# IMG_25611.07IMG_256

## IMG_256JS组成

ECMAScript 语法

DOM 文档对象模型

BOM 浏览器对象模型

IMG_256

## JS书写方式

### 行内

<button onclick="js代码">按钮</button>

### 内部

<script>js代码</script>

### 外联

<script src="js文件路径"></script>

## IMG_256输出内容的方式

alert(数据); 警告框

console.log(数据); 控制台

document.write(数据); 网页中

prompt(提示内容); 提示框

IMG_256

## 数据类型

### 简单的数据类型

①Number 数字

八进制 : 前缀：0开头，使用时 会自动转换为十进制

十六进制 : 前缀：0x开头，序列范围： 0-9及a-f ，使用时 会自动转换为十进制

NaN : 是数字类型，但是不是正常的数值 isNaN(数据或变量); 返回布尔值

最大值 : Number.MAX\_VALUE

最小值 : Number.MIN\_VALUE

浮点数（小数）: 小数的精确度：17位 计算时会有误差。避免小数运算

当一个正数字和0相除时，得的结果是Infinity

当一个负数字和0相除时，得的结果是-Infinity

② String 字符串

alert('我很喜欢\"黑马\'程序猿\'');

alert("程序'媛'");

对于+号：  
 ① 数学中的相加。两个数字相加。

② 连接符（拼接符），两个字符串相加。

③ Boolean 布尔

工具：检测一个数据是否是NaN

isNaN(数据或变量); 返回布尔值true是NaN，false.不是NaN

④ Undefiend

Undefined:未定义

意义：当变量定义没有赋值时，默认值是undefined;

⑤ NULL 空（空对象）

typeof(null)；检测数据类型

检测类型时结果为object.称为NULL

IMG_256

## 运算符

1. 小括号 () 优先级最高

2. 一元运算符 ++ -- ! +（正） -（负）

3. 算术运算符 先\* / % 后 + -

4. 关系运算符 > >= < <= 运算的结果始终是一个布尔值

5. 相等运算符 == 、 != 、 === 、!=

6. 逻辑运算符 优先级：! > && > ||

7. 赋值运算符

IMG_256

## 数据类型转换

### 强制转换

IMG_256其他类型数据转数字：

Number(其他数据);

Number('abc'); NaN | Number('12abc'); NaN | Number(true); 1

Number(undefined); NaN | Number(null); 0

parseInt(其他数据);

parseInt('123.34'); 123 | parseInt('12abc'); 12 | parseInt(true); NaN

parseInt(undefined); NaN | parseInt(null); NaN

parseFloat(其他数据);

parseFloat('3.14'); 3.14 其他都和parseInt相同 遇到字符串就会停止

IMG_256其他类型数据转字符串：

所有的数据在转换成字符串时，其实就是在其他数据两边加上双引号或单引号。

变量名.toString();

在使用变量.toString()方法时，变量的值不能为undefined和null，否则会报错

String(其他数据);

IMG_256

其他数据类型转布尔：

Boolean(其他数据)

null、undefined、0、NaN、'' 转换为boolean结果为false,其他的数据转换的结果为true。

### 隐式转换

❀ 转数字 Number

在做算术运算时（除了和字符串做相加运算外），其他非数字类型的数据会自动的转换为Number类型的数据。

特殊：任何数据和NaN做算术运算时，结果始终都是NaN

❀ 转字符串 String

和字符串做相加运算 true + '我'; 'true我'

❀ 转布尔 Boolean

取反、分支结构、循环结构

console.log(!1); //false ;

### 检测数据类型

typeof(数据);    适用于基本类型以及function,遇到null失效

typeof(null);    //object

可以用严格等于判断null===null

typeof(undefined);    //undefined

typeof(1);    //number.

typeof(true);    //boolean

typeof("ccc");    //string

typeof(Function);    //function

### 补充

Num -0    转换为数字

String +""    转换为字符串

！"str/num"    转换为布尔

NaN==NaN；            // false

null == undefined       // true

IMG_256

## 三元表达式

条件表达式 ? 结果1:结果2; 当条件成立时，执行结果1，否则执行结果2。

age>=18?'成年':'未成年';

