JavaScript进阶

---JS作用域及执行上下文

河北师范大学软件学院
Software College of Hebei Normal University

内容提纲

- **JS作用域及其特点**
- **JS执行上下文与调用栈（Call Stack）**
- **作用域链与执行上下文**

JS作用域及其特点

- 什么是作用域
  - 作用域就是变量与函数的可访问范围（变量生效的区域范围，即在何处可以被访问到）
  - 作用域控制着变量与函数的可见性和生命周期，它也是根据名称查找变量的一套规则

```
var a = 10,                    全局作用域
    b = 20;
function fn(){                 fn作用域
    var a = 100,
        c = 200;
    function bar(){            bar作用域
        var a = 500,
            d = 600;
    }
}
```

左侧实例（嵌套作用域）中：
变量d只能在bar作用域中被访问到，
变量c只能在fn和bar作用域中被访问到

在bar中访问a时为500（覆盖性）
在bar中访问c时为200（链式关系）

参见实例demo01 作用域实例

JS作用域及其特点

- JS作用域特点（词法作用域）
  - JS采用的是词法作用域（静态性），这种静态结构决定了一个变量的作用域
  - 词法作用域不会被函数从哪里调用等因素影响，与调用形式无关（体现了静态性）

```
> var name = 'Jack';
  function echo() {
      console.log(name);
  }
  echo();//输出Jack

  Jack
```

```
> var name = 'Jack';
  function echo() {
      console.log(name);
  }
  function env() {
      var name = "Bill";
      echo();
  }
  env();//Bill or Jack
```


参见实例demo02 静态词法作用域实例

JS作用域及其特点

- JS作用域特点（静态词法作用域补充部分）
  - 通过new Function创建的函数对象不一定遵从静态词法作用域
  - 对比下边两个例子（通过不同形式定义的函数对象，访问到的scope的区别）

```
var scope = "global";
function checkScope() {
    var scope = "local";
    return function(){
        return scope;
    };
}
console.log(checkScope()());
```

```
local
```

```
var scope = "global";
function checkScope() {
    var scope = "local";
    return new Function("return scope;");
}
console.log(checkScope()());
```

```
global
```

JS作用域及其特点（关于块作用域）

- 大多数语言都有块级作用域
  - 变量“存活”在最近的代码块中，比如Java中

```
public static void main(String[] args){
    {//block start
        int foo = 4;
    }//block end
    System.out.println(foo);//报错，无法访问到foo
}
```

- JS（ES5）采用的是函数级作用域，没有块作用域

```
{//block start
    var foo = 4;
}//block end
console.log(foo);//正常输出4
```

```
4
```

JS作用域特点（关于块作用域）

- 无块作用域的问题（变量污染、变量共享问题）

```
//变量污染问题,尤其是异步执行的情况下
var userId = 123;
document.onclick = function () {
    console.log(userId);
};
//...
```

```
//...
//一长串代码后，忘记了上边定义的userID
var a=2,b=3;
if(a<b){
    var userId = 234;//重复定义，变量污染
}
```

- 解决方案IIFE（更多内容参见IIFE部分）

```
//通过IIFE引入一个新的作用域来限制变量的作用域
(function(){
    var a=2,b=3;
    if(a<b){
        var userId = 234;
    }
}());
```


参见实例demo03 变量污染及解决办法

内容提纲

- JS作用域及其特点
- JS执行上下文与调用栈（Call Stack）
- 作用域链与执行上下文

JS执行上下文和调用栈

- 执行上下文（context，举例生活中的上下文环境）
  - 执行上下文指代码执行时的上下文环境（包括局部变量、相关的函数、相关自由变量等）
  - JS运行时会产生多个执行上下文，处于活动状态的执行上下文环境只有一个

```
1  console.log("全局上下文-start");
2  var x = 1;
3  function foo(){
4      console.log("foo上下文-start");
5      var y = 2;
6      function bar(){
7          console.log("bar上下文");
8          var z = 3;   z = 3
9          console.log(x+y+z);
10         console.log("bar上下文");
11     }
12     bar();
13     console.log("foo上下文-end");
14 }
15 foo();
16 console.log("全局上下文-end");
```

Paused on breakpoint

▶ Watch

▼ Call Stack

bar	demo.js:10
foo	demo.js:12
(anonymous)	demo.js:15

▼ Scope

▼ Local

▶ this: Window

z: 3

▶ Closure (foo)

