

[Sign up](#)[mqyqingfeng / Blog](#) Public[Notifications](#)[Fork](#) 4.4k[Star](#) 26.5k[Code](#)[Issues](#) 178[Pull requests](#) 4[Actions](#)[Projects](#)[Wiki](#)[Security](#)

JavaScript深入之call和apply的模拟实现 #11

[New issue](#)[Open](#)

mqyqingfeng opened this issue on May 2, 2017 · 200 comments



mqyqingfeng commented on May 2, 2017 • edited

[Owner](#)

call

一句话介绍 call：

call() 方法在使用一个指定的 this 值和若干个指定的参数值的前提下调用某个函数或方法。

举个例子：

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

Assignees

No one assigned

Labels

[深入系列](#)

Projects

None yet

Milestone

No milestone

Development

```
bar.call(foo); // 1
```

注意两点：

1. call 改变了 this 的指向，指向到 foo
2. bar 函数执行了

模拟实现第一步

那么我们该怎么模拟实现这两个效果呢？

试想当调用 call 的时候，把 foo 对象改造成如下：

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

这个时候 this 就指向了 foo，是不是很简单呢？

但是这样却给 foo 对象本身添加了一个属性，这可不行呐！

不过也不用担心，我们用 delete 再删除它不就好了~

所以我们模拟的步骤可以分为：

1. 将函数设为对象的属性
2. 执行该函数

No branches or pull requests

129 participants



3. 删除该函数

以上个例子为例，就是：

```
// 第一步
foo.fn = bar
// 第二步
foo.fn()
// 第三步
delete foo.fn
```

fn 是对象的属性名，反正最后也要删除它，所以起成什么都无所谓。

根据这个思路，我们可以尝试着去写第一版的 **call2** 函数：

```
// 第一版
Function.prototype.call2 = function(context) {
  // 首先要获取调用call的函数，用this可以获取
  context.fn = this;
  context.fn();
  delete context.fn;
}

// 测试一下
var foo = {
  value: 1
};

function bar() {
  console.log(this.value);
}

bar.call2(foo); // 1
```

正好可以打印 1 哎！是不是很开心！(～▽～)～

模拟实现第二步

最开始也讲了，call 函数还能给定参数执行函数。举个例子：

```
var foo = {  
  value: 1  
};  
  
function bar(name, age) {  
  console.log(name)  
  console.log(age)  
  console.log(this.value);  
}  
  
bar.call(foo, 'kevin', 18);  
// kevin  
// 18  
// 1
```

注意：传入的参数并不确定，这可咋办？

不急，我们可以从 Arguments 对象中取值，取出第二个到最后一个参数，然后放到一个数组里。

比如这样：

```
// 以上个例子为例，此时的arguments为：  
// arguments = {  
//   0: foo,  
//   1: 'kevin',  
//   2: 18,  
//   length: 3  
// }  
// 因为arguments是类数组对象，所以可以用for循环  
var args = [];  
for(var i = 1, len = arguments.length; i < len; i++) {
```

```
    args.push('arguments[' + i + ']');
}

// 执行后 args为 ["arguments[1]", "arguments[2]", "arguments[3]"]
```

不定长的参数问题解决了，我们接着要把这个参数数组放到要执行的函数的参数里面去。

```
// 将数组里的元素作为多个参数放进函数的形参里
context.fn(args.join(','))
// (0_o)??
// 这个方法肯定是不行的啦!!!
```

也许有人想到用 ES6 的方法，不过 call 是 ES3 的方法，我们为了模拟实现一个 ES3 的方法，要用到 ES6 的方法，好像.....，嗯，也可以啦。但是我们这次用 eval 方法拼成一个函数，类似于这样：

```
eval('context.fn(' + args + ')')
```

