <option checked=*"checked"*>10</option>  <option checked=*"checked"*>11</option>  <option checked=*"checked"*>12</option>  </select>月  <select id=*"day"*>  <option checked=*"checked"*>1</option>  <option checked=*"checked"*>2</option>  <option checked=*"checked"*>3</option>  <option checked=*"checked"*>4</option>  <option checked=*"checked"*>5</option>  <option checked=*"checked"*>6</option>  <option checked=*"checked"*>7</option>  <option checked=*"checked"*>8</option>  <option checked=*"checked"*>9</option>  <option checked=*"checked"*>10</option>  <option checked=*"checked"*>11</option>  <option checked=*"checked"*>12</option>  </select>日</td>  </tr>  <tr>  <td>&nbsp;</td>  <td><input name=*"sub"* type=*"submit"* value=*"注册"* /> <input name=*"cancel"* type=*"reset"* value=*"清除"* /></td>  </tr>  </form>  </table>  </div>  </body>  </html> |

### 数字验证方法

数字验证使用方法isNaN(s)，来验证s是否是合法的数字。

isNaN() 函数通常用于检测 parseFloat() 和 parseInt() 的结果，以判断它们表示的是否是合法的数字。

示例：

isNaN(123);

isNaN(-1.23);

isNaN(5-2);

isNaN(0);

isNaN("Hello");

isNaN("2005/12/12");

### 正则表达式

* 概念：使用单个字符串来描述、匹配一系列符合某个句法规则的字符串。
* 语法格式：
* 常用元字符：

|  |  |  |  |
| --- | --- | --- | --- |
| 符号 | 等价类 | | 说明 |
| . | [^\n\r] | | 匹配除换行符以外的任意字符 |
| \d | [0-9] | | 匹配数字字符 |
| \D | [^0-9] | | 非数字字符 |
| \s | [\t\n\x0B\f\r] | | 匹配任意的空白符 |
| \S | [^\t\n\x0B\f\r] | | 非空白字符 |
| \w | [a-zA-Z\_0-9] | | 匹配字母或数字或下划线 |
| \W | [^a-zA-Z\_0-9] | | 非单词字符 |
| \b | | 匹配单词的开始或结束 | |
| ^ | | 匹配字符串的开始 | |
| $ | | 匹配字符串的结束 | |

* 常用限定符：

|  |  |
| --- | --- |
| \* | 重复零次或更多次 |
| + | 重复一次或更多次 |
| ? | 重复零次或一次 |
| {n} | 重复n次 |
| {n,} | 重复n次或更多次 |
| {n,m} | 重复n到m次 |

* 转义字符：

如果你想查找元字符本身的话，比如你查找.,或者\*,就出现了问题：你没办法指定它们，因为它们会被解释成别的意思。这时你就得使用\来取消这些字符的特殊意义。因此，你应该使用\.和\\*。当然，要查找\本身，你也得用\\。

* 用法：

var reg = /正则表达式/;

var r = new RegExp(reg);//此句话可写可不写

r.test(str);

或者

var reg = /正则表达式/;

reg.test(str);

* 应用

var pattern = /规则/;

if(pattern.test(验证字符串)==true){

alert("符合要求！");

return true;

}else{

alert("格式非法！");

return false;

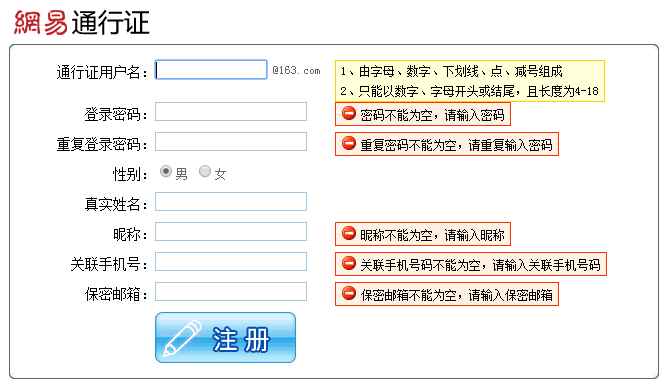
}

* 以下是一个使用到正则表达式进行表单验证的例子：

|  |
| --- |
| <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  <html xmlns=*"http://www.w3.org/1999/xhtml"*>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=utf-8"* />  <title>无标题文档</title>  <style type=*"text/css"*>  **div,ul,li,ol,dl,img**{margin:*0px*;padding:*0px*}  *#all*{width:*706px*;height:*344px*; background-image:*url(images2/backbg.jpg)*}  *#login*{float:*right*;width:*340px*;height:*90px*;margin-top:*140px*}  </style>  <script language=*"javascript"*>  **function** yan(){  **var** use=document.getElementById("use");  **var** pass=document.getElementById("pass");  **var** re= /[a-zA-Z0-9]{6,16}/;  **if**(use.value.length==0){  alert("用户名不能为空");  **return** **false**;  }**else** **if**(use.value.length<6){  alert("用户名长度不能小于6");  **return** **false**;  }**else** **if**(!re.test(use.value)){  alert("用户名必须是字母或数字");  **return** **false**;  }  **if**(pass.value.length==0){  alert("密码不能为空");  **return** **false**;  }**else** **if**(pass.value.length<6){  alert("密码长度不能小于6");  **return** **false**;  }  }  </script>  </head>  <body>  <div id=*"all"*>  <div id=*"login"*>  <form method=*"post"*action=*"success.html"* onsubmit="return yan();" >  <table>  <tr>  <td>用户名：</td>  <td><input type=*"text"* id=*"use"*/></td>  </tr>  <tr>  <td>密码：</td>  <td><input type=*"password"*/ id=*"pass"*></td>  </tr>  <tr>  <td colspan=*"2"* align=*"center"*><input type=*"submit"* value=*"登录"* /></td>  </tr>  </table>  </form>  </div>  </div>  </body>  </html> |

