闭包是什么?

来源	描述
JavaScript 高级程序设计	闭包指的是那些引用了**另一个函数作用域**中变量的函数，通常是在**嵌套函数**中实现的。
JavaScript 权威指南	函数对象可以通过**作用域链相互关联**起来，函数体内部的变量都可以保存在函数作用域内，这种特性在计算机科学文献中称为闭包。从技术角度讲，所有是Javascript函数都是闭包
Google Search	In JavaScript, a closure is **a function that references variables in the outer scope from its inner scope**. The closure preserves the outer scope inside its inner scope.（在 JavaScript 中，闭包是一个函数，它从**内部作用域引用外部作用域**中的变量。闭包将**外部作用域保留在其内部作用域内**。）
MDN	一个函数和对其周围状态（lexical environment，词法环境）的引用捆绑在一起（或者说函数被引用包围），这样的组合就是闭包（closure）。也就是说，闭包让你可以在一个内层函数中访问到其外层函数的作用域
百度百科	闭包就是能够读取其他函数内部变量的函数。例如在javascript中，只有函数内部的子函数才能读取局部变量，所以闭包可以理解成“定义在一个函数内部的函数“。在本质上，闭包是将函数内部和函数外部连接起来的桥梁。
w3schools	A closure is a function having access to the parent scope, even after the parent function has closed.https://www.w3schools.com/js/js_function_closures.asp（闭包是一个具有对父范围的访问权限的函数，即使在父函数关闭之后也是如此。）
其他	闭包是捆绑在一起(封闭)的函数与其周围状态(词法环境)的引用的组合。

如何产生闭包?

在此之前，先了解一下变量环境，词法环境，函数作用域和作用域链。

变量环境

装着引起变量提升的变量的容器。

- 什么是变量提升?

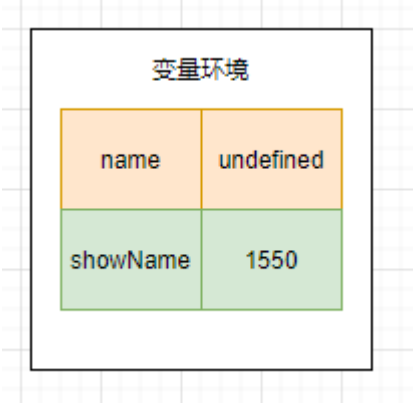
```
showName();

function showName(){
  console.log(name);
}

var name = "32323";
const age = 23;
```

以上代码为什么不报错？因为JavaScript 引擎把变量的声明部分和函数的声明部分提升到代码开头，并且把变量赋值为undefined。以上代码相当于

```
/** 声明部分提升了 */
var name = undefined;
function showName(){
  console.log(name);
}

showName();
name = "32323";
const age = 23;
```

