# ECMAScript6

第2天课堂笔记

[ECMAScript6 1](#_Toc10363)

[复习 3](#_Toc902)

[数组拓展方法 7](#_Toc28700)

[对象复制 7](#_Toc4389)

[Symbol类型 8](#_Toc6619)

[代理 8](#_Toc7095)

[默认参数 10](#_Toc25565)

[获取剩余参数 10](#_Toc32044)

[下午复习 11](#_Toc2858)

[解构 13](#_Toc1504)

[箭头函数 13](#_Toc20567)

[Set对象 14](#_Toc1816)

[WeakSet 15](#_Toc3731)

[Map对象 15](#_Toc27223)

[WeakMap 16](#_Toc24607)

[迭代器 16](#_Toc1149)

[Generator函数 18](#_Toc25459)

[Promise对象 18](#_Toc2770)

# 复习

Typescript

类的接口

Interface 类的名称 {

// 定义类的结构

}

接口只能定义结构，不要具体的实现

模块

Module 模块名称 {

// 模块的具体实现

}

模块中暴漏的接口用export定义

我们可以通过模块的点语法访问这些接口

ES6

定义变量

Let 定义块作用域变量

1 声明前置

Let不能声明前置，只能先定义再使用

Var 可以声明提前

2 作用域

Let 是块作用域

Var 是一个函数级作用域

3 循环中

Let定义的循环变量，可以保存在定义函数中

Var 定义循环变量，不能保存在定义的函数中

Const 定义常量

1 一旦定义不能被修改，修改会抛出错误

2 不支持声明前置，先定义再使用

3 是块作用域

字符串拓展

子字符串位置

startsWith 是否以参数字符串开头

第一个参数 子字符串

第二个参数 判断的起始位置

返回值是一个布尔值

当传递第二个参数时，截取的是后面的一部分

endsWith 是否以参数字符串结尾

第一个参数 子字符串

第二个参数 判断结束位置

返回值也是一个布尔值

当传递第二个参数时候，截取的是前面的一部分

Includes 判断是否包含该子字符串

第一个参数 子字符串

第二个参数 截取的位置

返回值也是布尔值

当传递第二个参数时候，截取的是后面的一部分

模板字符串

定义模板字符串的语法 ： ``

模板字符串可以是多行的

定义模板的语法 ： ${表达式}

表达式可以是变量，运算，方法

重复字符串

Repeat

参数表示重复的次数

原始字符串

Raw是String对象的静态方法，必须通过String对象使用

String.raw`字符串`

返回的结果，不会对转义字符转义

数字对象拓展

isNaN 是否是一个NaN

全局作用域中有一个isNaN方法，会做类型转换

Number对象的isNaN不会做类型转换

isFinity 判断是否是有限的

全局作用域中有一个isFinity,会做类型转换

Number对象的这个isFinity不会做类型转换

isInteger 判断是否是整形

这三个方法返回值都是布尔值

Math对象中sign

五个返回值

0 判断是正零

-0 判断负零

1 正数

-1 负数

NaN 非数字

拓展了一些数学方法

对数组的拓展

From 将类数组对象转化成数组

第一个参数表示类数组对象

第二个参数（可选）回调函数

第一个参数表示值

第二个参数表示索引值

必须有返回值。返回值是新数组的成员值

方法的返回值就是一个有属性值构成的一个新数组

Of 创建数组的

参数就是数组的成员

参数如果是方法不会执行

Find 搜索数组中成员的位置

返回值是成员值

findIndex搜索数组中成员的位置

返回值是成员索引值

参数都是一个函数

第一个参数都是value成员值

第二个参数都是key索引值

返回值表示判断的条件

非贪婪搜索，一旦搜索到就返回，不会再去执行了

对对象拓展

对象定义

1 如果对象的属性名称与属性值变量相同，属性名称可以省略

2 定义属性名称时候可以使用表达式（写在[]语法内）

3 定义方法时候，function关键字可以省略，跟定义类方法一样定义对象方法

Is方法 判断两个参数是否全等的

返回值是布尔值

解决全等符号两个问题

1 不能判断0和-0

Is方法返回是false

2 不能判断NaN

Is方法返回是true

其他判断结果跟全等符号是一样

# 数组拓展方法

Keys 获取去数组的所有索引值

Values 获取数组所有值

Entries 获取数组所有索引值以及成员值

返回的是一个数组

第一个成员表示索引值index

第二个成员表示成员值value

这三个方法返回值是一个数组迭代器对象

遍历数组迭代器内容，只能使用for of方法

不能使用for循环，for in循环遍历数组迭代器对象

迭代器第二个特征是可以通过next方法访问每一个值

返回一个对象

Done表示是否还可以遍历

Value表示这一次遍历的值

一旦遍历完成，done永远是true，vakue永远是undefined

|  |
| --- |
| 1. var arr = ['小沈阳', '宋小宝', '尼古拉斯赵四', '刘能']; 2. var keys = arr.keys(); 3. var values = arr.values(); 4. var entries = arr.entries() |

