相对路径：从代码所在的这个文件出发

绝对路径：D:\web\img\logo.gif

当所有网页引用同一个文件时，所使用的路径都是一样的

/与\

块级元素 div

宽度默认是容器（父级宽度）的100%

行内元素 span

（2）高、宽直接设置是无效的。

（3）默认宽度就是它本身内容的宽度。

（4）**行内元素只能容纳文本或则其他行内元素。**

行内块元素 <img />、<input />

<div class="pink fontWeight font20"> 多类名

float属性会改变元素display属性。

任何元素都可以浮动。浮动元素会生成一个块级框，而不论它本身是何种元素

**相对定位**是元素**相对**于它原来在标准流中的位置来说的。就算是相对上一个同级元素

相对于 自己原来在标准流中位置来移动的

原来**在标准流的区域继续占有**，后面的盒子仍然以标准流的方式对待它。

**绝对定位**是元素以带有定位的父级元素来移动位置

1. **完全脱标** —— 完全不占位置；
2. **父元素没有定位**，则以**浏览器**为准定位（Document 文档）。
3. **父元素要有定位**

将元素依据最近的已经定位（绝对、固定或相对定位）的父元素（祖先）进行定位。

#### 属性选择器

<a title="baidu" href="https://www.baidu.com/">baidu</a>

[title="baidu"] {

color: red;

}

#### 后代选择器

.box li {

color: coral;

}box下的所有Li，是他的孙子也变

#### 子选择器

.box > li {

color: coral;

}只变box的儿子，不变孙子

#### 相邻兄弟选择器

/\* 与p标签下面的第一个li标签 \*/

p + li

#### 伪类选择器

链接在没有点击的时候有个样式，在鼠标放上去有另外的样式，还有在点击完成以后又会又一个样式

**给既存的元素模拟新添加一个类来实现某种效果**

* :link 选取未访问过的超链接元素。
* :visited 选取访问过的超链接元素
* :hover 选取鼠标悬停的元素
* :active 选取点中的元素
* :focus，选取获得焦点的元素

.box {

display: flex;

flex-direction: column; /\*设置主轴的方向\*/

flex-wrap: nowrap; /\* 设置子元素是否可以换行 \*/

flex-flow: column nowrap; /\* flex-direction和flex-wrap属性的简写 \*/

justify-content: center; /\* 规定主轴上的对齐方式 \*/

align-items: center; /\* 规定交叉轴上的对齐方式\*/

align-content: center; /\* 规定出现多根轴线时的对齐方式 仅有一根轴线时该属性无效\*/

}

**旋转rotate**、**扭曲skew**、**缩放scale**和**移动translate**

scale

缩放，相对原来元素

ScaleX（2）相对原来的元素在X轴放大两倍

transform-origin

变换原点

##### var的声明提升

使用var时，变量的声明会自动提升到作用域（函数）的顶部。并不是全局！

提升的只有变量，而没有赋值操作

function foo(){

console.log(age); //undefined

var age = 21;

}

**let声明的范围是块级作用域，而var声明范围是函数作用域。**

let声明的变量被限制在了if块内，而var没有这样的限制

Let声明只在函数块内有效

Var可以到函数外面

**使用let不能重复声明同一变量，而var可以。**

使用let在不同作用域下声明相同名称的变量不会报错

**let创建的变量不会加在window对象身上，而var会**

var name = 'Billy';

let age = 21;

console.log (window.name); //Billy

console.log (window.age); //undefined

const和let基本相同，有一个重要的区别就是变量必须初始化

并且修改使用const声明的变量的值会报错(只能声明一次)

使用const声明的变量一定是个常量，常量的值不可修改。

**原始值**和**引用值**

原始值就是简单的数据（基本数据类型）let num1 = 1；

而引用值就是由多个值构成的对象（引用数据类型）let num2 = num1;

