



JavaScript Is Sexy

Master JavaScript and Frontend Development



Understand JavaScript's "this" With Clarity, and Master It

By
Richard Bovell

Posted on
July 5, 2013 (July 5, 2013)

(Also learn all the scenarios when *this* is most misunderstood.)

Prerequisite: A bit of JavaScript.

Duration: about 40 minutes.

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.

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."

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 sent  
    // ence earlier?:  
    console.log(this.firstName + " " + this.lastName);  
    // We could have also written this:  
    console.log(person.firstName + " " + person.lastName);  
  }  
}
```

In Collaboration With:



Become an Elite, Highly Paid, Specialized Engineer. Frontend, Backend, AI, Blockchain, Mobile, etc.

By the founder of JavascriptIsSexy

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 o
n the apply () method later.
// "this" gets the value of whichever object invokes the "this" Functio
n, 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*:

1. 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](#)

2. **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](#)

3. 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 this keyword is bound to the user object
showUserData (); // P. Mickelson 43
```

4. **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 () th
at 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 averag
e 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 argumen  
t 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 gameControll  
er object, even though we borrowed the avg () method from the appCont  
roller 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.

TAGGED IN

Beginner JavaScript

Intermediate Javascript

JavaScript this

JavaScript Tips

By Richard Bovell

COMMENTS

Pingback: Apply, Call, and Bind Methods are Essential for JavaScript Professionals; Understand Them Well | JavaScript is Sexy

Pingback: Understand JavaScript's "this" &ldquo...

Pingback: Understand JavaScript's "this" &ldquo...



Malwat
6 years ago

I think I can say i fully understand how JS this now. Good job, Man. Great article.



Richard Bovell AUTHOR
6 years ago

Great. I am happy to hear that.



JF
6 years ago

Nice article, very well exposed.
Thank you !

Pingback: 16 JavaScript Concepts JavaScript Professionals Must Know Well | JavaScript is Sexy



JCh
6 years ago

Very nice article, your examples are straightforward, thanks for helping me understand JS better.
PS the site's overall design also makes it a breeze to read.



Richard Bovell AUTHOR
6 years ago

Thanks, JCh, and I'm happy to hear the formatting and the article are helpful.



Jian
6 years ago

Thanks man, I am a junior JS developer, this helps me to become a pro 😊



Richard Bovell AUTHOR
6 years ago

Sweet!



Julian Kigwana
6 years ago

Thank you for all your the time and effort in explaining this so clearly, you are a great teacher. I am dyslexic and often need to resort to video tutorials to understand such concepts fully. But you make this easy. I shall try to find the time to read the rest of your site.



Richard Bovell AUTHOR
6 years ago

You made my day, Julian. I am very happy that this post has helped you understand the topic very well. I know how difficult it is sometimes for learn technical stuff.



MHD
6 years ago

You are Great Developer, I am understand this keyword completely, thanks



Richard Bovell AUTHOR
6 years ago

Thanks, MHD, and it is very encouraging to hear that the article helped you understand JavaScript's "this."



Mike
6 years ago

thanks Richard. This article is just awesome.



MuthuGanapathyNathan
6 years ago

Really what an explanation. I am seeing to buy book for javascript, but after seeing your blog, I have to rethink my decision. Great Explanation.!!!

Thank you.



Richard Bovell AUTHOR
6 years ago

You will likely still need a JS book, but you are correct, I do cover quite a bit of JS topics here on the blog. Thanks for kind words.



CR
6 years ago

Thanks for the Post! Very interesting and gr8 article!



Nick
6 years ago

Great article, Richard.

Noticed one thing in the last example, though:

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

Can be rewritten without the second argument as:

```
appController.avg.apply(gameController);
```

Since 1) avg() doesn't take any parameters and 2) avg() doesn't reference the arguments pseudo array. You're already referring to the correct this.scores.reduce, anyway.



Mike
5 years ago

Thanks for pointing this out. Makes more sense now.

