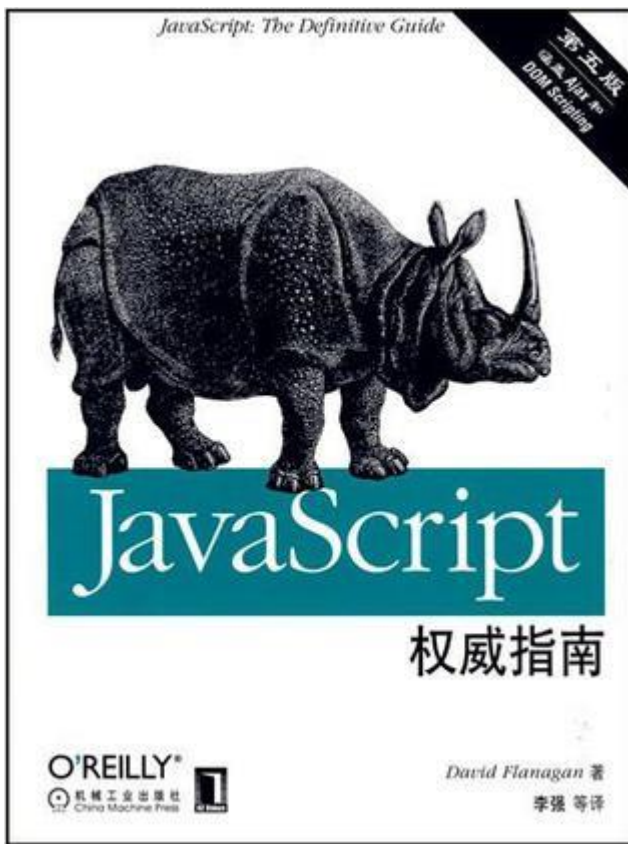


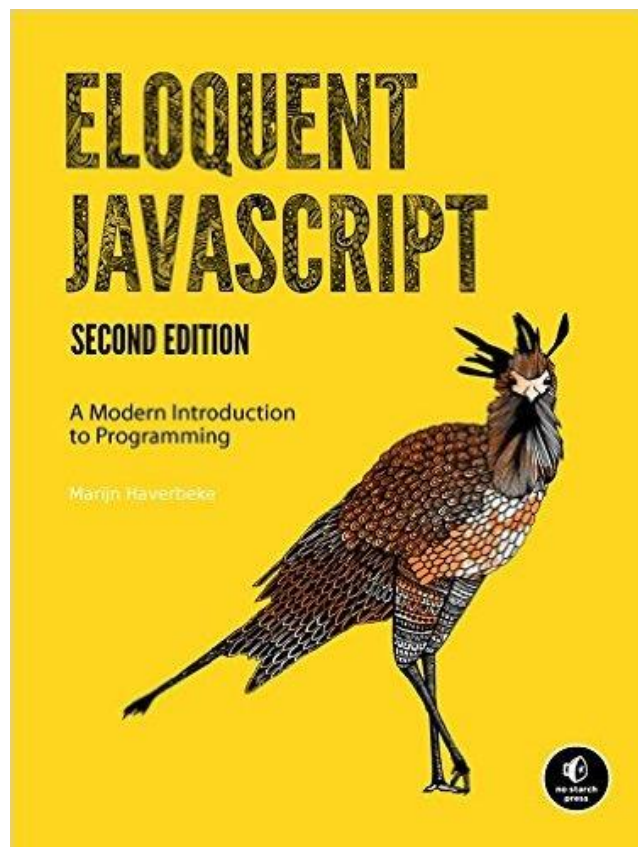
JavaScript

凯盛软件

推荐书籍

凯盛软件



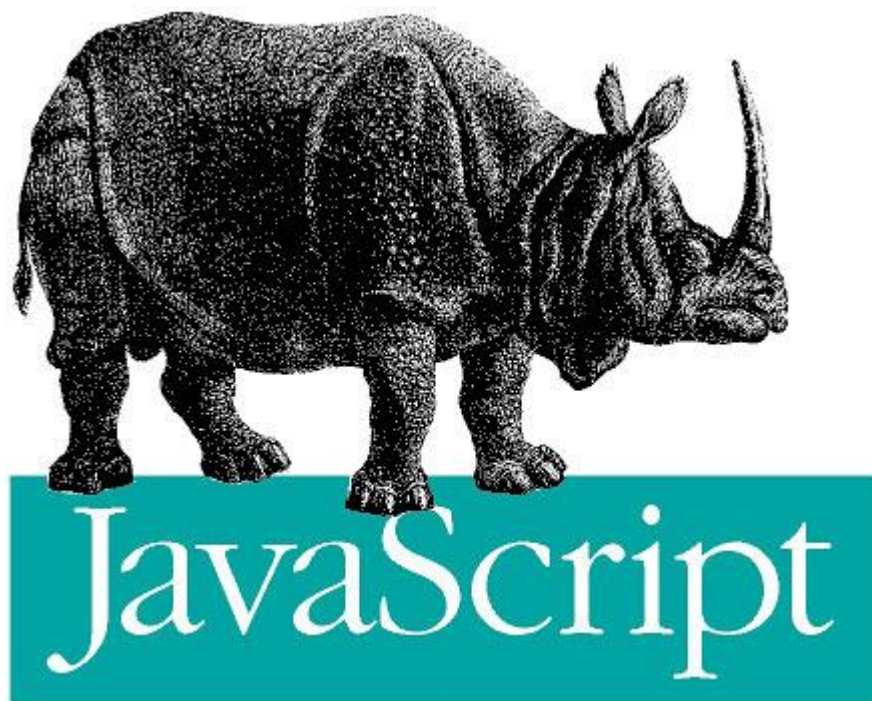


<http://eloquentjavascript.net/>

JavaScript vs Java

凯盛软件





1995


NetScape

livescript→javascript

Brendan Eich





凯盛软件






ruanyf @ruanyf · 8月15日


今日网友金句：“JavaScript语言的确传奇。如此大的一帮人用如此低劣的语言，在如此恶劣的兼容性环境下，搞出那么多炫彩夺目的网站，真是佩服码农的坚韧。”（via @rudaoshi）


 60
  25
 



陈永仁 @jnozsc · 7月9日





javascript 是最好的语言之：Why does ++[[]][+[]][+[]] return the string "10"? stackoverflow.com/questions/7202...

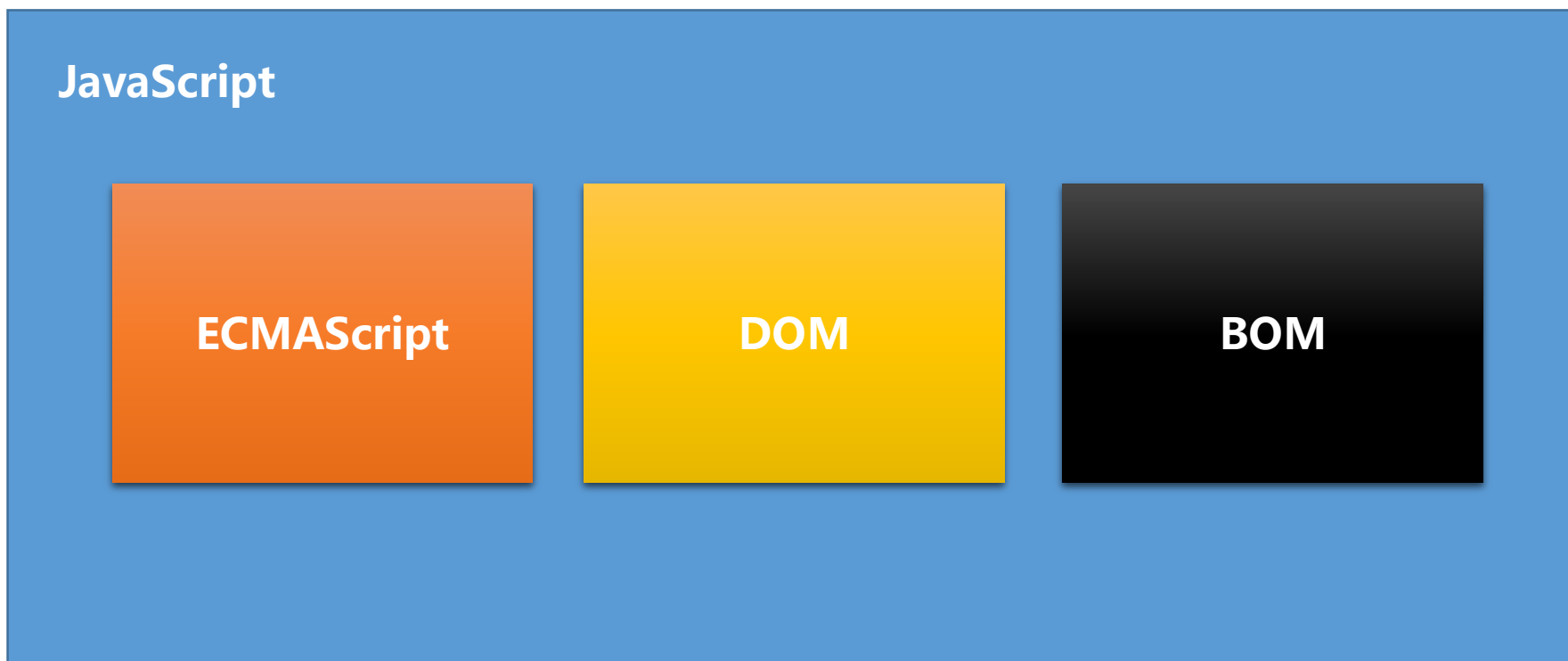


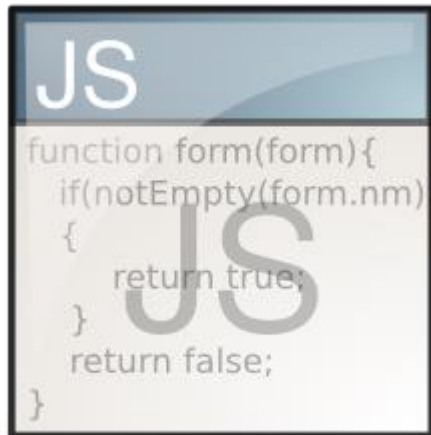
Why does ++[[]][+[]][+[]] return the string "10"?

