



**外文文献翻译**

学 院 计算机学院

专 业 网络工程

年级班别 2013级（4）班

学 号 3113006587

学生姓名 曾诗永

指导教师 汪明慧

2017 年 5月

# Understand JavaScript’s “this” With Clarity, and Master It

The this keyword in JavaScript confuses new and seasoned JavaScript developers alike. This article aims to elucidate this in its entirety. By the time we make it through this article, this will be one part of JavaScript we never have to worry about again. We will understand how to use this correctly in every scenario, including the ticklish situations where it usually proves most elusive.

Bov Academy Logo

Bov Academy

of Programming and Futuristic Engineering

Invest in Your Future: Become a Frontend/Fullstack Engineer

Within 6–8 Months, Earn > the Avg. New CS Graduate Earns

30% Discount for the May Session

By the founder of JavaScriptIsSexy

We use this similar to the way we use pronouns in natural languages like English and French. We write, “John is running fast because he is trying to catch the train.”

Table of Contents

Receive Updates

JavaScript’s this Keyword Basics

The Biggest Gotcha with JavaScript “this” keyword

The use of this in the global scope

When this is most misunderstood and becomes tricky

Fix this when used in a method passed as a callback

Fix this inside closure

Fix this when method is assigned to a variable

Fix this when borrowing methods

Note the use of the pronoun “he.” We could have written this: “John is running fast because John is trying to catch the train.” We don’t reuse “John” in this manner, for if we do, our family, friends, and colleagues would abandon us. Yes, they would. Well, maybe not your family, but those of us with fair-weather friends and colleagues. In a similar graceful manner, in JavaScript, we use the this keyword as a shortcut, a referent; it refers to an object; that is, the subject in context, or the subject of the executing code. Consider this example:

var person = {

firstName: "Penelope",

lastName: "Barrymore",

fullName: function () {

​// Notice we use "this" just as we used "he" in the example sentence earlier?:​

console.log(this.firstName + " " + this.lastName);

​// We could have also written this:​​

console.log(person.firstName + " " + person.lastName);

}

}

Receive Updates

Your Email:

Go

If we use person.firstName and person.lastName, as in the last example, our code becomes ambiguous. Consider that there could be another global variable (that we might or might not be aware of) with the name “person.” Then, references to person.firstName could attempt to access the firstName property from the person global variable, and this could lead to difficult-to-debug errors. So we use the “this” keyword not only for aesthetics (i.e., as a referent), but also for precision; its use actually makes our code more unambiguous, just as the pronoun “he” made our sentence more clear. It tells us that we are referring to the specific John at the beginning of the sentence.

Just like the pronoun “he” is used to refer to the antecedent (antecedent is the noun that a pronoun refers to), the this keyword is similarly used to refer to an object that the function (where this is used) is bound to. The this keyword not only refers to the object but it also contains the value of the object. Just like the pronoun, this can be thought of as a shortcut (or a reasonably unambiguous substitute) to refer back to the object in context (the “antecedent object”). We will learn more about context later.

JavaScript’s this Keyword Basics

First, know that all functions in JavaScript have properties, just as objects have properties. And when a function executes, it gets the this property—a variable with the value of the object that invokes the function where this is used.

The this reference ALWAYS refers to (and holds the value of) an object—a singular object—and it is usually used inside a function or a method, although it can be used outside a function in the global scope. Note that when we use strict mode, this holds the value of undefined in global functions and in anonymous functions that are not bound to any object.

this is used inside a function (let’s say function A) and it contains the value of the object that invokes function A. We need this to access methods and properties of the object that invokes function A, especially since we don’t always know the name of the invoking object, and sometimes there is no name to use to refer to the invoking object. Indeed, this is really just a shortcut reference for the “antecedent object”—the invoking object.

Ruminate on this basic example illustrating the use of this in JavaScript:

var person = {

firstName :"Penelope",

lastName :"Barrymore",

// Since the "this" keyword is used inside the showFullName method below, and the showFullName method is defined on the person object,​

// "this" will have the value of the person object because the person object will invoke showFullName ()​

showFullName:function () {

console.log (this.firstName + " " + this.lastName);

}

​

}

​

person.showFullName (); // Penelope Barrymore

And consider this basic jQuery example with of this:

// A very common piece of jQuery code​

​

$ ("button").click (function (event) {

// $(this) will have the value of the button ($("button")) object​

​// because the button object invokes the click () method​

console.log ($ (this).prop ("name"));

});

I shall expound on the preceding jQuery example: The use of $(this), which is jQuery’s syntax for the this keyword in JavaScript, is used inside an anonymous function, and the anonymous function is executed in the button’s click () method. The reason $(this) is bound to the button object is because the jQuery library binds $(this) to the object that invokes the click method. Therefore, $(this) will have the value of the jQuery button ($(“button”)) object, even though $(this) is defined inside an anonymous function that cannot itself access the “this” variable on the outer function.

