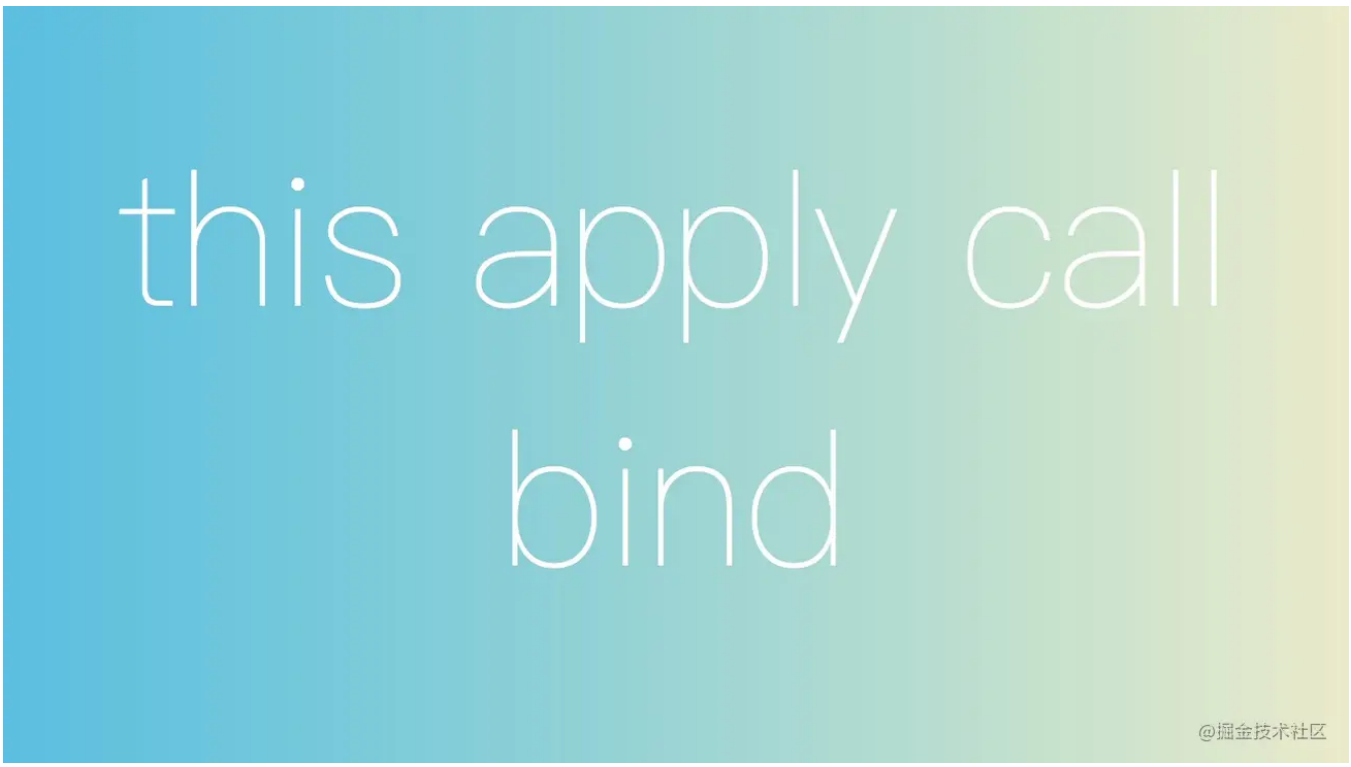
sunshine小小倩 Lv6
2017年09月18日 阅读 102260



this、apply、call、bind

这又是一个面试经典问题~/(ㄒoㄒ)/~~也是 ES5中众多坑中的一个，在 ES6 中可能会极大避免 this 产生的错误，但是为了一些老代码的维护，最好还是了解一下 this 的指向和 call、apply、bind 三者的区别。

> 本文首发于我的个人网站：cherryblog.site/

this 的指向

在 ES5 中，其实 this 的指向，始终坚持一个原理：**this 永远指向最后调用它的那个对象**，来，跟着我朗读三遍：**this 永远指向最后调用它的那个对象，this 永远指向最后调用它的那个对象，this 永远指向最后调用它的那个对象**。记住这句话，this 你已经了解一半了。

