ES6规范

数据类型的声明，除了原有的var，新增const、let、class。同时新增“解构”操作，对变量进行赋值。

基础数据类型除了原有的六种（string、boolean、number、null、undefined、object）,新增Symbol类型，基本数据类型都有新增的属性和方法。

提供对象基本操作的API，用户可以对get、apply、constructor等行为进行底层操控。

数组提供for...of...操作，能够进行generator操作。

提供generator，优化遍历操作。

提供Promise操作，将异步操作执行同步任务。

# 声明（declare）

## 1.1 const

静态资源变量声明，不能修改变量的指向。

## 1.2 let

### 1.2.1不能重复声明，没有变量提升。

先定义后使用，变量定义不会自动提升，否则会报变量未定义的错误。

### 1.2.2块作用域。

每个大括号被视为一个作用域。逐级外层作用域查找，直到找到为止。

### 1.2.3 for语句视为一个作用域

//demo\_01

(function(){

var a = [];

for (let i = 0; i < 10; i++) {

a[i] = function () {

console.log(i);

};

}

a[6](); // 6

})();

For的特殊使用：

//demo\_02

(function(){

for(let i = 0;i<3;i++){

let i = "abc";

console.log(i)

}

for(let i = 0;i<3;i++){

console.log(i)

}})();

# 基本数据类型

JS基本数据类型有String、Number、Boolean、Null、Undefined，ES6新增Symbol的基本数据类型。

## 2.1 String

### 2.1.1 扩展

startsWith、endsWith、padStart、padEnd、repeat

### 2.1.2 Unicode编码

对于大于\uFFFF的数值，通过\u{20BB7}可以正常识别。

ES5的charAt不能识别大于\uFFFF的字符，ES6提供at。

### 2.1.3 字符与Unicode之间的转换

codePointAt //返回字符的Unicode的编码

String.fromCodePoint() //返回Unicode编码对应的字符

### 2.1.2 模板字符串

//demo\_03

(function(){

let [name,time] = ["Bob","today"];

let str = `Hello ${name}, how are you ${time}?`

console.log(str)

})();

## 2.2 Number

isFinite、isNaN、isInteger

//demo\_04

let a = Number.isNaN("abc")

let b = Number.isNaN(Number("abc"));

console.log(a,b) //false true

## 2.3 Null 传导运算符

//尚未实现

let firstName = message?.body?.user?.firstName || 'default';

console.log(firstName)

## 2.4 Symbol

表示独一无二的值，没有两个相等的Symbol。一般用于对象的属性名称，避免属性的误操作。

//demo\_05

let [a,b] = [Symbol(),Symbol()];

let [c,d] = [Symbol('abc'),Symbol('abc')];

let [e,f] = [Symbol.for('swl'),Symbol.for('swl')];

let rs\_a\_b = Object.is(a,b);

let rs\_c\_d = Object.is(c,d);

let rs\_e\_f = Object.is(e,f);

console.log(rs\_a\_b,rs\_c\_d,rs\_e\_f) //false false true

# 三、引用数据类型

JS引用数据类型Array、Object，新增Set、WeakSet、Map、WeakSet。

## 3.1解构

### 3.1.1 数组解构

赋值操作：

(function(){

let [a,b,c] = [1,2,3];

console.log(a,b,c);

let [d,...e] = [11,22,33,]; //rest操作

console.log(d,e);

})();

默认值：

(function(){

let [x, y = 'b',z = 13] = ['a',undefined,]; // undefined或者不写任何东西，默认值才生效

console.log(x,y,z);

})();

惰性执行：

//如果默认值是一个表达式，那么这个表达式是惰性求值的，即只有在用到的时候，才会求值。

(function(){

function a(){

console.log("\*\*");

return 5;

}

let [aa=a()] = []; //会输出“\*\*”

let [bb=a()] = [1]; //不会输出“\*\*”

})();

### 3.1.2 对象解构

两种解构方式：

(function(){

let { foo } = { foo: "aaa", bar: "bbb" };

console.log(foo) ///aaa

let { foo: bar ,bar:foe} = { foo: "aaa", bar: "bbb" }; //通过模式解构

console.log(bar,foe) //aaa bbb

})();

嵌套解构：

(function(){

let obj = {

p: [

'Hello',

{ y: 'World' }

]

};

let { p: [x, { y }] } = obj;

console.log(x,y) //Hello World

})();

字符串解构：可以按照数组和对象类型解构

(function(){

let str = "qwert"

let [a,b,c,d] = str;

let {length:len} = str;

console.log(a,b,c,d,len)

})();

## 3.2 数组

新增遍历：for...of...、entries、keys、values

### 3.2.1 for...of...

这是最简洁、最直接的遍历数组元素的语法；

这个方法避开了for-in循环的所有缺陷；

与forEach()不同的是，它可以正确响应break、continue和return语句。

解构：

let arr = [{name:'abc',id:1},{name:'def',id:2},{name:'hij',id:3}]

for(let {id,name} of arr){

console.log(id,name)

