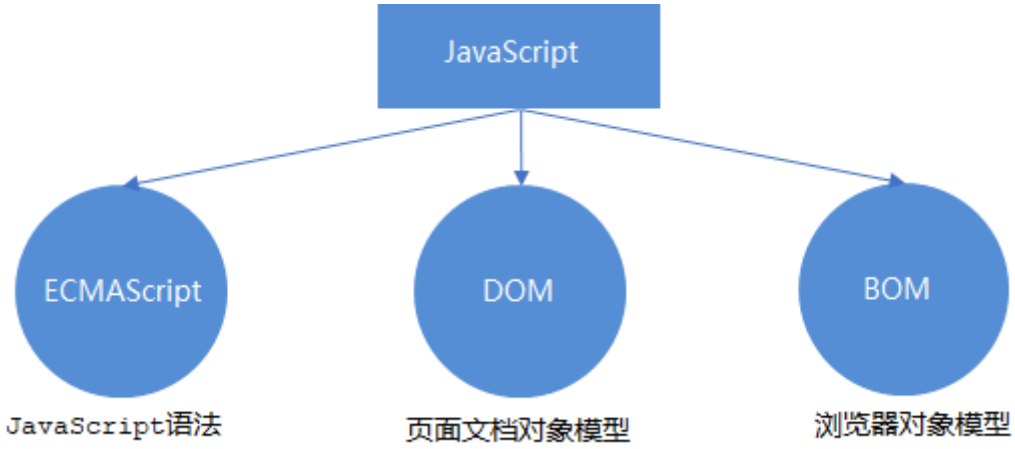
js的ECMAScript语法基础

js的三种写入方式




```
0.
 <!-- 1. 行内式的js 直接写到元素的内部 -->
 <input type="button" value="xxx" onclick="alert('hahaha')>";
1.
<!-- 内嵌式的js -->
    <script>
        // alert('!!!!!!1');
    </script>
2.
    <!-- 外部js script 双标签 -->
    <script src="my.js"></script>
```

js的输出语句

输入框：**prompt("xxxxxxx");**

弹出警示框：**alert("xxxxxx");**

在浏览器控制台输出：**console.log("xxxxx")**

js中的变量

```
// 1. 声明了一个age 的变量
        var age;
//使用let 声明变量
        let ages;
        // 2. 赋值  把值存入这个变量中
```

```
        age = 18;
        // 3. 输出结果
        console.log(age);
        // 4. 变量的初始化
        var myname = 'alix';
        console.log(myname);
//。
//在函数中声明了var，整个函数内都是有效的，比如说在for循环内定义的一个var变量，实际上其在for循环以外也是可以访问的。
//而let由于是块作用域，所以如果在块作用域内定义的变量，比如说在for循环内，在其外面是不可被访问的，所以for循环推荐用let。
```

var是函数作用域，let是块作用域
let不能在定义之前访问该变量，但是var可以
let不能被重新定义，但是var是可以的

什么是变量

本质：变量是程序在内存中申请的一块用来存放数据的空间。

只声明不赋值 结果是 undefined 未定义的

不声明 不赋值 直接使用某个变量会报错

不声明直接赋值也可以使用

情况	说明	结果
var age;console.log(a);	只声明，不赋值	undefined
console.log(a);	不声明，不赋值，直接使用	报错
age=10;console.log(a)	不声明 只赋值	100

js的数据类型

js是弱类型的语言 变量的数据类型是可以变化的，跟python一样

不需要手动去声明变量的数据类型

1.Number类型

Number类型包含整数和浮点数（浮点数数值就是小数，且小数点后面至少有一位数字）两种值。

浮点数将会自动转换为整数。

NaN:非数字类型。特点：

① 涉及到的 任何关于NaN的操作，都会返回NaN

② NaN不等于自身。

isNaN()函数：

用于判断是否是一个非数字类型。如果传入的参数是一个**非数字类型**，那么返回true；否则返回false;

isNaN()函数，传入一个参数，函数会先将参数转换为数值（is not a number）

转换为Number类型的函数

Number()转型函数，可以用于任何数据类型；

parseInt()，将值转换为整型，用的比较多；

parseFloat()；将值转换为浮点型。

```
console.log(parseInt('3.14')); // 3 取整
console.log(parseInt('120xxx')); // 120 会去到这个xxx单位，只识别数字
 console.log(parseInt('rem120px')); // NaN


 console.log(parseFloat('120嘻嘻嘻')); // 120 会去掉这个嘻嘻嘻
 console.log(parseFloat('rem120px')); // NaN
```

2.String类型

字符串类型中的 单引号 与 双引号 的效果完全一样。

字符串有length属性。可以取得字符串的长度。

```
var str = “hello”;

console.log(str.length);//5
```

字符串的值是不可变的。要改变一个字符串的值：**首先要销毁原来的字符串，再用另一个包含新值的字符串去填充该字符串。**

```
var lang = “java”;
lang += “script”;
```

先创建一个能容纳10个字符的字符串，然后在这个字符串中填充java和script字符串，最后销毁原来的字符串java和scrip字符串，因为这两个字符串此时已经没用了。过程是在后台发生的。

*字符串的拼接 + 只要有**字符串和其他类型相拼接 最终的结果是字符串类型***

字符串转换

转型函数String(),适用于任何数据类型（null,undefined 转换后为null和undefined）；

```
var c = "txwd";
var d = null;
var e = undefined;
console.log(c.toString());
//console.log(d.toString());//error 报错
//console.log(e.toString());//error 报错
console.log("------");
console.log(String(c));
console.log(String(d));
console.log(String(e));
```

3.Boolean类型

该类型只有两个值，true和false

true 参与运算当1来看

false 参与运算当 0来看

NaN undefined 和数字相加 最后的结果是 NaN

转换为boolean值:

转型函数Boolean(),将某个值转换为Boolean类型

4.Null类型

null类型被看做空对象指针，null类型也是空的对象引用。只有一个值，即null值，所以用typeof 操作符去检测null类型的值时，结果是object类型。

如果定义一个变量，但是想在以后把这个变量当做一个对象来用，那么将该对象初始化为null值。

5.Undefined类型

只有一个值，即undefined值。使用var声明了变量，但未给变量初始化值，那么这个变量的值就是undefined。

6.Object类型

js中对象是一组属性与方法的集合。

运算符

运算符	介绍
++ -- !	自增、自减、去反
* / %	用于Number类型的乘、除、求余
+ -	相加、相减
> >= < <=	用于比较的运算符
== === != !==	比较两者是否相等
& ^ |	按位"与" 、按位"异或"、按位"或"
&& ||	逻辑与、逻辑或 （这个是具有短路效果的）
?.	条件运算
= += -= *= /=	赋值、赋值运算
``	`````

