

js简介

js作用

浏览器执行js简介

js三部分组成

JS三种书写位置

js注释

标识符，关键字，保留字

js输入输出语句

js语法结构

变量

数据类型

数字型Number

字符串String

布尔型Boolean

Undefined和Null

typeof检测数据类型

字面量

数据类型转换

运算符

流程控制

分支流程控制if语法结构

三元表达式：

分支流程控制switch语句

语法结构

循环

for循环

断点调试

while循环

do while循环

continue关键字

break关键字

数组(Array)

获取数组元素

数组遍历

数组的长度

数组新增元素

函数

函数的使用

函数的参数

函数的返回值 return

arguments的使用

匿名函数

箭头函数

调用另一个函数

作用域

预解析

对象

创建对象三种方式

遍历对象

内置对象

查阅文档

Math对象

Date对象

- JSON对象
- RegExp对象
- BOM与DOM操作
- BOM操作
 - window子对象
 - history对象
 - location对象(掌握)
 - 弹出框
 - 计时器相关
- DOM操作
 - 查找标签
 - 节点操作
 - 获取值操作
 - class、css操作
 - 原生JS事件绑定

计算机不能直接理解任何除机器语言外的语言，所以必须要把程序员所编写的程序语言翻译成机器语言，才能执行程序，程序语言翻译成机器语言的工具，被成为翻译器。

- 翻译器翻译方式有两种，一个是**编译**，一个是**解释**，两种方式之间的区别在于**翻译的时间点不同**
- 翻译器实在**代码执行之前进行编译**，生成中间代码文件
- 解释器是在**运行时进行及时解释**，并立即执行(当编译器以解释器方式运行的时候,也被称之为解释器)

js简介

1. js一门编程语言，可以写后端代码
2. nodejs支持js代码跑在后端服务器
- JavaScript是脚本语言，解释语言
- javascript是一种轻量级的编程语言
- JavaScript是可插入HTML页面的编程代码
- JavaScript插入HTML页面后，可由所有的现代浏览器执行
- 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，运行过程中由解释器（js引擎）逐行来进行解释并执行
- 可以基于Node.js技术进行服务端编程

js作用

1. 表单验证（密码强度检测）（js产生最初的目的）
2. 网页特效
3. 服务端开发(Node.js)
4. 桌面程序(Electron)
5. App* Cordova)
6. 控制硬件-物联网（Ruff)
7. 游戏开发（cocos2d-js）
- 8.

浏览器执行js简介

浏览器分成两部分**渲染引擎**和**js引擎**

- **渲染引擎**：用来解析HTML和CSS，俗称内核，
- **JS引擎**：也成为JS解释器，用来读取网页中JavaScript代码，对其处理后运行

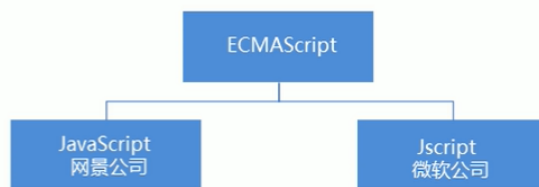
浏览器本身不会执行js代码，而是通过内置的**JavaScript引擎（解释器）**来执行js代码，JS引擎执行完代码逐行解释每一句源码（转化为机器语言），然后通过计算机去执行，所以JavaScript语言归于脚本语言，会逐行解释执行

js三部分组成

- ECMAScript(javascript语法)

1. ECMAScript

ECMAScript是由ECMA国际（原欧洲计算机制造商协会）进行标准化的一门编程语言，这种语言在万维网上应用广泛，它往往被称为JavaScript或JScript，但实际上后两者是ECMAScript语言的实现和扩展。



ECMAScript：ECMAScript规定了JS的编程语法和基础核心知识，是所有浏览器厂商共同遵守的一套JS语法工业标准。

- DOM（页面文档对象模型）

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）。

- BOM（浏览器对象模型）

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三种书写位置

行内，内联，外部

1. 行内式JS

```
1 <body>
2   <input type="button" value="提交" onclick="alert('你好啊')">
3 </body>
```

- 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**中推荐使用**双引号**，**JS**中推荐使用**单引号**
- 可读性差，在HTML中编写大量的代码时，不便于阅读


- 引号易错，引号多层嵌套匹配时，容易弄混

2. 内嵌式JS

```
1 <head>
2   <meta charset="UTF-8">
3   <meta http-equiv="X-UA-Compatible" content="IE=edge">
4   <meta name="viewport" content="width=device-width, initial-scale=1.0">
5   <title>Document</title>
6   <script>
7       alert('你好');
8   </script>
9 </head>
```

3. 外部JS文件

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

- 利用HTML页面代码结构化，可以把大段JS代码独立到HTML页面中
- 引用外部JS文件的script标签中间不可写代 


js注释

```
1 //单行注释
2
3 /*多行注释
4  多行注释
5  */
```

标识符，关键字，保留字

标识符是  开发人员为变量，函数，参数取的名字

标识符不能是关键字或者保留字

关键字是  JS本身已经使用了的字，不能再去看他们当变量名、方法名

包括：break case、catch、continue、default、do、else、finally、for、function、instanceof、new、return、switch、this、throw、try、typeof、var、void、while、with等

保留字就是预留的关键字，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，同样不能使用他们当变量名和方法名

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

js输入输出语句

方法	说明
alert(msg)	浏览器弹出警示框

方法	说明
console.log(msg)	浏览器 控制台打印输出信息 程序员测试用的
prompt(msg)	浏览器弹出输入框，用户可以输入

js语法结构

- 1 | js是以分号作为语句的结束
- 2 | 但是不写分号，也能够正常执行，但是它相当于没有结束符

变量

变量是用于存放数据的容器，可以通过变量名，获取数据，甚至修改数据

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

变量的使用：

1. 声明变量

```
1 | var name;
```

- var是一个JS关键字，用来声明变量 (variable)，使用该关键字声明变量后，计算机会自动为变量分配内存空间，不需要程序员管
- name是程序员自定义的变量名，通过变量名来访问内存中分配的空间

2. 赋值

```
1 | name='zhao';
```

- =号来把右面的值赋给左边的变量空间中，
- 变量值是程序员保存到变量空间里的值

3. 变量的初始化

声明一个变量并赋值，称之为变量的初始化

```
1 | var name ='zhao'
```

```
1 |
2 | js中首次定义变量名 时需要用关键字声明：
3 |     1.var name='json'
4 |     esr推出的新语法：
5 |     2.let name='json'
6 | 如果编辑器支持的 版本是5.1,则无法使用let
7 | 如果是6.0则向下兼容 var let
8 |
9 |
10 | /*var与let区别*/
```

4. 变量语法扩展

1. 修改变量

```
1 var name='zhao';  
2 name='lisi';
```

2. 声明多个变量

同时声明多个变量，只需要写一个var,多个变量名之间用英文逗号隔开

```
1 var name='zhangsan',age=18,gender='male';
```

3. 声明变量特殊情况

```
1 1. 声明不赋值  
2 var sex;  
3 console.log(sex);/*Undefined*/  
4 2. 不声明，不赋值：  
5 console.log(name);/*报错*/  
6 3. 不声明直接赋值  
7 qq=110  
8 console.log(qq);
```

5. 命名规范

- 由字母 (A-Za-z)，数字 (0-9)，下划线 (_)，美元符号(\$)组成
- 严格区分大小写，var app;和 var App;是两个变量
- 不能以数字开头
- 不能是关键字，保留字
- 变量名必须有意义，做到见名知意
- 遵循驼峰命名法，
- 推荐翻译网站：有道，爱词霸

数据类型

在计算机中，不同的数据类型所需要占用的存储空间是不同的，为了便于把数据分成所需要内存大小不同的数据，充分利用存储空间，便定义了不同的数据类型

变量是用来存储值的所在处，他们有名字和数据类型，变量的数据类型决定了如何将代表这些值的位存储到计算机内存中，JavaScript是一种弱类型或者说是动态语言，这意味着不用提前声明变量的类型，在程序中运行过程中，类型会被自动确定

```
1 var age =10; //这是数字型  
2 var sex ='female'; //字符串类型
```

在代码运行时，变量的数据类型是由JS引擎 根据=号右边变量值的数据类型来判断的，运行完毕后，变量就确定了数据类型

JavaScript拥有动态类型，同时也意味着相同的变量可作用不同的类型

数字型Number

```
1 var x=5;    //数字类型
2 var x='hello'; //字符串类型
```

1. 数据类型分类

- 简单数据类型(Number,String,Boolean,Undefined,Null)
- 复杂数据类型 (object)

简单数据类型	说明	默认值
Number	数字型包含整形值和浮点型值	0
Boolean	布尔值类型 true,fales等价于1和0	false
String	字符串类型，带引号	""
Undefined	var a;声明了变量a但是没有赋值，此时a=undefined	undefined
Null	var a =null; 声明了变量a为空值	null

