DAY14 this 和 ES6

2023年5月9日 17:01

this关键字

每一个函数内部都有一个关键字是this

函数内部的 this 只和函数的调用方式有关系，和函数的定义方式没有关系

解读：

this是ES里面的关键字，仅存在函数内部，是函数内部的一个特殊的对象。

随着函数调用的不同，this会发生变化，但有一个总的指向原则，谁调用了这个函数，this就指向谁。

特殊情况，没有调用对象，this指向window　var声明的变量是window的属性，函数是window下面的方法

```
// function fn() {
//   console.log(this);
// }
// fn();//特殊情况，没有调用对象，this指向window　var声明的变量是window的属性，函数是window下面的方法
// window.fn();

// document.onclick = function () {
//   console.log(this); // 指向document
// };
// var obj = {
//   name: 'zhangsan',
//   showname: function () {
//     console.log(this);//this-obj
//     console.log(this.name);//zhangsan
//   }
// }
// obj.showname();

// var list = document.querySelectorAll('li');
// for (var i = 0; i < list.length; i++) {
//   list[i].onclick = function () {
//     // console.log(i);//6 循环结束才能拥有点击事件，i是循环结束的值。
//     // console.log(this);//谁调用函数，this指向谁。
//     console.log(this.innerHTML);
//   }
// }

// var obj = {
//   age: 20,
//   obj1: {
//     age: 19,
//     showage: function () {
//       console.log(this);// obj1
//       console.log(this.age);//19
```

```
//     }
//   }
// }
// obj.obj1.showage();
```

函数对象的属性和方法

1.函数对象的属性 - length - 函数参数长度

```
// function fn(a, b, c) { }
// console.log(fn.length);//3
```

2.函数的方法

2.1.call方法是附加在函数调用后面使用，可以忽略函数本身的this指向(改变函数内部的this指向)。

call的参数:

第一个参数表示this新的指向　从第二个参数开始表示函数自身的参数。

```
// var num = 100;  // var声明的变量是window的属性
// var obj = {
//   num: 1000
// }
// function sum(n1, n2) {
//   console.log(this.num + n1 + n2);
// }
// sum(1, 2)//1->n1  2->n2  this.num = 100     103
// sum.call(obj, 2, 3);// 2->n1  3->n2  this.num = 1000  1005
```

2.2.apply方法附加在函数调用后面使用，可以忽略函数本身的this指向(改变函数内部的this指向)。

apply的参数:

第一个参数表示this新的指向　第二个参数开始表示函数自身的参数,数组表示。

```
// var num = 100;  // var声明的变量是window的属性
// var obj = {
//   num: 1000
// }
// function sum(n1, n2) {
//   console.log(this.num + n1 + n2);
// }
// sum(1, 2)//1->n1  2->n2  this.num = 100     103
// sum.apply(obj, [2, 3]);// 2->n1  3->n2  this.num = 1000  1005
```

2.3.bind方法是附加在函数调用后面使用，可以忽略函数本身的this指向(改变函数内部的this指向)。

bind的参数:

第一个参数表示this新的指向　从第二个参数开始表示函数自身的参数。

区别是bind返回的是函数体，需要再次调用

```
// var num = 100;  // var声明的变量是window的属性
// var obj = {
//   num: 1000
// }
```

```
// function sum(n1, n2) {
//   console.log(this.num + n1 + n2);
// }
// sum.bind(obj, 2, 3)();// 2->n1  3->n2  this.num = 1000  1005    格式1
// sum.bind(obj)(2, 3);// 2->n1  3->n2  this.num = 1000  1005      格式2
```

call，apply，bind的区别 - 非常重要

它们的作用是相同的，都是动态的修改当前函数内部的this的指向。

1.执行方式不同：

call和apply是改变后就立即执行函数，bind改变后不会立即执行；而是返回一个新的函数，需要再次调用。

```
// var num = 1;
// var obj = {
//   num: 10
// }
// function sum() {
//   console.log(this.num);
// }
// sum();//this.num = 1
// sum.call(obj);//this.num = 10
// sum.apply(obj);//this.num = 10
// sum.bind(obj)();//再次调用
```