}

### 3.2.2 entries、keys、values

返回一个迭代器，通过next进行操作。而Object.entries、Object.keys、Object.values，返回的只是一个数组，不能使用next操作。

var arr = ["a", "b", "c"];

let a = arr.entries(); //迭代器也可以使用for...of...遍历

console.log(a.next()) //{ value: [ 0, 'a' ], done: false }

console.log(a.next()) //{ value: [ 1, 'b' ], done: false }

console.log(a.next()) //{ value: [ 2, 'c' ], done: false }

console.log(a.next()) //{ value: undefined, done: true }

## 3.3 ArrayBuffer

存储二进制数据，主要在前端会使用：File API、XMLHttpRequest、Fetch API、Canvas、WebSockets

## 3.4 Object

### 3.4.1 声明

(()=>{

let [name,obj\_a,obj\_b] = ['abc',{},{}];

let obj = {

name,

[obj\_a]:'obj a',

[obj\_b]:'obj b', // '[object Object]':'obj b'

['desc']:'this is description',

[Symbol('sym')]:'this is a Symbol'

}

console.log(obj)

})();

### 3.4.2 Object.is(a,b)

返回Boolean变量，它用来比较两个值是否严格相等，与严格比较运算符（===）的行为基本一致。

(()=>{

Object.is('foo', 'foo')

// true

Object.is({}, {})

+0 === -0 //true

NaN === NaN // false

Object.is(+0, -0) // false

Object.is(NaN, NaN) // true

})();

### 3.4.3 Object.entries、Object.keys、Object.values

返回由对象的键值对、健、值组成的数值，可使用for...of...遍历。注意：若参数不是个对象，隐式转化为对象。Boolean、Number不能被转化为对象，所以返回一个空数组。

let a = true;

console.log(Object.entries(a)) //[ ]

console.log(Object.keys(a)) //[ ]

console.log(Object.values(a)) //[ ]

### 3.4.4 原型与属性获取

|  |  |  |
| --- | --- | --- |
| 方法 | 描述 | 备注 |
| Object.getPrototypeOf() | 获取原型属性 | 不能获取\_\_proto\_\_设置的属性 |
| Object.setPrototypeOf() | 设置原型属性 |  |
| Object.getOwnPropertyDescriptors() | 获取自身属性 | { name:  { value: 'swl',  writable: true,  enumerable: true,  configurable: true }} |
| Object.getOwnPropertySymbols() | 获取自身的Symbol类型属性 |  |
| Object.getOwnPropertyNames() | 获取自身的属性名 | 不包含Symbol |
| Reflect.ownKeys(obj) | 获取非原型属性名 |  |

### 3.4.5 Object.assign

合并对象，如果参数不是对象，则先转化为对象。Null和undefined不能转化对象，会报错。

执行浅拷贝...

### 3.4.6 Map和WeakMap

(()=>{

let map = new Map([["name","swl"],["age",12],["sex","boy"]]);

for(let [key,value] of map){

console.log(`key: ${key} ,value: ${value}`)

}

let entries = map.entries();

for(let [key,value] of entries){

console.log(`key: ${key} ,value: ${value}`)

}

})();

Map可接受对象作为key。

WeakMap 与 Map 在 API 上的区别主要是两个，一是没有遍历操作（即没有key()、values()和entries()方法），也没有size属性。因此，WeakMap只有四个方法可用：get()、set()、has()、delete()。一般用于存储HTML DOM节点。

### 3.4.7 Set和WeakSet

(()=>{

let set = new Set(['a','b','c']);

for(let ele of set){

console.log(ele)

}

let entries = set.entries();

for(let [key,value] of entries){

console.log(key,value)

}

})();

Set可接受NaN值。

WeakSet只接受对象，一般用于存储HTML DOM节点。WeakSet 不能遍历（即没有key()、values()和entries()方法），也没有size属性。WeakSet只有四个方法可用：add()、has()、delete()。

# 箭头函数

箭头函数没有自己的this，this指向宿主环境。若没有宿主环境，并且显示调用’use strict’，this为undefined；非显式调用，this指向window（global）。

箭头函数不能使用new。

let getName = name => name;

console.log(getName('swl')); //swl

let sum = (a,b) => {

let sum = a + b;

return sum;

}

console.log(sum(2,3)); //5

# 拦截器 Proxy

Proxy 用于修改某些操作的默认行为，等同于在语言层面做出修改，所以属于一种“元编程”（meta programming），即对编程语言进行编程。Proxy 可以理解成，在目标对象之前架设一层“拦截”，外界对该对象的访问，都必须先通过这层拦截，因此提供了一种机制，可以对外界的访问进行过滤和改写。

