[笔记][LIKE-JS][1-JavaScript语法篇-入门]

JavaScript

001. JS基础-学前须知

002. JS基础-开篇介绍(了解)

编程语言

- 编程语言是人和计算机交流的工具，工程师通过编程语言基于计算机去开发一款款软件
- 编程语言就是一门语言，只不过语法比较特殊，我们得学习之后才能用这门语言来开发相应的东西
- 编程语言大致可以分为以下几类： 机器语言、汇编语言、高级语言

JavaScript的诞生

1995年4月，网景公司录用了34岁的系统程序员Brendan Eich, 他只用10天时间就把Javascript设计出来。

JavaScript语言的前世今生

- 1995.2月 Netscape公司发布LiveScript，后临时改为JavaScript，为了蹭上Java的热浪。
- 欧洲计算机制造商协会（ECMA）英文名称是European Computer Manufacturers Association
- 1997 年，以JavaScript 1.1 为基础。由来自 Netscape、Sun、微软、Borland 和其他一些对脚本编程感兴趣的公司的程序员组成的 TC39（ECMA的小组） 锤炼出了 ECMA-262，也就是ECMAScript1.0。
- 1998年6月，ECMAScript 2.0版发布。
- 1999年12月，ECMAScript - 3.0版发布，成为JavaScript的通行标准，得到了广泛支持。
- 2007年10月，ECMAScript 4.0版草案发布：分歧太大，失败告终。
- 2009年12月，ECMAScript 5.0版正式发布
- 2015年6月17日，ECMAScript 6发布正式版本，即ECMAScript 2015。

> 现在使用最多的是 `ECMAScript 5.0` 和 `ECMAScript 6`

JavaScript来源于借鉴

- 借鉴C语言的基本语法；
- 借鉴Java语言的数据类型和内存管理；
- 借鉴Scheme语言，将函数提升到”第一等公民”（first class）的地位；
- 借鉴Self语言，使用基于原型（prototype）的继承机制。

什么是JavaScript?

JavaScript是一种脚本语言，其源代码在发往客户端运行之前不需经过编译，而是将文本格式的字符代码发送给浏览器由浏览器解释运行。

解释型语言：程序执行之前，不需要编译，直接运行时边解析边执行的语言
编译型语言：程序执行之前，需要一个专门的编译过程，把程序编译成为机器语言的文件，比如exe文件

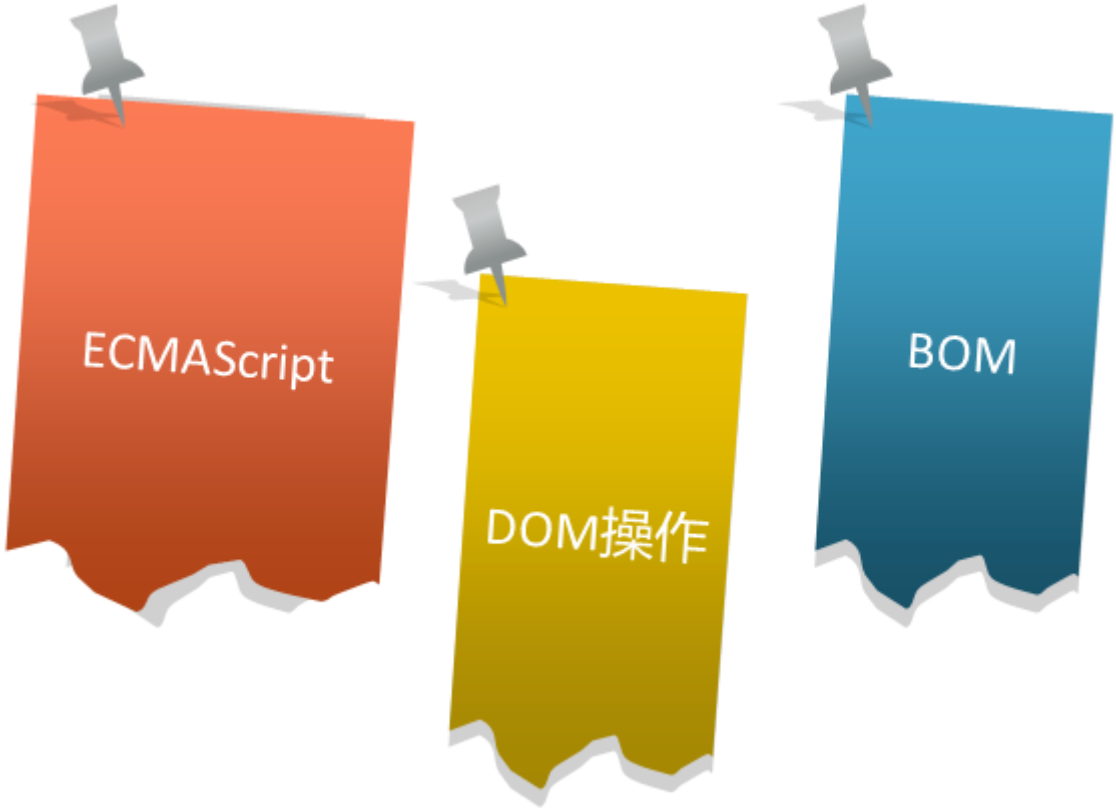
什么是JavaScript?