# 对象复制

Object拓展了一个方法assign

对对象进行复制

Jquery中有个方法是extend

第一个参数

如果是布尔值true就是深复制，

如果是对象就是浅复制

深复制 复制的时候，不仅仅可以将值类型的数据复制过来，还可以将引用类型的数据复制过来，注意不是地址的指向

浅复制 复制的时候，只能将值类型的数据复制过来，对于引用类型的数据，只是一个地址的指向

Assign方法

第一个参数表示复制的目标对象

从第二个参数开始被复制的对象

返回值就是拓展后的第一个参数对象

从第二个参数开始，将参数对象中的属性复制给第一个参数对象

在复制属性的时候，后面的对象中同名属性，会覆盖前面对象中的

|  |
| --- |
| 1. /\*\* 2. \* 对对象进行浅复制 3. \* @target 复制的目标的对象 4. \* 从第二个参数开始是复制的对象 5. \* 返回值 复制后的目标对象 6. \*\*/ 7. function assign(target) { 8. // 要获取所有复制对象，要通过arguments获取 9. var arg = [].slice.call(arguments, 1); 10. // 遍历这些对象，将对象中的属性复制过来 11. for (var i = 0; i < arg.length; i++) { 12. // 每一个对象是arg[i],遍历其属性并复制给target 13. for (var key in arg[i]) { 14. // 复制属性:将key属性添加给target，并赋值属性值 15. // 判断是否对象上直接属性hasOwnProperty 16. target[key] = arg[i][key] 17. } 18. } 19. // target 20. return target; 21. } |

# Symbol类型

ES6新增的一个数据类型，表示独一无二的，

使用：有时候我们拿到一个库，为其添加方法，我们往往不会读这个库，而是直接对他新增属性，新增方法，这是很危险的，原因是我们新增的属性，新增的方法可能会替换掉原有的属性或者方法，原理就是属性可能会同名，es6新增symbol数据类型是为了解决这个问题，通过symbol新增的属性是永远不会同名的

创建symbol数据通过symbol函数创建的，不要使用new关键字

Symbol第一个字母大写，创建直接执行

我们用typeof 查看数据类型，结果就是symbol

可以传递参数，传递不传递参数，得到结果都表示独一无二的数据

结果是Symbol(参数名称)

一旦通过symbol类型为对象添加属性，访问这个属性值，只能通过symbol变量

|  |
| --- |
| 1. var key1 = Symbol() 2. obj[key1] = 'blue' |

# 代理

代理对象Proxy

应用：在一个大型项目中，总有一些数据是不希望每个人都能访问，js中定义的数据我们是可以随意的修改的，代理的目的就是说，不允许用户直接访问这些隐私数据，而是通过一个代理对象来访问这些数据，通过这个代理对象屏蔽一些危险操作

ES6中代理是一个类

所以使用代理要通过new关键创建一个代理

第一个参数表示被代理的对象

第二个参数表示处理代理对象的接口方法代理对象

Get获取对象属性的

第一个参数表示被代理的对象

第二个参数表示获取是属性名称

作用域是代理对象

必须有返回值，就死活获取的属性值

在返回之前我们是可以处理这次操作的

Set 设置对象的属性值的

第一个参数表示被代理的对象

第二个参数表示设置的属性名称key

第三个参数表示设置的属性值value

作用域是代理对象

通过代理对象的属性就可以获取或者设置被代理对象数据

|  |
| --- |
| 1. var star = { 2. // 名字可以被访问 3. names: '尼古拉斯赵四', 4. // 女朋友数据很有隐私性，因此不允许每个人方法 5. girlFriend: '尼古拉斯赵五' 6. } 7. // 明星这个对象有一些数据可以访问，一些数据不可以访问，所以将这个对象直接暴漏给用户是很危险的，因此我们可以添加一个代理 8. var starProxy = new Proxy(star, { 9. // 取值方法 10. get: function (obj, key) { 11. // grilfriend不允许访问 12. if (key === 'girlFriend') { 13. return '不告诉你' 14. } 15. // 其他的是属性正常返回 16. return obj[key] 17. }, 18. // 赋值方法 19. set: function (obj, key, value) { 20. // 如果修改女朋友是不允许的 21. if (key === 'girlFriend') { 22. console.log('没有女朋友') 23. return ; 24. } 25. // 如果修改名称可以 26. obj[key] = value 27. } 28. }) 29. // 查看代理对象 30. // console.log(starProxy.girlFriend) |

