2、let和const命令

- ES6新增了let和const来声明变量，主要是解决var声明变量所造成的困扰和问题：
  - var存在变量提升
  - var可以重复声明变量
  - var不支持块级作用域
  - var不能用于定义常量
- 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。
- const声明一个只读的常量。一旦声明，常量的值就不能改变。
- 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。

① 变量的提升问题

- 由var声明的变量存在变量提升
- var声明的变量可以在声明之前使用，相当于默认为其声明其值为undefined

```
function text1(){
    console.log(name); //undefined
    console.log(age);  //undefined
    var name = "zhangsan";
    var age = 20;
    console.log(name); //zhangsan
    console.log(age);  //20
}
text1();

//等价于如下
function text2(){
    var name,age;
    console.log(name); //undefined
    console.log(age);  //undefined
    name = "zhangsan";
    age = 20;
    console.log(name); //zhangsan
    console.log(age);  //20
}
text2();

//注意：在函数内加var为局部变量，不加var则是全局变量（在执行当前函数之后）
```

- let声明的变量一旦用let声明，那么在声明之前，此变量都是不可用的，术语称为“暂时性死区”。

```
console.log(a);   //undefined
//console.log(b); //引用错误ReferenceError: Cannot access 'b' before
initialization
var a = 10;
let b = 20;
console.log(a); //10
console.log(b); //20
```

对'暂时性死区'的理解

- 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）。

```
var tmp ="aaa";
if(true){
    // TDZ开始
    //tmp = 'bbb'; // ReferenceError
    //console.log(tmp); // ReferenceError

    let tmp; // TDZ结束
    console.log(tmp); // undefined

    tmp = "ccc";
    console.log(tmp); // ccc
}
```

② 重复声明变量

- var可以重复声明变量
- let不允许在相同作用域内，重复声明同一个变量。

```
function demo(c){
    var a = 10;
    var a = 20; //可以使用var重复声明var已经声明过的变量a

    //let a = 30; //报错，不可以使用let声明已经被var声明过的变量a
    //错误信息：SyntaxError: Identifier 'a' has already been declared

    let b = 30;
    //let b = 40; //报错,不可以使用let重复声明变量b
    //var b = 50; //报错,不可以使用var声明已经被let声明过的变量
    //SyntaxError: Identifier 'b' has already been declared

    //let c = 70; //报错，不可以使用let重复声明已存在的参数c
```

```
    //SyntaxError: Identifier 'c' has already been declared
}
demo(60);
```

③ 块级作用域

- let命令所在的代码块内有效，并且所用域也仅限于当前代码有效

```
//案例1：使用let声明变量只在代码块中有效
{
    var a = 10;
    let b = 20;
}
console.log(a); //10
//console.log(b); //报错 ReferenceError: b is not defined

//案例2: for循环的计数器，就很合适使用let命令。
for(var i=0;i<10;i++){}
console.log(i); //10

for(let j=0;j<10;j++){}
//console.log(j); //报错：ReferenceError: j is not defined

//案例3
var m = [];
for (var i = 0; i < 10; i++) {
    m[i] = function () {
        console.log(i);
    };
}
m[6](); // 10
//数组m成员里面的i，都指向的是同一个i，导致运行时输出的是最后一轮的i的值，也就是 10

var n = [];
for (let i = 0; i < 10; i++) {
    n[i] = function () {
        console.log(i);
    };
}
n[6](); // 6
//for循环变量的这部分是一个父作用域，而循环体内部是一个单独的子作用域
//而let，声明的变量仅在块级作用域内有效，最后输出的是6
/*
类似于如下格式
var aa = [];
{
    let i = 1;
    {
        let k = i
```

```
        aa[k] = function(){
            console.log(k)
        }
    }
    i++;
    {
        let k = i
        aa[k] = function(){
            console.log(k)
        }
    }
    ...
}
*/
```

④ 定义常量--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。类似于java中的final关键字。
- 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。
- 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。
- const声明的常量，也与let一样不可重复声明。

```
//案例1：常量不可修改，重复声明
const PI = 3.1415926;
console.log(PI)
//PI = 3.14 //常量不可再次赋值：
// 原因：TypeError: Assignment to constant variable.
//const PI = 3.14 //不可重复声明
// 原因：SyntaxError: Identifier 'PI' has already been declared

//案例2：常量声明只在块级别所用域内有效
{
    const CEO = "首席执行官"
    console.log(CEO)
}
//console.log(CEO) //报错：ReferenceError: CEO is not defined

//案例3：常量也是不可提升，及必须在声明后使用常量
//console.log(CTO) //ReferenceError: Cannot access 'CTO' before initialization
const CTO = "首席技术官"
console.log(CTO)
```

本质

> const实际上保证的，并不是变量的值不得改动，而是变量指向的那个内存地址所保存的数据不得改动。
> 对于简单类型的数据（数值、字符串、布尔值），值就保存在变量指向的那个内存地址，因此等同于常量。
> 但对于复合类型的数据（主要是对象和数组），变量指向的内存地址，保存的只是一个指向实际数据的指针，
> const只能保证这个指针是固定的（即总是指向另一个固定的地址），至于它指向的数据结构是不是可变的，就完全不能控制了。

```
//定义一个人这个常量对象
const person = {name:'zhangsan',age:20};

//尝试修改人对象，是不可以修改的。
//person = {name:"lisi",age:22}
//TypeError: Assignment to constant variable. 分配给常量变量

//但是修改person对象中的一个属性值，可以成功
person.age = 30;
console.log(person); //{name: "zhangsan", age: 30}

//若是连属性都不可以修改的话，可以使用ES5中的Object.freeze()
const p = Object.freeze(person);
p.age = 25; //没有报错，但是修改不了属性值
console.log(person); //{name: "zhangsan", age: 30}
```