## 课后作业

### 网易邮箱注册页面及表单验证



|  |
| --- |
| Html代码：  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  <html xmlns=*"http://www.w3.org/1999/xhtml"*>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=gb2312"* />  <title>新用户注册页面</title>  <link type=*"text/css"* rel=*"stylesheet"* href=*"css/register.css"* />  <script type=*"text/javascript"* src=*"js/register.js"*></script>  </head>  <body>  <div id=*"header"*><img src=*"images/register\_logo.gif"* alt=*"logo"*/></div>  <div id=*"main"*>  <table width=*"100%"* border=*"0"* cellspacing=*"0"* cellpadding=*"0"*>  <tr>  <td class=*"bg bg\_top\_left"*></td>  <td class=*"bg\_top"*></td>  <td class=*"bg bg\_top\_right"*></td>  </tr>  <tr>  <td class=*"bg\_left"*></td>  <td class=*"content"*>  <form action=*"success.html"* method=*"post"* name=*"myform"* onsubmit="return checkForm()">  <dl>  <dt>通行证用户名：</dt>  <dd><input type=*"text"* id=*"userName"* class=*"inputs userWidth"* onfocus="userNameFocus()" onblur="userNameBlur()" /> @163.com</dd>  <div id=*"userNameId"*></div>  </dl>  <dl>  <dt>登录密码：</dt>  <dd><input type=*"password"* id=*"pwd"* class=*"inputs"* onfocus="pwdFocus()" onblur="pwdBlur()"/></dd>  <div id=*"pwdId"*></div>  </dl>  <dl>  <dt>重复登录密码：</dt>  <dd><input type=*"password"* id=*"repwd"* class=*"inputs"* onblur="repwdBlur()"/></dd>  <div id=*"repwdId"*></div>  </dl>  <dl>  <dt>性别：</dt>  <dd><input name=*"sex"* type=*"radio"* value=*""* checked=*"checked"*/>男 <input name=*"sex"* type=*"radio"* value=*""* />女 </dd>  </dl>  <dl>  <dt>真实姓名：</dt>  <dd><input type=*"text"* id=*"realName"* class=*"inputs"* /></dd>  </dl>  <dl>  <dt>昵称：</dt>  <dd><input type=*"text"* id=*"nickName"* class=*"inputs"* onfocus="nickNameFocus()" onblur="nickNameBlur()"/></dd>  <div id=*"nickNameId"*></div>  </dl>  <dl>  <dt>关联手机号：</dt>  <dd><input type=*"text"* id=*"tel"* class=*"inputs"* onfocus="telFocus()" onblur="telBlur()" /></dd>  <div id=*"telId"*></div>  </dl>  <dl>  <dt>保密邮箱：</dt>  <dd><input type=*"text"* id=*"email"* class=*"inputs"* onfocus="emailFocus()" onblur="emailBlur()" /></dd>  <div id=*"emailId"*></div>  </dl>  <dl>  <dt></dt>  <dd><input name=*" "* type=*"image"* src=*"images/button.gif"*/></dd>  </dl>  </form>  </td>  <td class=*"bg\_right"*></td>  </tr>  <tr>  <td class=*"bg bg\_end\_left"*></td>  <td class=*"bg\_end"*></td>  <td class=*"bg bg\_end\_right"*></td>  </tr>  </table>  </div>  </body>  </html>  JS代码：  **function** $(elementId){  **return** document.getElementById(elementId);  }    **function** userNameFocus(){  **var** userNameId=$("userNameId");  userNameId.className="import\_prompt";  userNameId.innerHTML="1、由字母、数字、下划线、点、减号组成<br/>2、只能以数字、字母开头或结尾，且长度为4-18";  }    **function** userNameBlur(){  **var** userName=$("userName");  **var** userNameId=$("userNameId");  **var** reg=/^[0-9a-zA-Z][0-9a-zA-Z\_.-]{2,16}[0-9a-zA-Z]$/;  **if**(userName.value==""){  userNameId.className="error\_prompt";  userNameId.innerHTML="通行证用户名不能为空，请输入通行证用户名";  **return** **false**;  }  **if**(reg.test(userName.value)==**false**){  userNameId.className="error\_prompt";  userNameId.innerHTML="1、由字母、数字、下划线、点、减号组成<br/>2、只能以数字、字母开头或结尾，且长度为4-18";  **return** **false**;  }  userNameId.className="ok\_prompt";  userNameId.innerHTML="通行证用户名输入正确";  **return** **true**;  }  **function** pwdFocus(){  **var** pwdId=$("pwdId");  pwdId.className="import\_prompt";  pwdId.innerHTML="密码长度为6-16";  }  **function** pwdBlur(){  **var** pwd=$("pwd");  **var** pwdId=$("pwdId");  **if**(pwd.value==""){  pwdId.className="error\_prompt";  pwdId.innerHTML="密码不能为空，请输入密码";  **return** **false**;  }  **if**(pwd.value.length<6 || pwd.value.length>16){  pwdId.className="error\_prompt";  pwdId.innerHTML="密码长度为6-16";  **return** **false**;  }  pwdId.className="ok\_prompt";  pwdId.innerHTML="密码输入正确";  **return** **true**;  }  **function** repwdBlur(){  **var** repwd=$("repwd");  **var** pwd=$("pwd");  **var** repwdId=$("repwdId");  **if**(repwd.value==""){  repwdId.className="error\_prompt";  repwdId.innerHTML="重复密码不能为空，请重复输入密码";  **return** **false**;  }  **if**(repwd.value!=pwd.value){  repwdId.className="error\_prompt";  repwdId.innerHTML="两次输入的密码不一致，请重新输入";  **return** **false**;  }  repwdId.className="ok\_prompt";  repwdId.innerHTML="两次密码输入正确";  **return** **true**;  }  **function** nickNameFocus(){  **var** nickNameId=$("nickNameId");  nickNameId.className="import\_prompt";  nickNameId.innerHTML="1、包含汉字、字母、数字、下划线以及@!#$%&\*特殊字符<br/>2、长度为4－20个字符<br/>3、一个汉字占两个字符";  }    **function** nickNameBlur(){  **var** nickName=$("nickName");  **var** nickNameId=$("nickNameId");  **var** k=0;  **var** reg=/^([\u4e00-\u9fa5]|\w|[@!#$%&\*])+$/;  **var** chinaReg=/[\u4e00-\u9fa5]/g;  **if**(nickName.value==""){  nickNameId.className="error\_prompt";  nickNameId.innerHTML="昵称不能为空，请输入昵称";  **return** **false**;  }  **if**(reg.test(nickName.value)==**false**){  nickNameId.className="error\_prompt";  nickNameId.innerHTML="只能由汉字、字母、数字、下划线以及@!#$%&\*特殊字符组成";  **return** **false**;  }    **var** len=nickName.value.replace(chinaReg,"ab").length;  **if**(len<4||len>20){  nickNameId.className="error\_prompt";  nickNameId.innerHTML="1、长度为4－20个字符<br/>2、一个汉字占两个字符";  **return** **false**;  }    nickNameId.className="ok\_prompt";  nickNameId.innerHTML="昵称输入正确";  **return** **true**;  }  **function** telFocus(){  **var** telId=$("telId");  telId.className="import\_prompt";  telId.innerHTML="1、手机号码以13，15，18开头<br/>2、手机号码由11位数字组成";  }    **function** telBlur(){  **var** tel=$("tel");  **var** telId=$("telId");  **var** reg=/^(13|15|18)\d{9}$/;  **if**(tel.value==""){  telId.className="error\_prompt";  telId.innerHTML="关联手机号码不能为空，请输入关联手机号码";  **return** **false**;  }  **if**(reg.test(tel.value)==**false**){  telId.className="error\_prompt";  telId.innerHTML="关联手机号码输入不正确，请重新输入";  **return** **false**;  }  telId.className="ok\_prompt";  telId.innerHTML="关联手机号码输入正确";  **return** **true**;  }  **function** emailFocus(){  **var** emailId=$("emailId");  emailId.className="import\_prompt";  emailId.innerHTML="请输入您常用的电子邮箱";  }      **function** emailBlur(){  **var** email=$("email");  **var** emailId=$("emailId");  **var** reg=/^\w+@\w+(\.[a-zA-Z]{2,3}){1,2}$/;  **if**(email.value==""){  emailId.className="error\_prompt";  emailId.innerHTML="保密邮箱不能为空，请输入保密邮箱";  **return** **false**;  }  **if**(reg.test(email.value)==**false**){  emailId.className="error\_prompt";  emailId.innerHTML="保密邮箱格式不正确，请重新输入";  **return** **false**;  }  emailId.className="ok\_prompt";  emailId.innerHTML="保密邮箱输入正确";  **return** **true**;  }  **function** checkForm(){  **var** flagUserName=userNameBlur();  **var** flagPwd=pwdBlur();  **var** flagRepwd=repwdBlur();  **var** flagNickName=nickNameBlur();  **var** flagTel=telBlur();  **var** flagEmail=emailBlur();  **if**(flagUserName==**true** &&flagPwd==**true** &&flagRepwd==**true** &&flagNickName==**true**&&flagTel==**true**&flagEmail==**true**){  **return** **true**;  }  **else**{  **return** **false**;  }  } |