# 默认参数

ES6允许我们在参数中使用赋值符号，定义默认参数

以往定义默认参数要通过运算符，比较麻烦

|  |
| --- |
| 1. // 创建一个函数 2. function drawColor (color = 'green') { 3. console.log(color.toUpperCase()) 4. } |

# 获取剩余参数

Sass，stylus中都学过获取剩余参数 arg...(三个点语法，加在变量后面了)

ES新增获取剩余参数语法，

语法 ...arg

获取的arg变量是一个数组，可以直接调用数组的方法

前面定义的变量可以正常使用

应用：箭头函数中不能使用arguments，但是我们可以使用获取剩余参数语法

|  |
| --- |
| 1. // 可以通过...语法，获取剩余参数 2. function add(num1, ...num) { 3. // 至少有两个参数 4. if (num.length) { 5. // console.log(Array.isArray(num)) 6. // 调用reduce求和 7. return num1 + num.reduce(function (res, value) { 8. return res + value 9. }) 10. // 只有一个参数 11. } else if(num1) { 12. return 10 + num1 13. // 没有参数 14. } else { 15. return 0 16. } 18. } 19. // 测试 20. console.log(add()) 21. console.log(add(2)) 22. console.log(add(3, 4)) |

# 下午复习

数组三个拓展方法

Keys获取数组中所有成员的索引值

Values 获取数组中所有成员值

Entries 获取数组中所有成员与索引值的键值对

每一组是一个数组

第一个成员表示索引值

第二个成员表示value

三个方法返回值都是一个数组迭代器对象，只能用for of循环

可以通过next方法访问每一个成员值

返回值是一个对象

Done 是否遍历完成，true表示遍历完成

Value表示每一次遍历的值，遍历完成值是undefined

对象拓展assign

对象浅复制

第一个参数表示目标对象

从第二个参数开始表示复制的对象  
 返回值是拓展后的第一个参数对象

Symbol类型

表示独立无二

创建只能通过执行函数，不能使用new关键字

Typeof判断结果是symbol

可以接受参数，得到的结果就是得到的值可以看到参数

作用：为对象拓展属性时不会覆盖已有属性

代理对象

Proxy类

第一个参数表示被代理的对象

第二个参数代理对象

Get获取对象属性的

第一个参数表示被代理的对象

第二个参数表示属性名称

作用域是代理对象

必须有返回值

Set 设置对象属性的

第一个参数表示被代理的对象

第二个参数表示属性名称

第三个参数表示属性值

作用域代理对象

定义函数默认参数

在函数参数中可以使用赋值符号定义默认参数

获取剩余参数

语法 ...arg

获取的剩余参数是一个数组

前面定义的其他变量可以正常使用

# 解构

解析一个数据的结构

一种是对对象的解构

语法 {属性名称} = 对象

大括号中的每一个名称被定义成变量，可以直接访问对象中是属性

变量名称一定要与对象的属性名称一致

一般是在模块化开发中使用，这样定义变量更安全



一种是数组结构

语法 [变量1， 变量2] = 数组

每一个变量表示一个成员

如果想获取所有剩余成员可以使用...语法

...语法获取到的是一个数组



...语法在函数调用时候使用

可以将一个数组的每一个成员转化成函数参数



|  |
| --- |
| 1. // 对象解构 2. var {Model, Collections, View} = Backbone 3. // 数组解构 4. var colors = ['red', 'green', 'blue', 'yellow', 'orange', 'pink']; 5. var [color1, color2, blue, ...color] = colors; |

# 箭头函数

ES6中新增了一种定义函数的方式，就是箭头函数

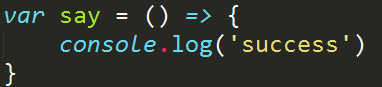
定义语法 () => {}

一个箭头函数由三个部分组成

第一个部分是函数的参数集合 ()

