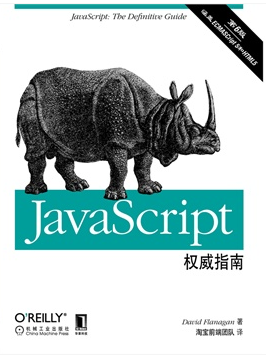
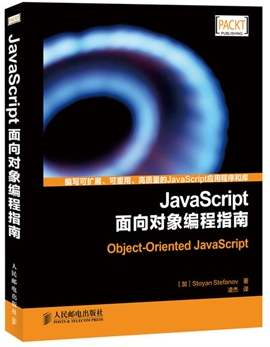
**核心Javascript培训教程**

**讲师: 张晓飞**

# 引言:

## 1.1. 网页三要素:

* HTML(内容)
* CSS(外观)
* Javascript(行为)

## 1.2. OOP的相关概念

### 1). 对象,方法和属性

* 对象就是指"事物"在程序设计语言中的表现形式
* 对象往往是用名词表示
* 方法往往是动词
* 属性往往是形容词

(The black cat sleeps on my head)

clas Cat{

private String color;

public void sleep(Object location) {}

}

### 2). 类

* 类是对象的模板,对象是类的一个实例
* javascript没有类, 它是通过构造器函数和原型(prototype)来实现类似类的概念

Class c = Class.forName(“com.atguigu.Person”);

method = c.getMethod(methodname, Class… params );

c.getField(name);

c.getConstructor(Class… params);

Class

Constructor[]

Method[]

Field[] fields

### 3). 封装

* 相关的数据(用属性存储)
* 基于这些数据所能做的事(用方法存储)

### 4). 聚合

* 指将几个现有的对象合并为一个新的对象的过程

### 5). 重用与继承

* javascript中没有类, 继承只能发生在对象之间

### 6). 多态

* 父类引用指向子类对象
* 通过父引用调用方法时调用的是子类重写的方法

# javascript介绍

## javascript的发展历史

LiveScript(网景Netscape)-->JavaScript(推广)

ECMAScript

(欧洲电脑厂商协会)

--->JScript(微软)

## javascript是什么?

* 总述: JavaScript一门解释型(不是编译型)脚本语言,它的主要特点有:面向对象,弱类型,动态,基于原型(对象)(不基于类)

### 1). 解释型(不是编译型)

* 代码编写好后, 不需要编译, 直接由js的解析引擎来解析执行(动态性更好,但开销较大)

### 2). 弱类型

* 在声明变量时,不用指定其数据类型, 而且它可以被随时赋值为任意类型的数据,解释器会根据上下文自动对其造型

|  |
| --- |
| **var** a = 1;  //alert(typeof a=="number"); //"number"  a = "tt";  //alert(typeof a); //string  a = [1, "tt", **null**];  //alert(typeof a);//object  a = **function**(){  alert("----");  };  //alert(typeof a);//function |

### 3). 面向对象

* 在JavaScript中“一切皆对象”，在这一方面，它比其他的OO语言来的更为彻底，即使作为代码本身载体的function，也是对象，数据与代码的界限在JavaScript中已经相当模糊

|  |
| --- |
| **var** b = {  name: "Tom",  age: 12,  getName: **function**(){**return** **this**.name}  };  //json Javascript Object Notation  //alert(b.name); //Tom  //alert(b.getName);//指向函数对象  //alert(b.getName()); //调用指向的函数对象 |

funtion

b=0x123

name=tom

b2=0x123

### 4). 动态

* 对象的属性可以在对象创建了之后, 动态的设置, 且属性值可以是任意类型的数据(自然可以是一个函数)

|  |
| --- |
| b.test = **function**() {  **return** **function**() {  alert("---");  };  };  b.test();  //b.test()(); |

### 5). 基于原型(对象)(不基于类)

* 在js中没有真正类的概念, 如何实现代码的复用呢?, 那就是prototype

|  |
| --- |
| **function** Person(name, age) {  **this**.name = name;  **this**.age = age;  }  **var** p = **new** Person("tom", 12);  Person.prototype.getAge = **function**() {  **return** **this**.age;  };  //alert(p.getAge()); |

