# ECMAScript6

第1天课堂笔记

[ECMAScript6 1](#_Toc17845)

[复习 3](#_Toc30895)

[类接口 7](#_Toc559)

[模块 7](#_Toc15899)

[ES6 8](#_Toc516)

[定义变量 9](#_Toc3183)

[常量 9](#_Toc29479)

[下午复习 10](#_Toc21443)

[字符串拓展 11](#_Toc16280)

[判断字符中子字符串的位置 11](#_Toc12108)

[模板字符串 11](#_Toc29190)

[重复字符串 12](#_Toc12812)

[原始字符串 12](#_Toc24650)

[数学对象 12](#_Toc13773)

[数字对象拓展 12](#_Toc21245)

[数学对象拓展 12](#_Toc9971)

[ES6对数组拓展的 13](#_Toc23157)

[From 13](#_Toc23535)

[Of方法 13](#_Toc17644)

[Find和findIndex 14](#_Toc15716)

[copyWithin 14](#_Toc1696)

[对象拓展 14](#_Toc27696)

[Is方法 15](#_Toc11304)

# 复习

ES5中特性

定义特定方法

defineProperty

第一个参数表示对象

第二个参数表示对象上的属性

第三个参数表示该属性特性对象

Value 定义属性值的

Writable 属性是否是可以被修改的

True是可以被修改

False 不能被修改

Enumerable 属性是否可以被枚举（能否通过for in循环遍历）

True 可以被枚举

False不能被枚举

Configurable 特性对象是否可以再次被修改

True 可以修改

False 不可以被修改

Get 获取属性值的时候，调用该方法

没有参数

作用域就是这个对象

必须有返回值，

不能获取这个属性

Set 为属性赋值时候调用的方法

参数就是赋予新的值

作用域是这个对象

不能修改这个属性

Get和set方法不能与value以及writable同时出现

定义多个属性特性

defineProperties

第一个参数表示对象

第二个参数要被设置的属性对象

属性名称（key）表示对象中要被设置的属性名称

属性值（value）表示属性的特性对象

获取对象所有属性名称

getOwnPropertyNames

参数就是一个对象

返回值是一个数组，每个成员是一个属性名称

不能获取原型上的属性名称

获取属性特性对象

getOwnPropertyDescriptor

第一个参数表示对象

第二个参数表示属性名称

返回值就是特性对象

对象的封闭

Seal方法封闭对象

isSealed查看对象封闭状态

对象冻结

Freeze方法冻结对象

isFrozen 查看对象是否被冻结

原型

isPrototypeOf 在原型上调用的方法，

getPrototypeOf 参数是对象，获取原型方法

这个方法是用来替换\_\_proto\_\_属性的

Typescript语言

定义变量

Var 变量:类型 = 值;

数据类型

Any定义任意类型

数组类型

Var 数组名称:类型[] = 数组

此时数组每一个成员都要符合这个类型

枚举类型

Enum定义枚举类型

Enum 枚举名称 {

// 枚举内容

}

特征：

既可以通过属性名称访问索引值

也可以通过索引值访问属性名称

函数

Ts中对函数的参数以及返回值做了处理

参数

一定要定义类型

参数可有可无，在参数后面加上?

使用函数时候，参数一定要与定义时候参数一致

返回值

必须定义返回值类型

如果函数没有返回值 定义:void

类

语法

Class 类的名称 {

// 类的具体实现内容

}

可以定义属性方法，一定不要忘记类型

定义属性

定义的属性如果赋值会添加在构造函数里

定义属性如果没有赋值，编译时候会删除

定义在构造函数内的属性，无论有没有赋值，都会保留在构造函数内

定义方法

定义的方法添加在原型上

定义的方法不要忘记返回值的类型，以及参数的类型

构造函数是不能有返回值类型

Public，private关键字没有意义

Static有意义

定义类的静态属性和方法

定义的属性或者方法，添加了static，会保留在类上，所以在类的实例化对象中访问不到的，只能通过类访问

如果属性没有值，编译时也会删除

继承

Extends实现继承

在子类中如果继承父类，不仅仅继承了构造函数，原型，还继承了静态属性方法

Ts中继承是寄生组合式继承，还拓展了对静态属性方法的继承

在子类中如果定义了与父类相同的属性方法，会覆盖掉父类中的

如果在子类中定义了构造函数，在构造函数中想继承父类，一定要通过super关键字继承

Super一定要写在最前面

接口

对象接口

Interface 接口名称 {

// 定义接口内容

}

属性以及方法要以;结尾

接口中定义的属性方法，对象中一定要实现，如果属性或者方法是可有可无的要加上?

