

## 04 | 函数表达式：涉及大量概念，函数表达式到底该怎么学？

2020-03-24 李兵

图解 Google V8

[进入课程 >](#)



讲述：李兵

时长 15:35 大小 14.28M



你好，我是李兵。

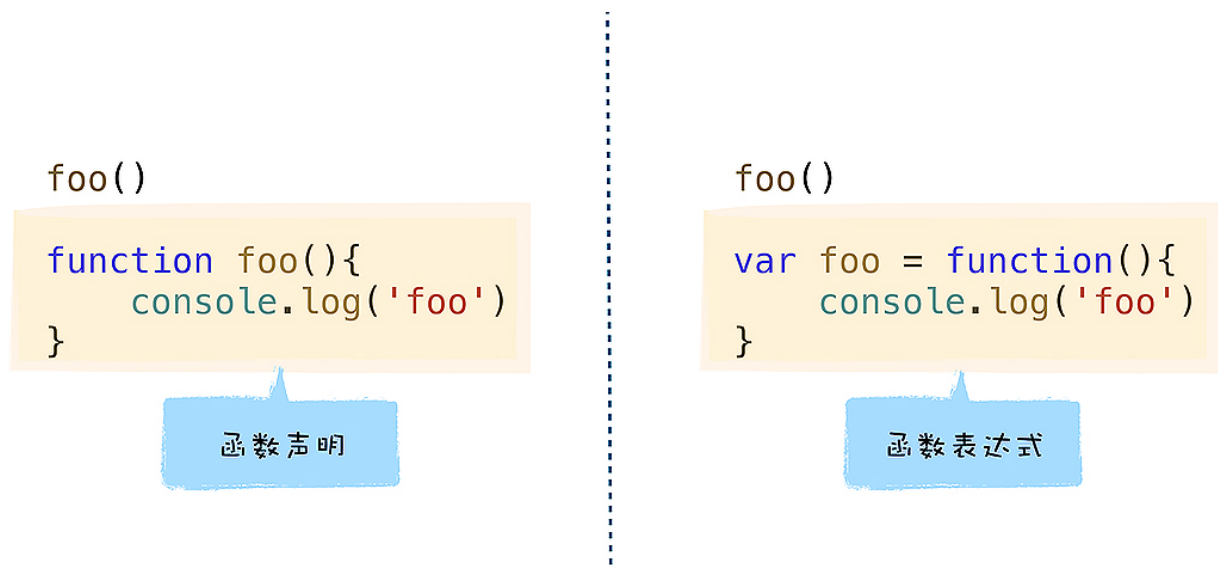
前面几节我们聊了 V8 中的对象和函数，并介绍了函数为什么会被称为是一等公民，了解这些之后，我们就可以来学习函数表达式了。

函数表达式在 JavaScript 中非常基础也非常重要，使用函数表达式可以用来实现代码隐藏，还可以实现变量隔离，所以函数表达式被广泛地应用在各个项目中，了解函数表达式的底层工作机制，可以帮助我们更加深刻地理解项目。



但是，学好函数表达式并不容易。因为它涉及到了很多底层概念，比如表达式、语句、函数即对象（在 JavaScript 中）等，而且函数表达式和函数声明看起来类似，都是定义一个函

数，然后再调用该函数，很容易把二者搞混淆了。




你知道两者的区别吗？

实际上，函数表达式和函数声明有着本质上的差异。理解了这种差异，你对函数表达式的理解也就加深了。

## 函数声明与函数表达式的差异

那么它们具体有什么差异呢？我们先来看一段代码：

```
1 foo()
2 function foo(){
3     console.log('foo')
4 }
```

 复制代码

在这段代码中，我声明了一个 `foo` 函数，然后在 `foo` 函数之前调用了 `foo` 函数，执行这段代码，我们看到 `foo` 函数被正确执行了。（你可能会好奇，代码不是至上而下执行吗，为什么在函数声明之前就可以调用该函数了呢？这个问题我们先留一边，后文中会进行解答。）

再来看另外一段代码：

```
1 foo()  
2 var foo = function () {  
3     console.log('foo')  
4 }
```