Pingback: Beautiful JavaScript: Easily Create Chainable (Cascading) Methods for Expressiveness | JavaScript is Sexy



Santosh Marigowda
6 years ago

Very good article Thank you for your time and effort.



Richard Bovell AUTHOR
6 years ago

You got it, Santosh. Thanks.



Sam
6 years ago

Hi Richard,

Thanks for your great article! It has helped me a lot to learn javascript.

When I changed your code sample for callback as following, calling `user.clickHandler()` in a anonymous callback, it works well.

```
$(".buttonError").click (function(event) {  
  user.clickHandler();  
})  
); // undefined
```

Can you shed the light how it also works well?

Thanks in advance!



Richard Bovell AUTHOR
6 years ago

Excellent question, Sam. You are correct that using an anonymous function inside the jQuery's click method works correctly.

The reason it works correctly is because code executed inside the anonymous function is executed in the global scope, not in the scope of the containing function (Not bound to the `click()` method of the button object). And because the `user` object invokes the `clickHandler()` method, the "this" keyword will have the value of the `user` object. So "this" will correctly get the `this.data` property inside the `user` object.



mp
4 years ago

we are just passing `user.clickHandler`, not executing (like assigning) the example above....but in anonymous function you are calling `user.clickHandler()`;



Jason
6 years ago

Would "this" have to be set with the `bind` method if, in your example above, you used vanilla JavaScript instead of jQuery?



Richard Bovell AUTHOR
6 years ago

Hi Jason,
Can you post the example code from the article to which you are referring? Thanks.



Jason
6 years ago

It's your example #1 above. It's not just a jQuery thing – even using regular, plain JavaScript, the browser sets 'this' to the element which fires the event. Could this be done with `call` or `apply` instead of `bind`? If so, how?



Richard Bovell AUTHOR
6 years ago

Yes, you can use the `call ()` or `apply ()` methods instead of `bind` set "this". Here is an example:
`user.clickHandler.call (user);`

See the article below for much more examples of setting the "this" value with Call or Apply:

JavaScript's Apply, Call, and Bind Methods are Essential for JavaScript Professionals



Shrichand Gupta
6 years ago

This seems the article I was looking for, Concept to demonstrate THIS is excellent, now I am feeling I understand this. "this" contains value in it of the "invoker" object not of that in which "this" was defined. Great Job !! Thank you very much.



Richard Bovell AUTHOR
6 years ago

I am very happy to hear that the article has helped you totally understand the "this" concept in JavaScript.



Abhijit Mallik
6 years ago

Thanks Richard.No words to say.Just speechless.Article is just so much descriptive and well versed which really clarified my doubts .I have been looking for "this keyword" concept but failed but your article is just awesome.



Richard Bovell AUTHOR
6 years ago

Hello Abhijit,
You made my day 😊

Thanks for the kind compliment and, most important, I am very happy to hear that the articles are helping you to better understand some of the JavaScript's core concepts.



yougen
6 years ago

Hi Richard, I have two question about this:

1. I have following code, this is the callback case:

```
var city = "OuterCity";

var myNav = {
  city: 'InnerCity',
  goDestination:function(callback){
    callback(this.city);
  }
}

myNav.goDestination(function anoy(city){
  console.log(this);
  console.log("go to ", this.city);
});
```

in the goDestination(), callback is invoked by window(suppose no strict mode)? Actually is window.callback()?

2. In "Fix this inside closure" part, this.data.forEach(function(){...}), the inner anonymous function context is always global(in browser, namely window?) Suppose we call anonymous function as "callback", in the forEach, always like window.callback?

Appreciate for your article.



Richard Bovell

AUTHOR

6 years ago

For number 1:

```
var myNav = {
  city: "InnerCity",
  goDestination:function(callback){
    callback("The myNav City: " + this.city);
  }
}

myNav.goDestination(function (aCity){
  console.log("What is this: " + this);
  console.log("go to THIS.City: " + this.city);
  console.log("Argument Passed to callBack: " + aCity);
});
```

Here are the results:

```
// What is this: [object Window]
// go to THIS.City: undefined
// Argument Passed to callBack: The myNav City: InnerCity
```

Indeed the anonymous callback function is invoked on the Window object, so "this" is the Windows object. But as you can see in my modified code above, you can get the correct city info from the myNav object by simply allowing the callback to print the parameter to the console.

