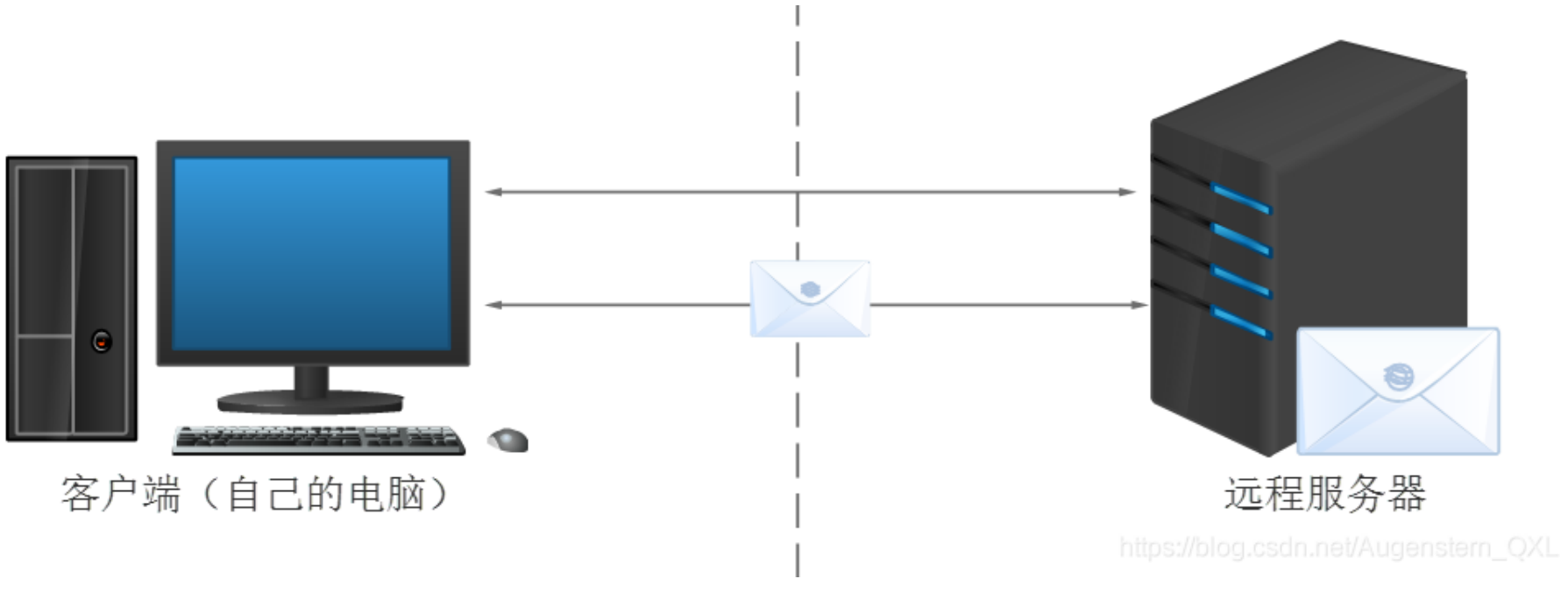
JavaScript基础大总结

🔥JavaScript帝国之行🔥

内容	地址
JavaScript基础大总结(一) 🔥	https://blog.csdn.net/Augenstern_QXL/article/details/119249534
JavaScript基础之函数与作用域(二) 🔥	https://blog.csdn.net/Augenstern_QXL/article/details/119250991
JavaScript基础之对象与内置对象(三) 🔥	https://blog.csdn.net/Augenstern_QXL/article/details/119250137
JavaScript进阶之DOM技术(四) 🔥	https://blog.csdn.net/Augenstern_QXL/article/details/115416921
JavaScript进阶之BOM技术(五) 🔥	https://blog.csdn.net/Augenstern_QXL/article/details/115406408
JavaScript提高之面向对象(六) 🔥	https://blog.csdn.net/Augenstern_QXL/article/details/115219073
JavaScript提高之ES6(七) 🔥	https://blog.csdn.net/Augenstern_QXL/article/details/115344398

