**JavaScript篇**

1. **使用javascript.**

1、Javascript位置在<head></head>之间：

格式为<script type=”text/javascript” src=”demo.js”></script>

2、在script中间的代码无法运行：

<script type="text/javascript" src="demo1.js">alert('我很可怜，执行不到！')</script>

1. **语法，关键保留字及变量.**

1、Javascript区分大小写：

Text和text是两种不同的标识符

2、标识符的开头必须是字母，下划线或美元符号；其他标识符可以除了它们还可以是数字：

a、\_、$ ； 1

3、不能把关键字，保留字，true、false、null作为标识符

4、注释方法：

①单行注释：

//alert(‘标识符’);

②多行注释：

/\*

Alert(‘标识符’);

\*/

5、字面量，就是程序中直接显示出来的数据值，包括：

100 数字字面量； ‘你好’ 字符串字面量； false 布尔字面量；

/js/gi 正则表达式字面量； null 对象字面量；

字面量表达式：

{x:1, y:2} 对象字面量表达式； [1,2,3,4,5] 数组字面量表达式

6、关键字和保留字：

关键字是程序中已经开始使用的字符，如var。保留字指目前还没使用，但是以后可能会使用的字符。关键字和保留字不能作为标识符。

7、变量：

第一步：声明变量：var box

第二部：声明变量并初始化：var box=100

第三部：以弹窗方式输出box的值：

var box=100;

alert(box);

所谓变量，就是可以初始化后可以再次改变的量，所以以下写法：

var box=money;

box=100;

alert(box);

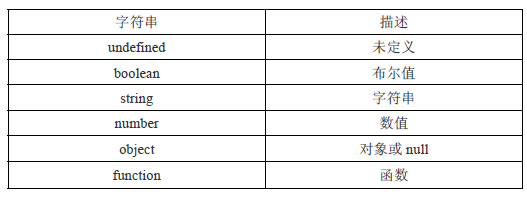
将输出值为100而不是money。

1. **数据类型：**

数据类型包括5种简单数据类型：Undefined、Null、Boolean、Number和String。还有一种复杂的数据类型：Object。

1. typeof操作符：

typeof 操作符是用来检测变量的数据类型。对于值或变量使用typeof 操作符会返回如下字符串：



如：

var box = '李炎恢';

alert(typeof box);

这个就是字符串，会返回string。

1. Undefined类型：

Undefined 类型只有一个值，即特殊的undefined。

var box;

alert(box);

当变量值没有的时候就是undefined。

1. Null类型：

Null 类型是一个只有一个值的数据类型，即特殊的值null。它表示一个空对象引用(指

针)，而typeof 操作符检测null 会返回object。

var box = null;

alert(typeof box);

1. Boolean类型：

Boolean类型有两个值(字面量)：true和false。而true不一定等于1，false不一定等于0。

JavaScript 是区分大小写的，True 和False 或者其他都不是Boolean 类型的值。

1. Number类型：
2. 最基本的数值字面量是十进制整数。

var box = 100; //十进制整数

1. 八进制数值字面量，(以8 为基数)，前导必须是0，八进制序列(0~7)。

var box = 070; //八进制，56

var box = 079; //无效的八进制，自动解析为79

var box = 08; //无效的八进制，自动解析为8

1. 十六进制字面量前面两位必须是0x，后面是(0~9 及A~F)。

var box = 0xA; //十六进制，10

var box = 0x1f; //十六进制，31

1. 浮点类型，就是该数值中必须包含一个小数点，并且小数点后面必须至少有一位数字。

var box = 3.8;

var box = 0.8;

var box = .8; //有效，但不推荐此写法

由于保存浮点数值需要的内存空间比整型数值大两倍，因此ECMAScript 会自动将可以

转换为整型的浮点数值转成为整型。

var box = 8.; //小数点后面没有值，转换为8

var box = 12.0; //小数点后面是0，转成为12

1. 对于那些过大或过小的数值，可以用科学技术法来表示(e 表示法)。用e 表示该数值的

前面10 的指数次幂。

var box = 4.12e9; //即4120000000

var box = 0.00000000412; //即4.12e-9

1. string类型：

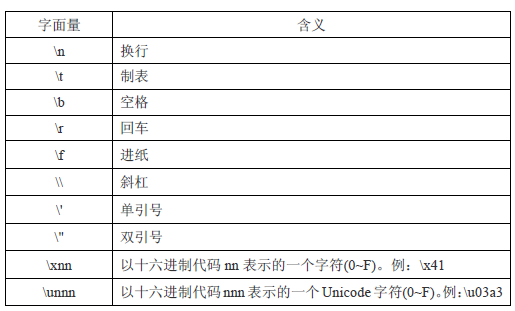
String 类型用于表示由于零或多个16 位Unicode 字符组成的字符序列，即字符串。字

