11 | this：从JavaScript执行上下文的视角讲清楚this

2019-08-29 李兵

《浏览器工作原理与实践》 课程介绍 >

讲述：李兵

时长 12:04 大小 13.82M

在上篇文章中，我们讲了词法作用域、作用域链以及闭包，并在最后思考题中留了下面这样一段代码：

复制代码

```
var bar = {
    myName:"time.geekbang.com",
    printName: function () {
        console.log(myName)
    }
}
function foo() {
    let myName = "极客时间"
    return bar.printName
}
let myName = "极客邦"
let _printName = foo()
_printName()
bar.printName()
```

相信你已经知道了，在 printName 函数里面使用的变量 myName 是属于全局作用域下面的，所以最终打印出来的值都是“极客邦”。这是因为 JavaScript 语言的作用域链是由词法作用域决定的，而词法作用域是由代码结构来确定的。

不过按照常理来说，调用bar.printName方法时，该方法内部的变量 myName 应该使用 bar 对象中的，因为它们是一个整体，大多数面向对象语言都是这样设计的，比如我用 C++ 改写了上面那段代码，如下所示：

复制代码

```
#include <iostream>
using namespace std;
class Bar{
    public:
    char* myName;
    Bar(){
      myName = "time.geekbang.com";
    }
    void printName(){
       cout<< myName <<endl;
    }
} bar;

char* myName = "极客邦";
int main() {
  bar.printName();
  return 0;
}
```

在这段 C++ 代码中，我同样调用了 bar 对象中的 printName 方法，最后打印出来的值就是 bar 对象的内部变量 myName 值——“time.geekbang.com”，而并不是最外面定义变量 myName 的值——“极客邦”，所以**在对象内部的方法中使用对象内部的属性是一个非常普遍的需求**。但是 JavaScript 的作用域机制并不支持这一点，基于这个需求，JavaScript 又搞出来另外一套 **this 机制**。

所以，在 JavaScript 中可以使用 this 实现在 printName 函数中访问到 bar 对象的 myName 属性了。具体该怎么操作呢？你可以调整 printName 的代码，如下所示：

复制代码

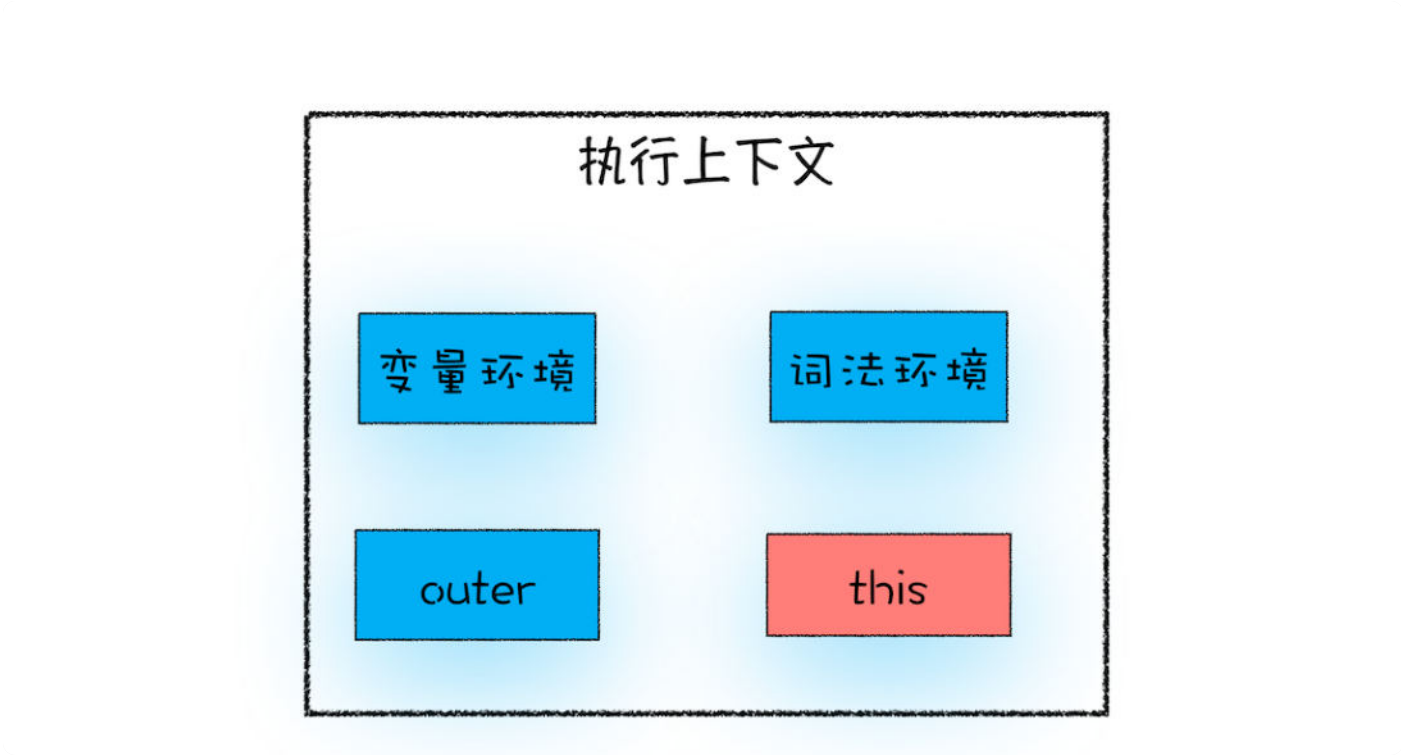
```
printName: function () {
```

```
        console.log(this.myName)
    }
```