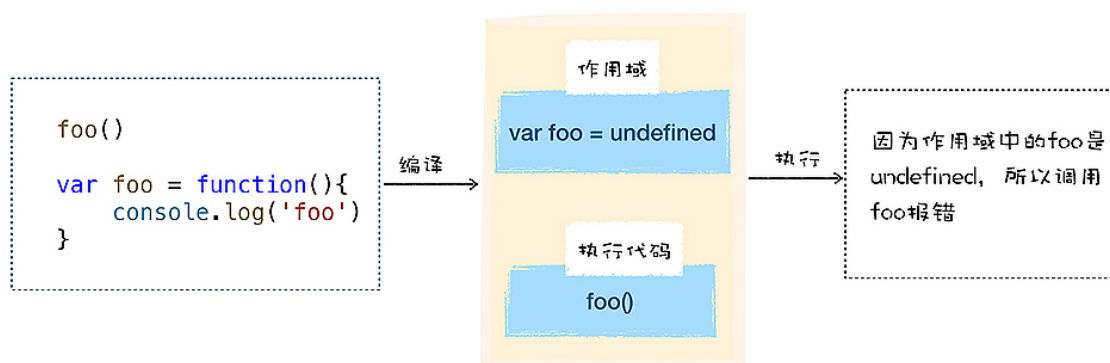
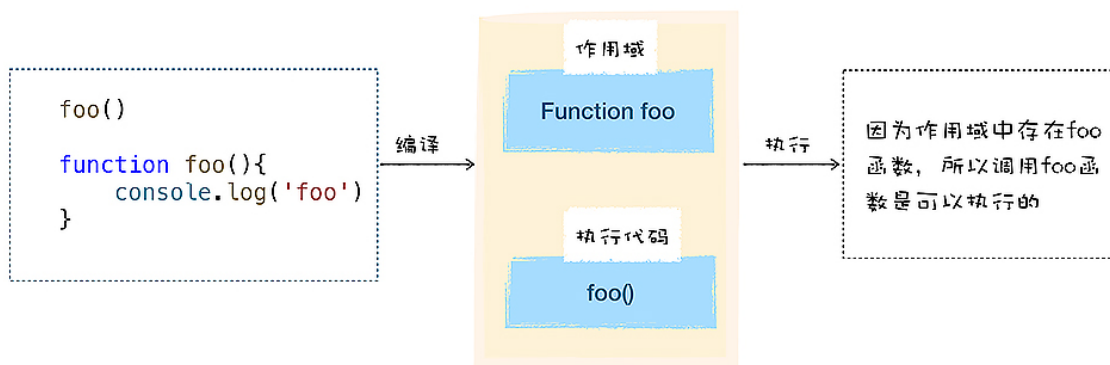
在这段代码中，我定义了一个变量 `foo`，然后将一个函数赋值给了变量 `foo`，同样在源码中，我们也是在 `foo` 函数的前面调用 `foo`，执行这段代码，我们发现报错了，提示的错误信息如下所示：

```
1 VM130:1 Uncaught TypeError: foo is not a function  
2     at <anonymous>:1:1
```

这是告诉我们，变量 `foo` 并不是一个函数，所以无法被调用。

同样是在定义的函数之前调用函数，第一段代码就可以正确执行，而第二段代码却报错，这是为什么呢？

其主要原因是这两种定义函数的方式具有不同语义，不同的语义触发了不同的行为。



不同的语义，触发不同的行为


因为语义不同，所以我们给这两种定义函数的方式使用了不同的名称，第一种称之为**函数声明**，第二种称之为**函数表达式**。

下面我们就来分别分析下，函数声明和函数表达式的语义，以及 V8 是怎么处理函数声明和函数表达式的。

## V8 是怎么处理函数声明的？

我们先来看函数声明，**函数声明**定义了一个**具有指定参数的函数**，其声明语法如下所示：


```
1 function name([param,[, param,[..., param]]]) {  
2     [statements]  
3 }
```

