JAVASCRIPT

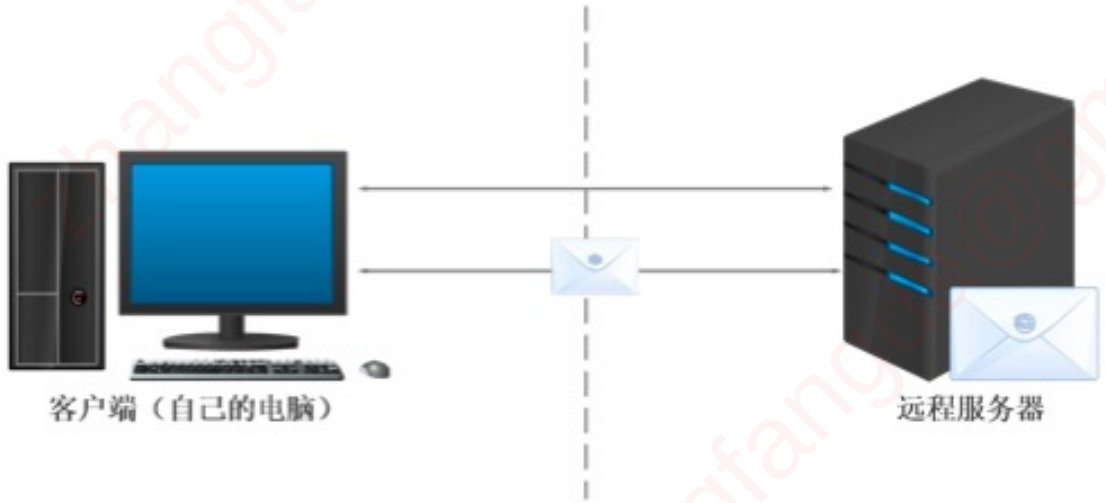
What is JavaScript

1.1 JavaScript 是什么

- 布兰登·艾奇（Brendan Eich，1961年~）。
- 神奇的大哥用10天完成 JavaScript 设计。
- 最初命名为 LiveScript，后来在与 Sun 合作之后将其改名为 JavaScript。

What is JavaScript

- JavaScript 是世界上最流行的语言之一，是一种运行在客户端的脚本语言 （Script 是脚本的意思）
- 脚本语言：不需要编译，运行过程中由 js 解释器(js 引擎）逐行来进行解释并执行
- 现在也可以基于 Node.js 技术进行服务器端编程



JavaScript 的作用

- 表单动态校验（密码强度检测）　（JS 产生最初的目的）
- 网页特效
- 服务端开发(Node.js)
- 桌面程序(Electron)
- App(Cordova)
- 控制硬件-物联网(Ruff)
- 游戏开发(cocos2d-js)