This is valid and returns the string "10" in JavaScript (more examples here): ++[[]][+[]][+[]] Why? What is happening here?

stackoverflow.com


 2
  1
 





JavaScript文件以.js结尾

在HTML中引入外部的JavaScript文件

```
<html>
```

```
  <head>
```

```
    <script type="text/javascript" src="js/app.js"></script>
```

```
    <script type="text/javascript"
      src="http://lib.sinaapp.com/jquery.js"></script>
```

```
  </head>
```

```
</html>
```

当HTML中引入了多个外部JavaScript文件时，浏览器会按照从上到下的顺序去加载

在HTML中嵌入JavaScript

凯盛软件

```
<head>  
  <script type="text/javascript">  
    alert("Hello,JavaScript ! ");  
  </script>  
</head>
```

使用引入外部JavaScript文件的优点：

1. 可维护性：所有的JavaScript代码都在一个统一的文件夹中
2. 可缓存：浏览器可以缓存外部引用的JavaScript文件，使得页面显示速度加快

不是所有的浏览器都支持JavaScript，浏览器可以禁用JavaScript，当浏览器不支持或禁用JavaScript时要给客户提示：

```
<body>  
    <noscript>你的浏览器不支持或已禁用JavaScript</noscript>  
</body>
```

<script> 标签的位置

通常<script> 标签放在<head> 标签中

<head>

```
<script type="text/javascript" src="js/app.js"> </script>
```

</head>

缺点：当载入外部文件较大或JavaScript内容较多时，页面显示必须等到外部文件下载完成、解析和执行完成后才可以进行

最好的做法是将<script> 标签放在<body> 标签的最后

<body>

```
<!-- body中应该出现的内容 -->
```

```
<script type="text/javascript" src="js/app.js"> </script>
```

</body>

ECMAScript

- JavaScript中变量、函数名都是区分大小写的
- JavaScript中推荐使用驼峰命名法
- JavaScript中语句的结束使用;
结束
- 命名不能使用关键字和保留字

break case catch continue default delete
do else finally for function if in instanceof
new return switch this throw try typeof var
void while with

abstract boolean byte char class const debugger
double enum export extends final float goto
implements import int interface long native package
private protected public short static super
synchronized throws transient volatile

//单行注释

```
/*  
* 多行注释  
*/
```

定义变量使用关键字`var`：

```
var message = "Hi";
```

```
var message = "Hi",age = 22,money = 23.5;
```

JavaScript中变量的数据类型是根据值来决定的：

```
var message = "Hi"; //String
```

```
message = 23; //Number
```


在函数中使用var关键字声明的变量为局部变量，省略var关键字后声明的变量为全局变量

```
function test() {  
    var name = "tom";  
}  
test();  
alert(name); //error
```

```
function test() {  
    name = "tom";  
}  
test();  
alert(name);
```

在函数中声明全局变量的做法是不推荐的！

```
<script>  
    "use strict";  
    console.log("这是严格模式。");  
</script>
```

https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Strict_mode

http://www.ruanyifeng.com/blog/2013/01/javascript_strict_mode.html

基本数据类型

- undefined
- null
- boolean
- number
- string

复杂数据类型

- Object

typeof用于检测变量的数据类型，返回值如下：

“undefined” ——值未定义

“boolean” ——值为布尔型

“string” ——值为字符串

“number” ——值为数字型

“object” ——值为对象或null

“function” ——值为函数

```
var message = "Tom";
```

```
alert(typeof message);
```

- undefined类型只有一个值就是undefined
- 当var声明了一个变量，但是未给变量赋初始值时，变量的值就是undefined

```
var message;
```

```
alert(message); //undefined
```

```
alert(message == undefined); //true
```

- 未初始化的变量执行typeof会返回undefined。为声明的变量进行typeof操作也会返回undefined值

```
var message;
```

```
alert(typeof message); //undefined
```

```
alert(typeof age); //age未声明
```

- 声明变量时，推荐给变量赋初始值。

- null类型只有一个值，值为null
- null值代表一个对象的空指针引用，如果声明一个变量，未来该变量会指向一个对象，那么可以将该变量的初始值设置为null

```
var user = null;
```

```
alert(user);
```

- typeof操作null值时，返回值为object

```
var user = null;
```

```
alert(typeof user);
```

- undefined值派生自null值，所以进行boolean判断时，结果总为true

```
alert(null == undefined);
```

- boolean类型只有两个值，true和false
- 调用Boolean()方法可以将所有类型转换为boolean类型

```
var name = "tom";
```

```
var result = Boolean(name);
```

```
alert(result);
```

```
var name = "tom";
```

```
if(name) {  
    alert("value is true");  
}
```

数据类型	转换为true的值	转换为false的值
Boolean	true	false
string	任何非空字符串	"" 空字符串
number	任何非零数字值	0和NaN
object	任何对象	null
undefined		undefined

- number类型用来表示整数和浮点数
- 整数可以表示十进制、八进制和十六进制

```
var num = 100;
```

```
var num1 = 070; //八进制
```

```
var num2 = 0xA; //十六进制
```

- 浮点数

```
var num = 1.1;
```

```
var num1 = 3.125e10; //科学计数法
```

```
var num2 = 2.0; //整数，被解析为2
```

- NaN，即非数值(Not a Number)是Number类型中一个特殊的值，该数值表示一个本来要返回数值的操作数未返回数值的情况
- 任何涉及NaN操作的结果总是NaN，例如NaN/10
- NaN不和任何值相等，包括NaN自己
- isNaN()函数用于判断参数是否可以转换为数值型，如果不能转换则返回true

```
alert(isNaN(NaN)); //true
```

```
alert(isNaN(10)); //false
```

```
alert(isNaN("100")); //false
```

```
alert(isNaN("tom")); //true
```

```
alert(isNaN(true)); //false 结果为1
```


3个函数可以将非数值类型转换为数值类型：

- `number()`
- `parseInt()`
- `parseFloat()`

转换规则如下：

- 如果是boolean值，true和false分别转换为1和0
- 如果是数值，只是简单的传入和返回
- 如果是null值，返回0
- 如果是undefined，返回NaN
- 如果是字符串
 - 如果字符串中只包含数字文本，则转换为对应的十进制数值
 - 如果字符串是空的，则转换为0
 - 如果字符串含有非数字文本，则转换为NaN

Number()

```
var num = Number("hello"); // NaN
```

```
var num1 = Number(100); // 100
```

```
var num2 = Number("0011"); // 11
```

```
var num3 = Number(true); // 1
```

```
var num4 = Number(""); // 0
```

- 在转换字符串时，如果第一个字符为非数字或非负号，则返回NaN
- 如果字符串第一个字符为数字或负号，则会解析第二个字符，直到解析完所有字符或碰到非数字字符为止
- 如果字符串以" 0x" 开头，则认为是16进制
- 如果字符串以" 0" 开头，则认为是8进制

```
var num = parseInt("hello"); // NaN
```

```
var num1 = parseInt("123Tom"); // 123
```

```
var num2 = parseInt("0xa",16); // 10 以16进制解析
```

```
var num3 = parseInt(true); // NaN
```

```
var num4 = parseInt(""); // NaN
```

```
var num5 = parseInt("22.9"); //22
```

```
var num6 = parseInt("070",8); //56 以8进制解析
```

parseFloat()

parseFloat () 只能解析将字符串解析为十进制

```
var num = parseFloat("1234Tom"); //1234
```

```
var num1 = parseFloat("0xa"); //0
```

```
var num2 = parseFloat("22.5"); //22.5
```

```
var num3 = parseFloat("22.3.4"); //22.3
```

```
var num4 = parseFloat("0987.23"); //987.23
```

```
var num5 = parseFloat("3.14e10") //31400000000
```

- string类型可以使用双引号或单引号表示
- 支持java中的转义字符，例如\n，\t等
- 使用length属性获取字符串的长度
- 和java一样，string类型的值不可变的，改变某个变量的字符串值，会将原有值销毁，再创建新的值
- 通过toString()方法或String()方法将其他类型转换为字符串

```
var str = "Hi";
```

```
var message = 'Hello \n tom';
```

```
alert(str.length); //2
```

toString()

```
var age = 11;  
var ageAsString = age.toString();  
var result = true;  
var resultAsString = result.toString();  
var num = 10;  
alert(num.toString());  
alert(num.toString(2));  
alert(num.toString(10));  
alert(num.toString(8));  
alert(num.toString(16));
```

```
var str;  
alert(str.toString());
```

```
var str = null;  
alert(str.toString());
```