符串可以由双引号(")或单引号(')表示。

var box = 'Lee';

var box = "Lee";

String 类型包含了一些特殊的字符字面量，也叫转义序列。



列子：

alert(“Le\ne”)

将打印出：

le

e

1. **运算符：**

一．什么是表达式

表达式是ECMAScript 中的一个“短语”，解释器会通过计算把它转换成一个值。最简

单的表达式是字面量或者变量名。例如：

5.96 //数值字面量

'Lee' //字符串字面量

true //布尔值字面量

null //空值字面量

/Java/ //正则表达式字面量

{x:1, y:2} //对象字面量、对象表达式

[1,2,3] //数组字面量、数组表达式

function(n) {return x+y;} //函数字面量、函数表达式

box //变量

当然，还可以通过合并简单的表达式来创建复杂的表达式。比如：

box + 5.96 //加法运算的表达式

typeof(box) //查看数据类型的表达式

box > 8 //逻辑运算表达式

二．一元运算符

只能操作一个值的运算符叫做一元运算符。

1.递增++和递减--

var box = 100;

++box; //把box 累加一个1，相当于box = box+1

--box; //把box 累减一个1，相当于box = box-1

box++; //同上

box--; //同上

2.前置和后置的区别

在没有赋值操作，前置和后置是一样的。但在赋值操作时，如果递增或递减运算符前置，

那么前置的运算符会先累加或累减再赋值，如果是后置运算符则先赋值再累加或累减。

var box = 100;

var age = ++box; //age 值为101

var height = box++; //height 值为100

三．算术运算符

ECMAScript 定义了5 个算术运算符，加减乘除求模(取余)。如果在算术运算的值不是

数值，那么后台会先使用Number()转型函数将其转换为数值(隐式转换)。

1.加法

var box = 1 + 2; //等于3

var box = 1 + NaN; //NaN，只要有一个NaN 就为NaN

var box = Infinity + Infinity; //Infinity

var box = -Infinity + -Infinity; //-Infinity

var box = Infinity + -Infinity; //NaN，正无穷和负无穷相加等NaN

var box = 100 + '100'; //100100，字符串连接符，有字符串就不是加法

var box = '您的年龄是：' + 10 + 20; //您的年龄是：1020，被转换成字符串

var box = 10 + 20 + '是您的年龄'; //30 是您的年龄，没有被转成字符串

var box = '您的年龄是：' + (10 + 20); //您的年龄是：30，没有被转成字符串

var box = 10 + 对象//10[object Object]，如果有toString()或valueOf()

则返回10+返回数的值

2.减法

var box = 100 - 70; //等于30

var box = -100 - 70 //等于-170

var box = -100 - -70 //-30，一般写成-100 - (-70)比较清晰

var box = 1 - NaN; //NaN，只要有一个NaN 就为NaN

var box = Infinity - Infinity; //NaN

var box = -Infinity - -Infinity; //NaN

var box = Infinity - -Infinity; //Infinity

var box = -Infinity - Infinity; //-Infinity

var box = 100 - true; //99，true 转成数值为1

var box = 100 - ''; //100，''转成了0

var box = 100 - '70'; //30，'70'转成了数值70

var box = 100 - null; //100，null 转成了0

var box = 100 - 'Lee'; //NaN，Lee 转成了NaN

var box = 100 - 对象//NaN，如果有toString()或valueOf()