2. I don't think I completely understand your second question, but I think the example in number 1 above will answer it.



yougen

6 years ago

Thanks for your answer, "anonymous callback function is invoked on the Window object", it has already answer the second question. Sorry for the indistinct question.



Minh Nguyen

5 years ago

All I want to say is: Awesome work!!!. It helps me so much! Thanks you!



Richard Bovell

AUTHOR

5 years ago

I am always very grateful to receive compliments, so thank you, Minh. And much success to you as you master JavaScript.



Subhan

5 years ago

Great tuts on this keyword, very good explained and covered almost all scenarios for this that is very common in scale-able and modern JavaScript.

Thanks a lot for it

Pingback: [this context bind apply call](#) | johngslater

Hari



5 years ago

An excellent article for those who want to learn and master in Javascript.



Sean

5 years ago

Hey Richard, awesome clarity on using the "this" within Javascript..Quick question, regarding the last example:

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

How can we explicitly use or update the appController object again, to get the property of appController.avgScore, or even start to use "this" again within the appController object. Once we borrow the actual method/or object within it, do we loose access to the "this"/object.. Is there a way to regain/update the "this"/object again... If that makes any sense???.. Thanks



Richard Of Stanley

AUTHOR

5 years ago

Hi Sean,

Since we are just borrowing the appController method, everything remains intact for the appController object. So you can use "this" and access the appController.avgScore property as normal.

I am not sure I answered your question. Please let me know if I didn't, and I will try to respond promptly with a follow up.



Andreas C

5 years ago

I just want to thank you for all the hard work you must have put in to writing these tutorials and articles. I found this blog a couple of days ago and I just got started reading through the material here.

When I did the Javascript course on Codecademy some time ago I tried to do exactly whats covered in number 3 in this article, using "this" in a method that I assigned to a variable and was confused why it didn't work. Now I know. 😊



John

5 years ago

I posted a question on stackoverflow regarding this article and specifically how "this" works inside closures. Seems like I am still none the wiser.

<http://stackoverflow.com/questions/20242351/this-inside-a-closure-function>



John
5 years ago

My conclusion is that the points you make in "1. Fix this when used in a method passed as a callback" are incorrect.

See this JSFIDDLE – <http://jsfiddle.net/EqHJ7/>

You mention several times that this refers to button object now because button object was the one invoking it. However, you are wrong because this is only referring to button object because jquery automatically binds "this" to button object (using either `call` or `apply`). Otherwise, the "this" would point to the window object since the "this" function is being called by another function (`jquery.click`) and as we know, nested functions or closures point to window object or undefined (depending on strict mode).

So your following statements are incorrect.

"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."



Arjun
5 years ago

In,
// 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);`

I do not understand, Why we need to pass `gameController.scores` array? It works without it! I tried it on JSbin.



Todd
5 years ago

I was thinking the same thing Arjun. The avg function wasnt requesting arguments.



Chad
5 years ago

I cannot possibly emphasize enough how excited I am to have found your JS blog. Words simply escape me.

As a Java Developer, I had been struggling in the world of JavaScript due to the nuances surrounding "this". The source of my agony was because, in it's most basic usage, "this" performs how I would have expected; thus, resulting in a false understanding of it. However, that false understanding would be exposed the moment more complex code utilized "this". In a nutshell, I found myself in a programming paradox where, as it seemed, a piece of code performed one way in one place, yet the exact same piece of code performed completely differently in another place, and I could not understand why. Thanks to your tutorial, not only did my lack of understanding become completely obvious, but it now all "makes sense" – which is priceless and essential attribute for any programmer.

