JS中对象、数组、原型链都是按引用传递的也叫按址传递（就是指向的同一个内存地址，修改了大家都改变了）

JS中数字、字符串、布尔类型就是按值传递（按值传递是会把值分到另一个内存地址当中和以前的值没有任何关系了）

函数提升的级别比变量高
构造函数里的属性优先级比原型链要低

new 函数是把函数的原型链拿出来

```
this.a = 20;
var test = {
    a:40,
    init:() => {
        console.log(this.a);
        function go(){
            //this.a = 60;
            //console.log(this)
            console.log(this.a);
        }
        go.prototype.a = 50;
        return go;
    }
}
//var p = test.init();
//p();
new(test.init())();//20,50
箭头函数里面并没有 this ，故在箭头函数中的this，指向window
```

```
按引用传递继承常规要求（典型的JS实现面向对象的过程）：
1.拿到父类原型链上的方法
2.不能让构造函数执行2次    （如果用new 函数就会出现父类函数执行了2次）
3.引用的原型链不能是按址引用（所有Object.create就重新创建个对象，建立个新地址）
4.修正子类的constructor

function A() {}
A.prototype.run=function(){
    console.log('run')
}
A.mm = function() {}
var B = function(){
    A.call(this);
}
var __proto = Object.create(A.prototype);
__proto.constructor = B;
B.prototype=__proto;
B.prototype.jump=function(){
    cosole.log('jump');
}
var result = new A();
console.log(result);
var result2 = new B();
console.log(result2);
```

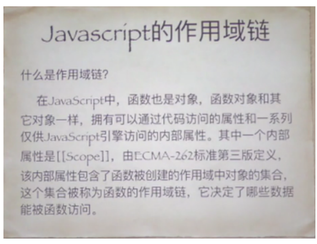
类似功能封装

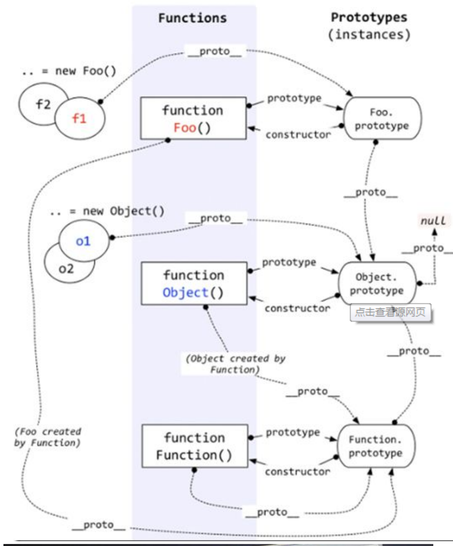
```
function inherits(child,parent){
    var __proptotype = Object.create(parent.prototype);
        __proptotype.constructor = child.prototype.constructor;
        child.prototype = __proptotype;
}
```

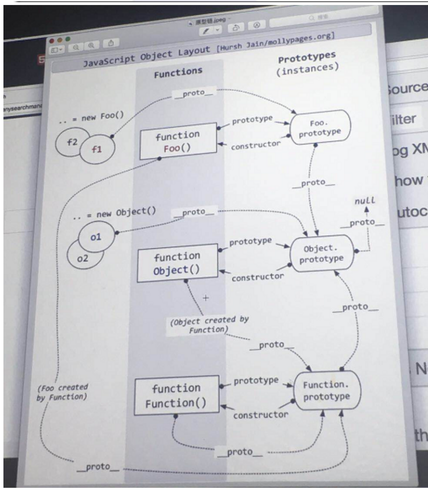
ES6语法实现

```
class B extends A{
    constructor(name, age, language){
        super(name, age); //必须写才能拿到父级的方法
        this.language = language;
    } //构造函数不写就完全是父级的构造函数在书写方法就会重写方法

    xx(){}//书写方法
}
```







视频地址: 精英班预读班 -> ES5核心技术