# Dom操作

## 章节目标

掌握获取元素的方法；

掌握获取和设置属性的方法；

掌握DOM的常用属性；

掌握动态的创建标记；

## 什么是DOM

DOM是文档对象模型(Document Object Model)，是基于浏览器编程的一套API接口，一套对文档的内容进行抽象和概念化的方法。

W3C对DOM的定义是：“一个与系统平台和编程语言无关的接口，程序和脚本可以通过这个接口动态的访问和修改文档的内容、结构和样式”。 今天，几乎所有的浏览器都内置了对DOM的支持。

DOM中的D:

文档。当创建了一个网页并把它加载到web浏览器中时，DOM就在幕后悄然而生，它把你编写的网页文档转化为一个文档对象。

DOM中的O:

对象。对象是一种自足的数据集合。与某个对象相关联的变量称为这个对象的属性；这能通过某个特定的对象去调用的函数称为这个对象的方法。本章我们重点来看一下document对象的属性和方法。

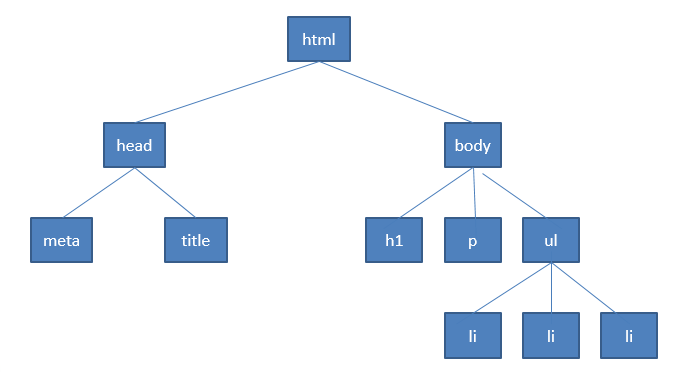
DOM中的M:

模型。含义是某种事物的表现形式。DOM代表着加载到浏览器窗口的当前网页，浏览器则提供了网页的模型，我们可以通过JavaScript去读取这个模型。

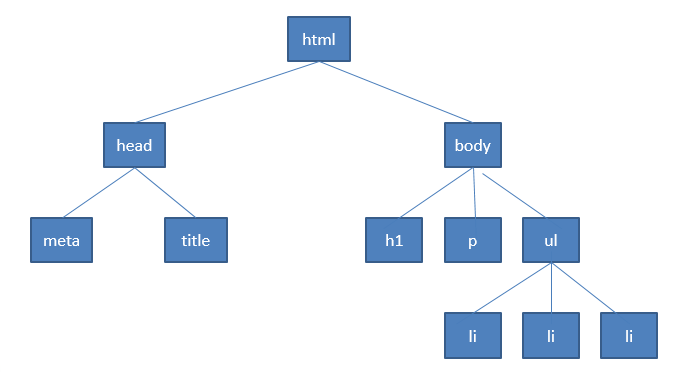
DOM把一篇文档表示成一棵树，如下：

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset=*"UTF-8"*>  <title>Insert title here</title>  </head>  <body>  <h1>你最喜欢哪些明星</h1>  <p title=*"my answer" id="answer"*>我最喜欢的明星有：</p>  <ul id=*"stars"*>  <li>范冰冰</li>  <li>刘德华</li>  <li>王宝强</li>  </ul>  </body>  </html> |

在DOCTYPE之后，一个打开了的<html>标签标识整个文档的开始，网页中的所有其他元素均包含在这个元素中，这个<html>标签就是树根。深入一层，有<head>和<body>两个分支，它们位于同一层次且互不包含，它们是兄弟关系，有着共同的父元素<html>，又有各自的子元素。<head>元素有两个子元素：<meta>和<title>。<body>元素有三个子元素：<h1>、<p>和<ul>。同样，<ul>也是一个父元素，它有三个子元素，它们都是<li>元素。



我们将这个图倒过来不就是一棵树吗：



## 获取元素的方法

有3种方法可获取元素节点，分别是通过元素ID、通过标签名字和通过类名字来获取。

### getElementByID

它是document对象特有的函数。该方法返回一个与那个有着给定id属性值的元素节点对应的对象（注意：JavaScript语言区分大小写）。既然是方法，方法名后面必须跟一对圆括号，这里圆括号包含着方法的参数。getElementByID方法只有一个参数：想要获得元素id属性的值，这个id值必须放在单引号或双引号中。



该方法调用将返回一个对象（可以用typeof操作符验证这一点）。这个对象对应着document对象里的一个独一无二的元素，那个元素的HTML id属性值是stars。

