爱创课堂前端培训

# ES6

第2天课堂笔记（本课程共3天）

班级：北京前端训练营19期

讲师：李兰波

日期：2019年2月17日

爱创课堂官网 ：[www.icketang.com](http://www.icketang.com)

# 复习

ES6: ECMAScript的第六个大版本 也叫做ECMAScript2015 从该版本开始，ECMA规定，版本号以年号为单位

新特性：块级作用域、let、const、各个内置构造函数的新方法、多行字符串的定义方式

块级作用域：代码块现在也可以生成一个词法作用域。

let: 定义变量的关键字

通过let定义的变量遵循ES6规范 通过var定义的变量遵循以前的ES5、3的规范  
 let定义的变量具备块级作用域var定义的变量只具备函数作用域

let定义的变量没有声明提升 var定义的变量具备声明提升

let定义的变量不能够重复定义 var定义的变量可以重复定义

let定义的全局变量不能够挂载到window对象上 var定义的全局变量会自动挂载到window上作为一个属性

let定义的变量在for循环中可以直接使用无需担心闭包的现象 var定义的变量在for循环中作为循环变量时，如果循环体内定义了函数，并且函数还使用到循环变量。则不论哪个函数执行，都统一是循环变量的最大值

const: 定义常量的关键字

常量：静态变量 一旦被定义 只能被访问 不能被=修改

命名规范：全大写 单词之间使用下划线分开 const PI = 3.14; const ICKT\_ADDRESS = "北京市昌平区北七家镇白庙村平西王府古玩城2楼C30"

注意：当常量保存的是引用类型数据时，可以通过.语法和[]语法给地址新增、修改以及删除属性 但是不可以使用=改变常量的地址

ES3中模拟常量：使用闭包

ES5中模拟常量：使用对象的冻结方法去模拟

字符串的新方法：

startsWith: 判定一个字符串是否是以另外一个字符串开头

str: 开头字符串

pos: 偏移量 默认是0

结果：判定调用startsWith方法的字符串是否是从pos位置以str开始

endsWith: 判断一个字符串是否是以另外一个字符串结尾

str: 结尾字符串

pos: 偏移量 默认是原字符串的长度

结果：判定调用startsWith方法的字符串是否是以pos位置以str结尾

includes: 判定一个字符串是否包含另外一个字符串

str: 被包含的字符串

pos: 偏移量 默认是0

结果：判定调用includes方法的字符串是否包含str 注：从pos位置开始往后截取

repeat：重复某一个字符串

num: 被重复的次数

返回值：被重复了num次之后的新字符串

字符串的新定义方式：

单行字符串： 可以使用""、''进行定义

多行字符串： 使用``进行定义 里面可以包含回车及换行等内容

插值语法：

在多行字符串中可以进行插值语法： ${}

let username = "wanglaowu";

let str = `<div>${username.toUpperCase()}</div>`;

最终得到的字符串str是: <div>WANGLAOWU</div>

简单理解：${}等价于我们之前自己定义的<%%> 只是ES6给内置了 而且里面可以调用JS的方法 因为${}之间是一个JS执行环境

数组方法：

Array.of: 该方法用于定义数组 参数就是数组成员

Array.from: 该方法用于将类数组对象转化成数组

类数组对象：拥有length属性、拥有下标属性的对象

find: 该方法用于模糊查询一个成员

参数是一个函数

当调用find方法时，其实会依次将数组的每一个成员和下标代入到该函数中执行 该函数如果返回true则停止执行 如果返回false则继续执行。

最终的返回值就是符合函数描述的成员

findIndex: 该方法用于模糊查询一个成员的下标

参数是一个函数

当调用find方法时，其实会依次将数组的每一个成员和下标代入到该函数中执行 该函数如果返回true则停止执行 如果返回false则继续执行。

最终的返回值就是符合函数描述的成员的下标

copyWithin: 该方法用于数组的内部复制

pos: 粘贴的开始位置(包含)

start: 复制的开始位置(包含)

end: 复制的结束位置(不包含)