- &&：**从左到右依次判断**，如果遇到一个假值，就返回假值FALSE，以后不再执行，否则返回最后一个真值；
- ||：为取真运算，**从左到右依次判断**，如果遇到一个真值tTRUE，就返回真值，以后不再执行，否则返回最后一个假值；

条件语句

1. if 的语法结构

```
 if（条件表达式）{

执行语句

}
// 语法结构  if else

   if（条件表达式）{
          // 执行语句1
        } else {
        // 执行语句2
     }

  // 执行思路  如果表达式结果为真  那么执行语句1   否则   执行语句2

 if（条件表达式1）{
            // 语句1;
        } else if（条件表达式2）{
            // 语句2;
        } else if（条件表达式3）{
            // 语句3;
        } else {
            // 最后的语句;
        }
```

执行思路 如果 if 里面的条件表达式结果为真 true 则执行大括号里面的 执行语句

** 如果if 条件表达式结果为假 则不执行大括号里面的语句 则执行if 语句后面的代码**

if里面的语句1 和 else 里面的语句2 最终只能有一个语句执行 2选1
else 后面直接跟大括号

三元表达式

```
var num = 10;
var result = num > 5 ? 'yes' : 'nono';
```

如果条件表达式结果为真 则 返回 表达式1 的值 如果条件表达式结果为假 则返回 表达式2 的值

switch分支语句

```
// 语法结构 switch 转换、开关  case 小例子或者选项的意思
      switch (表达式) {
          case value1:
              执行语句1;
              break;
          case value2:
              执行语句2;
              break;
               ...
           default:
              执行最后的语句;
        }
// 执行思路  利用我们的表达式的值 和 case 后面的选项值相匹配 如果匹配上，就执行该case 里面的语句  如果都没有匹配上，那么执行 default里面的语句
```

注意:

num 的值 和 case 里面的值相匹配的时候是全等》 必须是值和数据类型一致才可以 num === 1

break 如果当前的case里面没有break 则不会退出switch 是继续执行下一个case

循环语句

```
for (初始化变量; 条件表达式; 操作表达式) {
            // 循环体
        }
```

打印n行n列的星星数

```
// 打印n行n列的星星
        var rows = prompt('请您输入行数:');
        var cols = prompt('请您输入列数:');
        var str = '';
        for (var i = 1; i <= rows; i++) {
            for (var j = 1; j <= cols; j++) {
                str = str + '★';
            }
            str += '\n';
        }
        console.log(str);
```

打印倒三角形案例

```
// 打印倒三角形案例
        var str = '';
        for (var i = 1; i <= 10; i++) {
        // 外层循环控制行数
            for (var j = i; j <= 10; j++) {
            // 里层循环打印的个数不一样
                j = i
                str = str + '★';
            }
            str += '\n';
        }
        console.log(str);
```

九九乘法表

```
// 九九乘法表
        // 一共有9行，但是每行的个数不一样，因此需要用到双重 for 循环
        // 外层的 for 循环控制行数 i ，循环9次 ，可以打印 9 行
        // 内层的 for 循环控制每行公式  j
        // 核心算法：每一行 公式的个数正好和行数一致， j <= i;
        // 每行打印完毕，都需要重新换一行
        var str = '';
        for (var i = 1; i <= 9; i++) { // 外层循环控制行数
            for (var j = 1; j <= i; j++) { // 里层循环控制每一行的个数  j <= i
                // 1 × 2 = 2
                // str = str + '★';
                str += j + '×' + i + '=' + i * j + '\t';
            }
            str += '\n';
        }
        console.log(str);
```

while循环案列

```
<script>
        // 1. while 循环语法结构  while 当...的时候
   while (条件表达式) {
        // 循环体
         }
        // 2. 执行思路  当条件表达式结果为true 则执行循环体 否则 退出循环
        var num = 1;
        while (num <= 100) {
        // 完成计数器的更新 防止死循环
            console.log('好啊有');
            num++;          // 初始化变量
        }

   </script>
```

数组

1. 数组(Array)： 就是一组数据的集合 存储在单个变量下的优雅方式
2. 利用new 创建数组（开辟内存空间）
3. 利用数组字面量创建数组 []

4. 我们数组里面的数据一定用逗号分隔
5. 数组里面的数据 比如1,2， 我们称为数组元素
6. 获取数组元素 格式 数组名[索引号] 索引号从 0开始
7. 没有这个数组元素 输出结果会是 undefined

数组长度 数组名.length （这个length以1开始计数）

```
var arr = ['关羽', '张飞', '马超', '赵云'];
console.log(arr.length);
        for (var i = 0; i < arr.length; i++) {
            console.log(arr[i]);
        }
```

数组的长度是元素个数 不要跟索引号混淆

arr.length 动态监测数组元素的个数

翻转数组

```
 <script>
        // 将数组 ['red', 'green', 'blue', 'pink', 'purple'] 的内容反过来存放
        // 1、声明一个新数组 newArr
        // 2、把旧数组索引号第4个取过来（arr.length - 1），给新数组索引号第0个元素
(newArr.length)
        // 3、我们采取 递减的方式  i--
        var arr = ['red', 'green', 'blue', 'purple', 'pink'];
        var newArr = [];
        for (var i = arr.length - 1; i >= 0; i--) {
            newArr[newArr.length] = arr[i]
        }
        console.log(newArr);
    </script>
```

冒泡排序

```
    <script>
        // 冒泡排序
        // var arr = [5, 4, 3, 2, 1];
        var arr = [4, 1, 2, 3, 5];
        for (var i = 0; i <= arr.length - 1; i++) { // 外层循环管趟数
            for (var j = 0; j <= arr.length - i - 1; j++) { // 里面的循环管 每一趟
的交换次数
                // 内部交换2个变量的值 前一个和后面一个数组元素相比较
                if (arr[j] < arr[j + 1]) {
                    var temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }

            }
        }
        console.log(arr);
    </script>
```