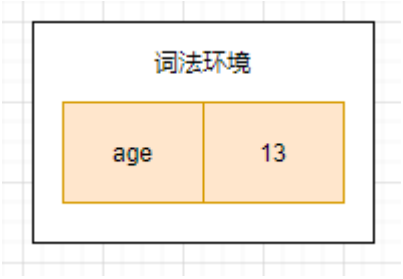
这就是变量提升。

- 因此，可以把变量环境看做一个容器：



词法环境

装着let和const声明的变量（不会引起变量提升的变量）的容器。



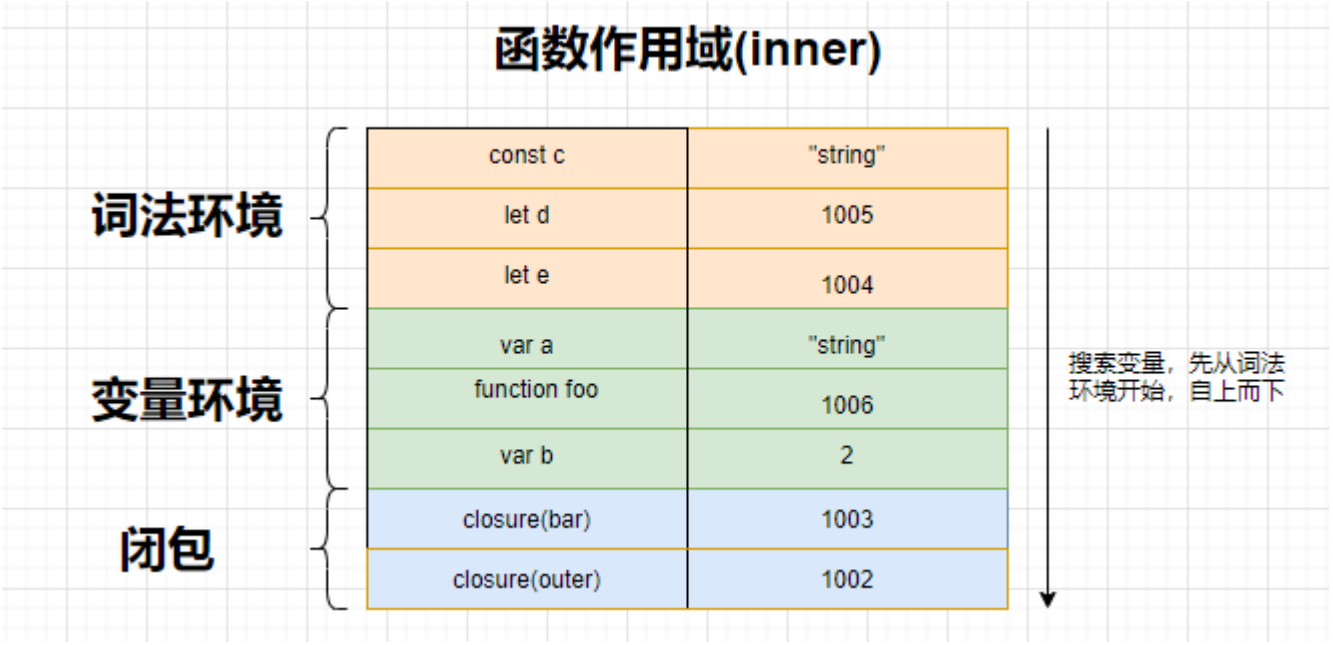
函数作用域

函数作用域由**变量环境**，**词法环境**，**闭包**组成。

```
 function outer(){
     const hello = "hello";

     return function bar(){
     const world = "world";

       return function inner(){
         var a = "string"
         function foo(){
```

```
            console.log("foo");
          }
          const c = "string";
          let d = {};
          let e = function(){
            console.log("e")
          };
          var b = 2;
          debugger;
          console.log("inner:",hello,world);
        }
      }
}

outer()()();
```