Your post allowed me to understand this subject more than any other tutorial I have read online, and I simply cannot thank you enough for it. I truly appreciate the time, effort, and your talent in explaining this previously frustrating topic.



zack
5 years ago

I think this one of the most important and one of the best webpage I've ever read in my entire life. I'm almost indebted to you. I have been struggling with this "this" enigma, until it started to unravel before me right here.



angel
5 years ago

Richard thanks for the article...I'm not pretty sure with the closure case...this would not be a callback to? is a function which is called inside the map method from array object

why this

[1,2,3].map(function(x){console.log(this)}) the this is the global or windows while

while

\$("#button").click(funthis....) this is the button

both are pretty similar in structure...why in one ref to global and in the other refers to button
thanks



Laker
4 years ago

Heya, I'm no expert or anything, but from what I've read here it's simply because the first example is plain javascript while the second one is jquery.

So in the first one you are using anonymous function, which in plain JS (without strict mode) refers to the global, or window object.

In the second example you are using jquery, which automatically assigns this to jquery object, in this case the button.

Hope it helps... a year later ... :/



tc
5 years ago

awesome, thanks! just what i needed



Colin
5 years ago

Great article, but I am a bit confused by the last example. Do you need the second argument to apply? The function doesn't take any extra parameters, as some other people mentioned.



Binh Thanh Nguyen
5 years ago

Thanks, nice post



Tarak
5 years ago

Hats Off to the great teacher "Richard"



froi
5 years ago

Thank you! Great Article.. very helpful! 😊



Johan
5 years ago

Best javascript article i have ever seen in my life . it's only you who explained it well or i was stupid !!!!
😊



Sunil Gautam
5 years ago

"this" is this best article on this :).



Bill
5 years ago

```
function myFunction(this)
{
  alert(this.id);
}
```

This is a paragraph.

Hi, Richard, first of all, I want to thank you for the amazing posts, and second, I have an issue hoping you could help me get this clarified. As you can see the code above, I used an inline event handler with the this keyword, but when I click on the paragraph, it won't return the id at all nor does anything, can you please help me with this? I am really struggling with event arguments right now, even want to cry : (. Thank you very much in advance Richard



Bill
5 years ago

```
function myFunction(this)
{
  alert(this.id);
}
```

Click on a paragraph. An alert box will alert the element that triggered the event

for some reason the first comment cut off my code.



bill
5 years ago

Hi Richard, great article sir, by the way, I saw you mentioned "global function", could you please tell me how to determine the scope of a function? Thank you very much.



yet
5 years ago

Thanks for connecting the dots in my understanding of this... One important question, how can you test the value of this if lost as to what invoke it?....



bill
5 years ago

Great website, keep writing more please.

Yours:

```
var randomNum = ((Math.random () * 2 | 0) + 1) - 1;
```

Why not?:

```
var randomNum = (Math.random () * 2 | 0);
```

Going through everyone of your examples,



Bhargav
5 years ago

Great article. Great content. Very useful day-to-day practical examples. Keep up the good work.



Yoni
5 years ago

Great post man! it really helped me get a grip of this



Ordirobo
5 years ago

Great article! It really helps me get out the mystery of this. Wish to see more!



Seshu Vuggina
4 years ago

Great article! It really helps me alot



vivek pothagoni
4 years ago

Great Job!!! I am so confused with 'this'. After reading the article, I when back to code where I had problem. Easily I was able to fix the problem.



Ranjith Raghavan
4 years ago

this is a great article. Thanks

By the way, ever considered teaching grammar to JavaScript devs? 😊



mug896
4 years ago

in last example " appController.avg.apply (gameController, gameController.scores); "

apply() is not needed just bind() enough like this

appController.avg.bind(gameController());



Michael
4 years ago

Hi-

Judging by the comments, I must be missing something. When I paste the code in the very first example into the console in Firebug and run it, in the console log I get: "undefined" Shouldn't it be outputting the first and last name twice?

Thanks,

Michael



Pavel Demeshchik

4 years ago

Great article, good job Richard!



TylerF.

4 years ago

Good article. I don't think I fully comprehend everything but it is definitely informative.



Richard AUTHOR

4 years ago

TylerF, Let me know what specifically you don't understand. I will try to clarify, and I may even add the clarification to the article, so that the article is more comprehensible.



