08 改变 this 指向、深入理解 call/apply/bind 的原理

更新时间：2020-08-10 14:48:08

> 天才免不了有障碍，因为障碍会创造天才。——罗曼·罗兰

在上文中，我们翻来覆去讲的，都是“多数情况下”，this 遵循的指向机制。在另外一些情况下 this 是不遵循这个机制的。改变 this 的指向，我们主要有两条路：

通过改变书写代码的方式做到（比如上一节提到的箭头函数）。

显式地调用一些方法来帮忙。

两条路都是命题热点。其中第一条路，因为比较简单，我们就先拿它开刀：

改变书写代码的方式，进而改变 this 的指向

唱反调的箭头函数

箭头函数我们在上文已经讲过。在本节再强调一下（因为确实是个非常热门的考点，重复 1w 次也不过分）：

```
var a = 1

var obj = {
  a: 2,
  // 声明位置
  showA: () => {
    console.log(this.a)
  }
}

// 调用位置
obj.showA() // 1
```

当我们将普通函数改写为箭头函数时，箭头函数的 this 会在书写阶段（即声明位置）就绑定到它父作用域的 this 上。无论后续我们如何调用它，都无法再为它指定目标对象 —— 因为箭头函数的 this 指向是静态的，“一次便是一生”。

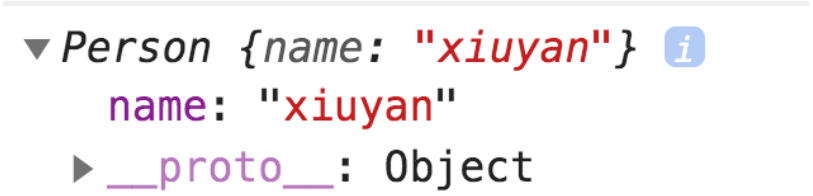
构造函数里的 this

当我们使用构造函数去 new 一个实例的时候：

```
function Person(name) {
  this.name = name
  console.log(this)
}

var person = new Person('xiuyan')
```

构造函数里面的 this 会绑定到我们 new 出来的这个对象上：



显式地调用一些方法来帮忙

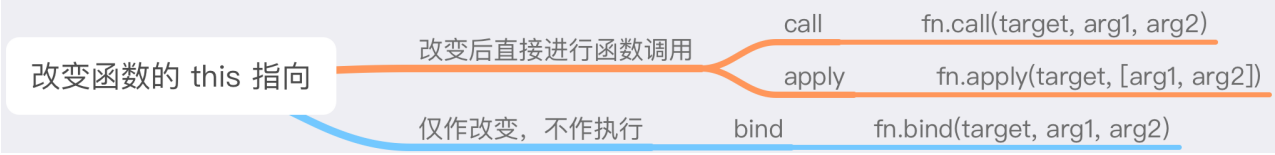
改变 this 指向，我们常用的是 call、 apply 和 bind 方法。

考虑到实际开发中我们改变 this 指向的场景非常多，所以这三种方法的使用在面试中考察的频率也比较高。最常见的考法，是问询三种方法的**使用及区别**。但很多时候，为了能够进一步试探你对 this 相关概念理解和掌握的深度， 面试官会考察你 call、apply 和 bind 的**实现机制**，甚至可能会要求你手写代码。

因此，针对 call、 apply 和 bind，我们不仅要会用、会辨析，更要对其原理知根知底。接下来，我们将这三种方法的考察方式汇聚到两道题里面，大家若能掌握这两个问题，就可以做到举一反三，知一解百。

基本问答题：call、apply 和 bind 是干嘛的？如何使用？它们之间有哪些区别？

解析：这里我给大家画了一张思维导图：



结合这张图来说明，会清楚得多：

call、apply 和 bind，都是用来**改变函数的 this 指向**的。

call、apply 和 bind 之间的区别比较大，前者在改变 this 指向的同时，也会把目标函数给执行掉；后者则只负责改造 this，不作任何执行操作。

call 和 apply 之间的区别，则体现在对入参的要求上。前者只需要将目标函数的入参逐个传入即可，后者则希望入参以数组形式被传入。

进阶编码题：模拟实现一个 call/apply/bind 方法

这三种方法在实现层面上非常相似，我们以 call 方法的实现为例，带大家深入理解一下这类方法的模拟思路：

call 方法的模拟

在实现 call 方法之前，我们先来看一个 call 的调用示范：

```
var me = {
  name: 'xiuyan'
}

function showName() {
  console.log(this.name)
}

showName.call(me) // xiuyan
```