这里 args 会自动调用 Array.toString() 这个方法。

所以我们的第二版克服了两个大问题，代码如下：

```
// 第二版
Function.prototype.call2 = function(context) {
    context.fn = this;
    var args = [];
    for(var i = 1, len = arguments.length; i < len; i++) {
        args.push('arguments[' + i + ']');
    }
    eval('context.fn(' + args + ')');
    delete context.fn;
}

// 测试一下
```

```
var foo = {
  value: 1
};

function bar(name, age) {
  console.log(name)
  console.log(age)
  console.log(this.value);
}

bar.call2(foo, 'kevin', 18);
// kevin
// 18
// 1
```

(๑•̀ㅂ•́)و✧

模拟实现第三步

模拟代码已经完成 80%，还有两个小点要注意：

1.this 参数可以传 null，当为 null 的时候，视为指向 window

举个例子：

```
var value = 1;

function bar() {
  console.log(this.value);
}

bar.call(null); // 1
```

虽然这个例子本身不使用 call，结果依然一样。

2.函数是可以有返回值的！

举个例子：

```
var obj = {
  value: 1
}

function bar(name, age) {
  return {
    value: this.value,
    name: name,
    age: age
  }
}

console.log(bar.call(obj, 'kevin', 18));
// Object {
//   value: 1,
//   name: 'kevin',
//   age: 18
// }
```

不过都很好解决，让我们直接看第三版也就是最后一版的代码：

```
// 第三版
Function.prototype.call2 = function (context) {
  var context = context || window;
  context.fn = this;

  var args = [];
  for(var i = 1, len = arguments.length; i < len; i++) {
    args.push(arguments[i]);
  }

  var result = eval('context.fn(' + args + ')');
```

```
        delete context.fn;
        return result;
    }

    // 测试一下
    var value = 2;

    var obj = {
        value: 1
    }

    function bar(name, age) {
        console.log(this.value);
        return {
            value: this.value,
            name: name,
            age: age
        }
    }

    bar.call2(null); // 2

    console.log(bar.call2(obj, 'kevin', 18));
    // 1
    // Object {
    //   value: 1,
    //   name: 'kevin',
    //   age: 18
    // }
```

到此，我们完成了 call 的模拟实现，给自己一个赞 b (￣▽￣) d

apply的模拟实现

apply 的实现跟 call 类似，在这里直接给代码，代码来自于知乎 @郑航的实现：


```
Function.prototype.apply = function (context, arr) {  
    var context = Object(context) || window;  
    context.fn = this;  
  
    var result;  
    if (!arr) {  
        result = context.fn();  
    }  
    else {  
        var args = [];  
        for (var i = 0, len = arr.length; i < len; i++) {  
            args.push('arr[' + i + ']');  
        }  
        result = eval('context.fn(' + args + ')');  
    }  
  
    delete context.fn;  
    return result;  
}
```

下一篇文章

[JavaScript深入之bind的模拟实现](#)

重要参考

[知乎问题 不能使用call、apply、bind，如何用js实现 call 或者 apply 的功能？](#)

深入系列


JavaScript深入系列目录地址：<https://github.com/mqyqingfeng/Blog>。

JavaScript深入系列预计写十五篇左右，旨在帮大家捋顺JavaScript底层知识，重点讲解如原型、作用域、执行上下文、变量对象、this、闭包、按值传递、call、apply、bind、new、继承等难点概念。

如果有错误或者不严谨的地方，请务必给予指正，十分感谢。如果喜欢或者有所启发，欢迎star，对作者也是一种鼓励。

 113  5  8  3  17  8  4

  **mqqyqingfeng** added the **深入系列** label on May 2, 2017

  **mqqyqingfeng** changed the title ~~JavaScript深入之call和apply的模拟实现~~ **JavaScript深入之call和apply的模拟实现** on May 2, 2017

  **mqqyqingfeng** mentioned this issue on May 3, 2017

JavaScript深入之bind的模拟实现 #12

 Open

  **mqqyqingfeng** mentioned this issue on May 4, 2017

JavaScript深入之new的模拟实现 #13

 Open

 **JuniorTour** commented on May 4, 2017 ...

受益匪浅！学到了很多，谢谢前辈！

有一个小问题：call2第三版和apply的函数内，是不是不必要 `var context = ...`，直接`context=...`即可？

 12



mqyqingfeng commented on May 4, 2017

Owner

Author

...