- 简单易用
  可以使用任何文本编辑工具编写
  只需要浏览器就可以执行程序
- 解释执行（解释语言）
  事先不编译
  逐行执行
  无需进行严格的变量声明
- 基于对象
  内置大量现成对象，编写少量程序可以完成目标

> 解释型语言：程序执行之前，不需要编译，直接运行时边解析边执行的语言
> 编译型语言：程序执行之前，需要一个专门的编译过程，把程序编译成为机器语言的文件，比如exe文件

JavaScript的组成

- ECMAScript：JavaScript的语法标准
  ECMA是一个组织，即欧洲计算机制造商协会
  ECMAScript是ECMA制定的脚本语言的标准, 规定了一种脚本语言实现应该包含的基本内容
  JavaScript是脚本语言的一种,所以JavaScript也必须遵守ECMAScript标准,包含ECMAScript标准中规定的基本内容
- DOM：JavaScript操作网页上的元素的API
- BOM：JavaScript操作浏览器的部分功能的API



JavaScript的使用场景

- 前端开发
- 后端开发
- 移动端开发
- 游戏开发

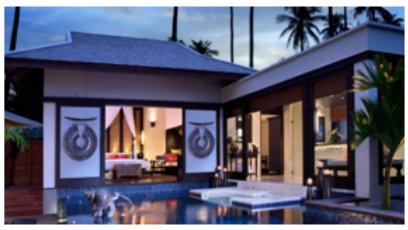
> 当然，学习这门语言最开始的突破口在于前端开发。

JavaScript和HTML、CSS的关系

- Html：是用来制作网页，简单来说就是编写网页结构
- CSS： 美化网页（样式）
- Javascript: 实现网页与客户之间互动的桥梁，让网页具有丰富的生命力

毛坯房

精装修



生活

---

003. JS基础-书写格式(掌握)

JavaScript的语法规范
JavaScript有三种书写格式, 分别是"行内式"、"页内式"、"外链式"

- **行内式**

```
<button onclick="alert('今天天气很好! ');">今天天气很好! </button>
```

> `CSS` 用双引号，行内式的 `JS` 一定要用单引号。但是如果是单独写 `JS` ，单引号双引号都可以。

- **页内式**

```
</body>
     ......
    <script type="text/javascript">
       alert("今天天气很好! ");
    </script>
</body>
```

在实际使用中， `script` 标签写在 `body` 的最后面。
如果要放在 `head` 里面，需要这么写（不建议）：

```
<script>
    window.onload = function () {
        所有JS代码;
    }
</script>
```

