JavaScript深入之bind的模拟实现

冴羽 Lv6

2017年05月03日 02:08 · 阅读 21501


```
Function.prototype.bind2 = function(obj){
    let args = [];
    for(let i = 1; i < arguments.length; i++){
        args.push(arguments[i]);
    }
    let that = this;
    function fun(){
        let anotherArgs = [];
        for(let i = 0; i < arguments.length; i++){
            anotherArgs.push(arguments[i]);
        }
        if(this instanceof that){
            let newObj = {};
            newObj.fn = that;
            let result = newObj.fn();
            delete newObj.fn;
            return newObj;
        }else{
            obj.fn = that;
            return eval("obj.fn("+args.concat(anotherArgs)+")")
;
        }
    }
    fun.prototype = this.prototype;
    return fun;
}

let obj = {
    name:"123",
}

function outter(a,b,c){
    this.name = ":)";
    this.age = 18;
}

const newFun = outter.bind2(obj,1,2,);
const result = new newFun(3);
console.log(result);
```

> JavaScript深入系列第十一篇，通过bind函数的模拟实现，带大家真正了解bind的特性

bind

一句话介绍 bind:

> bind() 方法会创建一个新函数。当这个新函数被调用时，bind() 的第一个参数将作为它运行时的 this，之后的一序列参数将会在传递的实参前传入作为它的参数。(来自于 MDN)

由此我们可以首先得出 bind 函数的两个特点：

1. 返回一个函数
2. 可以传入参数

返回函数的模拟实现

从第一个特点开始，我们举个例子：

复制代码

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

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

```
var bindFoo = bar.bind(foo);

bindFoo(); // 1
```

关于指定 this 的指向，我们可以使用 call 或者 apply 实现，关于 call 和 apply 的模拟实现，可以查看《JavaScript深入之call和apply的模拟实现》。我们来写第一版的代码：

复制代码

```
// 第一版
Function.prototype.bind2 = function (context) {
    var self = this;
    return function () {
        self.apply(context);
    }

}
```

传参的模拟实现

接下来看第二点，可以传入参数。这个就有点让人费解了，我在 bind 的时候，是否可以传参呢？我在执行 bind 返回的函数的时候，可不可以传参呢？让我们看个例子：

复制代码

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

```
    console.log(this.value);
    console.log(name);
    console.log(age);

}

var bindFoo = bar.bind(foo, 'daisy');
bindFoo('18');
// 1
// daisy
// 18
```

函数需要传 name 和 age 两个参数，竟然还可以在 bind 的时候，只传一个 name，在执行返回的函数的时候，再传另一个参数 age!

这可咋办？不急，我们用 arguments 进行处理：

复制代码

```
// 第二版
Function.prototype.bind2 = function (context) {

    var self = this;
    // 获取bind2函数从第二个参数到最后一个参数
    var args = Array.prototype.slice.call(arguments, 1);

    return function () {
        // 这个时候的arguments是指bind返回的函数传入的参数
        var bindArgs = Array.prototype.slice.call(arguments);
        self.apply(context, args.concat(bindArgs));
    }
```

构造函数效果的模拟实现

完成了这两点，最难的部分到啦！因为 bind 还有一个特点，就是

> 一个绑定函数也能使用new操作符创建对象：这种行为就像把原函数当成构造器。提供的 this 值被忽略，同时调用时的参数被提供给模拟函数。

也就是说当 bind 返回的函数作为构造函数的时候，bind 时指定的 this 值会失效，但传入的参数依然生效。举个例子：

复制代码

```
var value = 2;

var foo = {
    value: 1
};

function bar(name, age) {
    this.habit = 'shopping';
    console.log(this.value);
    console.log(name);
    console.log(age);
}

bar.prototype.friend = 'kevin';
```

```
var obj = new bindFoo('18');
// undefined
// daisy
// 18
console.log(obj.habit);
console.log(obj.friend);
// shopping
// kevin
```

注意：尽管在全局和 foo 中都声明了 value 值，最后依然返回了 undefind，说明绑定的 this 失效了，如果大家了解 new 的模拟实现，就会知道这个时候的 this 已经指向了 obj。

