JavaScript执行（三）：你知道现在有多少种函数吗?

2019-02-28 winter

重学前端

进入课程 >

讲述：winter

时长 13:16　大小 12.16M

在前一篇文章中，我们大致了解了执行上下文是什么，也知道了任何语句的执行都会依赖特定的上下文。

一旦上下文被切换，整个语句的效果可能都会发生改变。那么，切换上下文的时机就显得非常重要了。

在 JavaScript，切换上下文最主要的场景是函数调用。在这一课，我们就来讲讲函数调用切换上下文的事情。我们在讲函数调用之前，首先来认识一下函数家族。

函数

在 ES2018 中，函数已经是一个很复杂的体系了，我在这里整理了一下。

第一种，普通函数：用 function 关键字定义的函数。

示例：

```
function foo(){
    // code
}
```

第二种，箭头函数：用 => 运算符定义的函数。

示例:

```
const foo = () => {
    // code
}
```

第三种，方法：在 class 中定义的函数。

示例：

```
class C {
    foo(){
        //code
    }
}
```

第四种，生成器函数：用 function * 定义的函数。

示例：

```
function* foo(){
    // code
}
```

第五种，类：用 class 定义的类，实际上也是函数。

示例：

```
class Foo {
    constructor(){
        //code
    }
}
```

第六 / 七 / 八种，异步函数：普通函数、箭头函数和生成器函数加上 async 关键字。

示例：

```
async function foo(){
    // code
}
const foo = async () => {
    // code
}
async function foo*(){
    // code
}
```

ES6 以来，大量加入的新语法极大地方便了我们编程的同时，也增加了很多我们理解的心智负担。要想认识这些函数的执行上下文切换，我们必须要对它们行为上的区别有所了解。

对普通变量而言，这些函数并没有本质区别，都是遵循了“继承定义时环境”的规则，它们的一个行为差异在于 this 关键字。

那么，this 关键字是什么呢，我们一起来看一看。

this 关键字的行为

this 是 JavaScript 中的一个关键字，它的使用方法类似于一个变量（但是 this 跟变量的行为有很多不同，上一节课我们讲了一些普通变量的行为和机制，也就是 var 声明和赋值、let 的内容）。

this 是执行上下文中很重要的一个组成部分。同一个函数调用方式不同，得到的 this 值也不同，我们看一个例子：

复制代码

```
function showThis(){
    console.log(this);
}

var o = {
    showThis: showThis
}

showThis(); // global
o.showThis(); // o
```

在这个例子中，我们定义了函数 showThis，我们把它赋值给一个对象 o 的属性，然后尝试分别使用两个引用来调用同一个函数，结果得到了不同的 this 值。

普通函数的 this 值由“调用它所使用的引用”决定，其中奥秘就在于：我们获取函数的表达式，它实际上返回的并非函数本身，而是一个 Reference 类型（记得我们在类型一章讲过七种标准类型吗，正是其中之一）。

Reference 类型由两部分组成：一个对象和一个属性值。不难理解 o.showThis 产生的 Reference 类型，即由对象 o 和属性“showThis”构成。

当做一些算术运算（或者其他运算时），Reference 类型会被解引用，即获取真正的值（被引用的内容）来参与运算，而类似函数调用、delete 等操作，都需要用到 Reference 类型中的对象。

在这个例子中，Reference 类型中的对象被当作 this 值，传入了执行函数时的上下文当中。

至此，我们对 this 的解释已经非常清晰了：**调用函数时使用的引用，决定了函数执行时刻的 this 值。**

实际上从运行时的角度来看，this 跟面向对象毫无关联，它是与函数调用时使用的表达式相关。

这个设计来自 JavaScript 早年，通过这样的方式，巧妙地模仿了 Java 的语法，但是仍然保持了纯粹的“无类”运行时设施。

如果，我们把这个例子稍作修改，换成箭头函数，结果就不一样了：

复制代码

```
const showThis = () => {
    console.log(this);
}

var o = {
    showThis: showThis
}

showThis(); // global
o.showThis(); // global
```