新增数组元素

```
<script>
        // 1. 新增数组元素 修改length长度
        var arr = ['one', 'two', 'three'];
        console.log(arr.length);
        arr.length = 5; // 把我们数组的长度修改为了 5  里面应该有5个元素
        console.log(arr);
        console.log(arr[3]); // undefined
        console.log(arr[4]); // undefined

        // 2. 新增数组元素 修改索引号 追加数组元素
        var arr1 = ['one', 'two', 'three'];
        arr1[3] = 'four';
        console.log(arr1);
        arr1[4] = 'five';
        console.log(arr1);
        arr1[0] = 'oooo'; // 这里是替换原来的数组元素
        console.log(arr1);
        arr1 = '???????';
        console.log(arr1); // 不能直接给 数组名赋值 否则里面的数组元素都没有了
    </script>
```

函数

```
function 函数名(参数1, 参数2, ... , 参数3) {
    // 需要执行的代码块
}
```

1、function是一个关键字, 和var、typeof一样, 都是关键字, 后面要加空格;
2、函数名的命名规范和变量命名一样, 只能是字母、数字、下划线、美元符号, 不能以数字开头, 一般采用驼峰式命名大;
3、函数名后的()中放置函数形参, 形参可以为任意多个(意味着可以没有形参), 如有有多个形参用","隔开;
4、函数{}中就是需要执行的代码块。

函数的调用

```
//函数名(实参1, 实参2, ... , 实参3);
function demo1(func1, func2, func3, func4){
    var a = func1;
    var b = func2;
    var c = func3;
    var d = func4;
    return a, b, c, d;
}

//函数的调用
demo1(1, 2, 3, 4)
```

函数表达式

匿名函数
使用function关键字声明一个函数，**但未给函数命名**，所以叫匿名函数

```
function (a, b) {
   console.log(a + b);
}
```

函数表达式
使用function关键字声明一个函数，**但未给函数命名，最后将匿名函数赋予一个变量，叫函数表达式。**

函数也是一种数据类型, 属于Object对象类型中的一种

```
var fun = function(a, b) {  // 匿名函数
   console.log(a + b);
};
fun(12, 13);
```

这个函数表达式很常用

arguments 的使用

只有函数才有 arguments对象 而且是每个函数都内置好了这个arguments

console.log(arguments); // 里面存储了所有传递过来的实参 arguments = [1,2,3, 4]

arguments的length属性
判断用户输入，如果只输入两个数，就返回两个数的和，如果返回三个数，就返回三个数的和。

```
function sum(a, b, c) {
    // 注意：数字和undefined做加法运算返回值为NaN
    var sum = a + b;
    if (arguments.length == 3) {
        sum += c;
    }
    return sum;
    }

console.log(sum(10, 20, 30));
```

arguments的callee属性

```
function fun(a, b) {
    // 由此可以看出，arguments.callee就是当前正在执行的函数
    console.log(arguments.callee);
    console.log(fun);        //  fun == arguments.callee
}
fun(10, 20);
```

利用函数冒泡排序 sort 排序

```
<script>
        // 利用函数冒泡排序 sort 排序
        function sort(arr) {
            for (var i = 0; i < arr.length - 1; i++) {
                for (var j = 0; j < arr.length - i - 1; j++) {
                    if (arr[j] > arr[j + 1]) {
```

```
                        var temp = arr[j];
                        arr[j] = arr[j + 1];
                        arr[j + 1] = temp;
                    }
                }
            }
            return arr;
        }
        var arr1 = sort([1, 4, 2, 9]);
        console.log(arr1);
        var arr2 = sort([11, 7, 22, 999]);
        console.log(arr2);
    </script>
```

函数是可以相互调用的

```
<script>
        // 函数是可以相互调用的
        // function fn1() {
        //     console.log(11);
        //     fn2(); // 在fn1 函数里面调用了 fn2 函数
        // }
        // fn1();

        // function fn2() {
        //     console.log(22);

        // }

        function fn1() {
            console.log(111);
            fn2();
            console.log('fn1');
        }

        function fn2() {
            console.log(222);
            console.log('fn2');
        }
        fn1();
    </script>
```

js的作用域

通常来说，一段程序代码中所用到的名字并不总是有效和可用的，而限定这个名字的可用性的代码范围就是这个名字的**作用域**。作用域的使用提高了程序逻辑的局部性，增强了程序的可靠性，减少了名字冲突。

```
<script>
        // 1.JavaScript作用域 ： 就是代码名字（变量）在某个范围内起作用和效果 目的是为了提高程序的可靠性更重要的是减少命名冲突
        // 2. js的作用域（es6）之前 ： 全局作用域  局部作用域
        // 3. 全局作用域： 整个script标签 或者是一个单独的js文件
        var num = 10;
        var num = 30;
        console.log(num);
```

```
        // 4. 局部作用域（函数作用域） 在函数内部就是局部作用域 这个代码的名字只在函数内部起效果和作用
        function fn() {
            // 局部作用域
            var num = 20;
            console.log(num);

        }
        fn();
    </script>

<script>
        // 变量的作用域： 根据作用域的不同我们变量分为全局变量和局部变量
        // 1. 全局变量： 在全局作用域下的变量 在全局下都可以使用
        // 注意 如果在函数内部 没有声明直接赋值的变量也属于全局变量
        var num = 10; // num就是一个全局变量
        console.log(num);

        function fn() {
            console.log(num);

        }
        fn();
        // console.log(aru);

        // 2. 局部变量 在局部作用域下的变量 后者在函数内部的变量就是 局部变量
        // 注意： 函数的形参也可以看做是局部变量
        function fun(aru) {
            var num1 = 10; // num1就是局部变量 只能在函数内部使用
            num2 = 20;
        }
        fun();
        // console.log(num1);
        // console.log(num2);
        // 3. 从执行效率来看全局变量和局部变量
        // (1) 全局变量只有浏览器关闭的时候才会销毁，比较占内存资源
        // (2) 局部变量 当我们程序执行完毕就会销毁， 比较节约内存资源
    </script>
```

全局变量： 在全局作用域下的变量 在全局下都可以使用， 全局变量只有浏览器关闭的时候才会销毁，比较占内存资源， 局部变量 当我们程序执行完毕就会销毁， 比较节约内存资源

块作用域由 { } 包括

在其他编程语言中（如 java、c#等），在 if 语句、循环语句中创建的变量，仅仅只能在本 if 语句、本循环语句中使用

Js中没有块级作用域（在ES6之前），之后就有了，一般let关键字用来声明块作用域

全局变量

在全局作用域下声明的变量叫做**全局变量**（在函数外部定义的变量）。

- 全局变量在代码的任何位置都可以使用
- 在全局作用域下 var 声明的变量 是全局变量
- 特殊情况下，在函数内不使用 var 声明的变量也是全局变量（不建议使用）