2.传参方式不同：

call 第一个参数this指向，从第二个参数开始就是函数的参数。

bind 第一个参数this指向，从第二个参数开始就是函数的参数,返回的是函数体，继续通过调用可再次传入参数。

apply第一个参数this指向，第二个参数是数组，函数自身的参数放到数组里面。

```
// var num = 1;
// var obj = {
//   num: 10
// }
// function sum(a, b) {
//   console.log(this.num + a + b);
// }
// sum.call(obj, 1, 2);//this.num = 10
// sum.apply(obj, [1, 2]);//this.num = 10
// sum.bind(obj)(1, 2);//再次调用
// sum.bind(obj, 1, 2)();//再次调用
// sum.bind(obj, 1)(2);//再次调用
```

3.修改this的性质不同

call、apply只是临时的修改一次，修改就是call和apply方法使用的那一次；当再次调用原函数的时候，它的指向还是原来的指向。

bind是永久修改函数this指向，但是它修改的不是原来的函数；而是返回一个修改过后新的函数，此函数的this永远被改变了，绑定了就修改不了。

```
// function fn() {
//   console.log(this);
// }
// fn();//this->window
// fn.call(1);//this->1
// fn.call(2);//this->2 重新修改，临时修改使用的这一次
// fn.apply(2);//this->2 重新修改，临时修改使用的这一次

// var fn1 = fn.bind(3);//fn1返回的函数体里面的this不是原来的函数，是返回的新的函数里面的this是永久修改的。
// fn1();
// fn1();
// fn1();
// fn1();
// fn1.call(100);//this->3   永久修改
// fn1.apply(100);//this->3  永久修改
```

ES5和ES6

一.ES5和ES6(2015)

1.ECMAScript就是js的语法

2.我们所说的 ES5 和 ES6 其实就是在 js 语法的发展过程中的一个版本而已.

3.ES6是核心,ES6是一个里程碑。

4.ES6兼容性问题，IE6-IE9基本不识别

二.列出ES6学习的核心内容 - https://es6.ruanyifeng.com/

```
// let 和 const 命令
// 变量的解构赋值
// 字符串的扩展 - 模板字符串
// 函数的扩展 - 箭头函数
// 数组的扩展
// 对象的扩展
// 对象的新增方法
// 运算符的扩展
// Symbol
// Set 和 Map 数据结构
// Proxy
// Promise 对象
// Iterator 和 for...of 循环
// Generator 函数的语法
// async 函数
```

// Class 的基本语法
// Class 的继承
// Module 的语法
// Module 的加载实现

let和const的命令

ES6新增了let命令，用来声明变量。它的用法类似于var，但是所声明的变量，只在let命令所在的代码块内({})有效。

1.回顾变量的特点

变量的值可以改变 - let和const支持

变量是松散类型，可以直接使用而无需提前声明是什么类型 - let和const支持

变量可以同时声明多个 - let和const支持

变量写入内存 - let和const支持

变量提升 - let和const废除了 - 注意

2.let和const本身的意义

2.1.let是声明变量的关键字，它的用法类似于var

2.2.const声明一个只读的常量,一旦声明，常量的值就不能改变,常量就是必须初始化。

3.let和var的区别

3.1.let具有块作用域，声明的变量只在代码块内({})有效，var具有全局和局部作用域(函数)

let变量具有全局和块级作用域

```
// for (let j = 1; j <= 10; j++) {
//   console.log(j);
// }
// console.log(j);// j is not defined 报错
// let a = 10;//全局
// function fn() {
//   let b = 20;//块级
// }
// fn();
```

3.2.let声明的变量绑定在块作用域中。绑定就是声明的变量每一次的改变都是存在作用域中

```
// for (let i = 1; i <= 5; i++) {//同步    这里的i每一次的改变都绑定在{}里面，let本身具有的特点。
//   window.setTimeout(function () {//异步
//     console.log(i);//1,2,3,4,5
//   }, 1000);
// }

<ul>
  <li>1111</li>
  <li>2222</li>
```

```
    <li>3333</li>
    <li>4444</li>
  </ul>

  // var list = document.querySelectorAll('li');
  // for (var i = 0; i < list.length; i++) {
  //   list[i].onclick = function () {
  //     console.log(i);//4 循环的最后一次值，循环不完成，不具有点击事件，只要能够点击，说明循环结束了
  //   }
  // }

  // for (let i = 0; i < list.length; i++) {
  //   list[i].onclick = function () {
  //     console.log(i);//0,1,2,3   绑定
  //   }
  // }
```