HTML/CSS/JS 的关系

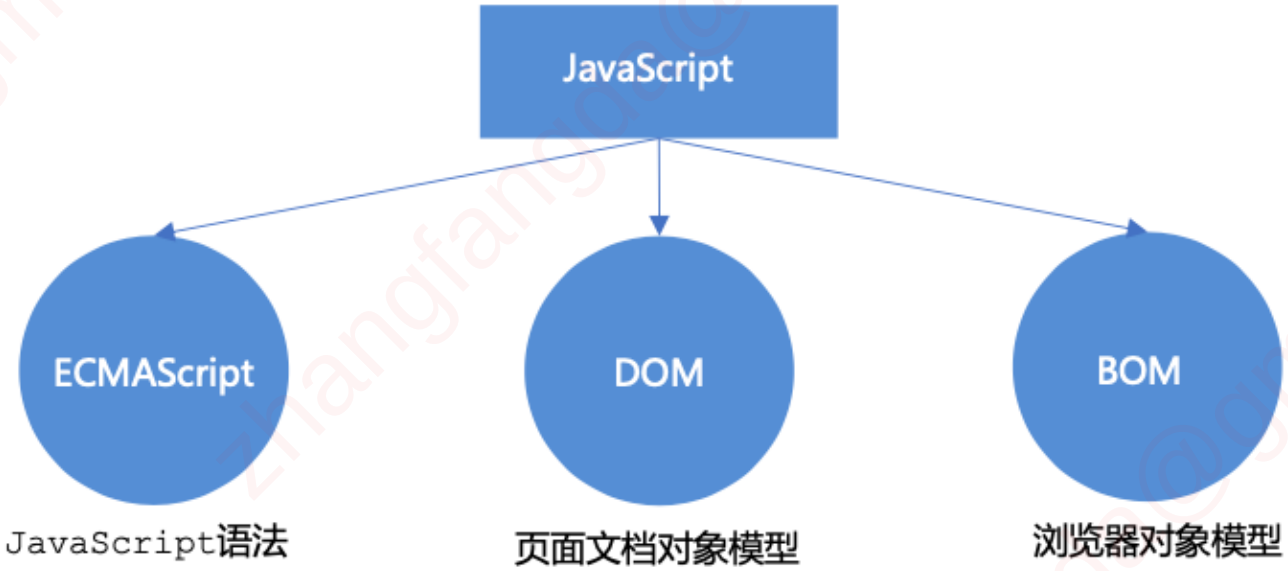
HTML/CSS 标记语言--描述类语言

- HTML 决定网页结构和内容(决定看到什么)，相当于人的身体
- CSS 决定网页呈现给用户的模样(决定好不好看)，相当于给人穿衣服、化妆

JS 脚本语言--编程类语言

- 实现业务逻辑和页面控制(决定功能)，相当于人的各种动作

JS 的组成



JS 的组成

2. DOM ——文档对象模型

文档对象模型（Document Object Model，简称DOM），是W3C组织推荐的处理可扩展标记语言的标准编程接口。

通过 DOM 提供的接口可以对页面上的各种元素进行操作（大小、位置、颜色等）。

JS 的组成

3. BOM ——浏览器对象模型

BOM (Browser Object Model，简称BOM) 是指浏览器对象模型，它提供了独立于内容的、可以与浏览器窗口进行互动的对象结构。通过BOM可以操作浏览器窗口，比如弹出框、控制浏览器跳转、获取分辨率等。

JS 有3种书写位置，分别为行内、内嵌和外部。

1. 行内式 JS

```
<input type="button" value="点我试试" onclick="alert('Hello World')" />
```

- 可以将单行或少量 JS 代码写在HTML标签的事件属性中（以 on 开头的属性），如：onclick
- 注意单双引号的使用：在HTML中我们推荐使用双引号, JS 中我们推荐使用单引号
- 可读性差， 在html中编写JS大量代码时，不方便阅读；
- 引号易错，引号多层嵌套匹配时，非常容易弄混；
- 特殊情况下使用

JS 有3种书写位置，分别为行内、内嵌和外部。

2. 内嵌 JS

```
<script>
    alert('Hello  World~!');
</script>
```

- 可以将多行JS代码写到 <script> 标签中
- 内嵌 JS 是学习时常用的方式

JS 有3种书写位置，分别为行内、内嵌和外部。

3. 外部 JS文件

```
<script src="my.js"></script>
```

- 利于HTML页面代码结构化，把大段 JS代码独立到 HTML 页面之外，既美观，也方便文件级别的复用
- 引用外部 JS文件的 script 标签中间不可以写代码
- 适合于JS 代码量比较大的情况