IMG_256

## switch结构

· switch结构仅仅只能做等值判断。

· 当对于多种情况，全都是等值判断时，可以选择使用switch，因为siwtch对于等值判断结构更加清晰，执行速度要比多分支结构要快一些。

· //细节1：switch在和情况比较相等时，用的是全等比较（必须数据类型也一致）

//细节2：case具有穿透性

//细节3：可以用break打破穿透性的规则并且结束switch语句

switch(传入的值){

case 值1:

//执行语句

break;

case 值2:

//执行语句

break;

default:

//执行语句

break;

}

IMG_256

## do-while循环

do {

//循环体

}while(条件表达式);

先执行一次循环体，然后是否周而复始的执行循环体，根据条件表达式是否成立决定。

IMG_256

## break 和 continue

### IMG_256 break

可以结束当前整个循环

配合while(true){ }死循环 用break给定结束条件

### IMG_256 continue

结束当前这一次的循环，并开始下一次循环

//打印 100以内所有的偶数

for(var num = 1;num<=100;num++){

if(num%2!=0){ //将不符合条件的过滤掉

continue;

}

console.log(num);

}

IMG_256

## 数组

创建数组 var 变量名 = []; //空的数组

新增数据 数组名[数组名.length] = 数据;

IMG_256

## 函数

### 创建方式

❀ 方式一 函数声明

function 函数名(){

}

❀ 方式二 函数表达式

var 函数名 = function(){

}；

### 参数

❀ 形参

function 函数名(形参,形参,形参...) {

}

❀ 实参

函数名(数据,数据,数据...); //实参，就是实际的数据

若没有写实参，则默认为undefined

### return

若函数体内没有return 则默认返回undefined

return可以结束函数

### arguments

当不确定数组参数个数时使用

function getSum(){

var sum = 0;

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

sum = sum + arguments[index];

}

return sum;

}

var result = getSum(10,20,30,40,50); //

alert(result);

### 预解析

把用var关键字所声明的变量名（仅仅是变量名），提升到当前执行环境的顶部。

值不提前，值为undefined

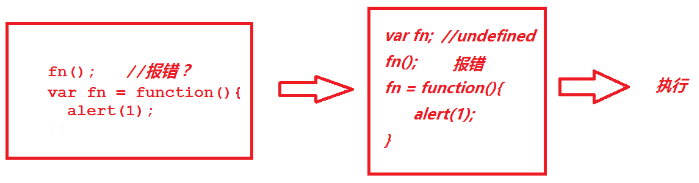
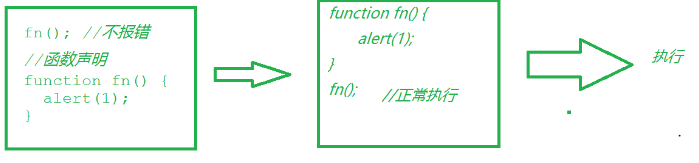
把用函数声明所创建的函数（函数整体），提升到当前执行环境的顶部。

Function (a,b) { Function() {

相当于 Var a;

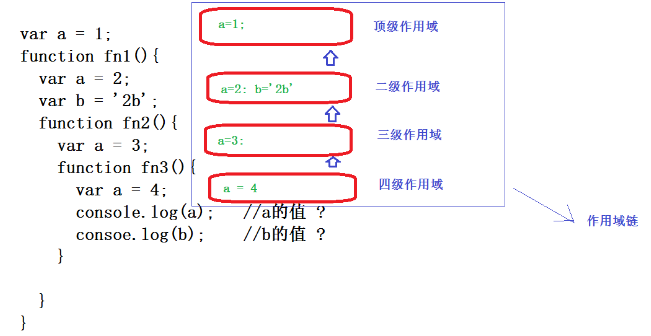
Var b;