例：test.js文件中



**注意：该方法只能用于 document 对象**

文档中的每一个元素都是一个对象。利用DOM提供的方法能得到任何一个对象。

我们经常需要与表单打交道，在提交表单时，我们经常需要做很多验证，比如非空验证、长度验证、内容验证等等。进行验证的前提就是我们能够获取表单元素的值。那么怎么样获得呢？首先是要获取到这个表单元素的对象吧，这时候我们的就可以使用这个方法来完成：

|  |
| --- |
| Html文件如下：  <!DOCTYPE html>  <html>  <head>  <meta charset=*"UTF-8"*>  <title>Insert title here</title>  <script type=*"text/javascript"* src=*"Form.js"*></script>  </head>  <body>  <form action=*""*>  <input type=*"text"* id=*"name"* />  <input type=*"submit"* value=*"提交"* onclick="showValue()">  </form>  </body>  </html>  js文件如下：  **function** showValue()  {  **var** name = document.getElementById("name");  alert(name.value);  } |

### getElementsByTagName

我们知道，并不是每个元素都定义一个独一无二的id值。getElementsByTagName方法可返回一个对象数组，每个对象分别对应着文档里有着给定标签的一个元素。该方法也只有一个参数，该参数是标签的名字。（注：不要与上面的方法混淆）

例：



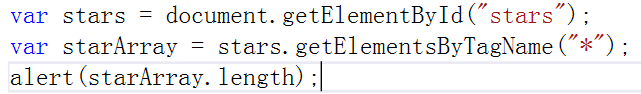
该调用返回一个对象数组，每个对象分别对应着document对象中的一个列表项元素。与其它数组一样，我们也可以利用length属性获得这个数组中元素的个数。



若想知道某份文档中总共有多少个元素节点，可以使用通配符“\*”作为参数：



还可以把上面两种方法结合起来使用，例如：我们想要获得id属性值是stars的元素包含多少个列表项，那么必须通过一个更具体的对象去调用这个方法：



### getElementsByClassName

HTML5 DOM中新增的一个方法。这个方法让我们能通过class属性中的类名来访问元素。



该方法也可以结合getElementById来使用。

### getElementsByName

getElementsByName() 方法可返回带有指定名称的对象的集合

格式：document.getElementsByName(name)

该方法与 getElementById() 方法相似，但是它查询元素的 name 属性，而不是 id 属性。

另外，因为一个文档中的 name 属性可能不唯一（如 HTML 表单中的单选按钮通常具有相同的 name 属性），所有 getElementsByName() 方法返回的是元素的数组，而不是一个元素。

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset=*"UTF-8"*>  <title>getElementsByName</title>  <script type=*"text/javascript"*>  **function** getElements()  {  **var** x=document.getElementsByName("myInput");  alert(x.length);  }  </script>  </head>  <body>  <input name=*"myInput"* type=*"text"* /><br />  <input name=*"myInput"* type=*"text"* /><br />  <input name=*"myInput"* type=*"text"* /><br />  <input type=*"button"* onclick="getElements()"  value=*"一共多少name为'myInput'的元素?"* />  </body>  </html> |

## 获取和设置属性的方法

上一小节，介绍了三种获取特定元素的方法：getElementById、getElementsByTagName、getElementsByClassName。得到需要的元素后我们就可以设法获取它的各个属性。

### getAttribute方法

getAttribute方法只有一个参数，我们要查询的属性的名字。该方法不属于document对象，只能通过元素节点对象调用。比如与getElementById方法合用：

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset=*"UTF-8"*>  <title>Insert title here</title>  <script type=*"text/javascript"*>  **function** getAttr() {  **var** answer = document.getElementById("answer");  alert(answer.getAttribute("title"));  }  </script>  </head>  <body>  <h1>你最喜欢哪些明星</h1>  <p title=*"my answer"* id=*"answer"*>我最喜欢的明星有：</p>  <ul id=*"stars"*>  <li>范冰冰</li>  <li>刘德华</li>  <li>王宝强</li>  </ul>  <input type=*"button"* value=*"获取属性"* onclick="getAttr()">  </body>  </html> |

首先通过getElementById方法获得id是“answer”的元素，然后通过getAttribute方法获得该元素的属性为“title”的值。

### setAttribute方法

上面介绍的方法都是用来获取信息，setAttribute方法允许我们用属性节点的值做修改。与getAttribute一样，setAttribute也只能用于元素节点。

格式：**element.setAttribute(attributeName,attributeValue);**

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset=*"UTF-8"*>  <title>Insert title here</title>  <script type=*"text/javascript"*>  **function** setAttr() {  **var** answer = document.getElementById("answer");  alert(answer.getAttribute("title"));  answer.setAttribute("title", "your answer");  alert(answer.getAttribute("title"));  }  </script>  </head>  <body>  <h1>你最喜欢哪些明星</h1>  <p title=*"my answer"* id=*"answer"*>我最喜欢的明星有：</p>  <ul id=*"stars"*>  <li>范冰冰</li>  <li>刘德华</li>  <li>王宝强</li>  </ul>  <input type=*"button"* value=*"设置属性"* onclick="setAttr()">  </body>  </html> |

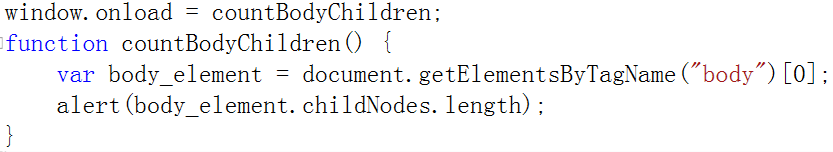
对title属性节点重新赋值为“you answer”。

注意：若要赋值的元素属性不存在，则会先创建这个属性，然后再设置它的值。SetAttribute做出的修改不会反映在文档本身的源代码里。这种“表里不一”的现象源自DOM的工作模式：先加载文档静态内容、再动态刷新，动态刷新不影响文档的静态内容（这正是DOM的真正威力：对页面内容进行刷新却不需要在浏览器里刷新页面）。

## 几个常用DOM属性

### childNodes属性

childNodes属性可以用来获取任何一个元素的所有子元素，是一个包含这个元素全部子元素的数组。例如，我们要获取某个文档的body元素的全体子元素，首先我们可以用getElementsByTagName得到body元素，每份文档只有一个body元素，所以它将是getElementsByTagName(“body”)方法中所返回数组中的第一个元素，这里，我们把它写在了一个方法中：



