# 闭包的使用

一个线程一次只能处理一个事情，多个线程可以同时处理多个事情。但是JS是单线程的，在JS中，有三个任务：

1. 渲染任务
2. JS代码执行任务
3. 事物处理任务

JS代码的执行顺序：先执行主任务（代码任务），再执行次要（如setTImeOut中的回调）

for(var i = 0; i < 5; i++){  
 setTimeout(function () {  
 console.log(i); *//打印5次 5* },1)  
}

这里打印5次5的原因是：只能在主任务完成后才能执行回调函数，如果我们需要保证循环，又要保证每次循环都要延迟输出循环序列，那么修正方式如下：

for(var i = 0; i < 5; i++){  
 function foo(j) {  
 return function () {  
 console.log(j)  
 };  
 }  
 var f = foo(i);  
 setTimeout(f,1);  
}

案例：有若干li，点击顺序中的li，弹出所处顺序

使用ES6的let我们很容易解决该问题：

<div>我是第1个div</div>  
<div>我是第2个div</div>  
<div>我是第3个div</div>  
<script>  
 let divs = document.getElementsByTagName('div');  
 for(let i = 0; i < divs.length; i++){  
 divs[i].onclick = function () {  
 alert('我是第' + (i+1) + '个div')  
 }  
 }  
</script>

而当我们使用ES5的var来定义 i，就会出现点击所有的li都会弹出：我是第4个li（i循环到最后的值是3）。

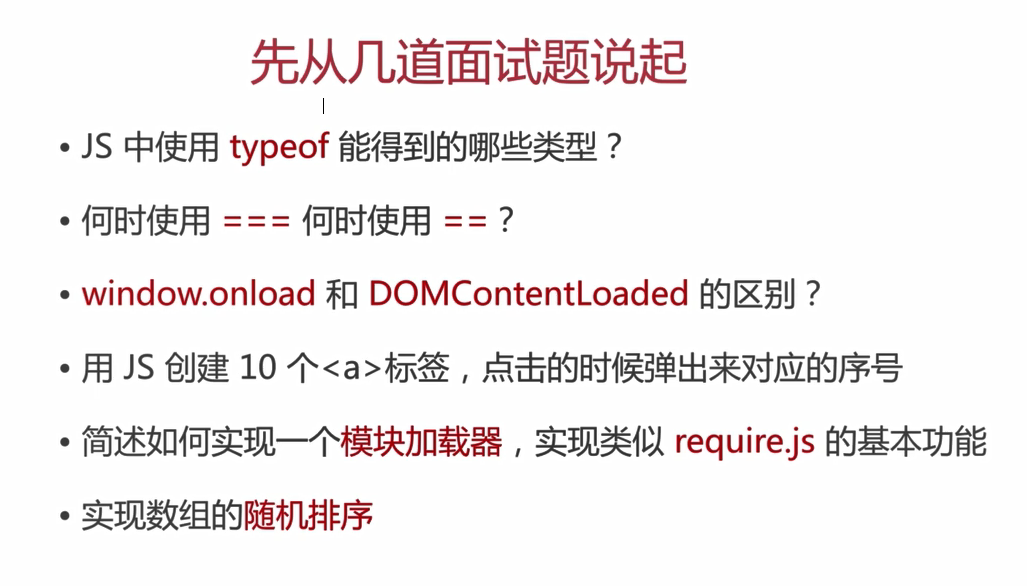
这是因为var的定义是全局的，而let的定义是具备代码块限制的。

在ES5我们使用闭包来处理这个现象：

<div>我是第1个div</div>  
<div>我是第2个div</div>  
<div>我是第3个div</div>  
<script>  
 var divs = document.getElementsByTagName('div');  
 for(var i = 0; i < divs.length; i++){  
 divs[i].onclick = function (j) {  
 return function () {  
 alert('我是第' + (j+1) + '个div');  
 }  
 }(i);  
 }  
</script>

# 作用域

function Foo() {  
 getName = function(){ alert(1); };  
 return this;  
}  
  
function getName(){ alert(5); }  
  
getName = function() { alert(4); };  
  
Foo().getName(); *// ?1  
  
//1 设置值的时候，也是访问变量  
//2. 获取值的时候，是访问变量  
//3.并不是在函数内部写了变量，这个变量就属于这个函数的作用域，  
// 而是必须使用var来声明变量，这个变量才会属于这个作用域*



# 包装对象

基本数据类型如String本身只是一个字符串，并不是对象，却具备可以使用的方法，比如charAt()，这是因为有包装对象String的存在。

String.prototype.charAt();

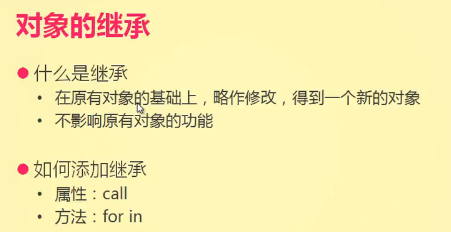
除了null与undefined，基本数据类型都有对应的包装对象。

# 原型链

实例对象与原型之间的链接就叫做原型链。



# 继承



## 1 拷贝继承

*//Person对象*function Person(name) {  
 this.name = name;  
}  
Person.prototype.say = function () {  
 console.log('hi');  
};  
  
*//直接继承*function Student(name,className) {  
 Person.call(this,name); *//继承属性：因为直接使用Person，不能让window调用，必须用call* this.className = className; *//添加新属性*}  
*// //继承方法，但是这样写直接饮用了，如果给子类添加方法，父类也会拥有该方法，这是错误的  
// Student.prototype = Person.prototype;  
  
//需要使用拷贝继承的方法*function extend(obj1,obj2) {  
 for(let attr in obj2){  
 obj1[attr] = obj2[attr];  
 }  
}  
extend(Student.prototype,Person.prototype);  
  
Student.prototype.study = function () { *//添加新方法，* console.log('study');  
};  
  
let s = new Student('lisis','3班级');  
s.say();

## 2原型继承

借助原型来实现继承。