深入理解闭包、作用域

使用词法环境跟踪变量的作用域

```
let x="edison"
// 当执行 console.log(x) 时，会对x进行词法环境的查询
// 通常，作用域指的就是词法环境
console.log(x)
```

代码嵌套

```
function x(){
  let a=1
  function y(){
    let b=2
    function z(){
      //通过代码嵌套可以获得 a,b变量
      //那么JS引擎时如何跟踪这些变量的呢
      //即如何去判断 a b 在z函数中可用，这就涉及到了词法环境的作用
      let z=a+b
    }
  }
}
```

代码嵌套与词法环境

- 无论何时去调用函数，都会被推入执行栈中
- 不如执行到 z() 时，x、y、z都已经入栈
- 同时，还会创建一个词法环境。并且存储在 [[inviroment]] 的内部属性上，即我们无法去直接访问或者操作
- 所以，当 z() 执行时，会先检查当前执行上下文，即 z() 中。然后去找他的词法环境。当找不到时，会查找他的外部环境，即 y() z()。。。。。。

执行上下文

- 在JS中，通过函数1调用函数2，函数2调用3，以此类推。那么，JS引擎时如何跟踪函数的执行的呢？并且还能最后回到初始函数？
- 所以，执行上下文也可以分为全局执行上下文；函数执行上下文
- 首先，JS中代码有两种类型：全局和函数。
  - 同时需要注意，全局上下文是在JS程序开始执行时就创建了；而函数执行上下文是在执行函数时创建，并且函数执行完就被销毁
- 具体执行的效果自己想象栈数据结构
  - 1.一开始，全局执行上下文就入栈
  - 2.当执行到一个函数时，全局停止，然后函数执行上下文入栈
  - 3.当函数执行完后，其上下文被销毁，然后重新继续执行全局上下文。
  - 4.然后反复这个过程

使用闭包

- 原生JS不支持实现私有变量
- 实现私有变量

```
function Fn(){
  let private=0
  this.getPri=function(){
    return private
  }
  this.addPri=function(){
    return ++private
  }
}
let fn=new Fn()
console.log(fn.private) //undefined  可见实现了私有变量
console.log(fn.getPri()); //0
console.log(fn.addPri()); //1 说明这个私有变量是可以通过内部函数调用的
```

- 回调函数
  - 闭包内的函数不仅可以在创建时去访问闭包内的变量，而且当这些函数执行时，还可以去更新这些变量的值。
  - 闭包并不是在创建时的那一刻的状态，而是一个真实的状态封装，只要这个闭包还存在，就可以对变量进行修改
  - 就好比上面的代码。private=0只是这个状态的封装，而里面的addPri是可以访问并且修改这个变量的
- 要牢记，闭包是和作用域强相关的

理解闭包

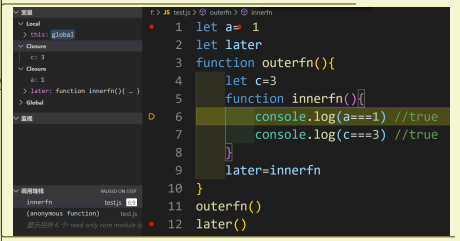
- 闭包允许函数访问并且操作函数外部的变量
- 只需要这些变量或者函数在这个声明函数的作用域内即可

直接看一个例子

1.为什么第一个打印为true？

- 第一个好理解，变量a是一个全局变量，在任何地方都可见
- 第二个看似是false，因为执行later（）时，因为later被innerfn赋值，所以并没有经过变量c，看起来好像这个c此时时不存在的，变量c因该存在于outerfn执行时。
- 可是事实并非如此
- 那么这是为什么呢？其实在声明outerfn时，就已经创建了一个闭包，这个闭包不仅包含它本身，还包括了outerfn所在的作用域内所有的变量，虽然最后在执行innerfn时，这个作用域已经消失了，但是所有的变量都通过闭包保存了下来。
- innerfn在声明时也是如此，他也会将变量c在闭包中保存下来。 这点可以在调试工具中看到

2.为什么第二个打印为true？



```
let a= 1
let later
function outerfn(){
    let c=3
    function innerfn(){
        console.log(a===1) //true
        console.log(c===3) //true
    }
    later=innerfn
}
outerfn()
later()
```

- closure单词意思就是闭包。可以看出当执行later时，有两个闭包，第一个闭包就是执行later创建的，第二个就是执行outerfn时创建的

注意点

- 在使用闭包保存的这些信息时，会直接影响性能
- 要谨记，每一个通过闭包访问变量的函数都具有一个作用域链，作用域链包含闭包的全部信息
- 不能过度使用闭包，因为如上所说，这些保存的信息都会存储在内存中，直到JS引擎确保不在使用它们（即安全地行进垃圾回收）或当页面卸载时，才会去清理他们