现在，我们想让这个函数在页面加载时执行，需要使用onload事件处理函数，在代码的末尾添加：



刷新页面，我们可以看到显示的数字和我们预期的好像并不一样。这是因为文档树的节点类型并非只有元素节点这一种。childNodes属性返回的数组包含所有类型的节点，而不仅仅是元素节点。事实上，文档中几乎每一样东西都是一个节点，甚至连空格和换行符都会被解释为节点，它们也全都包含在childNodes属性返回的数组中。所以上例中返回的结果才会比预期大。

### nodeType属性

每一个节点都有一个nodeType属性，可以使用nodeType属性来获取节点类型。nodeType属性共有12种取值，其中仅有三种较常用：

元素节点的nodeType属性值是1；

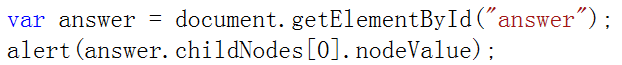
属性节点的nodeType属性值是2；

文本节点的nodeType属性值是3。



### nodeValue属性

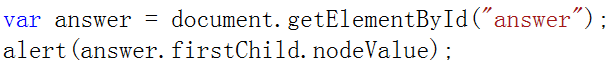
如果想改变一个文本节点的值，那就使用DOM提供的nodeValue属性，它可以用来得到（设置）一个节点的值。



首先，通过id获得p元素，p元素本身的nodeValue属性是一个空值，包含在p元素中的文本是另一种节点，它是p元素的第一个节点。因此，我们要用childNodes[0]获得第一个元素。

### firstChild、lastChild属性

数组元素childNodes[0]有个更直观易读的同义词：firstChild



更加简短，且更具有可读性。

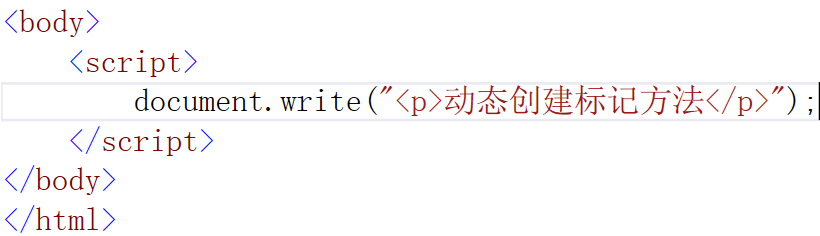
DOM还提供了一个与之对应的lastChild属性。代表childNodes数组的最后一个元素。

## 动态创建标记

### 传统方法

* **document.write**

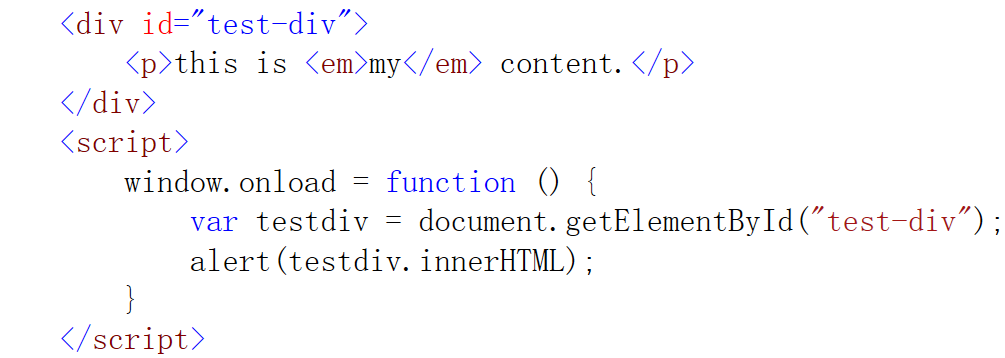
document对象的write()方法可以方便快捷的把字符串插入到文档中。



但document.write的最大缺点是它违背了“行为应该与表现分离”的原则，即使把document.write语句挪到外部函数里，也要在标记的<body>部分使用<script>标签才能调用那个函数。这样就无法享受到把行为和结构分离开来的好处。将结构、样式和行为分开永远都是一个好的做法。最好用外部的JavaScript文件去控制网页的行为。

* **innerHTML属性**

如今的浏览器几乎都支持innerHTML属性，这个属性并不是W3C DOM标准的组成部分，但现已包含到HTML5规范中。InnerHTML属性可以用来读、写给定元素的HTML内容。



当需要把一大段HTML内容插入网页时，innerHTML属性更适用，它既支持读取，又支持写入。使用innerHTML属性可以把JavaScript代码从标记中分离出来。innerHTML属性也是HTML专业属性，不能用于其他任何标记语言文档，浏览器在呈现正宗的XHTML文档（MIME类型是application/xhtml+xml的XHTML文档）时会直接忽略掉innerHTML属性。

任何时候，标准的DOM都可以用来替代innerHTML，这往往需要编写一些额外的代码才能达到同样的效果，但DOM同时提供了更高的精确性和更强大的功能。

### DOM方法

使用getElementById和getElementsByTagName等方法可以把文档结构和内容的信息检索出来，使用setAttribute方法可以改变某个节点的属性，当并未改变文档的物理内容。浏览器实际显示的是那颗DOM节点树，在浏览器看来DOM节点树才是文档，那么如果想在节点树上添加内容，就必须插入新的节点。如果想添加一些标记到文档，就必须插入元素节点。

* **createElement方法**

创建新的元素。



方法的返回值：是一个指向新建节点的引用指针。返回值是一个元素节点，所以它的 nodeType 属性值等于 1。

**新元素节点不会自动添加到文档里**，新节点没有 nodeParent 属性，它只是一个存在于 JavaScript 上下文的对象.

这样就创建出了p元素，它同其他节点一样有了自己的DOM属性。可以验证它的nodeType和nodeName值：

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset=*"UTF-8"*>  <title>Insert title here</title>  <script type=*"text/javascript"*>  **function** create() {  **var** para = document.createElement("p");  **var** info ="nodeName";  info+=para.nodeName;  info+="nodeType";  info+=para.nodeType;  alert(info);  }  </script>  </head>  <body>  <h1>你最喜欢哪些明星</h1>  <p title=*"my answer"* id=*"answer"*>我最喜欢的明星有：</p>  <ul id=*"stars"*>  <li>范冰冰</li>  <li>刘德华</li>  <li>王宝强</li>  </ul>  <input type=*"button"* value=*"创建元素"* onclick="create()">  </body>  </html> |

* **createTextNode方法**

创建一个包含着给定文本的新文本节点。这个方法的返回值是一个指向新建文本节点引用指针。

格式：var reference = document.createTextNode(text);

方法只有一个参数：新建文本节点所包含的文本字符串

方法的返回值：是一个指向新建节点的引用指针。它是一个文本节点，所以它的 nodeType 属性等于 3.

