执行上下文

2020年03月09日, 星期一 09:13

1.JavaScript 是一种具有函数优先的轻量级，**解释型或即时编译型**的编程语言

- **编译型语言（C语言）**写的程序执行之前，需要一个专门的编译过程，把所有程序编译成为机器语言的文件
- **解释型语言（JS语言）**在运行程序的时候才编译，每执行一段代码就要翻译一段代码
- 两种方式只是编译的**时机不同**

2.JS运行环境

JavaScript 代码的执行过程

- JavaScript 引擎是**一段一段**地运行代码的
- JavaScript 代码执行时，会为当前代码创建相应的运行环境

JavaScript 运行环境

- 全局环境：代码运行起来后会进入全局环境
- 函数环境：当函数被调用执行时，会进入当前函数中执行代码
- eval 环境：不建议使用，不做介绍

3.执行上下文（=运行环境）

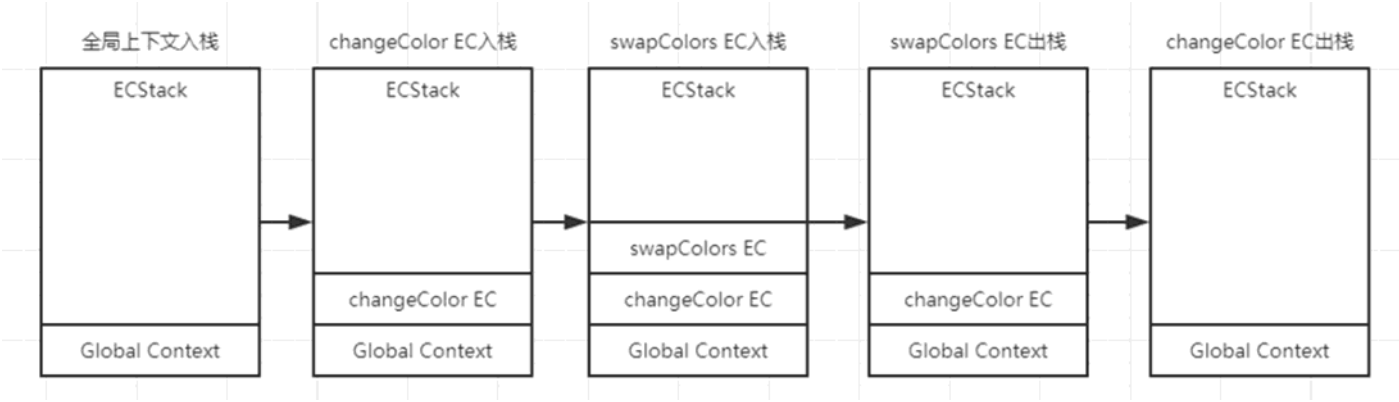
- 可以理解为当前代码的运行环境
- 作用是用来保存当前代码运行时所需要的数据
- 在全局环境、函数环境中会创建执行上下文

执行上下文栈

- 执行上下文栈**按照函数的调用顺序**来管理执行上下文
- **栈底**永远是全局上下文，**栈顶**是当前正在执行的函数
- 特点：**先进后出**

4.

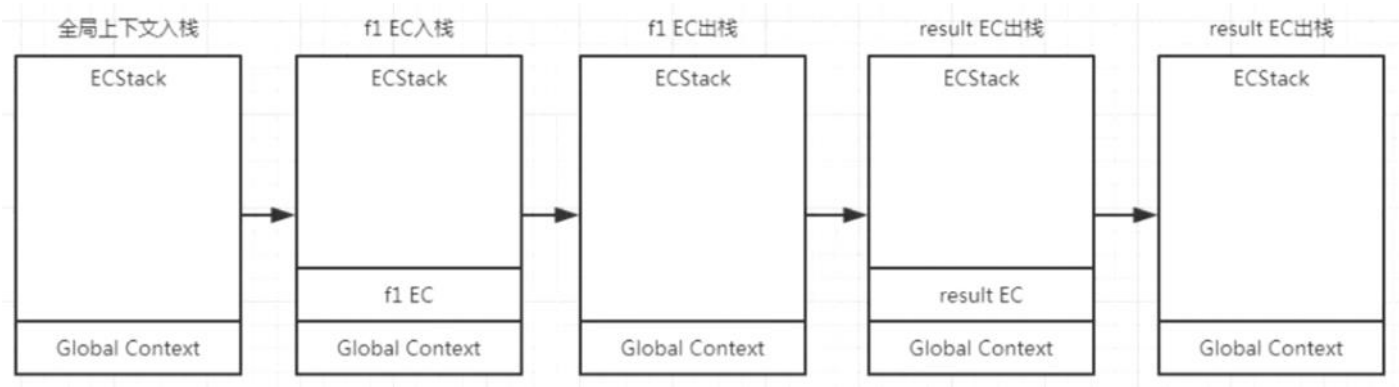
```
var color = 'blue';

function changeColor() {
    var anotherColor = 'red';

    function swapColors() {
        var tempColor = anotherColor;
        anotherColor = color;
        color = tempColor;
    }
    swapColors();
}
changeColor();
```



思考：Global Context 什么时候出栈或者销毁？ 关闭浏览器（退出程序）时才出栈或销毁

5.

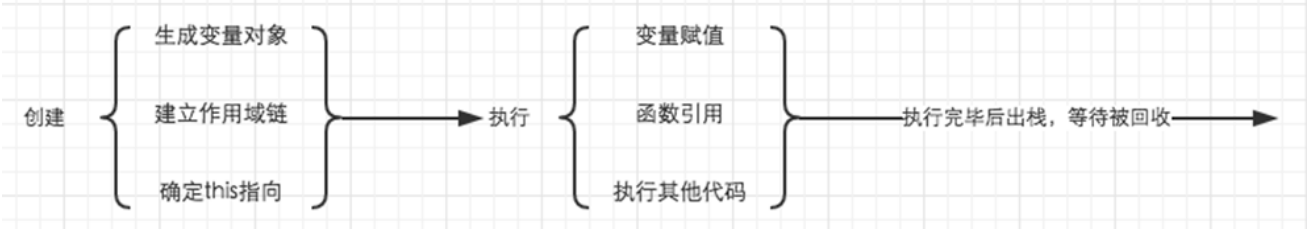
```
function f1() {
    var n = 999;

    function f2() {
        console.log(n++);
    }
    return f2;
}
var result = f1();
result();
```



注意：函数执行时才会创建执行上下文

6.生命周期

- 当一个函数调用时，一个**新的**执行上下文就会被创建。
- 执行上下文的生命周期
  - 创建阶段
  - 执行阶段
  - 等待回收阶段



7.

```
function test() {
    console.log(a);
    console.log(foo());

    var a = 1;
    function foo() {
        return 2;
    }
}
test();
```

每个执行上下文都有一个与之关联的**变量对象**（variable object）和一个**作用域链**（scope chain）

```
testEC = {
    VO: {},          //变量对象
    scopeChain: [], //作用域链
    this: {}         //this指向
}
```

8.return语句

return 语句的作用

- 返回值
- 终止函数的执行（销毁当前执行上下文，弹出执行上下文栈）

```
function fn() {
    return;
    console.log('hello world');
}
fn();
```

//不会输出 ‘hello world’

9.