Note that the button is a DOM element on the HTML page, and it is also an object; in this case it is a jQuery object because we wrapped it in the jQuery $() function.

UPDATE: the following (“Biggest Gotcha” section) was added a couple of days after I published the article

The Biggest Gotcha with JavaScript “this” keyword

If you understand this one principle of JavaScript’s this, you will understand the “this” keyword with clarity: this is not assigned a value until an object invokes the function where this is defined. Let’s call the function where this is defined the “this Function.”

Even though it appears this refers to the object where it is defined, it is not until an object invokes the this Function that this is actually assigned a value. And the value it is assigned is based exclusively on the object that invokes the this Function. this has the value of the invoking object in most circumstances. However, there are a few scenarios where this does not have the value of the invoking object. I touch on those scenarios later.

The use of this in the global scope

In the global scope, when the code is executing in the browser, all global variables and functions are defined on the window object. Therefore, when we use this in a global function, it refers to (and has the value of) the global window object (not in strict mode though, as noted earlier) that is the main container of the entire JavaScript application or web page.

Thus:

var firstName = "Peter",

lastName = "Ally";

​

function showFullName () {

// "this" inside this function will have the value of the window object​

// because the showFullName () function is defined in the global scope, just like the firstName and lastName​

console.log (this.firstName + " " + this.lastName);

}

​

var person = {

firstName :"Penelope",

lastName :"Barrymore",

showFullName:function () {

// "this" on the line below refers to the person object, because the showFullName function will be invoked by person object.​

console.log (this.firstName + " " + this.lastName);

}

}

​

showFullName (); // Peter Ally​

​

// window is the object that all global variables and functions are defined on, hence:​

window.showFullName (); // Peter Ally​

​

// "this" inside the showFullName () method that is defined inside the person object still refers to the person object, hence:​

person.showFullName (); // Penelope Barrymore

When this is most misunderstood and becomes tricky

The this keyword is most misunderstood when we borrow a method that uses this, when we assign a method that uses this to a variable, when a function that uses this is passed as a callback function, and when this is used inside a closure—an inner function. We will look at each scenario and the solutions for maintaining the proper value of this in each example.

A bit about “Context” before we continue

The context in JavaScript is similar to the subject of a sentence in English: “John is the winner who returned the money.” The subject of the sentence is John, and we can say the context of the sentence is John because the focus of the sentence is on him at this particular time in the sentence. Even the “who” pronoun is referring to John, the antecedent. And just like we can use a semicolon to switch the subject of the sentence, we can have an object that is current context and switch the context to another object by invoking the function with another object.

Similarly, in JavaScript code:

var person = {

firstName :"Penelope",

lastName :"Barrymore",

showFullName:function () {

​// The "context"​

console.log (this.firstName + " " + this.lastName);

}

}

​

​// The "context", when invoking showFullName, is the person object, when we invoke the showFullName () method on the person object.​

​// And the use of "this" inside the showFullName() method has the value of the person object,​

person.showFullName (); // Penelope Barrymore​

​

​// If we invoke showFullName with a different object:​

​var anotherPerson = {

firstName :"Rohit",

lastName :"Khan"​

};

​

​// We can use the apply method to set the "this" value explicitly—more on the apply () method later.​

​// "this" gets the value of whichever object invokes the "this" Function, hence:​

person.showFullName.apply (anotherPerson); // Rohit Khan​

​

​// So the context is now anotherPerson because anotherPerson invoked the person.showFullName () method by virtue of using the apply () method​

The takeaway is that the object that invokes the this Function is in context, and we can change the context by invoking the this Function with another object; then this new object is in context.