(哈哈，我这是为我的下一篇文章《JavaScript深入系列之new的模拟实现》打广告)。

所以我们可以通过修改返回的函数的原型来实现，让我们写一下：

复制代码

```
// 第三版
Function.prototype.bind2 = function (context) {
    var self = this;
    var args = Array.prototype.slice.call(arguments, 1);

    var fbound = function () {

        var bindArgs = Array.prototype.slice.call(arguments);
        // 当作为构造函数时，this 指向实例，self 指向绑定函数，因为下面一句 `fbound.prototype = this.prototype;`，已经修改了 fbound.prototype 为 绑定
        // 当作为普通函数时，this 指向 window，self 指向绑定函数，此时结果为 false，当结果为 false 的时候，this 指向绑定的 context。
```

```
    // 修改返回函数的 prototype 为绑定函数的 prototype，实例就可以继承函数的原型中的值
    fbound.prototype = this.prototype;
    return fbound;
}
```

如果对原型链稍有困惑，可以查看《JavaScript深入之从原型到原型链》。

构造函数效果的优化实现

但是在这个写法中，我们直接将 fbound.prototype = this.prototype，我们直接修改 fbound.prototype 的时候，也会直接修改函数的 prototype。这个时候，我们可以通过一个空函数来进行中转：

复制代码

```
// 第四版
Function.prototype.bind2 = function (context) {

    var self = this;
    var args = Array.prototype.slice.call(arguments, 1);

    var fNOP = function () {};

    var fbound = function () {
        var bindArgs = Array.prototype.slice.call(arguments);
        self.apply(this instanceof self ? this : context, args.concat(bindArgs));
    }
    fNOP.prototype = this.prototype;
```

```
}
```

到此为止，大的问题都已经解决，给自己一个赞！o(￣▽￣)d

三个小问题

接下来处理些小问题:

1.apply 这段代码跟 MDN 上的稍有不同

在 MDN 中文版讲 bind 的模拟实现时，apply 这里的代码是：

复制代码

```
self.apply(this instanceof self ? this : context || this, args.concat(bindArgs))
```

多了一个关于 context 是否存在的判断，然而这个是错误的！

举个例子：

复制代码

```
var value = 2;
var foo = {
    value: 1,
```

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

foo.bar() // 2
```

以上代码正常情况下会打印 2，如果换成了 context || this，这段代码就会打印 1！

所以这里不应该进行 context 的判断，大家查看 MDN 同样内容的英文版，就不存在这个判断！

2.调用 bind 的不是函数咋办?

不行，我们要报错！

复制代码

```
if (typeof this !== "function") {
  throw new Error("Function.prototype.bind - what is trying to be bound is not callable");
}
```

3.我要在线上用

那别忘了做个兼容：

复制代码

```
Function.prototype.bind = Function.prototype.bind || function () {
    ……
```

当然最好是用es5-shim啦。

最终代码

所以最最后的代码就是：

复制代码

```
Function.prototype.bind2 = function (context) {

    if (typeof this !== "function") {
      throw new Error("Function.prototype.bind - what is trying to be bound is not callable");
    }

    var self = this;
    var args = Array.prototype.slice.call(arguments, 1);
    var fNOP = function () {};

    var fbound = function () {
        self.apply(this instanceof self ? this : context, args.concat(Array.prototype.slice.call(arguments)));
    }

    fNOP.prototype = this.prototype;
    fbound.prototype = new fNOP();

    return fbound;