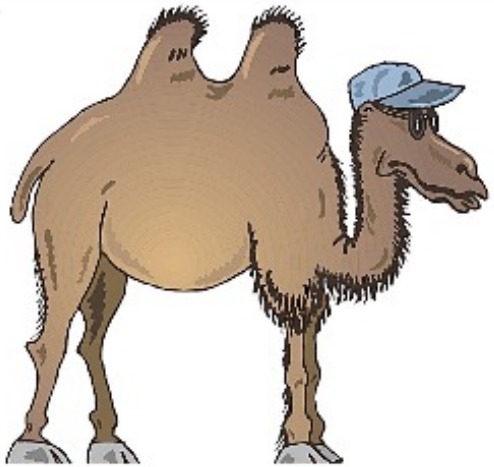
变量 Variable

声明变量特殊情况

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

命名规范

- 由字母(A-Za-z)、数字(0-9)、下划线(_)、美元符号($)组成，如：usrAge, num01, _name
- 严格区分大小写。var app; 和 var App; 是两个变量
- 不能 以数字开头。 18age 是错误的
- 不能 是关键字、保留字。例如：var、for、while
- 变量名必须有意义。 MMD BBD nl → age
- 遵守驼峰命名法。首字母小写，后面单词的首字母需要大写。 myFirstName
- 推荐翻译网站： 有道 爱词霸



以下哪些是合法的变量名?

第一组	第二组	第三组
var a	var userName	var theWorld
var 1	var $name	var the world
var age18	var _sex	var the_world
var 18age	var &sex	var for

变量，属性，函数，参数取的名字不能是关键字或保留字

关键字：是指 JS本身已经使用了的字，不能再用它们充当变量名、方法名。

包括：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 等。

保留字：实际上就是预留的“关键字”，意思是现在虽然还不是关键字，但是未来可能会成为关键字，同样不能使用它们当变量名或方法名。

包括：boolean、byte、char、class、const、debugger、double、enum、export、extends、fimal、float、goto、implements、import、int、interface、long、mative、package、private、protected、public、short、static、super、synchronized、throws、transient、volatile 等。

注意：如果将保留字用作变量名或函数名，那么除非将来的浏览器实现了该保留字，否则很可能收不到任何错误消息。当浏览器将其实现后，该单词将被看做关键字，如此将出现关键字错误。

变量小结

- 为什么需要变量?
- 变量是什么?
- 变量的本质是什么?
- 变量怎么使用的?

- 什么是变量的初始化?
- 变量命名规范有哪些?
- 交换2个变量值的思路?

- 因为我们一些数据需要保存，所以需要变量
- 变量就是一个容器，用来存放数据的。方便我们以后使用里面的数据
- 变量是内存里的一块空间，用来存储数据。
- 我们使用变量的时候，一定要声明变量，然后赋值
- 声明变量本质是去内存申请空间。
- 声明变量并赋值我们称之为变量的初始化
- 变量名尽量要规范，见名知意——驼峰命名法
- 区分哪些变量名不合法
- 学会交换2个变量

Let const 和var区别

在 ES5，JavaScript 只有两种作用域：**全局作用域，函数作用域**。

ES6新增了**块级作用域**，块作用域由 { } 包括，if语句和 for语句里面的{ }也属于块作用域。

1.Var 声明的变量会挂载在 window 上，而 let 和 const 声明的变量不会

2.Var 声明的变量存在变量提升，let 和 const 不存在变量提升

3.同一作用域下 var 可以多次声明同名变量，let 和 const不可以

4.Let 和 const 声明会形成**块级**作用域

5.Let,const 暂存死区

6.Const 一旦声明必须赋值，不能用 null 占位，声明后不能再修改，如果声明的是复合类型数据，可以修改属性

数据类型 Data type

为什么需要数据类型

在计算机中，不同的数据所需占用的存储空间是不同的，为了便于把数据分成所需内存大小不同的数据，充分利用存储空间，于是定义了不同的数据类型。

简单来说，数据类型就是数据的类别型号。比如姓名“张三”，年龄18，这些数据的类型是不一样的。

简单数据类型

JavaScript 中的简单数据类型及其说明如下：

简单数据类型	说明	默认值
Number	数字型，包含 整型值和浮点型值，如 21、0.21	0
Boolean	布尔值类型，如 true 、false，等价于 1 和 0	false
String	字符串类型，如 "张三" 注意咱们js 里面，字符串都带引号	""
Undefined	var a; 声明了变量 a 但是没有给值，此时 a = undefined	undefined
Null	var a = null; 声明了变量 a 为空值	null