2.数字型

```
1 //八进制 0表示八进制
2 var num1=010;
3 console.log(num1);//控制台结果为8    默认是是十进制，010八进制转为十进制就是8
4 // 十六进制 0x
5 var num2=0xa; //结果为10
6
```

3. 数字类型范围

JavaScript中数的最大值和最小值

```
1 alert(Number.MAX_VALUE); //1.7976931348623157e+308
2 alert(Number.MIN_VALUE); //5e-324
```

特殊值：

```
1 alert(Infinity);
2 alert(-Infinity);
3 alert(NaN);
```

- Infinity:代表无穷大，大于任何数值
- -Infinity:代表无穷小，小于任何值
- NaN (Not a number) ,代表一个非数值

isNaN()方法用来判断非数字，并返回True或者False，如果是数字返回False，非数字返回True

字符串String

字符串可以是引号中的任意文本，语法是双引号和单引号

引号嵌套：JS可以用单引号嵌套双引号，或者双引号嵌套单引号（外双内单，外单内双）

1. 字符串转义符：

转义符都是 `\` 开头的

转义符	解释说明
<code>\n</code>	换行符，n是newline的意思
<code>\\</code>	斜杠\
<code>\'</code>	'单引号
<code>\"</code>	"双引号
<code>\t</code>	tab缩进
<code>\b</code>	空格，b是blank的意思

2. 字符串长度

字符串是由若干个字符组成的，这些字符的数量就是字符串的长度，通过字符串的length属性可以获取整个字符串的长度

```
1 var str='my sdnafaf';
2 console.log(str.length);
```

3. 字符串拼接

- 多个字符串之间可以使用 `+` 进行拼接，其拼接方式：字符串 + 任何类型的数据 = 拼接后的新字符串
- 拼接前会把与字符串相加的任何类型转成字符串，再拼接成一个新的字符串

```
1 console.log('zhao' + 18);
2 var age=18;
3 console.log('kangyi' + age + '岁');
```

- **+ 号口诀：**数值相加，字符相连

```
1 var age = prompt('请输入年龄');
2 var str = '今年'+ age + '岁了';
3 alert(str);
```

- 模板字符串


```

1  # 模板字符串不仅可以定义多行文本还可以实现格式化字符串操作
2  var name = 'zhao'
3  var age = 18
4  var res= `
5      my name is ${name} my age is ${age}
6  `
7  res

```

方法	说明
.length	返回长度
.trim()	移除空白
.trimLeft()	移除左边空白
.trimRight()	移除右边空白
.charAt(n)	返回第n个字符
.concat(value,.....)	拼接
.indexOf(substring,start)	子序列位置
.substring(from,to)	根据索引获取子序列
.slice(start,end)	切片
.toLowerCase()	小写
.toUpperCase()	大写
.split(dilimiter,limit)	分割

```

1  var name = 'zhaoDSB'
2  undefined
3  name.length
4  7
5  var name1 = '  zhaoDSB  '
6  undefined
7  name1.trim()    //不能加括号指定去除的内容
8  'zhaoDSB'
9  name1.trimLeft()
10 'zhaoDSB  '
11 name1.trimRight()
12 '  zhaoDSB'
13 var name2 = '$$jason$$'
14 undefined
15 name.charAt(3)
16 'o'
17 name.indexOf('a')
18 2
19 name.substring(0,4)
20 'zhao'
21 name.slice(0,5)
22 'zhaoD'

```

```

23 name.slice(0,4)
24 'zhao'
25 name.substring(0,-1) //不识别负数
26 ''
27 name.slice(0,-1); //推荐使用slice
28 'zhaods'
29 var demo = 'dlaDLjLLJUB666dser'
30 undefined
31 demo.toLowerCase()
32 'dla dljlljub666dser'
33 demo.toUpperCase()
34 'DLADLJLLJUB666DSER'
35 var test = 'tank|zhao|liao|'
36 undefined
37 name.split('|')
38 ['zhaodsB']
39 test.split('|',10) //第二个参数不是限制切割字符的个数，而是获取切割之后元素的个数
40 (4) ['tank', 'zhao', 'liao', '']
41 test.split('|')
42 (4) ['tank', 'zhao', 'liao', '']
43 test.concat(name) //js是弱类型语言，内部会转成相同的数据类型做操作
44 'tank|zhao|liao|zhaodsB'
45
46
47
48 //python代码
49 l=[1,2,3,4]
50 res=(' ').join(l) //直接报错 列表里的数据必须是字符串类型
51 print(res)

```

布尔型Boolean

布尔类型只有true 和 false，其中true表示真（对）， false表示假（错）

```

1  """
2  1.python中布尔值是首字母大写
3      True
4      False
5  2. js中布尔值是全小写
6      true
7      false
8  # 布尔值是false的有：
9      空字符串、0、null、undefined、Nan
10 """

```

Undefined和Null

一个声明后没有赋值，会有一个默认值undefined

一个声明变量给null值，里面存的值为空

typeof检测数据类型

typeof用来获取变量的数据类型

字面量

字面量（literal）是用于表达源代码中一个固定值的表示法，通俗讲，就是字面量如何让表达这个值

- 数字字面量：8, 9, 10
- 字符串字面量：'html','前端'
- 布尔字面量：true false

数据类型转换

使用表单、prompt获取过来的数据默认字符串类型的，此时不能直接简单的进行加减法，而需要转换变量的数据类型，通俗讲，就是把一种数据类型的变量转成另外一种数据类型

1.转为字符串类型

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+'字符串');

2.转为数字型（重点）

方式	说明	案例
parseInt(string)函数	将string类型转换为整数数值型	parseInt('12.3') 取整
parseFloat(string)函数	将string类型转换为整浮点数数值型	parseFloat('12.3')
Number()强制转换函数	将string类型转为数值型	Number('12')
js 隐式转换(- * /)	利用算术运算隐式转换为数值型	'12'-0

```
1 console.log(parseInt('123px')); //截取到123
2 console.log(parseInt('12.2')) //取整
3 console.log(parseFloat('12.002'));
4 console.log(Number('123.15'));
5 console.log(parseFloat("132.21px")); //截取到132.21
6 console.log('12'-0); //先把 '12' 转成整型12
7 console.log('123'-'120');
```

3. 转换为布尔类型

方式	说明	案例
Boolean()函数	其他类型转换为布尔值	Boolean('True');

- 代表空、否定的值会被转换为false，如："、0、NaN、null、undefined

小案例：

1. 要求页面中弹出输入框，输入出生年份后，计算出我们的年龄

```
1 var year=prompt("输入年份: ")
2 var age=2022-year;
3 alert('您的年龄是' + age + '岁');
4
```

2. 计算两个数的值，用户输入第一个值后，弹出第二个框输入第二个值，最后通过弹出的窗口计算两次输入值相加的结果

```
1 var num1 = prompt('请输入第一个数字: ');
2 var num3 = prompt('请输入第二个数字: ');
3 alert(parseInt(num1)+parseInt(num3));
```

3.

```
1 var name=prompt("请输入姓名: ");
2 var age=prompt('请输入年龄: ');
3 var sex=prompt('请输入性别: ');
4 alert('您的姓名是: '+name+'\n'+ '您的年龄是: '+age+'\n'+ '您的性别是: '+sex);
```

运算符

运算符(operator)也被称为**操作符**，用于实现赋值，比较，执行数运算等功能的符号

1. 算术运算符

运算符	描述
+	加
-	减
*	乘
/	除
%	取余

浮点数进行运算会出现精度问题

表达式和返回值

表达式：由数字、运算符、变量等以求和的数的有意义排列方法所得的组合，简单讲就是由数字、运算符、变量等组成的式子

表达式最终都会有一个结果，返回给我们，称为返回值

2. 递增递减运算符

如果需要反复给数字添加变量添加或减去1，可以使用**递增（++）和递减（--）**运算符来完成。

在JavaScript中，递增（++）和递减（--）既可以放在变量前面，也可以放在变量后面，**放在变量后面称为后置递增（递减）运算符，放在变量前面，称为前置递增（递减）运算符**

注意：递增递减必须和变量配合使用

2.1前置递增运算符

>: 先自加，后返回值

```
1 var age=10;
2 ++age; //类似于age =age+1;
3 var kig=10;
4 console.log(++kig + 10); //21
```

2.2后置递增运算符

>: 先返回原值，后自加

```
1 var num= 10;
2 console.log(num++ + 10); //20
```

```
1 var a =10;
2 ++a; //++a=11 a=11
3 var b = ++a + 2; // a = 12 ++a=12
4 console.log(b); //14
5
6 var c = 10;
7 c++; //c+=11 c=11
8 var d = c++ + 2 ; //c+= 11 c=12
9 console.log(d); //13
10
11 var e=10;
12 var f = e++ + ++e; //e++ =10 e=11 ++e=12 e=12
13 console.log(f) //22
```