复制代码

```
var name = "windowsName";
function a() {
    var name = "Cherry";

    console.log(this.name);          // windowsName

    console.log("inner:" + this);    // inner: Window
}
a();
console.log("outer:" + this)         // outer: Window
```

这个相信大家都知道为什么 log 的是 windowsName，因为根据刚刚的那句话“**this 永远指向最后调用它的那个对象**”，我们看最后调用 `a` 的地方 `a();`，前面没有调用的对象那么就是全局对象 window，这就相当于是 `window.a()`；注意，这里我们没有使用严格模式，如果使用严格模式的话，全局对象就是 `undefined`，那么就会报错 `Uncaught TypeError: Cannot read property 'name' of undefined`。

再看下这个例子：
例 2：

复制代码

```
var name = "windowsName";
var a = {
    name: "Cherry",
    fn : function () {
        console.log(this.name);      // Cherry
    }
}
a.fn();
```

在这个例子中，函数 fn 是对象 a 调用的，所以打印的值就是 a 中的 name 的值。是不是有一点清晰了呢~

我们做一个小小的改动：
例 3：

复制代码

```
var name = "windowsName";
    var a = {
        name: "Cherry",
        fn : function () {
```

```
window.a.fn();
```

这里打印 Cherry 的原因也是因为刚刚那句话“**this 永远指向最后调用它的那个对象**”，最后调用它的对象仍然是对象 a。

我们再来看一下这个例子：
例 4：

复制代码

```
var name = "windowsName";
var a = {
    // name: "Cherry",
    fn : function () {
        console.log(this.name);      // undefined
    }
}
window.a.fn();
```

这里为什么会打印 `undefined` 呢？这是因为正如刚刚所描述的那样，调用 fn 的是 a 对象，也就是说 fn 的内部的 this 是对象 a，而对象 a 中并没有对 name 进行定义，所以 log 的 `this.name` 的值是 `undefined` 。

这个例子还是说明了：**this 永远指向最后调用它的那个对象**，因为最后调用 fn 的对象是 a，所以就算 a 中没有 name 这个属性，也不会继续向上一个对象寻找 `this.name` ，而是直接输出 `undefined` 。

再来看一个比较坑的例子：
例 5：

复制代码

```
var name = "windowsName";
var a = {
    name : null,
    // name: "Cherry",
    fn : function () {
        console.log(this.name);      // windowsName
    }
}

var f = a.fn;
f();
```

用，所以 `fn()` 最后仍然是被 window 调用的。所以 this 指向的也就是 window。

由以上五个例子我们可以看出，this 的指向并不是在创建的时候就可以确定的，在 es5 中，永远是**this 永远指向最后调用它的那个对象**。

再来看一个例子：
例 6：

复制代码

```
var name = "windowsName";

function fn() {
    var name = 'Cherry';
    innerFunction();
    function innerFunction() {
        console.log(this.name);      // windowsName
    }
}

fn()
```

读到现在了应该能够理解这是为什么了吧(o゜▽゜)o。

怎么改变 this 的指向

改变 this 的指向我总结有以下几种方法：

- 使用 ES6 的箭头函数
- 在函数内部使用 `_this = this`
- 使用 `apply`、`call`、`bind`
- new 实例化一个对象

例 7：

复制代码

```
var name = "windowsName";

var a = {
    name : "Cherry",

    func1: function () {
        console.log(this.name)
```

```
        setTimeout(  function () {
            this.func1()
        },100);
    }

};

a.func2()     // this.func1 is not a function
```

在不使用箭头函数的情况下，是会报错的，因为最后调用 `setTimeout` 的对象是 window，但是在 window 中并没有 func1 函数。

我们在改变 this 指向这一节将把这个例子作为 demo 进行改造。

箭头函数

众所周知，ES6 的箭头函数是可以避免 ES5 中使用 this 的坑的。**箭头函数的 this 始终指向函数定义时的 this，而非执行时。**，箭头函数需要记着这句话：“箭头函数中没有 this 绑定，必须通过查找作用域链来决定其值，如果箭头函数被非箭头函数包含，则 this 绑定的是最近一层非箭头函数的 this，否则，this 为 undefined”。