则返回10-返回数的值

3.乘法

var box = 100 \* 70; //7000

var box = 100 \* NaN; //NaN，只要有一个NaN 即为NaN

var box = Infinity \* Infinity; //Infinity

var box = -Infinity \* Infinity ; //-Infinity

var box = -Infinity \* -Infinity ; //Infinity

var box = 100 \* true; //100，true 转成数值为1

var box = 100 \* ''; //0，''转成了0

var box = 100 \* null; //0，null 转成了0

var box = 100 \* 'Lee'; //NaN，Lee 转成了NaN

var box = 100 \* 对象//NaN，如果有toString()或valueOf()

则返回10 - 返回数的值

4.除法

var box = 100 / 70; //1.42....

var box = 100 / NaN; //NaN

var box = Infinity / Infinity; //NaN

var box = -Infinity / Infinity ; //NaN

var box = -Infinity / -Infinity; //NaN

var box = 100 / true; //100，true 转成1

var box = 100 / ''; //Infinity，

var box = 100 / null; //Infinity，

var box = 100 / 'Lee'; //NaN

var box = 100 / 对象; //NaN，如果有toString()或valueOf()

则返回10 / 返回数的值

5.求模

var box = 10 % 3; //1，余数为1

var box = 100 % NaN; //NaN

var box = Infinity % Infinity; //NaN

var box = -Infinity % Infinity ; //NaN

var box = 100 % true; //0

var box = 100 % ''; //NaN

var box = 100 % null; //NaN

var box = 100 % 'Lee'; //NaN

var box = 100 % 对象; //NaN，如果有toString()或valueOf()

则返回10 % 返回数的值

四．关系运算符

用于进行比较的运算符称作为关系运算符：小于(<)、大于(>)、小于等于(<=)、大于等

于(>=)、相等(==)、不等(!=)、全等(恒等)(===)、不全等(不恒等)(!==)

和其他运算符一样，当关系运算符操作非数值时要遵循一下规则：

1.两个操作数都是数值，则数值比较；

2.两个操作数都是字符串，则比较两个字符串对应的字符编码值；

3.两个操作数有一个是数值，则将另一个转换为数值，再进行数值比较；

4.两个操作数有一个是对象，则先调用valueOf()方法或toString()方法，再用结果比较；

var box = 3 > 2; //true

var box = 3 > 22; //false

var box = '3' > 22; //false

var box = '3' > '22'; //true

var box = 'a' > 'b'; //false a=97,b=98

var box = 'a' > 'B'; //true B=66

var box = 1 > 对象; //false，如果有toString()或valueOf()

则返回1 > 返回数的值

在相等和不等的比较上，如果操作数是非数值，则遵循一下规则：

1.一个操作数是布尔值，则比较之前将其转换为数值，false 转成0，true 转成1；

2.一个操作数是字符串，则比较之前将其转成为数值再比较；

3.一个操作数是对象，则先调用valueOf()或toString()方法后再和返回值比较；

4.不需要任何转换的情况下，null 和undefined 是相等的；

5.一个操作数是NaN，则==返回false，!=返回true；并且NaN 和自身不等；

6.两个操作数都是对象，则比较他们是否是同一个对象，如果都指向同一个对象，则返

回true，否则返回false。

7.在全等和全不等的判断上，比如值和类型都相等，才返回true，否则返回false。

var box = 2 == 2; //true

五．逻辑运算符

逻辑运算符通常用于布尔值的操作，一般和关系运算符配合使用，有三个逻辑运算符：

逻辑与(AND)、逻辑或(OR)、逻辑非(NOT)。

1.逻辑与(AND) ：&&

var box = (5 > 4) && (4 > 3) //true，两边都为true，返回true

如果两边的操作数有一个操作数不是布尔值的情况下，与运算就不一定返回布尔值，此

时，遵循已下规则：

1.第一个操作数是对象，则返回第二个操作数；

2.第二个操作数是对象，则第一个操作数返回true，才返回第二个操作数，否则返回false;

3.有一个操作数是null，则返回null；

4.有一个操作数是undefined，则返回undefined。

var box = 对象&& (5 > 4); //true，返回第二个操作数

var box = (5 > 4) && 对象; //[object Object]

var box = (3 > 4) && 对象; //false

var box = (5 > 4) && null; //null

逻辑与运算符属于短路操作，顾名思义，如果第一个操作数返回是false，第二个数不

管是true 还是false 都返回的false。

var box = true && age; //出错，age 未定义

var box = false && age; //false，不执行age 了

2.逻辑或(OR)：||

var box = (9 > 7) || (7 > 8); //true，两边只要有一边是true，返回true

如果两边的操作数有一个操作数不是布尔值的情况下，与运算就不一定返回布尔值，此

时，遵循已下规则：

1.第一个操作数是对象，则返回第二个操作数；

2.第二个操作数是对象，则第一个操作数返回true，才返回第二个操作数，否则返回false;