## 1.2. javascript程序执行的宿主环境

* web浏览器(js解析引擎)
* 服务器端 Node.js
* 桌面
* 富媒体

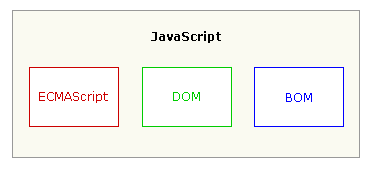
## javascript的分类

### 1). 按所处的位置分:

* 客户端js
* 服务器端js

### 2). 按模块分:

* 核心JavaScript(ECMAScript)
* BOM(Browser Object Model) window
* DOM(Document Object Model)
* 其它扩展: node.js



# 3. 基本概念

## 3.1. 变量

1). 理解: 变量是一块用来存储可变数据的内存的标识

2). 定义: 通过var来定义

|  |
| --- |
| **var** i = 1;  alert(i);  i = "abc";  alert(i); |

## 3.2. 数据类型

### 1). 数据的类型分类:

a. 基本类型: 数值(number),字符串(string),布尔值(boolean),undefined

b. 对象类型: 对象(object)，数组(array)，函数(function)

### 2). 变量的类型:

a. 基本类型: 保存基本类型数据的变量

b. 引用类型: 保存对象类型数据的地址值的变量

### 3). 详解:基本类型数据

a. 数值类型(number)

整数与小数

Infinity

NaN

b. 字符串类型(string)

定义: 一对单引号或双引号包的数据

c. 布尔值类型(boolean)

只有true和false两种值

d. undefined类型(不同于"undefined")

它的值’undefined’,如果变量未赋值, 即其值为undefined

|  |
| --- |
| **var** str = "Hello, world";//字符串  **var** i = 10;//整型数  **var** f = 2.3;//浮点数  **var** b = **true**;//布尔值  **var** u; //undefined(没有初始化或没有定义)  //alert(typeof str);  //alert(typeof i);  //alert(typeof f);  //alert(typeof b);  //alert(typeof u); |

### 4). 简介:对象类型数据

a. 对象(object): 属性的集合，即键值的散列表

b. 数组(array): 有序的列表

c. 函数(function): 包含可执行的代码的对象数据

|  |
| --- |
| **var** str = "Hello, world";  **var** obj = **new** Object(); //var obj = {};  obj.str = str;  obj.num = 2.3;  array = [ "foo", "bar", "zoo" ];  **function** func() {  alert("I am a function here");  }  //alert(typeof obj); //object  //alert(typeof array); //object  //alert(array instanceof Array);  //alert(typeof func); //function  //注意: null为一个对象类型数据  **var** n = **null**;  //alert(typeof n); |

### 5). 数据类型之间的转换:

a. 基本类型的字符串--->对象类型的字符串

b. 数值类型的对象字符串--->number

c. 任意类型--->boolean型

1. 能自动转为false的数据:空字符串, null, undefined, 0, NaN

2. 其它所有的都为true

|  |
| --- |
| **var** str = "Java";  //alert(typeof str); //string 基本类型  //alert(str.length);//4 自动转换为String对象类型  str = "123";  str = str / 1;  //alert(typeof str);//number 自动转换为基本类型数值  str = "123";  str = str + 1; //是一个连接符  //alert(str); //"1231"  str = str \* 1;//1231  str = str + 1;//1232 是一个算术运算符  //alert(str);  str = **null**;  **if** (!str) {  //alert("----1");  }  str = "a";  **if** (str) {  //alert("----2");  }  str = 0;  **if** (!str) {  //alert("----3");  } |

## 3.3. 操作符(特别的)

1). 小括号: ()用来调用一个函数

2). 中括号运算符: [ ]用来创建一个数组/获取数组元素

3). 大括号运算符: { }用来创建一个对象

4). 冒号运算符: 冒号用来分隔对象的属性名和属性值

5). typeof: 用来得到变量的数据类型所对应的字符串

6). instanceof: 判断一个对象是否是指定的类型

7). delete: 删除指定的数据

|  |
| --- |
| **function** fun3() {  //alert("----");  }  fun3();  **var** arr3 = [ 1, "abc", [ 1, 2 ] ];  //alert(arr3[2][1]);  **var** obj3 = {  name : "Tom",  "my age" : 12  };  //alert(obj3.name);  //alert(typeof obj=="object")  //alert(typeof arr3=="object")  //alert(arr3 instanceof Array);  **delete** arr3[0];  //alert(arr3[0]);  **delete** obj3.name;  //alert(obj3.name); |

