* JavaScript 简介

JavaScript 是脚本语言

JavaScript 是世界上最流行的编程语言

JavaScript 是一种轻量级的编程语言。

JavaScript 是可插入 HTML 页面的编程代码。

JavaScript 插入 HTML 页面后，可由所有的现代浏览器执行。

这门语言可用于 HTML 和 web，更可广泛用于服务器、PC、笔记本电脑、平板电脑和智能手机等设备。

**1.JavaScript：写入 HTML 输出**

实例

document.write("<h1>This is a heading</h1>");

document.write("<p>This is a paragraph</p>");

**2.JavaScript：对事件作出反应**

alert() 函数在 JavaScript 中并不常用，但它对于代码测试非常方便。

onclick 单击事件。

**3.JavaScript：改变 HTML 内容**

使用 JavaScript 来处理 HTML 内容是非常强大的功能。

x=document.getElementById("demo") //查找元素

x.innerHTML="Hello JavaScript"; //改变内容

您会经常看到 document.getElementByID("some id")。这个方法是 HTML DOM 中定义的。

DOM（文档对象模型）是用以访问 HTML 元素的正式 W3C 标准。

**4.JavaScript：改变 HTML 图像**

JavaScript 能够改变任意 HTML 元素的大多数属性，而不仅仅是图片。

JavaScript：改变 HTML 样式

改变 HTML 元素的样式，属于改变 HTML 属性的变种。

x=document.getElementById("demo") //找到元素

x.style.color="#ff0000"; //改变样式

**5.JavaScript：验证输入**

JavaScript 常用于验证用户的输入。

if isNaN(x) {alert("Not Numeric")};

**6.提示：**

JavaScript 与 Java 是两种完全不同的语言，无论在概念还是设计上。

Java（由 Sun 发明）是更复杂的编程语言。

ECMA-262 是 JavaScript 标准的官方名称。

JavaScript 由 Brendan Eich 发明。它于 1995 年出现在 Netscape 中（该浏览器已停止更新），并于 1997 年被 ECMA（一个标准协会）采纳。

* JavaScript 使用

HTML 中的脚本必须位于 <script> 与 </script> 标签之间。

脚本可被放置在 HTML 页面的 <body> 和 <head> 部分中。

**1.<script> 标签**

如需在 HTML 页面中插入 JavaScript，请使用 <script> 标签。

<script> 和 </script> 会告诉 JavaScript 在何处开始和结束。

<script> 和 </script> 之间的代码行包含了 JavaScript：

<script>

alert("My First JavaScript");

</script>

您无需理解上面的代码。只需明白，浏览器会解释并执行位于 <script> 和 </script> 之间的 JavaScript。

JavaScript 是所有现代浏览器以及 HTML5 中的默认脚本语言。

<body> 中的 JavaScript

在本例中，JavaScript 会在页面加载时向 HTML 的 <body> 写文本：

<!DOCTYPE html>

<html>

<body>

<script>

document.write("<h1>This is a heading</h1>");

document.write("<p>This is a paragraph</p>");

</script>

</body>

</html>

**2.JavaScript 函数和事件**

上面例子中的 JavaScript 语句，会在页面加载时执行。

通常，我们需要在某个事件发生时执行代码，比如当用户点击按钮时。

如果我们把 JavaScript 代码放入函数中，就可以在事件发生时调用该函数。

<head> 或 <body> 中的 JavaScript

提示：我们把 JavaScript 放到了页面代码的底部，这样就可以确保在 <p> 元素创建之后再执行脚本。

**3.外部的 JavaScript**

也可以把脚本保存到外部文件中。外部文件通常包含被多个网页使用的代码。

外部 JavaScript 文件的文件扩展名是 **.js**。

如需使用外部文件，请在 <script> 标签的 "src" 属性中设置该 .js 文件：

在 <head> 或 <body> 中引用脚本文件都是可以的。实际运行效果与您在 <script> 标签中编写脚本完全一致。

提示：外部脚本不能包含 <script> 标签。

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>Insert title here</title>

<!-- 2.嵌入式:在script标签内写JS.

该标签可以写在网页的任意位置. -->

<script>

//js注释可以这样

/\*也可以这样\*/

//1.js中的函数都是公有的

//2.js的函数返回值类型不用声明

function f2() {

//js中单引号和双引号作用相同

alert("范传奇");

}

</script>

<!-- 3.文件调用式:在独立的js文件中写js. -->

<!-- script标签不能同时引入js和书写js. -->

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

</head>

<body>

<!--

事件:是用户的操作(动作),是JS调用的时机.