3. 比较运算符

运算符名称	说明	案例	结果
>	大于	1>2	false
<	小于	1<2	true
>=	大于等于	2>=2	true
<=	小于等于	3<=2	false
==	判断等号(会转型)	37==37	true
!=	不等号	32!=32	false
=== !==	全等，要求值和数据类型都一致	37=== "37"	false

== 会默认转换数据类型,会把字符串转为整型

```
1 console.log(18==18); //true
2 console.log(18=='18'); //true
```

4.逻辑运算符

逻辑运算符是用布尔值运算的运算符，其返回值也是布尔型，
常用于多个条件的判断

逻辑运算符	说明	案例
&&	"逻辑与"，简称"与" and	true && false
	"逻辑或"，简称"或" or	true false
!	"逻辑非"，简称"非" not	! true

逻辑与两边都是真才为true，否则返回false

逻辑或有一个为真就为 true

逻辑非也叫取反符，用来取一个布尔值相反的值，

4.1短路运算（逻辑中断）

原理：当有多个表达式（值），左边表达式值可以确定结果时，右边不继续运算右边表达式的值

逻辑与短路运算：左边返回值为假，右边就不运算，直接返回false；如果左边返回值为真，那么就要看右边是否也为真，右边为真就返回true,右边返回假就返回false

逻辑或短路运算：如果左边返回真，就直接返回true，右边不再运算；如果左边返回假，那么要看右边返回的值，右边返回真就返回true，右边返回值为假就返回false

5.赋值运算符

赋值运算符	说明	案例
=	直接赋值	var name="zhao"
+=、-=	加、减一个数后再赋值	var age=10; age+=5;
=、/=、%=	乘、除、取模后再赋值	var age=2; age=5;

6. 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

流程控制

顺序结构：按代码的先后顺序，依次执行

分支结构：根据不同的条件，执行不同的路径代码，得到不同的结果

循环结构

分支流程控制if语法结构

- 单分支

条件为真，则执行语句，否则什么也不做

```

1  if (表达式){
2      //执行语句
3  }
4

```

- 双分支

条件为真执行语句1，否则执行语句2

```

1  if (表达式){
2      //执行语句1
3      }else{
4      //执行语句2
5  }
6
7
8  var age =prompt("请输入年龄：")
9      if(age>=18){ //比较运算符可以自动转换类型
10         alert('可以进入网吧');
11     }else{
12         alert("未成年不可进入");
13     }
14

```

案例:

判断闰年:能被4整除且不能被100整除的为闰年, 或者能够被400整除的就是闰年

```
1   var year= prompt('请输入年份: ')
2   if(year%4==0 && year%100!=0 || year%400==0){
3       alert('您输入的年份时闰年')
4   }else{
5       alert('您输入的年份不是闰年')
6   }
```

多分支

```
1   if(表达式1){
2       //语句1;
3   }else if(表达式2){
4       //语句2;
5   }else if(表达式3){
6       //语句3;
7   }else{
8       //最终的语句;
9   }
10
```

案例:

要求: 接受用户输入的分, 根据分数输出对应的字母ABCDE

其中:

1. 90分以上 (包含90) 输出: A
2. 80 (含) --90 (不含) 输出: B
3. 70 (含) --80 (不含) 输出: C
4. 60 (含) --70 (不含) 输出: D
5. 60分以下 (不含) 输出: E

```
1   var grade = prompt('请输入成绩: ')
2   if(grade >=90){
3       alert('A');
4   }else if(grade >=80){
5       alert('B');
6   }else if(grade >=70){
7       alert('C');
8   }else if(grade>=60){
9       alert('D');
10  }else{
11      alert('E')
12  }
13
```


三元表达式：

由三元运算符组成的式子

语法结构： 条件表达式 ? 表达式1 : 表达式2

执行思路： 如果条件表达式为真，则返回表达式1，如果表达式结果为假，则返回表达式2

```
1 var age=10;
2 var result = age>12? 'No' : 'Yes';
3 console.log(result);
```

案例：用户输入0-59的数字，若数字小于10，则前面补0.比如01，09，如果数字大于10，则不需要补，比如20

```
1 var time=prompt('请输入0-59的一个数字')
2 var result= time<10 ? '0' + time : time;
3 console.log(result)
```

分支流程控制switch语句

switch也是多分支语句，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，可以用switch

语法结构

```
1 switch(表达式){
2     case value1:
3         执行语句1;
4         break;
5     case value2:
6         执行语句2;
7         break;
8
9     .....
10    default:
11        执行最后的语句;
12 }
```

执行思路：利用表达式的值和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，如果没匹配上，那么执行default里面的语句

```
1 switch (2){
2     case 1:
3         console.log('这是1');break;
4     case 2:
5         console.log('这是2');break;
6     case 3:
7         console.log('这是3');break;
8     default:
9         console.log('没有匹配到结果')
10 }
```

案例：

用户在弹出的 输入框输入一个水果，如果有就弹出该水果的价格，如果没有就弹出 '没有此水果'

```
1 var fruit = prompt('请输入一个水果名字: ');
2 switch( fruit){
3     case '橘子':
4         alert('3.5/斤');
5         break;
6     case '苹果':
7         alert('2.8/斤');
8         break;
9     case '榴莲':
10        alert('10/斤');
11        break;
12    case '桃子':
13        alert('5.1/斤');
14        break;
15    default:
16        alert('没有'+fruit +'这个水果');
17 }
```

- **switch和if else if 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。

循环

- 许多有规律性的重复操作
- 在程序中，一组被重复执行的语句被称为循环体，能否继续重复执行，取决于循环的终止条件，由循环体及循环的终止条件组成的语句，被称之为循环语句

for循环

把代码重复若干次，通常跟计数有关。

```
1 for(初始化变量;条件表达式;操作表达式){
2     //循环体
3 }
```

初始化变量：就是用var 声明的一个普通变量，通常用于作为计数器使用,整个循环里只执行一次

条件表达式：就是用来决定每一次循环是否继续执行，就是**终止条件**

操作表达式：就是每次循环**最后的执行代码**，经常用于我们计数器变量**进行的递增或递减**

```
1 for(let i =1;i<=100;i++){
2     console.log(i);
3 }
```

执行过程：

1. 先执行计数器变量，var i =1 ,但是这句话在for循环里只执行一次，.
2. 然后去i<=100里判断是否满足条件，如果满足，就去执行循环体，不满足就退出循环
3. 最后去执行 i++ i++单独写的代码，递增， 第一轮结束
第二轮：
4. 接着去执行i<=100,如果满足，就去执行循环体，不满足就退出循环
5. 继续去执行 i++
6.

案例：

求1-100之间所有整数累加和

```
1 var sum=0;
2 for(var i =1;i<=100;i++){
3     sum+=i;
4 }
5 alert(sum);
```

求1-100平均值

```
1 var sum=0;
2 var average=0;
3 for (var i =1; i<=100;i++){
4     sum+=i;
5 }
6 average=sum/100;
7 alert(average);
```

1-100所有整数偶数的和，奇数的和

```
1 var sum_a=0;
2 var sum_a=0;
3 var sum_b=0;
4 for(var i =1;i<=100;i++){
5     if(i%2==0){
6         sum_a+=i;
7     }else{
8         sum_b+=i;
9     }
10 }
11 alert('偶数和: '+sum_a);
12 alert('奇数和: '+sum_b);
```

求1-100之间所有能被3整除的数字之和

```

1  var sum=0;
2  for(var i =1;i<=100;i++){
3      if(i%3==0){
4          sum+=i;
5      }
6  }
7  alert('被3整除的和'+sum)
8

```

用户输入班级人数，之后依次输入每个学生的成绩，最后打印出该班级总的成绩以及平均分

```

1  var sum=0;
2
3  var numbers=prompt('班级总人数: ')
4  for (var i =1;i<=numbers;i++){
5      var stu =prompt('第'+i+'个学生成绩为');
6      var score=parseFloat(stu);
7      sum+=score;
8  }
9  alert('总成绩为: '+sum);
10 alert('平均成绩为: '+ sum/numbers);

```

打印五行五列♥

```

1  var str='';
2  for(var i=1;i<=5;i++){
3      for(var j=1;j<=5;j++){
4          str=str+'♥';
5
6      }
7      str+='\n'
8  }
9  console.log(str);

```

打印倒三角

```

1  var str='';
2  for(var i=1;i<=10;i++){
3      for(var j=10;j>=i;j--){
4          str=str+'★';
5      }
6      str=str+'\n';
7  }
8  console.log(str);

```

```

1  var str='';
2  for(var i=1;i<=10;i++){
3      for(var j=i;j<=10;j++){
4          str+='★';
5      }
6      str+='\n';
7  }
8  console.log(str);