哈哈，确实可以，没有注意到这点，感谢指出，(๑•̀ㅂ•́)و✧



6



2



2



3



fantasy123 commented on May 17, 2017

...

```
arr.push('arguments['+i+']');
```

请问这里为什么是一个拼接操作呢？



1



jawil commented on May 17, 2017 • edited ▼

...

eval函数接收参数是个字符串

定义和用法

eval() 函数可计算某个字符串，并执行其中的 JavaScript 代码。

语法：

```
eval(string)
```

string必需。要计算的字符串，其中含有要计算的 JavaScript 表达式或要执行的语句。该方法只接受原始字符串作为参数，如果 string 参数不是原始字符串，那么该方法将不作任何改变地返回。因此请不要为 eval() 函数传递 String 对象来作为参数。

简单来说吧，就是用JavaScript的解析引擎来解析这一堆字符串里面的内容，这么说吧，你可以这么理解，你把 `eval` 看成是 `<script>` 标签。

```
eval('function Test(a,b,c,d){console.log(a,b,c,d)};Test(1,2,3,4)')
```

@fantasy123

👍 77 🤔 7 🎉 8 😞 2 ❤️ 9 👁 4



mqyqingfeng commented on May 17, 2017

Owner Author ...

@jawil 感谢回答哈~

@fantasy123 最终目的是为了拼出一个参数字符串，我们一步一步看：

```
var args = [];  
for(var i = 1, len = arguments.length; i < len; i++) {  
    args.push('arguments[' + i + ']');  
}
```

最终的数组为：

```
var args = [arguments[1], arguments[2], ...]
```

然后

```
var result = eval('context.fn(' + args + ')');
```

在eval中，args 自动调用 args.toString()方法，eval的效果如 jawil所说，最终的效果相当于：

```
var result = context.fn(arguments[1], arguments[2], ...);
```

这样就做到了把传给call的参数传递给了context.fn函数



99



2



6



 **mqqyingfeng** mentioned this issue on May 23, 2017

JavaScript深入之创建对象的多种方式以及优缺点 #15

Open



 **mqqyingfeng** mentioned this issue on May 23, 2017

JavaScript深入之继承的多种方式和优缺点 #16

Open



 **mqqyingfeng** mentioned this issue on May 24, 2017

JavaScript深入之参数按值传递 #10

Open



 **mqqyingfeng** mentioned this issue on May 26, 2017

JavaScript深入系列15篇正式完结！ #17

Open



 **yy9306** mentioned this issue on May 26, 2017

JavaScript深入之bind的模拟实现 yy9306/yy9306.github.io#1

Open



 **yy9306** mentioned this issue on May 26, 2017

JavaScript深入之new的模拟实现 yy9306/yy9306.github.io#3

Open



yy9306 mentioned this issue on May 26, 2017

JavaScript深入之参数按值传递 yy9306/yy9306.github.io#6

Open



yy9306 mentioned this issue on May 26, 2017

JavaScript深入之创建对象的多种方式以及优缺点
yy9306/yy9306.github.io#7

Open



yy9306 mentioned this issue on May 26, 2017

JavaScript深入之继承的多种方式和优缺点 yy9306/yy9306.github.io#11

Open



lz-lee commented on May 29, 2017

...

apply郑航的实现，循环是不是应该从 $i = 1$ 开始？



mqyqingfeng commented on May 29, 2017

Owner

Author

...