接下来咱们就展开来介绍 this，不过在讲解之前，希望你能区分清楚**作用域链**和 **this** 是两套不同的系统，它们之间基本没太多联系。在前期明确这点，可以避免你在学习 this 的过程中，和作用域产生一些不必要的关联。

JavaScript 中的 this 是什么

关于 this，我们还是得先从执行上下文说起。在前面几篇文章中，我们提到执行上下文中包含了变量环境、词法环境、外部环境，但其实还有一个 this 没有提及，具体你可以参考下图：



执行上下文中的 this

从图中可以看出，**this 是和执行上下文绑定的**，也就是说每个执行上下文中都有一个 this。前面《08 | 调用栈：为什么 JavaScript 代码会出现栈溢出？》中我们提到过，执行上下文主要分为三种——全局执行上下文、函数执行上下文和 eval 执行上下文，所以对应的 this 也只有这三种——全局执行上下文中的 this、函数中的 this 和 eval 中的 this。

不过由于 eval 我们使用的不多，所以本文我们对此就不做介绍了，如果你感兴趣的话，可以自行搜索和学习相关知识。

那么接下来我们就重点讲解下**全局执行上下文中的 this** 和**函数执行上下文中的 this**。

全局执行上下文中的 this

首先我们来看看全局执行上下文中的 this 是什么。

你可以在控制台中输入`console.log(this)`来打印出来全局执行上下文中的 this，最终输出的是 window 对象。所以你可以得出这样一个结论：全局执行上下文中的 this 是指向 window 对象的。这也是 this 和作用域链的唯一交点，作用域链的最底端包含了 window 对象，全局执行上下文中的 this 也是指向 window 对象。

函数执行上下文中的 this

现在你已经知道全局对象中的 this 是指向 window 对象了，那么接下来，我们就来重点分析函数执行上下文中的 this。还是先看下面这段代码：

复制代码

```
function foo(){
  console.log(this)
}
foo()
```

我们在 foo 函数内部打印出来 this 值，执行这段代码，打印出来的也是 window 对象，这说明在默认情况下调用一个函数，其执行上下文中的 this 也是指向 window 对象的。估计你会好奇，那能不能设置执行上下文中的 this 来指向其他对象呢？答案是肯定的。通常情况下，有下面三种方式来设置函数执行上下文中的 this 值。

1. 通过函数的 call 方法设置

你可以通过函数的 **call** 方法来设置函数执行上下文的 this 指向，比如下面这段代码，我们就并没有直接调用 foo 函数，而是调用了 foo 的 call 方法，并将 bar 对象作为 call 方法的参数。