```

打印正三角形

```

1  var str='';
2  for(var i=1;i<=10;i++){
3      for(var j=10;j<=i;j--){
4          str+='★';
5      }
6      str+='\n';
7  }
8  console.log(str);

```

九九乘法表

```

1  var str='';
2  for(var i =1;i<10;i++){
3      for(var j=1;j<=i;j++){
4
5          str+=j + 'x' + i + '=' + i*j;
6          str+='\t';
7
8      }
9      str+='\n';
10 }
11 console.log(str);

```

断点调试

断点调试：

断点调试是指自己在程序的某一行设置一个断点，调试时，程序运行到这一行就会停住，然后你可以一步一步往下调试，调试过程中可以看各个变量当前的值，出错的话，调试到出错的代码行即显示错误，停下。

断点调试可以帮助我们观察程序的运行过程

浏览器中按 F12 --> sources --> 找到需要调试的文件 --> 在程序的某一行设置断点

Watch: 监视，通过 watch 可以监视变量的值的变化，非常的常用。

F11: 程序单步执行，让程序一行一行的执行，这个时候，观察 watch 中变量的值的变化。

代码调试的能力非常重要，只有学会了代码调试，才能学会自己解决 bug 的能力。初学者不要觉得调试代码麻烦就不去调试，知识点花点功夫肯定学的会，但是代码调试这个东西，自己不去练，永远都学不会。

while循环

当...的时候

- 执行思路：当条件表达式结果为true，则执行循环体，否则退出
- 里面也应该有计数器，初始化变量
- 里面应该有操作表达式，完成计数器自增或者自减，防止死循环

```
1 while(条件表达式){  
2     //循环体  
3 }
```

```
1 var i=1;  
2 while(i<=100){  
3     //循环体  
4     console.log(i);  
5     i++;  
6 }
```

1-100和

```
1 var sum=0;  
2 var j=1;  
3 while(j<=100){  
4     sum+=j;  
5     j++;  
6 }  
7 console.log(sum);
```

弹出一个提示框：你爱我吗？，如果输入我爱你，就提示结束，否则一直提示

```
1 var message=prompt('你爱我吗?');  
2 while(message !== '我爱你'){  
3     message=prompt('你爱我吗? ');  
4 }  
5 alert('我也爱你!❤');
```

do while循环

```
1 do{  
2     //循环体  
3 }while (条件表达式)
```

执行思路：跟while不同的是do while先执行一次循环体，在判断条件，如果条件表达式结果为真，则继续执行循环体，否则退出循环体

do while至少执行一次循环体代码

```
1 do{  
2     console.log('heool are you?');  
3     i++;  
4 }while(i<=100)
```

```

1  var i=1;
2  do{
3      console.log('今年'+i+'岁了');
4      i++;
5  }while(i<=100)

```

```

1  do{
2      var mes=prompt('你爱我吗? ');
3  }while(mes!='我爱你')
4  alert('我也爱你♥')

```

continue关键字

用于立即跳出本次循环，继续下一次循环，

```

1  for(var i=1;i<=5;i++){
2      if(i==3){
3          continue;
4      }
5      console.log(i);
6  }

```

1-100之间，除了能被7整除的数之和

```

1  var sum=0;
2  for(var i=1;i<=100;i++){
3      if(i%7==0){
4          continue;
5      }
6      sum+=i;
7  }
8  console.log(sum);

```

break关键字

立即跳出整个循环，循环结束

数组(Array)

数组：一组数据的集合，存储在单个变量下的优雅方式

- 数组里的数据用逗号隔开，
- 数组里的数据称为数组元素
- 数组中可以存放任意类型的数据

利用new创建数组

```

1  var arr = new Array() //创建另一个空数组

```

利用数字字面量创建数组

```

1  var 数组名 = []; //创建空数组
2  var 数组名 = ['小白', '小黑', '蜘蛛']; //创建带初始值的数组

```

获取数组元素

1. 数组的索引

索引（下标）：用来访问数组元素的序号（数组下标从0开始）

可以通过**索引**来访问设置修改对应的数组元素，可以通过 **数组名[索引]** 的形式获取数组中的元素

```
1 var arr=[1,2,15,421,3130];
2 alert(arr[1]); //获取第二个元素
```

数组遍历

遍历：就是把数组中每个元素从头到尾都访问一遍

```
1 var arr = [1,2,15,45,'zhangsan'];
2 for (var i=0;i<5;i++){
3     console.log(arr[i]);
4 }
```

数组的长度

使用 **数组名.length** 可以访问数组元素的个数（数组长度）

```
1 var arr = [1,2,15,45,'zhangsan'];
2 for (var i=0;i<arr.length;i++){
3     console.log(arr[i]);
4 }
```

案例：求数组[2.6,1.7,4]里面所有元素的和，以及平均值

```
1 var sum=0;
2 var arr = [2.6,1.7,4];
3 for (var i=0;i<arr.length;i++){
4     sum+=arr[i];
5 }
6 console.log(sum);
7 console.log(sum/arr.length);
```

数组中最大值

```
1 var arr =[20,2,54,5461,121,1,3,13,456,4,970];
2 var max=arr[0];
3 for (var i=1;i<arr.length;i++){
4     if (arr[i]>max){
5         max=arr[i];
6     }
7 }
8 console.log(max);
```

数组转换为分割字符串


```

1 var arr=['zhao','lisi','yiming','haihong','xuan'];
2 var str='';
3 for (var i=0;i<arr.length;i++){
4     str+=arr[i];
5     str+='|';
6 }
7 console.log(str);

```

数组新增元素

1. 通过修改length长度新增数组元素

- 可以通过修改length长度来实现数组扩容的目的
- length属性是可读写的

```

1 var arr = ['refd','a','f'];
2 arr.length=7;
3 console.log(arr);
4 console.log(arr[4]);

```

其余索引号3, 4, 5, 6的空间没有给值, 就是声明变量为赋值, 默认值就是undefined

2. 通过修改数组索引新增数组元素

- 可以通过修改数组索引的方式追加数组元素

```

1 var arr=['red',1,2];
2 arr[3]='hello';
3 console.log(arr);
4 arr[4]='world';
5 console.log(arr.length);
6 arr[0]='black'; //替换原来的数组元素
7 console.log(arr);

```

不要给数组名直接赋值, 否则会替换掉数组里的所有元素

```

1 var arr=[1,2,15];
2 arr='zhao'; //替换数组所有的元素
3 console.log(arr);

```

新建一个数组, 里面存放10个整数 (1-10)

```

1 var arr=[];
2 for (var i=0;i<10;i++){
3     arr[i]=i+1;
4 }
5 console.log(arr);

```

要求把数组中大于等于10的元素筛选出来, 放入新的数组

```

1  var arr=[2,12,1210,31,45,12,10,200,5];
2  var j=0;
3  var newArr=[];
4  for (var i=0;i<arr.length;i++){
5      if (arr[i]>=10){
6          //新数组下标应该从0开始，。依次递增
7          newArr[j]=arr[i];
8          j++;
9      }
10 }
11 console.log(newArr);

```

```

1  var arr=[2,12,1210,31,45,12,10,200,5];
2
3  var newArr=[];
4  for (var i=0;i<arr.length;i++){
5      if (arr[i]>=10){
6          //新数组下标应该从0开始，。依次递增
7          newArr[newArr.length]=arr[i];
8
9      }
10 }
11 console.log(newArr);

```

删除指定数组元素

将指定数组中的0去掉后，形成一个不包含0的新数组

```

1      var arr = [2, 0, 3, 45, 461, 2, 3, 0, 0, 25, 7];
2      var newArray = [];
3      for (var i = 0; i < arr.length; i++) {
4          if (arr[i] != 0) {
5              newArray[newArray.length] = arr[i];
6          }
7      }

```

反转数组，内容都反过来存放

```

1      var arr=[1,32,121,0,1214,'dog'];
2      var newArray=[];
3      for (var i= arr.length-1; i>=0;i--){
4          newArray[newArray.length]=arr[i]
5      }
6      console.log(newArray);
7

```

冒泡排序：是一种算法，把一系列的数据按照一定的顺序进行**两两比较**排列显示（从小到大或从大到小）

```

1  var arr = [5, 4, 3, 2, 1];
2      for (var i = 0; i <= arr.length - 1; i++) { //外层循环趟数
3          for (var j = 0; j < arr.length - i - 1; j++) { //里层循环管每趟交换
的      次数
4              if (arr[j] > arr[j + 1]) {
5                  var temp = arr[j];
6                  arr[j] = arr[j + 1];
7                  arr[j + 1] = temp;
8              }
9          }
10     }
11
12 }

