# [JavaScript作用域和闭包](http://www.cnblogs.com/chengzp/p/scopechain.html)

在本文中，笔者将用通俗的语言和简单的代码，介绍以下几种概念：

* 变量提升
* this的使用场景
* 作用域
* 闭包的应用

最后还有一个例题

## **变量提升**

首先我们要知道，js的执行顺序是由上到下的，但这个顺序，并不完全取决于你，因为js中存在变量的声明提升。

这里比较简单，直接上代码

[IMG_256](http://www.cnblogs.com/chengzp/p/javascript:void(0);)

console.log(a) //undefinedvar a = 100

fn('zhangsan')function fn(name){

age = 20

console.log(name, age) //zhangsan 20

var age

}

[IMG_257](http://www.cnblogs.com/chengzp/p/javascript:void(0);)

结果

IMG_258

打印a的时候，a并没有声明，为什么不报错，而是打印undefined。

执行fn的时候fn并没有声明，为什么fn的语句会执行？

这就是变量的声明提升，代码虽然写成这样，但其实执行顺序是这样的。

[IMG_259](http://www.cnblogs.com/chengzp/p/javascript:void(0);)

var a

function fn(name){

age = 20

console.log(name, age)

}

console.log(a)

a = 100

fn('zhangsan')

[IMG_260](http://www.cnblogs.com/chengzp/p/javascript:void(0);)

js会把所有的声明提到前面，然后再顺序执行赋值等其它操作，因为在打印a之前已经存在a这个变量了，只是没有赋值，所以会打印出undefined，为不是报错，fn同理。

这里要注意函数声明和函数表达式的区别。上例中的fn是函数声明。接下来通过代码区分一下。

[IMG_261](http://www.cnblogs.com/chengzp/p/javascript:void(0);)

fn1('abc')function fn1(str){

console.log(str)

}

fn2('def')var fn2 = function(str){

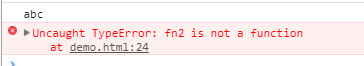
console.log(str)

}

[IMG_262](http://www.cnblogs.com/chengzp/p/javascript:void(0);)

本例中fn1是函数声明，而fn2是函数表达式。函数表达式中的函数体是不会被提升的。

结果



可以看到fn1被提升了，而fn2的函数体并没有被提升。

效果等同于

var fn2

fn2('def')

fn2 = function(str){

console.log(str)

}

这下应该明白为什么报错了吧。变量提升就说这么多，接下来看this

## **this**

this简单理解就是调用函数的那个对象。

要搞懂this，首先要理解一句话：

**this要在执行时才能确认，定义时无法确认。**

接下来还是在代码中解释这句话

[IMG_264](http://www.cnblogs.com/chengzp/p/javascript:void(0);)

var a = {

name: 'A'

fn: function () {

console.log(this.name)

}

}

[IMG_265](http://www.cnblogs.com/chengzp/p/javascript:void(0);)

看这段代码，this指向谁？

现在说指向谁都是不对的，this在定义时是无法确认的，只有执行时才能确认。

继续上面代码，判断以下this的指向。

[IMG_266](http://www.cnblogs.com/chengzp/p/javascript:void(0);)

a.fn() //this === a

a.fn.call({name:'B'}) //this === {name: 'B'}

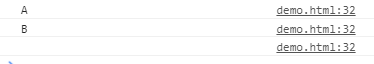
var fn1 = a.fn

fn1() //this === window

[IMG_267](http://www.cnblogs.com/chengzp/p/javascript:void(0);)

代码中已经给出答案了。虽然fn定义在a对象里，但是fn中this的指向并不总是指向a，谁调用fn，this就指向谁。

看一下输出结果



window没有name属性，所以最后一行为空。

**this都有哪种使用场景呢？**

主要由以下4点

1. 作为构造函数执行
2. 作为对象属性执行
3. 作为普通函数执行
4. call apply bind

[IMG_269](http://www.cnblogs.com/chengzp/p/javascript:void(0);)

//作为构造函数执行  
function (name){

this.name = name

}var f = new Foo('zhangsan')

//作为对象属性var obj = {

name: 'A'

printName:function(){

console.log(this.name)

}

}

obj.printName()

//作为普通对象function fn(){

console.log(this) //此时的this指向window}

fn()