复制代码

```
let bar = {
  myName : "极客邦",
  test1 : 1
}
```

```
function foo(){
  this.myName = "极客时间"
}
foo.call(bar)
console.log(bar)
console.log(myName)
```

执行这段代码，然后观察输出结果，你就能发现 foo 函数内部的 this 已经指向了 bar 对象，因为通过打印 bar 对象，可以看出 bar 的 myName 属性已经由“极客邦”变为“极客时间”了，同时在全局执行上下文中打印 myName，JavaScript 引擎提示该变量未定义。

其实除了 call 方法，你还可以使用 **bind** 和 **apply** 方法来设置函数执行上下文中的 this，它们在使用上还是有一些区别的，如果感兴趣你可以自行搜索和学习它们的使用方法，这里我就不再赘述了。

2. 通过对象调用方法设置

要改变函数执行上下文中的 this 指向，除了通过函数的 call 方法来实现外，还可以通过对象调用的方式，比如下面这段代码：

复制代码

```
var myObj = {
  name : "极客时间",
  showThis: function(){
    console.log(this)
  }
}
myObj.showThis()
```

在这段代码中，我们定义了一个 myObj 对象，该对象是由一个 name 属性和一个 showThis 方法组成的，然后再通过 myObj 对象来调用 showThis 方法。执行这段代码，你可以看到，最终输出的 this 值是指向 myObj 的。

所以，你可以得出这样的结论：**使用对象来调用其内部的一个方法，该方法的 this 是指向对象本身的**。

其实，你也可以认为 JavaScript 引擎在执行`myObject.showThis()`时，将其转化为了：

```
myObj.showThis.call(myObj)
```

复制代码

接下来我们稍微改变下调用方式，把 showThis 赋给一个全局对象，然后再调用该对象，代码如下所示：

复制代码

```
var myObj = {
  name : "极客时间",
  showThis: function(){
    this.name = "极客邦"
    console.log(this)
  }
}
var foo = myObj.showThis
foo()
```

执行这段代码，你会发现 this 又指向了全局 window 对象。

所以通过以上两个例子的对比，你可以得出下面这样两个结论：

- **在全局环境中调用一个函数，函数内部的 this 指向的是全局变量 window。**
- **通过一个对象来调用其内部的一个方法，该方法的执行上下文中的 this 指向对象本身。**

3. 通过构造函数中设置

你可以像这样设置构造函数中的 this，如下面的示例代码：

复制代码

```
function CreateObj(){
  this.name = "极客时间"
}
var myObj = new CreateObj()
```

在这段代码中，我们使用 new 创建了对象 myObj，那你知道此时的构造函数 CreateObj 中的 this 到底指向了谁吗？

其实，当执行 new CreateObj() 的时候，JavaScript 引擎做了如下四件事：

- 首先创建了一个空对象 tempObj；
- 接着调用 CreateObj.call 方法，并将 tempObj 作为 call 方法的参数，这样当 CreateObj 的执行上下文创建时，它的 this 就指向了 tempObj 对象；
- 然后执行 CreateObj 函数，此时的 CreateObj 函数执行上下文中的 this 指向了 tempObj 对象；
- 最后返回 tempObj 对象。

为了直观理解，我们可以用代码来演示下：

```
var tempObj = {}
CreateObj.call(tempObj)
return tempObj
```

这样，我们就通过 new 关键字构建好了一个新对象，并且构造函数中的 this 其实就是新对象本身。

关于 new 的具体细节你可以参考这篇文章，这里我就不做过多介绍了。

this 的设计缺陷以及应对方案

就我个人而言，this 并不是一个很好的设计，因为它的很多使用方法都冲击人的直觉，在使用过程中存在着非常多的坑。下面咱们就来一起看看那些 this 设计缺陷。

1. 嵌套函数中的 this 不会从外层函数中继承

我认为这是一个严重的设计错误，并影响了后来的很多开发者，让他们“前赴后继”迷失在该错误中。我们还是结合下面这样一段代码来分析下：

```
var myObj = {
  name : "极客时间",
  showThis: function(){
    console.log(this)
```

```
    function bar(){console.log(this)}
    bar()
  }
}
myObj.showThis()
```

