# 一 ES6重要特性

## 1 变量定义

定义变量 let name = “lisi”;

定义常量 const PI = “3.14”;

这两种定义方式带来的影响：使变量具备了块级作用域。

for (var i = 0; i < 1; i++) {  
}  
for (let j = 0; j < 1; j++) {  
}  
  
console.log(i); *//打印1*console.log(j); *//报错：j is not defined*

## 2 箭头函数

function (参数,参数){  
 函数体  
}  
  
(参数,参数)=>{  
 函数体  
}

let add = a=>a+3; *//有且仅有1个参数，参数括号可省略*console.log(add(5)); *//函数体只有一句return，大括号可省略*

箭头函数的重要作用：绑定了调用者对象

普通函数：this取决于谁调用 this老变  
 箭头函数：this取决于所处环境，比如window/在某个对象内等 this恒定  
 注意：如果箭头函数使用了bind，this仍然是以箭头函数为依据。

## 3 字符串模板

let json = {name:"lisi", age:"18"};  
console.log(`我是：${json.name}`);

## 4 模块化

默认导出： export default \*\*\*;

声明导出： export let obj = \*\*\*; //可以导出多个

导入： import \*\* from './\*\*.js'

注意：浏览器不支持模块化语法，需要第三方插件或者webpack等打包工具来实现。

# 二 ES6面向对象

## 1 基本写法

class Student {  
  
 constructor(name,){ *//构造函数* this.name = name; *//对象属性* }  
  
 study(){ *//对象方法* console.log(this.name + ' is studying...');  
 }  
  
 static run() {  
 console.log('running...');  
 }  
}  
  
let p = new Student("lisi"); *//创建对象*p.name = "zs"; *//修改属性*p.study(); *//zs is studying...*Student.run(); *//running....*

## 2 set与get

class Person {  
  
 constructor(name){  
 this.name = name;  
 }  
  
 get title(){ //name属性的访问器  
 return this.name;  
 }  
  
 set title(name) { //name属性的设置器  
 this.name = name;  
 }  
 run(){  
 console.log(this.name + ' is running...');  
 }  
}  
  
let p = new Person("lisi");  
p.title = "zs"; //通过设置器设置属性  
p.run(); *//zs is running....*

## 3 对象属性直接写

let name = 'lisi', age = 30;  
let Person = {  
 name,  
 age,  
 run(){  
 console.log('run...');  
 }  
};

注意：当一个对象的属性名有空格，还可以用[]获取：Person[“na me”];

## 4 继承

鹦鹉继承鸟类：

class Bird {  
  
 constructor(name){  
 this.name = name;  
 }  
  
 run(){  
 console.log('Bird can fly!');  
 }  
}  
  
class Nautilus extends Bird{ *//鹦鹉类* constructor(name){  
 super(name);  
 }  
 speak(){  
 console.log(this.name + ' can speak!');  
 }  
}  
  
let t = new Nautilus("Yii");  
t.run();  
t.speak();

# 三 ES6异步回调处理

## 1 ES6 异步回调解决方案

三种：Promiss generator async

## 2 Promiss

### 2.1 Promiss的基础使用

let p = new Promise((resolve,reject)=>{  
 $.ajax({  
 url:'1.txt',  
 dataType:'json',  
 success(json){  
 console.log('正确');  
 resolve(json);  
 },  
 error(err){  
 console.log('错误');  
 reject(err);  
 }  
 });  
});  
  
*//其实这里第一个箭头函数就是resolve 第二个是reject*p.then(json=>{  
 console.log('成功');  
 console.log(json);  
},err=>{  
 console.log('失败');  
});

### 2.2. Promise.all()

let p1=new Promise((resolve, reject)=>{  
 $.ajax({  
 url: '1.txt',  
 dataType: 'json',  
 success(json){  
 resolve(json);  
 },  
 error(err){  
 reject(err);  
 }  
 });  
});  
  
let p2=new Promise((resolve, reject)=>{  
 $.ajax({  
 url: '2.txt',  
 dataType: 'json',  
 success(json){  
 resolve(json);  
 },  
 error(err){  
 reject(err);  
 }  
 });  
});  
  