执行到**示例源码**的 `debugger` 处的 `inner` 函数作用域示例图：

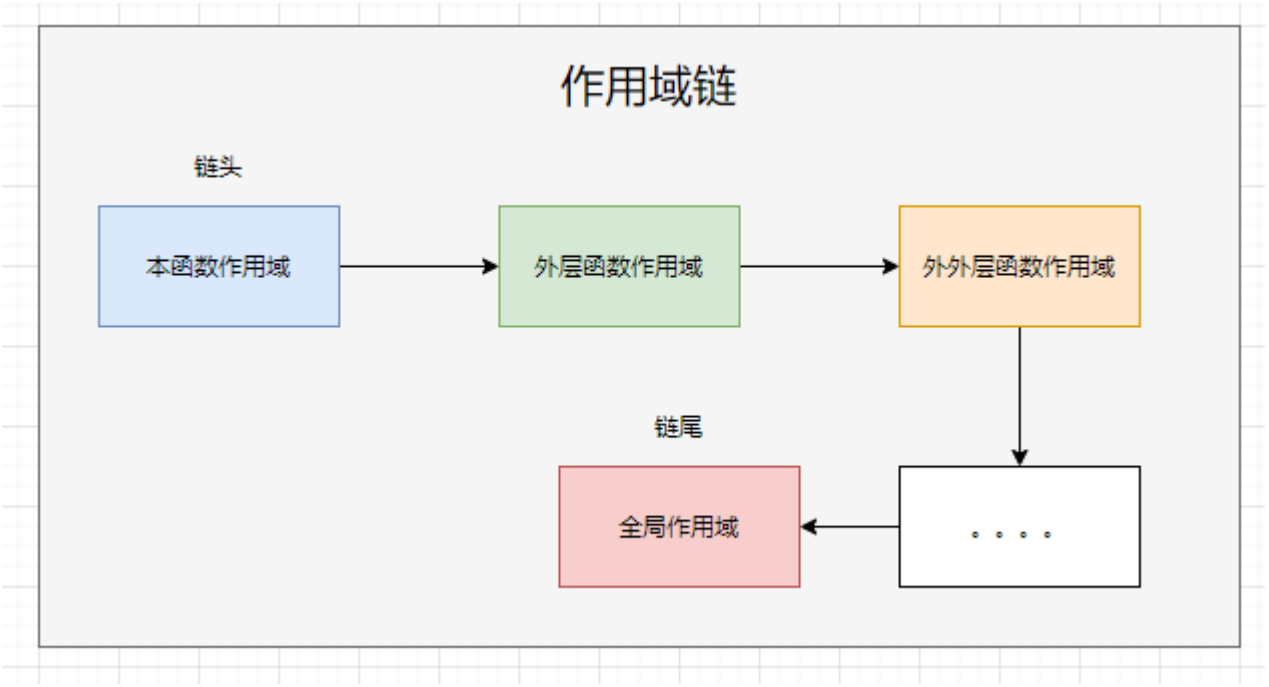


因此：

- 闭包是编译时就产生的。
- 闭包里放着**使用到的**外层函数作用域的变量
- 函数作用域决定了变量的使用范围。
- JavaScript引擎搜索变量是先从**词法环境**开始搜索，接着是**变量环境**，然后是**闭包**，顺着**作用域链**查找，直到**全局作用域**。

作用域链

- 每个函数执行上下文被压入调用栈时，会初始化函数的作用域链。链头是本**函数作用域**，紧接着是**外层函数作用域**，直到**全局作用域**。
- 作用域链是函数上下文内获取变量的依据，变量的取值得顺着作用域来。



回到最初的问题，闭包是如何产生的?

严格来说，闭包需要同时满足2个条件：

- 【1】内层函数访问外层函数作用域的变量
- 【2】内层函数在外层函数作用域外被调用

【1】内层函数访问外层函数作用域的变量

```
function outerFunction () {
  let outerVariable = 1;

  function innerFunction(){
    return outerVariable + 1;
  }
}
```

【2】内层函数在外层函数作用域外被调用

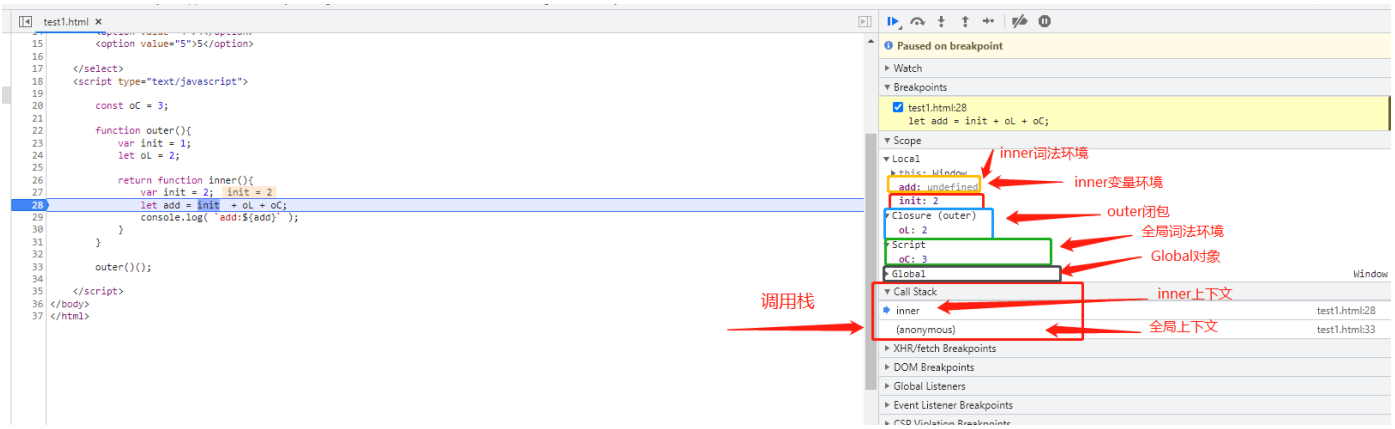
```
function outerFunction () {
  let outerVariable = 1;

  return function innerFunction(){
    return outerVariable + 1;
  }
}

let func =  outerFunction();
func();
```