如:单击事件,双击事件...

-->

<!-- 1.事件定义式:在定义事件时直接写JS. -->

<input type="button" value="按钮1"

onclick="alert('苍老师');"/>

<input type="button" value="按钮2"

onclick="f2();"/>

<input type="button" value="按钮3"

onclick="f3();"/>

</body>

</html>

* JavaScript 输出

JavaScript 通常用于操作 HTML 元素。

如需从 JavaScript 访问某个 HTML 元素，您可以使用 **document.getElementById(id)** 方法。

请使用 "id" 属性来标识 HTML 元素：

通过指定的 id 来访问 HTML 元素，并改变其内容：

<p id="demo">My First Paragraph</p>

document.getElementById("demo").innerHTML="My First JavaScript";

JavaScript 由 web 浏览器来执行。在这种情况下，浏览器将访问 id="demo" 的 HTML 元素，并把它的内容（innerHTML）替换为 "My First JavaScript"。

警告

请使用 document.write() 仅仅向文档输出写内容。

如果在文档已完成加载后执行 document.write，整个 HTML 页面将被覆盖：

Windows 8 中的 JavaScript

提示：微软支持通过 JavaScript 创建 Windows 8 app。

对于因特网和视窗操作系统，JavaScript 都意味着未来。

* JavaScript 语句

JavaScript 语句向浏览器发出的命令。语句的作用是告诉浏览器该做什么。

下面的 JavaScript 语句向 id="demo" 的 HTML 元素输出文本 "Hello World"：

document.getElementById("demo").innerHTML="Hello World";

分号 ;分号用于分隔 JavaScript 语句。

通常我们在每条可执行的语句结尾添加分号。

使用分号的另一用处是在一行中编写多条语句。

在 JavaScript 中，用分号来结束语句是可选的。

**1.JavaScript 代码**

JavaScript 代码（或者只有 JavaScript）是 JavaScript 语句的序列。

浏览器会按照编写顺序来执行每条语句。

本例将操作两个 HTML 元素：

document.getElementById("demo").innerHTML="Hello World";

document.getElementById("myDIV").innerHTML="How are you?";

**2.JavaScript 代码块**

JavaScript 语句通过代码块的形式进行组合。

块由左花括号开始，由右花括号结束。

块的作用是使语句序列一起执行。

JavaScript 函数是将语句组合在块中的典型例子。

下面的例子将运行可操作两个 HTML 元素的函数：

实例

function myFunction(){

document.getElementById("demo").innerHTML="Hello World";

document.getElementById("myDIV").innerHTML="How are you?";

}

**3.JavaScript 对大小写敏感。**

JavaScript 对大小写是敏感的。

当编写 JavaScript 语句时，请留意是否关闭大小写切换键。

**空格**

JavaScript 会忽略多余的空格。您可以向脚本添加空格，来提高其可读性

对代码行进行折行

您可以在文本字符串中使用反斜杠对代码行进行换行。下面的例子会正确地显示：

document.write("Hello \World!");

提示：JavaScript 是脚本语言。浏览器会在读取代码时，逐行地执行脚本代码。而对于传统编程来说，会在执行前对所有代码进行编译。

* JavaScript 注释

JavaScript 不会执行注释。

我们可以添加注释来对 JavaScript 进行解释，或者提高代码的可读性。

单行注释以 // 开头。

多行注释以 /\* 开始，以 \*/ 结尾。

* JavaScript 变量

变量是存储信息的容器。

我们使用 var 关键词来声明变量：

var carname;

变量声明之后，该变量是空的（它没有值）。

如需向变量赋值，请使用等号：

carname="Volvo";

提示：一个好的编程习惯是，在代码开始处，统一对需要的变量进行声明。

**1.一条语句，多个变量**

您可以在一条语句中声明很多变量。该语句以 var 开头，并使用逗号分隔变量即可：

var name="Gates", age=56, job="CEO";

声明也可横跨多行：

var name="Gates",

age=56,

job="CEO";

Value = undefined

在计算机程序中，经常会声明无值的变量。未使用值来声明的变量，其值实际上是 undefined。

在执行过以下语句后，变量 carname 的值将是 undefined：

var carname;

**2.重新声明 JavaScript 变量**

如果重新声明 JavaScript 变量，该变量的值不会丢失：

在以下两条语句执行后，变量 carname 的值依然是 "Volvo"：

var carname="Volvo";

var carname;