局部变量

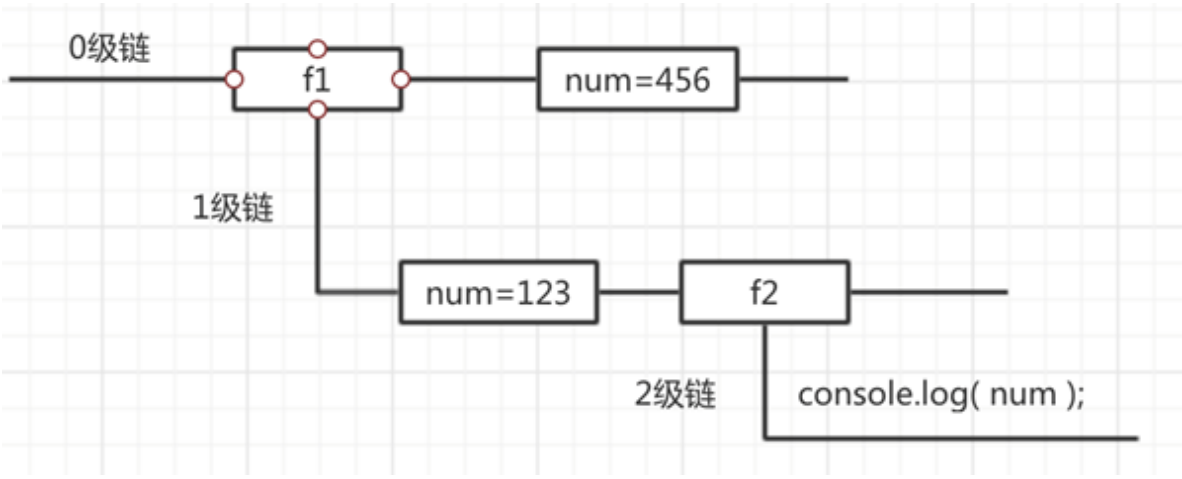
在局部作用域下声明的变量叫做**局部变量**（在函数内部定义的变量）

- 局部变量只能在该函数**内部**使用
- 在函数内部 var 声明的变量是局部变量
- 函数的**形参**实际上就是局部变量

全局变量和局部变量的区别

- 全局变量：在任何一个地方都可以使用，只有在浏览器关闭时才会被销毁，因此比较占内存
- 局部变量：只在函数内部使用，当其所在的代码块被执行时，会被初始化；当代码块运行结束后，就会被销毁，因此更节省内存空间

作用域链



作用域链：采取**就近原则**的方式来查找变量最终的值。

```
<script>
// 作用域链 ： 内部函数访问外部函数的变量，采取的是链式查找的方式来决定取那个值 这种结构我们称为作用域链  就近原则
        var num = 10;

        function fn() { // 外部函数
            var num = 20;

            function fun() { // 内部函数
                console.log(num);

            }
            fun();
        }
        fn();
    </script>


 <script>
        // 案例1 ： 结果是几?
        function f1() {
            var num = 123;

            function f2() {
                var num = 0;
                console.log(num); // 站在目标出发，一层一层的往外查找
```

```
            }
            f2();
        }
        var num = 456;
        f1();  // 结果为 ：  0
        // 案例2 ：结果是几?
        var a = 1;

        function fn1() {
            var a = 2;
            var b = '22';
            fn2();

            function fn2() {
                var a = 3;
                fn3();

                function fn3() {
                    var a = 4;
                    console.log(a); //a的值 ?
                    console.log(b); //b的值 ?
                }
            }
        }
        fn1();  // a==4 b==22
    </script>
```

什么是对象

在 JavaScript 中，对象是一组无序的相关属性和方法的集合，所有的事物都是对象，例如字符串、数值、数组、函数等

对象是由属性和方法组成的。

- 属性：事物的**特征**，在对象中用**属性**来表示（常用名词）
- 方法：事物的**行为**，在对象中用**方法**来表示（常用动词）

在 JavaScript 中，可以采用三种方式创建对象（object）：

- 利用字面量创建对象
- 利用 new Object 创建对象
- 利用构造函数创建对象

利用字面量创建对象

```
<script>
        // 1.利用对象字面量创建对象 {}
        // var obj = {};  // 创建了一个空的对象
        var obj = {
                uname: '张三疯',
                age: 18,
                sex: '男',
                sayHi: function() {
                    console.log('hi~');

                } // 称为方法
```

```
            }

// 2. 使用对象
         // (1). 调用对象的属性 我们采取 对象名.属性名 . 可以理解为 的
        console.log(obj.uname);
        // (2). 调用属性还有一种方法 对象名['属性名']
        console.log(obj['age']);
        // (3) 调用对象的方法 sayHi    对象名.方法名() 千万别忘记添加小括号
        obj.sayHi();
    </script>
```

(1) 里面的属性或者方法我们采取键值对的形式 键 属性名： 值 属性值
(2) 多个属性或者方法中间用逗号隔开的
(3) 方法冒号后面跟的是一个匿名函数

```
<script>
        // 变量、属性、函数、方法的区别
        // 1.变量和属性的相同点 他们都是用来存储数据的
        var num = 10;
        var obj = {
            age: 18,
            fn: function() {

            }
        } // 声明一个obj的对象

        function fn() {

        }
        console.log(obj.age);
        // console.log(age);


    </script>
```

1. **变量 单独声明并赋值 使用的时候直接写变量名 单独存在**
   属性 在对象里面的不需要声明的 使用的时候必须是 对象.属性
2. **函数和方法的相同点 都是实现某种功能 做某件事**
   函数是单独声明 并且调用的 函数名() 单独存在的
   方法 在对象里面 调用的时候 对象.方法()

利用new object创建对象

```
<script>
        // 利用 new Object 创建对象
        var obj = new Object(); // 创建了一个空的对象

        obj.uname = '张三';
        obj.age = 18;
        obj.sex = '男';
        obj.sayHi = function() {
                console.log('你好~');

            }
            // (1) 我们是利用 等号 = 赋值的方法 添加对象的属性和方法
```

```
            // (2) 每个属性和方法之间用 分号结束
        console.log(obj.uname);
        console.log(obj['sex']);
        obj.sayHi(); // 调用对象方法
    </script>
```