Uncaught TypeError: Cannot call method 'toString' of undefined

- String()函数可以将任何类型转换为字符串
- 如果值有toString()方法，则调用该方法(没有参数)并返回相应的结果
- 如果值为null，则返回“ null”
- 如果值为undefined，则返回“ undefined”

String()

```
var num = 10;
```

```
var result = true;
```

```
var message;
```

```
var name = null;
```

```
alert(String(num));
```

```
alert(String(result));
```

```
alert(String(message));
```

```
alert(String(name));
```

字符串的常用方法

凯盛软件

```
var color = "red";

alert(color.length); //获取字符串长度
alert(color.charAt(1)); //e 根据索引获取字符
alert(color.charCodeAt(1)); //101 获取当前索引字符的字符编码
alert(color.indexOf("e")); //1 根据元素查找索引，找不到返回-1
alert(color.indexOf("e",2)); //-1 从索引2的位置开始向后查找字符e
alert(color.lastIndexOf("e")); //1
alert(color.lastIndexOf("e",0)); //-1 从索引0的位置开始向前查找字符e
alert(color.trim()); //去除字符串前后空格
alert(color.toLowerCase()); //red 转换为小写
alert(color.toUpperCase()); //RED 转换为大写

var result = color.concat("-test"); //连接字符串
alert(result); //red-test

var txt = "tom,jerry,alex";
var txtArray = txt.split(",");
alert(txtArray);//["tom","jerry","alex"]
```

- 一元操作符(++ , --)
- 布尔操作符(! , && , ||)
- 算术运算符(+ - * /)
- 关系运算符(< > >= <= == !=)
- 全等和不全等(=== !==), 不仅判断值是否相等, 还判断是否是同一个数据类型

```
alert("50" == 50); //true
```

```
alert("50" === 50); //false
```

- 三元表达式
- 赋值运算符(= += -= *= /= %=)

```
var num = 10;

if(num == 10) {

    alert("A");
} else if(num == 20) {

    alert("B");
} else {

    alert("C");
}
```

```
var num = 10;

switch(num) {

    case 10:

        alert("A");

        break;

    case 20:

        alert("B");

        break;

    default:

        alert("C");

}
```

switch不仅可以使⽤数值型，还可以使⽤string类型

```
var temp = 0, total = 0;
do{
    temp++;
    total += temp;
}while(temp < 100);
alert(total);
```

```
var temp = 0, total = 0;
while(temp < 100){
    temp++;
    total += temp;
}
alert(total);
```

```
var total = 0;
for(var index = 0; index <= 100; index++) {
    total += index;
}
alert(total);
```

break和continue

```
for (var i = 0; i <= 10; i++) {  
    if(i == 3) {  
        break;  
    }  
};
```

```
for (var i = 0; i <= 10; i++) {  
    if(i == 3) {  
        continue;  
    }  
};
```

函数function相当于Java中的方法，定义一个函数的方法如下：

```
function sayHello(name) {  
    alert("Hello," + name);  
}
```

```
sayHello("Tom");
```

定义一个带返回值的函数：

```
function sayHello(name) {  
    return "Hello," + name;  
}
```

```
var message = sayHello("Tom");  
alert(message);
```

- JavaScript中函数的不介意传递参数的数量和数据类型，也就是说，定义一个函数需要传递两个参数，你可以选择传递一个或两个、三个，甚至可以不传递
- 参数在JavaScript内部是用一个名称为**arguments**数组来管理的

```
function sayHello(name) {  
    return "Hello," + arguments[0];  
}
```

```
var message = sayHello("Rose");  
alert(message);
```

- 可以通过调用arguments数组的length属性来获取参数的数量

```
function sayHello() {  
    alert(arguments.length);  
}  
sayHello("tom","jerry");  
sayHello();  
sayHello("Jack");
```



```
function add() {  
    var total = 0;  
    for(var temp in arguments) {  
        total += arguments[temp];  
    }  
    return total;  
}  
alert(add(1,2,3));  
alert(add(1,1,3,4,5));|
```

JavaScript中没有方法的重载

在JavaScript中函数名称不与某个特定的函数进行捆绑，函数名只是指向一个函数对象的指针而已。

```
//函数声明
function add(num1,num2) {
    return num1 + num2;
}

//函数表达式声明
var add = function(num1,num2){
    return num1 + num2;
};

var sum = add;

alert(sum(1,2)); //3
alert(add(1,2)); //3
```

解析器(浏览器)会率先读取函数声明，并使其在任何代码执行之前可用。

```
alert(add(2,3)); //5
```

```
//函数声明
```

```
function add(num1,num2) {  
    return num1 + num2;  
}
```

但是函数表达式声明的函数，必须等到解析器执行到它所在代码行，才会真正的被执行。

```
alert(add(2,3)); //Error
```

```
//函数表达式声明
```

```
var add = function(num1,num2) {  
    return num1 + num2;  
};
```

所有函数类型都拥有length属性，来获取参数列表中的数量

```
function test1(num1) {}  
function test2(num1,num2) {}  
function test3() {}
```

```
alert(test1.length); //1  
alert(test2.length); //2  
alert(test3.length); //0
```

计算阶乘

```
function factorial(num) {  
    if(num <= 1) {  
        return 1;  
    } else {  
        return num * factorial(num-1);  
    }  
}
```

如此调用会出现错误

```
var z = factorial;  
factorial = null;  
var result = z(5); //Error  
alert(result);
```

最佳实践：

```
var factorial = (function f(num) {  
    if(num <= 1) {  
        return 1;  
    } else {  
        return num * f(num-1);  
    }  
});
```

```
var z = factorial;  
factorial = null;  
var result = z(5); //ok  
alert(result);
```

闭包(closure)

闭包：有权访问另一函数作用域中的变量的函数(定义在一个函数内部的函数)

```
function outer() {  
    function inner() {  
    }  
}
```

```
function add(num1,num2) {  
    return function doAdd(){  
        return num1 + num2;  
    };  
}
```

```
var r = add(1,2);  
var result = r();  
alert(result);  
//或者调用方式如下  
var result = add(1,2)();  
alert(result);
```

闭包中对局部变量是按照引用传递的

```
function test(){  
    var num = 100;  
    var fn = function(){  
        alert(num);  
    };  
    num++;  
    return fn;  
}
```

test();

外部函数中所有的局部变量都在闭包内，即使这个局部变量在闭包后声明

```
function test(){  
    var fn = function(){  
        alert(name);  
    };  
    var name = "Jack";  
    return fn;  
}
```

test();

```
var name = "Jerry";  
function test() {  
    alert(name);  
    var age = 12;  
}  
test();  
alert(age); //Error
```

```
var name = "Jerry";  
function test() {  
    alert(name);  
    age = 12; ←  
}  
test();  
alert(age); //Ok
```

```
function test() {  
    if(true){  
        var address = "China";  
    }  
    alert(address); ←  
}  
test();  
alert(address); //Error ←
```

没有块级作用域

模仿块级作用域

```
function test() {  
    if(true) {  
        (function(){  
            var address = "jerry";  
            alert(address);  
        })();  
    }  
    alert(address);  
}  
test();
```

1 `var obj = new Object();`
`obj.name = "Tom";`
`obj.age = 23;`

`alert(obj.name);`
`alert(obj.age);`

2 `var obj = {};`
`obj.name = "Rose";`
`obj.age = 23;`

`alert(obj.name);`
`alert(obj.age);`

3 `var obj = {`
 `name : "Rose",`
 `age : 23`
`};`

`alert(obj.name);`
`alert(obj.age);`

4 `var obj = {`
 `name : "Rose",`
 `age : 23`
`};`
`obj.add = function(num1,num2){`
 `return num1 + num2;`
`}`

`alert(obj.add(12,23));`

5 `var obj = {`
 `name : "Rose",`
 `age : 23,`
 `add : function(num1,num2){`
 `return num1 + num2;`
 `}`
`};`

`alert(obj.add(12,23));`

6 `function sayHello(user) {`
 `alert("Hello," + user.name);`
`}`

`sayHello({name:"Hanks",age:23});`

函数中的this关键字获取值时根据执行对象不同，返回的值不同，在全局作用域中调用时，获取的是全局变量的值。在对象中调用时，获取的是对象的值

```
var color = "red";
var box = {color:"blue"};

function showColor() {
    alert(this.color); //获取全局变量
}

showColor(); //red
box.show = showColor;
box.show(); //blue
```

```
var name = "outer";

var obj = {
  name : "obj-name",
  sayName : function(){
    return function(){
      return this.name;
    };
  }
};

alert(obj.sayName()); //outer
```

```
var name = "outer";

var obj = {
  name : "obj-name",
  sayName : function(){
    var t = this; ←
    return function(){
      return t.name;
    };
  }
};

alert(obj.sayName()); //obj-name
```

- JavaScript的数组中可以保持任何类型的数据
- JavaScript的数组的大小可以动态调整
- 数组的声明：

```
var names = new Array();  
var names = new Array(20); //创建一个length为20的数组  
var names = new Array("Rose", "Jerry", "Alex");
```

- 同样数组的声明可以省略new关键字：

```
var names = Array();  
var names = Array(20); //创建一个length为20的数组  
var names = Array("Rose", "Jerry", "Alex");
```