新元素节点不会自动添加到文档里，新节点没有 nodeParent 属性

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset=*"UTF-8"*>  <title>Insert title here</title>  <script type=*"text/javascript"*>  **function** createTN() {  **var** para = document.createElement("p");  **var** txt=document.createTextNode("Hello World");  para.appendChild(txt);  }  </script>  </head>  <body>  <h1>你最喜欢哪些明星</h1>  <p title=*"my answer"* id=*"answer"*>我最喜欢的明星有：</p>  <ul id=*"stars"*>  <li>范冰冰</li>  <li>刘德华</li>  <li>王宝强</li>  </ul>  <input type=*"button"* value=*"创建文本节点"* onclick="createTN()">  </body>  </html> |

* **appendChild方法**

上面创建出了新的节点元素，把新创建的节点插入某个文档节点树的最简单方法：让它成为这个文档现有节点的一个子节点，使用appendChild方法。

格式：var reference = element.appendChild(newChild).

给定子节点 newChild 将成为给定元素节点 element 的最后一个子节点。

方法的返回值是一个指向新增子节点的引用指针。

该方法通常与 createElement() createTextNode() 配合使用

新节点可以被追加给文档中的任何一个元素

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset=*"UTF-8"*>  <title>Insert title here</title>  <script type=*"text/javascript"*>  **function** create() {  **var** para = document.createElement("p");  **var** txt=document.createTextNode("Hello World");  para.appendChild(txt);  document.body.appendChild(para);  }  </script>  </head>  <body>  <h1>你最喜欢哪些明星</h1>  <p title=*"my answer"* id=*"answer"*>我最喜欢的明星有：</p>  <ul id=*"stars"*>  <li>范冰冰</li>  <li>刘德华</li>  <li>王宝强</li>  </ul>  <input type=*"button"* value=*"追加节点"* onclick="create()">  </body>  </html> |

### 在已有元素前插入一个新元素

DOM中提供了一个名为：insertBefore()方法，该方法把一个新元素插入到现有元素的前面。调用此方法，需要知道三个元素：新元素、目标元素、目标元素的父元素。

把一个给定节点插入到一个给定元素节点的给定子节点的前面

格式：var reference = element.insertBefore(newNode,targetNode);

节点 newNode 将被插入到元素节点 element 中并出现在节点 targetNode 的前面.

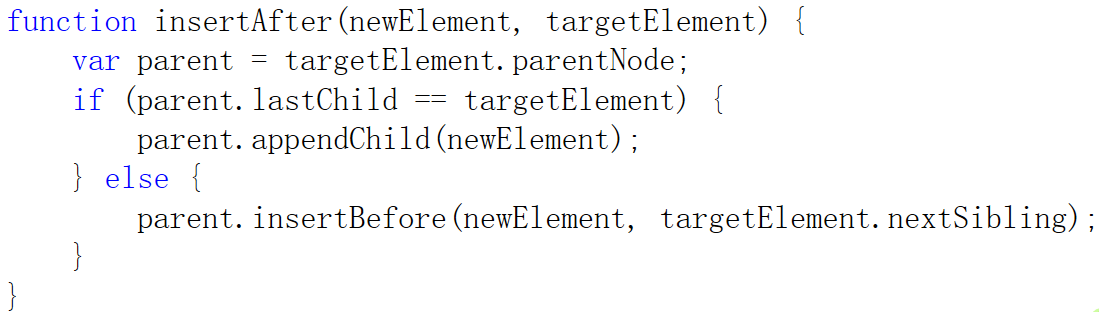
节点 targetNode 必须是 element 元素的一个子节点。

该方法通常与 createElement() 和 createTextNode() 配合使用

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset=*"UTF-8"*>  <title>Insert title here</title>  <script type=*"text/javascript"*>  **function** create() {  **var** para = document.createElement("h3");  **var** txt=document.createTextNode("insertBefore方法");  para.appendChild(txt);  **var** stars = document.getElementById("stars");  stars.parentNode.insertBefore(para, stars);  }  </script>  </head>  <body>  <h1>你最喜欢哪些明星</h1>  <p title=*"my answer"* id=*"answer"*>我最喜欢的明星有：</p>  <ul id=*"stars"*>  <li>范冰冰</li>  <li>刘德华</li>  <li>王宝强</li>  </ul>  <input type=*"button"* value=*"之前插入"* onclick="create()">  </body>  </html> |

### 在已有元素后插入一个新元素（了解）

有时我们希望在已有元素后插入一个新元素，很遗憾DOM没有提供insertAfter方法，需要我们自己编写该方法：



### 删除节点

从一个给定元素里删除一个子节点

**格式：var reference = element.removeChild(node);**

**返回值：返回值是一个指向已被删除的子节点的引用指针。**

**注意：某个节点被 removeChild() 方法删除时，这个节点所包含的所有子节点将同时被删除。**

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset=*"UTF-8"*>  <title>Insert title here</title>  <script type=*"text/javascript"*>  **function** rmChild()  {  **var** parent = document.getElementById("stars");  **var** child = document.getElementById("fanbingbing");  parent.removeChild(child);  }  </script>  </head>  <body>  <h1>你最喜欢哪些明星</h1>  <p title=*"my answer"*>我最喜欢的明星有：</p>  <ul id=*"stars"*>  <li id=*"fanbingbing"*>范冰冰</li>  <li>刘德华</li>  <li>王宝强</li>  </ul>  <input type=*"button"* value=*"删除子节点"* onclick="rmChild()">  </body>  </html> |

**如果想删除某个节点，但不知道它的父节点是哪一个，parentNode 属性可以帮忙。**

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset=*"UTF-8"*>  <title>Insert title here</title>  <script type=*"text/javascript"*>  **function** rmChild() {  **var** child = document.getElementById("fanbingbing");  **var** parent = child.parentNode;  parent.removeChild(child);  }  </script>  </head>  <body>  <h1>你最喜欢哪些明星</h1>  <p title=*"my answer"*>我最喜欢的明星有：</p>  <ul id=*"stars"*>  <li id=*"fanbingbing"*>范冰冰</li>  <li>刘德华</li>  <li>王宝强</li>  </ul>  <input type=*"button"* value=*"删除子节点"* onclick="rmChild()">  </body>  </html> |