数据类型转换

使用表单、prompt 获取过来的数据默认是字符串类型的，此时就不能直接简单的进行加法运算，而需要转换变量的数据类型。通俗来说，就是把一种数据类型的变量转换成另外一种数据类型。

我们通常会实现3种方式的转换：

- 转换为字符串类型
- 转换为数字型
- 转换为布尔型

转换为字符串

方式	说明	案例
toString()	转成字符串	var num= 1; alert(num.toString());
String() 强制转换	转成字符串	var num = 1; alert(String(num));
加号拼接字符串	和字符串拼接的结果都是字符串	var num = 1; alert(num+ "我是字符串");

- toString() 和 String() 使用方式不一样。
- 三种转换方式，我们更喜欢用第三种加号拼接字符串转换方式， 这一种方式也称之为隐式转换。

转换为数字型（重点）

方式	说明	案例
parseInt(string) 函数	将string类型转成整数数值型	parseInt('78')
parseFloat(string) 函数	将string类型转成浮点数数值型	parseFloat('78.21')
Number() 强制转换函数	将string类型转换为数值型	Number('12')
js 隐式转换(- * /)	利用算术运算隐式转换为数值型	'12' - 0

- 注意 parseInt 和 parseFloat 单词的大小写，这2个是重点
- 隐式转换是我们在进行算数运算的时候，JS 自动转换了数据类型

转换为boolean

方式	说明	案例
Boolean()函数	其他类型转成布尔值	Boolean('true');

- 代表空、否定的值会被转换为 false ，如 ''、0、NaN、null、undefined
- 其余值都会被转换为 true

```
console.log(Boolean('')); // false
console.log(Boolean(0)); // false
console.log(Boolean(NaN)); // false
console.log(Boolean(null)); // false
console.log(Boolean(undefined)); // false
console.log(Boolean('小白')); // true
console.log(Boolean(12)); // true
```

- 运算符
- 算数运算符
- 递增和递减运算符
- 比较运算符
- 逻辑运算符
- 赋值运算符
- 运算符优先级

前置递增运算符

1. 前置递增运算符

++num 前置递增，就是自加1，类似于 num = num + 1，但是 ++num 写起来更简单。

使用口诀：先自加，后返回值

```
var  num = 10;
alert(++num + 10);    // 21
```

后置递增运算符

num++ 后置递增，就是自加1，类似于 num = num + 1 ，但是 num++ 写起来更简单。

使用口诀：先返回原值，后自加

```
var  num = 10;
alert(10 + num++);  // 20
```

小结

- 前置递增和后置递增运算符可以简化代码的编写，让变量的值 + 1 比以前写法更简单
- 单独使用时，运行结果相同
- 与其他代码联用时，执行结果会不同
- 后置：先原值运算，后自加（先人后己）
- 前置：先自加，后运算（先己后人）
- 开发时，大多使用后置递增/减，并且代码独占一行，例如：num++; 或者 num--;

Comparator 比较运算符

概念：比较运算符（关系运算符）是两个数据进行比较时所使用的运算符，比较运算后，会返回一个布尔值（true / false）作为比较运算的结果。

运算符名称	说明	案例	结果
<	小于号	1 < 2	true
>	大于号	1 > 2	false
>=	大于等于号 (大于或者等于)	2 >= 2	true
<=	小于等于号 (小于或者等于)	3 <= 2	false
==	判等号（会转型）	37 == 37	true
!=	不等号	37 != 37	false
=== !==	全等 要求值和 数据类型都一致	37 === '37'	false