✍初识JavaScirpt

- 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



✍浏览器执行JS简介

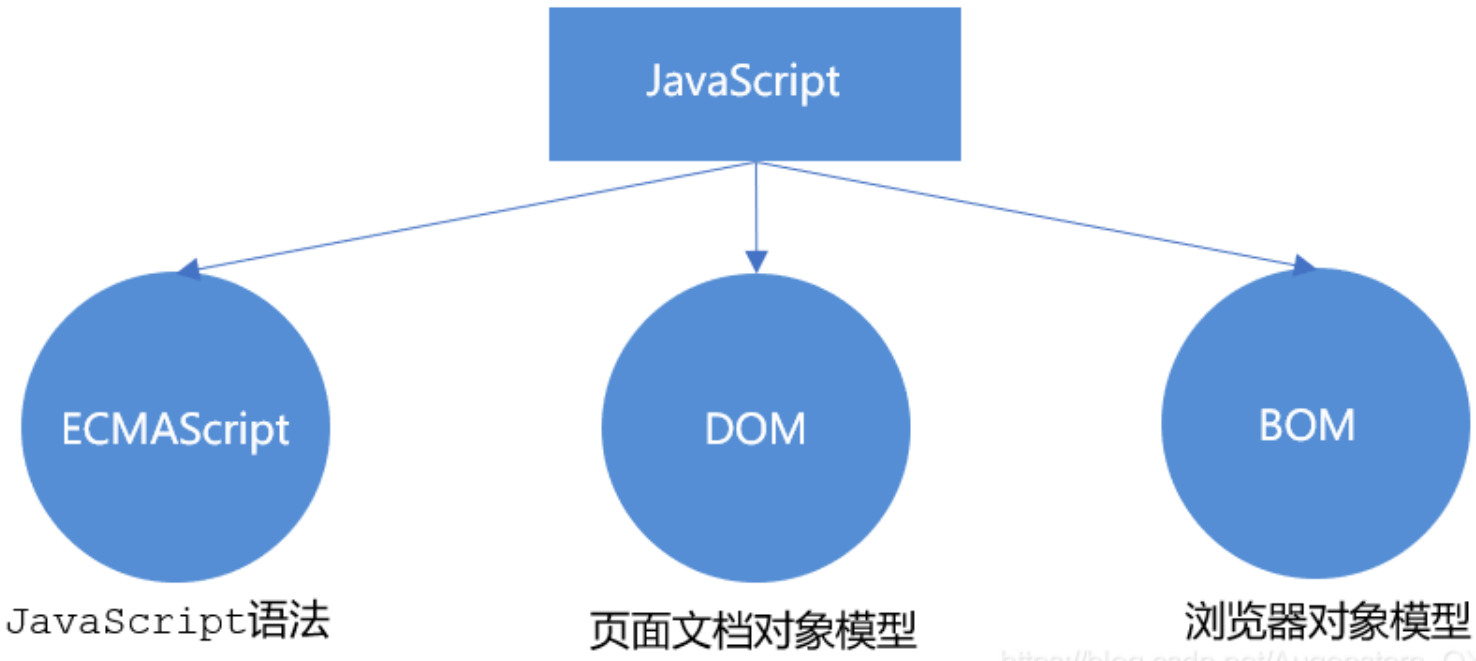
浏览器分成两部分：渲染引擎和 JS 引擎

- 渲染引擎：用来解析HTML与CSS，俗称内核，比如 chrome 浏览器的 blink ，老版本的 webkit
- JS 引擎：也称为 JS 解释器。 用来读取网页中的JavaScript代码，对其处理后运行，比如 chrome 浏览器的 V8

浏览器本身并不会执行JS代码，而是通过内置 JavaScript 引擎(解释器) 来执行 JS 代码 。JS 引擎执行代码时逐行解释每一句源码（转换为机器语言），然后由计算机去执行，所以 JavaScript 语言归为脚本语言，会逐行解释执行。

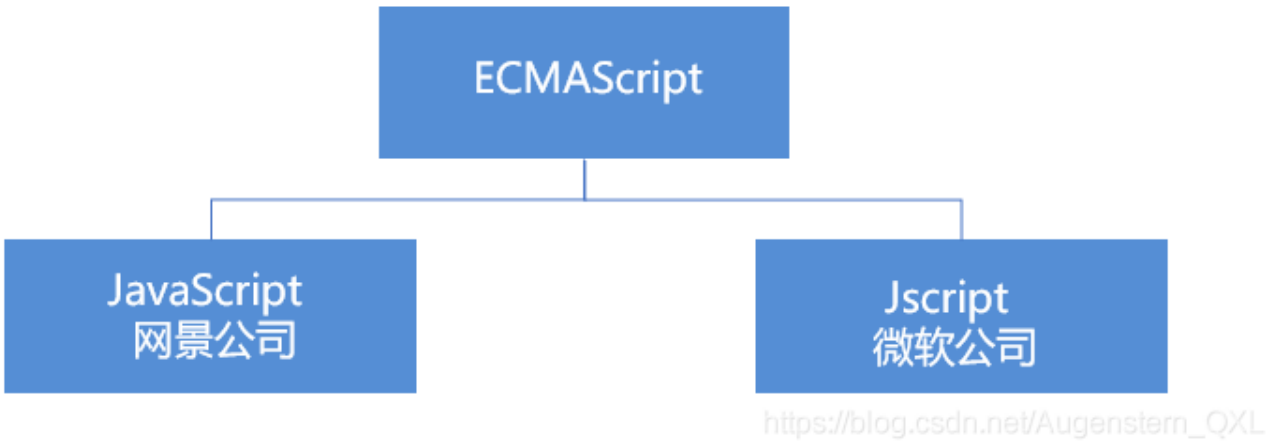
✍JS的组成

JavaScript 包括 ECMAScript、DOM、BOM



✍️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

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。

1、JS初体验🔥

1.1、行内式JS

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

1. 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： onclink
2. 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**
3. 可读性差，在 HTML 中编入 JS 大量代码时，不方便阅读
4. 特殊情况下使用

1.2、内嵌式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

1.3、外部JS🔥

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

1. 利于HTML页面代码结构化，把单独JS代码独立到HTML页面之外，既美观，又方便
2. 引用外部JS文件的script标签中间不可以写代码
3. 适合于JS代码量比较大的情况

2、JS基本语法 🔥

2.1、注释 🔥

2.1.1、单行注释 🔥

```
//  单行注释
```

- 快捷键 `ctrl + /`

2.1.2、多行注释 🔥

```
/*
    多行注释
*/
```

- 快捷键 `shift + alt + a`
- vscode中修改快捷键方式：vscode➡ 首选项按钮➡ 键盘快捷方式 ➡ 查找原来的快捷键➡ 修改为新的快捷键➡ 回车确认

2.2、输入输出语句 🔥

方法	说明	归属
alert(msg);	浏览器弹出警示框	浏览器
console.log(msg);	浏览器控制台打印输出信息	浏览器
prompt(info);	浏览看弹出输入框，用户可以输入	浏览器

- `alert()` 主要用来显示消息给用户
- `console.log()` 用来给程序员看自己运行时的消息

2.3、变量 🔥

- 变量是用于存放数据的**容器**，我们通过**变量名**获取数据，甚至数据可以修改
- **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

2.3.1、变量初始化 🔥

1. var是一个JS关键字，用来声明变量(variable变量的意思)。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。
2. age 是程序员定义的变量名，我们要通过变量名来访问内存中分配的空间

```
//声明变量同时赋值为18
var age = 18;
//同时声明多个变量时，只需要写一个 var， 多个变量名之间使用英文逗号隔开。

var age = 18, address ='火影村',salary = 15000;
```

