**LET命令**

1. 基本用法

ES6新增了let命令，用来声明变量。它的用法类似于var，但是所声明的变量，只在let命令所在的代码块内有效。

在for循环的时候很适合用let

var a = [];

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

a[i] = function () {

console.log(i);

};

}

a[6](); // 6而不是10

1. 不存在变量提升

console.log(foo); // 输出undefined

console.log(bar); // 报错ReferenceError

var foo = 2;

let bar = 2;

1. 暂时性死区

只要块级作用域内存在let命令，它所声明的变量就“绑定”（binding）这个区域，不再受外部的影响。

if (true) {

tmp = 'abc'; // ReferenceError

let tmp;

}

使用let命令声明变量之前，该变量都是不可用的。这在语法上，称为“暂时性死区”（temporal dead zone，简称TDZ）。“暂时性死区”也意味着typeof不再是一个百分之百安全的操作。

typeof x; // ReferenceError而以前会输出’underfined’

let x;

有些“死区”比较隐蔽，不太容易发现。

function bar(x = y, y = 2) {

return [x, y];

}

bar(); // 报错，在y定义之前使用y

1. 不允许重复声明

// 报错

function () {

let a = 10;

var a = 1;

}

// 报错

function () {

let a = 10;

let a = 1;

}

// 报错

function () {

var a = 10;

let a = 1;

}

不能在函数内部重新声明参数。

function func(arg) {

let arg; // 报错

}

function func(arg) {

**{**

let arg; // 不报错

**}**

}

**块级作用域**

ES5只有全局作用域和函数作用域，没有块级作用域，这带来很多不合理的场景。

第一种场景，内层变量可能会覆盖外层变量。

var tmp = new Date();

function f() {

console.log(tmp);

if (false) {

var tmp = "hello world";

}

}

f(); // undefined，因为在函数里，var声明提前了

第二种场景，用来计数的循环变量泄露为全局变量。

For循环的i变量没有被销毁。

1. Es6块级作用域

function f1() {

let n = 5;

if (true) {

let n = 10;

}

console.log(n); // 5

}

上面的函数有两个代码块，都声明了变量n，运行后输出5。这表示外层代码块不受内层代码块的影响。如果使用都用var定义变量n，最后输出的值就是10。

块级作用域的出现，实际上使得获得广泛应用的立即执行匿名函数（IIFE）不再必要了。

// IIFE写法

(function () {

var tmp = ...;

...

}());