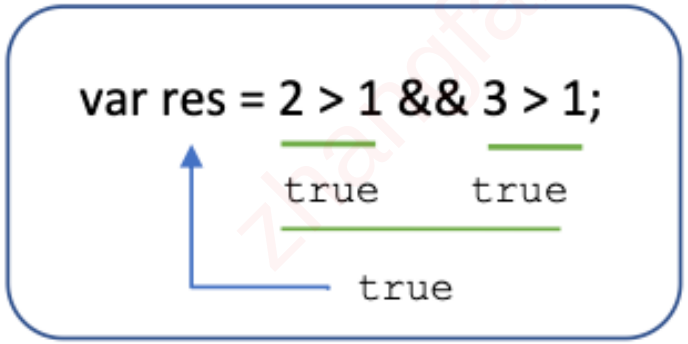
Comparator 比较运算符小结

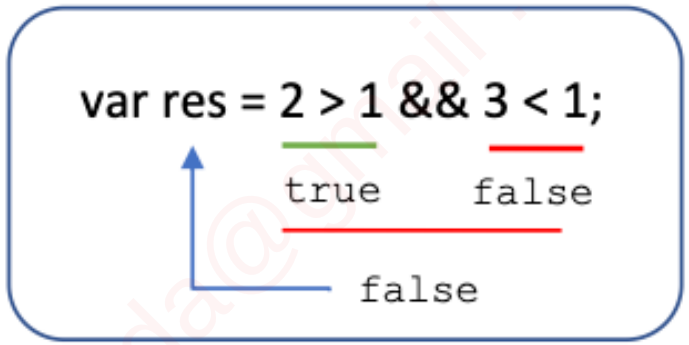
符号	作用	用法
=	赋值	把右边给左边
==	判断	判断两边值是否相等（注意此时有隐式转换）
===	全等	判断两边的值和数据类型是否完全相同

```
console.log(18 == '18');
console.log(18 === '18');
```

逻辑运算符&&

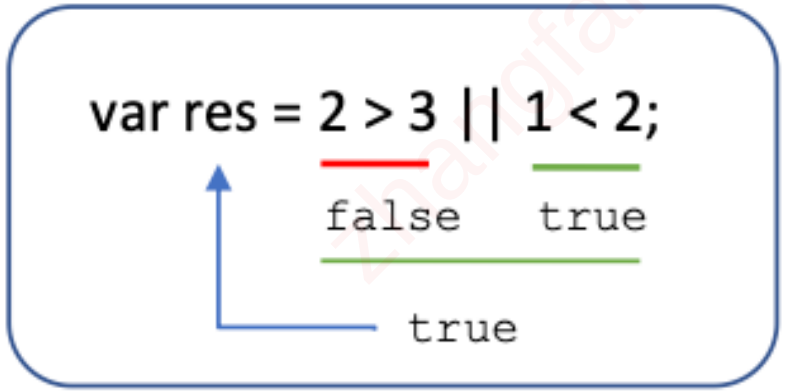
两边都是 true才返回 true，否则返回 false

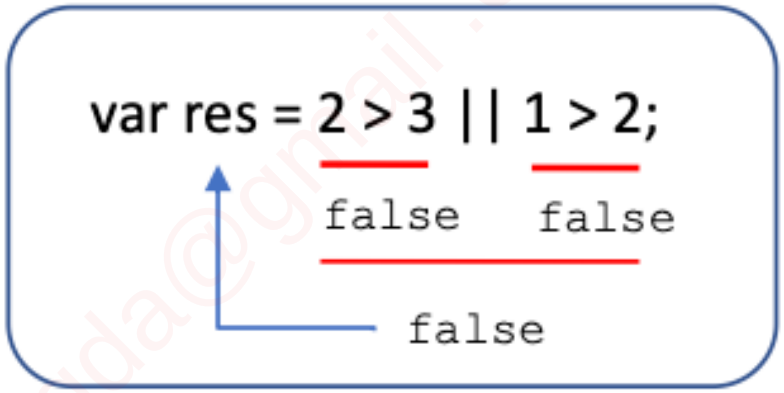




逻辑运算符||

两边都为 false 才返回 false，否则都为true





运算符优先级

优先级	运算符	顺序
1	小括号	()
2	一元运算符	++ -- !
3	算数运算符	**先 * / % 后 + -**
4	关系运算符	> >= < <=
5	相等运算符	== != === !==
6	逻辑运算符	**先 && 后 ||**
7	赋值运算符	=
8	逗号运算符	,