2.3.2、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

2.3.3、变量的命名规范 🔥

1. 由字母(A-Z,a-z)，数字(0-9)，下划线(_)，美元符号($)组成，如:usrAge,num01,__name
2. 严格区分大小写。 `var app;` 和 `var App;` 是两个变量
3. 不能以数字开头。
4. 不能是关键字，保留字。例如： `var,for,while`
5. 遵循驼峰命名法。首字母小写，后面单词的首字母需要大写。 `myFirstName`
6. 推荐翻译网站：有道 爱词霸

2.4、数据类型 🔥

JavaScript **是一种弱类型或者说动态语言。**这意味着不用提前声明变量的类型，在程序运行过程中，类型会被自动确定。

```
var age = 10;            //这是一个数字型
var areYouOk = '使得';   //这是一个字符串
```

- 在代码运行时，变量的数据类型是由 JS引擎 **根据 = 右边变量值的数据类型来判断** 的，运行完毕之后， 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用作不同的类型

```
var x = 6;      //x为数字
var x = "Bill"; //x为字符串
```

JS 把数据类型分为两类：

- 基本数据类型(Number,String,Boolean,Undefined,Null)
- 复杂数据类型(Object)

2.4.1、基本数据类型 🔥

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
Undefined	var a; 声明了变量a但是没有赋值，此时a=undefined	undefined（未定义的）
string	字符串类型，如“张三”	“”
Null	var a = null;声明了变量a为空值	null

2.4.2、数字型Number

JavaScript 数字类型既可以用来保存整数值，也可以保存小数(浮点数）。

```
var age = 12;       //整数
var Age = 21.3747;  //小数
```

2.4.2、数字型进制 🔥

最常见的进制有二进制、八进制、十进制、十六进制。

```
// 1.八进制数字序列范围：0~7
var num1 = 07;      //对应十进制的7
var Num2 = 019;     //对应十进制的19
var num3 = 08;      //对应十进制的8


// 2.十六进制数字序列范围：0~9以及A~F
var num = 0xA;
```

- **在JS中八进制前面加0，十六进制前面加 0x**

①数字型范围 🔥

- JS中数值的最大值： `Number.MAX_VALUE`
- JS中数值的最小值： `Number.MIN_VALUE`

```
consol.log(Number.MAX_VALUE);
consol.log(Number.MIN_VALUE);
```

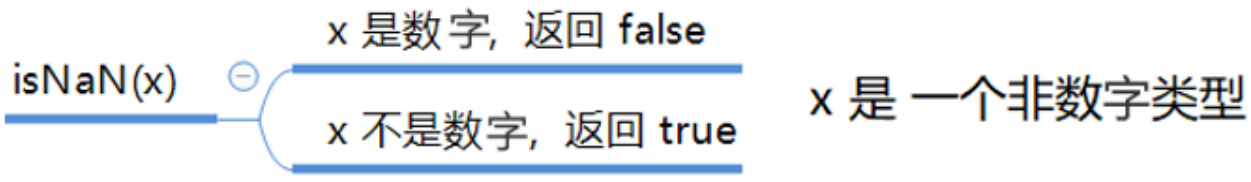
②数字型的三个特殊值 🔥

```
alert(Infinity);    //Infinity(无穷大)
alert(-Infinity);   //-Infinity(无穷小)
alert(NaN);         //NaN - Not a Number ,代表任何一个非数值
```

- Infinity， 代表无穷大，大于任何数值
- -Infinity， 代表无穷小，小于任何数值
- Nan， Not a Number，代表一个非数值

③isNaN 🔥

这个方法用来判断非数字，并且返回一个值，如果是数字返回的是false，如果不是数字返回的是true




```
var userAge = 21;
var isOk = isNan(userAge);
console.log(isOk);      //false,21不是一个非数字

var userName = "andy";
console.log(isNan(userName));   //true,"andy"是一个非数字
```

2.4.3、字符串型String 🔥

字符串型可以是引号中的任意文本，其语法为 “**双引号**” 和 "**单引号**”

```
var strMsg = "我爱北京天安门~";       //使用双引号表示字符串
var strMsg = '我爱北京';           //使用单引号表示字符串
```

因为 HTML 标签里面的属性使用的是双引号，JS 这里我们更推荐**使用单引号。**

①字符串引号嵌套 🔥

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

```
var strMsg ='我是一个“高富帅”' //可以用 ' ' 包含 " "
var strMsg2 ="我是'高富帅'" //可以用" "  包含  ''
```

②字符串转义符 🔥

类似HTML里面的特殊字符，字符串中也有特殊字符，我们称之为转义符。

转义符都是 \ 开头的，常用的转义符及其说明如下：

转义符	解释说明
\n	换行符，n是newline
\ \	斜杠\
\ ’	’ 单引号
\ “	“ 双引号
\ t	tab 缩进
\ b	空格，b是blank的意思

③字符串长度 🔥

字符串是由若干字符组成的，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。

```
//通过字符串的length属性可以获取整个字符串的长度
var strMsg = "我是高富帅！";
alert(strMsg.length);     //显示6
```

④字符串的拼接 🔥

- 多个字符串之间可以使用 + 进行拼接，其拼接方式为 **字符串 + 任何类型 = 拼接之后的新字符串**
- 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

注意：字符串 + 任何类型 =拼接之后的新字符串

```
//1 字符串相加
alert('hello' + ' ' + 'World');  //hello World

//2 数值字符串相加
alert('100' + '100'); //100100

//3 数值字符串+数值
alert('12'+12); //1212

//4 数值+数值
alert(12+12); //24
```

- + 号总结口诀：🌏数值相加，字符相连🌏

```
var  age = 18;
console.log('我今年'+age+'岁');
console.log('我今年'+age+'岁');  //引引加加，最终也是上面的形式
```

⑤字符串拼接加强🔥

```
console.log('Pink老师' + 18);           //只要有字符就会相连
var age = 18;
// console.log('Pink老师age岁了');       //这样不行,会输出 "Pink老师age岁了"

console.log('Pink老师' + age);        // Pink老师18
console.log('Pink老师' + age + '岁啦');  // Pink老师18岁啦
```

