Es6笔记

**一、Let命令和const命令**

1. **Let命令**

不存在变量提升

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。

ES6明确规定，如果区块中存在let和const命令，这个区块对这些命令声明的变量，从一开始就形成了封闭作用域。凡是在声明之前就使用这些变量，就会报错。

总之，在代码块内，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）。

if (true) {

// TDZ开始

tmp = 'abc'; // ReferenceError

console.log(tmp); // ReferenceError

let tmp; // TDZ结束

console.log(tmp); // undefined

tmp = 123;

console.log(tmp); // 123

}

typeof x; // ReferenceError

let x;

typeof undeclared\_variable // "undefined"

变量x使用let命令声明，所以在声明之前，都属于x的“死区”，只要用到该变量就会报错。因此，typeof运行时就会抛出一个ReferenceError。

undeclared\_variable是一个不存在的变量名，结果返回“undefined”

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

上面代码中，调用bar函数之所以报错（某些实现可能不报错），是因为参数x默认值等于另一个参数y，而此时y还没有声明，属于”死区“。如果y的默认值是x，就不会报错，因为此时x已经声明了。

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

return [x, y];

}

bar(); // [2, 2]

不允许重复声明

1. **块级作用域**
2. function f1() {
3. let n = 5;
4. if (true) {
5. let n = 10;
6. }
7. console.log(n); // 5
8. }

外层代码块不受内层代码块的影响。如果使用var定义变量n，最后输出的值就是10。

ES6允许块级作用域的任意嵌套。

{{{{{let insane = 'Hello World'}}}}};

内层作用域可以定义外层作用域的同名变量。

{{{{

let insane = 'Hello World';

{let insane = 'Hello World'}

}}}};

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

块级作用域与函数

ES5规定，函数只能在顶层作用域和函数作用域之中声明，不能在块级作用域声明。

// 情况一

if (true) {

function f() {}

}

// 情况二

try {

function f() {}

} catch(e) {

}

上面代码的两种函数声明，根据ES5的规定都是非法的。

但是，浏览器没有遵守这个规定，还是支持在块级作用域之中声明函数，因此上面两种情况实际都能运行，不会报错。不过，“严格模式”下还是会报错。

// ES5严格模式

'use strict';

if (true) {

function f() {}

}

// 报错

ES6引入了块级作用域，明确允许在块级作用域之中声明函数。

// ES6严格模式

'use strict';

if (true) {

function f() {}

}

// 不报错

并且ES6规定，块级作用域之中，函数声明语句的行为类似于let，在块级作用域之外不可引用。

function f() { console.log('I am outside!'); }

(function () {

if (false) {

// 重复声明一次函数f

function f() { console.log('I am inside!'); }

}

f();

}());

上面代码在ES5中运行，会得到“I am inside!”，因为在if内声明的函数f会被提升到函数头部，实际运行的代码如下。

// ES5版本

function f() { console.log('I am outside!'); }

(function () {

function f() { console.log('I am inside!'); }

if (false) {

}

f();

}());

ES6的运行结果就完全不一样了，会得到“I am outside!”。因为块级作用域内声明的函数类似于let，对作用域之外没有影响，实际运行的代码如下。

// ES6版本

function f() { console.log('I am outside!'); }

(function () {

f();

}());

考虑到环境导致的行为差异太大，应该避免在块级作用域内声明函数。如果确实需要，也应该写成函数表达式，而不是函数声明语句。

// 函数声明语句

{

let a = 'secret';

function f() {

return a;

}

}

// 函数表达式

{

let a = 'secret';

let f = function () {

return a;

};

}

另外，还有一个需要注意的地方。ES6的块级作用域允许声明函数的规则，只在使用大括号的情况下成立，如果没有使用大括号，就会报错。

// 不报错

'use strict';

if (true) {

function f() {}

}

// 报错

'use strict';

if (true)

function f() {}

**3.const命令**

const声明一个只读的常量。一旦声明，常量的值就不能改变。

const PI = 3.1415;

PI // 3.1415

PI = 3;

// TypeError: Assignment to constant variable.

const声明的变量不得改变值，这意味着，const一旦声明变量，就必须立即初始化，不能留到以后赋值。

const foo;

// SyntaxError: Missing initializer in const declaration

const的作用域与let命令相同：只在声明所在的块级作用域内有效。

if (true) {

const MAX = 5;

}

MAX // Uncaught ReferenceError: MAX is not defined

const命令声明的常量也是不提升，同样存在暂时性死区，只能在声明的位置后面使用。

if (true) {

console.log(MAX); // ReferenceError

const MAX = 5;

}

const声明的常量，也与let一样不可重复声明。

对于复合类型的变量，变量名不指向数据，而是指向数据所在的地址。const命令只是保证变量名指向的地址不变，并不保证该地址的数据不变，所以将一个对象声明为常量必须非常小心。

const foo = {};

foo.prop = 123;

foo.prop

// 123

foo = {}; // TypeError: "foo" is read-only

上面代码中，常量foo储存的是一个地址，这个地址指向一个对象。不可变的只是这个地址，即不能把foo指向另一个地址，但对象本身是可变的，所以依然可以为其添加新属性。