JavaScript 的 this 原理

作者： 阮一峰

日期： 2018年6月18日

一、问题的由来

学懂 JavaScript 语言，一个标志就是理解下面两种写法，可能有不一样的结果。

```
var obj = {
  foo: function () {}
};

var foo = obj.foo;

// 写法一
obj.foo()

// 写法二
foo()
```

上面代码中，虽然 obj.foo 和 foo 指向同一个函数，但是执行结果可能不一样。请看下面的例子。

```
var obj = {
  foo: function () { console.log(this.bar) },
  bar: 1
};

var foo = obj.foo;
var bar = 2;

obj.foo() // 1
foo() // 2
```

这种差异的原因，就在于函数体内部使用了 this 关键字。很多教科书会告诉你， this 指的是函数运行时所在的环境。对于 obj.foo() 来说， foo 运行在 obj 环境，所以 this 指向 obj ；对于 foo() 来说， foo 运行在全局环境，所以 this 指向全局环境。所以，两者的运行结果不一样。

这种解释没错，但是教科书往往不告诉你，为什么会这样？也就是说，函数的运行环境到底是怎么决定的？举例来说，为什么 `obj.foo()` 就是在 `obj` 环境执行，而一旦 `var foo = obj.foo` ， `foo()` 就变成在全局环境执行？

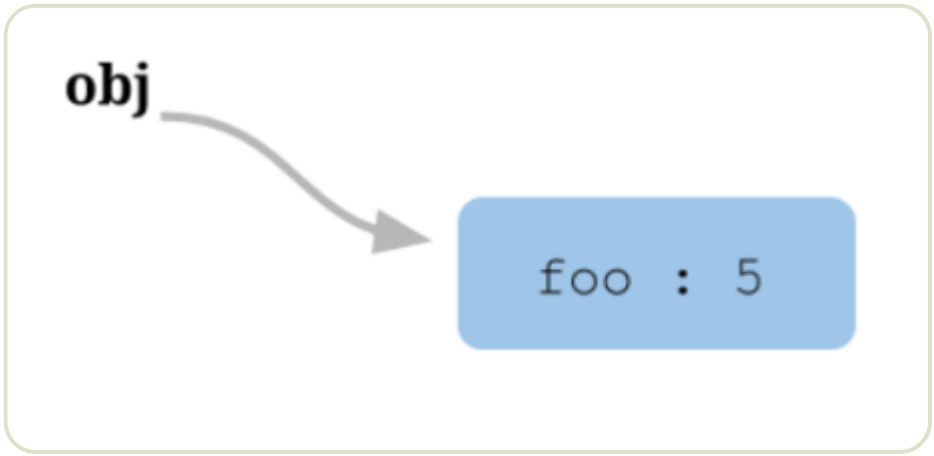
本文就来解释 JavaScript 这样处理的原理。理解了这一点，你就会彻底理解 `this` 的作用。

二、内存的数据结构

JavaScript 语言之所以有 `this` 的设计，跟内存里面的数据结构有关系。

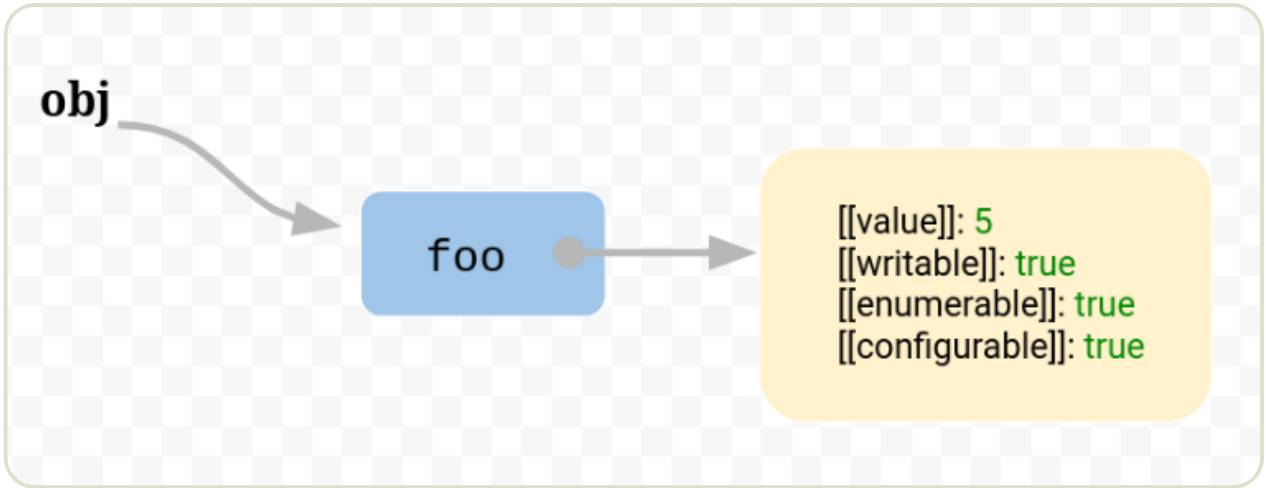
```
var obj = { foo:  5 };
```

上面的代码将一个对象赋值给变量 `obj` 。JavaScript 引擎会先在内存里面，生成一个对象 `{ foo: 5 }` ，然后把这个对象的内存地址赋值给变量 `obj` 。