一个闭包例子

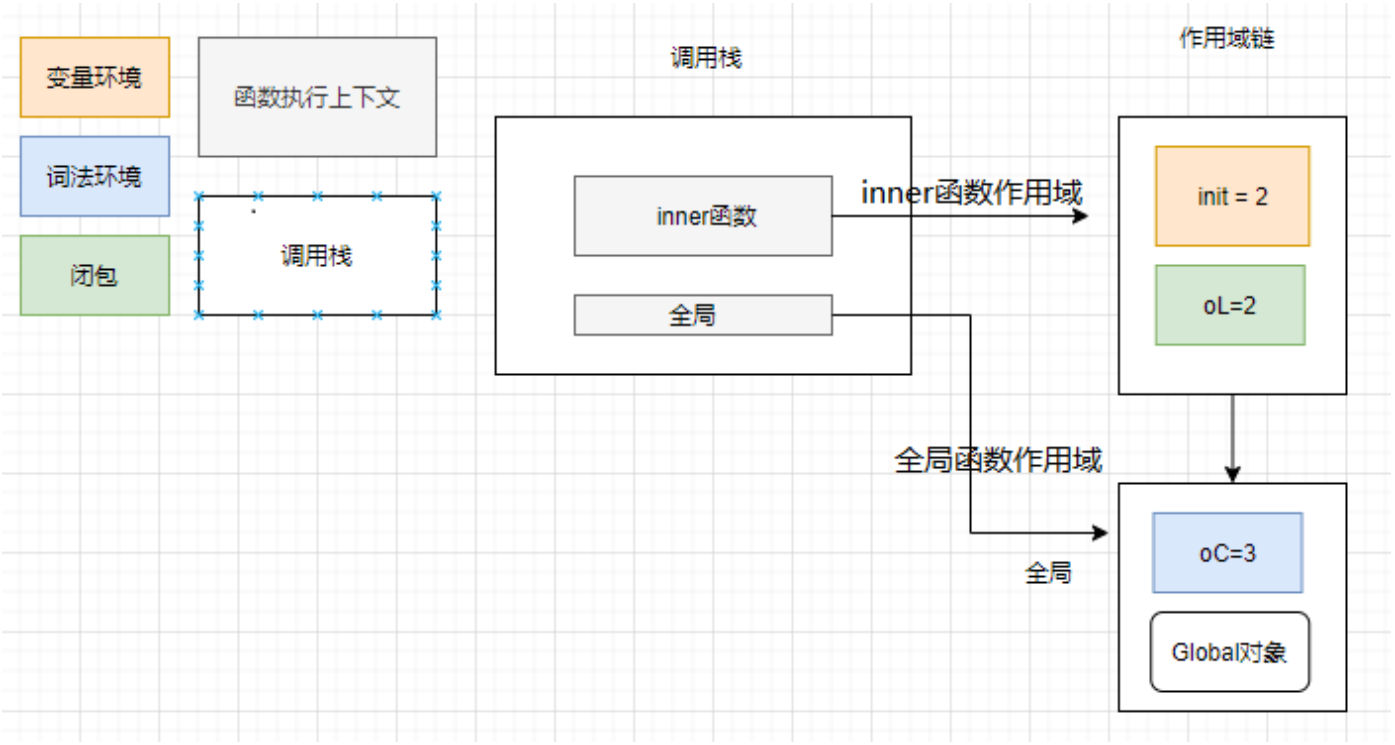
```
const oC = 3;
function outer(){
```

```
    var init = 1;
    let oL = 2;

    return function inner(){
      var init = 2;
      debugger;
      let add = init  + oL + oC;
      console.log( `add:${add}`);
    }
}

outer()();
```

- 执行到 `let add = init + oL + oC;` 时，调试如下：



- 画出作用域链：



一个典型的闭包

```
function counter (i) {
  let count = i;

```

```
  function add () {
    return ++count;
  }

  return add;
}

const counter1 = counter(1);

console.log(counter1()); // 2
console.log(counter1()); //3
console.log(counter1()); //4
```

其他

以上分析，如若有误，欢迎评论指正😘

posted @ 2022-08-06 17:30 胡姐姐 阅读(34) 评论(0)