## 3.4. 条件与循环

* if / switch / while,do-while,for,for-in

|  |
| --- |
| **var** obj3 = {  name : "tom",  age : 12,  getAge : **function**() {  **return** **this**.age;  }  };  **for** ( **var** v **in** obj3) {  **if** (obj3[v] == "function") {  //alert(obj3[v]());  } **else** {  //alert(obj3[v]);  }  } |

# 4. 函数

## 4.1. 定义函数(2+1种方式)

### 1). function

|  |
| --- |
| **function** sum(n, m) {  **return** n + m;  } |

### 2). var + function

|  |
| --- |
| **var** sum2 = **function**(n, m) {  **return** n + m;  }; |

### 3). new + Function

|  |
| --- |
| **var** sum3 = **new** Function("n", 'm', "return n+m;");  //自己不要这么写, 但它是定义函数的最终方式 |

## 4.2. 调用函数(传参问题)

* arguments: 在每个函数中都有一个内置的此引用变量,它指向代表调用时传的所有参数组成的数组

|  |
| --- |
| //alert(sum(2, 3));  //alert(sum2(2, 4));  //alert(sum3(2, 5));  //alert(sum(2)); //NaN  //利用函数的内建arguments数组对象来接收动态参数  **var** sum4 = **function**() {  **var** result = 0;  **for** (**var** i = 0; i < arguments.length; i++) {  result += arguments[i];  }  **return** result;  }  //alert(sum4(3, 2,6)); |

## 4.3. 系统预定义函数

1). parseInt() / parseFloat() : 将一个字符串转为数字

2). isNaN() : 判断是否不是一个数值

3). encodeURI() / decodeURI(): 对字符串进行编码与解码

4). eval() : 将Js代码格式的字符串当作js执行

|  |
| --- |
| //1). parseInt() parseFloat()  //alert(parseInt('12.3'));  //alert(parseInt('ab')); //NaN  //2). isNaN() 不是一个数值返回true  //alert(isNaN('ba')); //true  //alert(isNaN('12')); //false  //3). encodeURI() decodeURI()  //alert(encodeURI('好'));  //alert(decodeURI(encodeURI('好')));  //4). eval() 将传入的字符串当Js代码运行  //eval('alert("tttt");'); |

## 4.4. 变量的作用域

1). 变量的分类(按定义的位置分)

全局变量: 在函数外面定义的变量

局部变量: 在函数中用var定义的变量(包括形参)

静态变量

非静变量

2). 变量的作用域

全局变量: 在整个全局范围内都有效

局部变量: 在所在的函数范围内有效(不同于java)

|  |
| --- |
| //代码一:  **var** global = 1;  **function** f(arg){  **var** local = 2;  arg++;  global--;  **return** arg;  }  f(global);  //alert(window.local); //undifined  //alert(global); //0  //代码二  **var** i = 1;  **function** f2() {  //alert(i);  **var** i = 2;//局部变量的作用域是定义它的整个函数  }  f2(); //undifined |

## 4.5. 函数也是对象

### 1). 函数是一种对象数据

只是它的内容为一段可执行的代码, 如何证明一个数据是不是对象呢?

|  |
| --- |
| **var** a = **function**() {  **return** 2;  };  **var** b = a;  a.name = "tom";  //alert(b.name); |

### 2). 匿名函数

|  |
| --- |
| **function**(){  alert("-----");  } |

### 3). 自调用函数

|  |
| --- |
| (**function**(w){  w.$ = **function**(){  //alert("+++++");  };  })(window);  //$(); |

### 4). 回调用函数

|  |
| --- |
| **function** $$(f) {  **if** (**typeof** f == 'function') {  f();  } **else** {  alert('not a function');  }  }  //$$(function(){alert("----");});  //$$('tt'); |

### 5). 返回函数的函数

|  |
| --- |
| **function** aa() {  //alert('aa');  **return** **function**() {  //alert('bb');  };  }  //alert(typeof aa());  aa()(); |

### 6). 内部私有函数

|  |
| --- |
| **function** t() {  **function** tt(){  //alert("---");  }  tt();  }  t();  //tt();//不可见, 不可调用 |

## 4.6. 作用域链(扩展)

1. 回顾变量的作用域: 在所在的函数的整个范围内
2. 函数的作用域: 每个函数在定义(非执行)时, 解析引擎都会为其创建一个只属于它的环境(即作用域)

|  |
| --- |
| **function** f6() {  **var** i = 1;  f62();  }  **function** f62(){  alert(i);  }  f6(); |

