**函数**（es5中的语法（在各浏览器中直接使用））

（快捷键：Ctrl+点击函数名字）

函数：就是封装起来，用来完成特定功能的代码块。

优点：

1. 使代码更加简洁；
2. 维护起来更加容易（方便维护）。

**声明函数的几种方式：**

1. function关键字声明：function 函数名（）{函数体（执行的内容）}（同一块会被优先解析（同声明变量））

如：function fn=（）{alert（1）；}

fn（）；

同： fn（）；

function fn=（）{alert（1）；}

1. 字面量的形式/匿名函数：var aa=function（）{}

如：function name（）{}

Name（function（）{}）

1. 实例化构造函数：var fn=new Function（）；

例：var fn=new Function(“alert(1)”);

**调用函数：**

1. 函数名/变量名（）；如：fn（）；
2. 在事件后面调用 div.onclick=function(){}
3. 函数的自调用（function{alert（1）}）（）；（声明好直接调用（直接弹出1））也可以保护全局变量不受污染。

**参数：（参数和参数之间用逗号隔开）**

1. 形参
2. 实参
3. 如：function fn（形参）{alert（1）；}

fn（实参）；

1. 如function fn（num）{alert（num）；}

fn（num）；

**参数特点：**

1.个数和数据类型（个数很多但有限25，数据类型都可以）；

2.形参和实参是一一对应的；

3.形参和实参的个数可以不一致，形参个数大于实参个数，多出来的形参会被赋值为undefined；

如：想要形参为undefined时输出1的三种形式：

（1）function *fn*(num) {  
 if(num===undefined){  
 num=1;  
 }  
 alert(num);  
}  
*fn*();

（2）function *fn*(num) {  
 num=num||1;  
 alert(num);  
}  
*fn*();

（3function *fn*(num) {  
 num=num===undefined?1:num;  
 alert(num);  
}  
*fn*();

）

4.实参的个数大于形参，并不会有影响，可以通过argument对象来访问；

如：function fn(a，b){

console.log(a);

console.log(b);

console.log(argument[编号（0-5）])

onsole.log(argument.Length);（获取实参的个数）

alert(argument.callee);（访问到函数的自身（调用自己））

}

fn(1,2,3,4);

**返回值（return）：**

1. 返回值是可有可无的；
2. 返回值只能有一个；
3. 执行了return，后面的代码不能继续执行。

function fn（）{return 2;}

Alert（fn（））；

如：

1. 乘法：

function *sqrt*(num1,num2){  
pf=num1\*num2;  
return pf;  
}  
var *result1*=*sqrt*(4,6);  
alert(*result1*);

同：

function *sqrt*(num1,num2){  
 return num1\*num2;  
}  
alert(*sqrt*(4,6));

2. *个数不确定求和* function *jia*(){  
 var sum=0;  
 for(var i=0;i<arguments.length;i++){  
 sum+=arguments[i];  
 }  
 return sum;  
 }  
alert(*jia*(1,2,3,4));

**函数重载**：根据函数参数的个数和数据类型的不同执行不同的函数体。

模拟函数重载：利用arguments对象来执行

如：

*// 模拟函数重载（1）* function *fn*(val) {  
 if(typeof val=="number"){  
 return "这是一个数值类型"  
 }else if(typeof val=="string"){  
 return "这是一个字符串"  
 }  
 }  
 console.log(*fn*("asd"));  
*// 模拟函数重载（2）*var *sum*=0;  
function *jia*(){  
 if(arguments.length==1){  
 for(var i=0;i<=arguments[0];i++){  
 *sum*+=i;  
 }  
 }else{  
 for(var i=0;i<arguments.length;i++){  
 *sum*+=arguments[i];  
 }  
}  
 return *sum*;  
}  
alert(*jia*(100,100))