Promise.all([p1, p2, p3]).then(arr=>{  
 let [j1, j2]=arr;  
  
 alert('成功');  
 console.log(j1, j2);  
}, err=>{  
 alert('失败');  
});

### 2.3 Promise与jQuery

jQuery本身返回的其实是一个Promise对象。

Promise.all([  
 $.ajax({url: '1.txt', dataType: 'json'}),  
 $.ajax({url: '2.txt', dataType: 'json'}),  
 $.ajax({url: '3.txt', dataType: 'json'}),  
]).then(arr=>{  
 let [j1, a, j2]=arr;  
  
 console.log(j1, a, j2);  
}, err=>{  
 alert('失败');  
});

### 2.4 Promise总结

Promise可以解除异步操作，但是具有局限性：如果异步操作中带了逻辑，Promise无法解决。

Promise.all() 所有都成功后才会执行后续；

Promise.race() 只要有一个完成就会执行后续操作。

## 3 generator

generator生成器可以用来暂停。

yield：后面一般跟一个Promise对象，将这个对象返回。

注意：generator 函数不能写成箭头函数。

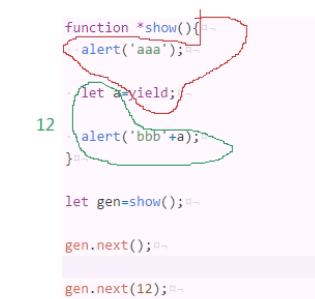
### 3.1 generator展示

function \*show() {  
console.log(11);  
yield;  
console.log(22);  
}  
*// show(); 直接输出没有结果*let gen = show(); *//执行生成器函数得到的是一个生成器对象*gen.next(); *//11 这样的停顿是为了给异步操作一定的时间*setTimeout(function () {  
 gen.next();  
},3000);

### 3.2 yield传参

function\* show() {  
 console.log(11);  
 let a = yield; *//yield传参* console.log(22 + a);  
}  
  
let gen = show();  
gen.next(); *//11*gen.next(10); *//32*

原理：



### 3.3 yield返回值

function \*show(){  
 alert('aaa');  
 yield 55;  
  
 alert('bbb');  
 return 89;  
}  
  
let gen=show();  
  
let res1=gen.next();  
console.log(res1); *//{value: 55, done: false}*let res2=gen.next();  
console.log(res2); *//{value: 89, done: true}*

### 3.4 generator与promise配合

function \*show(){  
 let data1=yield $.ajax({url: '1.txt', dataType: 'json'});  
  
 if(data1.a+data1.b<10){  
 let date2=yield $.ajax({url: '2.txt', dataType: 'json'});  
  
 alert('a'); *//12* }else{  
 let data3=yield $.ajax({url: '3.txt', dataType: 'json'});  
  
 alert('b'); *//blue* }  
}  
runner(show);

## 4 ES7的async

目前最优秀的异步解决方案。

语法：

async function(){

let 结果1 = await 异步操作(Promise generator async)

let 结果2 = await 异步操作(Promise generator async)

....

}

await后可以跟各种异步操作，当然也可以跟一个数值，这样就没有异步操作了，会直接跳过。

基本示例：

async function show(params) {

console.log(1);

await new Promise(function(resolve,rejec) {

setTimeout(function () {

resolve();

},3000);

});

console.log(2);

await new Promise(function(resolve,rejec) {

setTimeout(function () {

resolve();

},3000);

});

console.log(3);

}

show();

function sleep(sec){  
 return new Promise((resolve, reject)=>{  
 setTimeout(function (){  
 resolve();  
 }, sec\*1000);  
 })  
}

(async ()=>{  
 let data1 = await $.ajax({url: '1.txt', dataType: 'json'});  
 if(data1.a + data1.b < 10){  
 let data2 = await $.ajax({url: '2.txt', dataType: 'json'});  
 alert(data2[0]);  
 }else{  
 let data3 = await $.ajax({url: '3.txt', dataType: 'json'});  
 alert(data3.name);  
 }  
})();

async function show(){  
 try{  
 let data1=await $.ajax({url: '1.txt', dataType: 'json'});  
 let data2=await $.ajax({url: '33.txt', dataType: 'json'});  
 let data3=await $.ajax({url: '3.txt', dataType: 'json'});  
  
 console.log(data1, data2, data3);  
 }catch(e){  
 alert('有问题');  
 throw new Error('我错了....');  
 }  
}  
show();