@lz-lee call 的实现中，是通过 arguments 取各个参数，所以从 1 开始，省略掉为 0 的 context，而 apply 的实现中，arr 直接就表示参数的数组，循环这个参数数组，直接就从 0 开始。



4



lz-lee commented on May 29, 2017

...

粗心大意了。感谢提醒。@mqyqingfeng



lynn1824 commented on May 31, 2017

...

分析的很透彻，点个赞！



qianlongo commented on May 31, 2017

...

赞赞赞



mqyqingfeng commented on Jun 2, 2017

Owner

Author

...

@lynn1824 @qianlongo 感谢夸奖，写的时候我就觉得模拟实现一遍 call 和 apply 最能让大家明白 call 和 apply 的原理 (～▽～)～



lrx900515 commented on Jun 5, 2017

...

var context = Object(context) || window; 这里有问题吗？context为null时Object(null)返回空对象，不会被赋值为window



mqyqingfeng commented on Jun 6, 2017

Owner Author ...

@lzl900515 没有什么问题哈，非严格模式下，指定为 `null` 或 `undefined` 时会自动指向全局对象，郑航写的是严格模式下的，我写的是非严格模式下的，实际上现在的模拟代码有一点没有覆盖，就是当值为原始值（数字，字符串，布尔值）的 `this` 会指向该原始值的自动包装对象。

👍 2



hujiulong commented on Jun 13, 2017 • edited ▼

...

`context.fn = this;` 这里似乎漏掉了一个很关键的问题，如果 `context` 本来就有 `fn` 这个成员怎么办。这里只能给一个原来不存在的名字

```
var id = 0;
while ( context[ id ] ) {
  id ++;
}
context[ id ] = this;
```

不过这个方法似乎有点傻

👍 13

😬 5



mqyqingfeng commented on Jun 13, 2017

Owner Author ...

@hujiulong 哈哈，有道理哈~ 确实会覆盖之前对象的方法，还好模拟实现 `call` 和 `apply` 的目的在于让大家通过模拟实现了解 `call` 和 `apply` 的原理，实际开发的时候还是要直接使用 `call` 和 `apply` 的~



libin1991 commented on Jun 26, 2017

...

实在是佩服！



jasperchou commented on Jul 17, 2017

...

```
// 以上个例子为例，此时的arguments为：
// arguments = {
//     0: foo,
//     1: 'kevin',
//     2: 18,
//     length: 3
// }
// 因为arguments是类数组对象，所以可以用for循环
var args = [];
for(var i = 1, len = arguments.length; i < len; i++) {
    args.push('arguments[' + i + ']');
}

// 执行后 args为 [foo, 'kevin', 18]
```

|| // 执行后 args为 [foo, 'kevin', 18]

这一句可能造成误导。结果为：["arguments[1]", "arguments[2]"]
虽然后面确实会用eval执行，但是此处还没有。



194 hidden items

[Load more...](#)



hengistchan commented on Mar 15, 2021

...

```
Function.prototype.apply_ = function (context, args) {  
  context = Object(context) || window  
  context.fn = this;  
  let result = context.fn(...(args || []));  
  delete context.fn;  
  return result;  
}  
var foo = function (name) {  
  console.log(name)  
}  
var obj = {}  
foo.apply_(obj, ['张三'])
```

这样写行不行？

我和你差不多



lurenacm mentioned this issue on Mar 30, 2021

面试 | call, apply, bind 的模拟实现和经典面试题 [lurenacm/againJS#6](#)

[Open](#)



chinaOrange commented on Apr 13, 2021

...

call的实现中

```
var context = context || window;
```

是否应该改成

```
var context = Object(context) || window; // 原始类型要包装成对象
```



GalliumWang commented on Apr 26, 2021

...

现在可以把 eval 部分更新成 spread 操作符啦



xccjk mentioned this issue on Apr 26, 2021

浅谈JavaScript中call与apply的模拟实现 [xccjk/x-blog#56](#)

🔒 Closed



86driver commented on May 7, 2021

...

