let,const

ES6 定义变量

ES5声明变量关键字：

var 和 function << 没有块级作用域，使用自执行函数(闭包)去模拟 >>

弊端:

1) var 只是声明 、 function声明 + 定义都完成了(可以在定义前执行，函数可以在声明前面执行) 函数表达式和函数声明德区别

2) 全局下声明都挂在了window对象上的属性和方法 (污染顶级对象window)

3) 可以重复声明(没有函数的重载) 使用arguments和变量的类型去模拟函数的重载

4) 只有函数作用域和全局作用域，没有会级作用域(封不住变量if,for,while…)

5) 有预解释(本应该先声明后使用) 预解释五点要牢记

ES6声明变量关键字：

优化:

var function let const class export 声明变量

1) 有块级作用域(其实就是let和const的作用)

2) 在同一个作用域，不能重复声明

3) 声明的变量没有挂在window对象上，不污染全局对象

4) 没有预解释这一说，必须先声明后使用

5) 暂时性死区问题

6) for循环里面的父作用域，和里面的子作用域

7) const 定义常量，之后不能改变， 对于基本数据类型不能改值，对于引用数据类不能改变引用地址

8) let 和const 为ES6增加了块级作用域

9) 函数参数不能使用let和const在进行声明了

10) ES6全局变量将逐步与顶层对象的属性脱钩。(不污染顶级变量) [`var`命令和`function`命令声明的全局变量，依旧是顶层对象的属性；另一方面规定，`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]

型不能改变引用地址

11) const声明的变量必须要赋值，否则报错

12) for循环 , for -- in循环使用let替换var

----->>对比下

```
let a = 10;
a = 29;


const b = 20;
b = 30;
```

面试案例:

1. 作用域

```
{
    let a = 1;
    const b = 2;
    var c = 3;
    function d() {}
}
console.log(a, b, c, d );
```

2. 作用域

```
let ary = [];
for (var i=0;i<3;i++) {
    ary[i] = function() {
        console.log(i);
    }
}
ary[2](); // ==>> 3
ary[3](); //==>> 3
```

ES5改写和ES6改写下

ES5 --->>闭包方式(function(){})(); (function() {}())

ES6 ---->>var 改为let 让for循环有块级作用域和子作用域

3.

```
let x = 1;
functon fn() {
    let x = 2;
    if ([]) {
        let x = 3;
    }
    console.log(x);
}
fn();
console.log(x);
```

4. 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。

```
{
  let a =10;
  function fn() {
}
```

```
{
    let a = 10;
    let fn = v =>v;
}
```

5. const问题

1) const: 基本数据类型不可以改变值，引用数据类型不可以改变引用地址

2) z将对象冻结，应该使用`Object.freeze`方法。不能添加属性和方法，但是可以修改

3) 除了将对象本身冻结，对象的属性也应该冻结。下面是一个将对象彻底冻结的函

数。

```
// ===>> 冻结对象，递归调用
let freezeObj = obj => {
   Object.freeze(obj);
   let ary = Object.keys(obj);
   for (let i = 0, len = ary.length; i < len; i++) {
       let cur = obj[ary[i]];
       if (Object.prototype.toString.call(cur === '[Object
object]')) {
           freezeObj(cur);
       }
   }
};
```

6.作用域题;

A1)

```
 let i;  //全局作用域3
    let fn = (...val) => {  //函数作用域2
        console.log(val);
        for (let i = 0; i < 10; i++) {  //for循环里面的父作用域
1
            //let i = i + 1;    //暂时性死区,被绑定在了子作用域里
面，必须先声明后使用(凡是在声明之前就使用这些变量，就会报错)
            //子作用域0      << 0-->> 1--->>2--->>3 作用域链形成
            //console.log(i);
        }
        console.log(i); //-->>undefined  循环里面的变量i循环完毕
后销毁了，这里的i找不到，只好重当前作用域的上一级作用域里面查找i,找到了全局
作用域变量i,全局变量i声明没有赋值，默认值为undefined
    };
```

```
fn(1, 2, 3, 4);
```

A2)

```
//eg2:经典列子:
let fn1 = function () {
  let ary = [];
  for (var i=0;i<10;i++) {
      ary[i] = function () {
          console.log(i);
      };
  }
  return ary;
};
fn1()[1]();
fn1()[4]();


 //解决办法
//ES5解决办法： 闭包
let fn_1 = function () {
  let ary = [];
  for (let i= 0;i<10;i++) {
      ary[i] = (function (num) {    //---->>闭包
          return function () {
              console.log(num)
          }
      })(i);
  }
  return ary;
};
fn_1()[1]();
fn_1()[4]();
```

```
//ES6解决办法：var关键在改为let，多了块级作用域
let fn2 = function () {
    let ary = [];
    for (let i=0;i<10;i++) {
        ary[i] = function () {
            console.log(i);
        };
    }
    return ary;
};
fn2()[1]();
fn2()[4]();
```

简写：

```
//ES6简写
let fn3 = () => {
    let ary = [];
    for (let i=0;i<10;i++) {
        ary[i] = () => console.log(i);
    }
    return ary;
};

fn3()[1]();
fn3()[4]();
```

A3)

```
let x = 10;
    if (x) {
        let x = 'clh';
        console.log(x);
```

```
    }
    console.log(x);
```

对比：

```
 let x = 10;
    if (x) {
         x = 'clh';
        console.log(x);
    }
    console.log(x);
```