- 通过数组字面量方式来创建：

```
var names = ["Tom", "Jerry", "Rose"];  
var names = []; //创建空数组
```

```
var names = ["Tom", "Jerry", "Rose"];  
alert(names.length); //获取数组的长度  
  
alert(names[0]);  
names[1] = "zhangsan";  
  
names.length = 4; //设置数组的长度  
names[3] = "Peter";  
  
names[names.length] = "wangwu"; //往数组的末尾添加新项  
names[names.length] = "Jack";  
  
names[99] = "zhaosi";  
alert(names.length); //100
```

数组元素的默认值为undefined

数组的使用

凯盛软件

```
var names = ["Tom","Jerry","Rose"];  
alert(names);  
alert(names.toString());  
  
alert(names.join("-")); //更改数组分隔符
```

数组中的栈方法

凯盛软件

```
var names = [];  
//将数组元素推入栈中，返回当前数组的长度  
var count = names.push("Tom","Jerry");  
count = names.push("Alex");
```

```
alert(names.length); //3
```

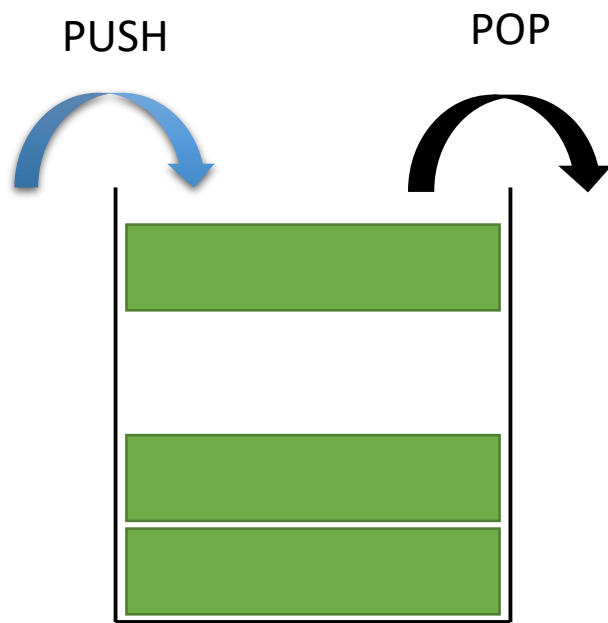
```
//获取栈顶端的元素
```

```
var item = names.pop(); // Alex  
alert(item);  
alert(names.length); //2
```

```
var names = ["Tom","Jerry"];
```

```
names.push("Alex");  
alert(names); // Tom,Jerry,Alex  
names[3] = "Rose";
```

```
var item = names.pop();  
alert(item); //Rose
```



LIFO (Last In First Out)

数组中的队列方法

凯盛软件

```
var names = [];  
names.push("Tom", "Jerry");  
var item = names.shift();  
alert(item); // Tom  
alert(names.length); //1
```



FIFO(First In First Out)

```
var names = [];  
var count = names.unshift("Tom", "Jerry");  
alert(names); //Tom, Jerry  
names.unshift("Alex");  
alert(names); //Alex, Tom, Jerry
```

```
var item = names.pop();  
alert(item); //Jerry
```



- 反转数组

```
var nums = [1,2,3,4,5];  
nums.reverse(); //反转数组  
alert(nums); //5,4,3,2,1
```

- 自然排序

```
var nums = [1,10,23,5,15];  
nums.sort();  
alert(nums); //1,10,15,23,5
```

- 根据排序函数排序

```
function compare(value1,value2) {  
    if(value1 < value2) {  
        return -1;  
    } else if(value1 > value2) {  
        return 1;  
    } else {  
        return 0;  
    }  
}
```

```
var nums = [1,10,23,5,15];  
nums.sort(compare);  
alert(nums); //1,5,10,15,23
```

比较函数简写

```
function compare(value1,value2) {  
    return value1 - value2;  
}
```

```
var nums = [1,10,23,5,15];  
nums.sort(compare);  
alert(nums); //1,5,10,15,23
```

- **排序函数规则**：如果第一个参数应该在第二个参数之前则返回一个负数，如果两个参数相等则返回0，如果第一个参数应该在第二个参数之后则应该返回一个正数

- **concat()**：基于当前数组中的所有项创建一个新数组。该方法会创建当前数组的一个副本，然后将接收到参数添加到这个副本的末尾。

```
var names = ["Tom", "Jerry"];
```

```
var names2 = names.concat(); //拷贝names数组的副本  
alert(names2); //Tom, Jerry
```

```
var names3 = names.concat("Alex", "Rose");  
alert(names3); //Tom, Jerry, Alex, Rose
```

```
var names4 = names.concat(["Lily", "Lucy"]);  
alert(names4); //Tom, Jerry, Lily, Lucy
```

```
var names5 = names.concat("Jim", ["Lily", "Lucy"]);  
alert(names5); //Tom, Jerry, Jim, Lily, Lucy
```

- **slice()**：基于当前数组中的一个或多个项创建一个新的数组。该方法接受一个或两个参数，分别代表返回项的起始和结束位置。如果只有一个参数，则从当前位置到数组结束。该方法操作不会影响到原数组

```
var names = ["Tom","Jerry","Lily","Lucy"];
```

```
var names2 = names.slice(1,3);  
alert(names2); //Jerry,Lily
```

```
var names3 = names.slice(2);  
alert(names3); //Lily, Lucy
```

```
var names4 = names.slice(-3,-1);  
alert(names4); //Jerry,Lily
```

- **splice()** : 可以对数组中元素进行插入、替换和删除操作。
 - **删除操作** : 指定两个参数(要删除第一项的位置和要删除的项数)

```
var names = ["Tom", "Jerry", "Lily", "Lucy"];

var removed = names.splice(0, 2);
alert(names); //Lily, Lucy
alert(removed); //Tom, Jerry
```
 - **插入操作** : 指定三个参数(起始位置, 0(要删除的项数), 要插入的项)

```
var names = ["Tom", "Jerry", "Lily", "Lucy"];

names.splice(1, 0, "Jim", "Rose");
alert(names); //Tom, Jim, Rose, Jerry, Lily, Lucy
```
 - **替换操作** : 指定三个参数(起始位置, 要删除的项数, 要插入的项)

```
var names = ["Tom", "Jerry", "Lily", "Lucy"];

names.splice(1, 2, "Jim", "Rose");
alert(names); //Tom, Jim, Rose, Lucy
```

- `indexOf()`和`lastIndexOf()`：根据下标查找指定的元素。找不到元素，该方法返回-1

```
var nums = [1,24,5,4,6,2,4,7,3];  
var startIndex = nums.indexOf(4); //3  
var lastIndex = nums.lastIndexOf(4); //6  
var index = nums.indexOf(4,5); //6 从下标为5的地方开始找4
```

```
var names = ["Tom", "Jerry", "Lily", "Lucy"];
```

```
for(var i = 0; i < names.length; i++) {  
    console.log(names[i]);  
}
```

```
for(var index in names) {  
    console.log(names[index]);  
}
```

```
names.forEach(function(item, index, array){  
    console.log(item);  
});
```



```
var now = new Date(); //获取到当前的日期和时间
```

```
//表示2012年12月21日 12:00:34
```

```
var endDate = new Date(Date.parse("2012-12-21 12:00:34"));
```

```
var endDate2 = new Date("2012-12-21 12:00:34");
```

```
var endDate3 = new Date(Date.UTC(2012,11,21,12,0,34));
```

```
var endDate4 = new Date(2012,11,21,12,0,34);
```



Date类型其他方法

凯盛软件

方法	描述	FF	IE
Date()	返回当日的日期和时间。	1	3
getDate()	从 Date 对象返回一个月中的某一天 (1 ~ 31)。	1	3
getDay()	从 Date 对象返回一周中的某一天 (0 ~ 6)。	1	3
getMonth()	从 Date 对象返回月份 (0 ~ 11)。	1	3
getFullYear()	从 Date 对象以四位数字返回年份。	1	4
getYear()	请使用 getFullYear() 方法代替。	1	3
getHours()	返回 Date 对象的小时 (0 ~ 23)。	1	3
getMinutes()	返回 Date 对象的分钟 (0 ~ 59)。	1	3
getSeconds()	返回 Date 对象的秒数 (0 ~ 59)。	1	3
getMilliseconds()	返回 Date 对象的毫秒 (0 ~ 999)。	1	4
getTime()	返回 1970 年 1 月 1 日至今的毫秒数。	1	3
getTimezoneOffset()	返回本地时间与格林威治标准时间 (GMT) 的分钟差。	1	3
getUTCDate()	根据世界时从 Date 对象返回月中的一天 (1 ~ 31)。	1	4
getUTCDay()	根据世界时从 Date 对象返回周中的一天 (0 ~ 6)。	1	4
getUTCMonth()	根据世界时从 Date 对象返回月份 (0 ~ 11)。	1	4
getUTCFullYear()	根据世界时从 Date 对象返回四位数的年份。	1	4
getUTCHours()	根据世界时返回 Date 对象的小时 (0 ~ 23)。	1	4
getUTCMinutes()	根据世界时返回 Date 对象的分钟 (0 ~ 59)。	1	4
getUTCSeconds()	根据世界时返回 Date 对象的秒数 (0 ~ 59)。	1	4
getUTCMilliseconds()	根据世界时返回 Date 对象的毫秒 (0 ~ 999)。	1	4
parse()	返回 1970 年 1 月 1 日午夜到指定日期 (字符串) 的毫秒数。	1	3