函数接口

Interface 接口名称 {

定义参数以及返回值类型

}

参数如果可有可无要加上?

# 类接口

定义语法

interface 类 {

// 定义类的接口

Title: type;

Say(ms:string):void;

}

只需要定义类的接口，不要定义类的具体实现

如果让类实现这个接口，要通过implements关键字

|  |
| --- |
| 1. // 定义类的接口 2. interface Star { 3. // 定义属性 4. name:string; 5. // 可有可无 的属性加上问号 6. girlFriend?:string; 7. // 定义方法 8. getAge():number; 9. } 10. // 定义明星类 11. // implements实现类的接口 12. class MovieStar implements Star { 13. name: string; 14. age: number; 15. // 定义构造函数 16. constructor(name:string, age:number) { 17. this.name = name; 18. this.age = age; 19. } 20. // 实现getAge方法 21. getAge():number { 22. return this.age; 23. } 24. // 实现获取名称方法 25. getName():string { 26. return this.name; 27. } 28. } 29. // 实现明星 30. var star = new MovieStar('尼古拉斯', 50) |

# 模块

在模块内部定义的任何数据，不会污染全局作用域

Ts中通过module实现了模块

语法

Module 模块名称 {

// 在模块体中定义模块的内容

}

对于一个模块来说，想使用里面的数据，模块必须暴漏接口（api），才能使用这个模块

在模块中定义接口用export

通过export定义的接口，我们在外部可以使用

通过export暴漏的接口，在外部要通过模块名称的点语法使用

|  |
| --- |
| 1. // 定义模块 2. module myModules { 3. // 定义内容 4. export var date:string = '12-9'; 5. // add方法 6. export function add(num1:number, num2:number): number{ 7. return num1 + num2 8. } 9. } |

# ES6

ES6是ECMAscript 2015年的标准

ES3是99年标准

ES3.1 03年

ES5 09年

ES6 15年（是这些版本中变化最大的一个），

ES6着眼于未来技术的发展，为了企业级开发而产生的一套标准

现在ES6的语法好的浏览器能支持80%

现在仍然有很多浏览器不支持，所以我们还要将es6编程ES5版本的js在发布产品

ES6跟ts一样，可以在浏览器端编译，也可以在工程化中编译

在浏览器中编译需要一个编译工具

需要引入两个库

Traceur.js

Traceur.bootstrap.js

ES6要将标签改成module类型

在工程化中使用fis-parser-babel2插件解析

第一步 获取所有ts文件

第二步 引入插件 parser:babel2

第三步 将拓展名改成.js

ES6中实现的是一个闭包安全类

|  |
| --- |
| 1. class myApp { 2. constructor() { 3. this.name = 123; 4. } 5. say() {} 6. } 7. console.log(new myApp()) |