}
```

下一篇文章

《JavaScript深入系列之new的模拟实现》

相关链接

《JavaScript深入之从原型到原型链》

《JavaScript深入之call和apply的模拟实现》

《JavaScript深入系列之new的模拟实现》

深入系列

JavaScript深入系列目录地址：github.com/mqyqingfeng...。

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。

分类： 前端 标签： JavaScript 前端

文章被收录于专栏：

JavaScript 深入系列

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...

关注专栏

评论

260　39　收藏

热门评论

xik_why_fe Lv1 前端 @ ByteDance 2年前

深夜意难平！今天面试我的百度面试官，记成了fBound没有返回值的错误版本，说我写错了！

3 1

小天yao爱吃糖糖... 3年前

但是在这个写法中，我们直接将 fbound.prototype = this.prototype，我们直接修改 fbound.prototype 的时候，也会直接修改函数的 prototype。emmmm,啥意思？没懂。是说直接继承的话，fbound从函数原型链上继承到的东西拿不到了吗？但是this也是一个函数啊，感觉这个问题不存在耶，求大神解答

点赞 2

冴羽 Lv6 （作者） 3年前

你跳到Github上看这一篇文章，底下有评论解释了这个问题

点赞 回复

小天yao爱吃糖糖糖... 回复 冴羽 3年前

get，谢谢！

> “你跳到Github上看这一篇文章，底下有评论解释了这个问题”

260 39 收藏

全部评论 39

最新 最热

本淡 Lv2 前端工程师 @ 字节跳动 8天前

老师，感觉原型这块有一点问题function testFn () {
this.name = '121'
// console.log(this.name);
}

testFn.prototype.getName = function(){...

展开

点赞 回复

NeoYu Lv2 前端开发工程师 @ 皮 5月前

赞一个 👍😲

点赞 回复

z0aiyx Lv1 1年前

bind不是直接绑定在Function.prototype上面吗，为什么还要判断调用bind的是不是函数呢，不是函数不是就没bind这个方法吗

4 回复

niayyy Lv2 前端 @ @ 2年前

260 39 收藏

return self.apply(...)
}
应该在 self.apply() 前面加上 return，不然不会有返回值

1 1

Sevennn72656 1年前

赞同

点赞 回复

LINGLONG Lv2 非前端 @ 自由职业 2年前

`我们直接将 fBound.prototype = this.prototype，我们直接修改 fBound.prototype 的时候，也会直接修改绑定函数的 prototype` 请问绑定函数的prototype是如何被修改的呢?

点赞 1

LINGLONG Lv2 2年前

我去github上看啦评论弄明白啦，thanks!

点赞 回复

Chang1ng 前端工程师 @ 欢聚时代 2年前

写的很好，赞一个

点赞 回复

260 39 收藏

我命油我 Lv1 前端开发 @ 腾讯CDC前端 2年前

最后一个代码的 fbound 函数应该要 return self.apply... 不然返回值就不对了

1 1

duola8789 Lv2 前端开发 2年前

有个疑问请教一下，在判断是否是new调用的时候，是否可以直接使用this instanceof fBound判断? 我认为在new调用的时候fBound中的this的原型属性会了解到构造函数fBound，而非new调用则不会
这样的话下方的手动构造原型链的两行代码我认为都可以省略
还请指教

点赞 1

xik_why_fe Lv1 前端 @ ByteDance 2年前

深夜意难平！今天面试我的百度面试官，记成了fBound没有返回值的错误版本，说我写错了！

3 1

田先僧 Lv1 web前端开发工程师 2年前

这一系列和木易杨说的一样哦。。。谁是原创

点赞 1

dadadadada 2年前

"但是在这个写法中，我们直接将 fbound.prototype = this.prototype，我们直接修改 fbound.prototype 的时候，也会直接修改函数的 prototype。" 博主你好，我这段话不是很懂是什么意思，您能解释下吗？ 感谢

260 39 收藏

思远在掘金 2年前

明白了 需要好好琢磨琢磨 😄

1 回复

senga Lv1 菜鸟前端 3年前

使用new构造函数时，返回结果undefined，这时的this指向的是bar吧?

点赞 回复

zlfHorror 前端 @ 焦点科技 3年前

我想问一个问题，就是，var fNOP = function () {}; fNOP.prototype = this.prototype;
fbound.prototype = new fNOP(); 这里使用一个中间函数替换原形，怕后续修改原形操作污染原函数的原形，那么这里我不用中间函数替换，直接用一个原函数的实例去置换新函数的原形对象不也是可以的吗？ fbound.prototype = new self (); 这样不也是完成了继承，并且原形不会污染，有什么很大的区别吗，这个好像是两种继承方式，这里为什么用这种呢?

点赞 1

风儿吹呀吹 Lv2 2年前

如果使用new self的话，可能会带有self函数创建的实例属性，所以这里最好是使用一个空函数

1 回复

plqqlp 前端 @ 腾讯 3年前

this instanceof self => this instanceof fNOP

260 39 收藏

小天yao爱吃糖糖... 3年前