也就是说，变量 `obj` 是一个地址（reference）。后面如果要读取 `obj.foo` ，引擎先从 `obj` 拿到内存地址，然后再从该地址读出原始的对象，返回它的 `foo` 属性。

原始的对象以字典结构保存，每一个属性名都对应一个属性描述对象。举例来说，上面例子的 `foo` 属性，实际上是以下面的形式保存的。



```
{
  foo: {
    [[value]]: 5
    [[writable]]: true
    [[enumerable]]: true
    [[configurable]]: true
  }
}
```

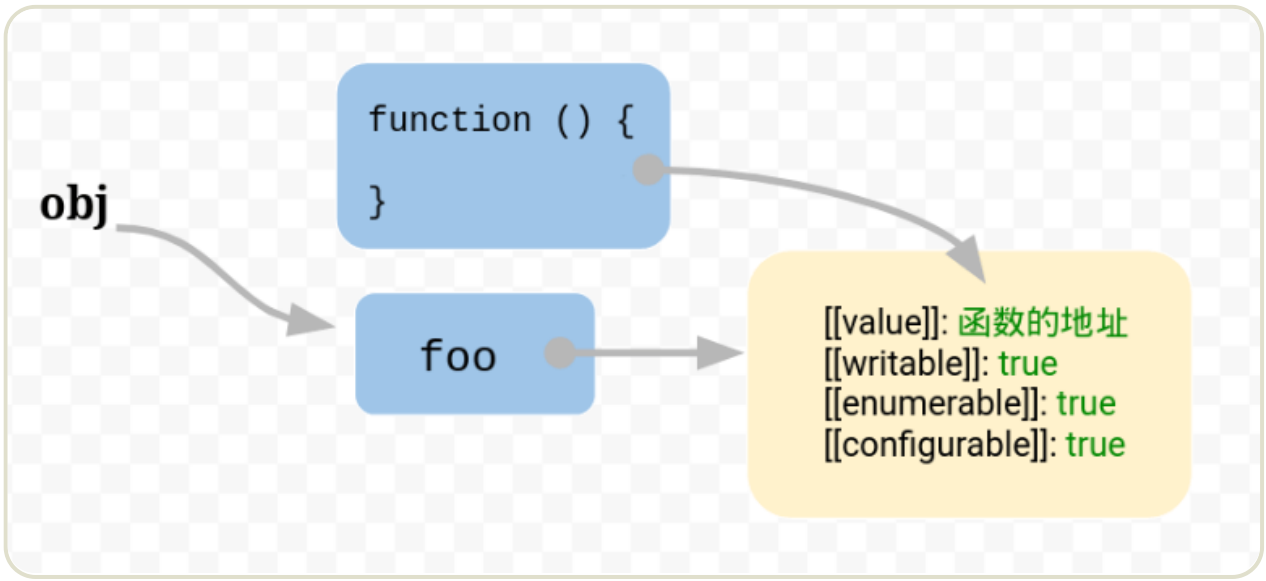
注意， `foo` 属性的值保存在属性描述对象的 `value` 属性里面。

三、函数

这样的结构是很清晰的，问题在于属性的值可能是一个函数。

```
var obj = { foo: function () {} };
```

这时，引擎会将函数单独保存在内存中，然后再将函数的地址赋值给 `foo` 属性的 `value` 属性。




```
{
  foo: {
    [[value]]: 函数的地址
    ...
  }
}
```

由于函数是一个单独的值，所以它可以在不同的环境（上下文）执行。

```
var f = function () {};
var obj = { f: f };

// 单独执行
f()

// obj 环境执行
obj.f()
```

四、环境变量

JavaScript 允许在函数体内部，引用当前环境的其他变量。

```
var f = function () {
  console.log(x);
};
```