Rafi

4 years ago

First i read your article on "context of JavaScript" and that was loud and clear and found out that you have other articles on JavaScript and started reading all, all of the are same loud and clear.

Pingback: [How to Learn JavaScript Properly | JavaScript is Sexy](#)

Pingback: Anonymous

Pingback: [FRAGE] Aufruf von Function klappt nicht



Huy Son

4 years ago

I think I found a mistake for the borrowing methods problem.

We only need to do :

`appController.avg.apply (gameController);`

Otherwise, the article is great.

I will read more !



Zim

4 years ago

Thanks. i have been working (kind of) successfully in the web dev arena for going on 15 years, using javascript in the front end. A good friend/colleague described it as the splatter gun approach. I had no formal training. I am finally beginning to understanding what javascript actually is and how it is put together. I blame ridiculous deadlines for this 😊

It is liberating for me. And these articles are REALLY REALLY helping. So just wanted to say thanks for doing this. Big respect for sharing your knowledge

Pingback: How to: JavaScript "this" keyword | SevenNet



Muddassir

4 years ago

Hi Richard,

You are awesome buddy, i learnt most of the fundamental concepts from your tutorials.

just checking Any update on the new courses, any tentative dates or anything, i am really excited for them.



Venkatesh Kumar

4 years ago

Guys, Beware before you use any event handlers with bind. You cannot unbind until you have assigned the binded function as callback.

Check out this link

[http://msdn.microsoft.com/en-in/library/ie/dn741342\(v=vs.94\).aspx](http://msdn.microsoft.com/en-in/library/ie/dn741342(v=vs.94).aspx)



ritika

4 years ago

Richard, you are a star!!! extremely helpful and well written concept.

Pingback: Solution: JavaScript "this" keyword #dev #it #computers | Technical information for you



Joynal Abedin
4 years ago

Awesome article i ever seen.



Joynal Abedin
4 years ago

Awesome article i ever seen.



Ko
4 years ago

Where are indentations in the code!! so hard to read



Mahavir
4 years ago

Awesome Article...!! Worth Reading...!!



Leandro
4 years ago

Great article and nice read. Thank you.



Gytis
4 years ago

this was a long read.. But very useful 😊



kedar parikh
4 years ago

Great article richard !! very well written, simple language and easy to understand... helped a lot in clearing all the things about this..

Thanx again 😊



Santosh

4 years ago

Great article. thank you

Santosh



gheorghe

4 years ago

I've just wrote like that

```
appController.avg.apply(gameController);
```

and it works without the second parameter, gameController.scores....

How that?



Jason Dufair

4 years ago

"Referent" is the target of a reference. "this" is a reference, not a referent.



David

4 years ago

Great job Richard! After reading this article, I can unequivocally say that I will be reading many of your other posts in the future. I am currently helping a set of students work through "this" and your article will prove an invaluable resource for both myself and the students with whom I work. There is only one change I would posit as a possible improvement. Perhaps within the context of the code snippets, you could indent lines of code to make it explicitly clear where outer functions begin and end. Just a small thought, but overall, your post remains one of the most concise and clear explanations of this traditionally tricky subject. Thanks again!



Mike Joyce

4 years ago

Excellent article. Your explanations are very clear and easy to understand. Thanks!



Rohit
4 years ago

Great Tutorial ! It's very helpful.

Thanks buddy



Bart Van Audenhove
4 years ago

Great article!

I have a question, concerning paragraph "The use of this in the global scope".

Do I understand correctly that in the example, the following:

```
this.showFullName ();
```

would produce the same result as

```
window.showFullName (); // Peter Ally
```

Maybe it would be nice to add this to the example, it would make it even clearer, I think...



Frank Topel
4 years ago

Thanks alot for your very useful articles! Readability of the source code sections could be greatly improved if you could indent stuff, though. Is that by any means possible?

Btw, your input fields above this comment field are completely unlabelled (Current Chrome/Win 7).