http://www.w3school.com.cn/js/jsref_obj_date.asp

setDate()	设置 Date 对象中月的某一天 (1 ~ 31)。	1	3
setMonth()	设置 Date 对象中月份 (0 ~ 11)。	1	3
setFullYear()	设置 Date 对象中的年份 (四位数字)。	1	4
setYear()	请使用 setFullYear() 方法代替。	1	3
setHours()	设置 Date 对象中的小时 (0 ~ 23)。	1	3
setMinutes()	设置 Date 对象中的分钟 (0 ~ 59)。	1	3
setSeconds()	设置 Date 对象中的秒钟 (0 ~ 59)。	1	3
setMilliseconds()	设置 Date 对象中的毫秒 (0 ~ 999)。	1	4
setTime()	以毫秒设置 Date 对象。	1	3
setUTCDate()	根据世界时设置 Date 对象中月份的一天 (1 ~ 31)。	1	4
setUTCMonth()	根据世界时设置 Date 对象中的月份 (0 ~ 11)。	1	4
setUTCFullYear()	根据世界时设置 Date 对象中的年份 (四位数字)。	1	4
setUTCHours()	根据世界时设置 Date 对象中的小时 (0 ~ 23)。	1	4
setUTCMinutes()	根据世界时设置 Date 对象中的分钟 (0 ~ 59)。	1	4
setUTCSeconds()	根据世界时设置 Date 对象中的秒钟 (0 ~ 59)。	1	4
setUTCMilliseconds()	根据世界时设置 Date 对象中的毫秒 (0 ~ 999)。	1	4
toSource()	返回该对象的源代码。	1	-
toString()	把 Date 对象转换为字符串。	1	4

toString()	把 Date 对象的时间部分转换为字符串。	1	4
toDateString()	把 Date 对象的日期部分转换为字符串。	1	4
toGMTString()	请使用 toUTCString() 方法代替。	1	3
toUTCString()	根据世界时，把 Date 对象转换为字符串。	1	4
toLocaleString()	根据本地时间格式，把 Date 对象转换为字符串。	1	3
toLocaleTimeString()	根据本地时间格式，把 Date 对象的时间部分转换为字符串。	1	3
toLocaleDateString()	根据本地时间格式，把 Date 对象的日期部分转换为字符串。	1	3
UTC()	根据世界时返回 1970 年 1 月 1 日 到指定日期的毫秒数。	1	3
valueOf()	返回 Date 对象的原始值。	1	4

第三方的Date工具类

凯盛软件

<http://momentjs.com/>

<http://www.datejs.com/>

正则表达式 (RegExp)

JavaScript中正则表达式格式如下：

```
var expression = /正则表达式部分/模式部分;
```

正则表达式部分可以是任何简单或复杂的表达式。

每个正则表达式可以带一个或多个模式，模式如下：

- **g**：代表全局模式，即模式将被应用于所有的字符串，而不是发现第一个匹配项时就停止
- **i**：表示不区分大小写
- **m**：表示多行模式，即在到达一行文本末尾时还会继续查找下一行中是否存在相匹配的项

正则表达式中的**元字符**需要转义，元字符包括：

`() [] {} \ ^ $ | ? * + .`

属性	描述	FF	IE
<u>E</u>	返回算术常量 e，即自然对数的底数（约等于2.718）。	1	3
<u>LN2</u>	返回 2 的自然对数（约等于0.693）。	1	3
<u>LN10</u>	返回 10 的自然对数（约等于2.302）。	1	3
<u>LOG2E</u>	返回以 2 为底的 e 的对数（约等于 1.414）。	1	3
<u>LOG10E</u>	返回以 10 为底的 e 的对数（约等于0.434）。	1	3
<u>PI</u>	返回圆周率（约等于3.14159）。	1	3
<u>SQRT1_2</u>	返回返回 2 的平方根的倒数（约等于 0.707）。	1	3
<u>SQRT2</u>	返回 2 的平方根（约等于 1.414）。	1	3

方法	描述	FF	IE
abs(x)	返回数的绝对值。	1	3
acos(x)	返回数的反余弦值。	1	3
asin(x)	返回数的反正弦值。	1	3
atan(x)	以介于 $-\pi/2$ 与 $\pi/2$ 弧度之间的数值来返回 x 的反正切值。	1	3
atan2(y,x)	返回从 x 轴到点 (x,y) 的角度（介于 $-\pi/2$ 与 $\pi/2$ 弧度之间）。	1	3
ceil(x)	对数进行上舍入。	1	3
cos(x)	返回数的余弦。	1	3
exp(x)	返回 e 的指数。	1	3
floor(x)	对数进行下舍入。	1	3
log(x)	返回数的自然对数（底为 e ）。	1	3
max(x,y)	返回 x 和 y 中的最高值。	1	3
min(x,y)	返回 x 和 y 中的最低值。	1	3
pow(x,y)	返回 x 的 y 次幂。	1	3
random()	返回 $0 \sim 1$ 之间的随机数。	1	3
round(x)	把数四舍五入为最接近的整数。	1	3
sin(x)	返回数的正弦。	1	3
sqrt(x)	返回数的平方根。	1	3
tan(x)	返回角的正切。	1	3
toSource()	返回该对象的源代码。	1	-
valueOf()	返回 Math 对象的原始值。	1	4

BOM

window对象是BOM的核心对象，它表示一个浏览器的实例。在浏览器中，window对象具有双重角色，既是通过JavaScript访问浏览器窗口的一个接口，有时ECMAScript中的全局对象。

```
var age = 10;
alert(age);
alert(window.age);

function test() {
    alert("window");
}
test();
window.test();
```

open()方法用于弹出窗口

```
window.open("https://www.google.com.hk");  
window.open("https://www.google.com.hk", "mywindow", "height=400,width=400,top=100,left=100,resizable=no");|
```

关闭弹出窗口

```
document.getElementById("btn").onclick = function(){  
    win.close();  
};
```

```
setTimeout(function(){  
    alert("hehe");  
},2000);
```

```
var count = 0;  
function test(){  
    count++;  
    alert("hehe");  
    var t = setTimeout(test,2000);  
    if(count == 3) {  
        clearTimeout(t);  
    }  
}
```

```
test();|
```

```
setInterval(function(){  
    alert("hehe");  
},2000);
```

```
var count = 0;  
var t = setInterval(function(){  
    count++;  
    if(count == 3) {  
        clearInterval(t);  
    }  
    alert("hehe");  
},2000);
```

```
alert("你好");
```

```
if(confirm("你确定要删除吗?")){  
    alert("删除成功! ");  
} else {  
    alert("已取消操作");  
}
```

```
var num1 = prompt("请输入第一个数字:");  
var num2 = prompt("请输入第二个数字:",0);
```

```
alert(parseInt(num1) + parseInt(num2));
```

```
window.print();|
```

浏览器显示尺寸：

`window.innerHeight`

`window.innerWidth`

浏览器窗口尺寸：

`window.outerHeight`

`window.outerWidth`

浏览器 resize 事件：

`window.onresize=function(){`

`}`

navigator 对象

凯盛软件

```
console.log("appName:" + navigator.appName);  
console.log("appCodeName:" + navigator.appCodeName);  
console.log("appVersion:" + navigator.appVersion);  
console.log("language:" + navigator.language);  
console.log("platform:" + navigator.platform);  
console.log("userAgent:" + navigator.userAgent);
```

<http://litten.github.io/2014/09/26/history-of-browser-useragent/>

screen 对象

凯盛软件

```
console.log("width:" + screen.width);  
console.log("height:" + screen.height);  
console.log("colorDepth:" + screen.colorDepth); //颜色位数 8 16 24
```

location对象

属性	描述	IE	F	O
hash	设置或返回从井号 (#) 开始的 URL (锚)。	4	1	9
host	设置或返回主机名和当前 URL 的端口号。	4	1	9
hostname	设置或返回当前 URL 的主机名。	4	1	9
href	设置或返回完整的 URL。	4	1	9
pathname	设置或返回当前 URL 的路径部分。	4	1	9
port	设置或返回当前 URL 的端口号。	4	1	9
protocol	设置或返回当前 URL 的协议。	4	1	9
search	设置或返回从问号 (?) 开始的 URL (查询部分)。	4	1	9

```
alert(location.protocol);|
```