> **页内式注意点**
>
> - `<script></script>` 标签中的 `js` 代码一般写在文档的尾部；
> - 网页是从上至下加载, 而js代码通常是给标签添加交互(操作元素), 所以需要先加载HTML, 否则如果执行js代码时HTML还未被加载, 那么js代码将无法添加交互(操作元素);
> - HTML页面中出现 `<script>` 标签后，就会让页面暂停等待脚本的解析和执行。无论当前脚本是内嵌式还是外链式，页面的下载和渲染都必须停下来等待脚本的执行完成才能继续。所以如果把js代码如果写在head中, 那么js代码执行完毕之前后续网页无法被加载。

外链式

- 格式

```
<script type="text/javascript" src="01-js书写格式.js"></script>
```

- 注意点：外链式的script代码块中不能编写js代码，即便写了也不会执行。必须另起一个 `script` 标签来写。

```
<script type="text/javascript" src="index.js">
            alert("今天天气很好！"); // 不会被执行
</script>
```

---

004. JS中常用的输出方式(掌握)

> 注意：加了 `type="text/html"` 的 `js` 代码不起作用

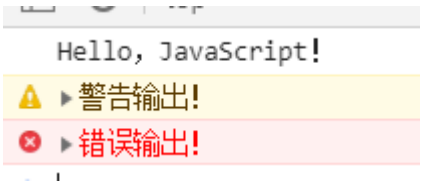
```
<script type="text/html">
    alert('世界你好！');
</script>
```

方式一
在网页中弹出显示框，显示信息

```
  <script>
        alert("Hello，JavaScript！");
  </script>
```

方式二
在控制台输出消息，一般用来调试程序。

```
<script>
    console.log("Hello，JavaScript！");
    console.warn("警告输出！");
    console.error("错误输出！");
</script>
```



方法一和方法二可以结合使用，`alert` 当做断点。

方式三
在网页中弹出输入框，一般用于接收用户输入的信息

```
<script>
    prompt("请输入名字");
</script>
```

方式四

在网页中弹出提示框，显示信息，该方法一般与if判断语句结合使用

```
<script>
    confirm("Hello, JavaScript! ");
</script>
```

> 与 `alert` 区别：多了一个取消

005. JS基础-书写规范和注释(掌握)

1. 一行代码结束后必须在代码最后加上 `;`
   如果不写分号，浏览器会自动添加，但是会消耗一些系统资源；
   此外，容易加错分号，所以在开发中分号必须要写。
2. 在JavaScript中是严格区分大小写的
3. JavaScript中会忽略多个空格和换行

JavaScript中注释语法

- 单行注释：`//`
- 多行注释：`/* ... */`

---

006. JS基础-常量和变量(掌握)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>JS中的常量和变量</title>
</head>
<body>
<script>
    // 数值常量
    console.log(3 + 5);
    console.log(3.3 + 5.5);

    // 字符串常量 '' ""
    console.log('A');
    console.log('中国');

    // 布尔常量 真 1，假 0
    console.log(true);
    console.log(false);

    // 特殊字符
    console.log('\n'); // 换行
    console.log('\t'); // 制表符
    console.log('\r'); // 回车
</script>
</body>
</html>
```

---

007. JS基础-变量的定义和使用(掌握)

- 变量表示一些可以变化的数据
  - 当一个数据的值需要经常改变或者不确定时，就应该用变量来表示；
  - 例如:超市中的储物格就相当于一个变量, 里面可以存储各种不同的东西, 里面存储的东西也可以经常改变；
  - 你去超市放东西到储物柜的格子中, 会给你一张纸条，你根据这个纸条才可以拿回自己的东西，储物柜的一格就是变量的内存空间，字条就是变量名，你拿和放就是修改变量名对应内存中的数据。

- 所以, 变量可以用来保存字面量，而且变量的值是可以任意改变的; 在开发中都是通过变量去保存一个字面量

```
<script>
    // 1. 定义变量
    var name;
    var age, sex;

    // 2. 使用变量
    // = 左右两边各加一个空格
    name = '漩涡鸣人';
    age = 12;
    sex = '男';
    console.log(name);
    console.log(age);
    console.log(sex);