例 8：

复制代码

```
var name = "windowsName";

var a = {
    name : "Cherry",

    func1: function () {
        console.log(this.name)
    },

    func2: function () {
        setTimeout( () => {
            this.func1()
        },100);
    }

};

a.func2()     // Cherry
```

如果不使用 ES6，那么这种方式应该是最简单的不会出错的方式了，我们是先将调用这个函数的对象保存在变量 `_this` 中，然后在函数中都使用这个 `_this`，这样 `_this` 就不会改变了。
例 9：

复制代码

```
var name = "windowsName";

var a = {

    name : "Cherry",

    func1: function () {
        console.log(this.name)
    },

    func2: function () {
        var _this = this;
        setTimeout( function() {
            _this.func1()
        },100);
    }

};

a.func2()       // Cherry
```

这个例子中，在 func2 中，首先设置 `var _this = this;`，这里的 `this` 是调用 `func2` 的对象 a，为了防止在 `func2` 中的 setTimeout 被 window 调用而导致的在 setTimeout 中的 this 为 window。我们将 `this(指向变量 a)` 赋值给一个变量 `_this`，这样，在 `func2` 中我们使用 `_this` 就是指向对象 a 了。

使用 apply、call、bind

使用 apply、call、bind 函数也是可以改变 this 的指向的，原理稍后再讲，我们先来看一下是怎么实现的：

使用 apply

例 10：

```
    func1: function () {
        console.log(this.name)
    },

    func2: function () {
        setTimeout(  function () {
            this.func1()
        }.apply(a),100);
    }

};

a.func2()            // Cherry
```

使用 call

例 11：

复制代码

```
var a = {
    name : "Cherry",

    func1: function () {
        console.log(this.name)
    },

    func2: function () {
        setTimeout(  function () {
            this.func1()
        }.call(a),100);
    }

};

a.func2()            // Cherry
```

使用 bind

例 12：

```
    func1: function () {
        console.log(this.name)
    },

    func2: function () {
        setTimeout(  function () {
            this.func1()
        }.bind(a)(),100);
    }

};

a.func2()            // Cherry
```

apply、call、bind 区别

刚刚我们已经介绍了 apply、call、bind 都是可以改变 this 的指向的，但是这三个函数稍有不同。

在 MDN 中定义 apply 如下；

> apply() 方法调用一个函数, 其具有一个指定的this值，以及作为一个数组（或类似数组的对象）提供的参数

语法：

> fun.apply(thisArg, [argsArray])

- thisArg：在 fun 函数运行时指定的 this 值。需要注意的是，指定的 this 值并不一定是该函数执行时真正的 this 值，如果这个函数处于非严格模式下，则指定为 null 或 undefined 时会自动指向全局对象（浏览器中就是window对象），同时值为原始值（数字，字符串，布尔值）的 this 会指向该原始值的自动包装对象。
- argsArray：一个数组或者类数组对象，其中的数组元素将作为单独的参数传给 fun 函数。如果该参数的值为null 或 undefined，则表示不需要传入任何参数。从ECMAScript 5 开始可以使用类数组对象。浏览器兼容性请参阅本文底部内容。

apply 和 call 的区别

call 的语法为：

复制代码

```
fun.call(thisArg[, arg1[, arg2[, ...]]])
```

所以 apply 和 call 的区别是 call 方法接受的是若干个参数列表，而 apply 接收的是一个包含多个参数的数组。

例 13：

复制代码

```
var a ={
    name : "Cherry",
    fn : function (a,b) {
        console.log( a + b)
    }
}

var b = a.fn;
b.apply(a,[1,2])     // 3
```