结合 call 表现出的特性，我们首先至少能想到以下两点：

- call 是可以被所有的函数继承的，所以 call 方法应该被定义在 Function.prototype 上
- call 方法做了两件事：

1. 改变 this 的指向，将 this 绑到第一个入参指定的的对象上去；
2. 根据输入的参数，执行函数。

结合这两点，我们一步一步来实现 call 方法。首先，**改变 this 的指向**：

showName 在 call 方法调用后，表现得就像是 me 这个对象的一个方法一样。

所以我们最直接的一个联想是，如果能把 showName 直接塞进 me 对象里就好了，像这样：

```
var me = {
  name: 'xiuyan',
  showName: function() {
    console.log(this.name)
  }
}

me.showName()
```

但是这样做有一个问题，因为在 call 方法里，me 是一个入参：

```
showName.call(me) // xiuyan
```

用户在传入 me 这个对象的时候， 想做的仅仅是让 call 把 showName 里的 this 给改掉，而不想给 me 对象新增一个 showName 方法。所以说我们在执行完 me.showName 之后，还要记得把它给删掉。遵循这个思路，我们来模拟一下 call 方法（注意看注释）：

```
Function.prototype.myCall = function(context) {
    // step1: 把函数挂到目标对象上（这里的 this 就是我们要改造的的那个函数）
    context.func = this
    // step2: 执行函数
    context.func()
    // step3: 删除 step1 中挂到目标对象上的函数，把目标对象"完璧归赵"
    delete context.func
}
```

有兴趣的同学，可以测试一下我们的 myCall:

```
showName.myCall(me) // xiuyan
```

在我们这个例子里，myCall 的执行结果结果与 call 无差，撒花～～

到这里，我们已经实现了 **“改变 this 的指向”** 这个功能点。现在我们的 myCall 还需要**具备读取函数入参的能力**，类比于 call 的这种调用形式：

```
var me = {
  name: 'Chris'
}

function showFullName(surName) {
  console.log(`${this.name} ${surName}`)
}

showFullName.call(me, 'Lee') // Chris Lee
```

读取函数入参，具体来说其实是读取 call 方法的第二个到最后一个入参。要做到这一点，我们可以借助数组的扩展符（注意阅读注释，如果对 ‘...’ 这个符号感到陌生，需要补习一下 ES6 中扩展运算符相关的知识）：

```
// '...'这个扩展运算符可以帮助我们把一系列的入参变为数组
function readArr(...args) {
    console.log(args)
}

readArr(1,2,3) // [1,2,3]
```

我们把这个逻辑用到我们的 myCall 方法里：

```
Function.prototype.myCall = function(context, ...args) {
  ...
  console.log('入参是', args)
}
```

就能通过 args 这个数组拿到我们想要的入参了。把 args 数组代表的目标入参重新展开，传入目标方法里，就大功告成了：

```
Function.prototype.myCall = function(context, ...args) {
  // step1: 把函数挂到目标对象上（这里的 this 就是我们要改造的的那个函数）
  context.func = this
  // step2: 执行函数，利用扩展运算符将数组展开
  context.func(...args)
  // step3: 删除 step1 中挂到目标对象上的函数，把目标对象"完璧归赵"
  delete context.func
}
```

现在我们来测试一下功能完备的 myCall 方法：

```
Function.prototype.myCall = function(context, ...args) {
  // step1: 把函数挂到目标对象上（这里的 this 就是我们要改造的的那个函数）
  context.func = this
  // step2: 执行函数，利用扩展运算符将数组展开
  context.func(...args)
  // step3: 删除 step1 中挂到目标对象上的函数，把目标对象"完璧归赵"
  delete context.func
}

var me = {
  name: 'Chris'
}

function showFullName(surName) {
  console.log(`${this.name} ${surName}`)
}

showFullName.myCall(me, 'Lee') // Chris Lee
```

结果与 call 方法无差！

以上，我们就成功模拟了一个 call 方法出来。

基于这个最基本的 call 思路，大家还可以为这个方法作能力扩充：

比如如果我们第一个参数传了 null 怎么办？是不是可以默认给它指到 window 去？函数如果是有返回值的话怎么办？是不是新开一个 result 变量存储一下这个值，最后 return 出来就可以了？等等等等 —— 这些都是小事儿。当面试官问你 “如何模拟 call 方法的实现的时候”，他最想听的其实就楼上这两个核心功能点的实现思路，其它的，都是锦上添花～

基于对 call 方法的理解，写出一个 apply 方法（更改读取参数的形式） 和 bind 方法（延迟目标函数执行的时机）不是什么难事，只需要大家在上面这段代码的基础上作改造即可。（前提是你对 apply 方法和 bind 方法的特性和用法要心知肚明～）。

课后作业

独立实现一个自己的 myApply 和 myBind 方法。（可以参考上面的思路，也可以放飞自我，自由发挥～）

这里鼓励大家在评论区贴出自己的答案，更鼓励各位一起来 review 评论区的代码。博取众长，相信各位最后一定能从他人的代码中汲取到不少养分～

}

07 this 基本指向原则解析

09 JS 更进一步：执行上下文与调用栈