[IMG_270](http://www.cnblogs.com/chengzp/p/javascript:void(0);)

前三种通过代码一看就明白了，不用多说，接下说一下call apply 和 bind

这三个的作用都是改变this的指向，call和apply不同的地方就在于传递参数的部分，apply要用数组，还是在代码中看。

[IMG_271](http://www.cnblogs.com/chengzp/p/javascript:void(0);)

function fn1(name,age){

console.log(name,age)

console.log(this)

}

//call和apply用法的不同

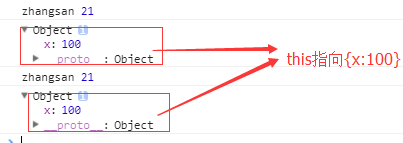
fn1.call({x:100},'zhangsan',21)

fn1.apply({x:100},['zhangsan',21]) //参数放在数组里

[IMG_272](http://www.cnblogs.com/chengzp/p/javascript:void(0);)

call和apply的第一个参数都是this的指向，之后是构造函数的参数。

输出结果



可以看到，两种方式的输出结果是一样的。

再然后是bind，bind和前面两个都不太一样，bind不是在函数执行时调用的，而是在函数声明时。

看代码

[IMG_274](http://www.cnblogs.com/chengzp/p/javascript:void(0);)

var fn2 = function(name,age){

console.log(name,age)

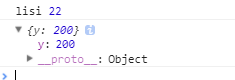
console.log(this)

}.bind({y:200}) //在函数声明时绑定this的指向

fn2('lisi',22)

[IMG_275](http://www.cnblogs.com/chengzp/p/javascript:void(0);)

结果·



一看就明白，不多解释了，但有一点需要注意

只有函数表达式才能使用bind，函数声明是不能用的。

来一个错误的演示

[IMG_277](http://www.cnblogs.com/chengzp/p/javascript:void(0);)

//错误演示 function fn3(name,age){

console.log(name,age)

console.log(this)

}.bind({z:300}) //错误演示

//错误演示

fn3('lisi',23)

[IMG_278](http://www.cnblogs.com/chengzp/p/javascript:void(0);)

 报错喽

IMG_279

## **作用域链**

首先要知道，JavaScript是没有块级作用域的

if(true){

var name = "zhangsan"

}

console.log(name)

正常打印，没有报错。

IMG_280

其次，还要知道全局作用域和函数作用域

[IMG_281](http://www.cnblogs.com/chengzp/p/javascript:void(0);)

var a = 100

function fn(){

var a = 200

console.log('fn',a) //这里的a是200}

console.log('global',a) // 这里的a是100

fn()

[IMG_282](http://www.cnblogs.com/chengzp/p/javascript:void(0);)

在函数里声明的变量，在是不会影响外面的变量的，看结果

IMG_283

这些都了解了后，我们来看作用域链。

说概念我也不知道咋说，直接在代码中看吧。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | var a = 100  function fn(){      var b = 200      console.log(a)  //a为自由变量      console.log(b)  }  fn() |

在fn中是没有a变量的，当在当前作用域下没有定义变量就是自由变量，当前作用域没有的话，就会去他的上级作用域找，这就是作用域链。

IMG_284

还要注意一点，作用域链实在定义时确定的，不是在执行时，不管在哪个作用域下调用某个函数，该函数内的作用域链是不会变得，再来看一段代码

[IMG_285](http://www.cnblogs.com/chengzp/p/javascript:void(0);)

var x = 100function F1(){

var y = 200

function F2(){

var z = 300

console.log(x) //自由变量

console.log(y) //自由变量 console.log(z)

}

F2()

}

F1()

[IMG_286](http://www.cnblogs.com/chengzp/p/javascript:void(0);)

我们看x变量，这是一个自由变量，js引擎在执行到console.log(x)时，会先在F2中寻找x，没找到就去当前作用域的父级作用域F1中找，还找不到就在往上找直到全局作用域。

因为定义时作用域链已经确定了，不管这条链上的函数是在哪里调用的，这条作用域链不会在改了。

通过这两个简单的代码你是不是已经了解了作用域链的工作原理了呢？接下来，介绍闭包

## **闭包**

出于种种原因，我们有时候需要得到函数内的局部变量。但是，前面已经说过了，正常情况下，这是办不到的，只有通过变通方法才能实现。

那就是在函数的内部，再定义一个函数,然后把这个函数返回。

[IMG_287](http://www.cnblogs.com/chengzp/p/javascript:void(0);)

function F1(){

var a = 100

//返回一个函数 (函数作为返回值)

return function (){

console.log(a)