原始值会被复制到新变量的位置

原始类型（字符串，数字，布尔值等）是复制值

对象是共享在内存中的地址

如果属性是基本类型，拷贝的就是基本类型的值，如果属性是引用类型，拷贝的就是内存地址 ，所以如果其中一个对象改变了这个地址，就会影响到另一个对象。

将一个对象从内存中完整的拷贝一份出来,从堆内存中开辟一个新的区域存放新对象,且修改新对象不会影响原对象

我们可以将对象变量（例如 user）想象成一张带有地址的纸。

当我们对对象执行操作时，例如获取一个属性 user.name，JavaScript 引擎将对纸上的地址进行搜索，并在实际对象上执行操作。

一个变量的引用值赋值给另一变量时，存储在变量中的值也会被复制到新变量所在的位置

let obj2 = obj1; 它们两个在内存里指向同一个对象

两个变量共享一个对象，当向对象中添加name属性时，object发生更新，因此两个指向同一个object的变量同时同步更新。

对象的指针？

**函数的参数都是按值传递的**

函数外的值会被复制到函数内部的参数中

只传值，函数里面的变化就不会影响到外面声明好的值

如果按引用传（把值在内存中的位置会保存在形参之中），函数改变了值就会影响原有的值

**内层作用域可以访问外层作用域的变量，外层作用域不能访问内层作用域。**

作用域的最主要作用就是**隔离变量**

变量的！

**全局作用域**

最外层函数自身 和 在最外层函数外面定义的变量

没有声明就直接赋值的变量

**函数作用域**

声明在函数内部的变量的作用范围

**块级作用域**。

块级作用域使用大括号创建的作用域 {}，使用let和const声明的变量在块级作用域中声明，外层仍然也无法访问

{

let a = 'test';

}

console.log (a); // Uncaught ReferenceError: a is not defined





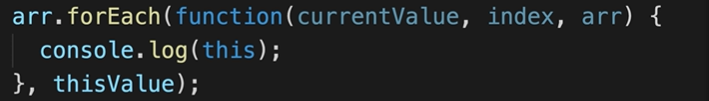
obj.gender = 'male'; 增加对象属性



需要使用非字符串的数据作为属性的话，就只能选择中括号语法。中括号中可以传任何类型的数据，甚至传递一个对象都是合法的。因为传递的数据会被toString()方法进行转换。

const n = null;