JS 对象

字符串、数字、布尔、数组、对象、Null、Undefined

**3.JavaScript 拥有动态类型**

JavaScript 拥有动态类型。这意味着相同的变量可用作不同的类型：

实例

var x // x 为 undefined

var x = 6; // x 为数字

var x = "Bill"; // x 为字符串

JavaScript 字符串

字符串是存储字符（比如 "Bill Gates"）的变量。

字符串可以是引号中的任意文本。您可以使用单引号或双引号：

**4.JavaScript 数字**

JavaScript 只有一种数字类型。数字可以带小数点，也可以不带：

实例

var x1=34.00; //使用小数点来写

var x2=34; //不使用小数点来写

极大或极小的数字可以通过科学（指数）计数法来书写：

实例

var y=123e5; // 12300000

var z=123e-5; // 0.00123

JavaScript 布尔

**5.布尔（逻辑）只能有两个值：true 或 false**。

var x=true

var y=false

**6.JavaScript 数组**

下面的代码创建名为 cars 的数组：

var cars=new Array();

cars[0]="Audi";

cars[1]="BMW";

cars[2]="Volvo";

或者 (condensed array):

var cars=new Array("Audi","BMW","Volvo");

或者 (literal array):

实例

var cars=["Audi","BMW","Volvo"];

数组下标是基于零的，所以第一个项目是 [0]，第二个是 [1]，以此类推。

**7.JavaScript 对象**

对象由花括号分隔。在括号内部，对象的属性以名称和值对的形式 (name : value) 来定义。属性由逗号分隔：

var person={firstname:"Bill", lastname:"Gates", id:5566};

上面例子中的对象 (person) 有三个属性：firstname、lastname 以及 id。

空格和折行无关紧要。声明可横跨多行：

var person={

firstname : "Bill",

lastname : "Gates",

id : 5566

};

**8对象属性有两种寻址方式：**

实例

name=person.lastname;

name=person["lastname"];

**9.Undefined 和 Null**

Undefined 这个值表示变量不含有值。

可以通过将变量的值设置为 null 来清空变量。

实例

cars=null;

person=null;

**10.声明变量类型**

当您声明新变量时，可以使用关键词 "new" 来声明其类型：

var carname=new String;

var x= new Number;

var y= new Boolean;

var cars= new Array;

var person= new Object;

JavaScript 变量均为对象。当您声明一个变量时，就创建了一个新的对象。

* JavaScript 函数

函数是由事件驱动的或者当它被调用时执行的可重复使用的代码块。

**1.JavaScript 函数语法**

函数就是包裹在花括号中的代码块，前面使用了关键词 function：

function functionname(){

这里是要执行的代码

}

当调用该函数时，会执行函数内的代码。

可以在某事件发生时直接调用函数（比如当用户点击按钮时），并且可由 JavaScript 在任何位置进行调用。

提示：JavaScript 对大小写敏感。关键词 function 必须是小写的，并且必须以与函数名称相同的大小写来调用函数。

调用带参数的函数

在调用函数时，您可以向其传递值，这些值被称为参数。

这些参数可以在函数中使用。

您可以发送任意多的参数，由逗号 (,) 分隔：

myFunction(argument1,argument2)

当您声明函数时，请把参数作为变量来声明：

function myFunction(var1,var2){

这里是要执行的代码

}

变量和参数必须以一致的顺序出现。第一个变量就是第一个被传递的参数的给定的值，以此类推。

函数很灵活，您可以使用不同的参数来调用该函数，这样就会给出不同的消息：

带有返回值的函数

有时，我们会希望函数将值返回调用它的地方。

通过使用 return 语句就可以实现。

在使用 return 语句时，函数会停止执行，并返回指定的值。

语法

function myFunction(){

var x=5;

return x;

}

上面的函数会返回值 5。

注释：整个 JavaScript 并不会停止执行，仅仅是函数。JavaScript 将继续执行代码，从调用函数的地方。

函数调用将被返回值取代：

var myVar=myFunction();

myVar 变量的值是 5，也就是函数 "myFunction()" 所返回的值。

即使不把它保存为变量，您也可以使用返回值：

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

"demo" 元素的 innerHTML 将成为 5，也就是函数 "myFunction()" 所返回的值。

您可以使返回值基于传递到函数中的参数：

实例

计算两个数字的乘积，并返回结果：

function myFunction(a,b){

return a\*b;