我们看到，改为箭头函数后，不论用什么引用来调用它，都不影响它的 this 值。

接下来我们看看“方法”，它的行为又不一样了：

复制代码

```
class C {
    showThis() {
        console.log(this);
    }
}
var o = new C();
var showThis = o.showThis;

showThis(); // undefined
o.showThis(); // o
```

这里我们创建了一个类 C，并且实例化出对象 o，再把 o 的方法赋值给了变量 showThis。

这时候，我们使用 showThis 这个引用去调用方法时，得到了 undefined。

所以，在方法中，我们看到 this 的行为也不太一样，它得到了 undefined 的结果。

按照我们上面的方法，不难验证出：生成器函数、异步生成器函数和异步普通函数跟普通函数行为是一致的，异步箭头函数与箭头函数行为是一致的。

this 关键字的机制

说完了 this 行为，我们再来简单谈谈在 JavaScript 内部，实现 this 这些行为的机制，让你对这部分知识有一个大概的认知。

函数能够引用定义时的变量，如上文分析，函数也能记住定义时的 this，因此，函数内部必定有一个机制来保存这些信息。

在 JavaScript 标准中，为函数规定了用来保存定义时上下文的私有属性[[Environment]]。

当一个函数执行时，会创建一条新的执行环境记录，记录的外层词法环境（outer lexical environment）会被设置成函数的[[Environment]]。

这个动作就是**切换上下文**了，我们假设有这样的代码：

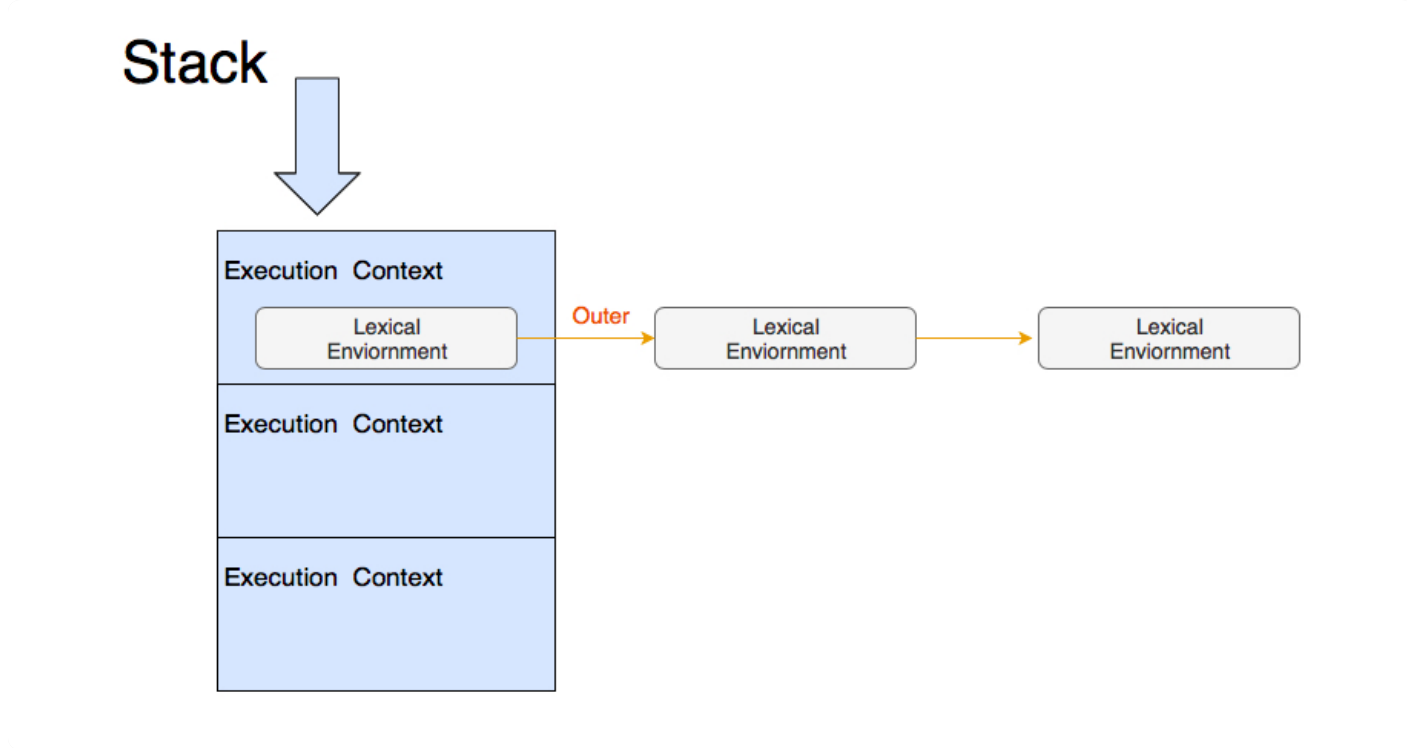
```
复制代码
var a = 1;
foo();

在别处定义了foo:

var b = 2;
function foo(){
    console.log(b); // 2
    console.log(a); // error
}
```