Here are scenarios when the this keyword becomes tricky. The examples include solutions to fix errors with this:

Fix this when used in a method passed as a callback

Things get a touch hairy when we pass a method (that uses this) as a parameter to be used as a callback function. For example:

// We have a simple object with a clickHandler method that we want to use when a button on the page is clicked​

var user = {

data:[

{name:"T. Woods", age:37},

{name:"P. Mickelson", age:43}

],

clickHandler:function (event) {

var randomNum = ((Math.random () \* 2 | 0) + 1) - 1; // random number between 0 and 1​

​

// This line is printing a random person's name and age from the data array​

console.log (this.data[randomNum].name + " " + this.data[randomNum].age);

}

}

​

// The button is wrapped inside a jQuery $ wrapper, so it is now a jQuery object​

// And the output will be undefined because there is no data property on the button object​

$ ("button").click (user.clickHandler); // Cannot read property '0' of undefined

In the code above, since the button ($(“button”)) is an object on its own, and we are passing the user.clickHandler method to its click() method as a callback, we know that this inside our user.clickHandler method will no longer refer to the user object. this will now refer to the object where the user.clickHandler method is executed because this is defined inside the user.clickHandler method. And the object that is invoking user.clickHandler is the button object—user.clickHandler will be executed inside the button object’s click method.

Note that even though we are calling the clickHandler () method with user.clickHandler (which we have to do, since clickHandler is a method defined on user), the clickHandler () method itself will be executed with the button object as the context to which “this” now refers. So this now refers to is the button ($(“button”)) object.

At this point, it should be apparent that when the context changes—when we execute a method on some other object than where the object was originally defined, the this keyword no longer refers to the original object where “this” was originally defined, but it now refers to the object that invokes the method where this was defined.

Solution to fix this when a method is passed as a callback function:

Since we really want this.data to refer to the data property on the user object, we can use the Bind (), Apply (), or Call () method to specifically set the value of this.

I have written an exhaustive article, JavaScript’s Apply, Call, and Bind Methods are Essential for JavaScript Professionals, on these methods, including how to use them to set the this value in various misunderstood scenarios. Rather than re-post all the details here, I recommend you read that entire article, which I consider a must read for JavaScript Professionals.

To fix this problem in the preceding example, we can use the bind method thus:

Instead of this line:

$ ("button").click (user.clickHandler);

We have to bind the clickHandler method to the user object like this:

$("button").click (user.clickHandler.bind (user)); // P. Mickelson 43

— View a working example of this on JSBin

Fix this inside closure

Another instance when this is misunderstood is when we use an inner method (a closure). It is important to take note that closures cannot access the outer function’s this variable by using the this keyword because the this variable is accessible only by the function itself, not by inner functions. For example:

var user = {

tournament:"The Masters",

data :[

{name:"T. Woods", age:37},

{name:"P. Mickelson", age:43}

],

​

clickHandler:function () {

// the use of this.data here is fine, because "this" refers to the user object, and data is a property on the user object.​

​

this.data.forEach (function (person) {

// But here inside the anonymous function (that we pass to the forEach method), "this" no longer refers to the user object.​

// This inner function cannot access the outer function's "this"​

console.log ("What is This referring to? " + this); //[object Window]​

console.log (person.name + " is playing at " + this.tournament);

// T. Woods is playing at undefined​

// P. Mickelson is playing at undefined​

})

}

​

}

​

user.clickHandler(); // What is "this" referring to? [object Window]

this inside the anonymous function cannot access the outer function’s this, so it is bound to the global window object, when strict mode is not being used.

Solution to maintain this inside anonymous functions:

To fix the problem with using this inside the anonymous function passed to the forEach method, we use a common practice in JavaScript and set the this value to another variable before we enter the forEach

method:

var user = {

tournament:"The Masters",

data :[

{name:"T. Woods", age:37},

{name:"P. Mickelson", age:43}

],

​

clickHandler:function (event) {

// To capture the value of "this" when it refers to the user object, we have to set it to another variable here:​

// We set the value of "this" to theUserObj variable, so we can use it later​

var theUserObj = this;

this.data.forEach (function (person) {

// Instead of using this.tournament, we now use theUserObj.tournament​

console.log (person.name + " is playing at " + theUserObj.tournament);

})

}

​

}

​

user.clickHandler();

// T. Woods is playing at The Masters​

// P. Mickelson is playing at The Masters

It is worth noting that many JavaScript developers like to name a variable “that,” as seen below, to set the value of this. The use of the word “that” is very awkward for me, so I try to name the variable a noun that describes which object “this” is referring to, hence my use of var theUserObj = this in the preceding code.