# 数组

## 5.1. 创建数组(2种方式)

### 1). [ ]

|  |
| --- |
| **var** arra = [ 1, "t", {  t : "aa"  }, **function**() {  alert("----");  } ]; |

### 2). new Array()

|  |
| --- |
| arra = **new** Array(1, "t", {  t : "aa"  }, **function**() {  alert("----");  }); |

## 5.2. 访问数组中的元素

* 通过下标index去访问

|  |
| --- |
| //alert(arra[1]);  //arra[3](); |

## 5.3. 数组的类型

1. typeof
2. instanceof

|  |
| --- |
| //1). typeof  //alert(typeof arra); //object  //2). instanceof  //alert(arra instanceof Array); |

## 5.4. 添加和更新数组元素

|  |
| --- |
| arra[0] = 2;  //alert(arra[0]);  arra[5] = "kk";  //alert(arra[5]);  //alert(arra[4]); //undefined |

## 5.5. 删除数组元素

|  |
| --- |
| **delete** arra[0];  //alert(arra[0]); //undefined |

## 5.6. 扩展:

js中数组元素也可不以index存取, 而以字符串key存取

|  |
| --- |
| arra['tt'] = "abc";  alert(arra['tt']); |

# 对象

## 6.1. 创建对象

### 1). 使用{}(比较数组)

|  |
| --- |
| **var** obj = {  name : "abruzzi",  age : 26,  birthday : **new** Date(1984, 4, 5),  addr : {  street : "Huang Quan Road",  xno : "135", //属性  getStreet : **function**(){**return** **this**.street} //方法  }  };    //alert(obj.addr.street);  //alert(obj.addr.getStreet()); |

### 2). new构造函数创建

a. 系统的

|  |
| --- |
| obj = **new** Object();  obj.name = "xfzhang";  obj.age = 21;  obj.getAge = **function**(){**return** **this**.age};  //alert(obj.getAge()); |

b. 自定义的

|  |
| --- |
| **function** Person(name, age) {  **this**.name = name;  **this**.age = age;  **this**.getAge = **function**(){**return** **this**.age};  }  **var** p = **new** Person("Kity", 23);  p.setAge = **function**(age){**this**.age = age}; //给对象动态的添加了一个方法  p.setAge(13);  //alert(p.getAge()); |

## 6.2. 相关概念的理解

### 1). 对象:

JavaScript对象其实就是属性的集合,具有确定性，无序性和互异性

|  |
| --- |
| obj = {  tt : "abc",  tt : **function**() {  alert("---");  }//将前面的tt属性给覆盖了  };  //alert(obj.tt); |

### 2). 对象的属性:

可以动态的添加和删除, 且其值可以指向任意类型的数据

|  |
| --- |
| obj.test = "bb"  //alert(obj.test);  obj.test = 2;  //alert(obj.test);  **delete** obj.test;  //alert(obj.test); |

### 3). 对象的方法:

当对象的属性指向的是一个函数时, 一般会称之为方法

|  |
| --- |
| obj.test = **function**() {  //alert("-----");  };  obj.test();  obj.test = **function**() {  //alert("++++");  };  obj.test(); |

### 4). 全局对象:

js执行的宿主环境一般会提供一个全局对象(浏览器端window)

|  |
| --- |
| //alert(window);  //alert(window==this);  //alert(window instanceof Window); |

### 5). 全局变量/全局函数:

全局对象的属性(在函数外面定义的变量)

|  |
| --- |
| //Person("BB", 12);  //alert(window.name);  alert(name); |

## 6.3. 对象的constructor属性

1). constructor是一个指向用来创建当前对象的构造函数的引用

2). 它指向创建对象的构造方法

|  |
| --- |
| //alert(p.constructor);  //alert(window.age);  //alert(p.constructor("mm", 12));  //alert(window.age);  Person("BB", 12);  //alert(window.age); |

## 内建对象(构造器函数)

### 1). 数据封装类对象

#### 1.1). Object : 所有对象(构造器)的父级对象

* toString()

|  |
| --- |
| **var** o = **new** Object();  o = {};  //alert(o==o.toString());  o.toString = **function**() {  **return** "my object"  };  //alert(o); |

