闭包，作用域链，预编译

2019/04/21 16:34

预编译（发生在函数执行的前一刻，函数开始执行时结束）

四部曲：（优先级递增）

1. 创建AO对象（activation object 【执行期上下文】）全局就是生成GO对象

   window就是GO

   先生成GO再生成AO

2. 找形参和变量声明，将变量和形参名作为AO属性名，值为undefined
3. 将实参值和形参统一 （形参变为实参值） 全局不存在此步骤
4. 把函数声明的名，作为Ao对象的属性名
5. 在函数体里找函数声明，赋值予函数体

（~~原先有值就赋值没有就是整个函数体~~）

eg:

```
function fn(a){
    console.log(a);
    var a = 123
    console.log(a);
    function a () {}    //预编译处理不用再看 函数声明整体提升，
                        //处理过后对a无影响
    console.log(a);
    var b = function () {}
    console.log(b);
    function d () {}
}
fn(1);
//在编译前一刻发生预编译！！
//生成AO对象
AO{
    a : function a () {}
    b : undefinded,
    d : function d () {}
}
```

结果：

```
f a() {}
123
123
```

f () {}

```
function test(a,b){
    console.log(a);
    c = 0;
    var c;
    a = 3;
    b = 2;
    console.log(b);
    function b () {}
    function d () {}
    console.log(b);
}
test(1);
AO{
    a : 1
    b : f b () {}
    c : undefined
    d : f d () {}

}
```

结果：

1

2

2

```
function test (){
            var a=b=3
             = console.log(b)
            console.log(a)
       }
       test()
       console.log(b)
 console.log(a)
```

在这里a 在函数内部声明不可被外部使用，而b没被声明属于全局变量，函数内外都可使用

var a=b=3的顺序是:先b赋值3，然后声明a,最后b的值赋给a

```
GO{
    a : undefined
}
function test () {
    console.log(b);
    if (a) {
        var b = 100;
    }
    c = 234;
```

```
    console.log(c);
}
var a;
test();
AO{
    b : undefined     //不用看if执不执行
}
a = 10;
console.log(c);
```

结果：

```
undefined
234
234
```

作用域链：

```
function a() {
    function b() {
        function c() {

        }
        c();
    }
    b();
}
a();
```

```
a defined a.[[scope]] -- > 0 : GO
a doing a.[[scope]] -- > 0 : AO
                       1 : GO
b defined b.[[scope]] -- > 0 : aAO
                       0 : GO
b doing b.[[scope]] -- > 0 : bAO
                       1 : aAO
                       2 : GO
c defined c.[[scope]] -- > 0 : bAO
                       1 : aAO
                       2 : GO
c doing c.[[scope]] -- > 0 : cAO
                       1 : bAO
                       2 : aAO
                       3 : GO
```

作用域链的形成方式

闭包

基本形式

```
function f1(){
    var a = 3;
    function f2(){
        console.log(a);
    }
    return f2;
   }
    var test = f1();
    test();                 //返回值是函数所以要把函数赋给一个变量使用
    //并且此变量还要用函数方式执行
```

极简写法

```
function f1(){
    var a = 3;
    return function(){
        console.log(a);
    }
}
f1()();
```

个人理解

本来函数只能访问外部变量，而函数内部变量却不可被拿出访问，

闭包就是将函数内部变量拿出外部来进行访问。

原理

```
a defined a.[[scope]] -- > 0 : GO
a doing a.[[scope]] -- > 0 : AO
                       1 : GO

b defined b.[[scope]] -- > 0 : aAO
                        0 : GO
//b生成瞬间，a执行结束      a的AO已经消失可是b还拿着a的AO
b doing b.[[scope]] -- > 0 : bAO
                       1 : aAO
                       2 : GO
```

累加器原理

```
function f1(){
    var a = 3;
    return function(){
        a++;
        console.log(a);
    }
}
var add = f1();
add();
add();
```

//这里f1赋值给add代表f1执行;
//之后执行add()就是执行里面的function
//AO一直被拿出来

闭包会导致原有作用域链不释放导致内存泄漏

内存泄露其实就是占用内存太多，导致内存减少像泄露了一样

闭包的作用

实现公有变量

eg：函数累加器

可以做缓存（存储结构）

eg：eater

可以实现封装，属性私有化。

eg：Person();

模块化开发，防止污染全局变量

立即执行函数

基本形式

```
var num = (function (形参1，形参2，……){

    //有没有名字都行，反正也就执行一次，而且执行到这里就执行
    return d
}(实参1，实参2，……))
console.log(num);
```

特点：只执行一次就销毁，以免占用内存

函数表达式能加()执行函数声明后加()不能执行

表达式：是由运算元和运算符(可选)构成，并产生运算结果的语法结构。

```
var fun = function(){
}()
+ function (){
}(); // + - ! 都行* /不行 && ||也行，但是前面要有东西
```

且执行完后都被销毁

```
function test(a,b,c,d){
    console.log(a+b+c+d);
}(1,2,3,4);
//理论上非表达式不能直接执行，()内没有参数必然报错，
//但是此括号内参数，系统会认为是逗号表达式，因此不报错
```

```
function test(){
    var arr = [];
    for (var i = 0; i<10 ;i++)
    {
        arr[i] = function(){
            document.write(i + " ");
        }
    }
    return arr;
}
var myArr = test();
for (var j = 0; j<10; j++){
    myArr[j]();
}
```

结果为：

10 10 10 10 10 10 10 10 10 10

赋值到arr中是并没有执行，而是test函数执行完毕里面的function才实行此时test执行完i = 10

闭包实现循环输出0~9

```
function test(){
    var arr = [];
    for (var i = 0; i < 10; i++)
     {
        (function (j){
            arr[j] = function () {
                document.write(j + " ");
            }
        }(i));
     }
    return arr;
  }
  var myArr = test();

    for(var j = 0; j<10; j++){
      myArr[j]();
    }
```

解析

每个内部function都用的是立即执行函数的AO每个立即执行函数的j都不一样