// A common practice amongst JavaScript users is to use this code​

var that = this;

— View a working example of this on JSBin

Fix this when method is assigned to a variable

The this value escapes our imagination and is bound to another object, if we assign a method that uses this to a variable. Let’s see how:

// This data variable is a global variable​

var data = [

{name:"Samantha", age:12},

{name:"Alexis", age:14}

];

​

var user = {

// this data variable is a property on the user object​

data :[

{name:"T. Woods", age:37},

{name:"P. Mickelson", age:43}

],

showData:function (event) {

var randomNum = ((Math.random () \* 2 | 0) + 1) - 1; // random number between 0 and 1​

​

// This line is adding a random person from the data array to the text field​

console.log (this.data[randomNum].name + " " + this.data[randomNum].age);

}

​

}

​

// Assign the user.showData to a variable​

var showUserData = user.showData;

​

// When we execute the showUserData function, the values printed to the console are from the global data array, not from the data array in the user object​

//​

showUserData (); // Samantha 12 (from the global data array)​

​

Solution for maintaining this when method is assigned to a variable:

We can fix this problem by specifically setting the this value with the bind method:

// Bind the showData method to the user object​

var showUserData = user.showData.bind (user);

​

// Now we get the value from the user object, because the <em>this</em> keyword is bound to the user object​

showUserData (); // P. Mickelson 43

Fix this when borrowing methods

Borrowing methods is a common practice in JavaScript development, and as JavaScript developers, we will certainly encounter this practice time and again. And from time to time, we will engage in this time-saving practice as well. For more on borrowing methods, read my in-depth article, JavaScript’s Apply, Call, and Bind Methods are Essential for JavaScript Professionals.

Let’s examine the relevance of this in the context of borrowing methods:

// We have two objects. One of them has a method called avg () that the other doesn't have​

// So we will borrow the (avg()) method​

var gameController = {

scores :[20, 34, 55, 46, 77],

avgScore:null,

players :[

{name:"Tommy", playerID:987, age:23},

{name:"Pau", playerID:87, age:33}

]

}

​

var appController = {

scores :[900, 845, 809, 950],

avgScore:null,

avg :function () {

​

var sumOfScores = this.scores.reduce (function (prev, cur, index, array) {

return prev + cur;

});

​

this.avgScore = sumOfScores / this.scores.length;

}

}

​

//If we run the code below,​

// the gameController.avgScore property will be set to the average score from the appController object "scores" array​

// Don't run this code, for it is just for illustration; we want the appController.avgScore to remain null​

gameController.avgScore = appController.avg();

​

The avg method’s “this” keyword will not refer to the gameController object, it will refer to the appController object because it is being invoked on the appController.

Solution for fixing this when borrowing methods:

To fix the issue and make sure that this inside the appController.avg () method refers to gameController, we can use the apply () method thus:

​

// Note that we are using the apply () method, so the 2nd argument has to be an array—the arguments to pass to the appController.avg () method.​

appController.avg.apply (gameController, gameController.scores);

​

// The avgScore property was successfully set on the gameController object, even though we borrowed the avg () method from the appController object​

console.log (gameController.avgScore); // 46.4​

​

// appController.avgScore is still null; it was not updated, only gameController.avgScore was updated​

console.log (appController.avgScore); // null

The gameController object borrows the appController’s avg () method. The “this” value inside the appController.avg () method will be set to the gameController object because we pass the gameController object as the first parameter to the apply () method. The first parameter in the apply method always sets the value of “this” explicitly.

— View a working example of this on JSBin

Final Words

I am hopeful you have learned enough to help you understand the this keyword in JavaScript. Now you have the tools (bind, apply, and call, and setting this to a variable) necessary to conquer JavaScript’s this in every scenario.

As you have learned, this gets a bit troublesome in situations where the original context (where this was defined) changes, particularly in callback functions, when invoked with a different object, or when borrowing methods. Always remember that this is assigned the value of the object that invoked the this Function.

Be good. Sleep well. And enjoy coding.

# 理解并掌握 JavaScript 中 this 的用法

在 JavaScript 中，this 这个关键字常常困扰着初学者甚至一些进阶的开发者。这篇文章旨在完完全全阐明 this。当你读完本文之后，你就再也不会为 this 所困惑了。你将会理解 this 的各种使用场景，包括那些最难懂的情形。

我们使用 this 的方式和在英语或法语中使用代词的方式十分类似。我们会这样写「李华正在飞快地跑着，因为他正在赶火车。」注意这里代词「他」的用法。我们也可以这样写：「李华正在飞快地跑着，因为李华正在赶火车。」我们通常不会把「李华」这个名字像这样重复使用，因为这样显得很神经。类似地，在 JavaScript 中，我们使用 this 作为一种指代。它指代一个对象（object），也就是那个上下文中的主语，或者说运行时的主体。考虑下面这个例子：

var person = {

firstName: "Penelope",

lastName: "Barrymore",

fullName: function () {

// 注意我们使用「this」关键字就像我们在上文中使用「他」一样

console.log(this.firstName + " " + this.lastName);