这里的 foo 能够访问 b（定义时词法环境），却不能访问 a（执行时的词法环境），这就是执行上下文的切换机制了。

JavaScript 用一个栈来管理执行上下文，这个栈中的每一项又包含一个链表。如下图所示：



当函数调用时，会入栈一个新的执行上下文，函数调用结束时，执行上下文被出栈。

而 this 则是一个更为复杂的机制，JavaScript 标准定义了 [[thisMode]] 私有属性。

[[thisMode]] 私有属性有三个取值。

- lexical：表示从上下文中找 this，这对应了箭头函数。
- global：表示当 this 为 undefined 时，取全局对象，对应了普通函数。
- strict：当严格模式时使用，this 严格按照调用时传入的值，可能为 null 或者 undefined。

非常有意思的是，方法的行为跟普通函数有差异，恰恰是因为 class 设计成了默认按 strict 模式执行。

我们可以用 strict 达成与上一节中方法的例子一样的效果:

复制代码

```
"use strict"
function showThis(){
    console.log(this);
}

var o = {
    showThis: showThis
```

```
}

showThis(); // undefined
o.showThis(); // o
```

函数创建新的执行上下文中的词法环境记录时，会根据[[thisMode]]来标记新纪录的[[ThisBindingStatus]]私有属性。

代码执行遇到 this 时，会逐层检查当前词法环境记录中的[[ThisBindingStatus]]，当找到有 this 的环境记录时获取 this 的值。

这样的规则的实际效果是，嵌套的箭头函数中的代码都指向外层 this，例如：

复制代码

```
var o = {}
o.foo = function foo(){
    console.log(this);
    return () => {
        console.log(this);
        return () => console.log(this);
    }
}

o.foo()()(); // o, o, o
```

这个例子中，我们定义了三层嵌套的函数，最外层为普通函数，两层都是箭头函数。

这里调用三个函数，获得的 this 值是一致的，都是对象 o。

JavaScript 还提供了一系列函数的内置方法来操纵 this 值，下面我们来了解一下。

操作 this 的内置函数

Function.prototype.call 和 Function.prototype.apply 可以指定函数调用时传入的 this 值，示例如下：

复制代码

```
function foo(a, b, c){
    console.log(this);
```

```
    console.log(a, b, c);
}
foo.call({}, 1, 2, 3);
foo.apply({}, [1, 2, 3]);
```

这里 call 和 apply 作用是一样的，只是传参方式有区别。

此外，还有 Function.prototype.bind 它可以生成一个绑定过的函数，这个函数的 this 值固定了参数：

复制代码

```
function foo(a, b, c){
    console.log(this);
    console.log(a, b, c);
}
foo.bind({}, 1, 2, 3)();
```