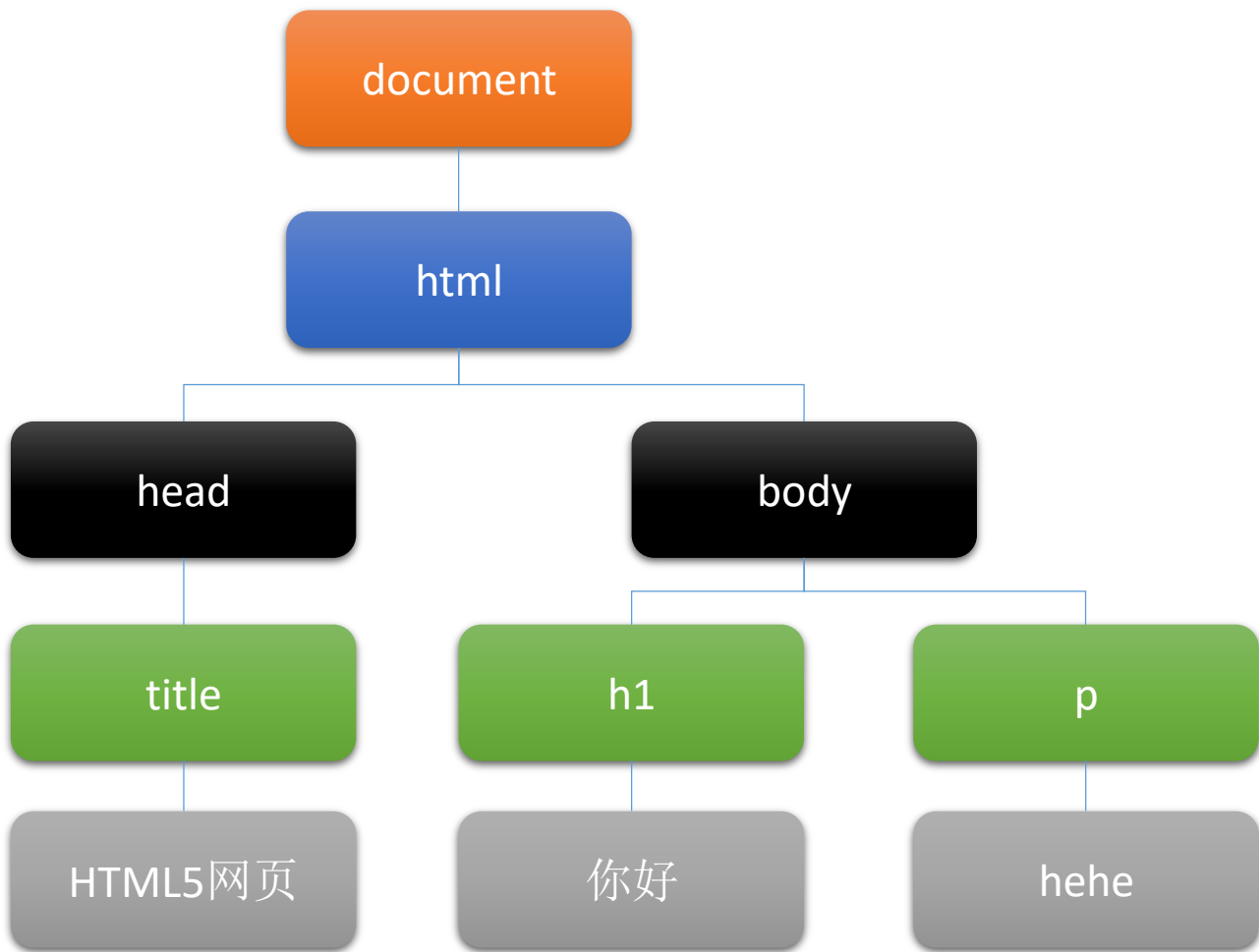
方法	描述	IE	F	O
back()	加载 history 列表中的前一个 URL。	4	1	9
forward()	加载 history 列表中的下一个 URL。	4	1	9
go()	加载 history 列表中的某个具体页面。	4	1	9

```
history.back();  
history.forward();  
history.go(1);  
history.go(-1);
```

HTML DOM



```
<!DOCTYPE HTML>
<html>
<head>
  <title>HTML5网页</title>
</head>
<body>
  <h1>你好</h1>
  <p>hehe</p>
</body>
</html>
```



```
<input type="button" value="btn1">
<input type="button" value="btn2" id="btn">
<input type="button" value="btn3" name="btn">

<script type="text/javascript">

    (function(){
        //根据input标签获取元素
        var btn1 = document.getElementsByTagName("input")[0];
        //根据ID属性获取元素
        var btn2 = document.getElementById("btn");
        //根据name属性获取元素
        var btn3 = document.getElementsByName("btn")[0];
        alert(btn1);
        alert(btn2);
        alert(btn3);

    })();
</script>
```

获取访问对象

凯盛软件

```
<div>  
  <p></p>  
</div>  
<p>zzz</p>
```

//获取div元素中的p元素

```
var div_p = document.getElementsByTagName("div")[0].getElementsByTagName("p")[0];  
div_p.innerHTML = "xxx";
```


Todo List

凯盛软件

Todo List

Add Todo

☒ 做十二月的工作计划

☐ 给Tom打电话联系

☐ 完成十一月工作总结

☐ 修复PMS系统的bug

<https://gist.github.com/4180488>

```
//创建HTML对象
var ck = document.createElement("input");
//创建文本对象
var txt = document.createTextNode(todoValue);
//给对象设置属性
ckTd.setAttribute("width", "20");
ck.setAttribute("type", "checkbox");
//添加HTML元素到父元素中
ckTd.appendChild(ck);
```

获取父节点：

`element.parentNode`

获取子节点集合：

`element.childNodes`

获取第一个子节点

`element.firstChild`

获取最后一个子节点

`element.lastChild`

获取当前节点的前一个节点

`element.previousSibling`

获取当前节点的后一个节点

`element.nextSibling`

在已知节点之前插入一个新的节点

`insertBefore(newnode,nowNode);`

将子节点替换为另一个节点

`replaceChild(newNode,oldNode);`

删除指定节点

`removeChild(node);`

复制节点

被复制节点.`cloneNode(true/false);`//true:复制当前节点及其子节点；false:仅复制当前节点

```
document.getElementById("btn").onclick=function(){  
    var div = document.getElementById("mydiv");  
    div.style.width = "300px";  
    div.style.height = "400px";  
    div.style.backgroundColor = "#ff7400";  
};
```

css中使用短划线分割的属性，例如background-color，必须在JavaScript中转换为驼峰命名法，
backgroundColor

Dom获取当前样式表的值

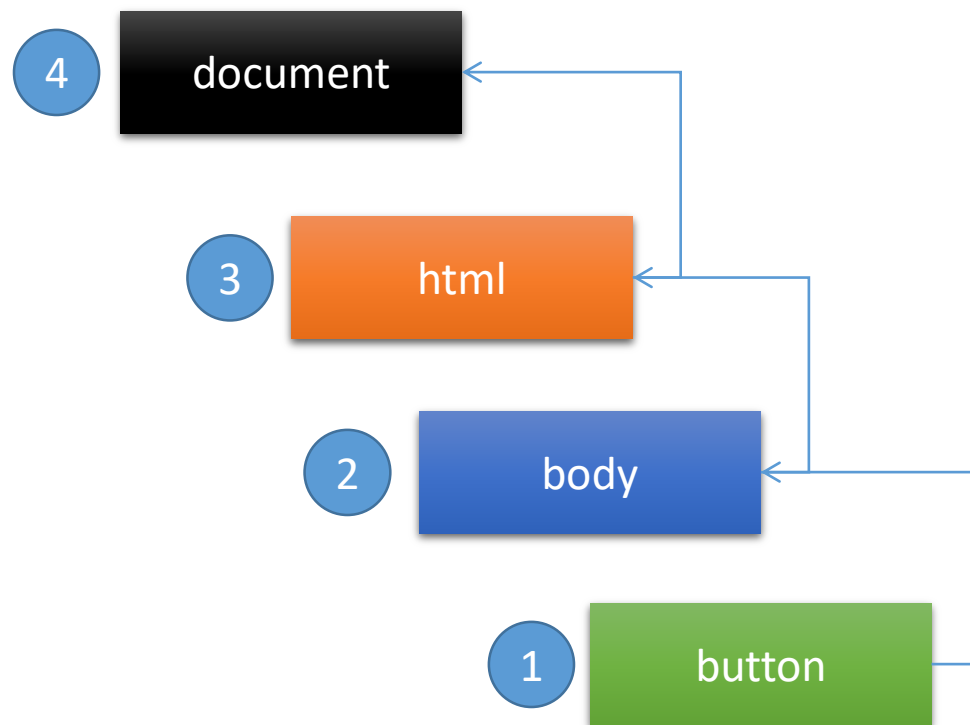
```
var div = document.getElementById("mydiv");

//获取行内样式表的值
alert(div.style.width);

//获取内嵌样式表及外部样式的值
if(document.defaultView) { //no ie
    var style = document.defaultView.getComputedStyle(div,null);
    alert(style.width);
}else if(div.currentStyle) { //ie
    alert(div.currentStyle.width);
}
```