这两种创建对象的方式一次只能创建一个对象

使用构造函数

```
<script>
        // 我们为什么需要使用构造函数

        // 就是因我们前面两种创建对象的方式一次只能创建一个对象
        var ldh = {
            uname: '刘德华',
            age: 55,
            sing: function() {
                console.log('冰雨');

            } // 方法
        }
        var zxy = {
                uname: '张学友',
                age: 58,
                sing: function() {
                    console.log('李香兰');

                }
            }

    </script>
```

因为我们一次创建一个对象，里面很多的**属性**和**方法**是大量相同的 我们只能复制
因此我们可以利用函数的方法 重复这些相同的代码 我们就把这个函数称为 构造函数
又因为这个函数不一样，里面封装的不是普通代码，而是 对象构造函数 就是把我们对象里面一些相同的属性和方法抽象出来封装到函数里面

```
<script>
        // 利用构造函数创建对象
        // 我们需要创建四大天王的对象  相同的属性：  名字  年龄  性别   相同的方法：  唱歌
        // 构造函数的语法格式
        // function 构造函数名() {
        //     this.属性 = 值;
        //     this.方法 = function() {}
        // }
        // new 构造函数名();
        function Star(uname, age, sex) {
            this.name = uname;
            this.age = age;
            this.sex = sex;
            this.sing = function(sang) {
                console.log(sang);

            }
        }
```

```
        var ldh = new Star('刘德华', 18, '男'); // 调用函数返回的是一个对象
        // console.log(typeof ldh);
        console.log(ldh.name);
        console.log(ldh['sex']);
        ldh.sing('冰雨');
        var zxy = new Star('张学友', 19, '男');
        console.log(zxy.name);
        console.log(zxy.age);
        zxy.sing('李香兰')



    </script>
```

1. 构造函数名字首字母要大写
2. 构造函数不需要return 就可以返回结果
3. 调用构造函数 必须使用 new
4. 只要new Star() 调用函数就创建一个对象 ldh {}
5. 属性和方法前面必须添加 **this**

this就代表的是实例化对象后的对象变量名

```
 <script>
        // 构造函数和对象
        // 1. 构造函数  明星 泛指的某一大类  它类似于 java 语言里面的  类(class)
        function Star(uname, age, sex) {
            this.name = uname;
            this.age = age;
            this.sex = sex;
            this.sing = function(sang) {
                console.log(sang);

            }
        }
        // 2. 对象 特指 是一个具体的事物 刘德华 ==  {name: "刘德华", age: 18, sex:
"男", sing: ƒ}
        var ldh = new Star('刘德华', 18, '男'); // 调用函数返回的是一个对象
        console.log(ldh);
        // 3. 我们利用构造函数创建对象的过程我们也称为对象的实例化
    </script>
```

new关键字执行过程

```
 <script>
        // new关键字执行过程
        // 1. new 构造函数可以在内存中创建了一个空的对象
        // 2. this 就会指向刚才创建的空对象
        // 3. 执行构造函数里面的代码 给这个空对象添加属性和方法
        // 4. 返回这个对象
        function Star(uname, age, sex) {
            this.name = uname;
            this.age = age;
            this.sex = sex;
            this.sing = function(sang) {
                console.log(sang);
```

```
            }
        }
        var ldh = new Star('刘德华', 18, '男');
    </script>
```

遍历对象

```
 <script>
        // 遍历对象
        var obj = {
                name: 'pink老师',
                age: 18,
                sex: '男',
                fn: function() {}
            }
            // console.log(obj.name);
            // console.log(obj.age);
            // console.log(obj.sex);
            // for in 遍历我们的对象
            // for (变量 in 对象) {

        // }
// for in  与Python中的用法大相径庭
        for (var k in obj) {
            console.log(k); // k 变量 输出  得到的是 属性名
            console.log(obj[k]); // obj[k] 得到是 属性值

        }
        // 我们使用 for in 里面的变量 我们喜欢写 k  或者  key
    </script>
```

一些内置对象的用法

math对象的一些方法

```
 <script>
        // Math数学对象 不是一个构造函数 ，所以我们不需要new 来调用 而是直接使用里面的属性和方法即可
        console.log(Math.PI); // 一个属性 圆周率
        console.log(Math.max(1, 99, 3)); // 99
        console.log(Math.max(-1, -10)); // -1
        console.log(Math.max(1, 99, 'pink老师')); // NaN
        console.log(Math.max()); // -Infinity
    </script>


// 1.绝对值方法
        console.log(Math.abs(1)); // 1
        console.log(Math.abs(-1)); // 1
        console.log(Math.abs('-1')); // 隐式转换 会把字符串型 -1 转换为数字型
        console.log(Math.abs('pink')); // NaN

        // 2.三个取整方法
        // (1) Math.floor()   地板 向下取整  往最小了取值
        console.log(Math.floor(1.1)); // 1
```

```
        console.log(Math.floor(1.9)); // 1
        // (2) Math.ceil()   ceil 天花板 向上取整  往最大了取值
        console.log(Math.ceil(1.1)); // 2
        console.log(Math.ceil(1.9)); // 2
        // (3) Math.round()   四舍五入  其他数字都是四舍五入，但是 .5 特殊 它往大了取
        console.log(Math.round(1.1)); // 1
        console.log(Math.round(1.5)); // 2
        console.log(Math.round(1.9)); // 2
        console.log(Math.round(-1.1)); // -1
        console.log(Math.round(-1.5)); // 这个结果是 -1


  // 1.Math对象随机数方法   random() 返回一个随机的小数  0 =< x < 1
        // 2. 这个方法里面不跟参数
        // 3. 代码验证
        console.log(Math.random());
        // 4. 我们想要得到两个数之间的随机整数 并且 包含这2个整数
        // Math.floor(Math.random() * (max - min + 1)) + min;
        function getRandom(min, max) {
            return Math.floor(Math.random() * (max - min + 1)) + min;
        }
        console.log(getRandom(1, 10));
        // 5. 随机点名
        var arr = ['张三', '张三丰', '张三疯子', '李四', '李思思', 'pink老师'];
        // console.log(arr[0]);
        console.log(arr[getRandom(0, arr.length - 1)]);
```