    // 3. 变量初始化
    // 3.1 先定义,后初始化
    var name;
    name = '钢铁侠';

    // 3.2 定义的同时进行初始化
    var age = 38;

    // 3.3 其它表现形式
    // 部分初始化
    var num1, num2 = 100, num3;
    var age1 = age2 = age3 = 4;
</script>
```

---

008. JS基础-关键字,保留字,标识符(掌握)

关键字
它们是被 `JavaScript` 语言赋予了特殊含义的英文单词

> 比如生活中的 `110` 就不能用做自己的电话号码

break	do	instanceof	typeof
case	else	new	var
catch	finally	return	void
continue	for	switch	while
debugger*	function	this	with
default	if	throw	delete
in	try		

保留字

JavaScript预留的关键字，他们虽然现在没有作为关键字，但在以后的升级版本中有可能作为关键字

第5 版把在非严格模式下运行时的保留字缩减为下列这些：

class	enum	extends	super
const	export	import	

在严格模式下，第5 版还对以下保留字施加了限制：

implements	package	public	interface
private	static	let	protected
yield			

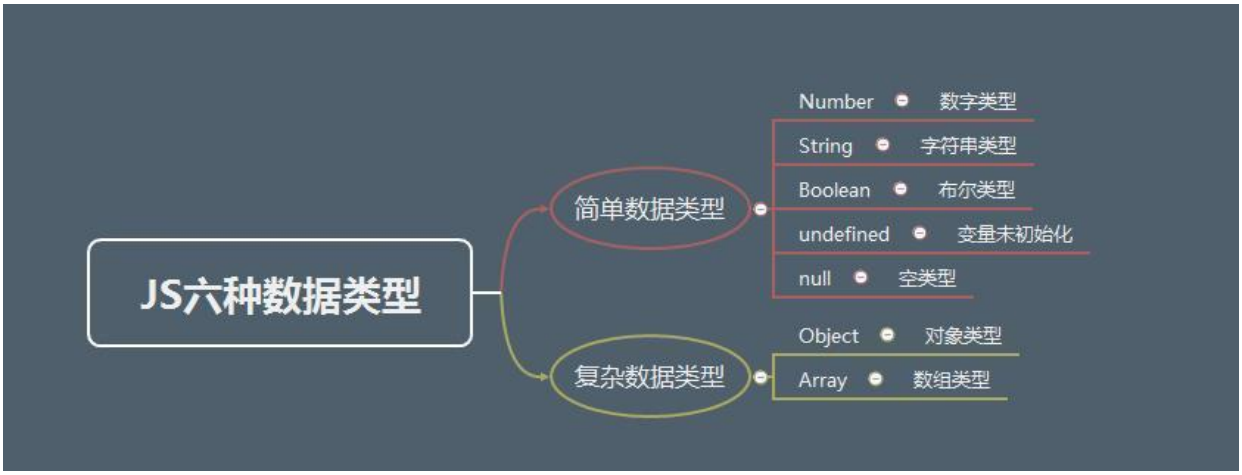
标识符

- 在JS中所有的可以由我们自主命名的都可以称为是标识符
- 比如变量名、函数名、属性名都属于标识符
- 命名一个标识符时需要遵守如下的规则：
  标识符中可以含有字母 、数字 、下划线 `_` 、 `$` 符号
  标识符不能以数字开头
  标识符不能是ES中的关键字或保留字
  严格区分大小写，比如name和Name是2个不同的标识符
  标识符一般都采用驼峰命名法（首字母小写，每个单词的开头字母大写，其余字母小写）
  JS底层保存标识符时实际上是采用的Unicode编码，所以理论上讲，所有的utf-8中含有的内容都可以作为标识符。

---

009. JS基础-数据和数据类型-上(掌握)

在JavaScript中一共有六种数据类型



> `ES6` 和 `ES7` 会有增加
> 简单的数据类型放到栈里面，复杂的数据类型放到堆里面。

JS中如何查看数据类型?

- `typeof` 操作符
  会将检查的结果以字符串的形式返回给我们

> 注意 `typeof` 不用加括号

数字类型（Number）

1. 最基本的数据类型
2. 不区分整型数值和浮点型数值
3. 能表示的最大值是±1.7976931348623157乘以10的308次方
   能表示的最小值是±5 乘以10的-324次方
4. 包含十六进制数据，以 0x开头 0到9之间的数字，a(A)-f(F)之间字母构成。 a-f对应的数字是10-15
5. 八进制直接以数字0开始，有0-7之间的数字组成。

注意

- 在JS中所有的数值都是Number类型(整数和小数)
- 由于内存的限制，ECMAScript 并不能保存世界上所有的数值