#### 1.2). Array:数组

* sort()
* reverse()
* push()
* pop()

|  |
| --- |
| **var** arr = **new** Array(620, '61b', '62a');  //alert(arr.reverse());  //alert(arr.sort());  //alert(arr);  //arr.push(9);  //alert(arr);  //arr.pop();  //alert(arr); |

#### 1.3). Function: 函数

##### 1.3.1). prototype属性

1. 它指向一个对象

2. 它只有在该函数被用作构造器时才会发挥作用

3. new出每个对象都自动拥有propotype的引用(\_\_proto\_\_), 并可以将其当做自身的属性来使用

|  |
| --- |
| **var** p2 = **new** Person("JAck", 13);  Person.prototype.height = 13;  alert(p2.height);  Person.prototype = "ttt"; |

object对象

Function对象

(包含代码)

height=13

prototype:”ttt”

Person: ox23

p2

name=Jack

age=13

\_\_proto\_\_:ox12

|  |
| --- |
| **var** some\_obj = {  name : 'TT'  };  **function** F() {  };  //alert(F.prototype);  //alert(new F().name);  F.prototype = some\_obj;  //alert(new F().name); |

|  |
| --- |
| **function** Person(name, age) {  **this**.name = name;  **this**.age = age;  }  **var** p1 = **new** Person("Tom",12);  Person.prototype.height = 12;  //alert(p1.height);  **var** p2 = **new** Person("Jack", 13);  //alert(p2.height);  //alert(p2.\_\_proto\_\_.height);  //alert(p2.\_\_proto\_\_==Person.prototype);  Person.prototype = {weight:120};  //alert(p2.height);  **var** p3 = **new** Person("dd", 23);  //alert(p3.weight); |

##### 1.3.2). length属性: 得到定义的参数个数

|  |
| --- |
| //alert(f.length); //区别于arguments |

##### 1.3.3). apply(obj, array)方法:

将当前函数应用到指定的对象上去执行, 如果没有指定对象则为window

|  |
| --- |
| **function** Product(name, price) {  **this**.name = name;  **this**.price = price;  }  **function** Toy(name, price) {  Product.apply(**this**, arguments);  **this**.category = 'toy';  }  Toy.prototype = **new** Product();  **var** f = **new** Toy("Tom", 12);  //alert(f); |

##### 1.3.4). call(obj, args)方法:

|  |
| --- |
| **function** Food(name, price) {  Product.call(**this**, name, price);  //this.Product(name, price)  //this.name = name;  ///this.price = price;  **this**.category = "food";  }  Food.prototype = **new** Product();  **var** f = **new** Food("cat", 12);  //alert(f.toString()); |

#### 1.4). String: 包装字符串的对象

* toUpperCase()
* toLowerCase()
* charAt()
* indexof()

#### 1.5). Number: 包装数值的对象

#### 1.6). Boolean: 包装boolea值的对象

### 2). 工具类对象

#### 2.1). Math: 包含一些数据计算的static方法

|  |
| --- |
| //alert(Math.random());  //alert(Math.min(1, 2)); |

#### 2.2). Date: 日期时间对象

|  |
| --- |
| //alert(new Date()); |

#### 2.3). RegExp: 正则表达式对象

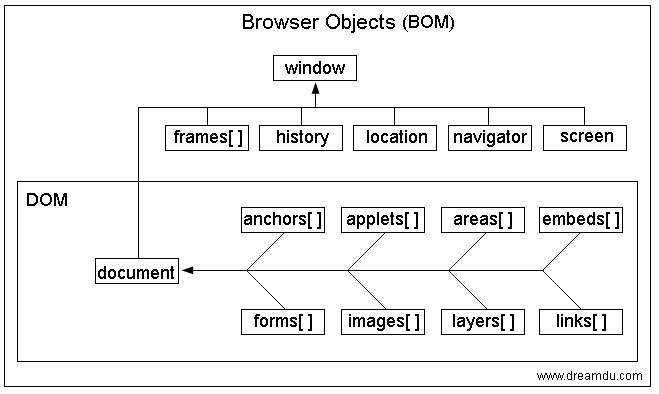
* 正则表达式使用单个字符串来描述、匹配一系列符合某个句法规则的[字符串](http://baike.baidu.com/view/56072.htm)

|  |
| --- |
| **var** reg=**new** RegExp("^\\d{6,8}$");  reg = /^\d{6,8}$/;  //alert(reg.test("123123222"));  //alert(reg.test("12312a")); |