3.3.let不允许在相同的作用域内重复声明。

```
  // let num = 1;
  // let num = 10;
  // console.log(num);//报错

  // let num = 10;//全局
  // function fn() {
  //   let num = 100;//块级
  // }
  // fn();
```

3.4.let不存在变量提升(暂时性死区)

在代码块内，使用let命令声明变量之前，该变量都是不可用的。这在语法上，称为“暂时性死区”

```
  // console.log(b);//报错
  // let b = 10;
  // console.log(b);
```

4.const关键字

const和let一样，没有变量提升，相同的作用域不能重复声明，具有块级作用域，声明的常量绑定在代码块区域内

区别是const声明的值不能改变

很多人叫const为声明一个不能改变的变量

```
  // const num = 1;
  // num = 10;//报错
```

特殊情况

对于数组对象来说，只要不改变地址就可以，里面的值可以随意改变，因为对象采用的是引用传递，不改

变地址就没有问题。

```
// const arr = [1, 2, 3, 4, 5];
// arr[0] = 100;
// console.log(arr);//[100, 2, 3, 4, 5]
// arr = [];//这样才会报错
```

箭头函数

1.箭头函数是函数的一种简洁的表示方式。

1.1.省略的function,大部分情况下都是省略function

1.2.函数的行参只有一个的时候可以不写参数()其余情况必须写

1.3.函数体只有一行代码的时候，可以不写 {} ，并且会自动return

```
// const fn = function (str) {
//   return str;
// }
// console.log(fn('hello'));

// const fn = str => str;//最简单的箭头函数
// console.log(fn('hehe'));
// window.setTimeout(()=>console.log('我是定时器里面的函数'),1000);
// const sum = function(n1,n2){
//   n1 = n1+n2;
//   n2 = n1+n2;
//   return n1+n2;
// }

// const sum = (n1,n2)=>{
//   n1 = n1+n2;
//   n2 = n1+n2;
//   return n1+n2;
// }
```

2.因为箭头函数是匿名函数，没有arguments,不能使用new关键字调用

3.箭头函数内部没有自己的this，而且里面this不能被改变，箭头函数的this来自父级，如果没有父级，指向window.

```
// 案例：点击对应的li元素延迟2s将里面的内容改变
// var list = document.querySelectorAll('li');
// for (let i = 0; i < list.length; i++) {
//   list[i].onclick = function () {
//     // this:当前的li
//     window.setTimeout(() => {
//       this.innerHTML = '修改后的内容'; //这里的this来自于父级，无法改变
//     }, 2000);
//   }
// }
```

4.箭头函数不适合复杂的逻辑,一般用来取代回调函数(函数做参数),箭头函数不适合在最外层，因为this指

向window(污染window)

```
const arr = ['zhangsan', 'lisi', 'wangwu'];
arr.forEach(function (item) {
  console.log(item);
})
arr.forEach(item => console.log(item))
setInterval(() => {}, 200);
```

解构赋值

解构赋值，就是快速的从对象或者数组中取出成员的一个语法方式

1.数组解构

```
// 案例
// const arr = ['zhangsan', 'lisi', 'wangwu'];
// const [a, b, c] = arr; //const [a, b, c] = ['zhangsan', 'lisi', 'wangwu']
// console.log(a, b, c);//zhangsan lisi wangwu
// 案例
// const arr = [1, 2, [3, 4, [5, 6, [7, 8, [9, 10]]]]];
// const [a, b, [c, d, [e, f, [g, h, [i, j]]]]] = arr;
// console.log(a, b, c, d, e, f, g, h, i, j);//1 2 3 4 5 6 7 8 9 10
```

2.对象的解构 - 核心

```
// const obj = {
//   name: 'zhangsan',
//   age: 18,
//   sex: '男'
// };
```

