链滴

ES6 语法简介

作者：Lee981265

原文链接：https://ld246.com/article/1562999386176

来源网站：链滴

ECMAScript是一种由Ecma国际（前身为欧洲计算机制造商协会,英文名称是European Computer Manufacturers Association）通过ECMA-262标准化的脚本程序设计语言。这种语言在万维网上应用广，它往往被称为JavaScript或JScript，但实际上后两者是ECMA-262标准的实现和扩展。<!--more-->

ES6

变量声明

- let:代码块内的变量声明
  1. 变量声明不会提升
  2. 块级作用域
  3. let不允许相同作用域内多次声明同一变量

- const:常量声明
  1. 声明后无法修改值
  2. 块级作用域
  3. const不允许相同作用域内多次声明同一变量

解构赋值

ES6允许我们对数组和对象中提取值，对变量进行赋值，这被叫做“解构”

- 数组：

```
1. var [a,b,c] = [1,2,3] //a=1,b=2,c=3
2. var [a,[[b],c]] = [10,[[20],30]] //a=10,b=20,c=30
3. var [a,...b] = [1,2,3,4] //a=1,b=[2,3,4];...表示剩余参数
```

- 对象：
  1. var {a,b}={a:’ html’ ,b:’ css’ }
  2. 变量必须与对象属性名相同，否则为undefined
  3. 如果变量名与属性名不相同，则必须写成以下格式才能取到值

```
var {a:test} ={a:'html',b:'css'} //test=>html
```

- 解构失败：

```
var [a]=[],[b]=1,[c]='jiegou',[d]=false //a,b,c,d都得到undefined
```

- 指定默认值：

```
var [a=true]=[]

var {a=10} = {}
```

> 解构同样适用于let和const
> PS:解构只能用于数组和对象，如果解构不成功，变量会返回undefined，但如果对undefined和nul
解构则会报错

解构用途

● 交换变量值

```
var [x,y] = [y,x];
```

● 函数返回多个值

```
function example(){
        return [1,2,3]
}
//接收
var [x,y,z] = example();
```

● 定义函数形参（重点）

函数的参数定义方式, 不用再考虑参数的顺序

```
function test({x,y,z}){}
```

//传参

```
test({x:10,y:20,z:30})
//参数可以设置默认值
fuction test({x=10,y=20,z}){}
```

字符串扩展

字符串方法

● includes

判断是否包含某个字符，返回布尔值

● startsWith/endsWith

是否以某一字符开头/结尾

```
  let str='google';
  str.startsWith('g');  //true
  str.endsWith('le');    //true
```

● repeat(n)

得到字符串重复n次后的结果，n可以为小数，但不能为负数

```
'laoxie'.repeat(2);//laoxielaoxie
```

字符串模板, 使用反引号`表示(重点)

你可以通过一种更加美观、更加方便的方式向字符串中插入变量

格式：${变量|函数},

```
  你好，我的名字叫${username},接下来是我的自我介绍：${introduce()}
```

模板字符串中所有的空格、新行、缩进，都会原样输出在生成的字符串中。

数组扩展

遍历

- for..of

```
var arr = [10,12,18,30]
  for (var value of arr) {
    console.log(value);
  }
```

- 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> for...of跟for-in的区别很明显，就是直接取值，而不再取标了
- 与forEach()不同的是，它可以正确响应break、continue和return语句
- for-of循环不支持普通对象
- for-of循环也可以遍历其它的集合
- DOM节点
- 字符串
- Set/Map集合

对象扩展

- Object.assign(obj1,obj2,...objN);

合并对象

```
Object.assign({a:1},{b:2},{b:4,c:3}); //{a:1,b:4,c:3}
```

- 只支持浅拷贝（对于引用类型，只拷贝引用）
- 忽略不可枚举属性
- 简写

ES6允许在对象之中直接写变量，如

```
    //@属性简写
    var myName = 'laoxie';
    var obj = {myName};//属性名为变量名, 属性值为变量的值
```

```
    //等效于
    var obj = {myName:'laoxie'}

    //使用变量值作为属性名
    var obj = {
        [myName]:18
    }
    //等效于
    var obj = {laoxie:18}

    //@方法简写
    var obj = {
        coding(){

        }
    }
    //等效于
    var obj = {
        coding:function(){

        }
    }
```

箭头函数=>（重点）

格式：标识符=>表达式

省略了function、return关键字和大括号。

参数与返回值
* 零个参数用 () 表示

```js
  //传统写法
    var sum = function(){
        return 10 + 10;
    }
    //es6箭头函数
    var sum = () => 10+10;
```

- 函数只有一个参数（可省略括号）

```
//传统函数写法
var test = function(x){
    return x+2;
}    //使用箭头函数
var test = x=>x+2;
```