// 我们也可以这样写

console.log(person.firstName + " " + person.lastName);

}

}

如果我们使用 person.firstName 和 person.lastName 这种写法的话，我们的代码就会变得有歧义。假设有一个全局变量（我们或许有意为之，或许根本没有意识到）的名字也叫 person ，那么 person.firstName 将会尝试读取那个全局变量 person 中的 firstName 属性，这将可能导致极难调试的错误。所以我们使用 this 关键字，不仅仅是因为这看起来十分优雅，还因为这样使用更加准确。使用 this 消除了我们代码中的歧义，就像在上文中使用「他」让我们的话显得更加清晰一样。它让我们明白我们想要指代的李华就是句子刚开头提到的那个李华。

就像代词「他」用来指代之前提到的人一样，this 这个关键字也是用来指代那个被当前函数（就是使用了 this 的函数）绑定的对象。this 这个关键字不仅仅是指代那个对象，并且包含了那个对象的值。这很类似代词，this 可以被视作是指代「上下文」中对象（也称为「祖先对象」）的一种便捷的方式（同时也是一种没有歧义的替换）。我们将在后面学习更多关于「上下文」 的概念。

JavaScript this 用法基础

首先，我们已经知道在 JavaScript 中，函数和对象一样都有属性。而当一个函数执行的时候，它就获得了 this 这个属性。而 this 其实就是一个具有调用当前函数的对象的值的变量。

this 这个变量 永远 指向 一个 对象，并且拥有这个对象的值。虽然 this 可以在全局作用域中出现，但它通常还是会在函数体内或对象的方法内。有一点要注意的是，当我们使用严格模式（strict mode）的时候，this 在全局函数中和匿名函数中的值是未定义的（undefined），不指向任何一个对象。

this 在一个函数体内出现的时候（设为函数 A ），它包含了调用函数 A 的那个对象的值。我们需要使用 this 来读取调用函数 A 的那个对象的方法或是属性。而这在我们不知道那个对象的名字，甚至有时候那个对象没有名字的情况下就变得尤为重要。实际上，this 真的仅仅就是对「祖先对象」，或者说调用这个函数的那个对象，的一个便捷的指代而已。

我们用一个例子来展示 JavaScript 中 this 的一些基本用法，也来回顾一下上文的内容：

var person = {

firstName: "Penelope",

lastName: "Barrymore",

// 因为 this 关键字在 showFullName 方法中被用到，而 showFullName 在 person 这个对象中被定义，

// 所以 this 将会具有 person 这个对象的值，因为 person 对象将会调用 showFullName()

showFullName: function() {

console.log (this.firstName + " " + this.lastName);

}

}

​

person.showFullName(); // Penelope Barrymore

再来看看 jQuery 中 this 用法的例子：

// 一段非常普遍的 jQuery 代码

​

$ ("button").click (function (event) {

// $(this) 将具有那个 ($("button")) 按钮对象的值

// 因为那个按钮对象调用了 click() 方法

console.log ($(this).prop("name"));

});

我来解释一下上面的这个 jQuery 示例：$(this) 是 jQuery 中与 JavaScript 中 this 类同的语法，它被用在一个匿名函数中，而这个匿名函数在一个按钮的 click() 方法中被执行。$(this) 之所以具有这个按钮对象的值是因为 jQuery 库把 $(this) 和那个调用了 click 方法的对象手动 绑定 （bind）在一起了。 因此，即使 $(this) 是在一个匿名函数中被定义，并且自身不能读取外部函数中的 this 变量，它仍然能够具有那个 jQuery 按钮对象 ($("button")) 的值。

注意，按钮（button）是一个 HTML 页面上的 DOM 元素，同时也是一个对象；在上面这个例子中的按钮是一个 jQuery 对象，因为我们把它包装在 jQuery 的 $() 函数中了。

理解 JavaScript this 的关键

如果你理解了 JavaScript this 的以下这个原则的话，那你对 this 这个关键字就会有一个清晰的认识了：只有一个对象调用了包含 this 的函数的时候，this 才会被赋值。我们不妨把包含 this 的函数称作 this 函数。