```

```

1  var l =[1,2,3,4,5,6,7,8,10]
2  undefined
3  l.length
4  9
5  l.push(9) //尾部添加元素
6  10
7  l.pop() //删除最后一个值
8  9
9  l.unshift(123) //头部插入元素
10 10
11 1
12 (10) [123, 1, 2, 3, 4, 5, 6, 7, 8, 10]
13 l.shift() //头部移除元素
14 123
15 1
16 (9) [1, 2, 3, 4, 5, 6, 7, 8, 10]
17 l.slice(0,4) //切片
18 (4) [1, 2, 3, 4]
19 l.reverse() //反转
20 (9) [10, 8, 7, 6, 5, 4, 3, 2, 1]
21 l.join('&') //拼接, 跟python正好相反
22 '10&8&7&6&5&4&3&2&1'
23 1
24 (9) [10, 8, 7, 6, 5, 4, 3, 2, 1]
25 var j=l.join('^')
26 undefined
27 j
28 '10^8^7^6^5^4^3^2^1'
29 l.concat([12121,4545,4551,511,21313,]) //连接数组, 跟python里的extend相似
30 (14) [10, 8, 7, 6, 5, 4, 3, 2, 1, 12121, 4545, 4551, 511, 21313]
31 1
32 (9) [10, 8, 7, 6, 5, 4, 3, 2, 1]
33 l.sort() //排序
34 (9) [1, 10, 2, 3, 4, 5, 6, 7, 8]

```

```

1  var ll=[111,222,333,444,555]
2  undefined
3  ll.forEach(function(value){console.log (value)},ll)
4  VM933:1 111 //一个参数就是数组里面每一个元素对象
5  VM933:1 222

```

```

6 VM933:1 333
7 VM933:1 444
8 VM933:1 555
9 undefined
10 ll.forEach(function(value,index){console.log(value,index)},ll)
11 VM1084:1 111 0 //两个参数就是 元素+元素索引
12 VM1084:1 222 1
13 VM1084:1 333 2
14 VM1084:1 444 3
15 VM1084:1 555 4
16 undefined
17 ll.forEach(function(value,index,arr){console.log(value,index,arr)},ll) //三个参数就是 元素+元素索引+元素的数据来源
18 VM1189:1 111 0 (5) [111, 222, 333, 444, 555]
19 VM1189:1 222 1 (5) [111, 222, 333, 444, 555]
20 VM1189:1 333 2 (5) [111, 222, 333, 444, 555]
21 VM1189:1 444 3 (5) [111, 222, 333, 444, 555]
22 VM1189:1 555 4 (5) [111, 222, 333, 444, 555]

```

```

1 ll.splice(0,3) //两个参数，第一个是起始位置，第二个参数是删除的个数
2 (3) [111, 222, 333]
3 ll
4 (2) [444, 555]
5 ll.splice(0,1,777) //先删除，后增加
6 [444]
7 ll
8 (2) [777, 555]
9 ll.splice(0,1,[111,222,333,444])
10 [777]
11 ll
12 (2) [Array(4), 555]0: (4) [111, 222, 333, 444]1: 555length: 2[[Prototype]]: Array(0)

```

```

1 var ll =[11,22,33,44,55,66]
2 undefined
3 ll
4 (6) [11, 22, 33, 44, 55, 66]
5 ll.map(function(value){console.log(value)},ll)
6 VM1785:1 11
7 VM1785:1 22
8 VM1785:1 33
9 VM1785:1 44
10 VM1785:1 55
11 VM1785:1 66
12 (6) [undefined, undefined, undefined, undefined, undefined, undefined]
13 ll.map(function(value,indexedDB){return value*2},ll)
14 (6) [22, 44, 66, 88, 110, 132]
15 ll.map(function(value,indexed,arr){return value*2},ll)
16 (6) [22, 44, 66, 88, 110, 132]

```

函数

函数就是一段被重复执行调用的代码块。

函数的使用

分两步：声明函数，调用函数

1. 声明函数

```
1 function myfunc(){
2     //函数体
3 }
```

- **function**是声明函数的关键字
- 函数表达式(匿名函数)

```
1 var 变量名=function(){
2     console.log('我是匿名函数');
3 }
```

2. 调用函数

```
1 函数名() //通过调用函数名来执行函数体代码
```

- 调用函数的时候**不要忘了加小括号**

注意：声明函数本省不会执行代码。只有调用的时候才会执行函数体的代码

3. 函数的封装

- 函数的封装就是把一个或者多个功能通过**函数的方式封装起来**，对外只提供一个简单的的函数名

```
1 //函数声明
2 function getSum(){
3     var sum=0;
4     for(var i=0;i<=100;i++){
5         sum+=i;
6     }
7     console.log(sum);
8 }
9 //调用函数
10 getSum();
```

函数的参数

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，这些参数称为形参，而在调用函数时，同样也需要传递响应的参数，被称为实参。

```
1 function 函数名(形参1,形参2){
2
3 }
4 函数名(实参1,实参2);
```

- 形参是接收实参的
- 函数的参数可以有，可以没有

```
1 function getSum(num1,num2){
2     console.log(num1+num2);
3 }
4 getSum(1,2);
```

函数的返回值 return

```
1 function 函数名(){
2     return 需要返回的结果;
3 }
4 }
```

- 遇到return，就把后面的结果，返回给函数的调用者

```
1 function getsum(){
2
3     return 6696;
4 }
5 getsum();
```

- return之后的代码就不会被执行
- return只能返回一个值，如果用逗号隔开，以最后一个为准
- 如果有return则返回return后面的值
- 没有return则返回undefined
- 想返回多个可以写成数组的形式

```
1 function index(){
2     return [666,444,888];
3 }
```

arguments的使用

当我们不知道有多少个参数传递的时候，可以调用arguments来获取

JavaScript中，arguments实际上是当前函数的一个内置对象，所有函数都内置了一个arguments对象，arguments对象中存储了传递的所有实参

```
1 function getSum(){
2     console.log(arguments); //里面存储了所有传递过来的实参
3     console.log(arguments.length);
4     console.log(arguments[1]);
5 }
6 getSum(1,2);
```

arguments展示形式是一个伪数组，因此它可以遍历，伪数组具有以下特点：

- 具有length属性
- 按索引方式存储数据
- 不具有数组的push pop等方法

案例：利用函数求任意个数的最大值

```
1 function getMax(){
2     var max=arguments[0];
3     for (var i=1;i<arguments.length;i++){
4         if (arguments[i]>max){
5             max=arguments[i];
6         }
7     }
8     return max;
9 }
10 console.log(getMax(1,2,3));
11 console.log(getMax(1,2,3,4564,456411));
12 console.log(getMax(101215,15141,11));
```

匿名函数

没有名字的函数

```
1 var res=function(){
2     console.log('啊哈哈')
3 }
```

箭头函数

```
1 var func1 =v => v;//箭头左边的是形参，右边的是返回值
2 等价于
3 var func1 =function(v){
4     return v
5 }
6
7
8 var func2 =(arg1,arg2) => arg1+arg2
9 等价于
10 var func1 =function(arg1,arg2){
11     return arg1+arg2
12 }
```

调用另一个函数

```
1 function fn1(){
2     console.log(11);
3     fn2();
4 }
5 function fn2(){
6     console.log(22);
7 }
```

用户输入年份，输出当前年份2月份的天数

如果时间闰年，2月份时29天，如果是平年，则2月份是28天

```
1 function backDay() {
```

```

2      var year = prompt('请输入年份: ');
3      if (isRunyear(year)) {
4          alert('当前年份是闰年, 2月份有29天');
5      } else {
6          alert('当前年份是闰年, 2月份有29天');
7      }
8  }
9
10
11     function isRunyear() {
12         var flag = false;
13         if (year % 4 == 0 && year % 100 != 0 || year % 400 == 0) {
14             flage = true;
15         }
16         return flag;
17     }

```

作用域

跟Python一样

就是变量在某个范围内起作用，目的是为了程序的可读性，最重要的是减少冲突

局部作用域：在函数内部就是局部作用域，只在函数内部起效果

全局作用域：整个script标签或者是一个单独的js文件

变量的作用域：根据作用域的不同，变量分为全局变量和局部变量

在全局作用域下声明的变量叫**全局变量（在函数外部定义的变量）**

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）**

- 全局变量只有浏览器关闭才会销毁，比较占内存资源
- 局部变量当程序执行完就会销毁，比较省资源
- 局部变量只能在函数**内部**使用
- 全局变量在代码**任何位置**都可以使用
- 函数的形参实际上也是局部变量

作用域链

- 只要是代码，就至少有一个作用域
- 写在函数内部的局部作用域
- 如果函数中还有函数，那么在这个作用域中就可以诞生一个作用域
- 根据在内部函数可以访问外部函数变量的这种机制，用链式查找决定哪些数据能被内部函数访问，就称为作用域链

```

1  var num=10;
2  function fun(){
3      var num=20;
4      function fn(){
5          console.log(num); //20
6      }
7      fn();
8  }
9  fun();