封装自己的对象

```
 <script>
        // 利用对象封装自己的数学对象  里面有 PI 最大值和最小值
        var myMath = {
            PI: 3.141592653,
            max: function() {
                var max = arguments[0]; // 里面存储了所有传递过来的实参 ,取其中的第零个
值
                for (var i = 1; i < arguments.length; i++) {
                    if (arguments[i] > max) {
                        max = arguments[i];
                    }
                }
                return max;
            }, // 最大值方法
            min: function() {
                var min = arguments[0];
                for (var i = 1; i < arguments.length; i++) {
                    if (arguments[i] < min) {
                        min = arguments[i];
                    }
                }
                return min;
            }  // 最小值方法
        }
```

```
        console.log(myMath.PI);
        console.log(myMath.max(1, 5, 9));
        console.log(myMath.min(1, 5, 9));
    </script>
```

Date日期对象

```
<script>
        // Date() 日期对象  是一个构造函数 必须使用new 来调用创建我们的日期对象
        var arr = new Array(); // 创建一个数组对象
        var obj = new Object(); // 创建了一个对象实例
        // 1. 使用Date  如果没有参数 返回当前系统的当前时间
        var date = new Date();
        console.log(date);
        // 2. 参数常用的写法  数字型  2019, 10, 01  或者是 字符串型 '2019-10-1 8:8:8'
        var date1 = new Date(2019, 10, 1);
        console.log(date1); // 返回的是 11月 不是 10月
        var date2 = new Date('2019-10-1 8:8:8');
        console.log(date2);


  // 格式化日期 年月日
        var date = new Date();
        console.log(date.getFullYear()); // 返回当前日期的年  2019
        console.log(date.getMonth() + 1); // 月份 返回的月份小1个月    记得月份+1 呦
        console.log(date.getDate()); // 返回的是 几号
        console.log(date.getDay()); // 3  周一返回的是 1 周六返回的是 6 但是 周日返回
的是 0
        // 我们写一个 2019年 5月 1日 星期三
        var year = date.getFullYear();
        var month = date.getMonth() + 1;
        var dates = date.getDate();
        var arr = ['星期日', '星期一', '星期二', '星期三', '星期四', '星期五', '星期
六'];
        var day = date.getDay();
        console.log('今天是：' + year + '年' + month + '月' + dates + '日 ' +
arr[day]);
     // 格式化日期 年月日
        var date = new Date();
        console.log(date.getFullYear()); // 返回当前日期的年  2019
        console.log(date.getMonth() + 1); // 月份 返回的月份小1个月    记得月份+1 呦
        console.log(date.getDate()); // 返回的是 几号
        console.log(date.getDay()); // 3  周一返回的是 1 周六返回的是 6 但是 周日返回
的是 0
        // 我们写一个 2019年 5月 1日 星期三
        var year = date.getFullYear();
        var month = date.getMonth() + 1;
        var dates = date.getDate();
        var arr = ['星期日', '星期一', '星期二', '星期三', '星期四', '星期五', '星期
六'];
        var day = date.getDay();
        console.log('今天是：' + year + '年' + month + '月' + dates + '日 ' +
arr[day]);



// 获得Date总的毫秒数(时间戳)  不是当前时间的毫秒数 而是距离1970年1月1号过了多少毫秒数
```

```
        // 1. 通过 valueOf()  getTime()
        var date = new Date();
        console.log(date.valueOf()); // 就是 我们现在时间 距离1970.1.1 总的毫秒数
        console.log(date.getTime());
        // 2. 简单的写法 (最常用的写法)
        var date1 = +new Date(); // +new Date()  返回的就是总的毫秒数
        console.log(date1);
        // 3. H5 新增的 获得总的毫秒数
        console.log(Date.now());




    </script>
```

倒计时效果

```
 <script>
        // 倒计时效果
        // 1.核心算法：输入的时间减去现在的时间就是剩余的时间，即倒计时 ，但是不能拿着时分秒相减，比如 05 分减去25分，结果会是负数的。
        // 2.用时间戳来做。用户输入时间总的毫秒数减去现在时间的总的毫秒数，得到的就是剩余时间的毫秒数。
        // 3.把剩余时间总的毫秒数转换为天、时、分、秒 （时间戳转换为时分秒）
        // 转换公式如下：
        //  d = parseInt(总秒数/ 60/60 /24);     //  计算天数
        //  h = parseInt(总秒数/ 60/60 %24)    //    计算小时
        //  m = parseInt(总秒数 /60 %60 );      //    计算分数
        //  s = parseInt(总秒数%60);              //    计算当前秒数
        function countDown(time) {
            var nowTime = +new Date(); // 返回的是当前时间总的毫秒数
            var inputTime = +new Date(time); // 返回的是用户输入时间总的毫秒数
            var times = (inputTime - nowTime) / 1000; // times是剩余时间总的秒数
            var d = parseInt(times / 60 / 60 / 24); // 天
            d = d < 10 ? '0' + d : d;
            var h = parseInt(times / 60 / 60 % 24); //时
            h = h < 10 ? '0' + h : h;
            var m = parseInt(times / 60 % 60); // 分
            m = m < 10 ? '0' + m : m;
            var s = parseInt(times % 60); // 当前的秒
            s = s < 10 ? '0' + s : s;
            return d + '天' + h + '时' + m + '分' + s + '秒';
        }
        console.log(countDown('2019-5-1 18:00:00'));
        var date = new Date();
        console.log(date);
    </script>
```

创建数组的两种方式

创建数组的两种方式

1. 利用数组字面量

var arr = [1, 2, 3];

console.log(arr[0]);

2. 利用new Array()

var arr1 = new Array(); // 创建了一个空的数组

var arr1 = new Array(2); // 这个2 表示 数组的长度为 2 里面有2个空的数组元素

var arr1 = new Array(2, 3); // 等价于 [2,3] 这样写表示 里面有2个数组元素 是 2和3

console.log(arr1);

检测是否为数组方法

```
<script>
        // 翻转数组
        function reverse(arr) {
            // if (arr instanceof Array) {
            if (Array.isArray(arr)) {
                var newArr = [];
                for (var i = arr.length - 1; i >= 0; i--) {
                    newArr[newArr.length] = arr[i];

                }
                return newArr;
            } else {
                return 'error 这个参数要求必须是数组格式 [1,2,3]'
            }
        }
        console.log(reverse([1, 2, 3]));
        console.log(reverse(1, 2, 3));
        // 检测是否为数组
        // (1) instanceof  运算符 它可以用来检测是否为数组
        var arr = [];
        var obj = {};
        console.log(arr instanceof Array);
        console.log(obj instanceof Array);
        // (2) Array.isArray(参数);  H5新增的方法  ie9以上版本支持
        console.log(Array.isArray(arr));
        console.log(Array.isArray(obj));
    </script>
```