3.有一个操作数是null，则返回null；

4.有一个操作数是undefined，则返回undefined。

var box = 对象&& (5 > 4); //true，返回第二个操作数

var box = (5 > 4) && 对象; //[object Object]

var box = (3 > 4) && 对象; //false

var box = (5 > 4) && null; //null

逻辑与运算符属于短路操作，顾名思义，如果第一个操作数返回是false，第二个数不

管是true 还是false 都返回的false。

var box = true && age; //出错，age 未定义

var box = false && age; //false，不执行age 了

2.逻辑或(OR)：||

var box = (9 > 7) || (7 > 8); //true，两边只要有一边是true，返回true

逗号运算符

逗号运算符可以在一条语句中执行多个操作。

var box = 100, age = 20, height = 178; //多个变量声明

1. **流程控制语句.**
2. 语句的定义：

在ECMAScript 中，所有的代码都是由语句来构成的。语句表明执行过程中的流程、限

定与约定，形式上可以是单行语句，或者由一对大括号“｛｝”括起来的复合语句，在语法描

述中，复合语句整体可以作为一个单行语句处理。

二．if语句

if 语句即条件判断语句，一共有三种格式：

1. if (条件表达式) 语句;

var box = 100;

if (box > 50) alert('box 大于50'); //一行的if 语句，判断后执行一条语句

var box = 100;

if (box > 50)

alert('box 大于50'); //两行的if 语句，判断后也执行一条语句

alert('不管怎样，我都能被执行到！');

var box = 100;

if (box < 50) {

alert('box 大于50');

alert('不管怎样，我都能被执行到！');//用复合语句包含，判断后执行一条复合语句

}

对于if 语句括号里的表达式，ECMAScript 会自动调用Boolean()转型函数将这个表达式

的结果转换成一个布尔值。如果值为true，执行后面的一条语句，否则不执行。

PS：if 语句括号里的表达式如果为true，只会执行后面一条语句，如果有多条语句，那

么就必须使用复合语句把多条语句包含在内。

PS2：推荐使用第一种或者第三种格式，一行的if 语句，或者多行的if 复合语句。这样

就不会因为多条语句而造成混乱。

PS3：复合语句我们一般喜欢称作为：代码块。

//如果满足条件，不会执行下面任何分支

alert('甲');

} else if (box >= 90) {

alert('乙');

} else if (box >= 80) {

alert('丙');

} else if (box >= 70) {

alert('丁');

} else if (box >= 60) {

alert('及格');

} else { //如果以上都不满足，则输出不及格

alert('不及格');

}

三．switch语句

switch 语句是多重条件判断，用于多个值相等的比较。

var box = 1;

switch (box) { //用于判断box 相等的多个值

case 1 :

alert('one');

break; //break;用于防止语句的穿透

case 2 :

alert('two');

break;

case 3 :

alert('three');

break;

default : //相当于if 语句里的else，否则的意思

alert('error');

}

四．do...while语句

do...while 语句是一种先运行，后判断的循环语句。也就是说，不管条件是否满足，至

少先运行一次循环体。

var box = 1; //如果是1，执行五次，如果是10，执行1 次

do {

alert(box);

box++;

} while (box <= 5); //先运行一次，再判断

五．while语句

while 语句是一种先判断，后运行的循环语句。也就是说，必须满足条件了之后，方可

运行循环体。

var box = 1; //如果是1，执行五次，如果是10，不执行

while (box <= 5) { //先判断，再执行

alert(box);

box++;

}

六．for语句

for 语句也是一种先判断，后运行的循环语句。但它具有在执行循环之前初始变量和定

义循环后要执行代码的能力。

for (var box = 1; box <= 5 ; box++) { //第一步，声明变量var box = 1;

alert(box); //第二步，判断box <=5

} //第三步，alert(box)

