11-this：从JavaScript执行上下文的视角讲清楚this

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：

```
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：

```
#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**。

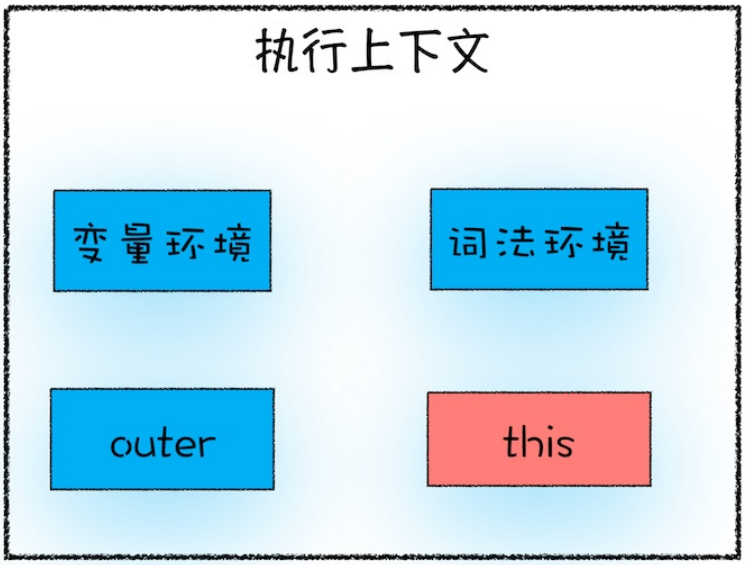
所以，在JavaScript中可以使用this实现在printName函数中访问到bar对象的myName属性了。具体该怎么操作呢？你可以调整printName的代码，如下所示：

```
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：

```
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。

```
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：

```
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)
```

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：

```
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：

```
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：

```
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)
```

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：

```
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。

精选留言：

- 悬炫 2019-08-29 09:43:04
  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？ [3赞]

- Winn 2019-08-30 06:44:47
  通俗易懂，由简入深，把this说得最清楚的文章 [1赞]

- William 2019-08-29 21:52:39
  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()
```

[1赞]

- 潘启宝 2019-08-29 10:45:03

```
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() [1赞]

- pyhhou 2019-08-29 04:25:53
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 [1赞]

- Geek_b42f75 2019-08-30 10:59:53
发现一个事，虽然setTimeout改成箭头函数了，里面的this指向userInfo这个对象了。
但是在console.log(userInfo.age)打印age的时候，为什么还是13，没有改成39呢？
我看不用setTimeout，直接在updateInfo方法里调用this.age = 39是能改变的。
let userInfo = {
name:"jack.ma",
age:13,
sex:'male',
updateInfo:function(){
// this.age = 39
// 模拟 xmlhttprequest 请求延时
setTimeout(() => {
this.name = "pony.ma"
this.age = 39
this.sex = 'female'
},100)
}
}
userInfo.updateInfo()
console.log(userInfo.age)

- This 2019-08-30 10:46:15
思路清晰，是我看过讲解最清楚的this文章

- stone 2019-08-29 20:31:01

let userInfo = {
name:"jack.ma",
age:13,
sex:male,
// 1,
const that = this
updateInfo:function(){

```
// 模拟 xmlhttprequest 请求延时
setTimeout(function(){
that.name = "pony.ma"
that.age = 39
that.sex = female
},100)
}

// 2
updateInfo :function(){
// 模拟 xmlhttprequest 请求延时
setTimeout(() => {
this.name = "pony.ma"
this.age = 39
this.sex = female
},100)
}
}

userInfo.updateInfo()
```

- monalisali 2019-08-29 17:49:24

思考题：
这份代码在开发中是很常见的一种操作，调用了api后，希望在callback中执行一些操作。但此时，callback中的this已经不是原先那个caller了（即题目中的updateInfo）, 而是callback

常见的方式是在后台的api中返回一个对象，如：{result: true, data:{name:'pony.ma',age:39, sex:'female'}};

```
let userInfo = {
name:"jack.ma",
age:13,
sex:'male',
updateInfo:function(){
// 模拟 xmlhttprequest 请求延时
setTimeout(function(resp){
if(resp.result){
var data = {name:'pony.ma', age:39, sex:'female'}
userInfo.name = data.name;
userInfo.age = data.age;
userInfo.sex = data.sex
}

},100)
}
}

userInfo.updateInfo()
```

- 谢海涛 2019-08-29 17:48:58

```
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)
}
}
```

- 依然爱你 2019-08-29 17:35:15
  和某位仁兄同样的问题，箭头函数没有自己的执行上下文，那么里面的变量环境和词法环境在哪？

- 羽蝶曲 2019-08-29 17:19:26
  ```
  var myObj = {
  name : " 极客时间 ",
  showThis: function(){
  this.name = " 极客邦 "
  console.log(this)
  }
  }
  var foo = myObj.showThis
  foo();
  (myObj.showThis)();
  ```
  老师，您好，我想问个问题，为何(myObj.showThis)()的this指向的是myObj而不是window呢，和foo = myObj.showThis的区别是什么呢？

- 羽蝶曲 2019-08-29 17:18:33
  ```
  var myObj = {
  name : " 极客时间 ",
  showThis: function(){
  this.name = " 极客邦 "
  console.log(this)
  }
  }
  var foo = myObj.showThis
  foo();
  (myObj.showThis)();
  ```

- 蓝配鸡 2019-08-29 16:40:48
  思考题个人看法

  settimeout中的回掉函数中的this是window

  所以最终结果window里多了几个变量
  调用的对象本身并没有update

- 歌在云端 2019-08-29 16:18:52
  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"
  // }.call(this), 100)
  setTimeout(() =>{
  this.name = "pony.ma"
  this.age = 39
  this.sex = "female"
  }, 100)
  }
  }
  试了一下用箭头函数和将this绑定到一个self变量上面都不可以，只有用call里面传入this才行。老师能讲一下为什么吗

- 羽蝶曲 2019-08-29 15:22:34
  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()

- 七月有风 2019-08-29 14:05:01
  做题面试写代码，这些就够了，但this到底是什么，还是不懂

- ytd 2019-08-29 13:29:48
  // 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)
}
}
```

- 许童童 2019-08-29 11:29:27

思考题：读了老师的文章，很容易解决这个问题。三种方式：
1.定义局部self：var self = this
2.使用箭头函数
3.内部function使用bind：setTimeout(function(){...}.bind(userInfo),100)

- Angus 2019-08-29 10:31:45
  setTimeout的第一个参数是一个函数，这个函数将会延迟执行，执行时，这个函数的this将会指向全局window。解决办法就像文中提到的两种方式：使用self变量保存this或者使用箭头函数。

  之前我是这么记忆this指向的：对于函数中的this，谁调用了这个函数，函数的this就指向谁；对于箭头函数，定义的时候就决定了this指向，在哪里定义的this就指向谁。

  另外关于改变this的第二种方式：通过对象调用方式设置。利用这种方式，可以用来模拟实现call/apply/bind方法。

  执行上下文包括：变量环境、词法环境、外部环境、this。

  除了这种标准概念式的东西，其实更像知道为何这样设计，浏览器是如何处理的。