(()=>{

const log = console.log;

var proxy = new Proxy({}, {

get(target, key, receiver) {

return `<span data-key='${key}'>${target[key]}</span>`;

},

set(target, key, value,receiver){

return Reflect.set(target, key, `TEST-${value}`,receiver);

}

});

proxy.name = 'profile';

proxy.time = '2017/9/9';

proxy.action = 'download';

log(proxy.time); //<span data-key='time'>TEST-2017/9/9</span>

log(proxy.name); //<span data-key='name'>TEST-profile</span>

log(proxy.action); //<span data-key='action'>TEST-download</span>

})();

构造器参数：对象，{}或者[ ]；处理器对象，可选处理方法get、set、apply、has...

# 五、Iterator和Generator

ES6规定对象只要具有Symbol.iterator属性，那么此对象就是可遍历的。Symbol.iterator的返回值是一个遍历器，可以通过next()和for...of...遍历。

(()=>{

let arr = ['a','b','c'];

let gen = arr[Symbol.iterator]()

console.log(gen.next())

console.log(gen.next())

console.log(gen.next())

console.log(gen.next())

console.log(gen.next())

})();

用户可以自定生成器，返回一个遍历器对象。

(()=>{

function\* he(){

yield 'hello';

yield 'world';

}

let a = he();

console.log(a.next())

console.log(a.next())

console.log(a.next())

})();

# 六、Promise

Promise具有三个状态，pending（进行中）、resolved[fulfilled]（已完成）、rejected[rejected]（已拒绝）。

未完成

已拒绝

已完成

new Promise((resolve,reject)=>{

resolve('fulfilled...')

}).then((val)=>{

console.log(val)

}).catch(error => {

console.log(error)

})

Method:

Promise.all()、Promise.race()、Promise.resolve()、Promise.reject()

# 七、async/await

同步的实现，从回调到Generator，再到Promise，再到async/await。

Node 7.6版本支持async/await的写法。

let timeout = (t)=>{

return new Promise(resolve=>{

setTimeout(resolve,t)

})

}

(async ()=>{

console.log(`两秒之后打印:`)

await timeout(2000)

console.log(`hello world`)

})();

# Class

## 8.1 构造器

class Persion {

constructor(name,age){

this.name = name;

this.age = age;

}

getName(){

return this.name;

}

static getDesc(){

return `this is Persion class`;

}

}

let persion = new Persion('bob',11);

console.log(persion.getName()); //bob

console.log(Persion.getDesc()); //this is Persion class

typeof Persion == ‘function’;

Constructor、普通方法是定义在class的原型上的，等同于Persion.prototype = {constructor,getName};静态方法，等同于Persion.getDesc = func;

## 8.2 class表达式

llet tool = class Tool {

static print(){

return Tool.getDesc();

}

static getDesc(){

return `this is Tool class`;

}

}

console.log(tool.getDesc()); //this is Persion class

Tool为内部使用，tool为外部使用。若不需要引用Tool，则可以省去Tool，即let tool = Class {......};

## 8.3 静态方法、私有方法、私有属性

静态方法：static+方法名，如，static getDesc

私有方法：\_+方法名，如，\_getName(){}

私有属性：#+属性名，如，#name

仅是一种提议，未实现。

## 8.4 new.target

返回：new命令作用于的那个constructor。

class Tool {

constructor(){

if(new.target == Tool){

throw new Error('this class can not be instanced')

}

}

static getLocalTime(){

return new Date().toLocaleString();

}

}

new Tool() //抛出一个Error

console.log(Tool.getLocalTime())

创建一个不允许被实例化的工具类。

# 类的继承

## 9.1 extends、supper

const log = console.log;

class Persion{

constructor(name,age){

this.name = name;

this.age = age;

}

toString(){

return `the presion's name is ${this.name} ,and age is ${this.age}`;

}

static getDesc(){

return 'this is Persion class'

}

}

class Man extends Persion{

constructor(name,age){

super(name,age);

}

toString(){

return super.toString(); //super指向父类的原型

}

static getOwnDesc(){

return super.getDesc();

}

}

let m = new Man("swl",2343);

log(m.toString()) //the presion's name is swl ,and age is 2343

log(Man.getOwnDesc()) //this is Persion class

log(Man.getDesc()) //this is Persion class

supper：既可以当方法用，也可以当做对象使用。

super作为函数调用时，代表父类的构造函数。ES6规定，必须在子类的constructor中调用一次supper，否则，报错。

super作为对象使用时，在普通方法中，指向父类的原型对象；在静态方法中，指向父类。

## 9.2 extends

extends A。

A可以是多种类型的值，只要A具有prototype就可以。由于函数都有prototype属性（除了Function.prototype函数），因此A可以是任意函数。