在一个对象方法中定义的 this 看起来好像指向了这个对象本身，但仍然只有在某个对象调用了这个 this 函数 的时候它才被赋值。并且被赋的那个值 只依赖于 调用了 this 函数 的那个对象。虽然在大多数情况下， this 都是那个调用了 this 函数 的那个对象，但也有一些情况不是这样的。我将会在后文中讲到这一点。

在全局作用域中使用 this

在全局作用域中，当代码在浏览器中执行的时候，所有的全局变量和函数都被定义在 window 对象上。因此，当我们在全局函数中使用 this 的时候，它会指向全局 window 对象并且拥有它的值（除非在严格模式下），此时的 this 就成了整个 JavaScript 应用程序或者说整个网页的主容器。

所以：

var firstName = "Peter",

lastName = "Ally";

​

function showFullName () {

// 在这个函数中，this 将会拥有 window 对象的值

// 因为 showFullName() 函数，和 firstName, lastName 一样是定义在全局作用域的

console.log (this.firstName + " " + this.lastName);

}

​

var person = {

firstName: "Penelope",

lastName: "Barrymore",

showFullName:function () {

// 下面这行中的 this 指代 person 对象，因为 showFullName 这个函数将会被 person 对象调用

console.log (this.firstName + " " + this.lastName);

}

}

​

showFullName (); // Peter Ally​

​

// 所有的全局变量和函数都定义在 window 对象上面，所以：

window.showFullName (); // Peter Ally​

​

// 在 person 对象中定义的 showFullName() 函数中的 this 仍然指向 person 对象，所以：

person.showFullName (); // Penelope Barrymore

this 最容易被误解和难以掌握的情景

this 关键字在以下场景中常常被误解：当我们借用一个使用了 this 的方法的时候；当我们把一个只用了 this 的方法赋给一个变量的时候；当一个使用了 this 的方法被当作回调函数传入的时候；当 this 在闭包中使用的时候。我们能过举例来详细地解释在上面的每一种情形中如何使 this 拥有合适的值。

一点重要的提示

在接下去讲之前，我们先来谈谈「上下文」（Context）这个概念

在 JavaScript 中，上下文的概念和一个英文句子中主语的概念相类似：「John is the winner who returned the money.」这句话中的主语是 John ，我们可以说这句话的语境（上下文）是 John ，因为这句话此时的关注点在 John 身上。代词「who」也是指代先行词 John。正如我们可以使用分号来切换句子的主语一样，我们可以通过让另一个对象去调用本对象的方法的方式来切换上下文。

用代码可以这样描述

var person = {

firstName: "Penelope",

lastName: "Barrymore",

showFullName: function() {

// 「上下文」

console.log(this.firstName + " " + this.lastName);

}

}

// 当我们在 person 对象上调用 showFullName() 方法的时候，「上下文」是 persion 对象。

// 这时在 showFullName() 方法里面使用的 this 就拥有了 person 对象的值

person.showFullName(); // Penelope Barrymore

// 当我们使用另一个对象来调用 showFullName 的时候

var anotherPerson = {

firstName: "Rohit",

lastName: "Khan"

};

// 我们可以使用 apply 方法来显式地设置 this 的值。关于 apply() 方法，我们将在后文中详细解释

// this 得到的永远是调用它的那个对象的值，因此：

person.showFullName.apply(anotherPerson); // Rohit Khan

// 所以现在上下文就变成了 anotherPerson ，因为是 anotherPerson 使用 apply() 方法调用了 person.showFullName() 方法

在下面这些情景中，this 关键字可能会变得十分难以理解。我们在示例中同时给出了解决有关 this 使用错误的方案。

1. 解决当包含 this 的方法被当做回调函数时遇到的问题

当我们把含有 this 的方法当做回调函数的时候代码往往变得十分难以理解。比如：

// 我们有一个简单的对象，它有一个 clickHandler 方法，我们想要使当页面上的一个按钮被点击时它被调用

var user = {

data: [

{name: "T. Woods", age: 37},

{name: "P. Mickelson", age: 43}

],

clickHandler: function(event) {

var randomNum = ((Math.random() \* 2 | 0) + 1) - 1; // 产生 0 到 1 之间的随机数

// 下面这行会随机打印出一个 data 数组中的人的姓名和年龄

console.log(this.data[randomNum].name + " " + this.data[randomNum].age);

}

}

// 这个 button 被 jQuery 的 $ 包装起来了，所以它变成了一个 jQuery 对象

// 下面这行会输出 undefined 因为 button 对象没有 data 属性

$("button").click(user.clickHandler);