第二个部分就是箭头 =>

第三个部分是函数体 {}



特征

箭头函数不能使用arguments

作用域是定义时候的作用域

箭头函数不能作为构造函数使用

省略

如果只有一个参数，小括号可以省略

如果函数体只有返回值，大括号可以省略，return也可以省略



定义箭头函数使用函数表达式

更改箭头函数作用域，只能更改箭头函数定义时候的作用域

|  |
| --- |
| 1. var say = () => { 2. console.log('success') 3. } 4. // 使用箭头函数 5. say(); 6. // 求一个数组中成员的平方 7. var arr = [1, 2, 3, 4, 5] // [1, 4, 9, 16, 25] 8. // 通过map方法实现 9. var result = arr.map(value => value \* value) |

# Set对象

在ES6中表示聚合类的数据一共有四种： 数组，对象，set对象，map对象

Set对象，是对数组的一个拓展，用来表示一组去重后的数组，

是一个构造函数数，参数是一个数组，会对数组中每一个成员去重

执行构造函数的结构会返回一个set实例化对象，有一些方法用来操作set对象的

Add ：为set对象添加成员的

只能接收一个参数，就是添加的成员，会去重的

返回值是set对象，因此可以链式调用

Delete 删除set对象中某个成员的

只有一个参数，表示删除的成员

返回值是布尔值

True表示删除成功

False表示删除失败

Has 是否拥有某个成员值

只有一个参数，表示的是成员

返回值是布尔值

True表示存在

False表示不存在

Clear清空set对象

forEach遍历set对象

参数是一个回调函数

第一个参数表示值

第二个参数表示值

第三个参数表示set对象

方法返回值是undefined

作用域是全局作用域

Size 表示set对象成员长度（实现迭代器接口的对象绝大部分都是用size表示长度）

Set实现了迭代器接口，所以可以通过for of遍历

Typeof判断set对象类型是object

想判断set对象类型用Object.prototype.toString.call判断



|  |
| --- |
| 1. var mySet = new Set([1, 2, 3, 1,2, 1, 2, 1]) |

# WeakSet

弱set对象，

成员只能是对象

一些set对象的原型方法不存在

Clear，forEach，size等属性以及方法不存在了

add方法，用来添加成员的

delete方法，用来删除成员的

has方法，判断成员是否存在

注意这个对象少用，因为不能被内存回收机制回收

不能使用for of遍历



|  |
| --- |
| 1. // 定义weakset对象 2. var ws = new WeakSet([document, document, window]); |

# Map对象

Map是对对象的一个拓展，可以看作是一个超对象

对象的属性名称通常是一个字符串

Map对象允许属性名称可以是任意类型的数据，null，对象，数组，function，undefined，数字，布尔等等都可以

通过map的构造函数可以创建一个map对象

set：为map对象添加新属性

第一个参数表示属性名称

第二个参数表示属性值

返回值是map对象，因此可以链式调用

get：获取map对象中某个属性值

参数表示属性名称

返回值表示属性值

clear：清空map对象

has：判断某个属性值是否存在

参数表示属性名称

返回值是布尔值表示是否存在

delete 删除某个属性值的

参数表示属性名称

返回值是布尔值，表示是否删除成功

forEach 遍历属性的

遍历map对象，参数是一个回调函数

作用域是全局作用域

第一个参数表示map属性值

第二个参数表示map属性名称

第三个参数表示map对象

Size: 长度属性



|  |
| --- |
| 1. // 创建一个map对象 2. var myMap = new Map(); 3. // 属性名称可以是任意类型的 4. myMap.set(window, 123) |