```



```
1  var city='Beijing';
2  function Bar(){
3      console.log(city);
4  }
5  function f(){
6      var city='ShanHai';
7      return Bar;
8  }
9  var res=f();
10 res();    //Beijing
```

```
1  //闭包
2  var city="Beijing";
3  function f(){
4      var city="ShanHai";
5      function inner(){
6          console.log(city);
7      }
8      return inner;
9  }
10 var res=f();
11 res();
```

预解析

JavaScript代码是由浏览器中的JavaScript解析器来执行的，JavaScript解析器在运行JavaScript代码的时候分两步：**预解析**和**代码执行**

按照代码书写的顺序从上往下执行

预解析会把所有的var 还有function提升到当前作用域的最前面

预解析分为**变量预解析**（变量提升）和**函数预解析**（函数提升）

- 变量提升就是把所有的变量声明提升到当前的作用域最前面，不提升赋值操作

```

console.log(num); // undefined
var num = 10;
// 相当于执行了以下代码
// var num;
// console.log(num);
// num = 10;

fun(); // 报错 坑2
var fun = function() {
    console.log(22);
}

// 相当于执行了以下代码
var fun;
fun();
fun = function() {
    console.log(22);
}

```

- 函数提升就是把所有的函数声明提升到当前作用域的最前面，不调用函数

```

1  fn();
2  function fn(){
3      console.log(11);
4  }
5
6      /*相当于
7      function fn(){
8          console.log(11);
9      }
10     fn()
11     */

```

预解析案例：

```

1  var num=10;
2  fun();
3  function fun(){
4      console.log(num);
5      var num=20;
6  }
7
8  //相当于执行了以下操作
9  var num;
10 function fun(){
11     var num;
12     console.log(num); //undefined
13     num=20;

```

```
14 }
15 num=10;
16 fun();
```

```
1 var num = 10;
2 function fn(){
3     console.log=(num);
4     var num=20;
5     console.log(num);
6 }
7 fn();
8 //相当于执行以下代码
9 var num;
10 function fn(){
11     var num;
12     console.log=(num); //undefined
13     num=20;
14     console.log(num); //20
15 }
16 num=10;
17 fn();
```

```
1 var a =18;
2 f1();
3 function f1(){
4     var b=9;
5     console.log(a);
6     console.log(b);
7     var a='123';
8 }
9 //相当于以下代码
10 var a;
11 function f1(){
12     var b;
13     var a;
14     b=9;
15     console.log(a); //undefined
16     console.log(b); //9
17
18     a='123';
19 }
20 a=18;
21 f1();
```

```
1 f1();
2 console.log(c);
3 console.log(b);
4 console.log(a);
5 function f1(){
6     var a=b=c=9;
7     console.log(a);
8     console.log(b);
9     console.log(c);
```

```

10 }
11 //相当于以下代码
12 function f1(){
13     var a;
14     a=b=c9;
15
16
17     console.log(a); //9
18     console.log(b); //9
19     console.log(c); //9
20 }
21 f1();
22 console.log(c); //9
23 console.log(b); //9
24 console.log(a); //报错

```

对象

万物皆对象，对象是一个具体的事务，看的见摸得到的实物，例如：一本书、一辆车、一个人，可以是对象，一个数据库，一张网页，一个与远程服务器的连接也可以是对象

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
- 方法：事物的**行为**，在对象中用**方法**来表示

保存一个值时，可以使用**变量**，保存多个值（一组值）时，可以使用**数组**，但是要保存一个人的完整信息就需要用到**对象**

创建对象三种方式

1. 利用字面量创建对象

对象字面量：就是花括号{}里面包含了表达这个具体事物（对象）的属性和方法

```

1 var obj={
2     name:'张三',
3     age:18,
4     sex:'男',
5     myfun:function(){
6         console.log('hello');
7     }
8 }

```

- 里面的属性或者方法，我们可以采取键值对的形式，键 属性名：值 属性值
- 多个属性或者方法中间用逗号隔开
- 调用对象的属性，采用**对象名.属性名 (对象的属性名)**
- 也可以采用**对象['属性名']**来调用

```

1 console.log(obj.name);
2 console.log(obj['name']);

```

- 调用对象的方法 **对象名.方法名()**

```
1 | obj.myfun();
```

2. 利用new Object()创建对象

```
1 | var obj=new Object() //创建了一个空对象
2 | //添加属性
3 | obj.name='李四';
4 | obj.gender='女';
5 | //添加方法
6 | obj.myfun=function(){
7 |     console.log('zhao');
8 | }
9 | //调用
10 | console.log(obj.name);
11 | console.log(obj.gender);
12 | console.log(obj.myfun);
```

3. 利用构造函数来创建对象

构造函数：是一种特殊的函数，主要是用来初始化对象，即为对象成员变量赋初始值，它总与new运算符一起使用，我们可以把对象中一些公共属性和方法抽取出来，封装到这个函数里面

```
1 |     function 构造函数名(){
2 |         this.属性=值;
3 |         this.方法=function(){}
4 |     }
5 |     new 构造函数名();
```

```
1 |     function Star(name, age, sex) {
2 |         this.name = name;
3 |         this.age = age;
4 |         this.sex = sex;
5 |     }
6 |     var result=new Star('黎明', 45, '男');//调用函数返回的是一个对象
7 |     // console.log(typeof result);
8 |     console.log(result.name);
9 |     console.log(result.sex);
10 |     var result1=new Star('张学友',50,'男');
11 |     console.log(result1.name);
12 |     console.log(result1.age);
```

- 构造函数首字母要大写
- 构造函数不需要return，就可以返回结果
- 调用构造函数必须使用new
- 属性和方法前面必须添加this
- 通过new关键字创建对象的过程称为**对象实例化**

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. 执行构造里面的的代码，给空对象添加属性和方法
4. 返回这个对象，（所以构造函数不需要return）

遍历对象

for..in语句用于对数组或者对象的属性和方法进行循环操作

```
1 for (变量 in 对象){
2
3 }
```

```
1 for (var key in obj){
2     console.log(key); //输出属性名
3     console.log(obj[key]); //输出属性值
4 }
```

内置对象

- JavaScript中对象分为3种：自定义对象、内置对象、浏览器对象
- 前两种时JS基础内容，属于ECMAScript ;第三个浏览器对象属于我们JS独有的
- 内置对象就是JS语言自带的一些对象，供开发者使用，并提供了一些常用的或者是最基本功能（属性和方法）
- JavaScript中常用的内置对象：Math、Date、Array、String等

查阅文档

[MDN](#)

Math对象

[Math.max\(\)](#)

- 返回给定的一组数字中的最大值。如果给定的参数中至少有一个参数无法被转换成数字，则会返回 [NaN](#)
- 如果没有参数，则结果为 - [Infinity](#)

```
1 console.log(Math.PI); //圆周率
2 console.log(Math.max(1,88,99)); //最大值
3 console.log(Math.max()); // -infinity  -的无穷大
4 console.log(Math.max(1,2,55,'lisi')); //NaN
```

封装自己的Math对象

```
1 var myMath={
2     PI:3.141592653,
3     max:function(){
4         var max=arguments[0];
5         for (var i =1;i<arguments.length;i++){
6             if(arguments[i]>max){
7                 max=arguments[i];
8             }
9         }
10        return max;
11    },
12    min:function(){
```

```

13         var min=arguments[0];
14         for (var i =1;i<arguments.length;i++){
15             if(arguments[i]<min){
16                 min=arguments[i];
17             }
18         }
19         return min;
20     }
21
22 }
23 console.log(myMath.PI,myMath.max(1,88,99));

```

其他方法

```

1 //绝对值方法
2 console.log(myMath.PI,myMath.max(1,88,99));
3 console.log(Math.abs(-1));
4 console.log(Math.abs('-1'));//隐式转换，会把字符串型-1转成数字型
5 console.log(Math.abs('lisi'));//NaN
6
7 //三个取整方法
8 //1.Math.floor 向下取整，往最小了取整
9 console.log(Math.floor(1.1));//1
10 console.log(Math.floor(1.9));//1
11 //2.Math.ceil 向上取整，往最大了取整
12 console.log(Math.ceil(1.1));//2
13 console.log(Math.ceil(1.9));//2
14 //3.四舍五入 Math.round()
15 console.log(Math.round(1.1));//1
16 console.log(Math.round(1.5));//2
17 console.log(Math.round(1.9));//2
18 console.log(Math.round(-1.1));//-1
19

```

[Math.random\(\)](#)