 复制代码

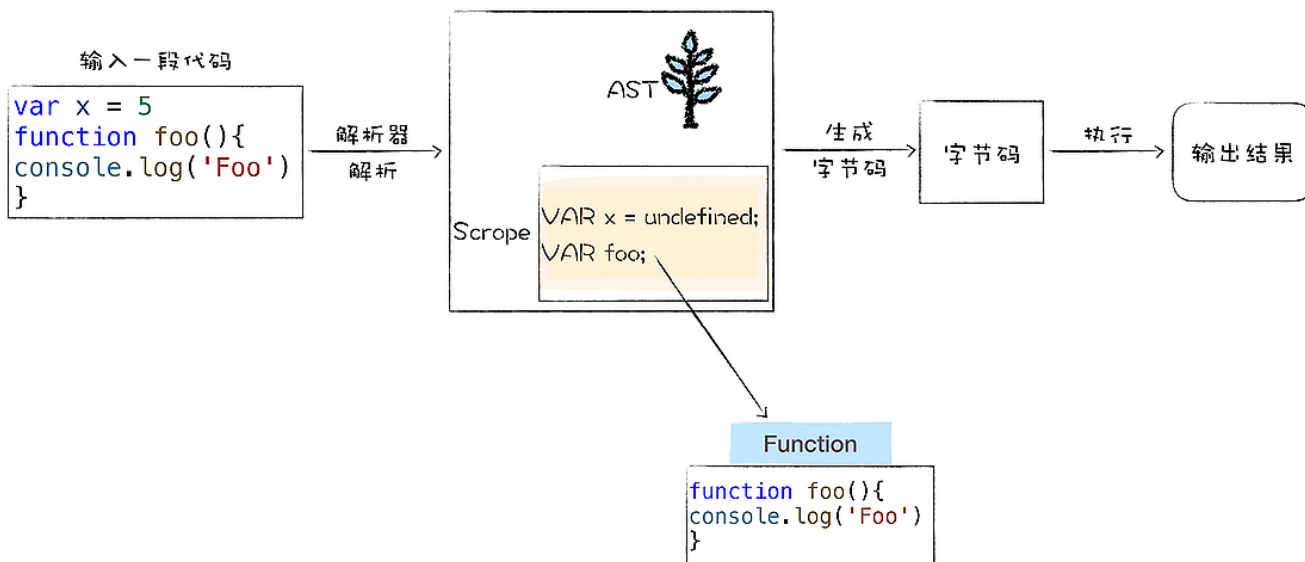
接下来我们来看看 V8 是怎么处理函数声明的。

我们知道，V8 在执行 JavaScript 的过程中，会先对其进行**编译**，然后再**执行**，比如下面这段代码：

```
1 var x = 5  
2 function foo(){  
3     console.log('Foo')  
4 }
```

 复制代码

V8 执行这段代码的流程大致如下图所示：



在编译阶段，如果解析到函数声明，那么 V8 会将这个函数声明转换为内存中的函数对象，并将其放到作用域中。同样，如果解析到了某个变量声明，也会将其放到作用域中，但是会将其值设置为 undefined，表示该变量还未被使用。

然后在 V8 执行阶段，如果使用了某个变量，或者调用了某个函数，那么 V8 便会去作用域查找相关内容。

关于作用域的数据，你也可以使用 D8 来查看，具体操作方式如下：

将这段代码保存到 test.js 中；

使用 “d8 --print-scopes test.js” 命令即可查看作用域的状态。

执行这段指令之后，打印出如下信息：

[复制代码](#)

```
1 Global scope:
2 global { // (0x7fb62281ca48) (0, 50)
3   // will be compiled
4   // 1 stack slots
5   // temporary vars:
6   TEMPORARY .result; // (0x7fb62281cfe8) local[0]
7   // local vars:
8   VAR x; // (0x7fb62281cc98)
9   VAR foo; // (0x7fb62281cf40)
10
11
12 function foo () { // (0x7fb62281cd50) (22, 50)
```


```
13      // lazily parsed
14      // 2 heap slots
15  }
16 }
```

上面这段就是 V8 生成的作用域，我们可以看到，作用域中包含了变量 x 和 foo，变量 x 的默认值是 undefined，变量 foo 指向了 foo 函数对象，foo 函数对象被 V8 存放在内存中的堆空间了，这些变量都是在编译阶段被装进作用域中的。

因为在执行之前，这些变量都被提升到作用域中了，所以在执行阶段，V8 当然就能获取到所有的定义变量了。我们把这种在编译阶段，将所有的变量提升到作用域的过程称为**变量提升**。

了解了变量提升，我们就能解释，为什么可以在函数声明之前调用该函数了，这是因为声明的函数在编译阶段就被提升到作用域中，在执行阶段，只要是在作用域中存在的变量或者对象，都是可以使用的。