```
<script>
    // 最大值
    // 1.7976931348623157e+308
    console.log(Number.MAX_VALUE);
    // 最小值
    // 5e-324
    console.log(Number.MIN_VALUE);

    // 无穷大
    // Infinity
    console.log(Number.MAX_VALUE + Number.MAX_VALUE);
```

```
    // 无穷大和无穷小都是数字类型
    console.log(typeof Infinity);
    console.log(typeof -Infinity);

</script>
```

> `Number` 就是一个内置对象

NaN 非法数字（Not A Number）（这个也属于 `Number` 类型）

- JS中当对数值进行计算时没有结果返回，则返回NaN

```
// NaN
console.log('中国' - 10);
```

Number类型注意点

- 整数精确
- 浮点数可能不精确

---

010. JS基础-数据和数据类型-下(掌握)

字符串类型（string）

- 字符串由单引号或双引号括起
- 单独一个字母也被称为字符串（例如：`'a'`）
- 不能一单一双，必须成对出现；相同引号不能嵌套，不同引号可以嵌套

布尔类型（Boolean）

- 布尔型也被称为逻辑值类型或者真假值类型
- 布尔型只能够取真（true）和假（false）两种数值, 也代表1和0，实际运算中true=1,false=0
- 虽然 `Boolean` 类型的字面值只有两个，但 `ECMAScript` 中所有类型的值都有与这两个 `Boolean` 值等价的值
  - 任何非零数值都是 `true`, 包括正负无穷大, 只有 `0` 和 `NaN` 是 `false`
  - 任何非空字符串都是 `true`, 只有空字符串是 `false`
  - 任何对象都是 `true`, 只有 `null` 和 `undefined` 是 `false`

> `Boolean` 可以把变量转换成布尔类型

```
var bool = Boolean(0);
console.log(bool); // false
```

Undefined

- `Undefined` 这是一种比较特殊的类型，表示变量未赋值，这种类型只有一种值就是 `undefined`

```
var num;
console.log(num);  //结果是undefined
```

- `undefined` 是 `Undefined` 类型的字面量
  前者 `undefined` 和 `10`，`"abc"` 一样是一个常量
  后者 `Undefined` 和 `Number`，`Boolean` 一样是一个数据类型
- 需要注意的是 `typeof` 对没有初始化和没有声明的变量都会返回 `undefined`。

```
<script>
    var str1 = undefined;
    console.log(typeof str1); // 结果是 undefined

    var str2;
    console.log(typeof str2); // 结果是 undefined

    console.log(typeof str3); // 结果是 undefined
</script>
```

Null

`Null` 类型是第二个只有一个值的数据类型，这个特殊的值是 `null`。

- 从语义上看 `null` 表示的是一个空的对象。所以使用 `typeof` 检查 `null` 会返回一个 `Object`

```
<script>
    var str = null;
    console.log(str); // null
    console.log(typeof str); // object
</script>
```

- `undefined` 值实际上是由 `null` 值衍生出来的，所以如果比较 `undefined` 和 `null` 是否相等，会返回 `true`