解构的变量名必须和对象的属性名相同(也可以利用语法进行修改)

顺序可以自由调整，对象是无序的。 6

```
// const { sex, age, name } = obj;
// console.log(name, age, sex);

// const { name } = obj;
// console.log(name);

// const { a, b, c } = obj;
// console.log(a, b, c);//undefined undefined undefined  名称不对应
```

利用语法进行修改得到的变量名称

```
const { name: a, age: b, sex: c } = obj;
console.log(a, b, c);//zhangsan 18 男
```

模版字符串

模板字符串是增强版的字符串，用反引号（`）标识。

它可以当作普通字符串使用，也可以用来定义多行字符串，或者在字符串中嵌入变量。

在 `` 里面的 `${}` 就是用来书写变量的位置，包括JavaScript表达式

```
// var time = prompt('请输入一个小时数：');
// var day = parseInt(time / 24);//记录天数
// var hour = time % 24;//记录小时数
// console.log('为抵抗洪水，战士连续作战' + time + '小时，编程计算共' + day + '天零' + hour + '小时');
// console.log(`为抵抗洪水，战士连续作战${time}小时，编程计算共${day}天零${hour}小时`);
```

展开运算符

一.展开运算符

ES6里面新添加了一个运算符 ... ，叫做展开运算符(扩展运算符)

作用是把数组或者类数组展开

1.数组合并

```
// const arr1 = [1, 2, 3];
// const arr2 = [4, 5, 6];
// const arr3 = [7, 8, 9];
// console.log(arr1.concat(arr2, arr3));//[1, 2, 3, 4, 5, 6, 7, 8, 9]
// console.log([...arr1, ...arr2, ...arr3]);//[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

2.求数组项最大值

```
// const arr = [1, 4, 7, 2, 5, 8, 9, 6, 3];
// console.log(...arr);//1 4 7 2 5 8 9 6 3
// console.log(Math.max(...arr));//9
```

3.引用传递

```
// let arr1 = [1, 2, 3];
// let arr2 = arr1;//引用传递，赋值的是地址
// arr2.push(4);
// console.log(arr1);//[1, 2, 3, 4]
// console.log(arr2);//[1, 2, 3, 4]

// let arr1 = [1, 2, 3];
// let arr2 = [...arr1];//新建数组，数组项是arr1的值
// arr2.push(4);
// console.log(arr1);//[1, 2, 3]
// console.log(arr2);//[1, 2, 3, 4]
```

4.**重点**，将类数组转换成真正的数组。

```
// let list = document.querySelectorAll('li');//类数组
// list.push(document.body);//将body元素push进类数组，报错。
// list = [...list]// 将类数组转换成真正的数组
// list.push(document.body);
// console.log(list);// [li, li, li, body]
```

二.rest参数

ES6 引入rest参数（形式为...变量名），用于获取函数的多余参数，这样就不需要使用arguments对象了。

rest 参数搭配的变量是一个数组，该变量将多余的参数放入数组中。

注意：rest只能作为最后的一个参数

```
// function sum(...num) {
//   // console.log(arguments.length);//100
//   // console.log(arguments[0]);//1
//   console.log(num);
//   console.log(num.length);
//   console.log(num[0]);
// }

// function sum(a, b, ...c) {
//   console.log(a);//1
//   console.log(b);//2
//   console.log(c);//[3, 4, 5, 6, 7, 8, 9, 10]
// }

// sum(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

数组和对象的扩展

一.数组的扩展

1.Array.from()方法用于将对象转为真正的数组(类数组转数组)

1.1.将对象转为真正的数组,对象的属性名是数字(索引)，必须具有length属性表示数组的长度

```
// const obj = {
//   0: 'zhangsan',
//   1: 'lisi',
//   2: 'wangwu',
//   length: 3
// }
// console.log(Array.from(obj));//['zhangsan', 'lisi', 'wangwu']

// const obj = {
//   length: 1000
// }
// console.log(Array.from(obj));
```

1.2.类数组转数组

```
// let list = document.querySelectorAll('li');
// list = Array.from(list);
// list.push(document.body);
// console.log(list);// [li, li, li, body]
```

// 案例：三种方式实现类数组转换成真正的数组。

// 1.展开运算符

// 2.Array.from方法
// 3.新建空数组，遍历类数组，将值逐个push进新数组。

// 梳理学过的类数组
// arguments
// 字符串
// 获取多个元素对象(querySelectorAll,getElementsByTagName,getElementsByClassName)
// childNodes
// children
// ......