添加或删除数组元素方法

```
<script>
        // 添加删除数组元素方法
        // 1. push() 在我们数组的末尾 添加一个或者多个数组元素    push : 推
        var arr = [1, 2, 3];
        // arr.push(4, 'pink');
        console.log(arr.push(4, 'pink')); // 在后面添加push括号里面的元素

        console.log(arr);
        // (1) push 是可以给数组追加新的元素
        // (2) push() 参数直接写 数组元素就可以了
        // (3) push完毕之后，返回的结果是 新数组的长度
        // (4) 原数组也会发生变化
        // 2. unshift 在我们数组的开头 添加一个或者多个数组元素
        console.log(arr.unshift('red', 'purple'));
```

```
        console.log(arr);
        // (1) unshift是可以给数组前面追加新的元素
        // (2) unshift() 参数直接写 数组元素就可以了
        // (3) unshift完毕之后，返回的结果是 新数组的长度
        // (4) 原数组也会发生变化

        // 3. pop() 它可以删除数组的最后一个元素
        console.log(arr.pop());
        console.log(arr);
        // (1) pop是可以删除数组的最后一个元素 记住一次只能删除一个元素
        // (2) pop() 没有参数
        // (3) pop完毕之后，返回的结果是 删除的那个元素
        // (4) 原数组也会发生变化
        // 4. shift() 它可以删除数组的第一个元素
        console.log(arr.shift());
        console.log(arr);
        // (1) shift是可以删除数组的第一个元素 记住一次只能删除一个元素
        // (2) shift() 没有参数
        // (3) shift完毕之后，返回的结果是 删除的那个元素
        // (4) 原数组也会发生变化
    </script>
```

筛选数组

```
 <script>
        // 有一个包含工资的数组[1500, 1200, 2000, 2100, 1800]，要求把数组中工资超过2000的删除，剩余的放到新数组里面
        var arr = [1500, 1200, 2000, 2100, 1800];
        var newArr = [];
        for (var i = 0; i < arr.length; i++) {
            if (arr[i] < 2000) {
                // newArr[newArr.length] = arr[i];
                newArr.push(arr[i]);
            }
        }
        console.log(newArr);
    </script>
```

数组排序

```
<script>
        // 数组排序
        // 1. 翻转数组
        var arr = ['pink', 'red', 'blue'];
        arr.reverse();
        console.log(arr);

        // 2. 数组排序（冒泡排序）
        var arr1 = [13, 4, 77, 1, 7];
        arr1.sort(function(a, b) {
            //  return a - b; 升序的顺序排列
            return b - a; // 降序的顺序排列
        });
```

```
        console.log(arr1);
    </script>
```

获取数组元素索引的方法

```
 <script>
        // 返回数组元素索引号方法  indexOf(数组元素)  作用就是返回该数组元素的索引号 从前面开始查找
        // 它只返回第一个满足条件的索引号
        // 它如果在该数组里面找不到元素，则返回的是 -1
        // var arr = ['red', 'green', 'blue', 'pink', 'blue'];
        var arr = ['red', 'green', 'pink'];
        console.log(arr.indexOf('blue'));
        // 返回数组元素索引号方法  lastIndexOf(数组元素)  作用就是返回该数组元素的索引号从后面开始查找
        var arr = ['red', 'green', 'blue', 'pink', 'blue'];

        console.log(arr.lastIndexOf('blue')); // 4
    </script>
```

数组去重

```
<script>
        // 数组去重 ['c', 'a', 'z', 'a', 'x', 'a', 'x', 'c', 'b'] 要求去除数组中重复的元素。
        // 1.目标： 把旧数组里面不重复的元素选取出来放到新数组中， 重复的元素只保留一个， 放到新数组中去重。
        // 2.核心算法： 我们遍历旧数组， 然后拿着旧数组元素去查询新数组， 如果该元素在新数组里面没有出现过， 我们就添加， 否则不添加。
        // 3.我们怎么知道该元素没有存在？ 利用 新数组.indexOf(数组元素) 如果返回时 - 1 就说明 新数组里面没有改元素
        // 封装一个 去重的函数 unique 独一无二的
        function unique(arr) {
            var newArr = [];
            for (var i = 0; i < arr.length; i++) {
                if (newArr.indexOf(arr[i]) === -1) {
                    newArr.push(arr[i]);
                }
            }
            return newArr;
        }
        // var demo = unique(['c', 'a', 'z', 'a', 'x', 'a', 'x', 'c', 'b'])
        var demo = unique(['blue', 'green', 'blue'])
        console.log(demo);
    </script>
```

数组转化为字符串

```
<script>
        // 数组转换为字符串
        // 1. toString() 将我们的数组转换为字符串
        var arr = [1, 2, 3];
        console.log(arr.toString()); // 1,2,3
        // 2. join(分隔符)
        var arr1 = ['green', 'blue', 'pink'];
        console.log(arr1.join()); // green,blue,pink
        console.log(arr1.join('-')); // green-blue-pink
        console.log(arr1.join('&')); // green&blue&pink
    </script>
```

基本包装类型

```
<script>
        // 基本包装类型
        var str = 'andy';
        console.log(str.length);
        // 对象 才有 属性和方法    复杂数据类型才有 属性和方法
        // 简单数据类型为什么会有length 属性呢？
        // 基本包装类型：   就是把简单数据类型 包装成为了 复杂数据类型
        // (1) 把简单数据类型包装为复杂数据类型
        var temp = new String('andy');
        // (2) 把临时变量的值 给 str
        str = temp;
        // (3) 销毁这个临时变量
        temp = null;
    </script>
```

字符串的不可变性

```
<script>
        // 字符串的不可变性
        var str = 'andy';
        console.log(str);
        str = 'red';
        console.log(str);
        // 因为字符串的不可变 所以不要大量的拼接字符串
        var str = '';
        for (var i = 1; i <= 1000000000; i++) {
            str += i;
        }
        console.log(str);
    </script>
```