}

}

//f1得到一个函数var f1 = F1()var a = 200

f1()

[IMG_288](http://www.cnblogs.com/chengzp/p/javascript:void(0);)

 打印出IMG_289

在本例中就实现了闭包，简单的说，闭包就是能够读取其他函数内部变量的函数。

下面说为什么打印的是100

看这句 var f1 = F1(); F1这个函数执行的结果是返回一个函数，所以就相当于把F1内的函数付给了f1变量，类似于这样

var f1 = function(){

console.log(a) //这里的a是一个自由变量

}

这里的a是一个自由变量，所以根据作用域链的原理，就应该去上一级作用域去找。之前说过，作用域链在定义时确定，和执行无关，那就去想上找，这个函数定义定义在F1中，所以会在F1中找a这个变量，所以这里会打印的100。

通过这种方式，我们在全局下就读取到了F1函数内部定义的变量，这就是闭包。

#### **闭包主要有以下两种应用场景**

* 函数作为返回值
* 函数作为参数来传递

在上例中我们已经实现了函数作为返回值的闭包的应用，接下来再来一例作为参数传递的

[IMG_290](http://www.cnblogs.com/chengzp/p/javascript:void(0);)

function F1(){

var a = 100

return function (){

console.log(a)

}

}var f1 = F1()

function F2(fn){

var a = 200

fn()

}

F2(f1)

[IMG_291](http://www.cnblogs.com/chengzp/p/javascript:void(0);)

还是输出100

IMG_292

例子比较简单，但能说明问题，在闭包的实际应用中，往往要很复杂，但掌握了闭包的原理后，就不难实现了。

## **例题**

来检验一下你对这部分内容的理解吧

创建10个<a>标签，点击的时候输出对应的序号

看到这道题，最直接的想法或许就i是这样的

[IMG_293](http://www.cnblogs.com/chengzp/p/javascript:void(0);)

//错误代码演示  
var i,afor (i = 0; i < 10; i++) {

a = document.createElement('a')

a.innerHTML = i + '<br />'

a.addEventListener('click',function(e){

e.preventDefault()

console.log(i)

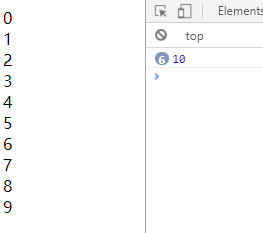
})

document.body.appendChild(a)

}

[IMG_294](http://www.cnblogs.com/chengzp/p/javascript:void(0);)

但这是不对的。



可以看到，我对着左边的标签一顿乱点，结果打印的都是10.为什么呢

因为上例代码都是同步执行的，在页面加载的一瞬间，for循环已经执行完毕，我再去点标签的时候，i的值已经是10了，所以不管我点第几个标签，打印的都会是10。

那怎么解呢?

这么解

[IMG_296](http://www.cnblogs.com/chengzp/p/javascript:void(0);)

for (var i = 0; i < 10; i++) {

(function(i){

a = document.createElement('a')

a.innerHTML = i + '<br />'

a.addEventListener('click',function(e){

e.preventDefault()

console.log(i)

})

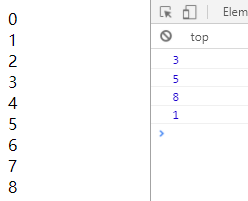
document.body.appendChild(a)

})(i)

}

[IMG_297](http://www.cnblogs.com/chengzp/p/javascript:void(0);)

功能实现。



这道题我就不解释了，如果本文中介绍的内容你已经掌握了话，本例是很容易理解的。

如果你对作用域链和闭包已经理解了，建议继续学习

[JavaScript原型链](http://www.cnblogs.com/chengzp/p/prototype.html" \t "http://www.cnblogs.com/chengzp/p/_blank)

[JavaScript类与继承](http://www.cnblogs.com/chengzp/p/extends.html" \t "http://www.cnblogs.com/chengzp/p/_blank)

最后，如果觉得本文对你有帮助话，点个赞吧^\_^