Vishal
4 years ago

The Fix this inside closure could have been done by passing this into the forEach method as the thisArg

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/forEach.

Pingback: Javascript this 关键字 – 剑客|关注科技互联网



Abhishek
4 years ago

Thanks for such an awesome article on JavaScript's this keyword!

Pingback: JavaScript Concepts we should know well |



RMD
4 years ago

In your example #4, you have the following code to call the "avg" method on the appController and have it be applied to the gameController:

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

However, I don't understand what you are passing in the gameController.scores, seeing as the avg() method does not take any parameters. Wouldn't it be enough to simply do:

```
appController.avg.apply (gameController);
```



Frank Topel
4 years ago

You might want to read <http://javascriptissexy.com/javascript-apply-call-and-bind-methods-are-essential-for-javascript-professionals/> to better understand what Class.Method.apply(MyObject) does.

Pingback: Program Outline | Tilly Codes

Pingback: 译文：理解并掌握 JavaScript 中 this 的用法 – 热前端

Pingback: Javascript | Pearltrees



snehashis
4 years ago

The Explanations are awesome and really helpful, but some of the solutions when tested under "use strict" is throwing error, is it advisable to consider them?



Khushboo
4 years ago

Most clear explanation I've got yet. I am so happy to find this site. Very helpful.

Thank you so much.



Khushi

4 years ago

Awesome! very well explained..liked the examples...got all the doubts cleared 😊



DY

4 years ago

Thanks for the simple and concise yet in depth explanation. Think my brain grew from reading this.
Looking forward to more posts from you



Marc

4 years ago

Great work!
Looking forward for new articles.



Cody

4 years ago

Thanks for another great article. I've been going through you collection of articles for the last week and a half now and have been finding them very enlightening. Keep up the great work 😊



Chrstronomy

4 years ago

Thanks so much for this article! It's been tremendously helpful to me and I'm sure it will continue to do so for a long time to come. Very well written and very clearly explained!

Pingback: JS interesting articles | Pearltrees



kevin

4 years ago

I am so grateful to have a teacher like you to break these concepts down to an accessible fundamental level. Thanks.



Azizul Haque
4 years ago

Very helpful and explained so clearly. JavaScript this is now clear to me.



programmer
4 years ago

Javascript is the worst language one can imagine... all the socalled features are actually workaround to the faulty design. The "handling" of "this" is most blatant example.



FearOfTheDark
4 years ago

Great post .

But in the case "Fix this when method is assigned to a variable" . You have to correct a mistake . The var

```
var data = [  
{name:"Samantha", age:12},  
{name:"Alexis", age:14}  
];
```

You need to change to that

```
var var1={  
data:[  
{name:"Samantha", age:12},  
{name:"Alexis", age:14}  
]  
}
```



Elvis
4 years ago

This is really an awesome article! Well done.



Bill Christo

4 years ago

In reference to the section titled: "The use of this in the global scope".

In your below snippet of code, you comment that "this" inside the function will have the value of the the window object but that is not true. `this.firstName` and `this.lastName` will be undefined. "this" will refer to each instance of `showFullName`, not the global scope.

```
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);  
}
```



Bill Christo

4 years ago

Correction to my previous post. Your article is in fact correct. Sorry my mistake. I didn't read the rest of the code where you are running the function without assigning it to a variable. Probably because I would never do that 😊

So, yes, when a function is called without an owner object, the value of "this" becomes the global object.



Bruno

4 years ago

Wonderful tutorial! I have one thing I didn't understand, though:

So, in `var obj` has a concrete function, this is applicable and refers to the `obj`.

BUT if we have an anonymous function(no matter how deep), this WILL NOT refer to `obj`.

Correct?

----- In your example here:

```
var user = {  
  tournament:"The Masters",  
  data :[  
    {name:"T. Woods", age:37},  
    {name:"P. Mickelson", age:43}  
  ],  
}
```

```
clickHandler:function () {  
  this.data.forEach (function (person) {
```



```
console.log ("What is This referring to? " + this);  
console.log (person.name + " is playing at " + this.tournament);  
})  
}  
}
```

(function (person) {.....} is an anonymous function. But if somehow clickHandler was an anonymous function, would that mean that 'this' would not be accessible in it?