例 14：

复制代码

```
var a ={
    name : "Cherry",
    fn : function (a,b) {
        console.log( a + b)
    }
}

var b = a.fn;
b.call(a,1,2)       // 3
```

bind 和 apply、call 区别

我们先来将刚刚的例子使用 bind 试一下

复制代码

```
var a ={
    name : "Cherry",
    fn : function (a,b) {
        console.log( a + b)
```

```
var b = a.fn;
b.bind(a,1,2)
```

我们会发现并没有输出，这是为什么呢，我们来看一下 MDN 上的文档说明：

> bind()方法创建一个新的函数, 当被调用时，将其this关键字设置为提供的值，在调用新函数时，在任何提供之前提供一个给定的参数序列。

所以我们可以看出，bind 是创建一个新的函数，我们必须要手动去调用：

复制代码

```
var a ={
    name : "Cherry",
    fn : function (a,b) {
        console.log( a + b)
    }
}

var b = a.fn;
b.bind(a,1,2)()           // 3
```

========== 更新==========

JS 中的函数调用

看到留言说，很多童靴不理解为什么 例 6 的 innerFunction 和 例 7 的 this 是指向 window 的，所以我就来补充一下 JS 中的函数调用。
例 6：

复制代码

```
var name = "windowsName";

function fn() {
    var name = 'Cherry';
    innerFunction();
    function innerFunction() {
        console.log(this.name);      // windowsName
    }
}

fn()
```

复制代码

```
var name = "windowsName";

var a = {
    name : "Cherry",

    func1: function () {
        console.log(this.name)
    },

    func2: function () {
        setTimeout(  function () {
            this.func1()
        },100);
    }

};

a.func2()     // this.func1 is not a function
```

函数调用的方法一共有 4 种

1. 作为一个函数调用
2. 函数作为方法调用
3. 使用构造函数调用函数
4. 作为函数方法调用函数（call、apply）

作为一个函数调用

比如上面的 例 1:
例 1:

复制代码

```
var name = "windowsName";
function a() {
    var name = "Cherry";

    console.log(this.name);          // windowsName

    console.log("inner:" + this);    // inner: Window
}
```

这样一个最简单的函数，不属于任何一个对象，就是一个函数，这样的情况在 JavaScript 的在浏览器中的非严格模式默认是属于全局对象 window 的，在严格模式，就是 undefined。

但这是一个全局的函数，很容易产生命名冲突，所以不建议这样使用。

函数作为方法调用

所以说更多的情况是将函数作为对象的方法使用。比如例 2：
例 2：

复制代码

```
var name = "windowsName";
var a = {
    name: "Cherry",
    fn : function () {
        console.log(this.name);      // Cherry
    }
}
a.fn();
```

这里定义一个对象 `a`，对象 `a` 有一个属性（`name`）和一个方法（`fn`）。

然后对象 `a` 通过 `.` 方法调用了其中的 fn 方法。

然后我们一直记住的那句话“**this 永远指向最后调用它的那个对象**”，所以在 fn 中的 this 就是指向 a 的。

使用构造函数调用函数

> 如果函数调用前使用了 new 关键字, 则是调用了构造函数。
> 这看起来就像创建了新的函数，但实际上 JavaScript 函数是重新创建的对象：

复制代码

```
// 构造函数:
function myFunction(arg1, arg2) {
    this.firstName = arg1;
    this.lastName  = arg2;
}
```

```
a.lastName;                              // 返回 "Cherry"
```

这就有要说另一个面试经典问题：new 的过程了，(ಠ_ಠ)
这里就简单的来看一下 new 的过程吧：
伪代码表示：

复制代码

```
var a = new myFunction("Li","Cherry");

new myFunction{
    var obj = {};
    obj.__proto__ = myFunction.prototype;
    var result = myFunction.call(obj,"Li","Cherry");
    return typeof result === 'obj'? result : obj;
}
```

1. 创建一个空对象 obj;
2. 将新创建的空对象的隐式原型指向其构造函数的显示原型。
3. 使用 call 改变 this 的指向
4. 如果无返回值或者返回一个非对象值，则将 obj 返回作为新对象；如果返回值是一个新对象的话那么直接直接返回该对象。