对于变量提升，函数和普通对象还是存在一些差异的，通过上面的分析我们知道，如果是一个普通变量，变量提升之后的值都是 undefined，如果是声明的函数，那么变量提升之后的值则是函数对象，我们可以通过下面的代码来实践下：

 复制代码

```
1 console.log(x)
2 console.log(foo)
3 var x = 5
4 function foo(){
5 }
```

执行上面这段代码，我们可以看到，普通变量 x 的值就是 undefined，而函数对象 foo 的值则是完整的对象，那这又是为什么呢？这就是涉及到表达式和语句的区别了。

简单地理解，表达式就是表示值的式子，而语句是操作值的式子。

比如：



[复制代码](#)

```
1 x = 5
```

就是表达式，因为执行这段代码，它会返回一个值。同样，`6 === 5` 也是一个表达式，因为它会返回 `False`。

而语句则不同了，比如你定义了一个变量：

[复制代码](#)

```
1 var x
```

这就是一个语句，执行该语句时，V8 并不会返回任何值给你。

同样，当我声明了一个函数时，这个函数声明也是一个语句，比如下面这段函数声明：

[复制代码](#)

```
1 function foo(){  
2   return 1  
3 }
```

当执行到这段代码时，V8 并没有返回任何的值，它只是解析 `foo` 函数，并将函数对象存储到内存中。

`x = 5`

表达式

执行表达式会返回一个值

```
if(1){  
  console.log(1)  
}
```


语句

可以操作表达式

表达式和语句

了解了表达式和语句的区别，接下来我们继续分析上面的问题。我们知道，在 V8 执行 `var x = 5` 这段代码时，会认为它是两段代码，一段是定义变量的语句，一段是赋值的表达式，如下所示：

```
1 var x = undefined
2 x = 5
```

 复制代码


首先，在变量提升阶段，V8 并不会执行赋值的表达式，该阶段只会分析基础的语句，比如变量的定义，函数的声明。

而这两行代码是在不同的阶段完成的，`var x` 是在编译阶段完成的，也可以说是在变量提升阶段完成的，而 `x = 5` 是表达式，所有的表达式都是在执行阶段完成的。

在变量提升阶段，V8 将这些变量存放在作用域时，还会给它们赋一个默认的 `undefined` 值，所以在定义一个普通的变量之前，使用该变量，那么该变量的值就是 `undefined`。

现在我们知道，**表达式是不会在编译阶段执行的**，那么函数声明是表达式还是语句呢？你可以看下面这段函数声明：

```
1 function foo(){
2   console.log('Foo')
3 }
```

 复制代码

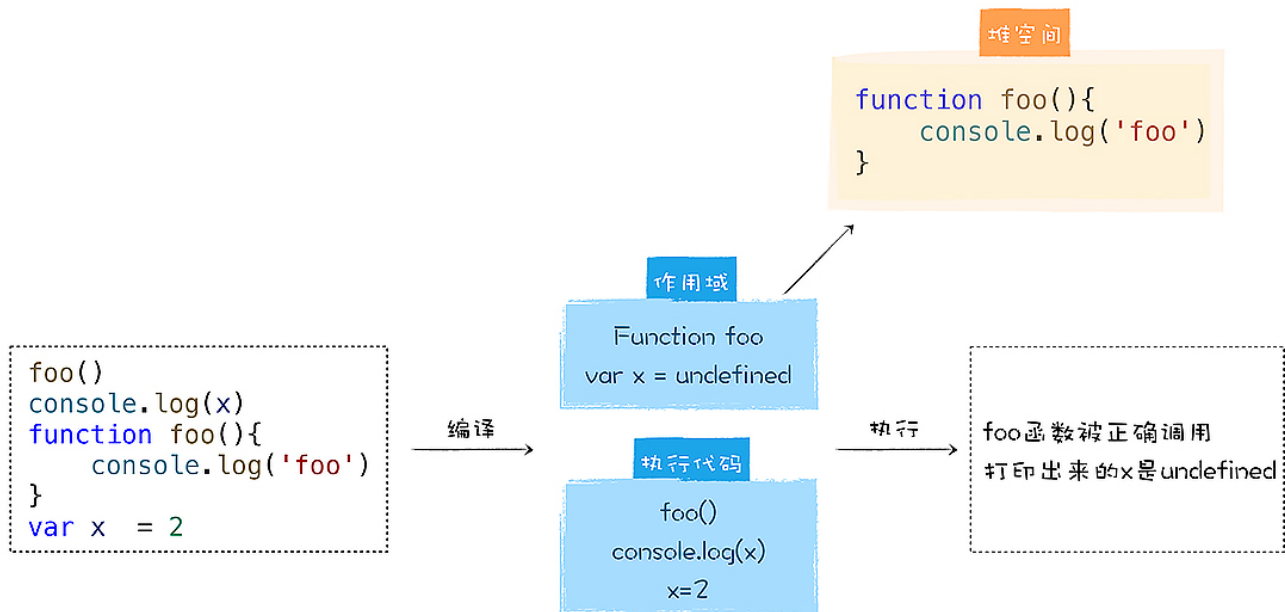
执行上面这段代码，它并没有输出任何内容，所以可以肯定，函数声明并不是一个表达式，而是一个语句。V8 在变量提升阶段，如果遇到函数声明，那么 V8 同样会对该函数声明执行变量提升操作。