- 多个参数

```
  // ES5
    var total = values.reduce(function (a, b) {
      return a + b;
```

```
}, 0);
// ES6
var total = values.reduce((a, b) => a + b, 0);
```

- 函数中包含多行代码时用代码块括起来

ES6中的规则是，紧随箭头的{被解析为块的开始，而不是对象的开始

```
// ES5
 $("#confetti-btn").click(function (event) {
  playTrumpet();      fireConfettiCannon();
});
// ES6
$("#confetti-btn").click(event => {
  playTrumpet();
  fireConfettiCannon();
});
```

- 使用了块语句的箭头函数不会自动返回值，你需要使用return语句将所需值返回
- 当使用箭头函数返回一个普通对象时，需要将对象包裹在小括号里

```
//传统写法
    var createPerson = function(){
        return {name:'laoxie',age:18}
    }
    // ES6
    var createPerson = ()=>{name:'laoxie',age:18};   // 这样写会报Bug!
    var createPerson = ()=>({name:'laoxie',age:18});
```

- 默认参数。

```
var func1 = (x = 1, y = 2) => x + y;
 func1(); // 得到 3
```

- 剩余参数

```
var func2 = (x, ...args) => { console.log(args) };
 func2(1,2,3); // 输出 [2, 3]
```

箭头函数中的this值

箭头函数没有它自己的this值，箭头函数内的this值继承自外围作用域

Symbol数据类型

ES6引入了一种新的原始数据类型Symbol，表示独一无二的值，一旦创建后就不可更改。

```
// 没有参数的情况
    var s1 = Symbol();
    var s2 = Symbol();
```

```
s1 === s2 // false
```

● Symbol函数可以接受一个字符串作为参数，表示对Symbol实例的描述，主要是为了标识和区分，调式非常有用

```
// 有参数的情况
var s1 = Symbol("foo");
var s2 = Symbol("foo");
s1 === s2 // false
```

● Symbol值不能与其他类型的值进行运算

用途

● 给对象创建私有属性

```
var mySymbol = Symbol();
 // 第一种写法
 var a = {};
 a[mySymbol] = 'Nani';
 // 第二种写法（注意加方括号，否则回被当作普通属性）
 var a = {
   [mySymbol]: 'Nani'
 };
 // 以上写法都得到同样结果
 a[mySymbol] // "Nani"


### 常用方法

* Symbol.for()
   有时我们希望重新使用同一个Symbol值，Symbol.for方法可以做到这一点，首先在全局中搜索**登记**的Symbol值，如果有，就返回这个Symbol值，否则就新建并返回一个以该字符串为名称的Symbol值> 直接用Symbol()方法创建的Symbol值不会被登记
```js
let one = Symbol("laoxie");
let two = Symbol.for("laoxie");
//由于创建了两个Symbol值，所以他们不相等
 console.log(one===two);//false
```

● Symbol.keyFor()

获取被登记的Symbol值> 直接使用Symbo()创建的Symbol值的键不会被登记，所以也就获取不到

Map集合

Map集合,即映射

● 设置

```
let map = new Map();
   map.set("S0", "张三");
```
```

```
map.set("S1", "李四");
map.set("S2", "王五");
```

- 获取

```
map.get("s2"); //王五
```

- 循环遍历，配合解构赋值

```
for(let [key,value] of map){
        console.log(key,value);
    }
```

方法

- keys() 获取所有键
- values() 获取所有值
- entries() 获取所有键值对，返回类数组

Set集合

Set集合，类似于数组，但是成员的值都是唯一的，没有重复的值。

```
let imgs = new Set();
    imgs.add(1) ;
    imgs.add(1);
    imgs.add(5);
    imgs.add("5");
    imgs.add(new String("abc")) ;
    imgs.add(new String("abc")) ;

    //打印的结果：  1  5  '5'  'abc'  'abc'
```

Set集合是默认去重复的，但前提是两个添加的元素严格相等，所以5和”5”不相等，两个new出来字符串不相等

- [案例] 去重数组

SET集合没有提供下标方式的访问，因此只能使用for...of来遍历。由于Set集合本质上还是一个map因此会有以下几种遍历方法

```
var imgs = new Set(['a','b','c']); //根据KEY遍历
for(let item of imgs.keys()){
     console.log(item);
}
//a
//b
//c

//根据VALUE遍历
for(let item of imgs.values()){
    console.log(item);
```

```
}
//a
//b
//c

//根据KEY-VALUE遍历
for(let item of imgs.entries()){
    console.log(item);
}
//['a','a']
//['b','b']
//['c','c']

//普通for...of循环
for(let item of imgs){
    console.log(item);
}
//a
//b
//c
```