ES6中大部分语法已经被高级浏览器实现了，所以今天学习的大部分案例我们可以直接写在js中

# 定义变量

ES6新增了一个关键字let

定义一个块作用域变量的

Let与var比较

1 声明提前

Var关键定义的变量可以声明前置

Let关键字定义的变量不能声明前置

2 作用域

Var关键字定义的变量的作用域是函数级的

Let关键字定义的变量的作用域是块级的（当然也支持函数级）

3 在循环中

Var关键字定义的循环变量，不能保存在创建的函数中

Let关键字定义循环变量，可以保存在创建的函数中

|  |
| --- |
| 1. let b = 234; |

# 常量

什么是常量？

一旦被定义就无法再被修改

通过const定义常量

1 常量一旦被修改就抛出错误

注意：在前端定义常量时候，常量名称通常是大写，多个单词之间用\_分隔

2 常量跟let一样，也支持块作用域

使用时候，一定要先定义在使用

3 Const定义的变量也不支持声明前置

在过去没有const，我们可以通过freeze冻结对象来定义常量

使用freeze定义常量要得到ES5的支持

在ES5之前的版本中，定义常量只能通过单例模式定义

单例模式中只定义取值器，不定义赋值器来实现常量的定义

|  |
| --- |
| 1. const PI = Math.PI; |

# 下午复习

类的接口

语法

Interface 类接口的名称 {

// 定义类的结构

}

类实现接口通过implements实现

Class 类名 implements 类的接口名称{}

模块

语法

Module 模块名称 {

// 定义模块的内容

}

我们要通过export 定义模块的接口，才能使用模块的数据

ES6

使用es6有两种方式

通过script标签引入两个库文件

Script标签类型是module书写es6语法

工程化中使用ES6

获取所有ts文件

引入插件

修改文件后缀名称

Let关键字

定义块作用域变量

与var比较

1 声明提前

Var可以提前，let不可以提前

2 作用域

Var函数级作用域，let块级作用域

3 循环中

Var定义循环变量不能保存在循环中定义的函数体内

Let定义的循环变量可以保存在循环中定义的函数体内

Const关键字

定义常量（一旦被定义就不能再被修改）

支持块作用域

不支持声明提前

# 字符串拓展

ES6对字符串拓展了一些方法

## 判断字符中子字符串的位置

startsWith 是否以参数子字符串开头的

第一个参数表示子字符串

第二个参数表示比较起始位置

返回值是布尔值，

True是以该子字符串开头的

False 不是以该子字符串开头的

endsWith

第一个参数表示子字符串

第二个参数表示比较结束位置

返回值是布尔值

True是以该子字符串结尾的

False 不是以该子字符串结尾的

Includes 判断是否包含字符串

第一个参数表示判断的子字符串

第二个参数表示截取的位置

返回值是一个布尔值

True表示包含

False表示不包含

startsWith截取字符串是后面的一部分

endsWith截取字符串时前面的一部分

Includes截取的字符串后面的一部分

|  |
| --- |
| 1. var result = str.startsWith('这是') 2. var result = str.endsWith('网站', 9) 3. var result = str.includes('爱创课堂') |

## 模板字符串

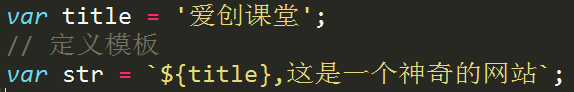
模板语法

${表达式}

表达式中可以直接写js运算符，语句等等

模板字符串必须写在``之间

``支持多行字符串，根据这一特性，我们可以渲染一个html片段



|  |
| --- |
| 1. var str = `${title}, 2. 这是一个神奇 3. 的网站` |

## 重复字符串

Repeat 重复输出字符串

参数就是表示重复的次数

返回值重复的结果

执行这个方法不会修改原始字符串

|  |
| --- |
| 1. var result = str.repeat(2); |

## 原始字符串

Raw将字符串中转义字符串输出

这个方法调用时候不需要括号，直接加上``

|  |
| --- |
| 1. var result = String.raw`这是\n一个字\n符串`; |

# 数学对象

## 数字对象拓展

ES6对数字对象做了拓展，主要是为了更准确的验证数字

isNaN 检测是否是NaN

全局作用域中isNaN在判断时候会做类型转换，判断不精确

Number对象中isNaN在判断的时候，不会做类型转换，

isFinity 检测是否是无穷的

全局作用域中的isFinity做了类型转换了

Number对象中的isFinity 没有做类型转换

其他类型都认识无限的

isInteger 检测是否是整形

不会做类型转换

## 数学对象拓展

Sign获取数字类型的标示

正数 返回 1 (包含正无穷)

负数 返回 -1 (包含负无穷)