函数也是一个对象，所以在编译阶段，V8 就会将整个函数对象提升到作用域中，并不是给该函数名称赋一个 `undefined`，理解这一点尤为重要。

总的来说，在 V8 解析 JavaScript 源码的过程中，如果遇到普通的变量声明，那么便会将其提升到作用域中，并给该变量赋值为 `undefined`，如果遇到的是函数声明，那么 V8 会



在内存中为声明生成函数对象，并将该对象提升到作用域中。



## V8 是怎么处理函数表达式?

了解了函数声明，在来看看函数表达式。我们在一个表达式中使用 `function` 来定义一个函数，那么就把该函数称为函数表达式。

比如：

```
1  foo = 1
```

[复制代码](#)

是一个表达式，这时候我们把右边的数字 1 替换成函数定义，那么这就变成了函数表达式，如下所示：

```
1  foo = function (){
2    console.log('foo')
3  }
```

[复制代码](#)

函数表达式与函数声明的最主要区别有以下三点：

函数表达式是在表达式语句中使用 function 的，最典型的表达式是 “a=b” 这种形式，因为函数也是一个对象，我们把 “a = function (){}” 这种方式称为函数表达式；

在函数表达式中，可以省略函数名称，从而创建匿名函数（anonymous functions）；


一个函数表达式可以被用作一个即时调用的函数表达式——IIFE（Immediately Invoked Function Expression）。

了解了函数表达式，我们就来分析这段代码：

 复制代码

```
1 foo()
2 var foo = function () {
3     console.log('foo')
4 }
```

当执行这段代码的时候，V8 在编译阶段会先查找声明语句，你可以把这段代码拆分为下面两行代码：

 复制代码

```
1 var foo = undefined
2 foo = function () {
3     console.log('foo')
4 }
```

第一行是声明语句，所以 V8 在解析阶段，就会在作用域中创建该对象，并将该对象设置为 undefined，第二行是函数表达式，在编译阶段，V8 并不会处理函数表达式，所以也就不会将该函数表达式提升到作用域中了。

那么在函数表达式之前调用该函数 foo，此时的 foo 只是指向了 undefined，所以就相当于调用一个 undefined，而 undefined 只是一个原生对象，并不是函数，所以当然会报错了。


## 立即调用的函数表达式（IIFE）

现在我们知道了，在编译阶段，V8 并不会处理函数表达式，而 JavaScript 中的立即函数调用表达式正是使用了这个特性来实现了非常广泛的应用，下面我们就来一起看看立即函数调

用表达式。

JavaScript 中有一个圆括号运算符，圆括号里面可以放一个表达式，比如下面的代码：


```
1 (a=3)
```

 复制代码

括号里面是一个表达式，整个语句也是一个表达式，最终输出 3。

如果在小括号里面放上一段函数的定义，如下所示：

```
1 (function () {  
2     //statements  
3 })
```

 复制代码

因为小括号之间存放的必须是表达式，所以如果在小括号里面定义一个函数，那么 V8 就会把这个函数看成是函数表达式，执行时它会返回一个函数对象。

存放在括号里面的函数便是一个函数表达式，它会返回一个函数对象，如果我直接在表达式后面加上调用的括号，这就称**立即调用函数表达式**（IIFE），比如下面代码：


```
1 (function () {  
2     //statements  
3 })()
```