返回值：从start位置开始复制 复制到end位置 得到一个数组 再将该数组中的成员一个一个从pos位置开始粘贴 最终改变原数组 返回原数组

# 对象的字面量改变

## 1.1 定义属性

当属性名与属性值写法一致时，可以省略冒号与属性值。

demo:

|  |
| --- |
| 1. // 定义一个变量 2. var username = "wanglaowu"; 3. // 再定义一个变量 4. var password = "password"; 5. // 以前的定义属性的方式 6. var obj = { 7. username: username, 8. password: password 9. }; 10. // 现在的定义方式 11. var obj1 = { 12. username, 13. password 14. } 15. // 输出 16. console.log(obj, "obj"); 17. console.log(obj1, "obj1"); |

console:

|  |
| --- |
|  |

## 1.2 定义属性名

现在定义属性名称时，可以使用[]语法

|  |
| --- |
| 1. var baaaaa = "baaaaa"; 2. // 定义属性名 3. var obj = { 4. ["a" + baaaaa]: "ssss" 5. } |

输出：

|  |
| --- |
|  |

## 1.3 定义方法

在定义方法时，可以省略: function

demo:

|  |
| --- |
| 1. var obj = { 2. hello: function() { 3. console.log("hello") 4. }, 5. hi() { 6. console.log("hi") 7. } 8. } |

console:

|  |
| --- |
|  |

# 对象的拓展

## 2.1 Object.is

Object.is(one, two); 该方法用于判定两者是否全等

one: 一个值

two: 另一个值

返回值: 布尔值

demo：

|  |
| --- |
| 1. // === 判定 2. console.log(NaN === NaN); // false 3. console.log(0 === -0); // true 4. // Object.is 判定 5. console.log(Object.is(NaN, NaN)); // true 6. console.log(Object.is(0, -0)); // false |

console：

|  |
| --- |
|  |

## 2.2 Object.assign

Object.assign(obj, obj1, obj2, obj3,……); 该方法用于将第二个参数往后的所有参数的属性和方法赋予第一个参数

obj: 被赋予的对象

剩余参数: 赋予者

注：该方法是一个浅复制方法

demo:

|  |
| --- |
| 1. // 定义一个对象 2. var obj = { 3. name: "王老五", 4. age: 30, 5. sex: "男", 6. card: { 7. type: "广发银行", 8. amount: 300000, 9. code: 123456 10. }, 11. height: 10 12. } 13. var obj2 = { 14. height: 300 15. } 16. // 定义另外一个对象 继承obj 17. var obj1 = {}; 18. // 调用Object.assign 19. Object.assign(obj1, obj, obj2); |

console:

|  |
| --- |
|  |

继承顺序：按照参数的传递顺序执行，如果有相同的属性，会以最后一个为准。

# 数字方法

## 3.1 Number.isNaN

ES6新增了一个Number.isNaN 该方法只对数字进行判定 如果参数不是数字 直接返回false

ES6之前也有一个window.isNaN 该方法会对参数尽可能的转为数字 再进行判定

demo：

|  |
| --- |
| 1. // 所谓的数字方法指的是Number身上拓展出的一些新方法 2. // Number.isNaN 该方法只对数字进行判定 3. // 之前也有一个window.isNaN 该方法的特点是会尽可能的将参数向数字类型转换 转换完毕之后的结果再进行判定是否是NaN 4. var num1 = 0; 5. var num2 = "0"; 6. var num3 = "oiasjdoiewaj"; 7. var num4 = NaN; 8. // window.isNaN 9. console.log(window.isNaN(num1)); // false 10. console.log(window.isNaN(num2)); // false 11. console.log(window.isNaN(num3)); // true 12. console.log(window.isNaN(num4)); // true 13. // Number.isNaN 14. console.log(Number.isNaN(num1)); // false 15. console.log(Number.isNaN(num2)); // false 16. console.log(Number.isNaN(num3)); // false 17. console.log(Number.isNaN(num4)); // true |

## 3.2 Number.isFinite

Number.isFinite用于判定一个数字是否是有限的

window.isFinite用于判定一个值是否是有限的  
demo:

|  |
| --- |
| 1. // window.isFinite 2. console.log(window.isFinite(num1)); // true 3. console.log(window.isFinite(num2)); // true 4. console.log(window.isFinite(num3)); // false 5. console.log(window.isFinite(num4)); // false 6. // Number.isFinite 7. console.log(Number.isFinite(num1)); // true 8. console.log(Number.isFinite(num2)); // false 9. console.log(Number.isFinite(num3)); // false 10. console.log(Number.isFinite(num4)); // false |