- 我们经常会将字符串和变量来拼接，因为变量可以很方便地修改里面的值
- 变量是不能添加引号的，因为加引号的变量会变成字符串
- 如果变量两侧都有字符串拼接，口诀==🌏“引引加加 ”，删掉数字🌏==变量写加中间

2.4.4、布尔型Boolean 🔥

- 布尔类型有两个值：true 和 false ， 其中 true 表示真（对），而 false 表示假（错）。
- 布尔型和数字型相加的时候， true 的值为 1 ， false 的值为 0。

```
var flag = true;
console.log(flag + 1); // 2 true当加法来看当1来看，flase当0来看
```

2.4.5、undefined未定义 🔥

- 一个**声明后没有被赋值**的变量会有一个默认值 undefined (如果进行相连或者相加时，注意结果)

```
// 如果一个变量声明未赋值，就是undefined 未定义数据类型
var str;
console.log(str);                //undefined
var variable = undefined;
console.log(variable + 'Pink'); //undefinedPink
console.log(variable + 18); //NaN
```

1.undefined 和 字符串 相加，会拼接字符串

2.undefined 和 数字相加，最后结果是**NaN**

2.4.6、空值null 🔥

- 一个声明变量给 null 值，里面存的值为空

```
var space = null;
console.log(space + 'pink'); //nullpink
console.llog(space + 1); // 1
```

2.4.7、typeof 🔥

- typeof 可用来获取检测变量的数据类型

```
var num = 18;
console.log(typeof num) // 结果 number
```

不同类型的返回值

类型	例	结果
string	typeof “小白”	“string”
number	typeof 18	“number”
boolean	typeof true	“boolean”
undefined	typeof undefined	“undefined”
null	typeof null	“object”

2.4.8、字面量

字面量是在源代码中一个固定值的表示法，通俗来说，就是字面量表示如何表达这个值。

- 数字字面量：8，9，10
- 字符串字面量：‘大前端’，‘后端’
- 布尔字面量：true、false

通过控制台的颜色判断属于哪种数据类型

黑色	字符串
蓝色	数值
灰色	undefined 和 null

2.5、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

①转换为字符串型 🔥

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+“我是字符串”);

```
//1.把数字型转换为字符串型 toString()  变量.toString()
var num = 10;
```

```
var str = num.toString();
console.log(str);

//2.强制转换
console.log(String(num));
```

- 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，这一方式也称为隐士转换

②转换为数字型 🔥

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

```
// 1.parseInt()
var age =prompt('请输入您的年龄');
consolo.log(parseInt(age));  //数字型18
consolo.log(parseInt('3.14'));  //3取整
consolo.log(parseInt('3.94'));  //3,不会四舍五入
consolo.log(parseInt('120px'));  //120,会去掉单位

// 2.parseFloat()
console.log(parseFloat('3.14'));  //3.14
consolo.log(parseFloat('120px'));  //120,会去掉单位


// 3.利用Number(变量)
var str ='123';
console.log(Number(str));
console.log(Number('12'));

// 4.利用了算术运算 - * /   隐式转换
console.log('12'-0);  // 12
console.log('123' - '120');  //3
console.log('123' * 1);  // 123
```

1.注意 parseInt 和 parseFloat， 这两个是重点

2.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

③转换为布尔型

方法	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
- 其余的值都会被被转换为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
```

2.6、运算符 🔥

运算符（operator）也被称为**操作符**，是用于实现赋值、比较和执行算数运算等功能的符号

JavaScript 中常用的运算符有：

- 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

2.6.1、算术运算符 🔥

概念：算术运算使用的符号，用于执行两个变量或值的算术运算。

运算符	描述	实例
+	加	10 + 20 =30
-	减	10 - 20 =-10
*	乘	10 * 20 =200
/	除	10 / 20 =0.5
%	取余数（取模）	返回出发的余数 9 % 2 =1

2.6.2、浮点数的精度问题 🔥

浮点数值的最高精度是17位小数，但在进行算数计算时其精确度远远不如整数

```
var result = 0.1 +0.2; //结果不是0.3，0.30000000000000004
console.log(0.07 * 100); //结果不是7，而是7.000000000000001
```

所以不要直接判断两个浮点数是否相等

2.6.3、递增和递减运算符 🔥

递增（++）

递减（- -）

放在变量前面时，我们称为**前置递增(递减)运算符**

放在变量后面时，我们称为**后置递增(递减)运算符**

注意：递增和递减运算符必须和变量配合使用。

①前置递增运算符 🔥

++num num = num + 1

使用口诀:**先自加，后返回值**

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

先自加 10+1=11，返回11，此时num=11

②后置递增运算符 🔥

num ++ num = num +1

使用口诀:**先返回原值，后自加**

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

③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
- 开发时，大多使用后置递增/减，并且代码独占一行

2.6.4、比较(关系)运算符 🔥

比较运算符是**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

①===== 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(注意此时有隐士转换)
===	全等	判断两边的值和数据类型是否完全相同

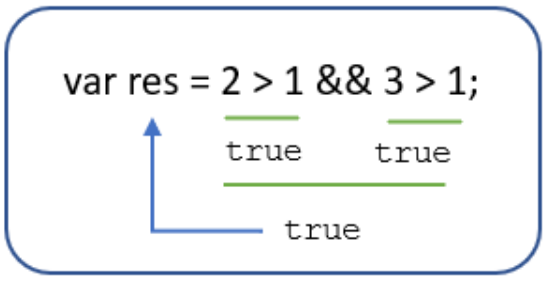
```
console.log(18 == '18');        //true
console.log(18 === '18');       //false
```

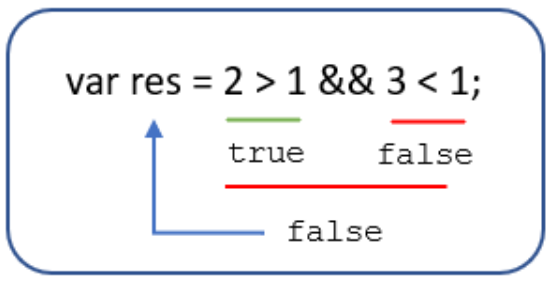
2.6.5、逻辑运算符 🔥

逻辑运算符是用来进行布尔值运算的运算符，其返回值也是布尔值

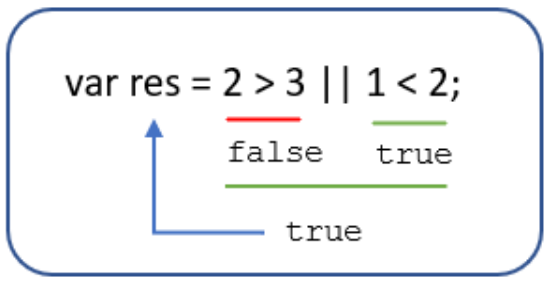
逻辑运算符	说明	案例
&&	“逻辑与”，简称"与" and	true && false
||	“逻辑或”，简称"或" or	true || false
!	“逻辑非”，简称"非" not	！true

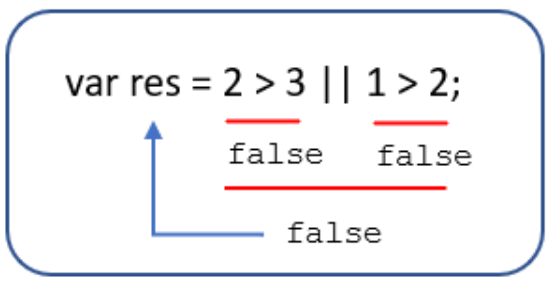
逻辑与：两边都是 true才返回 true，否则返回 false





逻辑或：两边都为 false 才返回 false，否则都为true





逻辑非：逻辑非（!）也叫作取反符，用来取一个布尔值相反的值，如 true 的相反值是 false