```
<script>
    var str1 = null;
    var str2 = undefined;
    console.log(str1 == str2); // true
    console.log(str1 === str2); // false
</script>
```

> 两个等于号是值比较，三个等于号是值比较和类型比较

---

011. JS基础-数据类型转化-上(掌握)

`toString()`

- 将其它类型的数据转换为字符串类型
- 注意：`null` 和 `undefined` 没有 `toString()` 方法，如果强行调用,则会报错
- `toString()` 不会影响到原变量，它会将转换的结果返回

> 快捷键 `.log` + `Tab` 自动 `console.log`

String()函数

- 有的时候, 某些值并没有 `toString()` 方法，这个时候可以使用 `String()`。比如：`undefined` 和 `null`。

```
<script>
    var num;
    console.log(String(num)); // undefined
</script>
```

加号 `+`

- ~~任何数据和 `+` 连接到一起都会转换为字符串~~
- 其内部实现原理和 `String()` 函数一样

> 这里其实讲的不太准确，应该通过 `+` 加号与字符串相加的结果，一定是字符串。

```
<script>
    // +
    var num = 123;
    var str = '';
    console.log(str + num); // 123
    console.log(typeof (str + num)); // string
</script>
```

---

012. JS基础-数据类型转化-下(掌握)

> 数字在控制台显示为蓝色

Number()函数
字符串转数字

- 如果是纯数字的字符串，则直接将其转换为数字
- 如果字符串中有非数字的内容，则转换为 `NaN`
- 如果字符串是一个空串或者是一个全是空格的字符串，则转换为 `0`

  > 这里说空白比较恰当，因为我用 `\n\t` 尝试也可以成功。

```
<script>
    var str = '\n\t\r  ';
    console.log(Number(str));
</script>
```

`undefined` 转数字：`NaN`

`null` 转数字：`0`

布尔转数字：`true` 转成 `1`，`false` 转成 `0`

---

parseInt()函数和parseFloat()函数

- Number()函数中无论混合字符串是否存在有效整数都会返回NaN, 利用parseInt()/parseFloat()可以提取字符串中的有效整数
- parseInt()和parseFloat()的区别是前者只能提取整数,后者可以提取小数
- parseInt()提取字符串中的整数
  从第一位有效数字开始, 直到遇到无效数字；如果第一位不是有效数字, 什么都提取不到, 会返回NaN
  第一个参数是要转换的字符串，第二个参数是要转换的进制

```
<script>
    var str = '123.12abc7.44';
    console.log(Number(str)); // NaN
    console.log(parseInt(str)); // 123
    console.log(parseFloat(str)); // 123.12
</script>
```

- parseFloat提取字符串中的小数
  会解析第一个 `.` 遇到第二个 `.` 或者非数字结束
  如果第一位不是有效数字, 什么都提取不到
  不支持第二个参数，只能解析 `10` 进制数
  如果解析的内容里只有整数，解析成整数

- 对非String使用parseInt()或parseFloat(), 会先将其转换为String然后在操作

---

013. JS基础-运算符操作-上(掌握)

运算符
告诉程序执行特定算术或逻辑操作的符号。

按照功能划分

- 算术运算符
- 位运算符
- 关系运算符
- 逻辑运算符

按照操作数个数划分

- 单目运算
- 双目运算
- 三目运算

运算符的优先级

- `1` 级优先级最高，`15` 优先级最低
- 先计算优先级高的，如果相同，左结合计算

运算符	描述
()	表达式分组
++ -- - ~ !	一元运算符
* / %	乘法、除法、取模
+ - +	加法、减法、字符串连接
<< >> >>>	移位
< <= > >=	小于、小于等于、大于、大于等于
== != === !==	等于、不等于、严格相等、非严格相等
&	按位与
^	按位异或
|	按位或
&&	逻辑与
||	逻辑或
?:	条件
= += -= *= /= %=	赋值运算
,	多重求值

运算符的结合性

- 左结合性（自左向右）
- 右结合性（自右向左）
  比如赋值运算符

---

014. JS基础-算术运算符-下(掌握)

算术运算符

- 加法运算
  任何值和 `NaN` 做运算都是 `NaN`
  任何值和字符串做加法，先转换为字符串，再拼接

  > `Python` 中这么相加会出错，不会隐式转换类型

- 减法运算
  非 `Number` 类型的值进行运算时，会将这些值转换为 `Number` 然后再运算
  （与加法不同的是，字符串与数字相减，得到的不是字符串而是数字）
  任何值和 `NaN` 做运算都是 `NaN`
- 乘法运算
  规律与减法运算一样
- 乘法运算
  除数为零的话，结果为 `Infinity`
- 取余运算 `m % n`
  `n` 为零，得到 `NaN`
  `m` 为零，得到 `0`
  `m > n` 正常求余
  `m < n` 为 `m`