所以我们可以看到，在 new 的过程中，我们是使用 call 改变了 this 的指向。

作为函数方法调用函数

> 在 JavaScript 中, 函数是对象。
>
> JavaScript 函数有它的属性和方法。
> call() 和 apply() 是预定义的函数方法。 两个方法可用于调用函数，两个方法的第一个参数必须是对象本身
>
> 在 JavaScript 严格模式(strict mode)下, 在调用函数时第一个参数会成为 this 的值， 即使该参数不是一个对象。
> 在 JavaScript 非严格模式(non-strict mode)下, 如果第一个参数的值是 null 或 undefined, 它将使用全局对象替代。

这个时候我们再来看例 6：
例 6：

```
function fn() {
    var name = 'Cherry';
    innerFunction();
    function innerFunction() {
        console.log(this.name);      // windowsName
    }
}

fn()
```

这里的 innerFunction() 的调用是不是属于第一种调用方式：作为一个函数调用（它就是作为一个函数调用的，没有挂载在任何对象上，所以对于没有挂载在任何对象上的函数，在非严格模式下 this 就是指向 window 的）

然后再看一下 例 7：
例 7：

复制代码

```
var name = "windowsName";

var a = {
    name : "Cherry",

    func1: function () {
        console.log(this.name)
    },

    func2: function () {
        setTimeout(  function () {
            this.func1()
        },100 );
    }

};

a.func2()     // this.func1 is not a function
```

这个简单一点的理解可以理解为“**匿名函数的 this 永远指向 window**”，你可以这样想，还是那句话 **this 永远指向最后调用它的那个对象**，那么我们就来找最后调用匿名函数的对象，这就很尴尬了，因为匿名函数名字啊，笑哭，所以我们是没有办法被其他对象调用匿名函数的。所以说 匿名函数的 this 永远指向 window。

函数调用啊，比如例 7 中的 setTimeout。

文章分类 前端 文章标签 JavaScript 前端

sunshine小小倩 Lv6
前端工程师 @ ELEME
获得点赞 19,251 · 获得阅读 719,412

热门评论

新年歌声 4年前

例6 没有看懂，大神讲解下啊。this 永远指向最后调用它的那个对象， 但是调用innerFunction，是在fn的内部啊。

点赞 12

搬砖小达人 Lv1 4年前

你可以理解为没有使用var声明，是个全局变量

点赞 回复

sunshine小... Lv6 （作者） 4年前

那你明白 例 1 的 a 为什么是被 window 调用吗，两个是一个道理

点赞 回复

ZachBird 4年前

还是不太明白例7中为什么setTimeout被window调用。。。例9应该和例8是对照组吧，那例9是不是不用箭头函数比较好呀...

1 8

sunshine小... Lv6 （作者） 4年前

那你明白 例 1 的 a 为什么是被 window 调用吗，两个是一个道理

点赞 回复

搬砖小达人 Lv1 4年前

你可以试着这样理解，setTimeout 也只是一个函数而已，函数必然有可能需要参数，我们把 function 当作一个参数传给 setTimeout 这个函数，就像它需要一个 fun 参数，在传入参数的时候，其实做了个这样的操作 fun = function ，看到没有，这里我们直接把 fun 指向 function 的引用；执行的时候其实是执行了 fun() 所以已经和 a 这个obj 无关了，它是被当作普通函数直接调用的，因此 this 指向全局对象

2 回复

查看更多回复

全部评论（224）

最新 最热

用户8972521205... 3天前

时隔四年，仍然经典啊

点赞 回复

.com君89761 Lv1 13天前

栗子1 没跑出来 没有指向Window

点赞 2

佛系码农 Lv1 12天前

在浏览器上就跑出来了

点赞 回复

.com君89761 Lv1 12天前

这是在node环境下 在浏览器环境下就可以跑出来

点赞 回复

bind 和 apply、call 区别 那里，调用时应该是：b.bind(a,1,2)() 才会打印出结果的

点赞 1

小半丶 20天前

反动ID举报了

点赞 回复

石人一只眼,, 1月前

var name = "windowsName"; var a = { name: "Cherry", fn() { let func = function() { console.log(this.name); // windowsName } func(); } } a.fn(); 请问此时是a调用的fn怎么解释里面的this指向了window