我有个疑问

```
// 第一版 Function.prototype.call2 = function(context) { // 首先要获取调用call的函数, 用this  
    可以获取 context.fn = this; context.fn(); delete context.fn; }
```

这里的this怎么指向的直接就是函数

按照作者在这篇中讲的去理解, 这里的this 不应该是 Function.prototype 吗

有没有大佬解答一下 ~~~



withf commented on May 22, 2021



我有个疑问

```
// 第一版 Function.prototype.call2 = function(context) { // 首先要获取调用call的函数，用  
this可以获取 context.fn = this; context.fn(); delete context.fn; }
```

这里的this怎么指向的直接就是函数

按照作者在这篇中讲的去理解，这里的this 不应该是 Function.prototype 吗

有没有大佬解答一下 ~~~

this是在运行时计算绑定的，Function.prototype.call2是给函数的原型上添加了call2的方法，但是实际调用的时候，例如：foo.call2(obj)，它的MemberExpression是foo.call2，根据this那篇知道this就是foo，所以在call2定义的时候，context.fn = this 就相当于 obj.fn = foo 了



1



daomingQian commented on May 29, 2021 • edited by mqyqingfeng ▾



```
// 手写call  
Function.prototype.myCall = function(obj){  
  obj = obj || window; //先判断是不是一个false值 如果是obj为window;  
  if(!(obj instanceof Object)){ //再判断obj是不是一个非对象的真值 例如:数组 字符串等等  
    obj = {};  
  }  
  var arges = []  
  for(var i=1, len = arguments.length; i<len; i++){  
    arges.push('arguments['+i+']');  
  }  
  obj.fn = this;  
  var result = eval('obj.fn('+arges+')');  
}
```

```
    delete obj.fn;
    return result;
}
console.log(getName.myCall(obj, 18, '男'));
```



szmxx commented on Jun 3, 2021 • edited by mqyqingfeng ▾

...

```
// 函数返回值
Function.prototype.call2 = function(context){
    context = toObject(context);
    const args = [].slice.call(arguments, 1);
    context.fn = this;
    const val = context.fn(...args);
    delete context.fn;
    return val;
}
// 处理null值和undefined值, 以及基本类型
function toObject(val){
    const type = typeof val;
    let result = val;
    switch(type){
        case "number":
        case "boolean":
        case "string":
            result = Object(val);
            break;
        default:
            result = result || window;
    }
    return result;
}
```



szmxx commented on Jun 3, 2021 • edited by mqyqingfeng ▾



```
Function.prototype.apply2 = function(context, args){
    context = toObject(context);
    // 类数组和数组
    args = [].slice.apply(args);
    context.fn = this;
    const val = context.fn(...args);
    delete context.fn;
    return val;
}
// 处理null值和undefined值, 以及基本类型
function toObject(val){
    const type = typeof val;
    let result = val;
    switch(type){
        case "number":
        case "boolean":
        case "string":
            result = Object(val);
            break;
        default:
            result = result || window;
    }
    return result;
}
```



lurenacm mentioned this issue on Jun 20, 2021

面试 | call, apply, bind的模拟实现和经典面试题 lurenacm/againJS#13

Open



gncag commented on Jul 20, 2021 • edited by mqyqingreng ▾

...