我们在这段代码的 showThis 方法里面添加了一个 bar 方法，然后接着在 showThis 函数中调用了 bar 函数，那么现在的问题是：bar 函数中的 this 是什么？

如果你是刚接触 JavaScript，那么你可能会很自然地觉得，bar 中的 this 应该和其外层 showThis 函数中的 this 是一致的，都是指向 myObj 对象的，这很符合人的直觉。但实际情况却并非如此，执行这段代码后，你会发现**函数 bar 中的 this 指向的是全局 window 对象，而函数 showThis 中的 this 指向的是 myObj 对象**。这就是 JavaScript 中非常容易让人迷惑的地方之一，也是很多问题的源头。

你可以通过一个小技巧来解决这个问题，比如在 showThis 函数中**声明一个变量 self 用来保存 this**，然后在 bar 函数中使用 self，代码如下所示：

复制代码

```
var myObj = {
  name : "极客时间",
  showThis: function(){
    console.log(this)
    var self = this
    function bar(){
      self.name = "极客邦"
    }
    bar()
  }
}
myObj.showThis()
console.log(myObj.name)
console.log(window.name)
```

执行这段代码，你可以看到它输出了我们想要的结果，最终 myObj 中的 name 属性值变成了“极客邦”。其实，这个方法的的本质是**把 this 体系转换为了作用域的体系**。

其实，**你也可以使用 ES6 中的箭头函数来解决这个问题**，结合下面代码：

```
var myObj = {
  name : "极客时间",
  showThis: function(){
    console.log(this)
    var bar = ()=>{
      this.name = "极客邦"
      console.log(this)
    }
    bar()
  }
}
myObj.showThis()
console.log(myObj.name)
console.log(window.name)
```

复制代码

执行这段代码，你会发现它也输出了我们想要的结果，也就是箭头函数 bar 里面的 this 是指向 myObj 对象的。这是因为 ES6 中的箭头函数并不会创建其自身的执行上下文，所以箭头函数中的 this 取决于它的外部函数。

通过上面的讲解，你现在应该知道了 this 没有作用域的限制，这点和变量不一样，所以嵌套函数不会从调用它的函数中继承 this，这样会造成很多不符合直觉的代码。要解决这个问题，你可以有两种思路：

- 第一种是把 this 保存为一个 self 变量，再利用变量的作用域机制传递给嵌套函数。
- 第二种是继续使用 this，但是要把嵌套函数改为箭头函数，因为箭头函数没有自己的执行上下文，所以它会继承调用函数中的 this。

2. 普通函数中的 this 默认指向全局对象 window

上面我们已经介绍过了，在默认情况下调用一个函数，其执行上下文中的 this 是默认指向全局对象 window 的。

不过这个设计也是一种缺陷，因为在实际工作中，我们并不希望函数执行上下文中的 this 默认指向全局对象，因为这样会打破数据的边界，造成一些误操作。如果要让函数执行上下文中的 this 指向某个对象，最好的方式是通过 call 方法来显示调用。

这个问题可以通过设置 JavaScript 的“严格模式”来解决。在严格模式下，默认执行一个函数，其函数的执行上下文中的 this 值是 undefined，这就解决上面的问题了。

总结

好了，今天就到这里，下面我们来回顾下今天的内容。

首先，在使用 this 时，为了避坑，你要谨记以下三点：

1. 当函数作为对象的方法调用时，函数中的 this 就是该对象；
2. 当函数被正常调用时，在严格模式下，this 值是 undefined，非严格模式下 this 指向的是全局对象 window；
3. 嵌套函数中的 this 不会继承外层函数的 this 值。