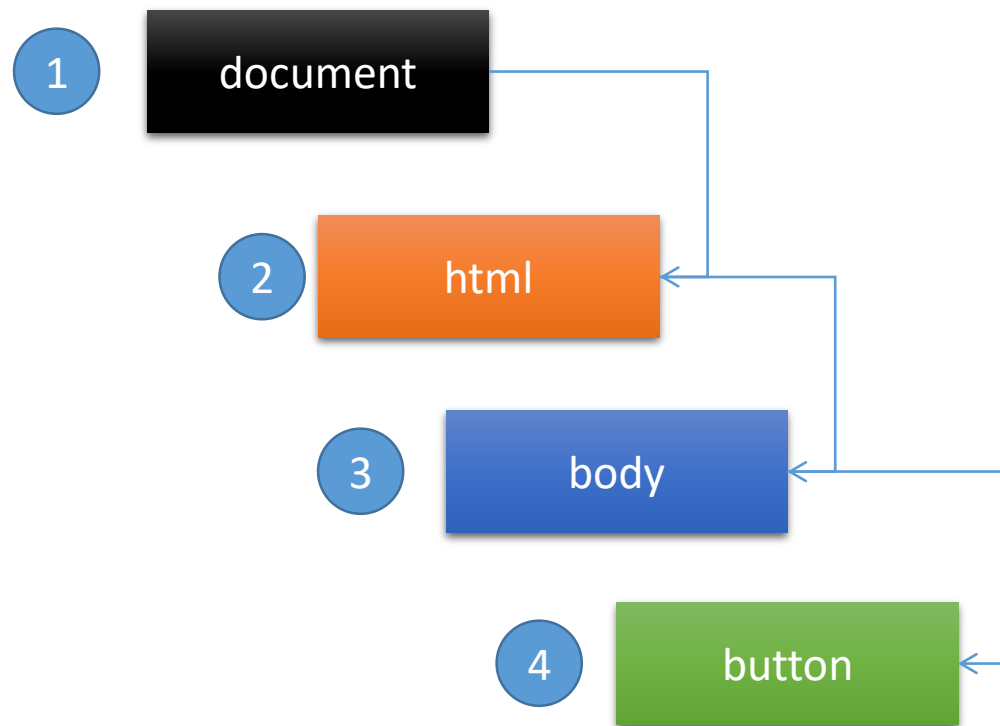
事件流：事件冒泡

凯盛软件



事件流：事件捕获

凯盛软件



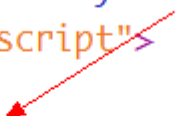
事件捕获

支持浏览器：IE9 Safari Chrome Opera Firefox


```
<button id="btn" onclick="sayHello()">btn</button>
```

```
<script type="text/javascript">
```

```
function sayHello() {  
    alert("hello,Button");  
}
```



```
<button id="btn">btn</button>
<script type="text/javascript">
    (function(){
        document.getElementById("btn").onclick = function() {
            alert("hello,Button");
        }
    })

```

```
document.getElementById("btn").addEventListener("click",function(){
    alert("Hello,Button");
},false); //第二个参数为true时代表在捕获阶段执行，为false时代表在冒泡阶段执行
```

//可以为同一个元素添加多个事件处理函数

```
document.getElementById("btn").addEventListener("click",function(){
    alert("Hello,Button2");
},false);
```

```
function handler() {
    alert("Hello,Button");
}
document.getElementById("btn").addEventListener("click",handler,false);
//删除事件
document.getElementById("btn").removeEventListener("click",handler,false);
```

IE9 Chrome Firefox Safari Opera支持

```
document.getElementById("btn").attachEvent("onclick",function(){  
    alert("Hello,Button");  
});
```

//添加多个事件，按照添加的相反顺序执行

```
document.getElementById("btn").attachEvent("onclick",function(){  
    alert("Hello,Button2");  
});
```

```
function handler() {  
    alert("Hello,Button");  
    alert(this == window);//在IE事件处理中，this不是当前点击的元素，而是window对象  
}
```

```
document.getElementById("btn").attachEvent("onclick",handler);  
//删除事件  
document.getElementById("btn").detachEvent("onclick",handler);
```

跨浏览器的事件处理程序

凯盛软件

```
var EventUtil = {
  addHandler : function(element,type,handler){
    if(element.addEventListener) {
      element.addEventListener(type,handler,false);
    } else if(element.attachEvent) {
      element.attachEvent("on"+type,handler);
    } else {
      element["on"+type] = handler;
    }
  },
  removeHandler : function(element,type,handler){
    if(element.removeEventListener) {
      element.removeEventListener(type,handler,false);
    } else if(element.detachEvent) {
      element.detachEvent("on"+type,handler);
    } else {
      element["on"+type] = null;
    }
  }
};

function handler() {
  alert("Hello,Button");
}
var btn = document.getElementById("btn");
EventUtil.addHandler(btn,"click",handler);
EventUtil.removeHandler(btn,"click",handler);
```

在触发某个事件时，会产生一个event对象，该对象中包含了所有与事件有关的信息。包括导致事件的元素，事件类型以及其他与特定事件相关的信息。所有浏览器都支持event对象，但是支持的方式不同。

```
var btn = document.getElementById("btn");
btn.onclick = function(event){
    alert(event.type); //获取事件类型 click
    alert(event.target); //获取触发事件的目标元素 button
};
```

```
function handler(event) {  
    switch(event.type) {  
        case "click":  
            alert("Click me");  
            break;  
        case "mouseover" :  
            event.target.style.backgroundColor = "blue";  
            break;  
        case "mouseout" :  
            event.target.style.backgroundColor = "";  
            break;  
    }  
}
```

```
var btn = document.getElementById("btn");  
btn.onclick = handler;  
btn.onmouseover = handler;  
btn.onmouseout = handler;
```

preventDefault()方法用于取消事件的特定行为。只有cancelable属性为true的事件，才能取消。

```
<a href="https://www.google.com.hk" id="mylink">Google</a>
```

```
<script type="text/javascript">
  (function(){
    document.getElementById("mylink").onclick = function(event){
      event.preventDefault();
    };
  });
</script>
```

stopPropagation()方法用于停止事件继续传播。

```
document.getElementById("btn").onclick = function(event){
  alert("Button");
  event.stopPropagation();
};
document.body.onclick = function(){
  alert("Body");
};
```



```
document.getElementById("btn").onclick = function(){  
    //event对象不能通过参数获得  
    var event = window.event;  
    alert(event.type); //获取事件类型  
    alert(event.srcElement); //获取触发事件的目标元素，与target相同  
    event.cancelBubble = true; //取消事件冒泡  
    event.returnValue = false; //取消默认行为  
};  
  
document.getElementById("btn").attachEvent("onclick", function(event){  
    alert(event.srcElement); //event可以从参数中获得，也可以使用window.event  
});
```

```
var EventUtil = {
  addHandler : function(element,type,handler){
    },
  removeHandler : function(element,type,handler){
    },
  getEvent : function(event){
    return event ? event : window.event;
  },
  getTarget : function(e){
    if(e.target) {
      return e.target;
    } else {
      return e.srcElement;
    }
  },
  preventDefault : function(e){
    if(e.preventDefault) {
      e.preventDefault();
    } else {
      e.returnValue = false;
    }
  },
  stopPropagation : function(e){
    if(e.stopPropagation) {
      e.stopPropagation();
    } else {
      e.cancelBubble = true;
    }
  }
};
```

```
function handler() {
  alert("Hello,Button");
  var e = EventUtil.getEvent(event);
  var target = EventUtil.getTarget(e);
  alert(target);
}
var btn = document.getElementById("btn");
EventUtil.addHandler(btn,"click",handler);
```

属性	此事件发生在何时...	IE	F	O	W3C
onabort	图像的加载被中断。	4	1	9	Yes
onblur	元素失去焦点。	3	1	9	Yes
onchange	域的内容被改变。	3	1	9	Yes
onclick	当用户点击某个对象时调用的事件句柄。	3	1	9	Yes
ondblclick	当用户双击某个对象时调用的事件句柄。	4	1	9	Yes
onerror	在加载文档或图像时发生错误。	4	1	9	Yes
onfocus	元素获得焦点。	3	1	9	Yes
onkeydown	某个键盘按键被按下。	3	1	No	Yes
onkeypress	某个键盘按键被按下并松开。	3	1	9	Yes
onkeyup	某个键盘按键被松开。	3	1	9	Yes
onload	一张页面或一幅图像完成加载。	3	1	9	Yes
onmousedown	鼠标按钮被按下。	4	1	9	Yes
onmousemove	鼠标被移动。	3	1	9	Yes
onmouseout	鼠标从某元素移开。	4	1	9	Yes
onmouseover	鼠标移到某元素之上。	3	1	9	Yes
onmouseup	鼠标按键被松开。	4	1	9	Yes
onreset	重置按钮被点击。	4	1	9	Yes
onresize	窗口或框架被重新调整大小。	4	1	9	Yes
onselect	文本被选中。	3	1	9	Yes
onsubmit	确认按钮被点击。	3	1	9	Yes
onunload	用户退出页面。	3	1	9	Yes

页面完全加载后，包括所有的图像，js文件，css文件加载完毕后，就会触发onload事件。

```
EventUtil.addHandler(window,"load",function(event){  
    alert("hello");  
});
```

```
<body onload="handler()">
```

```
    <button id="btn">btn</button>
```

```
    <a href="https://www.google.com.hk" id="mylink">Google</a>
```

```
</body>
```

- onfocus : 获得焦点

```
var text = document.getElementById("username");
text.onfocus = function(){
    document.getElementById("help").innerHTML = "获取焦点";
};
```

- onblur : 失去焦点

```
text.onblur = function() {
    document.getElementById("help").innerHTML = "失去焦点";
};
```

- onclick : 单击事件
- ondblclick : 双击事件
- onmousedown : 按下鼠标任意一个按键触发该事件
- onmouseup : 释放鼠标按钮时触发该事件
- onmouseover : 鼠标移到元素上触发该事件
- onmouseout : 鼠标从元素上离开触发该事件
- onmousemove : 鼠标在元素内部移动时触发该事件



- onkeydown : 按下键盘上任意键是触发该事件
- onkeypress : 按下字符键和esc键时触发该事件
- onkeyup : 是否键盘上的键时触发
- 获取按下的键值 :

```
text.onkeydown = function(event){  
    var e = EventUtil.getEvent(event);  
    span.innerHTML = e.keyCode;  
};
```