@fantasy123

```
arr.push('arguments['+i+']');
```

请问这里为什么是一个拼接操作呢？

哈哈 这里面我也啰嗦一下, 一开始没反应过来, 就像作者所说的, 主要是 `eval(string)` 的最终目的是为了拼出一个参数字符串并最终让它可执行, 要使用 `arguments[i]` 也可以 但是要保证最后是字符串 如 case1, 如果直接使用的话(除非参数是数字) 否则就是case2了

```
for(var i = 1, len = arguments.length; i < len; i++) {  
  //case1, 对于例子 bar.call2(foo, 'kevin', 18); 下面语句拿到的结果是 bar('kevin',18)  
  args.push('arguments[' + i + ']');  
  // args.push('' + arguments[i] + '');  
  // args.push('\'' + arguments[i] + '\');  
  // args.push('"' + arguments[i] + '"');  
  // args.push("`" + arguments[i] + "`");  
  
  //case2, 对于例子 bar.call2(foo, 'kevin', 18); 下面语句拿到的结果是 bar(kevin,18) ,  
  // 而对于例子 bar.call2(foo, 666, 18); 下面语句拿到的结果是 bar(666,18), 这种情况下自然  
  // args.push(arguments[i]);  
}  
eval('context.fn(' + args + ')');
```



guobaogang commented on Jul 29, 2021 • edited by mqyqingfeng ▾

...

`context.fn = this;` 这里似乎漏掉了一个很关键的问题, 如果context本来就有fn这个成员怎么办。这里只能给一个原来不存在的名字

```
var id = 0;
while ( context[ id ] ) {
    id ++;
}
context[ id ] = this;
```

不过这个方法似乎有点傻

兄弟，可以用Symbol，

```
var fn = Symbol()
context[fn] = this;
context[fn]();
delete context[fn];
```

不过Symbol也是es6的



hengistchan commented on Jul 29, 2021

...

context.fn = this; 这里似乎漏掉了一个很关键的问题，如果context本来就有fn这个成员怎么办。这里只能给一个原来不存在的名字

```
var id = 0;
while ( context[ id ] ) {
    id ++;
}
context[ id ] = this;
```

不过这个方法似乎有点傻

兄弟，可以用Symbol，
var fn = Symbol()
context[fn] = this;
contextfn;
delete context[fn];
不过Symbol也是es6的

还可以用Object.hasOwnProperty检查一下有没有成员，有得话就先保存下来，弄完后再赋值回去



windschaser commented on Aug 16, 2021 • edited by mqyqingfeng ▾

...

```
Function.prototype.call = function (target, ...args) {  
  if (target == null) target = globalThis;  
  if (typeof target !== 'object') target = Object(target);  
  const symbol = Symbol();  
  target[symbol] = this;  
  try {  
    return target[symbol](...args);  
  } finally {  
    delete target[symbol];  
  }  
}
```

一个相对完善的答案



Dragon-chen777 commented on Sep 1, 2021 • edited by mqyqingfeng ▾

...

apply的实现，再补充一个，就是当arr不是数组而是直接传参时的处理

```

Function.prototype.call2 = function (context, arr) {
  context = Object(context) || window
  context.fn = this;

  var res
  var args = []
  if (arr instanceof Array) {
    for (let i = 0, len = arr.length; i < len; i++) {
      args.push('arr[' + i + ']')
    }
    res = eval('context.fn(' + args + ')')
  } else {
    for (let i = 1, len = arguments.length; i < len; i++) {
      args.push('arguments[' + i + ']')
    }
    res = eval('context.fn(' + args + ')')
  }

  delete context.fn
  return res
}

```



a572251465 commented on Oct 19, 2021 • edited by mqyqingfeng ▾

...

大佬，思考很全面，但是我感觉有个地方不严谨，虽然我的答案不是100%严谨，但是能最大可能避免问题

```

Function.prototype.call1 = function (context) {
  // 1. 修改类型的方法
  var self = this
  // 2. 收集的参数
  var params = []
  // 3. 兼容null 如果是null的话 this指向window

```

```
context = context || window
// 4. 尽最大的可能性避免key重复 从而覆盖
var funName = 'fn' + +new Date()

// 5. 收集参数 从下标1开始
for (var i = 1; i < arguments.length; i++) {
  params.push('arguments[' + i + ']')
}
// 6. 赋值临时属性
context[funName] = self
// 7. 通过eval 执行方法
var result = eval('context[funName](' + params + ')')
// 8. 删除临时添加的属性
delete context[funName]
return result
}
```

大佬，在答案中使用了fn, 比如说全局中fn已经存在，不仅会覆盖，而且最后会删除。但是最好的办法是使用Symbol, 但是为了满足es6之前的语法，我感觉现在这种就能最大可能性进行避免



whatwg6 commented on Nov 30, 2021

...