```
<script>
    var res = 10 % 3;
    console.log(res); // 1
    res = 10 % 2.5;
    console.log(res); // 0
    res = 10 % 0;
    console.log(res); // NaN
</script>
```

---

015. JS基础-正负运算符(掌握)

正负运算符

- `null` 会转换成 `0`
- 非 `Number` 类型会转成 `Number` 类型

```
<script>
    var a = '12';
    a = +a;
    console.log(a);
    console.log(typeof a);
</script>
```

---

016. JS基础-赋值运算符(掌握)

> 简单数据类型存储在栈区，栈区进行值赋值的。

赋值运算符

- 左边只能是变量
- 多个赋值运算符可以组成赋值表达式，赋值表达式具备右结合性

复合赋值运算符

- `+=`、`-=`、`*=`、`/=`、`%=`
- 由于赋值运算符是右结合性，所以会先计算等号右边，然后再进行复合计算

> 在 `Python` 中这些叫做增量赋值

---

017. JS基础-自增,自减运算符(掌握)

- `i = i + 1` 对应的自增运算符 `i++` 或者 `++i`
- `i = i - 1` 对应的自减运算符 `i--` 或者 `--i`
- 后缀表达式 `i++` 和 `i--` 先用后变
- 前缀表达式 `i++` 和 `i--` 先变后用

---

018. JS基础-关系运算符(掌握)

关系运算符它的返回值正好就是Boolean类型的值, 也就是说关系运算符的返回值要么是true,要么是false

序号		运算符	结合性
1	>	大于	左结合
2	<	小于	左结合
3	>=	大于等于	左结合
4	<=	小于等于	左结合
5	==	判断值是否相等	左结合
6	!=	判断值是否不相等	左结合
7	===	判断值和类型是否相等	左结合
8	!==	判断值和类型是否不相等	左结合

注意

- 对于两个操作数，一个为数字，另一个为非数字，会将非数字转换为数字然后再比较
- 如果符号两侧的值都是字符串时，不会将其转换为数字进行比较，而会分别比较字符串中字符的 `Unicode` 编码
  - 比较字符编码时是一位一位进行比较
  - 如果两位一样，则比较下一位，所以借用它来对英文进行排序
  - 比较中文时没有意义
- `null`、`undefined`、`NaN` 比较。使用 `isNaN()` 函数判断是不是 `NaN`。

  > `null` 的比较行为有点奇怪？

```
    console.log(null < 1);  // true
    console.log(null == 0);  // false
    console.log(0 == null); // false
    console.log(7 > null); //  true

    console.log(undefined == 0);  // false
    console.log(NaN == 0);  // false
    console.log(NaN == NaN);  // false

    var num = 100;
    console.log(isNaN(num));  // false

    console.log(null == undefined);  // true
    console.log(undefined == null);  // true
    console.log(null === undefined);  // false
```

- 比较两个字符串型的数字，可能会得到不可预期的结果。所以，在比较两个字符串型的数字时，一定要转型。