2.includes方法：判断数组项是否存在，存在返回true，不存在返回false

```
// const arr = ['zhangsan', 'lisi', 'wangwu']
// console.log(arr.includes('lisi'));//true
// console.log(arr.includes('lisi123'));//false
```

3.fill方法：填充数组

arr.fill('hello', 1, 2);

第一个参数就是填充的内容，后面的两个参数就是索引位置

```
// const arr = ['zhangsan', 'lisi', 'wangwu']
// arr.fill('hello', 1, 2);//第一个参数就是填充的内容，后面的两个参数就是索引位置
// console.log(arr);//['zhangsan', 'hello', 'wangwu']
```

二.对象的扩展

1.对象的简洁表示法

属性名和属性值名称相同，写一个

函数省略functon

```
// let username = 'zhangsan';
// const obj = {
//   username: username, //第一个username自定义属性名    第二个username是上面的遍历
//   showname: function () {
//     console.log(this.username);
//   }
// }
// 简化后 - 重点
// const obj = {
//   username,
//   showname() {
//     console.log(this.username);
//   }
// }
// obj.showname();
```

2.对象扩展的方法

2.1.Object.keys()获取对象所有的key(属性)集合，返回一个数组。

2.2.Object.values()获取对象所有的value(属性值)集合，返回一个数组。

2.3.Object.assign()用于对象的合并，将源对象的所有可枚举属性，复制到目标对象。

```
// const obj1 = {
//   a: 1,
//   b: 2,
//   c: 3
// };
// const obj2 = {
//   c: 4,
//   d: 5,
//   e: 6
// };
// const obj3 = {
//   d: 7,
//   e: 8,
//   f: 9
// };
// const obj = Object.assign(obj1, obj2, obj3);//合并对象，如果属性名相同，后面会覆盖前面
// console.log(obj);
```

注意：合并后的对象集中到第一个对象上面,最终的返回值也是合并后的结果

```
// Object.assign(obj1, obj2, obj3);
// console.log(obj1);
// console.log(Object.assign({}, obj1, obj2, obj3));
```

2.4.Object.entries()返回一个给定对象自身可枚举属性的键值对数组

```
// const obj = {
//   name: 'zhangsan',
//   age: 18,
//   sex: '男'
// }
// console.log(Object.keys(obj));//['name', 'age', 'sex']
// console.log(Object.values(obj));//['zhangsan', 18, '男']
// console.log(Object.entries(obj));//[['name', 'zhangsan'],['age', 18],['sex', '男']]
```

Set和Map结构

1.set结构

ES6提供了数据结构Set. 它类似于数组，但是成员的值都是唯一的，没有重复的值。

Set本身是构造函数，用来生成Set数据结构，数组作为参数。

```
// let arr = [1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5];
// let s1 = new Set(arr);//利用set构造函数生成set对象  s1是set对象
// console.log(s1);//{1, 2, 3, 4, 5}

// 案例：利用set进行数组去重，最快的方式
// let arr = [1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5];
// console.log(Array.from(new Set(arr)));//[1, 2, 3, 4, 5]
```

```
// console.log([...new Set(arr)]);// [1, 2, 3, 4, 5]
```

set结构属性和方法

```
// let s1 = new Set(['zhangsan', 'lisi', 'wangwu']);
```

size:返回set结构的长度

```
// console.log(s1.size);//3
```

Set.add(value) 添加一个数据，返回Set结构本身，允许进行链式操作。

```
// s1.add('zhaoliu').add('sunqi').add('wangba');
// console.log(s1);
```

Set.delete(value) 删除指定数据，返回一个布尔值，表示删除是否成功。

```
// s1.delete('zhaoliu');
// console.log(s1);
```