//第四步，box++

//第五步，从第二步再来，直到判断为false

七．for...in语句

for...in 语句是一种精准的迭代语句，可以用来枚举对象的属性。

var box = { //创建一个对象

'name' : '李炎恢', //键值对，左边是属性名，右边是值

'age' : 28,

'height' : 178

};

for (var p in box) { //列举出对象的所有属性

alert(p);

}

八．break和continue语句

break 和continue 语句用于在循环中精确地控制代码的执行。其中，break 语句会立即退

出循环，强制继续执行循环体后面的语句。而continue 语句退出当前循环，继续后面的循环。

for (var box = 1; box <= 10; box++) {

if (box == 5) break; //如果box 是5，就退出循环

document.write(box);

document.write('<br />');

}

for (var box = 1; box <= 10; box++) {

if (box == 5) continue; //如果box 是5，就退出当前循环

document.write(box);

document.write('<br />');

}

九．with语句

with 语句的作用是将代码的作用域设置到一个特定的对象中。

var box = { //创建一个对象

'name' : '李炎恢', //键值对

'age' : 28,

'height' : 178

};

var n = box.name; //从对象里取值赋给变量

var a = box.age;

var h = box.height;

可以将上面的三段赋值操作改写成：

with (box) { //省略了box 对象名

var n = name;

var a = age;

var h = height;

}

1. **函数.**

函数是定义一次但却可以调用或执行任意多次的一段JS 代码。函数有时会有参数，即

函数被调用时指定了值的局部变量。函数常常使用这些参数来计算一个返回值，这个值也成

为函数调用表达式的值。

一．函数声明

函数对任何语言来说都是一个核心的概念。通过函数可以封装任意多条语句，而且可以

在任何地方、任何时候调用执行。ECMAScript 中的函数使用function 关键字来声明，后跟

一组参数以及函数体。

function box() { //没有参数的函数

alert('只有函数被调用，我才会被之执行');

}

box(); //直接调用函数

function box(name, age) { //带参数的函数

alert('你的姓名：'+name+'，年龄：'+age);

}

box('李炎恢',28); //调用函数，并传参

二．return返回值

带参和不带参的函数，都没有定义返回值，而是调用后直接执行的。实际上，任何函数

都可以通过return 语句跟后面的要返回的值来实现返回值。

function box() { //没有参数的函数

return '我被返回了！'; //通过return 把函数的最终值返回

}

alert(box()); //调用函数会得到返回值，然后外面输出

function box(name, age) { //有参数的函数

return '你的姓名：'+name+'，年龄：'+age;//通过return 把函数的最终值返回

}

alert(box('李炎恢', 28)); //调用函数得到返回值，然后外面输出

我们还可以把函数的返回值赋给一个变量，然后通过变量进行操作。

function box(num1, num2) {

return num1 \* num2;

}

var num = box(10, 5); //函数得到的返回值赋给变量

alert(num);

return 语句还有一个功能就是退出当前函数，注意和break 的区别。PS：break 用在循环

和switch 分支语句里。

function box(num) {

if (num < 5) return num; //满足条件，就返回num

return 100; //返回之后，就不执行下面的语句了

}

alert(box(10));

三．arguments对象

ECMAScript 函数不介意传递进来多少参数，也不会因为参数不统一而错误。实际上，

函数体内可以通过arguments 对象来接收传递进来的参数。

function box() {

return arguments[0]+' | '+arguments[1]; //得到每次参数的值

}

alert(box(1,2,3,4,5,6)); //传递参数

arguments 对象的length 属性可以得到参数的数量。

function box() {

return arguments.length; //得到6

}

alert(box(1,2,3,4,5,6));

我们可以利用length 这个属性，来智能的判断有多少参数，然后把参数进行合理的应用。

比如，要实现一个加法运算，将所有传进来的数字累加，而数字的个数又不确定。

function box() {

var sum = 0;

if (arguments.length == 0) return sum; //如果没有参数，退出

for(var i = 0;i < arguments.length; i++) { //如果有，就累加

sum = sum + arguments[i];

}

return sum; //返回累加结果

}

alert(box(5,9,12));

ECMAScript 中的函数，没有像其他高级语言那种函数重载功能。

function box(num) {

return num + 100;

}

function box (num) { //会执行这个函数

return num + 200;

}

alert(box(50)); //返回结果