## 3.3 Number.isInteger

该方法用于判定一个数字是否是整数

demo:

|  |
| --- |
| 1. // Number.isInteger 2. var num1 = 1.3; 3. var num2 = "1.0"; 4. var num3 = 3.0; 5. var num4 = NaN; 6. console.log(Number.isInteger(num1)); // false 7. console.log(Number.isInteger(num2)); // false 8. console.log(Number.isInteger(num3)); // true 9. console.log(Number.isInteger(num4)); // false |

# 代理

Proxy构造函数可以生成一个代理对象。

接收2个参数：被保护的对象， 处理方式对象

demo：

|  |
| --- |
| 1. var proxy = new Proxy(star, {   //get：用于拦截某个属性的读取操作，可以接收三个参数，依次为目标对象，属性名和Proxy实例本身（操作行为是针对的对象），其中最后一个参数可选。   1. get: function(star, key) { 2. // 当通过proxy访问属性的时候 其实会执行get函数 3. // console.log(arguments); 4. // 判定 5. if (key === "education") { 6. return "博士后"; 7. } 8. return star[key]; 9. }, 10. //set方法用来拦截某个属性的赋值操作，可以接受4个参数，依次为目标对象，属性名，属性值和Proxy实例本身，其中最后一个参数可选。 11. set: function(star, key, value) { 12. console.log(star, key, value); 13. if (key === "name") { 14. return; 15. } 16. star[key] = value; 17. } 18. }); |

当访问proxy对象的时候，所有的操作都会转嫁到star上。但是可以通过处理方式对象进行预处理操作。

proxy对象自身没有任何的属性和方法。

当访问proxy对象时，会执行get方法，当设置proxy对象时，会执行set方法。

# Symbol

这是一个新的数据类型。用于表示一种独一无二的符号。属于基本数据类型（值类型）。

## 5.1 定义

定义方式：通过函数 Symbol 定义 注：它不是一个构造函数 不可以使用new

|  |
| --- |
| 1. // 定义一个Symbol 2. var s1 = Symbol(); 3. console.log(s1); |

如果不小心调用了new：会报错。

|  |
| --- |
|  |

# ...语法

...语法可以用于获取剩余参数、可以用于解构、可以用于传递参数

## 6.1 获取剩余参数

因为箭头函数中，没有arguments了。所以可以使用...语法来获取剩余参数。

demo:

|  |
| --- |
| 1. // ...语法 是ES6新增的语法 用于替代 arguments 也可以用于解构 还可以用于传递参数 2. function fun(a, b, ...args) { 3. // console.log(arguments); 4. // console.log([].slice.call(arguments)) 5. console.log(a, b, args) 6. } 7. fun(1, 2, 3) 8. fun(1, 2, 3, 4) 9. fun(1, 2, 3, 4, 5) 10. fun(1, 2, 3, 4, 5, 6) |

console:

|  |
| --- |
|  |

## 6.2 传递参数

demo:

|  |
| --- |
| 1. // 定义一个数组 2. var arr = [1, 2, 3, 4, 5, 6, 7, 8]; 3. // 输出arr的每一项 4. console.log(...arr); // 这种使用方式 等价于 console.log(1, 2, 3, 4, 5, 6, 7, 8); |

说明：...arr可以将数组中的每一个成员当做函数的每一个参数，依次传递。

# 解构

所谓解构，指的是解除结构。

对象和数组都是数据结构 是用来装数据的

函数不是数据结构

## 7.1 解构对象

普通的解构方式：

|  |
| --- |
| 1. var obj = { 2. username: "王老四", 3. age: 33, 4. sex: "男" 5. }; |

解构：

|  |
| --- |
| 1. // 普通的解构方式 2. var username = obj.username; 3. var age = obj.age; 4. var sex = obj.sex; 5. console.log(username, age, sex); |

ES6的解构方式：

|  |
| --- |
| 1. var {username, age, sex} = obj; |

两种解构效果一致。

ES6的解构时，变量名必须与属性名称一致，变量的定义顺序没有要求。

## 7.2 解构数组

demo:

|  |
| --- |
| 1. var arr = ["张三", "李四", "王五", "赵六"]; 2. // 解构数组 3. var [b, a, c, d] = arr; 4. // 等价于 5. var b = arr[0]; 6. var a = arr[1]; 7. var c = arr[2]; 8. var d = arr[3]; |