### 3). 错误类对象

#### 3.1). Error

# 7. BOM与DOM



## 7.1. 什么是BOM?

* BOM是browser object model的缩写，简称浏览器对象模型
* BOM提供了独立于内容而与浏览器窗口进行交互的对象
* 由于BOM主要用于管理窗口与窗口之间的通讯，因此其核心对象是window
* BOM由一系列相关的对象构成，并且每个对象都提供了很多方法与属性
* BOM缺乏标准，JavaScript语法的标准化组织是ECMA，DOM的标准化组织是W3C
* BOM最初是Netscape浏览器标准的一部分

## 7.2. 什么是DOM?

DOM= Document Object Model，[文档对象模型](http://baike.baidu.com/view/758570.htm)， DOM 是 W3C（万维网联盟）的标准。DOM 定义了访问 HTML 和 XML 文档的标准：独立于平台和语言的接口. W3C DOM 标准被分为 3 个不同的部分：

* 核心 DOM - 针对任何结构化文档的标准模型

Node

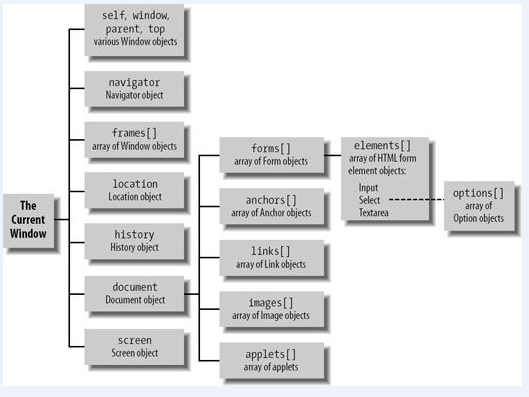
Document

Element

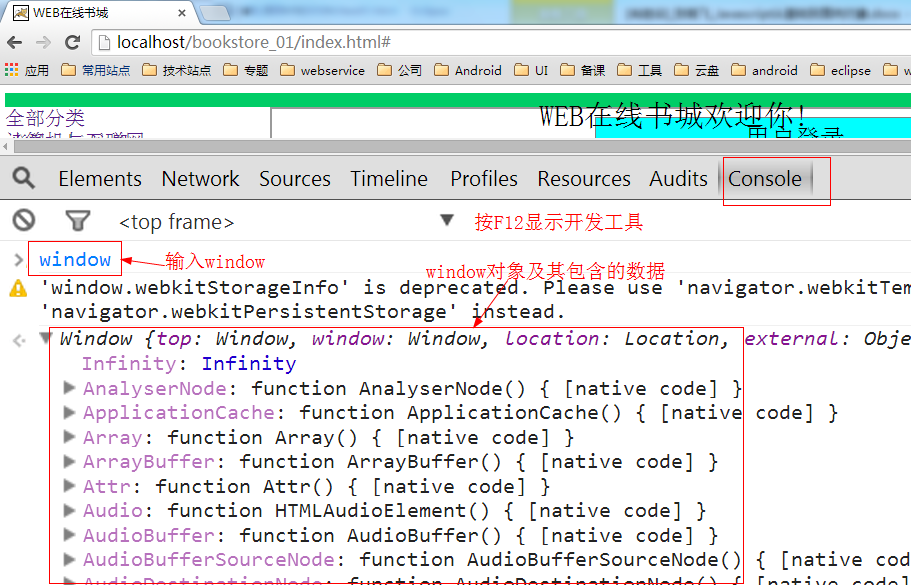
Attriubute

Text

* XML DOM - 针对 XML 文档的标准模型
* HTML DOM - 针对 HTML 文档的标准模型



## 7.3. BOM对象详解



### 1) Window对象

* Window 对象是浏览器端JavaScript的顶层对象(全局对象)
* Window 对象代表一个浏览器窗口或一个框架。
* Window浏览器自动创建。