} }



### 作用域链

当访问一个变量时，会先从本作用域中去找这个变量中去找，若找不到则向上一级作用域中去找，依次类推，就形成了一个作用域链。



### 回调函数

❀ 方式一：

function fn(f){

f();

}

var test = function(){

alert('执行');

}

fn(test);

❀ 方式二：

function fn(f){

f();

}

fn(function(){

alert('执行');

});

### 匿名函数和自执行函数

(function(){

alert(1);

})();

或者加运算符：

- function(){

alert(1);

}();

❀ 匿名函数和自定义函数的优点

可以避免全局变量污染​

可以模拟块级作用域

模拟块级作用域：

if(true){

var num = 123;

console.log(123); //123

}

console.log(123); //123;

IMG_256

## 对象

### IMG_256 创建对象

语法： var 对象名 = new Object();

### IMG_256 增加与调用

❀ 增加属性：

wuKong . name = '牛小小';  
wuKong . age = 13;  
wuKong ['gen' + 'der'] = '美少女';  
wuKong ['gender'] = '美少女';

❀ 调用属性：

wuKong.age  
wuKong['age']

❀ 增加方法：

wuKong.attack = function(yg){  
 alert(wuKong.name + '向'+yg+'发动攻击...');  
 console.log(this.name + '向'+yg+'发动攻击...');  
} ;

❀ 调用方法：  
wuKong . attack (‘光头强’);  
wuKong ['attack'] ('牛魔王');

### IMG_256 针对Object类创建对象的简写方式 （创建单个对象时）

var wuKong = {  
 name:'孙悟空', // 使用键值对表示 用逗号分隔  
 age:500,  
 attack:function(yg){ // 增加方法  
 console.log(this.name + '向' + yg + '发动攻击...');  
 }  
} ;  
wuKong.attack('白骨精'); // 调用方法不变

相当于把new Object() 替换成 { }

### IMG_256 删除属性和方法

delete 对象.键名;

delete 对象['键名'];

补充：

检测对中是否还有name属性

dog1.hasOwnProperty('name') 返回false表示不存在 true表示存在

### IMG_256 工厂模式

function createHero(n,a){ //将对象要赋的值作为形参  
 var myThis = new Object(); //创建了一个空的对象  
 myThis.name = n;  
 myThis.age = a;  
 myThis.attack = function(yg){  
 console.log(this.name + '向' + yg + '发动攻击...');  
 } ;  
 return myThis;  
}  
  
var wuKong = createHero('孙悟空',500) ;

wuKong.attack('牛牛'); //调用方法不变

### IMG_256 构造函数 （创建多个对象时）

创建构造函数（类）：  
 function 构造函数名( ) { //命名时首字母要大写  
 this.键 = 值;  
 this.键 = 值;  
 this.键 = 值;

....  
 }  
使用构造函数：  
var 对象名 = new 构造函数名();

❀ new关键字调用构造函数时做了什么事？  
 ① 在构造函数中创建了一个空的Object类型的对象（看不见）  
 ② 让this关键字去引用所创建的空的对象（看不见）  
 ③ 通过this给这个空的对象添加属性和方法（看的见）  
 ④ 把已经添加好属性和方法的对象返回（看不见）

❀ 补充：

检测一个对象的数据类型

对象 instanceof 构造函数名; //true表示属于，false表示不属于

p1 instanceof Person // 检测对象p1是否属于Person

### IMG_256 遍历键值对for key in

for(var key in 对象){

//key 是对象中的每一个键

}

调用： 对象[key];

————————————

例：

var obj = {  
 name:'孙悟空',  
 age:100,  
 gender:'男'  
};

for(var key in obj){  
 document.write(key + '-----' + obj[key]);  
}

输出内容：

name-----孙悟空

age-----100

gender-----男

—————————————

### IMG_256 this 指向问题

例子：

function Person(){

this.name = '张三';

console.log(this);