21 年看这 17 年的写法有点难受啊



Enochzzz mentioned this issue on Dec 10, 2021

【JS】手写call Enochzzz/notes#4

Open



daomingQian commented on Dec 31, 2021

...

我发现这2个手写函数 如果第一个参数 传普通类型和null 跟原call和apply函数或多或少有些不同 自己改进了一下.可以分享一下

```
Function.prototype.myCall = function () {  
  var self = arguments[0] ? Object(arguments[0]) : window  
  var rest = []  
  for (let i = 1; i < arguments.length; i++) {  
    rest.push('arguments[' + i + ']')  
  }  
  self.fn = this  
  var result = eval('self.fn(' + rest + ')')  
  delete self.fn  
  return result  
}  
Function.prototype.myApply = function (context, rest) {  
  var self = context ? Object(context) : window  
  self.fn = this  
  var result  
  if (rest) {  
    var argus = []  
    for (let i = 0; i < rest.length; i++) {  
      argus.push('rest[' + i + ']')  
    }  
    result = eval('self.fn(' + argus + ')')  
  } else {  
    result = self.fn()  
  }  
  delete self.fn  
  return result  
}
```



✉ **imloren** commented on Dec 31, 2021

...

这是来自QQ邮箱的假期自动回复邮件。

您好，我最近正在休假中，无法亲自回复您的邮件。我将在假期结束后，尽快给您回复。



hzzzzzzq commented on Jan 13 • edited ▾

...

我也来评论了，根据学习，写了一版 call，可以求大神帮我看看不。

```
Function.prototype.myCall = function (context) {  
  if (typeof this !== 'function') {  
    console.log('只有函数可以调用 myCall');  
    return;  
  }  
  context = context || window;  
  context.fn = this;  
  argus = [...arguments].slice(1); // 获取参数  
  const result = context.fn(...argus);  
  delete context.fn;  
  return result;  
};
```



Repository owner deleted a comment from **shenghuitian** on Jan 26



Repository owner deleted a comment from **wenwen1995** on Jan 26



Repository owner deleted a comment from **crystalYY** on Jan 26



 **mqqyqingfeng** mentioned this issue on Jan 26

apply模拟实现疑问 #249

Open



TTTJH commented 2 minutes ago



我问疑问 `// 第一版 Function.prototype.call2 = function(context) { // 首先要获取调用call的函数, 用this可以获取 context.fn = this; context.fn(); delete context.fn; }`

这里的这个怎么指向的直接就是函数

作者在这篇文章中讲的去理解, 这里的这个不应该是 `Function.prototype` 吗?

有没有大佬解答一下~~~

编写的call2函数是作为目标函数（需要被修改this的函数）的方法调用的，我们知道当函数作为方法调用时，该函数内部this指向就是其调用者，在这里call2的调用者就是 那个需要被修改this的函数，个人理解👁



 **crystalYY** commented 2 minutes ago



您好，您的邮件我已收到，会尽快回复！樊



✉ **wenwen1995** commented 2 minutes ago



您好，您的邮件我已收到，如果未能及时回复敬请谅解！



✉ **shenghuitian** commented 2 minutes ago



您好，您发给我的邮件已收到。

[Sign up for free](#) to join this conversation on **GitHub**. Already have an account? [Sign in to comment](#)

[Terms](#) [Privacy](#) [Security](#) [Status](#) [Docs](#) [Contact GitHub](#) [Pricing](#) [API](#) [Training](#) [Blog](#) [About](#)



© 2022 GitHub, Inc.