Set.has(value) 判断该值是否为Set的成员，返回一个布尔值。

```
// console.log(s1.has('wangwu'));//true
// console.log(s1.has('wangwu123'));//false
```

Set.clear() 清除所有的数据，没有返回值。

```
// s1.clear();
// console.log(s1);
```

forEach()：使用回调函数遍历每个成员

注意：每一项即是键也是值,键和值是相同的。

```
// s1.forEach((item, index, set) => {//注意：每一项即是键也是值,键和值是相同的。
//   console.log(item);
//   console.log(index);
// });
```

二.Map

ES6提供了Map数据结构

JavaScript的对象（Object），只能用字符串当作键，这给它的使用带来了很大的限制。

它类似于对象，但是“键”的范围不限于字符串，各种类型的值都可以当作键。

Map本身也是构造函数，用来生成map数据结构，二维数组做参数。

```
const m1 = new Map([
  ['name', 'zhangsan'],
  [true, 100],
  [null, 200],
  [undefined, 300],
  [function () { }, 400]
]);
console.log(m1);
//{'name' => 'zhangsan', true => 100, null => 200, undefined => 300, ƒ => 400}
```

属性和方法

size属性，表示长度

```
console.log(m1.size);//5
```

map.set(key,value) :设置键名key对应的键值value,然后返回整个map结构，可以链式操作。

```
m1.set(100, 500).set([], 600)
// console.log(m1);
```

map.get(key) get方法读取key对应的键值，如果找不到key，返回undefined

```
console.log(m1.get(100));//500
console.log(m1.get([]));//undefined  []==[]比较的是地址，不会相等
console.log(m1.get(undefined));//300
```

map.delete(key) 删除某个键，返回true.如果删除失败，返回false.

```
console.log(m1.delete(null));
console.log(m1);
```

map.has(key) 方法返回一个布尔值，表示某个键是否在当前map对象之中。

```
console.log(m1.has(true));
console.log(m1.has(false));
```

map.clear() 清除所有数据，没有返回值

```
// m1.clear();
// console.log(m1);
```

map.forEach() 使用回调函数遍历每个成员。

```
// m1.forEach((item, index) => {
//   console.log(item, index);//item值  index属性名
// })
```

Symbol 和BigInt

一.ES约定的数据类型

1.基本类型：number/string/boolean/null/undefined/Symbol/BigInt

2.引用类型：Object

二.基本类型和引用类型的简单的区别

1.基本类型遵循值传递，值和地址都是存储在栈内存中

2.引用类型遵循引用传递(地址传递)，值存储在堆内存中，地址存储在栈内存中。

```
// var a = 1;
// var b = a;//值传递

// var arr1 = [1, 2, 3];
// var arr2 = arr1;//引用传递
```

三.ES6新增的数据类型 - Symbol

ES5 的对象属性名都是字符串，这容易造成属性名的冲突。

ES6 引入了一种新的原始数据类型Symbol，表示独一无二的值。

```
// let s1 = Symbol();
// let s2 = Symbol();
// console.log(s1 == s2);//false

// 括号里面可以放置字符串格式参数，仅仅为了输出时能够看到区别，没有别的意思
// let s1 = Symbol('zhangsan');
// let s2 = Symbol('lisi');
// console.log(s1);
// console.log(s2);
// console.log(s1 == s2);//false
```

对象的属性名一定是字符串格式，如果使用变量代替，必须添加中括号。

```
// let a = 'name';
// let b = 'age';
// const obj = {
//   a: 'zhangsan',//这里的a仅仅是一个字符串，和上面的变量没有关系
//   b: 18
// }
// console.log(obj.a);//zhangsan
// const obj = {
//   [a]: 'zhangsan',//这里的a就是上面的变量
//   [b]: 18
// }
// console.log(obj.a);//undefined
// console.log(obj.name);//zhangsan
// obj.name = 'wangwu';
// console.log(obj.name);//wangwu
```

为了防止冲突，采用Symbol当作对象的属性名。

```
const obj = {
  [Symbol('zhangsan')]: 'zhangsan',
  [Symbol('lisi')]: 'lisi',
  [Symbol('wangwu')]: 'wangwu'
}
```