在上面的代码中，按钮 ($("button")) 是一个对象，我们把 user.clickHandler 传入它的 click() 方法作为一个回调函数，这时候我们就明白 user.clickHandler 方法里面的 this 已经不再指向 user 这个对象了。因为 this 是定义在 user.clickHandler 方法里的，所以它现在指向那个调用了 user.clickHandler 的对象。而那个对象就是 button 对象。也就是说，user.clickHandler 将会在 button 对象的 click 方法中被执行。

注意在调用 clickHandler() 时，我们虽然写成了 user.clickHander 的形式（事实上我们必须这么写，因为 clickHandler 是在 user 对象中被定义的），但 clickHandler 还会在 button 对象的上下文中被执行，this 也因而指向了 button 对象。

讲到这里，我们应该发现当上下文发生变化的时候，换句话说就是当我们在别的对象中调用了本对象内定义的方法的时候，this 关键字就不再指向定义 this 时的那个对象了，而是指向了调用了那个 this 所在方法的对象。

解决 this 方法被当作回调函数传递时指向错误的方法：

因为我们确实想要让 this.data 指向 user 对象的 data 属性，我们可以使用 bind(), apply(), call() 这三个方法来显式地设置 this 的值。

我还写了另一篇文章，Javascript 进阶：Apply, Call 和 Bind 方法详解 来详细解释这三种方法的用法，包括如何使用它们在各种容易出错情景下正确地设置 this 的值。我就不在这里贴出整篇文章了，推荐读者详细地阅读整篇文章，因为我认为要想成为 JavaScript 的高级开发者，和这三种方法打交道是不可避免的。

为了解决上面例子提到的那种问题，我们可以使用 bind 方法：

我们把下面这行：

$("button").click(user.clickHandler);

改正为下面这样，把 clickHandler 和 user 绑定起来：

$("button").click(user.clickHandler.bind(user));

查看 JSBin 上的在线示例

2. 解决当 this 出现在闭包内遇到的问题

另一个 this 常常被误解的情景是当我们使用闭包的时候。一个非常值得注意的地方是，闭包不能直接通过使用 this 来访问外层函数的 this 变量，因为 this 变量只有当前函数本身可以访问，而其内层函数是访问不到的。举个例子：

var user = {

tournament: "The Masters",

data: [

{name: "T. Woods", age: 37},

{name: "P. Mickelson", age: 43}

],

clickHandler: function() {

// 在这里使用 this.data 是可以的，因为 this 指向 user 对象，而 data 是 user 对象的一个属性

this.data.forEach(function(person)) {

// 但是在内层匿名函数中（就是我们传给 forEach 方法的函数），this 不再指向 user 对象了

// 这个内层函数不能访问外层函数的 this 变量了

console.log("What is This referring to? " + this); //[Object Window]

console.log(person.name + " is playing at " + this.tournament);

// T. Woods is playing at undefined

// P. Mickelson is playing at undefined

});

}

}

user.clickHandler(); // 现在 this 指向什么？[object Window]

在匿名函数内部的 this 不能获得外层函数 this 的值，所以当没有使用严格模式的时候，它就被绑定在了全局 window 对象上了。

在内层函数中维持 this 的值的方法：

为了解决传入 forEach 的匿名函数中 this 值不正确的问题，我们使用一个常用的解决办法，即当我们进入 forEach 的时候，提前把 this 的值存到另一个变量中去。

var user = {

tournament: "The Masters",

data: [

{name: "T. Woods", age: 37},

{name: "P. Mickelson", age: 43}

],

clickHandler: function(event) {

// 为了当 this 还指向 user 对象的时候把它的值保存下来，我们把它存到另一个变量中

// 我们把 this 保存到 theUserObj 变量中去，这样我们就可以在之后使用了

var theUserObj = this;

this.data.forEach(function(person) {

// 我们将 this.tournament 替换成 theUserObj.tournament

console.log(person.name + " is playing at " + theUserObj.tournament);

});

}

}

user.clickHandler();

// T. Woods is playing at The Masters

// P. Mickelson is playing at The Masters

值得注意的是，许多 JavaScript 开发者喜欢把 this 存在一个叫做 that 的变量中（就像下面的代码那样）。我觉得用 that 来命名使用的时候十分不方便，所以尽量使用一个合适的名词来描述 this 所指向的对象，所以我在上述代码中使用了 var theUserObj = this。

// 一种十分常见的写法

var that = this;