### 替换节点

replaceChild()：把一个给定父元素里的一个子节点替换为另外一个子节点

格式：**var reference = element.replaceChild(newChild,oldChild);**

**返回值是一个指向已被替换的那个子节点的引用指针**。

如果被插入的子节点还有子节点，则那些子节点也被插入到目标节点中

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset=*"UTF-8"*>  <title>Insert title here</title>  <script type=*"text/javascript"*>  **function** create() {  **var** stars=document.getElementById("stars");  **var** fanbingbing=document.getElementById("fanbingbing");  **var** bj=document.getElementById("bj");  stars.replaceChild(bj, fanbingbing);  }  </script>  </head>  <body>  <h1>你最喜欢哪些明星</h1>  <p title=*"my answer"* id=*"answer"*>我最喜欢的明星有：</p>  <ul id=*"stars"*>  <li id=*"fanbingbing"*>范冰冰</li>  <li id=*"liudehua"*>刘德华</li>  <li>王宝强</li>  </ul>  <p>你喜欢哪个城市?</p>  <ul id=*"city"*>  <li id=*"bj"* name=*"BeiJing"*>北京</li>  <li>上海</li>  <li>东京</li>  <li>首尔</li>  </ul>  <input type=*"button"* value=*"替换节点"* onclick="create()">  </body>  </html> |

### innerText和innerHTML

对于表单元素，我们可以通过对象的value属性来获取表单元素的值，那么对于非表单元素呢？？？这时候就要用到innerText和innerHTML。

格式：var strValue = document.getElementById('元素ID值').innerText;

var strValue = document.getElementById('元素ID值').innerHTML;

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset=*"UTF-8"*>  <title>Insert title here</title>  <script type=*"text/javascript"*>  **function** showValue() {  **var** answer = document.getElementById("answer");  alert("innerText:"+answer.innerText);  alert("innerHTML:"+answer.innerHTML);  }  </script>  </head>  <body>  <h1>你最喜欢哪些明星</h1>  <p title=*"my answer"* id=*"answer"*>我最喜欢的明星有：</p>  <ul id=*"stars"*>  <li>范冰冰</li>  <li>刘德华</li>  <li>王宝强</li>  </ul>  <input type=*"button"* value=*"获取非表单元素的值"* onclick="showValue()">  </body>  </html> |

那么他们之间有什么区别呢？从名字上大概也可以猜出来：

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset=*"UTF-8"*>  <title>Insert title here</title>  <script type=*"text/javascript"*>  **function** showValue() {  **var** answer = document.getElementById("answer");  alert("innerText:"+answer.innerText);  alert("innerHTML:"+answer.innerHTML);  }  </script>  </head>  <body>  <h1>你最喜欢哪些明星</h1>  <p title=*"my answer"* id=*"answer"*><a>我最喜欢的明星有：</a></p>  <ul id=*"stars"*>  <li>范冰冰</li>  <li>刘德华</li>  <li>王宝强</li>  </ul>  <input type=*"button"* value=*"获取非表单元素的值"* onclick="showValue()">  </body>  </html>  结果可想而知：取值时 innerText会把只会获取节点里面的文本信息，而innerHTML 会获取节点下面的所有标签。innerHTML是符合W3C标准的属性，而innerText只适用于IE浏览器，因此，尽可能地去使用innerHTML，而少用innerText |

那如何动态的给非表单元素赋值呢？

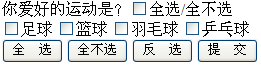
格式：document.getElementById('元素ID值').innerText = 动态值

document.getElementById('元素ID值').innerHTML = 动态值

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset=*"UTF-8"*>  <title>Insert title here</title>  <script type=*"text/javascript"*>  **function** showValue() {  **var** answer = document.getElementById("answer");  alert("innerText:"+answer.innerText);  alert("innerHTML:"+answer.innerHTML);  }  **function** setValue() {  **var** answer = document.getElementById("answer");  answer.innerHTML="<em>哈哈哈</em>";  }  </script>  </head>  <body>  <h1>你最喜欢哪些明星</h1>  <p title=*"my answer"* id=*"answer"*><a>我最喜欢的明星有：</a></p>  <ul id=*"stars"*>  <li>范冰冰</li>  <li>刘德华</li>  <li>王宝强</li>  </ul>  <input type=*"button"* value=*"获取非表单元素的值"* onclick="showValue()">  <input type=*"button"* value=*"动态赋值"* onclick="setValue()">  </body>  </html> |

## 课后作业

### 全选/全不选



当 checkbox 节点有 checked 属性时, 在 HTML 页面上即表现为 "被选中"

利用 setAttribute("checked", "checked"); 实现选中, removeAttribute("checked"); 实现不被选中

也可以利用 document.getElementById("checkedAll\_2").checked = "checked"; 实现选中, 利用 document.getElementById("checkedAll\_2").checked = null; 实现不被选中

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>Insert title here</title>  <script type=*"text/javascript"*>  //需求: 若 #checkedAll\_2 被选中, 则 name=items 的 checkbox 都被选中  //若 #checkedAll\_2 取消选中, 则 name=items 的 checkbox 都取消选中  //若 name=items 的 checkbox 都被选中, 则 #checkedAll\_2 的 checkbox 也被选中  //若 name=items 的 checkbox 有一个没有被选中, 则 #checkedAll\_2 取消选择.  //提示: 事件需要加给 #checkedAll\_2, 获取 name=items 的 checkbox 数组  //判断是否被选中, 若被选择, 则 name=items 的 checkbox 都要被选择  //若没有被选择, 则 name=items 的 checkbox 都要取消选择  //根据是否存在 checked 属性来判断是否被选择, 可以使其 checked = true 被选择  //checked = false 取消选择.  //还需要给每个 name=items 的 checkbox 加响应函数  //判断 name=items 的 checkbox 是否都被选中: 选择的个数和总个数是否相等.  //若都被选择: 则使 #checkedAll\_2 被选择  //若没有都被选择: 则使 #checkedAll\_2 取消选择    window.onload = **function**(){  document.getElementById("checkedAll\_2").onclick = **function**(){  **var** flag = **this**.checked;  **var** items = document.getElementsByName("items");  **for**(**var** i = 0; i < items.length; i++){  items[i].checked = flag;  }  }  **var** items = document.getElementsByName("items");  **for**(**var** i = 0; i < items.length; i++){  items[i].onclick = **function**(){  //记录有多少个 items 被选中了  **var** number = 0;  **for**(**var** j = 0; j < items.length; j++){  **if**(items[j].checked){  number++;  }  }  document.getElementById("checkedAll\_2").checked =  (items.length == number);  }  }  document.getElementById("CheckedAll").onclick = **function**(){  **for**(**var** i = 0; i < items.length; i++){  items[i].checked = **true**;  }  }  document.getElementById("CheckedNo").onclick = **function**(){  **for**(**var** i = 0; i < items.length; i++){  items[i].checked = **false**;  }  }  document.getElementById("CheckedRev").onclick = **function**(){  **for**(**var** i = 0; i < items.length; i++){  items[i].checked = !items[i].checked;  }  }  document.getElementById("send").onclick = **function**(){  **for**(**var** i = 0; i < items.length; i++){  **if**(items[i].checked){  alert(items[i].value);  }  }  }  }  </script>  </head>  <body>  <form method=*"post"* action=*""*>  你爱好的运动是？<input type=*"checkbox"* id=*"checkedAll\_2"* />全选/全不选  <br /> <input  type=*"checkbox"* name=*"items"* value=*"足球"* />足球 <input type=*"checkbox"*  name=*"items"* value=*"篮球"* />篮球 <input type=*"checkbox"* name=*"items"*  value=*"羽毛球"* />羽毛球 <input type=*"checkbox"* name=*"items"* value=*"乒乓球"* />乒乓球  <br /> <input type=*"button"* id=*"CheckedAll"* value=*"全　选"* /> <input  type=*"button"* id=*"CheckedNo"* value=*"全不选"* /> <input type=*"button"*  id=*"CheckedRev"* value=*"反　选"* /> <input type=*"button"* id=*"send"*  value=*"提　交"* />  </form>  </body>  </html> |