```
<script>
    console.log("1111123" < "124");  // true
</script>
```

---

019. JS基础-逻辑运算符-逻辑与(掌握)

- `&&`(与运算)
- `||`(或运算)
- `!`(非运算)

> `Python` 的与或非是 `and`、`or`、`not`

注意：短路逻辑！只要第一个是假的，就不继续往后求值。

短路写法：满足条件 `a`，就做什么事情（非主流写法）
`a && alert('哈哈哈');`

> 这个有点类似 `Python` 的取缺省值写法。
> `val = 结果 or 缺省值`
> 如果没有得到结果（为 `None`、空、假），那么就使用缺省值。

注意

- 对于非Boolean类型的数值, 逻辑与会自动将其转换为Boolean类型来判断
- 如果条件A不成立, 则返回条件A的数值本身
- 如果条件A成立, 不管条件B成不成立都返回条件B数值本身

```
<script>
    var result = 123 && 'abc';
    console.log(result);  // abc
    console.log(typeof result);  // string
</script>
```

---

020. JS基础-逻辑或,逻辑非(掌握)

逻辑或 `||`
一真全真：假如 `A` 成立，不会继续求值。

短路测试

```
<script>
    (7 < 30) || alert('我不会来');  // 不弹警告
    (7 < 3) || alert('我来了');  // 弹出警告
</script>
```

注意点

- 对于非Boolean类型的数值, 逻辑或自动会将其转换为Boolean类型来判断
- 如果条件A不成立, 则不管条件B成不成立都返回条件B数值本身
- 如果条件A成立, 则返回条件A的数值本身

逻辑非 `!`
`!条件`：对条件取反

注意点

- 对一个布尔类型进行两次取反，不会变化
- 对于非布尔类型，两次取非，等效于 `Boolean` 类型转换

---

021. JS基础-逗号运算符,三目运算符(掌握)

逗号运算符

- 在JavaScript中逗号“,”也是一种运算符,称为逗号运算符。 其功能是把多个表达式连接起来组成一个表达式, 称为逗号表达式。
- 一般形式形式: `表达式1，表达式2，… …，表达式n;`
- 例如: `a = 1 + 1, b = 3 * 4, c = 10 / 2;`
- 逗号表达式的运算过程是：先算表达式1，再算表达式2，依次算到表达式n
- 整个逗号表达式的值是最后一个表达式的值

```
var a, b, c, d;
/*
1.先计算表达式1, a = 2
2.再计算表达式2, b = 12
3.再计算表达式3, c = 5
4.将表达式3的结果返回给d
*/
d = (a = 1 + 1,b = 3 * 4, c = 10 / 2);
console.log(d); // 5
```

注意

- 程序中使用逗号表达式, 通常是要分别求逗号表达式内各表达式的值,并不一定要求整个逗号表达式的值。（也就是最左边不一定要有一个变量接收值）
- 并不是在所有出现逗号的地方都组成逗号表达式,例如在变量说明中,函数参数表中逗号只是用作各变量之间的间隔符。

三目运算符（取代 `if/else`）

- 格式: `条件表达式 ? 语句1 : 语句2;`
- 如果条件表达式为true，则执行语句1，并返回执行结果
- 如果条件表达式为false，则执行语句2，并返回执行结果

注意

- 条件运算符 `?` 和 `:` 是一对运算符,不能分开单独使用
- 如果条件的表达式的求值结果是一个非布尔值, 会将其转换为布尔值然后再运算

022. JS基础-练习讲解(掌握)

```
<script>
    // 用户输入三个数字，找到最大值并输出
    var num1, num2, num3;

    num1 = Number(prompt('请输入第一个数'));
    num2 = Number(prompt('请输入第二个数'));
    num3 = Number(prompt('请输入第三个数'));

    var max;

    max = num1 > num2 ? num1 : num2;
    max = max > num3 ? max : num3;

    console.log(max);

</script>
```

另一种紧凑版写法

```
<script>
    // 用户输入三个数字，找到最大值并输出
    var num1, num2, num3, max;

    num1 = Number(prompt('请输入第一个数'));
    num2 = +prompt('请输入第二个数');
    num3 = +(prompt('请输入第三个数'));

    // 不推荐
    max = (num1 > num2 ? num1 : num2) > num3 ? (num1 > num2 ? num1 : num
2) : num3;

    console.log(max);

</script>
```

完成于 `2019.02.23`