# 四 ES6其他特性

## 1 字符串新增方法

str.startsWith();

str.endsWith();

## 2 默认参数

let show2 = (a,b=5,c=5)=>{  
 console.log(a,b,c); *//省去了书写 var b = 5 || 0;*};  
show2(1,3);

## 3 参数展开

用法一：接收剩余参数（参数必须位于参数列表最后）

function show(a,b,...arg) {  
 console.log(arg);  
};  
show(1,2,3,4,5); *//arg成为[3,4,5]*

用法二：展开数组

let arr1 = [6,7,8];  
let arr2 = [1,2,...arr1,3];  
console.log(arr2);

## 4 获取函数名

let fn = function(){}

console.log(fn.name); //打印fn

## 5 Object.is

console.log(+0 === -0); *//true*console.log(Object.is(+0, -0)); *//false*console.log(NaN == NaN); *//false*console.log(Object.is(NaN, NaN)); *//true*

## 6 数组

### 6.1 map方法 映射

let arr1 = [65,23,12,76,89];  
let arr2 = arr1.map(function (item) {  
 return item+1;  
});  
console.log(arr2); *//[ 66, 24, 13, 77, 90 ]*

简写：

let arr3 = arr1.map(item=>item+1);

### 6.2 reduce方法 堆积

let arr = [7,4,9];  
let arrNew = arr.reduce((temp,item,index)=>{  
 console.log("temp=" + temp);  
 console.log("item=" + item);  
 console.log("index=" + index);  
 return temp + item;  
});  
console.log("arrNew=" + arrNew);  
*// temp=7 item=4 index=1  
// temp=11 item=9 index=2   
// arrNew=20*

### 6.2 filter方法 过滤

let arr1 = [12,5,7,9,8];  
let arr2 = arr1.filter(item=>item%2); *//去除掉偶数*console.log(arr2);

### 6.3 forEach方法 遍历

### 6.4 Array.from()

Array.from(伪数组) 将伪数组转换from转换后，可以使用数组的方法

## 7 set

let people = new Set('甲乙乙丙丁');  
people.add('庚');  
console.log(people); *//Set { '甲', '乙', '丙', '丁', '庚' }*console.log(people.size); *//5*console.log(people.has('丁')); *//true*people.delete('乙');  
people.forEach(p => {  
 console.log(p);  
});  
people.clear();  
console.log(people); *//Set {}*

## 8 map

let food = new Map();  
let fruit = {};  
let cook = function(){};  
  
food.set(fruit,"苹果");  
food.set(cook,function () {  
 console.log('cook...');  
});  
  
console.log(food); *//Map { {} => '苹果', [Function: cook] => [Function] }*console.log(food.size); *//2*console.log(food.get(cook)); *//[Function]*food.delete(fruit);  
food.forEach((v, k) => {   
 console.log(`${k} = ${v}`) *//function (){} = function () {console.log('cook...');}*});  
food.clear();

## 9 iterator迭代器

手工创建迭代器：

function chef(foods) {  
 let i = 0;  
 return {  
 next() {  
 let done = (i >= foods.length);  
 let value = !done ? foods[i++] : undefined;  
 return {  
 value: value,  
 done: done  
 }  
 }  
 }  
}  
  
let test = chef(["lisi","zs"]);  
console.log(test.next()); *//{ value: 'lisi', done: false }*console.log(test.next()); *//{ value: 'zs', done: false }*

## 10 generator

迭代器的生成器：

function\* chef() { *//生成器* yield 'lisi';  
 yield 'zs';  
}  
  
let test = chef(); *//迭代器*console.log(test.next()); *//{ value: 'lisi', done: false }*console.log(test.next()); *//{ value: 'zs', done: false }*