上面代码中，函数体里面使用了变量 `x` 。该变量由运行环境提供。

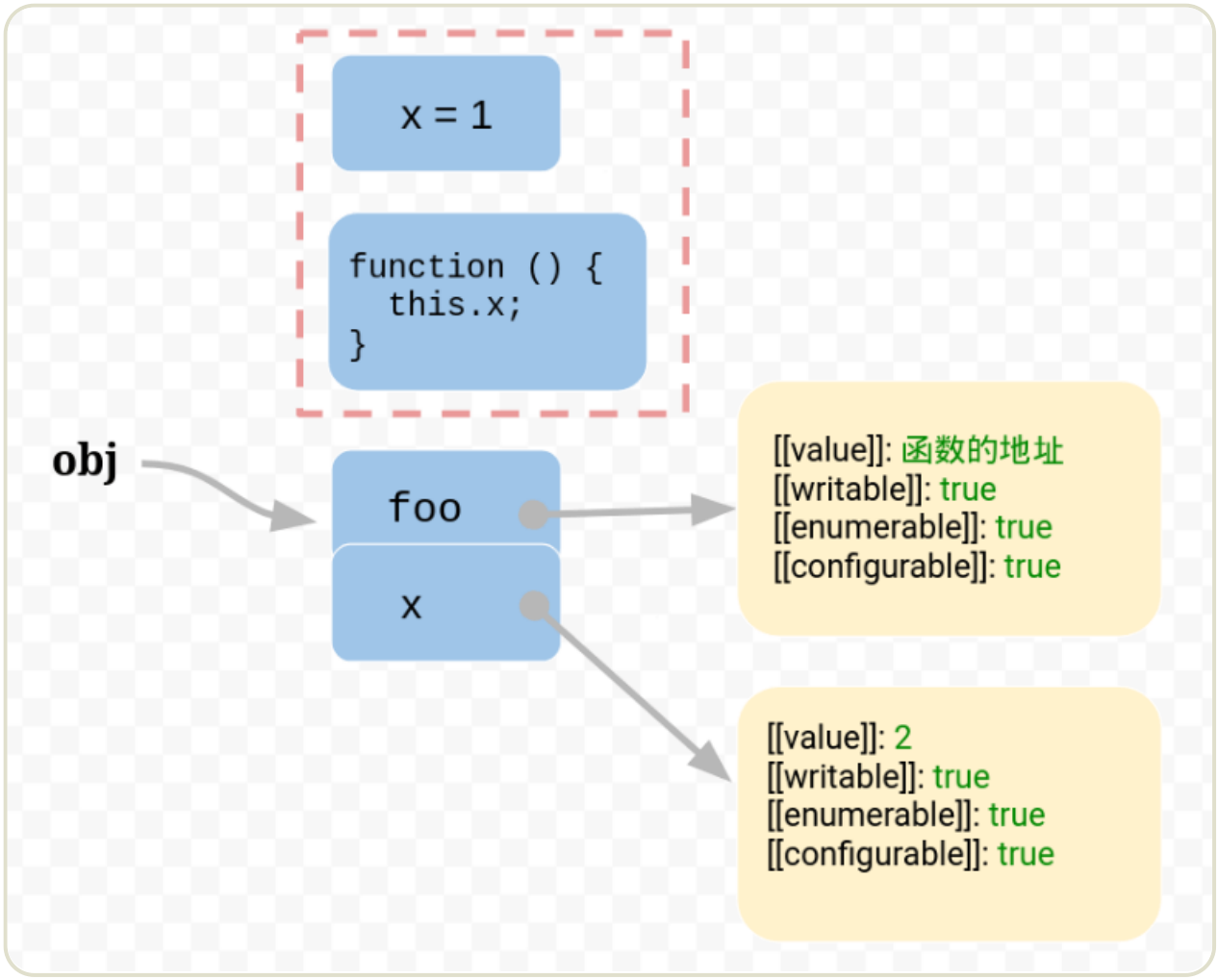
现在问题就来了，由于函数可以在不同的运行环境执行，所以需要有一种机制，能够在函数体内部获得当前的运行环境（context）。所以， `this` 就出现了，它的设计目的就是在函数体内部，指代函数当前的运行环境。

```
var f = function () {
  console.log(this.x);
}
```