根据字符返回位置

```
<script>
        // 字符串对象  根据字符返回位置  str.indexOf('要查找的字符', [起始的位置])
        var str = '改革春风吹满地，春天来了';
        console.log(str.indexOf('春'));
        console.log(str.indexOf('春', 3)); // 从索引号是 3的位置开始往后查找
    </script>
```

查找字符串"abcoefoxyozzopp"中所有o出现的位置以及次数

```
<script>
        // 查找字符串"abcoefoxyozzopp"中所有o出现的位置以及次数
        // 核心算法：先查找第一个o出现的位置
        // 然后 只要indexOf 返回的结果不是 -1 就继续往后查找
        // 因为indexOf 只能查找到第一个，所以后面的查找，一定是当前索引加1，从而继续查找
        var str = "oabcoefoxyozzopp";
        var index = str.indexOf('o');
        var num = 0;
        // console.log(index);
        while (index !== -1) {
            console.log(index);
            num++;
            index = str.indexOf('o', index + 1);
        }
        console.log('o出现的次数是: ' + num);
        // 课后作业 ['red', 'blue', 'red', 'green', 'pink','red'], 求 red 出现的位
置和次数
    </script>
```

根据位置返回字符

```
<script>
        // 根据位置返回字符
        // 1. charAt(index) 根据位置返回字符
        var str = 'andy';
        console.log(str.charAt(3));
        // 遍历所有的字符
        for (var i = 0; i < str.length; i++) {
            console.log(str.charAt(i));
        }
        // 2. charCodeAt(index)  返回相应索引号的字符ASCII值 目的： 判断用户按下了那个键
        console.log(str.charCodeAt(0)); // 97
        // 3. str[index] H5 新增的
        console.log(str[0]); // a
    </script>
```

统计出现最多的字符和次数

```
 <script>
        // 有一个对象 来判断是否有该属性 对象['属性名']
        var o = {
            age: 18
        }
        if (o['sex']) {
            console.log('里面有该属性');

        } else {
            console.log('没有该属性');

        }

        //  判断一个字符串 'abcoefoxyozzopp' 中出现次数最多的字符，并统计其次数。
        // o.a = 1
```

```
        // o.b = 1
        // o.c = 1
        // o.o = 4
        // 核心算法：利用 charAt() 遍历这个字符串
        // 把每个字符都存储给对象， 如果对象没有该属性，就为1，如果存在了就 +1
        // 遍历对象，得到最大值和该字符
        var str = 'abcoefoxyozzopp';
        var o = {};
        for (var i = 0; i < str.length; i++) {
            var chars = str.charAt(i); // chars 是 字符串的每一个字符
            if (o[chars]) { // o[chars] 得到的是属性值
                o[chars]++;
            } else {
                o[chars] = 1;
            }
        }
        console.log(o);
        // 2. 遍历对象
        var max = 0;
        var ch = '';
        for (var k in o) {
            // k 得到是 属性名
            // o[k] 得到的是属性值
            if (o[k] > max) {
                max = o[k];
                ch = k;
            }
        }
        console.log(max);
        console.log('最多的字符是' + ch);
    </script>


<script>
        // 字符串操作方法
        // 1. concat('字符串1','字符串2'....)
        var str = 'andy';
        console.log(str.concat('red'));

        // 2. substr('截取的起始位置', '截取几个字符');
        var str1 = '改革春风吹满地';
        console.log(str1.substr(2, 2)); // 第一个2 是索引号的2 从第几个开始  第二个2
是取几个字符
    </script>


 <script>
        // 1. 替换字符 replace('被替换的字符', '替换为的字符')  它只会替换第一个字符
        var str = 'andyandy';
        console.log(str.replace('a', 'b'));
        // 有一个字符串 'abcoefoxyozzopp'  要求把里面所有的 o 替换为 *
        var str1 = 'abcoefoxyozzopp';
        while (str1.indexOf('o') !== -1) {
            str1 = str1.replace('o', '*');
        }
        console.log(str1);

        // 2. 字符转换为数组 split('分隔符')     前面我们学过 join 把数组转换为字符串
```

```
        var str2 = 'red, pink, blue';
        console.log(str2.split(','));
        var str3 = 'red&pink&blue';
        console.log(str3.split('&'));
    </script>
```

数据类型

简单数据类型null

```
<script>
        // 简单数据类型 null  返回的是一个空的对象  object
        var timer = null;
        console.log(typeof timer);
        // 如果有个变量我们以后打算存储为对象，暂时没想好放啥， 这个时候就给 null
        // 1. 简单数据类型 是存放在栈里面 里面直接开辟一个空间存放的是值
        // 2. 复杂数据类型 首先在栈里面存放地址 十六进制表示  然后这个地址指向堆里面的数据
    </script>
```

```
 <script>
        // 简单数据类型传参
        function fn(a) {
            a++;
            console.log(a);
        }
        var x = 10;
        fn(x);
        console.log(x);
    </script>
```

堆和栈

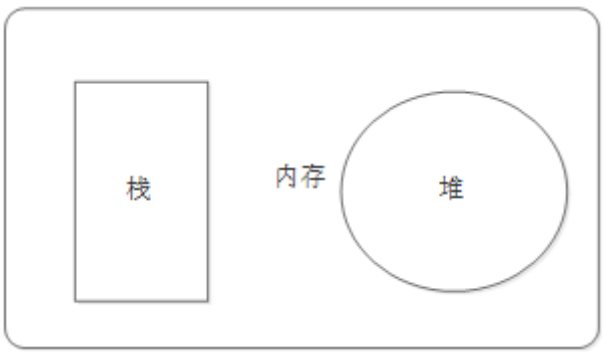
堆栈空间分配区别：

1、栈（操作系统）：由操作系统自动分配释放存放函数的参数值、局部变量的值等。其操作方式类似于数据结构中的栈；

简单数据类型存放到栈里面

2、堆（操作系统）：存储复杂类型(对象)，一般由程序员分配释放，若程序员不释放，由垃圾回收机制回收。

复杂数据类型存放到堆里面



复杂数据类型传参

```
<script>
        // 复杂数据类型传参
        function Person(name) {
            this.name = name;
        }

        function f1(x) { // x = p
            console.log(x.name); // 2. 这个输出什么 ?  刘德华
            x.name = "张学友";
            console.log(x.name); // 3. 这个输出什么 ?   张学友
        }
        var p = new Person("刘德华");
        console.log(p.name); // 1. 这个输出什么 ?   刘德华
        f1(p);
        console.log(p.name); // 4. 这个输出什么 ?   张学友
    </script>
```