`Math.random()` 函数返回一个浮点数，伪随机数在范围从**0 到小于 1**，也就是说，从 0（包括 0）往上，但是不包括 1（排除 1）

得到大于等于0.小于1的随机数

```

1 function getRandom() {
2     return Math.random();
3 }

```

Date对象

是一个构造函数，必须使用new来调用创建日期对象

```
Elements Console Sources Network
top Filter Dev
> let d3 = new Date()
< undefined
> d3.getDate
< f getDate() { [native code] }
> d3.getDate()
< 21
> d3.getTime()
< 1661062159070
> d3.getFullYear()
< 2022
> d3.toString()
< 'Sun Aug 21 2022'
> |
```

```
1 # 时间对象具体方法
2 let d6=new Date();
3 d6.getDate()           #获取日期
4 d6.getDay()            #获取星期
5 d6.getMonth()          #获取月份（0-11）
6 d6.getFullYear()       #获取完成年份
7 d6.getHours()          #获取小时
8 d6.getMinutes()       #获取分钟
9 d6.getSeconds()        #获取秒
10 d6.getMilliseconds()   #获取毫秒
11 d6.getTime()           #时间戳
```

JSON对象

```
1 """
2 在Python中序列化反序列化
3 dumps 序列化
4 loads 反序列化
5
6
7 在js中也有序列化和反序列化
8 JSON.stringify()      dumps
9 JSON.parse()          loads
10 """
11 let d1={'name':'zhao','age':18}
12 let res=JSON.stringify(d1)
13 '{"name":"zhao","age":18}'
14 JSON.parse(res)
15 {name: 'zhao', age: 18}
```



```
数据类型.html × JS\encoder.py × json\encoder.py ×
1 # -*- coding: UTF-8 -*-
2 # @Date : 2022/10/13 8:58
3 import json
4 json.JSONEncoder
```

ctrl键按住，点击查看

```
1 Supports the following objects and types by default:
2 +-----+-----+
3 | Python          | JSON          |
4 +=====+=====+
5 | dict            | object        |
6 +-----+-----+
7 | list, tuple     | array         |
8 +-----+-----+
9 | str             | string        |
10 +-----+-----+
11 | int, float      | number        |
12 +-----+-----+
13 | True            | true          |
14 +-----+-----+
15 | False           | false         |
16 +-----+-----+
17 | None            | null          |
18 +-----+-----+
```

RegExp对象

```
1 """
2 python中使用正则，需要借助re模块
3 Js中需要创建正则对象
4 """
5 #第一种，有点麻烦
6 let reg1=new RegExp('^[a-zA-Z][a-zA-Z0-9]{5,11}')
7 #第二种
8 let reg2=/^[a-zA-Z][a-zA-Z0-9]{5,11}/
9
10 #匹配内容
11 reg1.test('lisizhangsan')
12 reg2.test('zhangang')
13 #获取字符串里的字面
14 let ss='zhaosong so sosfaf asn'
15 ss.match(/s/) #拿到第一个停止
16 ['s', index: 4, input: 'zhaosong so sosfaf asn', groups: undefined]
17 ss.match(/s/g) #全局匹配，g就代表全局模式
```

```

18 (5) ['s', 's', 's', 's', 's']
19
20 let reg5 =/undefined/
21 undefined
22 reg5.test('zhao')
23 false
24 reg5.test()
25 true

```

BOM与DOM操作

```

1  """
2  BOM:浏览器对象模型 Brower Object Model
3      js代码操作浏览器
4  DOM: 文档对象模型 Document Object Model
5      js代码操作标签
6  """

```

BOM操作

```

1  #window对象: 指代的的就是浏览器窗口
2  window.innerHeight  浏览器窗口的高度
3  736
4  window.innerWidth  浏览器窗口的宽度
5  970
6
7  #新建窗口打开页面, 第二个参数写空即可, 第三个参数写新建窗口的大小和位置
8  window.open('https://www.csdn.net/', '', 'height=400px,width=400px,
9  top=400px,left=400px')
10
11  window.close()  关闭当前页面
12
13  #父子页面通信window.opener()

```

window子对象

```

1  window.navigator.appName
2  'Netscape'  #网景
3
4  window.navigator.appVersion
5  '5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
6  Chrome/104.0.0.0 Safari/537.36'
7  window.navigator.userAgent  #用来标识当前是否是一个浏览器.
8  'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
9  Gecko) Chrome/106.0.0.0 Safari/537.36'
10  """
11  防爬措施
12  1. 校验当前请求的发起者是否是一个浏览器
13  爬虫: 在代码中加上user-agent配置即可, 就可以UA伪装
14  """
15
16  window.navigator.platform  #平台

```

```
15 'win32'
16
17 # 如果时window的子对象，那么window可以省略不写
18 navigator.appVersion
```

history对象

```
1 window.history.back() #回退到上一页
2 window.history.forward()#前进到下一页
3 #对应的就是浏览器左上方的两个箭头
4
```

location对象(掌握)

```
1 window.location.href #获取当前页面的url
2 window.location.herf='url' #跳转到指定的url
3 window.location.reload() #刷新页面（浏览器左上方的刷新小圆圈）
```

弹出框

- 警告框

```
1 alert('你不要过来啊')
```

- 确认框

```
1 confirm('你确定吗') #点击取消，控制台显示false，点击确定，控制台显示true
```

- 提示框

```
1 prompt('请输入')#控制台显示输入的内容
```

计时器相关

- 过几秒钟后触发 `setTimeout()`
- 每隔一段时间触发一次（循环） `setInterval()`

```
1 <script>
2     function fun1(){
3         alert(123)
4     }
5     let t = setTimeout(fun1,3000) //毫秒为单位，3秒后自动执行fun1函数
6     clearTimeout(t) //取消定时任务，如果要清除定时任务，需要提前用变量指代定时任
   务
7
8
9     function fun2(){
10         alert(123)
11     }
12     function show(){
13         let t = setInterval(fun2,3000) //每隔三秒弹出一
14         function inner(){
```

```

15         clearInterval(t) //清除定时器
16     }
17     setTimeout(inner,9000) //9秒后触发
18 }
19 show()
20
21
22 </script>

```

DOM操作

```

1  DOM树的概念
2  所有的标签都可以称之为节点
3  Javascript可以通过DOM创建动态的HTML
4  Javascript能够改变页面中的所有HTML元素
5  Javascript能够改变页面中的所有HTML属性
6  Javascript能够改变页面中的所有CSS样式
7  Javascript能够改变页面中的所有事件做出反应
8
9
10 DOM操作 操作的是标签，而一个html页面上的标签有很多
11 1. 先学如何查找标签
12 2. 再学DOM操作标签
13
14 DOM需要用关键字document
15 ""

```

查找标签

- 直接查找（掌握）

```

1  ""
2  id查找
3  类查找
4  标签查找
5  ""
6  #下面三个方法的返回值是不一样的
7  document.getElementById('d2')
8  document.getElementsByClassName('c1')
9  document.getElementsByTagName('div')
10
11
12 let divEle =document.getElementsByTagName('div')[0]
13 divEle
14 ""
15 当用变量指代标签对象的时候，一般情况下写成xxxEle
16 ""

```

- 进阶查找（熟悉）

```

1  let pEle=document.getElementsByClassName('c1')[0]
2  undefined
3  pEle.parentElement #拿父节点

```

```

4 <div id="d2">...</div>
5 pEle.parentElement.parentElement
6 <body>...</body>
7 pEle.parentElement.parentElement.parentElement
8 <html lang="en"><head>...</head><body>...</body></html>
9
10
11 let divEle =document.getElementById('d2')
12 undefined
13 divEle.children #获取所有的子标签
14
15 divEle.children[0] //获取第一个子标签
16 <div>div1<div>div</div>
17 divEle.firstChild
18 "div1 "
19 divEle.firstChild //第一个子标签
20 <div>div1<div>div</div>
21 divEle.lastElementChild //最后一个子标签
22 <p>div1<p2</p>
23 divEle.nextElementSibling #同级别下面第一个
24 <div>div1div2</div>
25 divEle.previousElementSibling #同级别上面第一个
26 <div>H人类咯</div>

```

节点操作

```

1 ""
2 1. 通过DOM操作动态的创建img标签
3 2. 并给img加属性
4 3. 最后将标签添加到文本中
5 ""

```

```

1 let imgEle=document.createElement('img') #创建标签
2
3 imgEle.src='../tou.jpg' #给标签设置默认的属性
4 '../tou.jpg'
5
6 imgEle.username='zhao'#自定义的属性没办法以点的方式设置
7 'zhao'
8
9 imgEle.setAttribute('name','zhao')#既可以设置自定义的属性，也可设置默认的属性
10 undefined
11 imgEle
12 
13 imgEle.setAttribute('title','头像')
14 undefined
15 imgEle
16 
17 let divEle= document.getElementById('d1')
18 undefined
19 divEle.appendChild(imgEle)#标签内部添加元素（尾部追加）
20 