有趣的是，call、bind 和 apply 用于不接受 this 的函数类型如箭头、class 都不会报错。

这时候，它们无法实现改变 this 的能力，但是可以实现传参。

结语

在这一节课程中，我们认识了 ES2018 中规定的各种函数，我一共简单介绍了 8 种函数。

我们围绕 this 这个中心，介绍了函数的执行上下文切换机制。同时我们还讲解了 this 中的一些相关知识。包括了操作 this 的内置函数。

最后，留给你一个问题，你在日常开发中用过哪些函数类型呢？欢迎给我留言，我们一起讨论。

补充阅读：new 与 this

我们在之前的对象部分已经讲过 new 的执行过程，我们再来看一下：

- 以构造器的 prototype 属性（注意与私有字段[[prototype]]的区分）为原型，创建新对象；

- 将 this 和调用参数传给构造器，执行；
- 如果构造器返回的是对象，则返回，否则返回第一步创建的对象。

显然，通过 new 调用函数，跟直接调用的 this 取值有明显区别。那么我们今天讲的这些函数跟 new 搭配又会产生什么效果呢?

这里我整理了一张表：

函数类型	new
普通函数	新对象
箭头函数	报错
方法	报错
生成器	报错
类	新对象
异步普通函数	报错
异步箭头函数	报错
异步生成器函数	报错

我们可以看到，仅普通函数和类能够跟 new 搭配使用，这倒是给我们省去了不少麻烦。

上一篇 JavaScript执行（二）：闭包和执行上下文到底是怎么回事?

下一篇 JavaScript执行（四）：try里面放return，finally还会执行吗?

精选留言 (56)

写留言

奥斯特洛夫斯基

2019-02-28

var a = 1;
foo();

在别处定义了 foo：

var b = 2;
function foo(){
    console.log(b); // 2
    console.log(a); // error
}
为什么我执行出来是undefined ， 1

钟凯
2019-02-28

关于this，Kyle Simpson有四条总结：
1. 由new调用？绑定到新创建的对象。
2. 由call或者apply(或者bind)调用？绑定到指定的对象。
3. 由上下文对象调用？绑定到那个上下文对象。
4. 默认:在严格模式下绑定到undefined，否则绑定到全局对象。
例外：箭头函数不适用以上四条规则，它会继承外层函数调用的 this 绑定(无论 this 绑定到什么)。

2 62

郭鹏飞277
2019-02-28

老师，这个例子中的最后一行代码o.showThis(); // global
好像写错了，应该是C的实例o吧。

```
class C {
    showThis() {
        console.log(this);
    }
}
var o = new C();
var showThis = o.showThis;

showThis(); // undefined
o.showThis(); // global
```

作者回复: 哦哦 对的对的 我改一下

27

Rushan-Chen
2019-03-02

老师写的 "在别处定义了foo" 的意思是，这句话上下两部分的代码，不在同一个文件哒~
已经有同学贴了代码，是这样的：
```js
// 这是 foo.js 文件里的代码
var b = 2;
module.exports = function() { // 导出function
```

```
  console.log(b);
  console.log(a);
};
```

```js
// 这是test.js 文件里的代码
var foo = require("./foo.js"); // 引入function 为foo
var a = 1;
foo();
// node 执行 test.js 输出：
// -> 2
// -> ReferenceError: a is not defined
```

3 26

阿成

2019-03-16

这里先说声抱歉，之前可能误导了大家...
这里更新一下答案😅
@Rushan-Chen（虽然你并不能收到，希望极客时间赶紧增加@或者评论的功能，至少也展示个邮箱啊...不然找人都找不到，影响大家交流）
----------分割线----------
文中，winter老师所说的“在别处定义”的意思，应该就是指在另一个模块中定义，即：

在另一个模块中定义...这样引入这个模块时，b就定义且初始化了，而且在这个模块中访问不到变量a...

```
// module a.js
import {foo} from 'b.js'
var a = 1
foo()

// module b.js
var b = 2;
export function foo () {
  console.log(b); // 2
  console.log(a); // error
}
```

其实，只要foo访问不到变量a就可以了嘛:

```
var b = 2;
function foo () {
```

```
  console.log(b); // 2
  console.log(a); // error
}

void function () {
    var a = 1
    foo()
}()
```

12

Thomas Cho

2019-02-28

我发现啊，不能只看文中代码结果，还是得自己跑一下，我看了文章后很是疑惑，跑了一下[[Environment]]下方那段代码后，打印出来的是 undefined和1。而不是2和error，不知为何