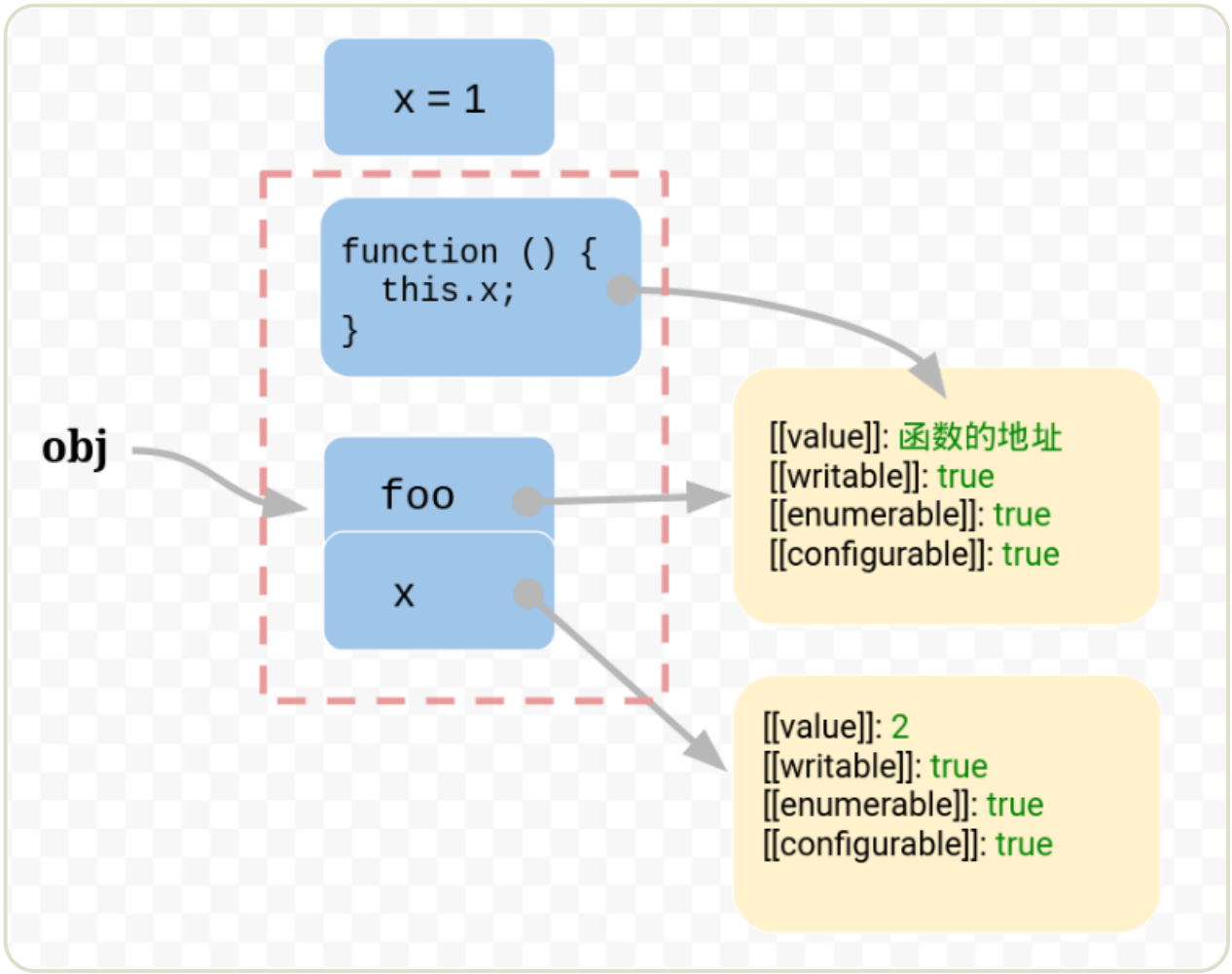
上面代码中，函数体里面的 `this.x` 就是指当前运行环境的 `x` 。

```
var f = function () {
  console.log(this.x);
}

var x = 1;
var obj = {
  f: f,
  x: 2,
};

// 单独执行
f() // 1

// obj 环境执行
obj.f() // 2
```

上面代码中，函数 `f` 在全局环境执行， `this.x` 指向全局环境的 `x` 。



在 `obj` 环境执行， `this.x` 指向 `obj.x` 。



回到本文开头提出的问题， `obj.foo()` 是通过 `obj` 找到 `foo` ，所以就是在 `obj` 环境执行。一旦 `var foo = obj.foo` ，变量 `foo` 就直接指向函数本身，所以 `foo()` 就变成在全局环境执行。

（完）

相关文章

- **2021.01.20:** 剪贴板操作 Clipboard API 教程

  > 一、简介 浏览器允许 JavaScript 脚本读写剪贴板，自动复制或粘贴内容。

- **2020.12.28:** Fetch API 教程

> fetch()是 XMLHttpRequest 的升级版，用于在 JavaScript 脚本里面发出 HTTP 请求。

- **2020.09.15:** 轻松学会 React 钩子：以 useEffect() 为例

> 五年多前，我写过 React 系列教程。不用说，内容已经有些过时了。

- **2020.08.20:** Node.js 如何处理 ES6 模块

> 学习 JavaScript 语言，你会发现它有两种格式的模块。

Weibo | Twitter | GitHub

Email: yifeng.ruan@gmail.com