最后，我们还提了一下箭头函数，因为箭头函数没有自己的执行上下文，所以箭头函数的 this 就是它外层函数的 this。

这是我们“JavaScript 执行机制”模块的最后一节了，五节下来，你应该已经发现我们将近一半的时间都是在谈 JavaScript 的各种缺陷，比如变量提升带来的问题、this 带来问题等。我认为了解一门语言的缺陷并不是为了否定它，相反是为了能更加深入地了解它。我们在谈论缺陷的过程中，还结合 JavaScript 的工作流程分析了出现这些缺陷的原因，以及避开这些缺陷的方法。掌握了这些，相信你今后在使用 JavaScript 的过程中会更加得心应手。

思考时间

你可以观察下面这段代码：

复制代码

```
let userInfo = {
  name:"jack.ma",
  age:13,
  sex:male,
  updateInfo:function(){
    //模拟xmlhttprequest请求延时
    setTimeout(function(){
      this.name = "pony.ma"
      this.age = 39
      this.sex = female
    },100)
  }
}

userInfo.updateInfo()
```

我想通过 updateInfo 来更新 userInfo 里面的数据信息，但是这段代码存在一些问题，你能修复这段代码吗？

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

上一篇　10 | 作用域链和闭包 ： 代码中出现相同的变量，JavaScript引擎是如何选择的?

下一篇　12 | 栈空间和堆空间：数据是如何存储的?

精选留言 (83)

写留言

ytd

2019-08-29

```
// 修改方法一：箭头函数最方便
let userInfo = {
  name:"jack.ma",
  age:13,
  sex:'male',
  updateInfo:function(){
    // 模拟 xmlhttprequest 请求延时
    setTimeout(() => {
      this.name = "pony.ma"
      this.age = 39
      this.sex = 'female'
    },100)
  }
}

userInfo.updateInfo()
setTimeout(() => {
  console.log(userInfo)
},200)

// 修改方法二：缓存外部的this
let userInfo = {
  name:"jack.ma",
  age:13,
  sex:'male',
  updateInfo:function(){
    let me = this;
    // 模拟 xmlhttprequest 请求延时
    setTimeout(function() {
      me.name = "pony.ma"
      me.age = 39
      me.sex = 'female'
    },100)
  }
}
```

```
userInfo.updateInfo()
setTimeout(() => {
  console.log(userInfo);
},200)

// 修改方法三，其实和方法二的思路是相同的
let userInfo = {
  name:"jack.ma",
  age:13,
  sex:'male',
  updateInfo:function(){
    // 模拟 xmlhttprequest 请求延时
    void function(me) {
      setTimeout(function() {
        me.name = "pony.ma"
        me.age = 39
        me.sex = 'female'
      },100)
    }(this);
  }
}

userInfo.updateInfo()
setTimeout(() => {
  console.log(userInfo)
},200)

let update = function() {
  this.name = "pony.ma"
  this.age = 39
  this.sex = 'female'
}

方法四: 利用call或apply修改函数被调用时的this值(不知掉这么描述正不正确)
let userInfo = {
  name:"jack.ma",
  age:13,
  sex:'male',
  updateInfo:function(){
    // 模拟 xmlhttprequest 请求延时
    setTimeout(function() {
```

```
      update.call(userInfo);
      // update.apply(userInfo)
    }, 100)
  }
}

userInfo.updateInfo()
setTimeout(() => {
  console.log(userInfo)
},200)

// 方法五: 利用bind返回一个新函数，新函数被调用时的this指定为userInfo
let userInfo = {
  name:"jack.ma",
  age:13,
  sex:'male',
  update: function() {
    this.name = "pony.ma"
    this.age = 39
    this.sex = 'female'
  },
  updateInfo:function(){
    // 模拟 xmlhttprequest 请求延时
    setTimeout(this.update.bind(this), 100)
  }
}
```