}

❀ this指向被创建的实例对象

//Person当做构造函数调用

var p1 = new Person();

❀ this指向 window对象

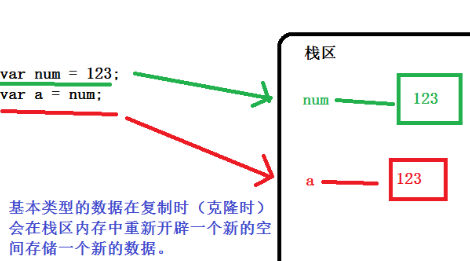
// Person当做普通函数调用

Person();

### IMG_256 栈和堆

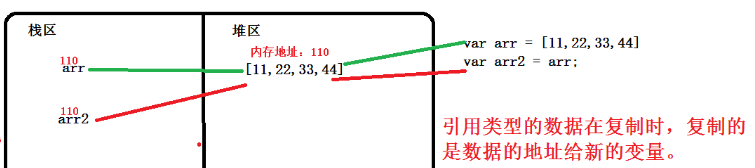
❀ 栈区：用来存储用var关键字创建的变量名 和 基本类型的数据。

number 数字、string 字符串、boolean布尔、undefined 未定义、 null 空



❀ 堆区：用来存储引用类型的数据。

数组、函数、对象



## 内置对象

### IMG_256 Math对象

Math.abs(x);

功能：取绝对值 参数：x 数值 返回值：数字

**var *r*** = **Math**.abs(-100);

Math.random();

功能：随机数 参数：无 返回值：数字 [0,1) 包含0不包含1

**var *r*** = **Math**.random();

求最大值

**Math**.max(11,22,10,33,55,44,66,35)

四舍五入

Math.round(0.60) // 1

### IMG_256Date对象

var obj = new Date(); 代表当前的时间

var obj = new Date(value);

value代表自1970年1月1日00:00:00 (世界标准时间) 起经过的毫秒数。

var obj =new Date(dateString);

其中dateString： 'year-month-date hh:mm:ss'

'year/month/date hh:mm:ss'

❀ 方法：

getFullYear年 getMonth月getDate日

getHours小时getMinutes分钟getSeconds秒getMilliseconds毫秒

❀ 注：

① date.getMonth( ) +1 获取月份时要加一 month：0（1）-11（12）；

② 1：1→01：01 month = month<10?'0' + month:month;

方法：

[getTime( )](mk:@MSITStore:E:\\前端就业班-11月\\11.13第五天\\day05-JS基础\\01-学习资料\\W3School离线手册(2017.03.11版).chm::/www.w3school.com.cn/jsref/jsref_gettime.asp.htm) 返回 1970 年 1 月 1 日至今的毫秒数。

[getFullYear( )](mk:@MSITStore:E:\\前端就业班-11月\\11.13第五天\\day05-JS基础\\01-学习资料\\W3School离线手册(2017.03.11版).chm::/www.w3school.com.cn/jsref/jsref_getfullyear.asp.htm) 从 Date 对象以四位数字返回年份。

[setFullYear( )](mk:@MSITStore:E:\\前端就业班-11月\\11.13第五天\\day05-JS基础\\01-学习资料\\W3School离线手册(2017.03.11版).chm::/www.w3school.com.cn/jsref/jsref_setfullyear.asp.htm) 设置 Date 对象中的年份（四位数字）。

IMG_256

### 数组

创建数组：

var arr = new Array(数据,数据,数据);

❀ 方法：

arr.valueOf( ) 获取对象的原始值

arr.toString( ) 字符串的形式打印数组 获得的只是一串字符串

arr.push('旺财','小强'); //向最后面添加数据

arr.unshift('光头强','灰太狼'); //向最前面添加数据

var r1 = arr.pop(); //删除最后一个数据

var r2 = arr.shift(); //删除第一个数据

❀ 终极删除大法：

数组对象.splice ( startIndex , count , 数据,数据,数据...) ;