# WeakMap

弱map对象

属性名称只能是对象

只有has，delete，set，get方法，使用方式同上

工作中少用，不能被内存回收机制回收



|  |
| --- |
| 1. var wm = new WeakMap(); |

# 迭代器

ES6中实现了迭代器接口，但是没有实现迭代器对象Iterator

迭代器使用

1 数据解构 数组结构，对象解耦

2 定义map对象，set对象，weakmap对象，weakset对象时候

3 使用for of循环时候

4 实现了迭代器接口的数据

实现迭代器接口的一个重要特征就是能否实现for of循环

For of循环可以说是专门为数组实现的一个循环

以前学过的循环

For循环

定义的循环变量可以在外界访问

当嵌套循环时候，访问成员值比较麻烦

可以使用

Break 跳过后面所有循环，

Continue 跳过当前一次循环

在方法中

break可以跳出循环并且继续执行后面的语句

return 跳出循环，不会执行后面的语句

迭代器方法（forEach）

解决访问成员繁琐的问题（成员作为参数访问）

不能使用Break和continue关键字

可以使用return关键字跳出当前循环，相当于continue

问题就是不能终止循环

For in循环

在遍历数组的时候可以访问到成员值以及索引值，

但是将索引值转化成字符串了

会遍历到原型上的属性方法

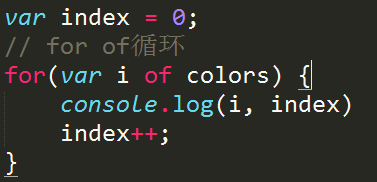
可以使用break，continue等关键字

For of 循环（是专门为数组定制的一个循环）

可以访问成员值

但是不能访问索引值，我们可以定义循环变量来实现

可以使用break，continue等关键



|  |
| --- |
| 1. var colors = ['red', 'green', 'blue', 'yellow', 'orange']; 2. var index = 0; 3. // for of循环 4. for(var i of colors) { 5. if (i === 'green') { 6. // break; 7. continue; 8. } 9. console.log(i, index) 10. index++; 11. } |

# Generator函数

这是一类函数，作用就是用来存储状态的

定义语法

function \* 函数名称 () {

}

与普通函数相比，就是函数名称前面多了一个\*

我们在函数内容通常只是定义状态

定义状态： yield 状态内容;

每一个yield定义了一个状态

最后一个状态用return 定义

Generator函数实现了迭代器接口，因此可以通过for of循环遍历所有状态

不能访问最后一个状体

所以也可以通过next访问下一个状态

方法返回值是一个对象

done表示是否遍历完成

Value表示状态值

|  |
| --- |
| 1. // 定义generator函数 2. function \* dayState () { 3. // 定义状态 4. yield '睁眼'; 5. yield '砸闹钟'; 6. yield '接着再睡'; 7. yield '再睁眼'; 8. yield '起来了'; 9. yield '上学'; 10. yield '学习'; 11. // 睡觉是最后一个状态 12. return '睡觉'; 13. } 14. // 使用generator函数必须执行 15. var ds = dayState(); 16. // 访问状态 17. console.log(ds.next()) |

# Promise对象

ES6中新增的一个允诺对象

作用：将异步操作转化成同步的写法

Promise对象内部维护一个状态，有三种情况

Pending 表示异步执行的操作还没有接触

Resolved 表示异步操作已经执行完成

Rejected 表示异步操作执行失败

在promise往往只能存在两种状态的转化，因此只能出现两个状态

一个是由pending进入resolved，说明异步操作执行成功

一个是由pending进入rejected，说明异步操作执行失败

Promise构造函数有一个回调函数作为参数，

回调函数有两个参数

一个叫resolve 用来将pending状态转换成resolved状态

一个叫reject用来将pending状态转换成rejected状态

作用域是全局作用域

在回调函数中我们执行异步操作，当执行完毕，我们要触发相应的状态改变

我们可以在promise实例化对象中通过then方法绑定回调函数

Then方法

第一个参数表示成功时执行回调函数

第二个参数表示失败时候执行的回调函数

这两个函数的参数是由执行resolve或者reject方法时候传递的参数

当第二次执行then方法的时候，回调函数获取不到参数了

允诺对象支持监听多个异步操作

通过Promise的all方法监听多个允诺对象

当监听多个允诺对象时候

如果有一个失败了就立即执行失败回调函数

如果都执行成功了，会执行成功时候的回调函数

|  |
| --- |
| 1. // 创建一个promise对象 2. var p = new Promise((resolve, reject) => { 3. // console.log(resolve, reject) 4. // console.log(this) 5. setTimeout(function () { 6. console.log('timeout') 7. //执行成功了 resolve 8. resolve('hello') 9. // 执行失败了 10. // reject('word') 11. }, 2000) 12. }); 13. // 绑定同步执行的方法 14. p.then(function () { 15. console.log(this, 111) 16. console.log(arguments) 17. }, function () { 18. console.log(this, 222) 19. console.log(arguments) 20. }) 21. .then(function () { 22. console.log(arguments, 333) 23. }) 24. // 定义第二个异步操作 25. var p2 = new Promise(function (resolve, reject) { 26. // 为按钮绑定一个事件 27. btn.onclick = function () { 28. console.log('click') 29. // 成功了 30. resolve('click') 31. } 32. }) 33. // 监听p和p2执行完毕 34. Promise 35. .all([p, p2]) 36. .then(function () { 37. console.log('success', arguments) 38. }, function () { 39. console.log('fail') 40. }) |