弊端:无法单个获取,独一无二

```
console.log(obj[Symbol('zhangsan')]);//undefined
```

提供获取的方法：

有一个Object.getOwnPropertySymbols()方法，可以获取指定对象的所有 Symbol 属性名。该方法返回一个数组，成员是当前对象的所有用作属性名的 Symbol 值。

```
// console.log(Object.getOwnPropertySymbols(obj));// [Symbol(zhangsan),
Symbol(lisi), Symbol(wangwu)]
```

```
// for (let i = 0; i < Object.getOwnPropertySymbols(obj).length; i++) {
//   console.log(obj[Object.getOwnPropertySymbols(obj)[i]]);//遍历属性名获取对应的值
// }
```

四.BigInt大整数(ES11里面的)

BigInt 是一种内置对象，它提供了一种方法来表示大于 2^53 - 1的整数。

BigInt 可以表示任意大的整数。

如何定义BigInt数字

可以用在一个整数字面量后面加 n 的方式定义一个 BigInt ，如：120n，或者调用函数BigInt()。

```
// console.log(9999999999999999999999999999);//1e+28
// console.log(9999999999999999999999999999n);//9999999999999999999999999999n
```

1.BigInt进行运算

+ - * / % **

```
// console.log(2 ** 3);//8
// console.log(2n ** 3n);//8n
// console.log(3n + 5n);//8n
// console.log(5n / 2n);//2n
// console.log(5n / 2);//报错
```

2.BigInt():将普通数字转换成bigint

```
// console.log(BigInt(100));//100n
// console.log(BigInt(true));//1n
```

3.BigInt():可以和其他数字进行比较

```
// console.log(1n == 1);//true
// console.log(1n === 1);//false
// console.log(5n > 4);//true
```

重要的是要记住，不能使用Number和BigInt操作数的混合执行算术运算，需要通过显式转换其中的一种类型。

Iterator接口和for...of遍历

一.Iterator接口和for...of遍历

1.遍历器（Iterator）它是一种接口，为各种不同的数据结构提供统一的访问机制。

2.任何数据结构只要部署Iterator接口，就可以完成遍历操作（即依次处理该数据结构的所有成员）。

3.一个数据结构只要部署了Symbol.iterator属性，就可以用for...of循环遍历它的成员。

```
// let str = 'hello';// 类数组
// let arr = ['zhangsan', 'lisi', 'wangwu'];
// let obj = {
//   a: 1,
//   b: 2,
//   c: 3
// };
// let s1 = new Set(['apple', 'banana', 'orange']);
// let m1 = new Map([
```

```
//   ['name', 'zhangsan'],
//   ['age', 18],
//   ['sex', 'women']
// ]);
// console.log(str[Symbol.iterator]());//StringIterator {}
// console.log(arr[Symbol.iterator]());
// console.log(s1[Symbol.iterator]());
// console.log(m1[Symbol.iterator]());
// console.log(obj[Symbol.iterator]);//undefined  不能使用forof遍历，for...in...是转为对象的遍历而生
```

二.for...of...的语法

语法格式：for(let 自定义变量 of 对象){ } 自定义变量接收对象的每一个成员(值)

for...of...是最优秀的遍历方式

```
// for (let item of str) {//item声明一个变量接收遍历的值
//   console.log(item);
// }
// for (let item of arr) {
//   console.log(item);
//   break;
// }
// arr.forEach(item => console.log(item));//forEach没有返回值，不能终断
// for (let item of s1) {
//   console.log(item);
// }
// for (let item of m1) {
//   console.log(item);
// }
```

三.对比for,forEach,forin,forof

与其他遍历语法的比较

1.以数组为例，JavaScript 提供多种遍历语法。最原始的写法就是for循环。

2.这种写法比较麻烦，因此数组提供内置的forEach方法。

3.forEach没有返回值，不能终断，采用for...in...