```
var isOk = !true;
console.log(isOk);  // false
//逻辑非（!）也叫作取反符，用来取一个布尔值相反的值，如 true 的相反值是 false
```

2.6.5.1、短路运算(逻辑中断) 🔥

短路运算的原理：当有多个表达式（值）时,左边的表达式值可以确定结果时,就不再继续运算右边的表达式的值

①逻辑与 🔥

- 语法：表达式1 && 表达式2
- 如果第一个表达式的值为真，则返回表达式2
- 如果第一个表达式的值为假，则返回表达式1

```
console.log(123 && 456);   //456
console.log(0 && 456);     //0
console.log(123 && 456 && 789);  //789
```

②逻辑或

- 语法：表达式1 || 表达式2
- 如果第一个表达式的值为真，则返回表达式1
- 如果第一个表达式的值为假，则返回表达式2

```
console.log(123 || 456); //123
console.log(0 || 456);   //456
console.log(123 || 456 || 789);  //123
```

```
var num = 0;
console.log(123 || num++);
// 先返回在加，相当于 (123 || 0)
console.log(num);    // 123
```

2.6.6、赋值运算符 🔥

概念：用来把数据赋值给变量的运算符。

赋值运算符	说明	案例
=	直接赋值	var usrName = ‘我是值’
+= ，-=	加，减一个数后再赋值	var age = 10； age+=5；//15
*=，/=，%=	成，除，取模后再赋值	var age = 2; age*=5; //10

```
var age = 10;
age += 5;  // 相当于 age = age + 5;
age -= 5;  // 相当于 age = age - 5;
age *= 10; // 相当于 age = age * 10;
```

2.6.7、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 ||（先与后或）**
7	赋值运算符	=
8	逗号运算符	,

1.一元运算符里面的**逻辑非**优先级很高

2.**逻辑与** 比 **逻辑或** 优先级高

3.练习题

```
console.log( 4 >= 6 || '人' != '阿凡达' && !(12 * 2 == 144) && true) // true
```

```
var a = 3 > 5 && 2 < 7 && 3 == 4;
console.log(a);     //false

var b = 3 <= 4 || 3 > 1 || 3 != 2;
console.log(b);     //true

var c = 2 === "2";
console.log(c);     //false

var d = !c || b && a ;
console.log(d);     //true
```

2.7、流程控制 🔥

流程控制主要有三种结构，分别是顺序结构、分支结构和循环结构，这三种结构代表三种代码执行的顺序

2.7.1、分支结构 🔥

JS 语言提供了两种分支结构语句：**JS 语句 switch语句**

①if语句 🔥

```
// 条件成立执行代码，否则什么也不做
if (条件表达式) {
    //条件成立执行的代码语句
}
```

案例：进入网吧

弹出一个输入框，要求用户输入年龄，如果年龄大于等于 18 岁，允许进网吧

```
var usrAge = prompt('请输入您的年龄:');
if(usrAge >= 18)
{
      alert('您的年龄合法，欢迎来到老子网吧享受学习的乐趣！');
}
```

②if else 语句 🔥

```
// 条件成立，执行if里面代码，否则执行else里面的代码
if(条件表达式)
{
    //[如果]条件成立执行的代码
}
else
    {
        //[否则]执行的代码
    }
```

案例：判断闰年

接收用户输入的年份，如果是闰年就弹出闰年，否则弹出是平年

算法：能被4整除且不能整除100的为闰年（如2004年就是闰年，1901年不是闰年）或者能够被 400 整除的就是闰年

```
var year = prompt('请输入年份');

if (year % 4 == 0 && year % 100 !=0 || year % 400 ==0)
{
   alert('这个年份是闰年');
}
else
{
  alert('这个年份是平年');
}
```

③if else if 语句 🔥

```
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
{
   //上述条件都不成立执行此处代码
}
```

案例:接收用户输入的分数，根据分数输出对应的等级字母 A、B、C、D、E

其中:

1. 90分(含)以上 ， 输出：A
2. 80分(含)~ 90 分(不含)，输出：B
3. 70分(含)~ 80 分(不含)，输出：C
4. 60分(含)~ 70 分(不含)，输出：D
5. 60分(不含) 以下，输出： E