```

```

1  /*
2  创建a标签
3  设置属性
4  设置文本
5  添加到标签内部，添加到指定的标签上面
6  */
7  let aEle=document.createElement('a')
8  undefined
9  aEle.href='http:www.baidu.com/'
10 'http:www.baidu.com/'
11 aEle
12 <a href="http:www.baidu.com/"></a>
13 aEle.innerText='别摸我'    #给标签设置文本内容
14 '别摸我'
15 aEle
16 <a href="http:www.baidu.com/">别摸我</a>
17 let divEle=document.getElementById('d1')
18 undefined
19 let pEle=document.getElementById('d2')
20 undefined
21 divEle.insertBefore(aEle,pEle)  #添加标签内容指定位置添加（a标签添加到div内部，p标签之前）
22 <a href="http:www.baidu.com/">别摸我</a>

```

补充

```

1  appendChild()
2  removeChild()
3  replaceChild()
4
5  setAttribute() 设置属性
6  getAttribute() 获取属性
7  removeAttrive() 移除属性
8
9
10 # innerText与innerHTML
11
12 divEle.innerText    #获取到标签内部所有的文本
13 '别摸我\n\ndiv>p\n\ndiv>span'
14 divEle.innerHTML    #内部文本和标签都拿到
15 '\n    <a href="http:www.baidu.com/">别摸我</a><p id="d2">div>p</p>\n
    <span>div>span</span>\n    '
16 divEle.innerText='hahahah'
17 'hahahah'
18 divEle.innerHTML='sb'
19 'sb'
20 divEle.innerText='<h1>二蛋</h1>'  #不识别HTML标签
21 '<h1>二蛋</h1>'
22 divEle.innerHTML='<h1>二蛋</h1>'  #识别HTML标签
23 '<h1>二蛋</h1>'

```

获取值操作

```
1 //获取用户数据集标签内部的数据
2 let selectEle =document.getElementById('d2')
3 selectEle.value
4 '111'
5 selectEle.value
6 '222'
7 selectEle.value
8 '333'
9 let inputEle = document.getElementById('d1')
10 inputEle.value
11 "sxcd"
12
13 //如何上传用户输入的文件
14 let fileEle =document.getElementById('d3')
15 undefined
16
17 fileEle.value //无法获取到文件数据，得到是文件路径
18 'C:\\fakepath\\d160681e16c2782af151165d1202dd45.jpeg'
19
20 fileEle.files[0] //获取文件数据
21 File {name: 'd160681e16c2782af151165d1202dd45.jpeg', lastModified:
1661066246763, lastModifiedDate: Sun Aug 21 2022 15:17:26 GMT+0800 (中国标准时
间), webkitRelativePath: '', size: 13627, ...}
22
23 fileEle.files //返回数组，【文件对象，文件对象.....】
24 FileList {0: File, length: 1}
```

class、css操作

```
1 let divEle =document.getElementById('d1')
2 undefined
3 divEle.classList //获取标签所有的类属性
4 DOMTokenList(3) ['c1', 'bg_green', 'bg_red', value: 'c1 bg_green bg_red']
5 divEle.classList.remove('bg_red') //移除某个类属性
6 undefined
7 divEle.classList.add('bg_red') //添加类属性
8 undefined
9 divEle.classList.contains('c1') //验证是否包含某个类属性
10 true
11 divEle.classList.contains('c2')
12 false
13 divEle.classList.toggle('bg_red')//有则删除，无则添加
14 false
15 divEle.classList.toggle('bg_red')
16 true
17 divEle.classList.toggle('bg_red')
18 false
19 divEle.classList.toggle('bg_red')
20 true
21
22
23 //DOM 操作标签样式，统一先用style
```

```

24
25 let pEle=document.getElementsByTagName('p')[0]
26 undefined
27 pEle.style.color='red'
28 'red'
29 pEle.style.fontSize='20px'
30 '20px'
31 pEle.style.backgroundColor='#eee'
32 '#eee'
33 pEle.style.border='3px solid red'
34 '3px solid red'
35
36 //会将css中横杠或者下划线去掉，然后将后面的单词变成大写
37 //background-color变成backgroundColor

```

原生JS事件绑定

```

1  ""
2  到达某个事先设定的条件，自动触发的动作
3  ""
4  //绑定事件的两种方式
5  <button onclick="fun1()">点击</button>
6  <button id="d1">点击</button>
7
8  <script>
9      //第一种绑定事件方式
10     function fun1(){
11         alert(1231)
12     }
13     //第二种
14     let btnEle =document.getElementById('d1');
15     btnEle.onclick=function(){
16         alert(222)
17     }
18 </script>
19
20 //script标签既可以放在head内，也可以放在body最底部，一般放在body最底部
21
22
23
24
25 //等待浏览器窗口加载完毕再执行代码
26 <script>
27     window.onload=function(){
28         //第一种绑定事件方式
29         function fun1(){
30             alert(1231)
31         }
32         //第二种
33         let btnEle =document.getElementById('d1');
34         btnEle.onclick=function(){
35             alert(222)
36         }
37     }
38 </script>

```


- 开关灯示例

```
1 <style>
2     .c1{
3         width: 400px;
4         height: 400px;
5         border-radius: 50%;
6     }
7     .bg_green{
8         background-color: green;
9     }
10    .bg_red{
11        background-color: red;
12    }
13 </style>
14 </head>
15 <body>
16     <div id="d1" class="c1 bg_green bg_red"></div>
17     <button id="d2">变色</button>
18
19     <script>
20         let btnEle=document.getElementById('d1')
21         let divEle=document.getElementById('d2')
22         btnEle.onclick=function(){ //绑定点击事件
23             //动态的修改类属性
24             divEle.classList.toggle('bg_red')
25         }
26     </script>
```

- input框获取焦点失去焦点案例

```
1 <input type="text" value="您好" id="d1">
2
3     <script>
4         let inEle=document.getElementById('d1');
5         //获取焦点事件
6         inEle.onfocus=function(){
7             //将input框内部的文本去除
8             inEle.value=''
9             // 点value就是获取，等号赋值是设置
10
11         }
12         //失去焦点事件
13         inEle.onblur=function(){
14             //给input标签重新赋值
15             inEle.value ='不好'
16         }
17     </script>
```

- 实时展示当前时间

```
1 <body>
```

```

2      <input type="text" id="d1" style="display: block; height: 40px; width:
200px">
3      <button id="d2">开始</button>
4      <button id="d3">结束</button>
5
6
7
8      <script>
9          //定义一个全局变量
10         let t=null;
11         let inputEle = document.getElementById('d1')
12         let startBtnEle = document.getElementById('d2')
13         let endtBtnEle = document.getElementById('d3')
14         //1.访问页面之后将访问的时间添加到input框中
15         //2.动态展示当前时间
16         //页面上加两个按钮。一个开始，一个结束。
17         function showTime() {
18             let currentTime = new Date();
19             inputEle.value = currentTime.toLocaleString()
20
21         }
22         // showTime()
23         // setInterval(showTime,1000)
24         startBtnEle.onclick = function () {
25             //限制定时器只能开一个
26             if(!t){
27                 t=setInterval(showTime, 1000) //每点击一次就会开设一个定时器，而
t只指代最后一个
28                 }
29             }
30         endtBtnEle.onclick = function () {
31             clearInterval(t)
32             //清楚定时器的時候，因該把t重置为空
33             t=null
34         }
35     </script>
36 </body>

```

- 省市联动

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Document</title>
8  </head>
9  <body>
10     <select name="" id="d1">
11         <option value="" selected disabled>--请选择--</option>
12     </select>
13     <select name="" id="d2"></select>
14
15     <script>

```

```
16     let proEle=document.getElementById('d1')
17     let cityEle=document.getElementById('d2')
18     //模拟省市数据
19     data={'河北省':['衡水','邯郸','唐山'],
20         '北京':['东城区','朝阳区'],
21         '山东':['青岛市','潍坊市'],
22         '上海':['浦东新区','黄浦区'],
23         '深圳':['南山区','宝安区','福田区']}
24 };
25 //for循环获取省
26 for (let key in data){
27     //将省信息做成一个option标签，添加到select框中
28     //1.创建option标签
29     let opEle=document.createElement('option')
30     //2.设置文本
31     opEle.innerText=key
32     //设置value值
33     opEle.value=key
34     //将创建好的option添加到select中
35     proEle.appendChild(opEle)
36 }
37 //文本域变化事件（change事件）
38 proEle.onchange=function(){
39     //获取到用户选择的省
40     let currentPro=proEle.value
41     //获取对应的市信息
42     let currentCityList=data[currentPro]
43     //清空市select中所有的option
44     let ss='<option disabled selected>请选择</option>'
45     cityEle.innerHTML=ss
46     //for循环所有的市渲染到第二个select中
47     for (let i=0; i<currentCityList.length;i++){
48         let currentCity=currentCityList[i]
49         //1.创建option标签
50         let opEle=document.createElement('option')
51         //2.设置文本
52         opEle.innerText=currentCity
53         //设置value值
54         opEle.value=currentCity
55         //将创建好的option添加到select中
56         cityEle.appendChild(opEle)
57     }
58
59 }
60 </script>
61 </body>
62 </html>
63
```