限制输入框中只能输入数字：

```
text.onkeydown = function(event){  
    var e = EventUtil.getEvent(event);  
    if(!(e.keyCode >= 48 && e.keyCode <= 57) && !(e.keyCode >= 96 && e.keyCode <= 105) && e.  
        keyCode != 8 && e.keyCode != 46 && e.keyCode != 37 && e.keyCode != 39){  
        EventUtil.preventDefault(e);  
    }  
};
```

屏蔽右键菜单：

```
text.oncontextmenu = function(event){  
    var e = EventUtil.getEvent(event);  
    EventUtil.preventDefault(e);  
};
```

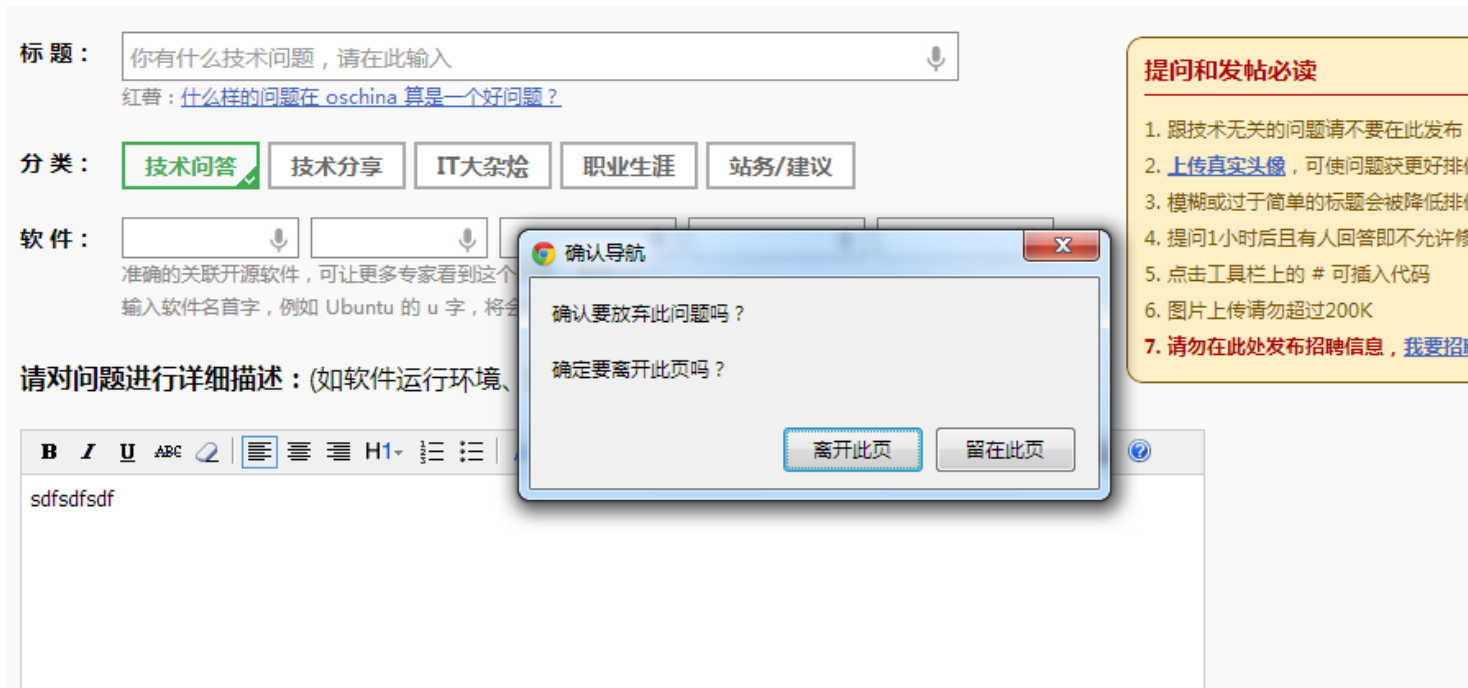


```
<ul id="ctxMenu" style="position:absolute;visibility:hidden;background-color:gray">  
  <li><a href="#">菜单1</a></li>  
  <li><a href="">菜单2</a></li>  
  <li><a href="">菜单3</a></li>  
</ul>
```

```
text.oncontextmenu = function(event){
    var e = EventUtil.getEvent(event);
    EventUtil.preventDefault(e);

    var pageX = e.pageX;
    var pageY = e.pageY;
    //IE
    if(pageX == undefined) {
        pageX = e.clientX + (document.body.scrollLeft || document.documentElement.scrollLeft);
    }
    if(pageY == undefined) {
        pageY = e.clientY + (document.body.scrollTop || document.documentElement.scrollTop);
    }

    var menu = document.getElementById("ctxMenu");
    menu.style.left = pageX + "px";
    menu.style.top = pageY + "px";
    menu.style.visibility = "visible";
};
document.onclick = function(){
    document.getElementById("ctxMenu").style.visibility = "hidden";
};
```



```
var text = document.getElementById("username");
window.onbeforeunload = function(){
    var v = text.value;
    if(v) {
        return "确认要放弃发帖吗? ";
    }
};
```

该事件在浏览器卸载页面前触发

表单

- 获取form

```
var fm = document.getElementById("myform");
```

```
var fm = document.forms[0];
```

```
var fm = document.forms["fm"];
```

```
var fm = document.fm;
```

- form常用方法和属性

- length : 表单中控件的数量
- action : 接受请求的URL地址
- method : 发送的HTTP请求类型
- name : 表单的name值
- target : 发送请求和接受响应的窗口的名称
- reset() : 将表单重置为默认值
- submit() : 提交表单

```
<form action="login.jsp" id="myform" name="fm">
    用户名: <input type="text" name="username" id="username"/>
    <input type="submit" value="提交">
</form>
```

```
var fm = document.getElementById("myform");
EventUtil.addHandler(fm, "submit", function(event){
    //验证表单
    var username = document.getElementById("username").value;
    if(!username) {
        alert("用户名必填");
        //阻止表单提交
        var e = EventUtil.getEvent(event);
        EventUtil.preventDefault(e);
    }
});
```

```
<form action="login.jsp" id="myform" onsubmit="return validate()">
    用户名: <input type="text" name="username" id="username"/>
    <input type="submit" value="提交">
</form>
```

```
<script type="text/javascript">

    function validate() {
        var username = document.getElementById("username").value;
        if(!username) {
            alert("用户名必填");
            //阻止表单提交
            return false;
        }
        return true;
    }
}
```

```
<form action="login.jsp" id="myform" onsubmit="return validate()">
  用户名: <input type="text" name="username" id="username"/>
  <input type="button" id="btn" value="提交">
</form>
```

```
document.getElementById("btn").onclick = function(){
  var username = document.getElementById("username").value;
  if(!username) {
    alert("用户名必填");
    //阻止表单提交
    return;
  }
  document.getElementById("myform").submit();
};
```


防止表单重复提交

```
document.getElementById("btn").onclick = function(){
    var username = document.getElementById("username").value;
    if(!username) {
        alert("用户名必填");
        //阻止表单提交
        return;
    }
    document.getElementById("myform").submit();
    this.disabled = "disabled"; ←
};
```

```
document.getElementById("btn").onclick = function(){
    var username = document.getElementById("username").value;
    if(!username) {
        alert("用户名必填");
        //阻止表单提交
        return;
    }
    document.getElementById("myform").submit();
    this.onclick = null; ←
};
```

```
var fm = document.getElementById("myform");  
var element = fm.elements[0]; //获取表单中的第一个表单元素  
alert(fm.elements.length); //获取表单中表单元素的数量  
var username = fm.elements["username"]; //获取name属性为username的表单元素
```

```
<input type="radio" name="fav" id="">  
<input type="radio" name="fav" id="">  
<input type="radio" name="fav" id="">
```

```
var fm = document.getElementById("myform");  
var radios = fm.elements["fav"]; //返回的是name为fav的数组  
alert(radios.length);
```

- disabled : 当前元素是否禁用 , boolean值
- form : 当前元素所属表单的指针
- name : 当前元素的name值
- readonly : 当前元素是否为只读 , boolean值
- tabIndex : tab切换的序号
- type : 当前字段的类型 , 例如 “text” , “checkbox”
- value : 提交给服务器的值
- focus () : 获得焦点方法
- blur () : 失去焦点方法

- onfocus() : 获得焦点时触发
- onblur() : 失去焦点时触发
- onchange() : 失去焦点并内容发生改变时触发

```
document.forms[0].elements[0].onmouseover = function(){  
    this.select();  
};
```

- `add(option,relOption)` : 在select中添加新的option选项，在relOption之前
- `multiple` : boolean值，表示是否允许多选
- `options` : 所有option元素的数组
- `remove(index)` : 移除index位置的option
- `selectedIndex` : 选中选项的索引，从0开始
- `size` : 可见的行数
- `text` : 选项文本
- `value` : 选项的value值
- `index` : 当前选项的索引

JQuery

将网站发布到互联网上

- 域名
- 服务器
- 网站代码

- 最好选择.com为后缀的域名，最好不要选择.cn后缀的域名
- 域名要好记好写
- 域名注册商
 - Godaddy (<http://www.godaddy.com/>)
 - Name (<http://www.name.com/>)
 - 万网 (<http://www.net.cn>)

- 要有固定的IP地址
- 服务器选择
 - VPS
 - cloud server（GAE，SAE，BAE，AS3）
 - 自有主机