作者回复: 很赞，总结的很全，这个可以做参考答案

共 9 条评论 >

154

悬炫

2020-05-01

老师的文章是我目前见过的，将浏览器原理讲的最生动易懂的文章了，没有之一，大赞

69

William

2019-08-29

setTimeOut() 函数内部的回调函数，this指向全局函数。修复：在外部绑this或者使用箭头函数。

```
```

```
let userInfo = {
  name:"jack.ma",
  age:13,
  sex: "male",
  updateInfo:function(){
    let that = this;
    // 模拟 xmlhttprequest 请求延时
    setTimeout(()=>{
      that.name = "pony.ma"
      that.age = 39
      that.sex = "female"
    },100)
  }
}

userInfo.updateInfo()
```

```
let userInfo = {
  name:"jack.ma",
  age:13,
  sex: "male",
  updateInfo:function(){
    // 模拟 xmlhttprequest 请求延时
    setTimeout(()=>{
      this.name = "pony.ma"
      this.age = 39
      this.sex = "female"
    },100)
  }
}

userInfo.updateInfo()
```

作者回复: 非常好!

补充下解释:
如果被setTimeout推迟执行的回调函数是某个对象的方法，那么该方法中的this关键字将指向全局环

境，而不是定义时所在的那个对象。

如果是严格模式，那么this会被设置为undefined。

这一点很容易让人混淆！！！

共 6 条评论 >　　25

悬炫
2019-08-29

关于箭头函数，文章中说其没有自己的执行上下文，难道箭头函数就像let定义的变量一样是哥块级作用域吗？其内部定义的变量都是存储在词法环境中是吗？

作者回复: 箭头函数在执行时比块级作用域的内容多，比函数执行上下文的内容少，砍掉了很多函数执行上下文中的组件。

不过在箭头函数在执行时也是有变量环境的，因为还要支持变量提升！所以变量环境的模块还是砍不掉的

共 6 条评论 >　　22

风一样的浪漫
2019-09-05

老师请问下outer的位置是在变量对象内还是外，第10节描述是在内部的，可是11节的图outer放在变量对象外面了

作者回复: 是在里面的，11为了图简单点，调整到外面了

共 2 条评论 >　　13

刘晓东
2020-03-14

老师您好，我想问您一下，这些知识您是怎么获得的？是看的书，还是自己研究了什么源代码？方便告知吗？

共 1 条评论 >　　8

凭实力写bug
2019-10-31

我记得执行上下文包括变量环境,词法环境,outer,this,如果箭头函数没有执行上下文,他的这些内容又是怎样的,还有他的作用域呢

共 1 条评论 >　　7

大威先生🐯🐯

2019-10-27

最后一个案例中，myObj对象的 showThis函数内部定义了bar函数，bar函数的执行环境具有全局性，因此this对象通常指向window；----摘要《JavaScript高级程序设计》

6

pyhhou

2019-08-29

思考题，有两种方法

1. 将 setTimeout 里面的函数变成箭头函数
2. 在 setTimeout 外将 this 赋值给其他的变量，setTimeout 里面的函数通过作用域链去改变 userInfo 的属性

很不错的文章，受益匪浅，感谢老师。这里有一个疑问就是，关于箭头函数，文章中说其没有自己的执行上下文，这里指的是箭头函数并不会创建自己的执行上下文变量并压栈，其只是被看作是一个块区域吗？那么在实际的开发中如何在普通函数和箭头函数之间做选择？关于这一点，老师有没有相关推荐的文章呢？谢谢老师

作者回复: 箭头要展开了得话一节来讲，关于箭头函数的最佳实践网上应该有不少资料，可以查查！

6

李懂

2019-08-29

文章只是简单讲了下this的几种场景，不像前面变量申明，可以很清晰的知道在执行上下文的位置，也没有画图，看完还是不知道不能深入理解，更多的是一种记忆，这种是指向window，那种是指向对象。能不能深入到是如何实现this，才能知道缺陷的原因，这里一直是没理解的难点！

作者回复: this的缺陷并不是浏览器实现机制导致的，而是浏览器按照标准来实现的。

其实浏览器说我可以实现得更好，但是标准摆在这儿，大家都只认标准！

共 5 条评论>

5

朱维娜🍍

2019-08-31

之前看到一种说法：this指向的永远是调用它的对象。按照这种说法，嵌套函数的调用者是window，与文中所述的“showThis调用内部函数不能继承this”有所出入，想请老师解答一下这

种说法是否正确?