0 返回0

-0 返回-0

其他的 返回NaN

所以根据这个方法可以判断0和-0

Math.cbrt：计算一个数的立方根。

Math.fround：返回一个数的单精度浮点数形式。

Math.hypot：返回所有参数的平方和的平方根。

Math.expm1(x)：返回ex - 1。

Math.log1p(x)：返回1 + x的自然对数。如果x小于-1，返回NaN。

Math.log10(x)：返回以10为底的x的对数。如果x小于0，则返回NaN。

Math.log2(x)：返回以2为底的x的对数。如果x小于0，则返回NaN。

三角函数方法

Math.sinh(x) 返回x的双曲正弦（hyperbolic sine）

Math.cosh(x) 返回x的双曲余弦（hyperbolic cosine）

Math.tanh(x) 返回x的双曲正切（hyperbolic tangent）

Math.asinh(x) 返回x的反双曲正弦（inverse hyperbolic sine）

Math.acosh(x) 返回x的反双曲余弦（inverse hyperbolic cosine）

Math.atanh(x) 返回x的反双曲正切（inverse hyperbolic tangent）

|  |
| --- |
| 1. console.log(Number.isNaN('abc')) 2. console.log(Number.isFinite(true)) 3. console.log(Number.isInteger('100')) 4. console.log(Math.sign(num1)) // NaN NaN 5. console.log(Math.sign(num2)) // Infinity 1 6. console.log(Math.sign(num3)) // -Infinity -1 7. console.log(Math.sign(num4)) // 0 0 8. console.log(Math.sign(num5)) // -0 -0 |

# ES6对数组拓展的

## From

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）表示回调函数

第一个参数表示遍历时候类数组对象属性值

第二个参数表示遍历时候类数组对象的索引值属性

必须有返回值是from方法执行的结果

|  |
| --- |
| 1. var newDivs = Array.from(divs, function (value, index) { 2. // 在回调函数中我们可以操作value 3. value.innerHTML = '新设置的索引值' + index 4. return value 5. }) |

## Of方法

用来创建数组

Of方法的出现是为了解决数组构造函数Array的一些问题：传递参数的不同，得到结果不一样

Array构造函数参数不一样结果不一样

var arr1 = new Array(); // []

var arr2 = new Array(3); // [] 长度为3的空数组

var arr3 = new Array(3, 4); // [3, 4]

var arr4 = new Array('5'); // ['5']

所以of是专门用来解决参数与结果不一样的问题的

Of中的参数都会作为数组成员

参数如果是函数不会执行

|  |
| --- |
| 1. console.log(Array.of()) 2. console.log(Array.of(3)) 3. console.log(Array.of(3, 4)) 4. console.log(Array.of('5')) |

## Find和findIndex

在数组中搜索某个成员

Find返回的是成员值

findIndex返回的是索引值

参数只能是回调函数

第一个参数表示成员值value

第二个参数表示索引值index

第三个参数表示原数组arr

## copyWithin

copyWithin 在数组中拷贝一段放在数组中

第一个参数表示复制位置

第二个参数表在原数组中复制起始位置

第三个参数表示在原数组红赋值的结束位置

[1,2,3,4,5,6,7].copyWithin(2, 4,6)

[1, 2, 5, 6, 5, 6, 7]

|  |
| --- |
| 1. var result = F4.find(function (value, index, arr) { 2. // 返回的是判断条件 3. return value.indexOf('123尼古拉斯') >= 0; 4. }) |

# 对象拓展

对象定义拓展

可以通过对象字面量来定义对象

1 在ES6中，如果定义对象时候

属性名称与变量名称相同可以省略属性名称

2 在ES6中，定义属性名称时候，可以使用表达式

3 在ES6中，定义方法，如果方法名称与函数名称相同，可以省略属性名称

可以像在类中定义方法一样定义对象的方法

省略function关键字

|  |
| --- |
| 1. // 省略属性名称 2. var obj = { 3. color, 4. title 5. } 6. // // 定义属性名称可以表达式 7. var obj = { 8. // choosegreen:green 9. ['choose' + color]: color 10. } 11. // 定义方法的拓展 12. var obj = { 13. color: color, 14. getColor() { 15. return this.color; 16. } 17. } |

# Is方法

用来判断数据的

对象拓展的，因此调用：Object.is

以前判断数据是否相等，更准确的方式用===

1 不能判断0和-0

存储地址是不同的，但是===判断不出来，我们可以通过is方法来判断

2 判断NaN有问题

在数字类型中，只有一个NaN，===判断结果是false

我们可以通过is方法来判断NaN结果是否是同一个

其他判断跟===基本是一样的

|  |
| --- |
| 1. // 判断正负0 2. console.log(0 === -0) 3. console.log(Object.is(0, -0)) 4. // 判断NaN 5. console.log(+'abc' === 0/0) 6. console.log(Object.is(+'abc', 0/0)) |