### 添加新员工



|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>Insert title here</title>  <script type=*"text/javascript"*>  /\*  \* 为 #employeetable 的所有的 a 节点添加 Onclick 响应函数:  \* 1. 弹出确认对话框: 确定要删除 xx 的信息吗? xx 为当前 a 节点所在的 td 所在的 tr 的  \* 第一个 td 子节点的文本值, 且要去除前后空格.  \* 2. 若点击 "确认" , 则删除 a 节点的所在的 行  \*  \* 注意:  \* 1. a 节点是一个超级链接, 可以在其 onclick 事件中通过返回  false, 来取消默认行为  \* 2. tr 的直接父节点为 tbody, 而不是 table  \* 3. 可以把去除前后空格的代码写成一个 trim(str) 函数.  \* 为 #addEmpButton 添加 onclick 响应函数:  \* 1. 获取 #name, #email, #salary 的文本框的值  \* 2. 利用 1 获取的文本值创建 3 个 td 节点, 在创建一个 tr 节点. 并把以上的 3 个 td  \* 节点价位 tr 节点的子节点  <tr>  <td>Tom</td>  <td>tom@tom.com</td>  <td>5000</td>  </tr>  \* 3. 在创建一个 td 节点: <td><a href="deleteEmp?id=xxx">Delete</a></td>  \* 4. 把 3 创建的 td 也加为 tr 的子节点.  \* 5. 再把 tr 加为 #employeetable 的 tbody 子节点的子节点.  \* 6. 为新创建的 tr 的 a 子节点添加 Onclick 响应函数, 使其也具备删除的功能.  \*  \*/  window.onload = **function**() {  **function** removeTr(aNoe) {  **var** trNode = aNoe.parentNode.parentNode;  **var** textContent = trNode.getElementsByTagName("td")[0].firstChild.nodeValue;  textContent = trim(textContent);  **var** flag = confirm("确定要删除" + textContent +  "的信息吗?");  **if** (flag) {  trNode.parentNode.removeChild(trNode);  }  **return** **false**;  }  **var** aNodes = document.getElementById("employeetable")  .getElementsByTagName("a");  **for** (**var** i = 0; i < aNodes.length; i++) {  aNodes[i].onclick = **function**() {  removeTr(**this**);  **return** **false**;  }  }  document.getElementById("addEmpButton").onclick =  **function**() {  **var** nameVal = document.getElementById("name").value;  **var** emailVal = document.getElementById("email").value;  **var** salaryVal = document.getElementById("salary").value;  **var** nameTd = document.createElement("td");  nameTd.appendChild(document.createTextNode(nameVal));  **var** emailTd = document.createElement("td");  emailTd.appendChild(document.createTextNode(emailVal));  **var** salaryTd = document.createElement("td");  salaryTd.appendChild(document.createTextNode(salaryVal));  **var** tr = document.createElement("tr");  tr.appendChild(nameTd);  tr.appendChild(emailTd);  tr.appendChild(salaryTd);  alert("aa");  //<td><a href="deleteEmp?id=xxx">Delete</a></td>  **var** aNode = document.createElement("a");  aNode.href = "deleteEmp?id=xxx";  aNode.appendChild(document.createTextNode("Delete"));  **var** aTd = document.createElement("td");  aTd.appendChild(aNode);  tr.appendChild(aTd);  aNode.onclick = **function**() {  removeTr(**this**);  **return** **false**;  }  document.getElementById("employeetable")  .getElementsByTagName(  "tbody")[0].appendChild(tr);  //value: 用于获取 html 表单元素的值  //alert(this.value); //  //nodeValue: 用于获取文本节点的文本值.  //alert(this.nodeValue); //null  }  **function** trim(str) {  **var** reg = /^\s\*|\s\*$/g;  **return** str.replace(reg, "");  }  }  </script>  </head>  <body>  <center>  <br> <br> 添加新员工 <br> <br>  name: <input type=*"text"* name=*"name"* id=*"name"* />&nbsp;&nbsp;  email: <input type=*"text"* name=*"email"* id=*"email"* />&nbsp;&nbsp;  salary: <input type=*"text"* name=*"salary"* id=*"salary"* /> <br> <br>  <button id=*"addEmpButton"* value=*"abc"*>Submit</button>  <br> <br>  <hr>  <br> <br>  <table id=*"employeetable"* border=*"1"* cellpadding=*"5"* cellspacing=*0*>  <tbody>  <tr>  <th>Name</th>  <th>Email</th>  <th>Salary</th>  <th>&nbsp;</th>  </tr>  <tr>  <td>Tom</td>  <td>tom@tom.com</td>  <td>5000</td>  <td><a href=*"deleteEmp?id=001"*>Delete</a></td>  </tr>  <tr>  <td>Jerry</td>  <td>jerry@sohu.com</td>  <td>8000</td>  <td><a href=*"deleteEmp?id=002"*>Delete</a></td>  </tr>  <tr>  <td>Bob</td>  <td>bob@tom.com</td>  <td>10000</td>  <td><a href=*"deleteEmp?id=003"*>Delete</a></td>  </tr>  </tbody>  </table>  </center>  </body>  </html> |