 复制代码

因为函数立即表达式也是一个表达式，所以 V8 在编译阶段，并不会为该表达式创建函数对象。这样的—个好处就是不会污染环境，函数和函数内部的变量都不会被其他部分的代码访问到。

在 ES6 之前，JavaScript 中没有私有作用域的概念，如果在多人开发的项目中，你模块中的变量可能覆盖掉别人的变量，所以使用函数立即表达式就可以将我们内部变量封装起来，避免了相互之间的变量污染。

另外，因为函数立即表达式是立即执行的，所以将一个函数立即表达式赋给一个变量时，不是存储 IIFE 本身，而是存储 IIFE 执行后返回的结果。如下所示：

 复制代码

```
1 var a = (function () {  
2     return 1  
3 })()
```

## 总结

今天我们要学习 V8 是如何处理函数表达式的。函数表达式在实际的项目应用中非常广，不过由于函数声明和函数表达式之间非常类似，非常容易引起人们的误解，所以我们先从通过两段容易让人误解的代码，分析了函数声明和函数表达式之间的区别。函数声明的本质是语句，而函数表达式的本质则是表达式。

函数声明和变量声明类似，V8 在编译阶段，都会对其执行变量提升的操作，将它们提升到作用域中，在执行阶段，如果使用了某个变量，就可以直接去作用域中去查找。

不过 V8 对于提升函数和提升变量的策略是不同的，如果提升了一个变量，那么 V8 在将变量提升到作用域中时，还会为其设置默认值 `undefined`，如果是函数声明，那么 V8 会在内存中创建该函数对象，并提升整个函数对象。

函数表达式也是表达式的一种，在编译阶段，V8 并不会将表达式中的函数对象提升到全局作用域中，所以无法在函数表达式之前使用该函数。函数立即表达式是一种特别的表达式，主要用来封装一些变量、函数，可以起到变量隔离和代码隐藏的作用，因此在一些大的开源项目中有广泛的应用。

## 思考题

留给你一道经典面试题，看看下面这两段代码打印出来的结果是什么？欢迎你在留言区与我分享讨论。

 复制代码

```
1 var n = 1;  
2 (function foo(){  
3     n = 100;  
4     console.log(n);
```

```
5 }())
6 console.log(n);
```

复制代码

```
1 var n = 1;
2 function foo(){
3     n = 100;
4     console.log(n);
5 }
6 console.log(n);
7 foo()
```

感谢你的阅读，如果你觉得这篇文章对你有所启发，也欢迎把它分享给你的朋友。

# 图解 Google V8

## 一门课搞懂 JavaScript 执行逻辑

李兵

前盛大创新院高级研究员



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 03 | 快属性和慢属性：V8采用了哪些策略提升了对对象属性的访问速度？

### 精选留言 (5)

写留言



刘大夫

2020-03-24

个人理解：第一题，编译阶段，先在全局作用域内声明变量 n 的值为undefined，然后声明函数表达式；编译阶段结束，开始执行阶段，将 1 赋值给 n，函数表达式“立即执行”，欲将 100 赋值给变量 n，但在当前作用域内没有找到 n 的声明，沿着作用域链向上查找，在全局作用域内找到了变量 n，将 100 赋给它，接着执行立即函数调用表达式中的打印语句，输出 n，为100；接着执行后面的打印语句，此时 n 的值已经由 1 被修改为...

展开 ▾



2



一步

2020-03-24

在文中一会说 解析阶段，一会说编译阶段，这里这两个概念是一样的吧，解析阶段就是编制阶段，生成 作用域和字节码

展开 ▾



我爱学习

2020-03-24

```
(function foo() {}());  
console.log(foo);  
VM2044:2 Uncaught ReferenceError: foo is not defined  
    at <anonymous>:2:13
```

...

展开 ▾



洋洋

2020-03-24

第一题输出: 100 100

第二题输出: 1 100



Stony.修行僧

2020-03-24

为什么要这么设计反人类的机制

展开 ▾