作者回复: 调用者是对象，函数内部是调用的地方，不能说是调用者。

obj.showThis ()

这里的obj是调用者，通过点操作符来实现的

共 3 条评论>

4

王玄

2019-09-19

```
// 硬绑定
  let userInfo = {
    name: "jack.ma",
    age: 13,
    sex: 'male',
    updateInfo: function () {
      // 模拟 xmlhttprequest 请求延时
      setTimeout(function () {
        this.name = "pony.ma"
        this.age = 39
        this.sex = 'female'
      }.bind(this), 100)
    }
  }

  userInfo.updateInfo()


  // 缓存this
  let userInfo = {
    name: "jack.ma",
    age: 13,
    sex: 'male',
    updateInfo: function () {
      const self = this;
      // 模拟 xmlhttprequest 请求延时
      setTimeout(function () {
        self.name = "pony.ma"
        self.age = 39
        self.sex = 'female'
      }, 100)
```

```
  }
}

userInfo.updateInfo()

// 箭头
let userInfo = {
  name: "jack.ma",
  age: 13,
  sex: 'male',
  updateInfo: function () {
    // 模拟 xmlhttprequest 请求延时
    setTimeout(() => {
      this.name = "pony.ma"
      this.age = 39
      this.sex = 'female'
    }, 100)
  }
}

userInfo.updateInfo()

// 跟缓存this差不多
let userInfo = {
  name: "jack.ma",
  age: 13,
  sex: 'male',
  updateInfo: function () {
    // 模拟 xmlhttprequest 请求延时
    setTimeout(function(self) {
      self.name = "pony.ma"
      self.age = 39
      self.sex = 'female'
    }, 100, this)
  }
}

userInfo.updateInfo()
```



Heartbeats

Heartbeats
2019-09-10

谁调用了它，它就指向谁

共 3 条评论>

3

White
2020-07-30

老师，关于箭头函数，文章中说”箭头函数并不会创建其自身的执行上下文“，那么在调用箭头函数时，是将什么推入调用栈了呢？其内部变量又放在哪里了呢？我看评论并没有说的很清楚呢？这块是否可以讲具体些呢？执行箭头函数时，是怎样一个过程呢？

3

爱吃锅巴的沐泡
2019-09-02

对于思考题的一些问题：
1、对象中定义的方法是一个全局函数嘛？
2、setTimeout()的回调函数中的this指向window 是因为widow调用setTimeout()？ 还是因为 定义的回调函数和 外部的updateInfo函数嵌套定义？或者还是因为其他的原因？
3、这里的setTimeout()的回调函数属于是一个字面量的函数定义作为参数进行传递，这种参数形式的函数定义与外面的updateInfo()函数可以算作嵌套定义嘛？（这里好像又回到了问题2）

2

七月有风
2019-08-29

做题面试写代码，这些就够了，但this到底是什么，还是不懂

2

我来人间一趟
2021-03-12

其实在外部函数中设置self和使用箭头函数原理是一致的，都是将this的机制转换到作用域的机制解决的，有兴趣的同学可以了解下箭头函数的实现，其实就是通过解析ast，找到parent节点然后在这个节点中声明一个类似self的东西，在遍历箭头函数中所有使用this的地方，把this变为self，之后在把ast的节点名转换成普通函数的命名，原理是一致，箭头函数只是一种语法糖的实现

1

Gopal
2020-10-31

https://juejin.im/post/6882527259584888845#heading-8

关于 this，你不知道的 JavaScript 中讲得也很精彩，做了一些笔记，对于判断 this 的指向很有帮助

1

Winn

2019-08-30

通俗易懂，由简入深，把this说得最清楚的文章

1

潘启宝

2019-08-29

```
let userInfo = {
  name:"jack.ma",
  age:13,
  sex:'male',
  updateInfo:function(){
    // 模拟 xmlhttprequest 请求延时
    setTimeout(function(){
      this.name = "pony.ma"
      this.age = 39
      this.sex = 'female'
    }.bind(this),100)
  }
}

userInfo.updateInfo()
```

作者回复: 使用bind没有问题

共 6 条评论>

1