- 一元运算符里面的逻辑非优先级很高
- 逻辑与比逻辑或优先级高

三元表达式

```
var time = prompt('请您输入一个 0 ~ 59 之间的一个数字');
// 三元表达式 表达式 ? 表达式1 : 表达式2
var result = time < 10 ? '0' + time : time; // 把返回值赋值给一个变量
alert(result);
```

switch

switch 语句也是多分支语句，它用于基于不同的条件来执行不同的代码。当要针对变量设置一系列的特定值的选项时，就可以使用 switch。

```
switch( 表达式 ){
    case value1:
        // 表达式 等于 value1 时要执行的代码
        break;
    case value2:
        // 表达式 等于 value2 时要执行的代码
        break;
    default:
        // 表达式 不等于任何一个 value 时要执行的代码
}
```

Switch和if else if的区别

① 一般情况下，它们两个语句可以相互替换

② switch...case 语句通常处理 case为比较确定值的情况， 而 if...else...语句更加灵活，常用于范围判断(大于、等于某个范围)

③ switch 语句进行条件判断后直接执行到程序的条件语句，效率更高。而if...else 语句有几种条件，就得判断多少次。

④ 当分支比较少时，if... else语句的执行效率比 switch语句高。

⑤ 当分支比较多时，switch语句的执行效率比较高，而且结构更清晰。

JS中的循环

在Js 中，主要有三种类型的循环语句：

- for 循环
- while 循环
- do...while 循环

JS中的循环

练习

① 求1-100之间所有数的平均值

② 求1-100之间所有偶数和奇数的和

③ 求1-100之间所有能被3整除的数字的和

While循环

while 语句可以在条件表达式为真的前提下，循环执行指定的一段代码，直到表达式不为真时结束循环。

while语句的语法结构如下：

```
while（条件表达式）{
    // 循环体代码
}
```

注意：

① 使用 while 循环时一定要注意，它必须要有退出条件，否则会成为死循环

② while 循环和 for 循环的不同之处在于 while 循环可以做较为复杂的条件判断，比如判断用户名和密码

While循环

课堂案例 1：

① 打印人的一生，从1岁到100岁

② 计算 1 ~ 100 之间所有整数的和

break

break 关键字用于立即**跳出整个循环**（循环结束）。

例如，吃5个包子，吃到第3个发现里面有半个虫子，其余的不吃了，其代码实现如下：

```
for (var i = 1; i <= 5; i++) {
    if (i == 3) {
        break; // 直接退出整个for 循环，跳到整个for下面的语句
    }
    console.log('我正在吃第' + i + '个包子呢');
}
```

continue

continue 关键字用于立即**跳出本次循环**，**继续下一次循环**（本次循环体中 continue 之后的代码就会少执行一次）。

例如，吃5个包子，第3个有虫子，就扔掉第3个，继续吃第4个第5个包子，其代码实现如下：

```
for (var i = 1; i <= 5; i++) {
    if (i == 3) {
        console.log('这个包子有虫子，扔掉');
        continue; // 跳出本次循环，跳出的是第3次循环
    }
    console.log('我正在吃第' + i + '个包子呢');
}
```