```
var score = prompt('请您输入分数:');
       if (score >= 90) {
           alert('宝贝，你是我的骄傲');
       } else if (score >= 80) {
           alert('宝贝，你已经很出色了');
       } else if (score >= 70) {
           alert('你要继续加油喽');
       } else if (score >= 60) {
           alert('孩子，你很危险');
       } else {
           alert('可以再努力点吗，你很棒，但还不够棒');
       }
```

2.7.2、三元表达式 🔥

- 语法结构 : 表达式1 ? 表达式2 : 表达式3
- 执行思路

如果表达式1为true，则返回表达式2的值,如果表达式1为false，则返回表达式3的值

案例：数字补0

用户输入数字，如果数字小于10，则在前面补0，比如01，09，

如果数字大于10，则不需要补，比如20

```
var figuer = prompt('请输入0~59之间的一个数字');
        var result = figuer < 10 ? '0' + figuer : figue
        alert(result);
```

2.7.3、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： 开关 转换， case： 小例子 选项

- 关键字 switch 后面**括号内**可以是**表达式或值**， 通常是一个**变量**
- 关键字 case , 后跟一个选项的表达式或值，**后面跟一个冒号**
- switch 表达式的值会与结构中的 case 的值做比较
- 如果存在匹配**全等**(===)，则与该 case 关联的代码块会被执行，并在遇到 **break 时停止**，整个 switch 语句代码执行结束
- 如果所有的 case 的值都和表达式的值不匹配，则执行 default 里的代码
- **执行case 里面的语句时，如果没有break，则继续执行下一个case里面的语句**

```
// 用户在弹出框里面输入一个水果，如果有就弹出该水果的价格， 如果没有该水果就弹出“没有此水果”
        var fruit = prompt('请您输入查询的苹果');
        switch (fruit) {
            case '苹果':
                alert('苹果的价格为3.5元/千克');
                break;
            case '香蕉':
                alert('香蕉的价格为3元/千克');
                break;
            default:
                alert('没有这种水果');
        }
```

3、断点调试 🔥

1. 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(在行数点一下)
2. 刷新浏览器
3. Watch: 监视，通过watch可以监视变量的值的变化，非常的常用
4. 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

4、循环 🔥

4.1、for循环 🔥

在程序中，一组被重复执行的语句被称之为**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(初始化变量;条件表达式;操作表达式)
{
   //循环体
}
```

1.输入10句"娘子晚安哈！"

```
//基本写法
for(var i = 1; i<=10; i++  )
    {
         console.log('娘子晚安哈');
    }
// 用户输入次数
var num = prompt('请输入次数:');
for(var i = 1; i<= num ;i++)
    {
        console.log('娘子晚安哈');
    }
```

2.求1-100之间所有整数的累加和

```
// 求1-100所以的整数和
var sum = 0;
for (var i = 1; i <= 100; i++) {
    var sum = sum + i;
}
console.log(sum);
```

3.求1-100之间所有数的平均值

```
 // 3.求1-100之间所有数的平均值
var sum = 0;
for (var i = 1; i <= 100; i++) {
```

```
    var sum = sum + i;
}
console.log(sum / 100);
```

4.求1-100之间所有偶数和奇数的和

```
//    4.求1-100之间所有偶数和奇数的和
var sum1 = 0;
var sum2 = 0;
for (var i = 1; i <= 100; i++) {
    if (i % 2 == 0) {
        sum1 = sum1 + i;
    } else {
        sum2 = sum2 + i;
    }
}
console.log('偶数和为' + sum1);
console.log('奇数和为' + sum2);
```

5.求1-100之间所有能被3整除的数字的和

```
// 5.求1-100之间所有能被3整除的数字的和
var sum = 0;
for (var i = 1; i <= 100; i++) {
    if (i % 3 == 0) {
        sum += i;
    }
}
console.log(sum);
```

6.要求用户输入班级人数，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成绩。

```
var num = prompt('请输入班级总的人数:'); // num 班级总的人数
var sum = 0; // 总成绩
var average = 0; // 平均成绩
for (var i = 1; i <= num; i++) {
     var score = prompt('请输入第' + i + '个学生的成绩');
    //这里接收的是str，必须转换为数值
     sum = sum + parseFloat(score);
}
average = sum / num;
alert('班级总的成绩是：' + sum);
alert('班级总的平均成绩是：' + average);

```

7.一行打印5个星星

我们采取追加字符串的方式，这样可以打印到控制台上

```
var star = '';
for (var i = 1; i <= 5; i++) {
     star += '☆';
 }
console.log(star);
```

4.2、双重for循环 🔥

循环嵌套是指在一个循环语句中再定义一个循环语句的语法结构，例如在for循环语句中，可以再嵌套一个for 循环，这样的 for 循环语句我们称之为双重for循环。

```
for(外循环的初始;外循环的条件;外形循环的操作表达式){
    for(内循环的初始;内循环的条件;内循环的操作表达式){
        需执行的代码;
    }
}
```

- 内层循环可以看做外层循环的语句
- 内层循环执行的顺序也要遵循 for 循环的执行顺序
- 外层循环执行一次，内层循环要执行全部次数

①打印五行五列星星

核心：

- 内层循环负责一行打印五个星星
- 外层循环负责打印五行

```
var star = '';
for(var j = 1;j<=5;j++)
{
   for (var i = 1; i <= 5; i++)
   {
     star += '☆'
   }
    //每次满5个星星就加一次换行
    star +='\n'
}
console.log(star);
```

②打印n行n列的星星

要求用户输入行数和列数，之后在控制台打印出用户输入行数和列数的星星

```
var star = '';
var row = prompt('请输入行数');
var col = prompt('请输入列数');
for (var j = 1; j <= col; j++) {
    for (var i = 1; i <= row; i++) {
        star += '☆';
    }
    star += '\n';
}
console.log(star);
```