}

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

"demo" 元素的 innerHTML 将是：12

在您仅仅希望退出函数时 ，也可使用 return 语句。返回值是可选的：

function myFunction(a,b){

if (a>b){

return;

}

x=a+b

}

如果 a 大于 b，则上面的代码将退出函数，并不会计算 a 和 b 的总和。

**2.局部 JavaScript 变量**

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

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

只要函数运行完毕，本地变量就会被删除。

**3.全局 JavaScript 变量**

在函数外声明的变量是全局变量，网页上的所有脚本和函数都能访问它。

**4.JavaScript 变量的生存期**

JavaScript 变量的生命期从它们被声明的时间开始。

局部变量会在函数运行以后被删除。

全局变量会在页面关闭后被删除。

向未声明的 JavaScript 变量来分配值

如果您把值赋给尚未声明的变量，该变量将被自动作为全局变量声明。

这条语句：

carname="Volvo";

将声明一个全局变量 carname，即使它在函数内执行。

* **示例:猜数字**

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>Insert title here</title>

<script>

//在页面加载时生成随机数,在页面加载后,

//用户点击猜时使用这个随机数.

var ran = parseInt(Math.random()\*100);

function guess() {

//获取文本框内的值

var n = document.getElementById("num").value;

//获取span

var span = document.getElementById("result");

//判断该值是不是数字

if(isNaN(n)) {

//不是,给予提示

span.innerHTML = "请输入数字";

} else {

//是,开始猜

if(n>ran) {

span.innerHTML = "大了";

} else if(n<ran) {

span.innerHTML = "小了";

} else {

span.innerHTML = "对了";

}

}

}

</script>

</head>

<body>

<input type="text" id="num"/>

<input type="button" value="猜"

onclick="guess();"/>

<span id="result"></span>

</body>

</html>

* JavaScript 内置对象

JavaScript 对象

JavaScript 中的所有事物都是对象：字符串、数值、数组、函数...

此外，JavaScript 允许自定义对象。

JavaScript 提供多个内建对象，比如 String、Date、Array 等等。

**1.属性和方法**

属性是与对象相关的值。

方法是能够在对象上执行的动作。

举例：汽车就是现实生活中的对象。

汽车的属性：

car.name=Fiat

汽车的方法：

car.start()

**2.JavaScript 中的对象**

在 JavaScript 中，对象是数据（变量），拥有属性和方法。

当您像这样声明一个 JavaScript 变量时：

var txt = "Hello";

您实际上已经创建了一个 JavaScript 字符串对象。字符串对象拥有内建的属性 length。对于上面的字符串来说，length 的值是 5。字符串对象同时拥有若干个内建的方法。

属性：

txt.length=5

方法：

txt.indexOf()

txt.replace()

txt.search()

提示：在面向对象的语言中，属性和方法常被称为对象的成员。

**3.创建 JavaScript 对象**

JavaScript 中的几乎所有事务都是对象：字符串、数字、数组、日期、函数，等等。

你也可以创建自己的对象。

创建新 JavaScript 对象有很多不同的方法，并且您还可以向已存在的对象添加属性和方法。

**4.访问对象的属性**

访问对象属性的语法是：

objectName.propertyName

本例使用 String 对象的 length 属性来查找字符串的长度：

var message="Hello World!";

var x=message.length;

**5.访问对象的方法**

您可以通过下面的语法调用方法：

objectName.methodName()

这个例子使用 String 对象的 toUpperCase() 方法来把文本转换为大写：

var message="Hello world!";

var x=message.toUpperCase();

**6.创建直接的实例**

这个例子创建了对象的一个新实例，并向其添加了四个属性：

实例

person=new Object();

person.firstname="Bill";

person.lastname="Gates";

person.age=56;

person.eyecolor="blue";

**7.使用对象构造器**

本例使用函数来构造对象：

function person(firstname,lastname,age,eyecolor){

this.firstname=firstname;

this.lastname=lastname;

this.age=age;

this.eyecolor=eyecolor;

}

一旦您有了对象构造器，就可以创建新的对象实例，就像这样：

var myFather=new person("Bill","Gates",56,"blue");

var myMother=new person("Steve","Jobs",48,"green");

**8.把属性添加到 JavaScript 对象**

您可以通过为对象赋值，向已有对象添加新属性：

假设 personObj 已存在 - 您可以为其添加这些新属性：firstname、lastname、age 以及 eyecolor：

person.firstname="Bill";

person.lastname="Gates";

person.age=56;

person.eyecolor="blue";

x=person.firstname;

