Let

特性：

1、不能重复声明

2、有块级作用域

const

特定：

1、声明的时候就马上初始化

2、声明后不能改变

3、如果是一个对象，可以改变对象的属性

const a = {};

a.b = 9;

console.log(a); //{b: 9}

函数方面的扩展

1、函数默认值

2、箭头函数（如果函数只有一个参数可以不写括号）

3、箭头后面可以直接写返回值

4、如果想返回一个对象，可以用括号括起来

let a = (a, c) => ({a: 1, b: 2, c: a, d: c});

5、this的指向问题，this指向会是函数声明时函数所在的上下文环境

模板字符串

声明模板字符串使用反引号，你可以使用${}在里面使用表达式

let firstName = ‘Yzs’;

let s = `My name is ${firstName}`;

s = `My name is ${ 3 > 2 ? ‘Yzs’ : ‘Fsn’}`;

变量的结构赋值

在函数参数里面可以直接解构

function logNames ({firstName, lastName}) {

console.log(`${firstName} ${lastName}`);

}

spread-rest

spread展开

在函数参数里面去展开一个数组

function fn (a, b, c) {

console.log(a, b, c);

}

let arr = [2, 3, 4];

fn(...arr);

fn(...[2, 3, 4]);

可以在一个数组里面展开另外一个数组

console.log([‘a’, ‘b’, ...arr, ‘c’, ‘d’]);

展开一个对象,后面的属性会覆盖前面的属性

let obj = {a: 1, b: 2};

console.log({

a: 89,

name: ‘flowke’,

...obj

});

rest剩余

在函数形参里面使用

function fun2 (a, ...b) {

console.log(a, b);

}

fun2(1, 2, 3); // 1 [2, 3]

在解构一个对象的时候， 使用rest

let obj = {a: 1, b: 2, c: 3, d: 4};

let {a, c, ...g} = obj;

console.log(a, c, g); // 1 3 {b: 2, d: 4};

类以及类的继承

function Human () {

this.eyes = 2;

this.hands = 2;

}

Human.prototype.sing = function () {

console.log(‘I can sing’);

}

let a = new Human();

console.log(a);

a.sing();

Es6

使用class关键字去声明类

类可以继承另外一个类，使用extends语法。可以继承到另外一个类的所有属性和方法

class Human {

// 就是类的构造函数，初始化一个对象的属性

constructor (eyes=2, hands=2) {

this.eyes = eyes;

this.hands = hands;

}

singing () {

console.log(‘I can sing’);

}

}

let a = new Human(3, 4);

a.singing();

class NormalMan extends Human {

constructor (name=”Folke”) {

// 调用父类的构造函数

// 当你继承了另外一个类的时候，你必须用

super(1000, 2000);

// this.eyes = 100;

// this.hands = 1;

this.feet = 2;

this.name = name;

this.singsing = this.singing.bind(this);

}

// 重写父类继承过来的方法

singing () {

console.log(‘${this.name} can sing’);

}

}

let b = new NormalMan();

console.log(b);

关于类里面方法的this指向

let fnn = b.singing;

fnn(); // 此时this报错为undefined,绑定了this.singsing = this.singsing.bind(this)后，解决

模块化语法ES6

输出： ES6的导出语法

导出的东西可以没有名字

export default只能使用一次

export default function () {console.log(‘’)};

export default function b () {

console.log(‘’);

}

// 可以使用多次 用export来导出

export let num = 789;

export let bvc = 123;

接收默认导出

接收默认导出的时候，可以任意命名

导入： import a from ‘./modules/a’

import{num, bvc} form ‘./modules/b’

模块化语法-CommonJs

这是CommonJS模块化规范的导出，nodejs就是使用这种模块化规范的语法

moudule.exports = 123;

moudule.exports = {

a: 1,

b: 2,

}

以下是CommonJs的导入

const module = =require(‘./CommonJs’);

数组方法-map（生成新数组）

let arr = [1, 2, 3, 4].map((elt, i) => {

return ‘g’ + elt;

})

console.log(arr);

数组的方法-filter(过滤数组)

如果返回值为true，当前遍历到的元素保留下来，false：当前遍历到的元素就会被移除出去

let arr = [1, 2, 3, 4].filter((elt, I) => {

return elt > 3;

})

console.log(arr);

数组方法-some

返回一个布尔值

回调函数返回值需要一个bool，true：只要有一个返回true，整体返回true，false： 所有的都返回false，才返回false

let arr = [1, 2, 3, 4].some((elt, i) => {

return elt > 3;

})

console.log(arr); // true

数组方法-every

同some，true:所有的都返回true。false：有一个返回false整体返回false

let arr = [1, 2, 3, 4].every((elt, i) => {

return elt > 3;

})

console.log(arr); // false

数组方法-reduce

第二个参数为accu的初始值

返回值：回调函数最后一次的返回值

没有指定accu初始值，把第一个值设定为初始值

如果数组只有一个值，里面的东西都不会生效，只返回这个第一个元素

如果数组没有值，会报错，如果有初始值返回初始值

1 + 2 + 3 + 4

let arr = [1, 2, 3, 4].reduce((eccu, elt, i) => {

console.log(accu, elt);

return accu + elt;

}, 0)

console.log(arr); //

let arr = [1, 2, 3, 4].reduce((eccu, elt, i) => {

if (elt >1) accu.push(elt);

return accu;

}, [])

运行hello React