③打印倒三角形

- 一共有10行，但是每行的星星个数不一样，因此需要用到双重 for 循环
- 外层的 for 控制行数 i ，循环10次可以打印10行
- 内层的 for 控制每行的星星个数 j
- 核心算法： 每一行星星的个数： j = i ; j <= 10; j++
- 每行打印完毕后，都需要重新换一行

```
var star = '';
var row = prompt('请输入行数');
var col = prompt('请输入列数');
for (var i = 1; i <= row; i++) {
    for (var j = i; j <= col; j++) {
        star += '☆';
    }
    star += '\n';
}
console.log(star);
```

4.3、while循环 🔥

```
while(条件表达式){
  //循环体代码
}
```

执行思路：

- 先执行条件表达式，如果结果为 true，则执行循环体代码；如果为 false，则退出循环，执行后面代码
- 执行循环体代码
- 循环体代码执行完毕后，程序会继续判断执行条件表达式，如条件仍为true，则会继续执行循环体，直到循环条件为 false 时，整个循环过程才会结束

注意：

- 使用 while 循环时一定要注意，它必须要有退出条件，否则会称为死循环
- 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

①打印人的一生

从1岁到99岁

```
var age = 0;
while (age <= 100) {
    age++;
    console.log('您今年' + age + '岁了');
}
```

②计算 1 ~ 100 之间所有整数的和

```
var figure = 1;
        var sum = 0;
        while (figure <= 100) {
            sum += figure;
            figure++;
        }
        console.log('1-100的整数和为' + sum);
```

4.4、do while循环 🔥

```
do {
  //循环体代码-条件表达式为true的时候重复执行循环一代码
}while(条件表达式);
```

执行思路：

1. 先执行一次循环体代码
2. 再执行表达式，如果结果为true，则继续执行循环体代码，如果为false，则退出循环，继续执行后面的代码
3. 先执行再判断循环体，**所以dowhile循环语句至少会执行一次循环体代码**

需求：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，一直询问

```
do {
    var love = prompt('你爱我吗？');
} while (love != '我爱你');
    alert('登录成功');
```

4.5、continue 关键字 🔥

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

```
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');
}
```

4.6、break关键字 🔥

break 关键字用于**立即跳出整个循环**

例如，吃5个包子，吃到第3个发现里面有半个虫子，其余的也不吃了

```
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');
 }

```

5、数组 🔥

数组(Array)是指一组数据的集合，其中的每个数据被称作元素，在数组中可以存放任意类型的元素。数组是一种将一组数据存储在单个变量名下的优雅方式。

```
//普通变量一次只能存储一个值
var num = 10;
//数组一次可以存储多个值
var arr =[1,2,3,4,5];
```

5.1、创建数组 🔥

JavaScript 中创建数组有两种方式：

- 利用 new 创建数组
- 利用数组字面量创建数组

①利用 new 创建数组 🔥

```
var 数组名 = new Array();
var arr = new Array(); //创建一个新的空数组
```

- 这种方式暂且了解，等学完对象再看
- 注意 `Array()`，A要大写

②利用数组字面量创建数组 🔥

```
// 1.利用数组字面量方式创建空的数组
var 数组名 =[];
// 2.使用数组字面量方式创建带初始值的数组
var 数组名 =['小白','小黑','小黄','瑞奇'];
// 3.数组中可以存放任意类型的数据，例如字符串，数字，布尔值等
var arrStus =['小白', 12,true,28.9];
```

- 数组的字面量是方括号 `[]`
- 声明数组并赋值称为数组的初始化
- 这种字面量方式也是我们以后最多使用的方式

5.2、数组的索引（下标） 🔥

索引 (下标)： 用来访问数组元素的序号（数组下标从 0 开始）

```
//定义数组
var arrStus = [1,2,3];
//获取数组中的第2个元素
alert(arrStus[1]);
```

5.3遍历数组 🔥

我们可以通过 for 循环索引遍历数组中的每一项

```
// 数组索引访问数组中的元素
var arr = ['red','green', 'blue'];
console.log(arr[0]) // red
console.log(arr[1]) // green
console.log(arr[2]) // blue

// for循环遍历数组
```

```
var arr = ['red','green', 'blue'];
for (var i = 0; i < arr.length; i++){
    console.log(arrStus[i]);
}
```

5.4、数组的长度🔥

使用“数组名.length”可以访问数组元素的数量（数组长度）

```
var arrStus = [1,2,3];
alert(arrStus.length);  // 3
```

注意：

- 此处数组的长度是**数组元素的个数**， 不要和**数组的索引号**混淆
- 当我们数组里面的元素个数发生了变化，这个 length 属性跟着一起变化

5.5、案例

1.请将 [“关羽”,“张飞”,“马超”,“赵云”,“黄忠”,“刘备”,“姜维”]; 数组里的元素依次打印到控制台

```
var arr = ["关羽","张飞","马超","赵云","黄忠","刘备","姜维"];
// 遍历  从第一个到最后一个
for(var i = 0; i < arr.length; i++ )  {
   console.log( arr[i] );
}
```

2.求数组 [2,6,1,7, 4] 里面所有元素的和以及平均值

- ①声明一个求和变量 sum。
- ①遍历这个数组，把里面每个数组元素加到 sum 里面。
- ①用求和变量 sum 除以数组的长度就可以得到数组的平均值。

```
var arr = [2, 6, 1, 7, 4];
var sum = 0;
var average = 0;
for (var i = 0; i < arr.length; i++) {
    sum += arr[i];
}
average = sum / i; //此时i为5
//      average = sum / arr.length;
console.log('和为' + sum);
console.log('平均值为' + average);
```