在以上代码执行后，x 的值将是：Bill

**9.JavaScript 类**

JavaScript 是面向对象的语言，但 JavaScript 不使用类。

在 JavaScript 中，不会创建类，也不会通过类来创建对象（就像在其他面向对象的语言中那样）。

JavaScript 基于 prototype，而不是基于类的。

**10.JavaScript for...in 循环**

JavaScript for...in 语句循环遍历对象的属性。

循环遍历对象的属性：

var person={fname:"Bill",lname:"Gates",age:56};

for (x in person){

txt=txt + person[x];

}

**示例:演示内置对象**

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>Insert title here</title>

<script>

//2.Number对象

var n = 3.1415926;

console.log(n.toFixed(3));

//3.Array对象

//3.1如何创建数组

//理解:js中的数组都是Object数组

//1)已知数据

var a1 = ["zhangsan",25,false];

console.log(a1[0]);

//2)未知数据

var a2 = new Array();

a2.push("lisi");

a2.push(28);

a2.push(true);

console.log(a2[0]);

//结论:任何方式创建的数组本质是一样的

//3.2数组倒转

var arr = [6,12,4,1,3,8];

arr.reverse();

console.log(arr);

//3.3数组排序

arr.sort();

console.log(arr);

//function x(a,b){return a-b}

arr.sort(function(a,b){

return a-b;

});

console.log(arr);

//4.Math

//5.Date

//5.1如何创建日期

var d1 = new Date();

var d2 = new Date("2017/05/01 11:11:11");

console.log(d1);

console.log(d2);

//5.2转为本地时间格式的字符串

console.log(d1.toLocaleDateString());

console.log(d1.toLocaleTimeString());

//5.3读取时间分量

var y = d1.getFullYear();

var m = d1.getMonth()+1;

var d = d1.getDate();

var today = y + "年" + m + "月" + d + "日";

console.log(today);

//6.RegExp对象

//6.1如何创建正则对象

var str = "you can you up,no can no bb.";

var reg = /no/;

//6.2 test()

//检测str中是否包含与reg相匹配的子串

console.log(reg.test(str));

//6.3 exec()

//1)普通模式

//从str中找出和reg相匹配的第一个子串

console.log(reg.exec(str));

//2)全局模式

//第N次调用则从str中找出

//和reg相匹配的第N个子串

reg = /no/g;

console.log(reg.exec(str));

console.log(reg.exec(str));

console.log(reg.exec(str));

console.log(reg.exec(str));

// && 和 &

console.log(true&&false);//false

console.log(true&&3);//3

console.log(true&false);//0

console.log(true&3);//1

//7.Function对象

function sum() {

var s = 0;

if(arguments.length) {

for(var i=0;i<arguments.length;i++) {

s += arguments[i];

}

}

return s;

}

//页面加载时直接调用sum函数

console.log(sum(1,2));

console.log(sum(1,2,3,4));