Function 函数

```
// 声明函数
function 函数名() {
  //函数体代码
}
```

- function 是声明函数的关键字,必须小写
- 由于函数一般是为了实现某个功能才定义的， 所以通常我们将函数名命名为动词，比如 getSum

形参和实参

在声明函数时，可以在函数名称后面的小括号中添加一些参数，这些参数被称为**形参**，而在调用该函数时，同样也需要传递相应的参数，这些参数被称为**实参**。

```
// 带参数的函数声明
function 函数名(形参1, 形参2 , 形参3...) { // 可以定义任意多的参数，用逗号分隔
    // 函数体
}
// 带参数的函数调用
函数名(实参1, 实参2, 实参3...);
```

注意:在JavaScript中， 形参的默认值是undefined。

return 语句

有的时候，我们会希望函数将值返回给调用者，此时通过使用 return 语句就可以实现。

return 语句的语法格式如下：

```
// 声明函数
function 函数名 () {
    ...
    return  需要返回的值;
}
// 调用函数
函数名();    // 此时调用函数就可以得到函数体内return 后面的值
```

- 在使用 return 语句时，函数会停止执行，并返回指定的值
- 如果函数没有 return ，返回的值是 undefined

函数都是有返回值的

1. 如果有return 则返回 return 后面的值
2. 如果没有return 则返回 undefined

break ,continue ,return 的区别

- break：结束当前的循环体（如 for、while）
- continue： 跳出本次循环，继续执行下次循环（如 for、while）
- return： 不仅可以退出循环，还能够返回 return 语句中的值，同时还可以结束当前的函数体内的代码

利用函数求任意一个数组中的最大值

求数组 [5,2,99,101,67,77] 中的最大数值。

function declaration

利用函数关键字 function 自定义函数方式。

```
// 声明定义方式
function fn() {...}
// 调用
fn();
```

- 因为有名字，所以也被称为命名函数
- 调用函数的代码既可以放到声明函数的前面，也可以放在声明函数的后面

function expression

利用函数表达式方式的写法如下：

```
// 这是函数表达式写法，匿名函数后面跟分号结束
var fn = function(){...};
// 调用的方式，函数调用必须写到函数体下面
fn();
```

- 因为函数没有名字，所以也被称为匿名函数
- 这个fn 里面存储的是一个函数
- 函数表达式方式原理跟声明变量方式是一致的
- 函数调用的代码必须写到函数体后面

Object对象

```
var star = {
    name : 'pink',
    age : 18,
    sex : '男',
    sayHi : function(){
        alert('大家好啊~');
    }
};
```

对象的调用

- 对象里面的属性调用：对象.属性名，这个小点 . 就理解为“ 的 ”
- 对象里面属性的另一种调用方式：对象['属性名']，注意方括号里面的属性必须加引号，我们后面会用
- 对象里面的方法调用：对象.方法名()，注意这个方法名字后面一定加括号

```
console.log(star.name)      // 调用名字属性
console.log(star['name'])   // 调用名字属性
star.sayHi();               // 调用 sayHi 方法,注意，一定不要忘记带后面的括号
```

遍历对象属性

for...in 语句用于对数组或者对象的属性进行循环操作。

其语法如下：

```
for (变量 in 对象名字) {
  // 在此执行代码
}
```

语法中的变量是自定义的，它需要符合命名规范，通常我们会将这个变量写为 k 或者 key。

```
for (var k in obj) {
    console.log(k);       // 这里的 k 是属性名
    console.log(obj[k]); // 这里的 obj[k] 是属性值
}
```

浅拷贝和深拷贝

shallow copy:

Array.from, object.assign,拓展运算符

Deep copy:

lodash,封装函数

JS常用的异步编码方式

- 回调函数----不利于代码阅读和维护，回调地狱
- 事件机制----任务执行不取决于代码定义的顺序，只有当事件发生时才会执行
- 消息订阅与发布 / 全局事件总线
- Promise----通过then的链式调用解决回调地狱的问题。
- async & await----基于promise的语法糖, 简化了promise对象的使用(不再使用回调函数）。是js异步编程的终极解决方案

作业

1 写一个函数，实现反转任意数组。

2 写一个函数，实现对数字数组的排序。