河北师范大学软件学院
Software College of Hebei Normal University

断点：实例中执行到断点时，就有一个当前断点所对应的执行上下文（对应当前执行环境）

JS执行上下文和调用栈

- 理解执行上下文
  - 小明回家
  - 在家-做作业中 1 ...
  - 在家-做作业中 2 ... 发现笔没油了
  - 去文具店
  - 在文具店-买文具中 ...
  - 在文具店-买文具中 ... 发现没带钱
  - 去银行
  - 在银行-取钱 ... 返回文具店
  - 在文具店-买好文具 ... 返回家
  - 在家-继续做作业...

```
console.log("小明回家");
var xx = ["书桌","书包","铅笔盒"];//小明家中
console.log("在家-做作业中 ");
function goToStore(){
    var yy = ["文具店老板","出售的文具"];//文具商店中
    console.log("在文具店-买文具中 ");
    console.log("在文具店-买文具中  发现没带钱");
    goToBank();
    console.log("在文具店-买好文具  返回家");
}
function goToBank(){
    var zz = ["银行职员","柜员机"];//银行中
    console.log("在银行-取钱 返回文具店");
}
console.log("在家-做作业中  发现笔没油了");
goToStore();//笔没油了，去商店买笔
console.log("在家-继续做作业");
```


参见实例demo04_1 理解执行上下文的例子

JS执行上下文和调用栈

- 调用栈（Call Stack）
  - 代码执行时JS引擎会以栈的方式来处理和追踪函数调用（函数调用栈 Call Stack）
  - 栈底对应的是全局上下文环境，而栈顶对应的是当前正在执行的上下文环境

```
1  console.log("全局上下文-start");
2  var x = 1;
3  function foo(){
4      console.log("foo上下文-start");
5      var y = 2;
6      function bar(){
7          console.log("bar上下文");
8          var z = 3;   z = 3
9          console.log(x+y+z);
10         console.log("bar上下文");
```

Paused on breakpoint

▶ Watch

▼ Call Stack

bar	demo.js:10
foo	demo.js:12
(anonymous)	demo.js:15

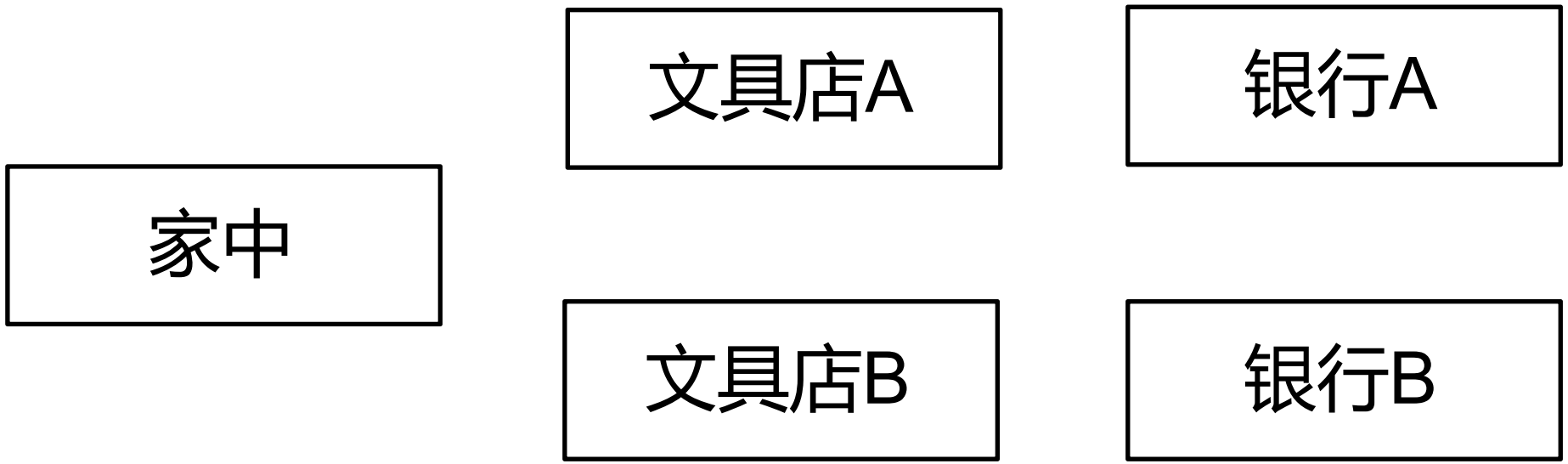
参见实例demo04_2 调用栈实例

内容提纲

- JS作用域及其特点
- JS执行上下文与调用栈（Call Stack）
- **作用域链与执行上下文**

JS执行上下文和调用栈

- 理解代码执行时形成的作用域链（继续小明的例子）
  - 如果有多个文具店和多个银行，那么执行就有多种可能，形成不同的链式关系
  - 依然要遵从静态词法作用域（在A文具店，应该有A店老板，而不应有B店老板）

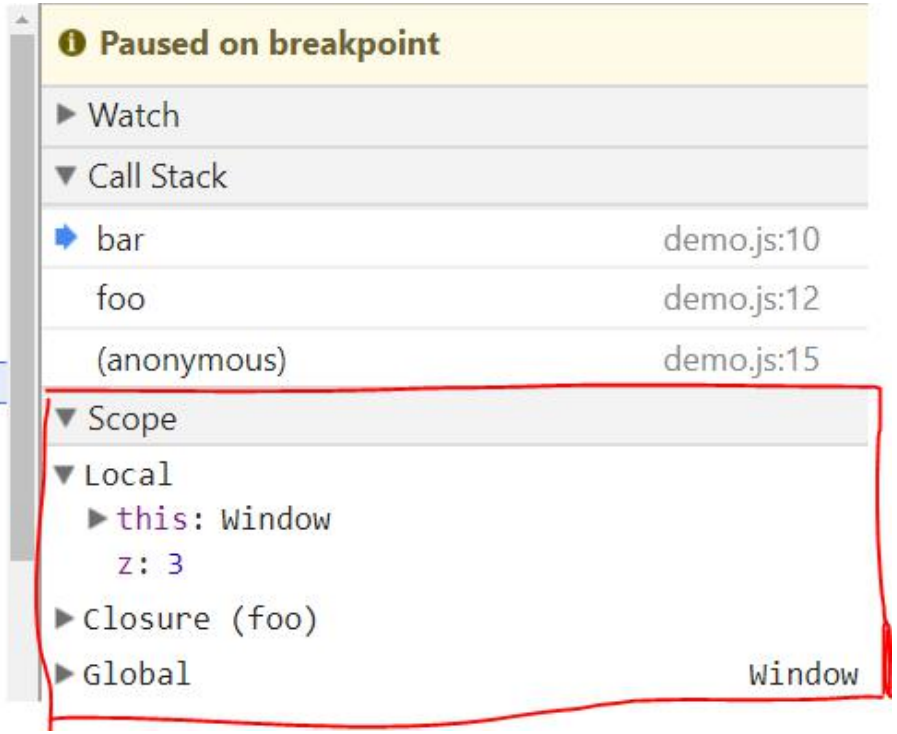


作用域链与执行上下文

•作用域链与执行上下文

- 执行时，当前执行上下文，对应一个作用域链环境来管理和解析变量和函数（动态性）
- 变量查找按照由内到外的顺序（遵循词法作用域），直到完成查找，若未查询到则报错
- 当函数执行结束，运行期上下文被销毁，此作用域链环境也随之被释放

```
1  console.log("全局上下文-start");
2  var x = 1;
3  function foo(){
4      console.log("foo上下文-start");
5      var y = 2;
6      function bar(){
7          console.log("bar上下文");
8          var z = 3;   z = 3
9          console.log(x+y+z);
10         console.log("bar上下文");
11     }
12     bar();
13     console.log("foo上下文-end");
14 }
15 foo();
16 console.log("全局上下文-end");
```



参见实例demo05 作用域链

总结

- JS作用域及其特点
- JS执行上下文与调用栈（call stack）
- 作用域链与执行上下文

Thank You!

补充：

- 环境：变量的管理
  - 当程序运行到变量所在的作用域时，变量被创建，此时需要一个存储的空间
  - JS中提供存储空间的数据结构被称为环境，每个函数都有自己的执行环境
  - 每个执行环境都有一个与之关联的变量对象，环境中所有变量和函数都保存在此对象中
  - Web浏览器中，全局执行环境为window对象
- 作用域链（在 ECMA262 中的解释，涉及到内部属性）
  - 任何执行上下文时刻的作用域，都是由作用域链 (scope chain) 来实现。 在一个函数被定义的时候，会将它定义时候的 scope chain 链接到这个函数对象的[[scope]]属性。 在一个函数对象被调用的时候，会创建一个活动对象 (也就是一个对象，然后对于每一个函数的形参，都命名为该活动对象的命名属性，然后将这个活动对象做为此时的作用域链 (scope chain) 最前端， 并将这个函数对象的 [[scope]] 加入到 scope chain 中