console.log (typeof n); //object （版本遗留问题）

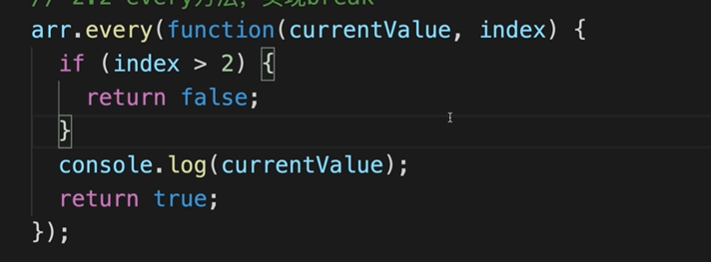


不支持break(跳出整个循环)和continue（跳出这一次循环，继续进行下一个循环）

用return间接实现break的效果

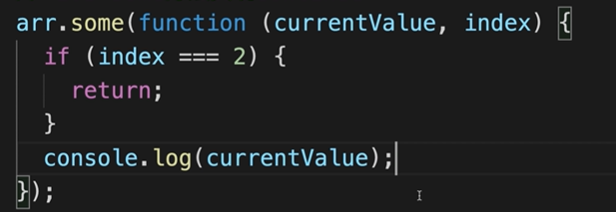
Every方法通过判断是否满足条件来遍历数组，要设true和false

测试一个数组内的所有元素是否都能通过某个指定函数的测试。它返回一个布尔值。



Some方法通过判断条件来遍历数组

方法测试数组中是不是至少有1个元素通过了被提供的函数测试。它返回的是一个Boolean类型的值。



Map方法

其结果是该数组中的每个元素是**调用一次提供的函数**后的返回值

返回一个新数组

数据类型转换

转换为字符串

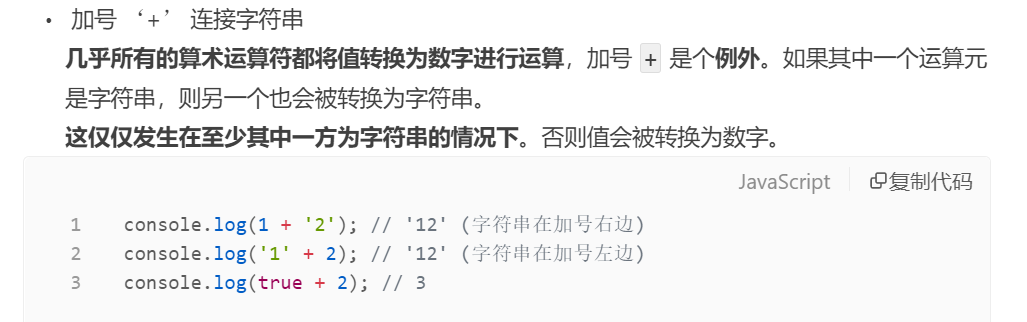
显式的类型转换  **String(value)**

隐式的类型转换  **value+''**

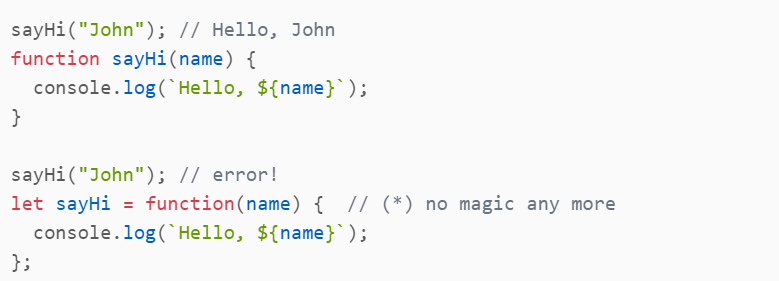
转换为数字

* 显示转换  **Number(value) 和 parseInt(string[, redix])**





函数声明和函数表达式



函数表达式是在let后才创建函数

**而函数声明被定义之前，它就可以被调用。**

一个全局函数声明对整个脚本来说都是可见的，无论它被写在这个脚本的哪个位置。

当JavaScript**准备**运行脚本时，首先会在脚本中寻找全局函数声明，并创建这些函数。在处理完所有函数声明后，才会开始执行代码。

arguments 是一个特殊的**类数组对象**，该对象按参数索引包含所有参数。

Rest 参数可以通过使用三个点 ... 并在后面跟着包含剩余参数的数组名称，来将它们包含在函数定义中。这些点的字面意思是“将剩余参数收集到一个数组中”。

当在函数调用中使用 ...arr 时，它会把可迭代对象 arr “展开”到参数列表中。

合并数组

console.log([...arr1, ...arr2, ...arr3]); // [ 'a', 'b', 'c', 'd', 'e' ]

console.log([...arr1, ...arr2, ...arr3, 'f']); // [ 'a', 'b', 'c', 'd', 'e', 'f']

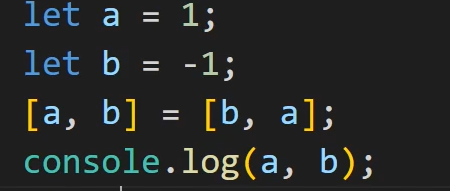
从数组和对象中提取值，对变量进行赋值，这被称为解构

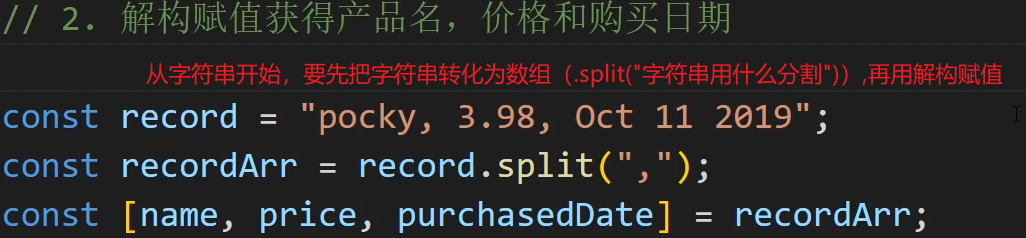


Const {bilibili:bili}= Person2.social



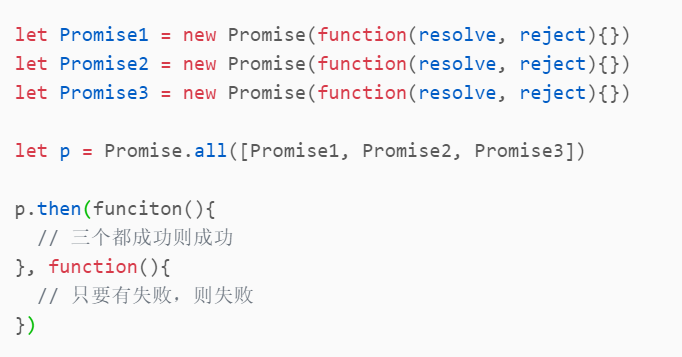
可以给变量一个默认值，在后端找不到时会返回这个默认值

用解构赋值互换变量



每一次then的执行中参数的data都为上一次异步函数执行的返回值

All方法



在一个回调中处理所有的返回数据