表示只有1个数时，进行累计加和，

**作用域：**一段代码起作用的范围

**环境**

1. **宿主环境**（浏览器）
2. **执行环境**
3. **全局环境**（script内的环境）
4. **函数环境**（函数内的环境）

**全局变量：**在全局环境中声明的变量；（作用域：全局环境；即在全局环境中定义，也可以在函数环境中访问到。）

**局部变量：**在函数环境中声明的变量。（作用域：函数环境；即在函数环境中定义只能在函数环境中可以访问到，但在全局环境中不可以访问。）

**变量的作用域：**定义变量的环境当中的任意位置。

**注：**

1. 当全局变量和局部变量重复，优先显示局部变量。
2. 形参也是局部变量（不仅可以通过实参来传递）。
3. 暂时性死区：

如：

var *val*=5;  
function *fn*() {  
 alert(val);(暂时性死区)  
 var val=10;（能解析到）  
}

表示可以访问到val，但是结果为undefined。

如：

var *val*=5;  
function *fn*(val) {  
 var val=10;

alert(val);  
}

fn（val）；

结果为10.

如：

var *val*=5;  
function *fn*(val) {  
 val++;

alert(val);  
}

fn（val）；

alert(val);

结果是先输出6，在输出5。

如：

var *val*=5;  
function *fn*(val) {  
 val++;

alert(val);  
}

fn（val++）；

alert(val);

结果都为6。

4.在函数中不适用var声明变量可以在函数外面访问到。

如：  
function *fn*() {  
 aa=5;  
}

fn（）；

alert(aa);

结果为5.

全局函数可以在全局环境中的任意位置显示。

**作用域链：**当多个作用域嵌套的时候，在内层作用域访问变量的时候会依次从最内层的作用域开始找起，直到找到全局作用域。

如：var *aa*=5;  
 function *fn1*() {  
 var aa=6;  
 function *fn2*() {  
 var aa=7;  
 function *fn3*() {  
 var aa=8;  
 alert(aa);  
 }  
 *fn3*();  
 }  
 *fn2*();  
 }  
*fn1*();

**回调函数：**（unction () { alert(1)）

当某一个函数作为另外一个函数的参数传递并且在函数内部被调用的函数为回调函数。

如：

function *fn1*(fun ){  
 fun();  
}  
*fn1*(function () {  
 alert(1);  
});

回调函数的形参callback

如：

计算器1

function *jsq*(num1,num2,ysf){  
 ysf=ysf||"+";  
 if(ysf=="/"&&num2==0){  
 return;  
 }  
 var r;  
 switch (ysf){  
 case "+":r=num1+num2;  
 break;  
 case "-":r=num1-num2;  
 break;  
 case "\*":r=num1\*num2;  
 break;  
 case "/":r=num1/num2;  
 break;  
 default:break;  
 }  
 return r;  
}  
console.log(*jsq*(111,10,"-"));

计算器2（回调函数）

function *jsq*(num1,num2,callback){  
 return callback(num1,num2);  
}  
var *r*=*jsq*(2,2,function (a,b) {  
 return a\*2+b\*4;  
});  
console.log(*r*);

**递归函数**：在函数的内部调用自身

如：function *jc*(num){  
 if(num==1){  
 return 1;  
 }else{  
 return num\**jc*(num-1)  
 }  
}  
console.log(*jc*(5));

闭包函数：局部变量在函数运行结束之后依然在内存中保存不被销毁掉，这种形式成为闭包。

缺点：占据内存

如：

function *fn*(){  
 var num=10;  
 function *inner*() {  
 num++;  
 console.log(num);  
 }  
 return inner;  
}  
var *out*=*fn*();  
*out*();  
*out*();  
*out*();

Es6中新增语法：

1.可在形参中直接赋值为默认值。

如：

function *demo1*(num1=1,num2=3) {  
 console.log(num1,num2);  
}  
*demo1*();*//1,3*function *demo1*(num1=1,num2=3) {  
 console.log(num1,num2);  
}  
*demo1*(10,10);*//10,10*function *demo1*(num1=1,num2=3) {  
 console.log(num1,num2);  
}  
*demo1*(10);*//10,3*function *demo1*(num1=1,num2=3) {  
 console.log(num1,num2);  
}  
*demo1*(undefined,10);*//1,10*function *demo1*(num1=1,num2=3) {  
 console.log(num1,num2);  
}  
*demo1*(,10);*//报错*