查看 JSBin 上的在线示例

3. 解决把一个 this 方法 赋给一个变量时出现的问题

当我们把一个使用了 this 的方法赋给一个变量的时候，this 的值很可能出乎我们的意料，指向了其他的对象。我们来看一个例子：

// 这个 data 变量是一个全局变量

var data = [

{name: "Samantha", age: 12},

{name: "Alexis", age: 14}

];

var user = {

// 这个 data 变量是 user 对象的一个属性

data: [

{name: "T. Woods", age: 37},

{name: "P. Mickelson", age: 43}

],

showData: function(event) {

var randomNum = ((Math.random() \* 2 | 0) + 1) - 1; // 0 和 1 之间的随机数

// 下面这行随机打印一个 data 数组中的人的信息

console.log(this.data[randomNum].name + " " + this.data[randomNum].age);

}

}

// 把 user.showData 赋值给一个变量

var showUserData = user.showData;

// 当我们执行 showUserData 函数的时候，打印在 console 中的值来自于全局的 data 数组，而不是 user 对象的 data 属性

showUserData(); // Samantha 12 （来自全局 data 数组）

当把含有 this 的方法赋值给一个变量时维持 this 的值的方法

我们可以使用 bind 方法来显式地设置 this 的值来解决这个问题：

// 把 showData 方法和 user 对象绑定起来

var showUserData = user.showData.bind(user);

// 现在我们可以从 user 对象中获取值了，因为 this 关键字和 user 对象绑定在一起了

showUserData(); // P. Mickelson 43

4. 解决当借用方法的时候 this 的值不正确的问题

在 JavaScript 开发中，借用方法（borrow methods）是一个很常见的用法，作为一个 JavaScript 开发者，我们肯定会在实践中不断地遇到这个问题。而且每次我们也乐于使用这种节约时间的方法。如果你想了解更多关于方法借用的问题，请阅读我的这篇详细解析的文章，Javascript 进阶：Apply, Call 和 Bind 方法详解。

让我们来看看当处于借用方法这样的上下文的时候，this 的相关表现：

// 我们有两个对象。其中一个有一个叫做 avg() 的方法，而另一个没有

// 所以我们想借用一下 (avg()) 这个方法

var gameController = {

scores: [20, 34, 55, 46, 77],

avgScore: null,

players: [

{name: "Tommy", playerID: 987, age: 23},

{name: "Pau", playerID: 87, age: 33}

]

}

var appController = {

scores: [900, 845, 809, 950],

avgScore: null,

avg: function() {

var sumOfScores = this.scores.reduce(function(prev, cur, index, array) {

return prev + cur;

});

this.avgScore = sumOfScores / this.scores.length;

}

}

// 如果我们执行下面的代码，

// gameController.avgScore 属性将会被设置为 appController 对象的 scores 数组的平均数

// 不要执行下面这行代码，这只是用来说明的，而我们现在想让 appController.avgScore 保持 null 值

gameController.avgScore = appController.avg();

在 avg 方法中的 this 不会指向 gameController 对象，而会指向 appController 对象，因为它是被 appController 对象所调用的。

解决当借用方法时 this 指向出错的问题

要解决这个问题，我们只要确保在 appController.avg() 中的 this 指向 gameController 就可以了。我们可以使用 apply() 方法来实现：

// 注意我们使用的是 apply() 方法，所以第二个参数必须是一个数组，这个数组中包含了要传入 appController.avg() 的参数

appController.avg.apply(gameController, gameController.scores);

// 即使我们从 appController 对象中借用了 avg() 方法，gameController 的 avgScore 属性仍被成功地设置了

console.log(gameController.avgScore); // 46.4

// appController.avgScore 的值仍然是 null。它没有被更新，只有 gameController.avgScore 被更新了

console.log(appController.avgScore); // null

gameController 对象借用了 appController 的 avg() 方法。在 appController.avg() 中的 this 的值会被设置成 gameController 对象，因为我们把 gameController 作为第一个参数传入了 apply() 方法中。传入 apply() 方法的第一个参数会被显式地设置为 this 的值。

查看 JSBin 上的在线示例

结语

我希望你对 JavaScript 中的 this 关键字已经理解了。现在你有了必需的工具（bind, apply, call 方法，和把 this 赋给一个变量）来帮你解决在各种情形下关于 this 的问题了。

正如我们在上文中看到的，this 在有些情况下可能会变得很难以处理，比如原始的上下文（就是 this 定义的地方）发生改变的时候，尤其是在回调函数中，或者被另一个对象调用的时候，再或者是当方法借用的时候。但是只要记住 this 永远具有那个调用 this 函数 的对象的值，就不会出错。

享受生活，享受代码。