|  |  |
| --- | --- |
| **属性** | **描述** |
| [document](mk:@MSITStore:E:\work\work02\140215\web\HTML_CSS_JS\w3school.CHM::/htmldom/dom_obj_document.asp.htm) | 对 Document 对象的只读引用 |
| [history](mk:@MSITStore:E:\work\work02\140215\web\HTML_CSS_JS\w3school.CHM::/htmldom/dom_obj_history.asp.htm) | 对 History 对象的只读引用 |
| [location](mk:@MSITStore:E:\work\work02\140215\web\HTML_CSS_JS\w3school.CHM::/htmldom/dom_obj_location.asp.htm) | 用于窗口或框架的 Location 对象 |
| parent | 返回父窗口。 |
| [self](mk:@MSITStore:E:\work\work02\140215\web\HTML_CSS_JS\w3school.CHM::/htmldom/prop_win_self.asp.htm) | 返回对当前窗口的引用。等价于 Window 属性 |
| window | window 属性等价于 self 属性 |
| [navigator](mk:@MSITStore:E:\work\work02\140215\web\HTML_CSS_JS\w3school.CHM::/htmldom/dom_obj_navigator.asp.htm) | 对 Navigator 对象的只读引用 |
| [screen](mk:@MSITStore:E:\work\work02\140215\web\HTML_CSS_JS\w3school.CHM::/htmldom/dom_obj_screen.asp.htm) | 对 Screen 对象的只读引用。 |

|  |  |
| --- | --- |
| **方法** | **描述** |
| [alert()](mk:@MSITStore:E:\work\work02\140215\web\HTML_CSS_JS\w3school.CHM::/htmldom/met_win_alert.asp.htm) | 显示带有一段消息和一个确认按钮的警告框。 |
| [confirm()](mk:@MSITStore:E:\work\work02\140215\web\HTML_CSS_JS\w3school.CHM::/htmldom/met_win_confirm.asp.htm) | 显示带有一段消息以及确认按钮和取消按钮的对话框。 |
| [prompt()](mk:@MSITStore:E:\work\work02\140215\web\HTML_CSS_JS\w3school.CHM::/htmldom/met_win_prompt.asp.htm) | 显示可提示用户输入的对话框。 |
| [open()](mk:@MSITStore:E:\work\work02\140215\web\HTML_CSS_JS\w3school.CHM::/htmldom/met_win_open.asp.htm) | 打开一个新的浏览器窗口或查找一个已命名的窗口。 |
| [close()](mk:@MSITStore:E:\work\work02\140215\web\HTML_CSS_JS\w3school.CHM::/htmldom/met_win_close.asp.htm) | 关闭浏览器窗口。 |

**注: window的属性和方法可以直接引用, 而不用通过window对象**

### 2). Location 对象

* 包含有关当前 URL 的信息, 并可以指定新的url
* 一般我们用它来请求一个新的资源:

|  |
| --- |
| window.location.href = “http://www.atguigu.com”;  window.location = “http://www.atguigu.com”; |

### 3). History 对象

* 包含浏览器前面浏览过的一系列URL 的信息
* 一般我们用它来显示上一个页面

|  |
| --- |
| window.history.back();  window.history.go(-1); |

### 4). Document对象

* Document对象代表整个HTML文档，可用来访问页面中的所有元素。
* Document对象window对象的一个部分,可通过window.document访问

|  |  |
| --- | --- |
| **方法** | **描述** |
| [getElementById()](mk:@MSITStore:E:\work\work02\140215\web\HTML_CSS_JS\w3school.CHM::/htmldom/met_doc_getelementbyid.asp.htm) | 返回对拥有指定id的第一个对象的引用 |
| [getElementsByTagName()](mk:@MSITStore:E:\work\work02\140215\web\HTML_CSS_JS\w3school.CHM::/htmldom/met_doc_getelementsbytagname.asp.htm) | 返回带有指定标签名的对象集合 |
| [write()](mk:@MSITStore:E:\work\work02\140215\web\HTML_CSS_JS\w3school.CHM::/htmldom/met_doc_write.asp.htm) | 向文档写 HTML 表达式 或 JavaScript 代码 |

# 8. js插件spket的安装

* **到**[**http://www.spket.com/download.html**](http://www.spket.com/download.html)**下载 Plugin 版本(spket-1.6.23.zip)，解压后直接放置于Eclipse的dropins目录下(..dropins/eclipse/..)，重启Eclipse.**
* **Window -> Preferences ->General-> Editors-> File Associations-> 选择\*.js,将Spket JavaScript Editor设为Default**

**配置jQuery**

* **Window -> Preferences -> Spket -> JavaScript Profiles -> New,输入“jQuery”点击OK； 选择“jQuery” 并点击“Add Library”然后在下拉条中选取“jQuery”； 选择 “jQuery”并点击“Add File”，然后选中你下载的jQuery.js 文件；设 成Default;**