3.求数组[2,6,1,77,52,25,7]中的最大值

- ①声明一个保存最大元素的变量 max。
- ②默认最大值可以取数组中的第一个元素。
- ③遍历这个数组，把里面每个数组元素和 max 相比较。
- ④如果这个数组元素大于max 就把这个数组元素存到 max 里面，否则继续下一轮比较。
- ⑤最后输出这个 max。

```
 var arr = [2, 6, 1, 77, 52, 25, 7];
        var max = arr[0];
        var temp;
        for (var i = 0; i < arr.length; i++) {
            if (max < arr[i]) {
                temp = max;
                max = arr[i];
                arr[i] = temp;
            }
        }
        console.log('最大值为' + max);


方法二：

```

```
var arrNum = [2,6,1,77,52,25,7];
var maxNum = arrNum[0]; // 用来保存最大元素,默认最大值是数组中的第一个元素
// 从0 开始循环数组里的每个元素
for(var i = 0;i< arrNum.length; i++){
    // 如果数组里当前循环的元素大于 maxNum，则保存这个元素和下标
    if(arrNum[i] > maxNum){
        maxNum = arrNum[i]; // 保存数值到变量 maxNum
    }
}

```

4.将数组 [‘red’, ‘green’, ‘blue’, ‘pink’] 里面的元素转换为字符串

思路：就是把里面的元素相加就好了，但是注意保证是字符相加

- ①需要一个新变量 str 用于存放转换完的字符串。
- ②遍历原来的数组，分别把里面数据取出来，加到字符串变量 str 里面。

```
var arr = ['red','green','blue','pink'];
var str ='';
for(var i = 0; i < arr.length; i++){
    str += arr[i];
}
console.log(str);
// redgreenbluepink
```

5.将数组 [‘red’, ‘green’, ‘blue’, ‘pink’] 转换为字符串，并且用 | 或其他符号分割

- ①需要一个新变量用于存放转换完的字符串 str。
- ①遍历原来的数组，分别把里面数据取出来，加到字符串里面。
- ①同时在后面多加一个分隔符。

```
var arr = ['red', 'green', 'blue', 'pink'];
var str = '';
var separator = '|';
for (var i = 0; i < arr.length; i++) {
   str += arr[i] + separator;
}
console.log(str);
// red|green|blue|pink
```

5.6、数组中新增元素 🔥

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

```
var arr = ['red', 'green', 'blue', 'pink'];
arr.length = 7;
console.log(arr);
console.log(arr[4]);
console.log(arr[5]);
console.log(arr[6]);
```

其中索引号是 4，5，6 的空间没有给值，就是声明变量未给值，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
- 不能直接给数组名赋值，否则会覆盖掉以前的数据
- 这种方式也是我们最常用的一种方式

```
var arr = ['red', 'green', 'blue', 'pink'];
arr[4] = 'hotpink';
console.log(arr);
```

5.7、数组中新增元素

1.新建一个数组，里面存放10个整数（ 1~10）， 要求使用循环追加的方式输出： [1,2,3,4,5,6,7,8,9,10]

- ①使用循环来追加数组。
- ②声明一个空数组 arr。
- ③循环中的计数器 i 可以作为数组元素存入。
- 由于数组的索引号是从0开始的， 因此计数器从 0 开始更合适，存入的数组元素要+1。

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

2.将数组 [2, 0, 6, 1, 77, 0, 52, 0, 25, 7] 中大于等于 10 的元素选出来，放入新数组

- ①声明一个新的数组用于存放新数据。
- ②遍历原来的数组，找出大于等于 10 的元素。
- ③依次追加给新数组 newArr。

实现代码1：

```
var arr = [2, 0, 6, 1, 77, 0, 52, 0, 25, 7];
var newArr = [];
// 定义一个变量 用来计算 新数组的索引号
var j = 0;
for (var i = 0; i < arr.length; i++) {
    if (arr[i] >= 10) {
        // 给新数组
        newArr[j] = arr[i];
        // 索引号 不断自加
        j++;
    }
}
console.log(newArr);
```

实现代码2：

```
var arr = [2, 0, 6, 1, 77, 0, 52, 0, 25, 7];
var newArr = [];
for (var i = 0; i < arr.length; i++) {
    if (arr[i] >= 10) {
        // 给新数组
        newArr[newArr.length] = arr[i];
    }
}
console.log(newArr);
```

5.8、删除指定数组元素 🔥

将数组[2, 0, 6, 1, 77, 0, 52, 0, 25, 7]中的 0 去掉后，形成一个不包含 0 的新数组。

```
var arr = [2, 0, 6, 1, 77, 0, 52, 0, 25, 7];
var newArr = [];
for(var i = 0; i <arr.length; i++){
    if(arr[i] != 0){
        newArr[newArr.length] = arr[i];
    }
}
console.log(newArr);
```

```
//老师代码
var arr = [2, 0, 6, 1, 77, 0, 52, 0, 25, 7];
var newArr = [];   // 空数组的默认的长度为 0
// 定义一个变量 i 用来计算新数组的索引号
for (var i = 0; i < arr.length; i++) {
    // 找出大于 10 的数
```

```
    if (arr[i] != 0) {
        // 给新数组
        // 每次存入一个值，newArr长度都会 +1
        newArr[newArr.length] = arr[i];
    }
}
console.log(newArr);

```

5.9、翻转数组 🔥

将数组 [‘red’, ‘green’, ‘blue’, ‘pink’, ‘purple’] 的内容反过来存放

```
// 把旧数组索引号的第4个取过来(arr.length - 1),给新数组索引号第0个元素(newArr.length)

var arr = ['red','green','blue','pink','purple'];
var newArr = [];
for (var i = arr.length -1; i>=0; i--){
    newArr[newArr.length] = arr[i];
}
console.log(newArr);
```

复制

5.10、数组排序 🔥

冒泡排序

将数组 [5, 4, 3, 2, 1]中的元素按照从小到大的顺序排序，输出： 1，2，3，4，5

```
var arr = [5,4,3,2,1];
for (var i = 0; i < arr.length-1; i++){ //外层循环管趟数，5个数共交换4躺
    for (var j = 0; j <= arr.length - i - 1; j++){
        //里层循环管每一趟交换的次数
        //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);
```