注：数组没有属性名称，所以解构时，定义变量的顺序有要求。

...语法解构数组：

|  |
| --- |
| 1. // 使用...语法解构数组 2. var [a, ...b] = arr; 3. console.log(a, b); |

输出结果：

|  |
| --- |
|  |

# 箭头函数

## 8.1 定义

箭头函数是ES6中新增的一个函数定义方式。

() => {}

() 参数列表

=> 箭头

{} 函数体

demo:

|  |
| --- |
| 1. // 定义箭头函数 2. var arrow = () => { 3. console.log("你好，我是一个箭头函数"); 4. } 5. // 执行函数 6. arrow(); |

console:

|  |
| --- |
|  |

## 8.2 参数默认值

该特点是所有函数都具备的特点，不仅仅限于箭头函数

demo: 普通函数的参数默认值

|  |
| --- |
| 1. // 参数默认值 2. var fun = function(a=1) { 3. console.log(a); 4. } 5. fun(); |

输出结果：

|  |
| --- |
|  |

## 8.3 不能作为构造函数

原因：因为ES6中实现了class关键字 不需要再使用构造函数去模拟

demo:

|  |
| --- |
|  |

## 8.4 没有了arguments

箭头函数中是没有arguments这个对象的。

demo:

|  |
| --- |
| 1. // 定义箭头函数 2. var fun = () => { 3. console.log(arguments); 4. } 5. fun(); |

console:

|  |
| --- |
|  |

## 8.5 函数中的this

通过function定义的函数，里面的this是谁，需要等到代码执行的时候才可以确定。

demo:

|  |
| --- |
| 1. // 定义一个箭头函数 2. var arrow = () => { 3. console.log(this); 4. } 5. arrow(); // window 6. document.onclick = arrow; // 当用户点击时 输出window 7. arrow.call(document.body); // window |

demo1:

|  |
| --- |
| 1. // 定义一个箭头函数 2. var obj = { 3. fun() { 4. return () => { 5. console.log(this); 6. } 7. }, 8. a: "asddsafdsadsa" 9. } 10. var arrow = obj.fun(); 11. arrow(); // obj 12. document.onclick = arrow; // 当用户点击时 输出obj 13. arrow.call(document.body); // obj |

注：箭头函数中的this，指向谁要看定义该箭头函数时的作用域中的this是谁。 一旦定义完毕，则不会再发生变化。

## 8.6 省略写法

demo:

|  |
| --- |
| 1. // 省略方式一 省略形参列表 当形参列表中只有一个参数时，可以省略左右() 2. // var arrow = a => { 3. // console.log(a); 4. // } 5. // arrow(1); 6. // 省略方式二 省略函数体和return 当函数体中 只有一条代码 并且该代码还是返回语句时 7. // var arrow = (a) => a \* a // 该方式等价于 var arrow = (a) => { return a \* a } 8. // var result = arrow(2); 9. // console.log(result); 10. // 省略方式三 既省略形参列表又省略函数体和return 11. // var arrow = a => a \* a; 12. // console.log(arrow(44)); |

# 迭代器

迭代器的定义：在访问某一个数据集合的时候，该集合能够在不暴露内部数据结构的情况下，顺序遍历里面的内容。

ES6中已经定义了迭代器，但是只实现了一部分迭代器。

## 9.1 keys

该方法用于顺序遍历数组的所有key

|  |
| --- |
| 1. // 定义数组 2. var arr = ["a", "b", "c", "d", "e", "f", "g", "h"]; 3. // 调用迭代器方法 4. var iterator = arr.keys(); 5. console.log(iterator); |

console:

|  |
| --- |
|  |

使用方式：

|  |
| --- |
| 1. // 定义数组 2. var arr = ["a", "b", "c", "d", "e", "f", "g", "h"]; 3. // 调用迭代器方法 4. var iterator = arr.keys(); 5. console.log(iterator); 6. // 得到的是iterator迭代器对象 该对象可以调用next方法进行遍历 7. // 第一次遍历的结果 8. console.log(iterator.next()); 9. // 第二次遍历的结果 10. console.log(iterator.next()); 11. console.log(iterator.next()); 12. console.log(iterator.next()); 13. console.log(iterator.next()); 14. console.log(iterator.next()); 15. console.log(iterator.next()); 16. console.log(iterator.next()); 17. console.log(iterator.next()); |

console:

|  |
| --- |
|  |

注：next方法是用于迭代一次的方法。返回一次结果。

返回的结果是一个对象，value属性表示本次迭代的结果 done属性表示是否迭代完毕。

## 9.2 values

与keys一致。也是得到一个迭代器对象。该迭代器调用next方法返回的对象依旧是value和done 只是这次的value的值是成员。

|  |
| --- |
| 1. // 定义数组 2. var arr = ["a", "b", "c", "d", "e", "f", "g", "h"]; 3. // 调用迭代器方法 4. var iterator = arr.values(); 5. console.log(iterator); 6. // 得到的是iterator迭代器对象 该对象可以调用next方法进行遍历 7. // 第一次遍历的结果 8. console.log(iterator.next()); 9. // 第二次遍历的结果 10. console.log(iterator.next()); 11. console.log(iterator.next()); 12. console.log(iterator.next()); 13. console.log(iterator.next()); 14. console.log(iterator.next()); 15. console.log(iterator.next()); 16. console.log(iterator.next()); 17. console.log(iterator.next()); |

console:

|  |
| --- |
|  |

## 9.3 entries

该方法与keys、values方法一致，都属于迭代器方法。唯一的不同是next返回的对象的value值不同。

keys的value值是下标

values的value值是成员

entries的value值是[下标, 成员]

|  |
| --- |
| 1. // 定义数组 2. var arr = ["a", "b", "c", "d", "e", "f", "g", "h"]; 3. // 调用迭代器方法 4. var iterator = arr.entries(); 5. console.log(iterator); 6. // 得到的是iterator迭代器对象 该对象可以调用next方法进行遍历 7. // 第一次遍历的结果 8. console.log(iterator.next()); 9. // 第二次遍历的结果 10. console.log(iterator.next()); 11. console.log(iterator.next()); 12. console.log(iterator.next()); 13. console.log(iterator.next()); 14. console.log(iterator.next()); 15. console.log(iterator.next()); 16. console.log(iterator.next()); 17. console.log(iterator.next()); |

console:

|  |
| --- |
|  |

# for……of循环

该循环专门用于循环迭代器 数组已经实现了迭代器接口 所以可以循环 对象还没有实现所以报错

demo:

|  |
| --- |
| 1. var arr = ["a", "b", "c", "d", "e", "f", "g", "h"]; 2. for (var i of arr) { 3. console.log(i); 4. } 5. var obj = { 6. a: 1, 7. b: 2, 8. c: 3, 9. d: 4 10. } 11. for (var i of obj) { 12. console.log(i); 13. } |

console:

|  |
| --- |
|  |

# 新的数据结构

## 11.1 Set

Set是ES6中新增的数据结构。可以理解为是一个去重的数组。

demo:

|  |
| --- |
| 1. // Set是一个新的数据结构 用于存放数据 2. var set = new Set([1, 2, 3, 77, 3, 3, 3, 3]); 3. console.log(set); |

console:

|  |
| --- |
|  |

add: 添加一项 如果已经存在则添加失败

clear: 清空

delete: 删除一项

entires: 迭代器方法

forEach: ES5的迭代器方法

has: 判定是否存在某一项 （返回布尔值）

keys: 迭代器方法

values: 迭代器方法

## 11.2 WeakSet

WeakSet是一个弱Set，成员只可以是引用类型。

demo:

|  |
| --- |
| 1. // 定义WeakSet 2. var ws = new WeakSet([1, 2, 3, 4]); |

console:

|  |
| --- |
|  |

WeakSet中保存内容时，不会被垃圾回收机制考虑在内

|  |
| --- |
| 1. var ws = new WeakSet([{}, {}, {}]); |

输出ws:

|  |
| --- |
| 1. 第一次输出时，是有内容的。但是第二次第三次输出时，内容消失。 |

## 11.3 Map

Map是一个对象相关的数据结构。

特点：键可以是任意类型。

初始化:

|  |
| --- |
| 1. var map = new Map(); |

输出:

|  |
| --- |
|  |

设置几个值：

|  |
| --- |
| 1. map.set("name", "张三"); 2. map.set(false, false); 3. map.set({}, [1, 2, 3, 4, 5]); |

输出：

|  |
| --- |
|  |

## 11.4 WeakMap

WeakMap是一个弱Map

Map的key可以是任意类型

WeakMap的key只能够是引用类型

初始化：

|  |
| --- |
| 1. // 初始化 2. var wm = new WeakMap(); 3. wm.set("name", "张三"); |

报错：

|  |
| --- |
|  |

它的特点与WeakSet一致，不会被垃圾回收机制计入。

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |