---

link: null
title: 珠峰架构师成长计划
description: null
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=98 sentences=303, words=1408

---

## 1.闭包 #

### 1.1 定义 #

- Closures (https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Closures)
- 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
- 在 JavaScript 中，每当创建一个函数，闭包就会在函数创建的同时被创建出来

### 1.2 规范 #

- https://262.ecma-international.org/5.1 (https://262.ecma-international.org/5.1/)
- http://es5.github.io (http://es5.github.io)
- ECMAScript5.1_CN (https://static.zhufengpeixun.com/ECMAScript51_gui_fan_zhong_wen_ban_1639732357672.pdf)
- https://262.ecma-international.org/6.0 (https://262.ecma-international.org/6.0/#sec-executable-code-and-execution-contexts))
- Ecma-262-6 (https://static.zhufengpeixun.com/Ecma2626_1639732488710.pdf)

## 2.可执行代码与执行环境 #

### 2.1 main.js #

- 可执行代码与执行环境 (https://www.processon.com/diagraming/61bcc05c7d9c087834ef1a64)

```
var a = 1;
function one() {
    var b = 2;
    console.log(a, b);
}
one();
```

### 2.2 src\index.js #

src\index.js

```
const ObjectEnvironmentRecords = require('./ObjectEnvironmentRecords');
const LexicalEnvironment = require('./LexicalEnvironment');
const ExecutionContext = require('./ExecutionContext');
const ExecutionContexts = require('./ExecutionContexts');
const FunctionInstance = require('./FunctionInstance');

const ECStack = new ExecutionContexts();

const globalEnvironmentRecord = new ObjectEnvironmentRecords(global);

const globalLexicalEnvironment = new LexicalEnvironment(globalEnvironmentRecord, null);

let globalExecutionContext = new ExecutionContext(globalLexicalEnvironment, global);

ECStack.push(globalExecutionContext);

ECStack.current.lexicalEnvironment.createBinding('a');
ECStack.current.lexicalEnvironment.setBinding('a', undefined);

let oneFn = new FunctionInstance('one', 'var b = 2;\nconsole.log(a, b);',
    ECStack.current.lexicalEnvironment);
ECStack.current.lexicalEnvironment.createBinding('one');
ECStack.current.lexicalEnvironment.setBinding('one', oneFn);

ECStack.current.lexicalEnvironment.setBinding('a', 1);

let oneLexicalEnvironment = LexicalEnvironment.NewDeclarativeEnvironment(oneFn.scope);

let oneExecutionContext = new ExecutionContext(oneLexicalEnvironment, global);

ECStack.push(oneExecutionContext);

ECStack.current.lexicalEnvironment.createBinding('b');
ECStack.current.lexicalEnvironment.setBinding('b', undefined);

ECStack.current.lexicalEnvironment.setBinding('b', 2);

console.log(ECStack.current.lexicalEnvironment.getIdentifierReference('a')
    , ECStack.current.lexicalEnvironment.getIdentifierReference('b'));

ECStack.pop();
```

### 2.3 ObjectEnvironmentRecords.js #

src\ObjectEnvironmentRecords.js

```
const EnvironmentRecord = require('./EnvironmentRecord');
class ObjectEnvironmentRecords extends EnvironmentRecord {

}
module.exports = ObjectEnvironmentRecords;
```

### 2.4 DeclarativeEnvironmentRecords.js #

src\DeclarativeEnvironmentRecords.js

```
const EnvironmentRecord = require('./EnvironmentRecord');
class DeclarativeEnvironmentRecords extends EnvironmentRecord {

}
module.exports = DeclarativeEnvironmentRecords;
```

### 2.5 EnvironmentRecord.js #

src\EnvironmentRecord.js

```
class EnvironmentRecord {
    constructor(bindings) {
        this.bindings = bindings || {};
    }

    createBinding(N) {
        this.bindings[N] = undefined;
    }

    setBinding(N, V) {
        this.bindings[N] = V;
    }

    hasBinding(N) {
        return N in this.bindings;
    }

    getBindingValue(N) {
        return this.bindings[N];
    }
}
module.exports = EnvironmentRecord;
```

### 2.6 LexicalEnvironment.js #

src\LexicalEnvironment.js

```
const DeclarativeEnvironmentRecords = require("./DeclarativeEnvironmentRecords");
const ObjectEnvironmentRecords = require("./ObjectEnvironmentRecords");

class LexicalEnvironment {
    constructor(environmentRecord, outer) {
        this.environmentRecord = environmentRecord;
        this.outer = outer;
    }

    createBinding(N) {
        return this.environmentRecord.createBinding(N);
    }

    setBinding(N, V) {
        return this.environmentRecord.setBinding(N, V);
    }

    hasBinding(N) {
        return this.environmentRecord.hasBinding(N);
    }

    getBindingValue(N) {
        return this.environmentRecord.getBindingValue(N);
    }

    getIdentifierReference(name) {
        let lexicalEnvironment = this;
        do {
            let exists = lexicalEnvironment.hasBinding(name);
            if (exists) {
                return lexicalEnvironment.getBindingValue(name);
            } else {
                lexicalEnvironment = lexicalEnvironment.outer;
            }
        } while (lexicalEnvironment);
    }

    static NewDeclarativeEnvironment(lexicalEnvironment) {
        let envRec = new DeclarativeEnvironmentRecords();
        let env = new LexicalEnvironment(envRec, lexicalEnvironment);
        return env;
    }

    static NewObjectEnvironment(object, lexicalEnvironment) {
        let envRec = new ObjectEnvironmentRecords(object);
        let env = new LexicalEnvironment(envRec, lexicalEnvironment);
        return env;
    }
}
module.exports = LexicalEnvironment;
```

### 2.7 ExecutionContexts.js #

src\ExecutionContexts.js

```
class ExecutionContexts {
    constructor() {
        this.executionContexts = [];
    }
    push(executionContext) {
        this.executionContexts.push(executionContext);
    }
    get current() {
        return this.executionContexts[this.executionContexts.length - 1];
    }
    pop() {
        this.executionContexts.pop();
    }
}
module.exports = ExecutionContexts;
```

#### 2.8 ExecutionContext.js #

src\ExecutionContext.js

```
class ExecutionContext {
    constructor(lexicalEnvironment, thisBinding) {
        this.lexicalEnvironment = lexicalEnvironment;
        this.thisBinding = thisBinding;
    }
}
module.exports = ExecutionContext;
```

#### 2.9 FunctionInstance.js #

src\FunctionInstance.js

```
class FunctionInstance {
    constructor(name, code, scope) {
        this.name = name;
        this.code = code;
        this.scope = scope;
    }
}
module.exports = FunctionInstance;
```

## 3.支持块级作用域 #

- 支持块级作用域 (https://www.processon.com/diagraming/61beb9795653bb5a3e2b12fa)

#### 3.1 main.js #

```
var a = 1;
function one() {
    var b = 2;
+   {
+       let c = 3;
+       console.log(a, b, c);
+   }
+   {
+       let c = 4;
+       console.log(a, b, c);
+   }
}
one();
```

#### 3.2 EnvironmentRecord.js #

src\EnvironmentRecord.js

```
class EnvironmentRecord {
    constructor(bindings) {
        this.bindings = bindings || {};
    }
    /**
     * 创建变量
     * @param {*} N 名称
     */
    createBinding(N) {
        this.bindings[N] = undefined;
    }
    /**
     * 给N设置值V
     * @param {*} N 名称
     * @param {*} V 值
     */
    setBinding(N, V) {
        this.bindings[N] = V;
    }
    /**
     * 是否绑定一个变量
     * @param {*} N 名称
     */
    hasBinding(N) {
        return N in this.bindings;
    }
    /**
     * 获取N的值
     * @param {*} N   名称
     */
    getBindingValue(N) {
+       let value = this.bindings[N];
+       if (value.type === 'let' && value.uninitialized) {
+           throw new Error(`ReferenceError: Cannot access '${N}' before initialization`);
+       }
+       return value;
    }
}
module.exports = EnvironmentRecord;
```

### 3.3 ExecutionContext.js #

src\ExecutionContext.js

```
class ExecutionContext {
    constructor(lexicalEnvironment, thisBinding) {
+       this.variableEnvironment = this.lexicalEnvironment = lexicalEnvironment;
        this.thisBinding = thisBinding;
    }
}
module.exports = ExecutionContext;
```

### 3.4 src\index.js #

src\index.js

```
const ObjectEnvironmentRecords = require('./ObjectEnvironmentRecords');
const LexicalEnvironment = require('./LexicalEnvironment');
const ExecutionContext = require('./ExecutionContext');
const ExecutionContexts = require('./ExecutionContexts');
const FunctionInstance = require('./FunctionInstance');

//创建执行上下文栈
const ECStack = new ExecutionContexts();
//创建全局环境记录对象
const globalEnvironmentRecord = new ObjectEnvironmentRecords(global);
//创建全局环境
const globalLexicalEnvironment = new LexicalEnvironment(globalEnvironmentRecord, null);
//创建全局执行上下文
let globalExecutionContext = new ExecutionContext(globalLexicalEnvironment, global);
//把全局执行上下文放入执行上下文栈
ECStack.push(globalExecutionContext);

//创建a变量并初始化为undefined
+ECStack.current.variableEnvironment.createBinding('a');
+ECStack.current.variableEnvironment.setBinding('a', undefined);
//创建fn变量并赋值为函数
let oneFn = new FunctionInstance('one', 'var b = 2;\nconsole.log(a, b);',
    ECStack.current.lexicalEnvironment);
+ECStack.current.variableEnvironment.createBinding('one');
+ECStack.current.variableEnvironment.setBinding('one', oneFn);

//开始执行代码,给a变量赋值为1
+ECStack.current.variableEnvironment.setBinding('a', 1);
//遇到函数则创建一个新的词法环境
let oneLexicalEnvironment = LexicalEnvironment.NewDeclarativeEnvironment(oneFn.scope);
//创建one函数执行上下文
let oneExecutionContext = new ExecutionContext(oneLexicalEnvironment, global);
//把one函数执行上下文推入执行上下文栈并成为最新的执行上下文
ECStack.push(oneExecutionContext);

//创建并绑定变量b,执行变量提升
+ECStack.current.variableEnvironment.createBinding('b');
+ECStack.current.variableEnvironment.setBinding('b', undefined);
//开始执行函数代码，给变量b赋值为2
+ECStack.current.variableEnvironment.setBinding('b', 2);
+//备份当前的词法作用域
+let oldEnv = ECStack.current.lexicalEnvironment;
+//创建新的词法环境
+let blockEnv = LexicalEnvironment.NewDeclarativeEnvironment(oldEnv);
+blockEnv.createBinding('c');
+blockEnv.setBinding('c', { type: 'let', uninitialized: true });
+//让blockEnv成为当前执行上下文的词法环境
+ECStack.current.lexicalEnvironment = blockEnv;
+//开始执行块级作用域中的代码
+ECStack.current.lexicalEnvironment.setBinding('c', 3);
+console.log(
+    ECStack.current.lexicalEnvironment.getIdentifierReference('a')
+    , ECStack.current.lexicalEnvironment.getIdentifierReference('b')
+    , ECStack.current.lexicalEnvironment.getIdentifierReference('c'));
+ECStack.current.lexicalEnvironment = oldEnv;
+//备份当前的词法作用域
+oldEnv = ECStack.current.lexicalEnvironment;
+//创建新的词法环境
+blockEnv = LexicalEnvironment.NewDeclarativeEnvironment(oldEnv);
+blockEnv.createBinding('c');
+blockEnv.setBinding('c', { type: 'let', uninitialized: true });
+//让blockEnv成为当前执行上下文的词法环境
+ECStack.current.lexicalEnvironment = blockEnv;
+//开始执行块级作用域中的代码
+ECStack.current.lexicalEnvironment.setBinding('c', 4);
+console.log(
+    ECStack.current.lexicalEnvironment.getIdentifierReference('a')
+    , ECStack.current.lexicalEnvironment.getIdentifierReference('b')
+    , ECStack.current.lexicalEnvironment.getIdentifierReference('c'));
+ECStack.current.lexicalEnvironment = oldEnv;
ECStack.pop();
```

## 4.标准闭包 #

- 标准闭包 (https://www.processon.com/diagraming/61bebb770e3e74525cd0137b)

### 4.1 main.js #

```
var a = 1;
function one() {
    var b = 2;
+   return function two() {
+       console.log(a, b);
+   }
}
+let two = one();
+two();
```

### 4.2 src\index.js #

src\index.js

```
const ObjectEnvironmentRecords = require('./ObjectEnvironmentRecords');
const LexicalEnvironment = require('./LexicalEnvironment');
const ExecutionContext = require('./ExecutionContext');
const ExecutionContexts = require('./ExecutionContexts');
const FunctionInstance = require('./FunctionInstance');

//创建执行上下文栈
const ECStack = new ExecutionContexts();
//创建全局环境记录对象
const globalEnvironmentRecord = new ObjectEnvironmentRecords(global);
//创建全局环境
const globalLexicalEnvironment = new LexicalEnvironment(globalEnvironmentRecord, null);
//创建全局执行上下文
let globalExecutionContext = new ExecutionContext(globalLexicalEnvironment, global);
//把全局执行上下文放入执行上下文栈
ECStack.push(globalExecutionContext);

//创建a变量并初始化为undefined
ECStack.current.variableEnvironment.createBinding('a');
ECStack.current.variableEnvironment.setBinding('a', undefined);
//创建fn变量并赋值为函数
let oneFn = new FunctionInstance('one', 'var b = 2;\nconsole.log(a, b);',
    ECStack.current.lexicalEnvironment);
ECStack.current.variableEnvironment.createBinding('one');
ECStack.current.variableEnvironment.setBinding('one', oneFn);

//开始执行代码,给a变量赋值为1
ECStack.current.variableEnvironment.setBinding('a', 1);
//遇到函数则创建一个新的词法环境
let oneLexicalEnvironment = LexicalEnvironment.NewDeclarativeEnvironment(oneFn.scope);
//创建one函数执行上下文
let oneExecutionContext = new ExecutionContext(oneLexicalEnvironment, global);
//把one函数执行上下文推入执行上下文栈并成为最新的执行上下文
ECStack.push(oneExecutionContext);

//创建并绑定变量b,执行变量提升
ECStack.current.variableEnvironment.createBinding('b');
ECStack.current.variableEnvironment.setBinding('b', undefined);
+ECStack.current.variableEnvironment.createBinding('two');
+let twoFn = new FunctionInstance('two', 'console.log(a, b);', ECStack.current.lexicalEnvironment);
+ECStack.current.variableEnvironment.setBinding('two', twoFn);

//开始执行函数代码, 给变量b赋值为2
ECStack.current.variableEnvironment.setBinding('b', 2);
+//退出one的执行上下文
+ECStack.pop();
+//回到全局执行上下文下执行two函数
+let twoVariableEnvironment = LexicalEnvironment.NewDeclarativeEnvironment(twoFn.scope);
+//创建one函数执行上下文并设置词法环境为 localEnv
+let twoExecutionContext = new ExecutionContext(twoVariableEnvironment, global);
+//把one执行上下语言推入执行上下文栈并成为最新的执行上下文
+ECStack.push(twoExecutionContext);
+console.log(
+    ECStack.current.lexicalEnvironment.getIdentifierReference('a')
+    , ECStack.current.lexicalEnvironment.getIdentifierReference('b'))
+//退出two的执行上下文
+ECStack.pop();
```

### 4.3 V8中的闭包优化 #

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Documenttitle>
head>
<body>
    <script>
        function A1() {
            var a1 = { name: 'a1' };
            var a2 = { name: 'a2' };
            return function A2() {
                console.log(a1);
            }
        }
        debugger
        let A2 = A1();
        A2();
    script>
body>
html>
```