点赞 回复

石人一只眼,, 1月前

例子1如果使用 let 声明变量，则调用不到this.name

点赞 回复

洛琮 Lv1 前端 1月前

全部答对了

点赞 回复

铠甲合体 Lv1 前端 3月前

悟空，悟净，悟能，三藏。

3 2

yogln Lv1 2月前

人才啊~

1 回复

铠甲合体 Lv1 回复 yogln 2月前

招人吗

“人才啊~”

点赞 回复

兮丶 Lv2 前端 3月前

call apply 并没有说清楚他们的this问题～

点赞 回复

定时器内的函数 .call(a) .apply(a) .bind(a)()会立即执行。影响到了定时器的作用。.bind(a)() 改成.bind(a)返回一个函数就行。另外两个有解决方案吗 😂

点赞 回复

---

sevensCode 前端开发 4月前

this 永远指向最后调用它的那个对象

点赞 回复

158968 4月前

例7为啥a.fn（）不是a调用的么，怎么变成全局了呢

点赞 2

> **用户87859...** 4月前
>
> a.func2是全局调用的没错，但是func2里面func1的调用是放在setTimeout里面的，setTimeout可以看成是window全局对象的函数，由全局调用，所以setTimeout函数里面的this在非严格模式下是指向全局对象的
>
> 1 回复

> **lyq162338...** 1月前
>
> 函数作为回调或者参数，根据隐式绑定规则会隐式丢失，使用默认绑定，就是window
>
> 点赞 回复

黎羽晨 前端工程师 4月前

小姐姐好厉害，我老是记不住这个

点赞 回复

---

poyoho Lv1 前端 8月前

悟了

3 回复

五更耗纸 Lv1 10月前

还是希望姐姐可以明确圈定一下匿名函数的范围（指匿名函数this都是window这句话），还有如果可以把“this永远指向最后调用它的对象”改成“除了箭头函数里的this，this永远指向最后调用它的对象”就更严谨了。 👍

3 1

> **用户24532...** 5月前
>
> 都说了在es5中了啊 es5没有箭头函数啊
>
> 点赞 回复

例7 var name = "windowsName"; var a = { name : "Cherry", func1: function () { console.log(this.name) }, func2: function () { setTimeout((() => { this.func1() }), 100) } }; a.func2() // cherry

1 1

fatdoge Lv1 10月前

虽然setTimeout里是一个匿名函数，但是这个匿名函数是箭头函数，他指向的还是a对象。

1 回复

若水666 11月前

call和apply只有传参的不同吗? 😂

点赞 1

小白_bj Lv1 7月前

是的 😃

1 回复

用户4336726955... 11月前

虽然内容很多，但是讲的不好

8 1

用户56846... 3月前

讲的不好那你来讲

点赞 回复

前半 Lv2 11月前

var name = "111"; var a = { name : '222', fn : { name: '333', func : () => { this.name = '4444' console.log(this.name); } } } var f = a.fn.func; f(); // window.f() -> 444 a.fn.func() // 444

1 回复

前半 Lv2 11月前

this 永远指向最后调用它的那个对象!!!! 举个例子， 很明显违背了!! var name = "111"; var a = { name : '222', fn : { name: '333', func: function () { this.name = '4444' console.log(this.name); } } } var f = a.fn.func; // window.f f(); // 444 a.fn.func() // 444

点赞 4

HelloMan 11月前

longwei0821 10月前

都在function里重新赋值了

1 回复

查看更多回复

前端做菜工程师 前端工程师 @ 远光软件... 1年前

我悟了！感谢大佬 😘

1 回复

查看全部 224 条回复

相关推荐

尼克陈 7月前 前端

面不面试的，你都得懂原型和原型链

不要为了面试而去背题，匆匆忙忙的，不仅学不进去，背完了几天后马上会忘记。 你可能会...