*//表格制作：* function *table*(rows=10,cols=10) {  
 document.write("<table>");  
 for(var i=0;i<rows;i++){  
 document.write("<tr>");  
 for(var j=0;j<cols;j++){  
 document.write("<td>11</td>");  
 }  
 document.write("</tr>");  
 }  
 document.write("</table>");  
 }  
 *table*();

**2.块级作用域：**（对var无影响，对let和const有影响）

Let不会将前面的覆盖掉。

1. 如何访问函数名。（.name）name属性

如：

var *fun*=function aa() {  
};  
alert(*fun*.name);

1. 箭头函数

如：

var *jia2*=(num1,num2)=>num1+num2;  
var *he*=*jia2*(11,11)  
alert(*he*);

等同于

var *jia*=function(num1,num2){  
 return num1+num2;  
};  
var *he1*=*jia*(11,11);  
console.log(*he1*);

如：2：

var *even*=num=>num%2==0;  
alert(*even*(22));

等同于：

function *even*(num) {  
 if(num%2==0){  
 alert(true);  
 }else{  
 alert(false);  
 }  
}  
*even*(23);

如：3：

var *jia2*=(num1,num2)=>{  
 console.log(num1,num2);  
 return num1+num2;  
}  
 console.log(*jia2*(1,1));  
 </script>

**内置顶层函数：**

1. Escape（）编码 unescape（）解码
2. Number（）

功能:将其他类型转化为数值类型

如：var *str*="asd";  
console.log( Number(*str*));*//NaN*var *str1*=("12");  
console.log( Number(*str1*));*//12*var *str2*=("0012");  
console.log( Number(*str2*));*//12*var *str3*=("12.100000");  
console.log( Number(*str3*));*//12.1*var *str4*=("");  
console.log( Number(*str4*));*//0*var *str5*=("12asfx");  
console.log( Number(*str5*));*//NaN*var *str6*=("0x11");（*0x十六进制；0o/00八进制；0b二进制*）  
console.log( Number(*str6*));*//转换为十进制*

ar *boo1*=true;(false)  
console.log( Number(*boo1*));*//1 0*var *un*=undefined;  
console.log( Number(*un*));*// NaN*var *obj*=null;  
console.log( Number(*obj*));*//0*

1. parseInt（）

特点：

1. 将字符串转化为整数类型(从前向后)

如：

var *str*=("122");  
console.log( parseInt(*str*));*//122*var *str1*=("adc");  
console.log( parseInt(*str1*));*//NaN*var *str2*=("12.9");  
console.log( parseInt(*str2*));*//12*var *str3*=("123asd");  
console.log( parseInt(*str3*));*//123*var *str4*=("abc123");  
console.log( parseInt(*str4*));*//NaN*

var *str5*=("");  
console.log( parseInt(*str5*)); *//NaN*

1. 传递第二个参数 后面的值取2-32 以几进制的方式转换为十进制数

如： var *str6*="11";  
console.log( parseInt(*str6*,2))（取2-32的值）

结果为3.

1. parseFloat

将字符串转换为小数（本身就是整数不会转换为小数）

如：

var *str6*="1.11";  
console.log( parseFloat(*str6*));

1. String(黑色的)

将其他类型转换为字符串

如：

var *num*=111;  
console.log( String(*num*));  
var *num*;  
console.log( String(*num*));*//undefined*var *boo*=true;  
console.log( String(*boo*));*//true*var *obj*=null;  
console.log( String(*obj*));*//null*

1. Boolean（true/false）

其中“0，undefined，NaN，false，null”结果是false

如：

console.log( String(undefined);//false

1. isNaN：判断一个数据是不是一个不能被转换为数值的数

true：是一个不能被转换为数值的数

false：是一个能被转换为数值的数

如：

console.log( isNaN("123"));*//false*console.log( isNaN("asd"));*//true*console.log( isNaN(undefined));*//true*console.log( isNaN(true));*//false*console.log( isNaN(12345));*//false*

**强制数据类型转化**

**隐藏数据类型转化**