参数：  
 startIndex表示从数组的哪个位置开始删除（下标，从0开始）number必写  
 count 表示删除几个 number 必写

数据,数据,数据... 要添加的数据 可选

例：arr.splice(1,0,"阿毛", "阿花",'阿油'); //增加三个数据

1 表示在第二个位置添加，把原来第二个数据挤到后面

arr.splice ( 2,1,'DengKun.Tom') ; //修改或替换

❀ 查询数据所在位置

indexOf(数据);

找到：返回对应的下标，找不到返回 -1

arr.lastIndexOf('数据') 当有多个相同数据时，返回最后面的一个

补充：

var f = confirm('提示信息'); //返回布尔值 true 或 false

❀ 数组倒置

userNames.reverse( ) ;

❀ 数组排序

从小到大：

数组名.sort(function(a,b){ // a，b代表相邻的两个数据  
 return a-b;   
 });  
从大到小：  
 数组名.sort(function(a,b){  
 return b-a; +++

});

❀ 针对对象排序

**var *students*** = [  
 {**name**:**'张三'**,**age**:11,**gender**:**'男'**},  
 {**name**:**'李四'**,**age**:12,**gender**:**'男'**},  
 {**name**:**'王五'**,**age**:10,**gender**:**'男'**},  
 {**name**:**'赵六'**,**age**:9,**gender**:**'男'**}  
];

***students***.sort(**function**(a,b){  
**return** a.**age** - b.**age**;  
});  
**console**.log(***students***);

❀ 合并数组

**var *arr3*** = ***arr1***.concat(***arr2***);

不会改变原有的数组，而是返回一个新的数组

❀ 显示数组中某段数据

**var *newArr*** = ***userNames***.slice(4,7); // 返回[4,7) 第567个数据

❀ 数据之间插入符号

**var *str*** = ***userNames***.join(**'❤'**);

❀ 符合条件的抽取出来

fileter

**var *newArr*** = ***arr***.filter(**function**(v){  
 **return** v<2000;  
}); //返回值为一个数组

❀ 清空数组

***arr***.**length** = 0;

***或 arr***=[];

## 字符串包装对象

var len = str.length;

Str不是对象，但是为什么可以像对象一样使用点出东西？

答：str在点的时候，在内存中会创建一个字符串包装对象并把引用赋值给str,等str调用完length之后，这个在内存中创建的临时对象会断开与str之间的引用并从内存中删除。str重新指向普通字符串'abc'。

把字符串包装成对象：

var strObj = new String('abc'); //把字符串包装成对象 可以不写

var len = strObj.length; //因为是对象，所以可以点出东西。

## 字符串的不可变性

var str = 'abc';

str = 'hello';

当重新给str赋值的时候，常量'abc'不会被修改，依然在内存中

重新给字符串赋值，会重新在内存中开辟空间，这个特点就是字符串的不可变

由于字符串的不可变，在大量拼接字符串的时候会有效率问题

## 字符串常用的方法

### IMG_256 Slice 截取

slice(start,end) // [start,end)

从start位置开始，截取到end位置，end取不到

### IMG_256 indexOf 获取位置

Var str=’星光闪耀’;

indexOf(‘闪耀’); //返回2 即开始的位置

IndexOf(‘闪耀妖妖’) //返回-1 不存在

lastIndexOf(‘xx’) //返回最后一个xx

### IMG_256 Trim 去空格

返回去掉两边空格之后的字符串

Var Str=’ 美 少女 ’；

Var newstr=str.trim(); // 返回美 少女

### toUpperCase() toLowerCase()

转换成大写、小写

Var code=prompt(【acBe】)；

Code.toUpperCase()==’acbe’; //用于验证码验证

### Replace 替换

Replace(old,new) 不会改变字符串本身，而是返回替换后的新字符串

Var newstr=str.replace(牛小小，小仙女); // 只能替换出现的第一个

全部替换：

### Split

分割字符 返回数组 与join功能相反

Var index=str.split(‘❤’)； // ❤之间的数据存在新的数组中