但是在这个写法中，我们直接将 fbound.prototype = this.prototype，我们直接修改 fbound.prototype 的时候，也会直接修改函数的 prototype。emmmm,啥意思？没懂。是说直接继承的话，fbound从函数原型链上继承到的东西拿不到了吗？但是this也是一个函数啊，感觉这个问题不存在耶，求大神解答

点赞 2

冴羽 Lv6 （作者） 3年前

你跳到Github上看这一篇文章，底下有评论解释了这个问题

点赞 回复

小天yao爱吃糖糖糖... 回复 冴羽 3年前

get，谢谢！

> “你跳到Github上看这一篇文章，底下有评论解释了这个问题”

点赞 回复

酱汁鱼 前端 3年前

哎呦，感冒了，回去翻翻书

1 回复

MissCuriosity Front-End Sonftware En... 4年前

260 39 收藏

点赞 1

lz-lee 4年前

var fNOP = function () {};
var fbound = function () {
self.apply(this instanceof self ? this : context, args.concat(Array.prototype.slice.call(arguments)));
}
当使用空函数中转时，this instanceof self 是不是应该改成 this instanceof fNOP?

点赞 2

冴羽 Lv6 （作者） 4年前

因为 fNOP.prototype = this.prototype 的缘故，所以使用 self 也并没有问题..... 不过我查了 MDN，使用的确实是 fNOP ，自己模拟实现的时候没有注意到这一点，非常感谢指出~ o(￣▽￣)d

点赞 回复

Tia_Brother 3年前

可能出现 Function has non-object prototype 'undefined' in instanceof check 的情况 self.prototype 不存在

点赞 回复

嗨起来_小马哥 4年前

但是在这个写法中，我们直接将 fbound.prototype = this.prototype，我们直接修改 fbound.prototype 的时候，也会直接修改函数的 prototype。这个时候，我们可以通过一个空函数来进行中转

冴羽 Lv6 （作者） 4年前

今天太晚了，明天给你写解释哈

点赞 回复

原来可以改名字 4年前

我个人感觉因为原型对象是引用的，如果直接直接将 fbound.prototype = this.prototype 的话，一旦修改了发 bound.prototype，实例（this）的原型对象会跟着改变，这不是我们需要的

点赞 回复

查看全部 39 条回复

相关推荐

摸鱼的春哥 | 2月前 | 前端 · JavaScript

2022，前端的天🌧要怎么变?

6.9w 683 258

前端小白的逆袭 | 1年前 | JavaScript

bind的原理和bind的实现

260 39 收藏

别动我的糖醋排骨 | 5月前 | JavaScript

JS 模拟实现 call、apply、bind 方法

124 2 评论

HighClassLickDog | 4月前 | 前端 · JavaScript

因为懒，我把公司的后管定制成了低代码中台

6.4w 459 171

土豪码农 | 2年前 | 前端

面试感悟,手写bind,apply,call

1.7w 265 32

火星飞鸟 | 6月前 | 前端 · JavaScript

JavaScript 实现call、apply、bind函数

661 10 评论

我是一个前端 | 3年前 | JavaScript

手写call、apply、bind实现及详解

2.3w 195 23

大帅老猿 | 10月前 | 前端 · JavaScript

产品经理：你能不能用div给我画条龙?

260 39 收藏

前端胖头鱼 | 5月前 | JavaScript · Vue.js · 前端

就因为JSON.stringify，我的年终奖差点打水漂了

7.6w 1067 233

Sunshine_Lin | 5月前 | 前端 · JavaScript · ECMAScript 6

「万字总结」熬夜总结50个JS的高级知识点，全都会你就是神！！！

7.5w 2181 107

OBKoro1 | 2年前 | JavaScript · 前端

js基础-面试官想知道你有多理解call,apply,bind？[不看后悔系列]

3.0w 670 81

CUGGZ | 5月前 | 前端 · JavaScript · 程序员

33个非常实用的JavaScript一行代码，建议收藏！

7.3w 1730 59

sunshine小小倩 | 4年前 | JavaScript · 前端

this、apply、call、bind

12.2w 3148 251

前端阿飞 | 4月前 | 前端 · JavaScript

10个常见的前端手写功能，你全都会吗？

260 39 收藏

浪里行舟 | 3年前 | JavaScript · 前端

九种跨域方式实现原理（完整版）

13.7w 2615 107

非优秀程序员 | 4月前 | 前端 · JavaScript

如何用 CSS 中写出超级美丽的阴影效果

31.7w 236 12

大帅老猿 | 9月前 | 前端 · JavaScript · HTML

2天赚了4个W，手把手教你用Threejs搭建一个Web3D汽车展厅！

5.9w 1421 240

MichaelHong | 1年前 | JavaScript

JavaScript面试之手写call、apply、bind

2992 36 19

yeyan1996 | 2年前 | JavaScript

字节跳动面试官：请你实现一个大文件上传和断点续传

21.6w 5050 533

LinYIYI | 1年前 | JavaScript

深入了解 call、apply、bind 和模拟实现

260 39 收藏