</script>

</head>

<body>

<p>用来演示js内置对象</p>

</body>

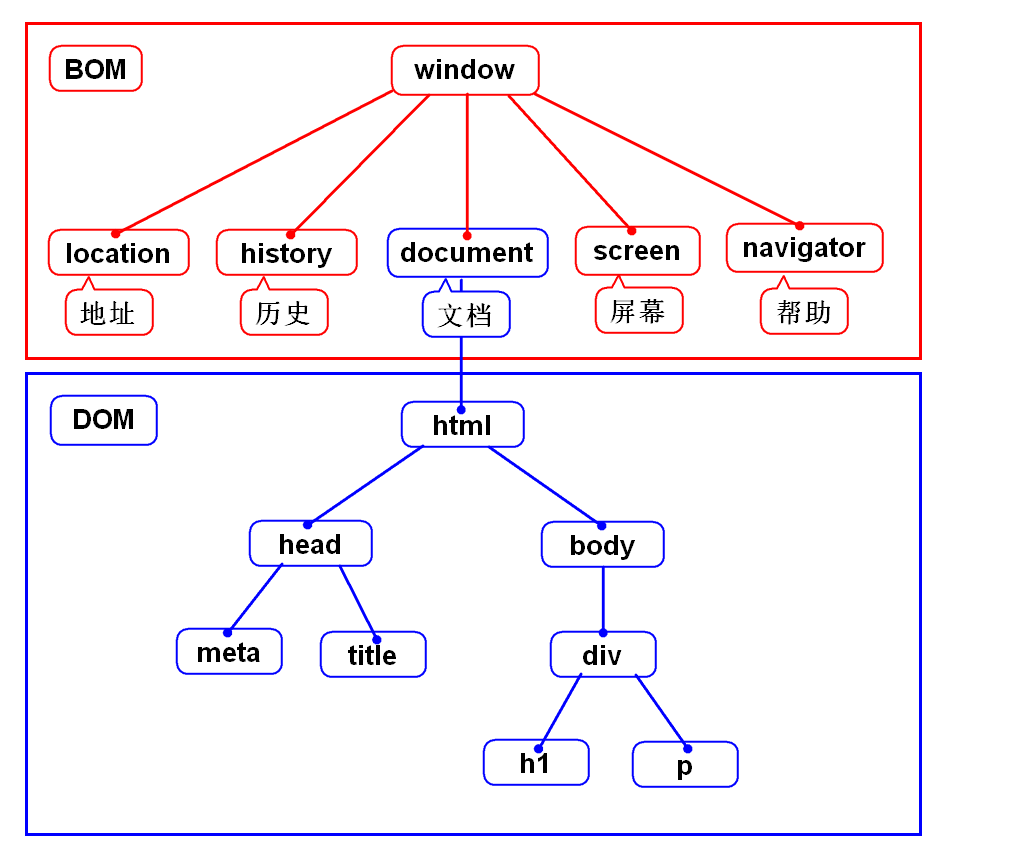
</html>

* JavaScript 外置对象**一.**

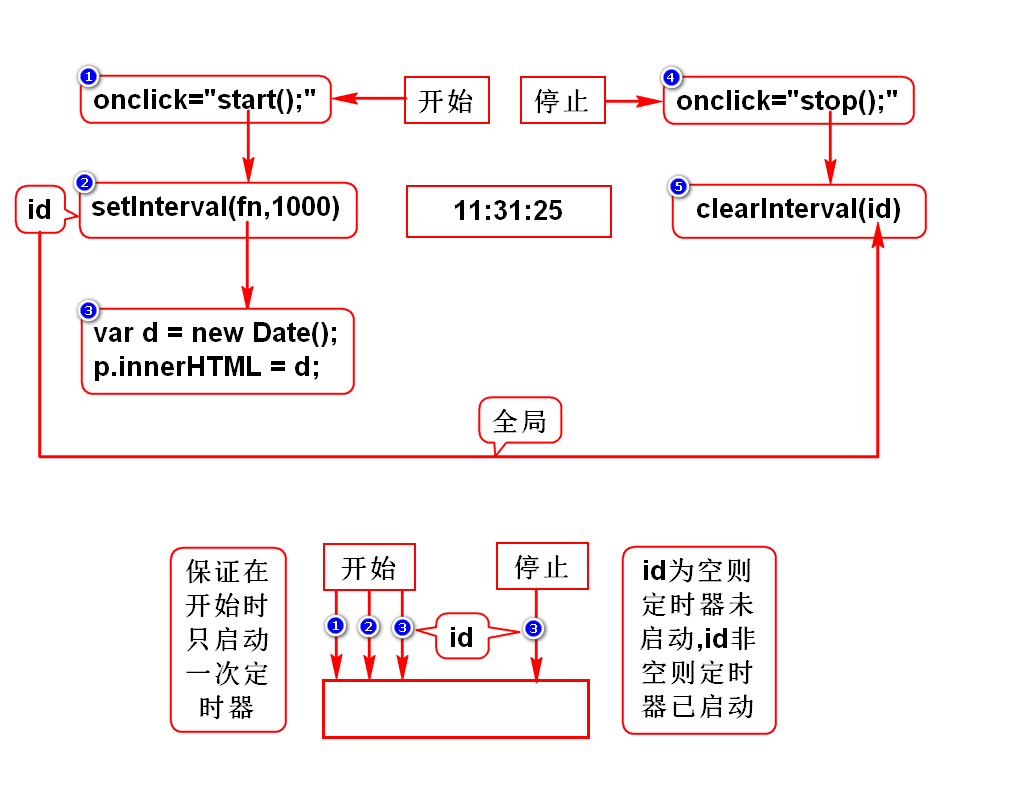
**外部对象概述**

外部对象本质上就是浏览器的API.

外部对象包括BOM和DOM,它们是包含关系.



**动态时钟**



**发送消息**