1 12

TY

2019-03-04

晕得一塌糊涂... 结合上一章的 let var 来看, js 这门语言居然还能火成这样, 世界实在是太奇妙了😢

1 8

zhangbao

2019-02-28

老师，您好，看完文章后，我有几个问题：

1. 代码段

```
class C {
    showThis() {
        console.log(this);
    }
}
var o = new C();

o.showThis(); // 这里打印的 this 应该是 o 吗?
```

2. 介绍函数时提到了“方法”，定义是“在 class 中定义的函数”。但是举代码例子时，举了对象方法的例子。方法定义成“作为属性值的函数”，是不是更贴切一点呢?

3. 执行代码段

```
var a = 1;
foo();

// 在别处定义了 foo：

var b = 2;
function foo(){
    console.log(b); // 2
    console.log(a); // error
}
```

后，控制台，打印出的 b 是 undefined，a 是 1。跟老师描述的不一样，是我理解错了吗?

麻烦老师您解答一下，谢谢啦!

1 4

DXYF2E
2019-12-27

觉得没看懂的同学，我觉得可以结合李兵老师的「浏览器工作原理与实践」中的第10、11节课一起阅读，可能理解程度会好一些

2 3

x
2019-06-24

es6中箭头函数是没有this的吧，所以他不能用作构造函数，他的this取决于外部环境

> 作者回复: 不是“没有this”，是“使用定义时的this”。
> 也不是“所以不能用作构造函数”，没有这个因果关系。

3

jacklovepdf
2019-06-19

按照老师的理解，应该少了一种，类的方法也是可以加async的，亲测有效。

3

study

2019-05-09

@奥斯特洛夫斯基
var和function，只是提升声明，代码提升完成是下面的代码：

```
var a,b;
function foo(){};
a = 1;
foo();
b = 2;
```

所以a的值为1，b为undefined

3

Geek_376ed4

2019-03-01

```
// foo.js
var b=2;
module.exports = function() {
 console.log(b);//2
 console.log(a);//error

};

var foo = require("./foo.js");

var a=1;

foo();
```

3

肉卷

2019-12-26

```
var a = 1;
foo();
```

在别处定义了 foo：

```
var b = 2;
function foo(){
    console.log(b); // 2
    console.log(a); // error
```

}
这个例子，改成如下例子应该更容易让人理解一些：

```
var a = 1
  function test() {
    var b = 2

    test1()
  }
  function test1() {
    console.log(2, a, b);
  }
  test()
```

2

itgou
2019-03-09

```
var a = 1;
foo();
```

在别处定义了 foo：

```
var b = 2;
function foo(){
    console.log(b); // 2
    console.log(a); // error
}
```

这段代码我在chrome上执行出来是undefined,1.
我是写在一个js文件中，然后通过HTML的script引入，不知道老师说的在别处定义是什么意思，是写在两个js文件吗?

如果是两个文件，HTML中引入文件的顺序不同，会有不同的结果，一种报错，一种两个值都正常打印，我都试了的，总之怎么也不会有老师说的那种结果。

请老师看看是不是我哪里理解错了，还是老师写错了

2

呜呼千里快哉风
2020-02-19

哈哈

1

Geek_fc1551
2019-11-27

winter老师普通函数不是方法吗?

1 1

奋奋
2019-10-11

老师我这样对Reference的理解对吗？？？
showThis(); // Reference中的对象是global
(false || showThis)() // Reference由于运算而被解引用，
然后触发this机制[[thisMode]]私有属性的global取值

作者回复: 没错

1

Mr.杨
2019-04-04

默认this 就是普通方法的this
隐式this 变量接收方法后的this 会发生隐式指向错误
显式this 对象.方法的this
固定this call apply bind 的this
分为四种

1

存
2019-03-18

var a = 1;
foo();

在别处定义了 foo：

var b = 2;
function foo(){
    console.log(b); // 2
    console.log(a); // error
}
为什么我执行出来是2，1

1