let

块级作用域：花括号内的代码范围｛｝

无变量提升：let无变量提升，var具有变量提升的特质

暂时性死区：当前作用域存在let声明的变量，但在声明语句之前，变量存在但不可获取，直到声明语句出现可获取和使用改变量

不允许重复声明：函数内部使用let声明参数报错

块级作用域的意义

- 内部变量可覆盖外部变量
- 用来计数的循环变量泄露为全局变量

es6的块级作用域

- let为js提供了块级作用域
- 块级作用域可任意嵌套
- 外层作用域无法读取内层作用域变量
- 块级作用域使立即执行函数没有必要

块级作用域内部声明函数

1. ES6 的浏览器

- 只在使用大括号的情况下成立允许在块级作用域内声明函数，否则报错
- 函数声明类似于var，即会提升到全局作用域或函数作用域的头部
- 同时，函数声明还会提升到所在的块级作用域的头部
- ES6 的块级作用域允许声明函数的规则

2. 其他环境

- 不用遵守，还是将块级作用域的函数声明当作let处理

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。

const

简介：声明一个只读的常量。一旦声明，常量的值就不能改变，即声明后立即初始化，不能留到以后赋值，否则报错

特性

- 只在声明所在的块级作用域内有效
- 无变量提升，存在暂时性死区

本质：const 变量指向的那个内存地址不得改动。

简单类型：值保存在内存地址，相当于常量

复合类型：固定指针，数据结构不能控制

如果真的想将对象冻结，应该使用Object.freeze方法。

彻底冻结对象

```
var constantize = (obj) => {
  Object.freeze(obj);
  Object.keys(obj).forEach( (key, i) => {
    if ( typeof obj[key] === 'object' ) {
      constantize( obj[key] );
    }
  });
};
```

es6声明变量的六种方法

1. var
2. function
3. let
4. const
5. import
6. class

顶层对象

简介：顶层对象，在浏览器环境指的是window对象，在 Node 指的是global对象。

ES5 之中，顶层对象的属性与全局变量是等价的。

顶层对象的属性与全局变量挂钩，被认为是 JavaScript 语言最大的设计败笔之一。

1. 没法在编译时就报出变量未声明的错误（只有运行时才能知道）；
2. 程序员很容易不知不觉地就创建了全局变量（比如打字出错）；
3. 顶层对象的属性是到处可以读写的，这非常不利于模块化编程。
4. window对象有实体含义，指的是浏览器的窗口对象，顶层对象是一个有实体含义的对象，也是不合适的

ES6的改进？

- var命令和function命令声明的全局变量，依旧是顶层对象的属性
- let命令、const命令、class命令声明的全局变量，不属于顶层对象的属性

从 ES6 开始，全局变量将逐步与顶层对象的属性脱钩。

ES5 的顶层对象在各种实现不统一！！

- 浏览器里面，顶层对象是window，但 Node 和 Web Worker 没有window
- 浏览器和 Web Worker 里面，self也指向顶层对象，但是 Node 没有self
- Node 里面，顶层对象是global，但其他环境都不支持。

this变量可在不同环境获取顶层对象 但有局限性…

- 全局环境中，this会返回顶层对象。但是，Node 模块和 ES6 模块中，this返回的是当前模块。
- 函数里面的this，如果函数不是作为对象的方法运行，而是单纯作为函数运行，this会指向顶层对象。但是，严格模式下，这时this会返回undefined。
- 不管是严格模式，还是普通模式，new Function('return this')()，总是会返回全局对象。但是，如果浏览器用了 CSP（Content Security Policy，内容安全政策），那么eval、new Function这些方法都可能无法使用。

没有一招吃遍天下鲜的办法…

```
// 方法一
(typeof window !== 'undefined'
   ? window
```

```
    : (typeof process === 'object' &&
    typeof require === 'function' &&
    typeof global === 'object')
    ? global
: this);

// 方法二
var getGlobal = function () {
    if (typeof self !== 'undefined') { return self; }
    if (typeof window !== 'undefined') { return window; }
    if (typeof global !== 'undefined') { return global; }
    throw new Error('unable to locate global object');
};
```

现在有一个提案，在语言标准的层面，引入global作为顶层对象。也就是说，在所有环境下，global都是存在的，都可以从它拿到顶层对象；垫片库system.global模拟了这个提案，可以在所有环境拿到global。

《ECMAScript 6 入门》 - 阮一峰

《let 和 const 命令》