1.9w 790 109

MoonBall 6月前 JavaScript

JS 中 this 指向问题

相信我，只要记住本文的 7️⃣ 步口诀，就能彻底掌握 JS 中的 this 指向。 先念口诀：箭头函数、new、bind、apply ...

9084 291 53

蔓蔓雒轩 3年前 JavaScript 前端 Promise

Promise不会？？看这里！！！史上最通俗易懂的Promise！！！

一般来说我们会碰到的回调嵌套都不会很多，一般就一到两级，但是某些情况下，回调嵌套很多时，代码就会非常繁...

4.7w 1101 23

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]

非严格模式下：thisArg指定为null，undefined，fun中的this指向window对象....

2.3w 549 68

我是一个前端 2年前 JavaScript

手写call、apply、bind实现及详解

若川 2年前 JavaScript 面试

面试官问：JS的this指向

面试官出很多考题，基本都会变着方式来考察this指向，看候选人对JS基础知识是否扎实。 读者可以先拉到底部看总...

2.9w 321 29

刘小夕 2年前 JavaScript 前端

嗨，你真的懂this吗?

this关键字是JavaScript中最复杂的机制之一，是一个特别的关键字，被自动定义在所有函数的作用域中，但是相信...

2.5w 777 144

萌m子 1年前 JavaScript

2020面试收获 - js原型及原型链

2020年是特殊的一年，由于疫情原因，大家都窝在家办公。而我则怀着梦想，从天津来到了北京，开启了人生的第一...

1.8w 330 52

圈圈Dei圈 2年前 JavaScript

新手理解 apply 和 call

本文仅适合新手, 我是说很新的那种, 旨在和大家共同了解js的call和apply方法. 笔者水平有限, 文中不免会有疏漏, 大...

1899 71 16

Big shark@LX 1月前 前端 JavaScript 面试

最新的前端大厂面经（详解答案）

Hello 大家好 我是鲨鱼哥 最新的大厂前端面试题出炉啦 助力大家金九银十拿到好offer 大家...

6.3w 2429 193

Jean 3月前 前端

DayNote(JS5-高级)

----------个人学习笔记----------

108 点赞 评论

土豪码农 2年前 前端

面试感悟,手写bind,apply,call

balabala讲了一堆,从http到https到http2,还补充了点http3的东西,巴拉巴拉讲了一堆,信心满满.谁知道一个问题就问...

1.3w 205 25

深入理解JavaScript作用域和作用域链

JavaScript中有一个被称为作用域(Scope)的特性。虽然对于许多新手开发者来说，作用域的概念并不是很容易理解...

1.9w 269 24

用户123 9月前 JavaScript

for循环中定时器打印值的问题?

在学习js的时候，或者面试的时候，会经常碰到这一道经典题目：熟悉这道题目的人立马就可以说出答案：结果是先...

474 11 评论

浪里行舟 3年前 JavaScript 前端

浅拷贝与深拷贝

浅拷贝是创建一个新对象，这个对象有着原始对象属性值的一份精确拷贝。如果属性是基本类型，拷贝的就是基本类...

5.5w 945 77

山间的风 5月前 JavaScript

new， call，apply，bind 原理剖析及手写实现

new， call，apply，bind的手写实现, 从底层原理剖析，让你从根本上明白并轻松地实现！

2091 39 8

OBKoro1 2年前 JavaScript ECMAScript 6

详解箭头函数和普通函数的区别以及箭头函数的注意事项、不适用场景

箭头函数是ES6的API，相信很多人都知道，因为其语法上相对于普通函数更简洁，深受大家...

1.8w 426 40

木易杨说 2年前 GitHub JavaScript 前端

JavaScript常用八种继承方案

更新：在常用七种继承方案的基础之上增加了ES6的类继承，所以现在变成八种啦。 构造函数、原型和实例之间的关...

4.5w 613 56

iconLee 4月前 面试

改变 this 指向、深入理解 call/apply/bind 的原理

前言 在“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...

642 11 3

小只前端攻城狮 4月前 面试 前端

896 20 5