4.for...in循环不仅遍历数组数字键名，还会遍历数组手动添加的其他键(不符合数组的特性)

5.for...in循环主要是为遍历对象而设计的，不适用于遍历数组

6.for...of循环有着同for...in一样的简洁语法，但是没有for...in那些缺点

7.for...of不同于forEach方法，它可以与break、continue和return配合使用

generator生成器函数

1.Generator 函数是一个普通函数，但是有两个特征。

一是，function关键字与函数名之间有一个星号；

二是，函数体内部使用yield表达式，定义不同的内部状态（yield在英语里的意思就是“产出”）。

```
function* fn() {
```

```
}
console.log(fn());//返回的是状态
```

2.首先可以把它理解成，Generator 函数是一个状态机，封装了多个内部状态。

```
// function* fn() {
//   yield '第一个状态';
//   yield '第二个状态';
//   yield '第三个状态';
// }
// let f = fn();//返回的是状态机，f就是遍历器对象
```

必须调用遍历器对象的next方法，才能查看里面对应的状态，
每次调用next方法，内部指针就从函数头部或上一次停下来的地方开始执行，
直到遇到下一个yield表达式（或return语句）为止。

```
// console.log(f.next());//{value: '第一个状态', done: false}
// console.log(f.next());//{value: '第二个状态', done: false}
// console.log(f.next());//{value: '第三个状态', done: false}
// console.log(f.next());//{value: undefined, done: true}
```

调用next方法，返回一个对象，对象里面包含value和done两个属性。
value属性表示当前的内部状态的值，是yield表达式后面那个表达式的值；
done属性是一个布尔值，表示状态是否遍历结束。
Generator 函数是分段执行的，yield表达式是暂停执行的标记，而next方法可以恢复执行。

```
// function* gen() {
//   yield 'hello';
//   yield 'world';
//   return 'ending';//最后的状态，或者结束
// }
// let g = gen();//返回遍历器对象
// console.log(g.next());//{value:hello,done:false}
// console.log(g.next());//{value:world,done:false}
// console.log(g.next());//{value:ending,done:true}
```

利用forof遍历拿到每一个状态的值，yield后面的值

```
// for (let item of g) {
//   console.log(item);//hello world
// }
// 获取每一个状态的值
// console.log(...g);//hello world
```

模块化

一.前面所有的演示为了凸显两个问题

1.产生冲突(代码之间)

2.无法根据文件看出依赖关系(无法一眼识别那个js文件引用了那个js文件里面的代码)

二.模块化的意义 - 冲突依赖

1.ES6 在语言标准的层面上，实现了模块功能，而且实现得相当简单，完全可以取代 CommonJS 和 AMD 规范，成为浏览器和服务器通用的模块解决方案。

2.一个js文件就是一个模块(来自于规范)

3.定义模块，调用模块，配置模块.

三.导出模块(定义模块)

export命令用于规定模块的对外接口(本质的意义就是通过export的命令将模块里面的代码导出)

重点：

一个模块就是一个独立的文件,该文件内部的所有变量，外部无法获取。

如果你希望外部能够读取模块内部的某个变量，就必须使用export关键字输出该变量。

export default 暴露一个变量或者函数，调用的时候自定义名称，

一个模块只能有一个默认输出，因此export default命令只能使用一次。

四.导入模块(调用模块)

可以通过import命令加载这个模块

格式： import {导入的模块暴露的对象} from './模块的名称(包含扩展名)'

import {double, ranNum, obj} from './module/definemodule1.js'

调用模块的时候，可以使用as关键字修改模块的名称。

五.配置模块

配置模块指的是导入模块时候必须添加 ./加载模块的路径

./的目的是告知当前的模块是本地模块，不是第三方的模块。

六.HTML页面加载规则

1.浏览器加载 ES6 模块，也使用script标签，但是要加入type="module"属性。

2.浏览器对于带有type="module"的script都是异步加载，不会造成堵塞浏览器

3.即等到整个页面渲染完，再执行模块脚本，等同于打开了script标签的defer(异步)属性。

4.如果网页有多个script type="module"，它们会按照在页面出现的顺序依次执行。

注意：模块化开发必须通过静态服务器访问

// vscode 按照插件 liveSever

七.注意事项

1.修改导入的模块名称 利用as关键字

2.如果仅导出一块代码，可以在导出的时候换一种方式导出方式(export defualt),调用的时候自定义名称