Bruno
4 years ago

Wonderful tutorial! I have one thing I didn't understand, though:
So, in var obj has a concrete function, this is applicable and refers to the obj.
BUT if we have an anonymous function(no matter how deep), this WILL NOT refer to obj.
Correct?

----- In your example here:

```
var user = {  
  tournament:"The Masters",  
  data :[  
    {name:"T. Woods", age:37},  
    {name:"P. Mickelson", age:43}  
  ],  

```

```
  clickHandler:function () {  
    this.data.forEach (function (person) {  
  
      console.log ("What is This referring to? " + this);  
      console.log (person.name + " is playing at " + this.tournament);  
    })  
  }  
}
```

(function (person) {.....} is an anonymous function. But if somehow clickHandler was an anonymous function, would that mean that 'this' would not be accessible in it?



Neil
4 years ago

Your explanations and examples about "this" are the best I've found on the whole internet. I really enjoyed it and I can now call myself as a 'this warrior'!!! 😊



Anil
4 years ago

Thank you a lot Sir, for the thorough explanation.
This site helps me a lot in learning javascript



Dana
3 years ago

Great article! thanks 😊



shaune
3 years ago

Thanks for the article, I like how you have put the effort in to try cover all bases and clear everything up regarding this!

Pingback: [Preparing Before Hack Reactor Begins | bash \\$ cat bitchblog](#)



hendrik
3 years ago

Well done.
You are a natural good teacher!



Ian
3 years ago

In example 4 you put:

```
// Don't run this code, for it is just for illustration; we want the appController.avgScore to remain null
gameController.avgScore = appController.avg();
```

But that example is assigning the result of the function, not the function itself.

Isn't it easier and more readable to simply assign the function to the object rather than use apply? The following both return the same result

```
console.log("assigning the function")
gameController.avg = appController.avg;
gameController.avg()
```

```
console.log(gameController.avgScore); //46.4  
  
console.log("using apply")  
appController.avg.apply (gameController, gameController.scores);  
console.log(gameController.avgScore); //46.4
```

Pingback: [How to know what the keyword "this" refers to at all times](#) | Hera Kim



AaronChen
3 years ago

Hi Richard,

Thanks for your post, and it is really helpful, and I got a question about the borrow part. It seems if I don't pass the gameController.scores. I mean the function is like this: appController.avg.apply (gameController); I think it still works.

BTW, Do you mind if I translate your article into Chinese?



Fran
3 years ago

Thanks for this tutorial. Very useful



Nad
3 years ago

Thank you. Great, clear and useful.

Pingback: [Javascript's This](#) | Brian Mesa



Janne
3 years ago

Your article was exactly what I needed, thanks.



Robert
3 years ago

You covered all the complexities of "this" but didn't mention the most basic thing. That fact that in a form element, "this" refers to that element. For example, suppose you want a blank text entry box to always be set to zero. Well, just add this HTML attribute: `onblur="if (this.value == '') this.value = 0"`



Aakash
3 years ago

Nice job man. Keep it up..(Y)



Noopur
3 years ago

Very well written..Great Expanation!

Pingback: Understanding 'this' in Javascript – Jagdeep Singh Bisht

Pingback: HackReactor's Admissions Challenge | FunStackDeveloper.com



Tom
3 years ago

Very helpful and well written, thanks!



Artem
3 years ago

Great work, sir. It's the most exhaustive article I've ever encountered on the subject (paired with your another article about Apply, Call, Bind). Thanks a lot. I belive, 'this' keyword in JS is far more clearer for me, since I've read this.

Pingback: JavaScript's "this" Foils an OO Guy | My name is Ozymandias

Pingback: angularjs - When is this needed in JavaScript / Angular - CSS PHP



Nalinda
3 years ago

Great article, reading this I can say I know this :). Thanks for all your efforts, one of the best when it comes to JS stuff.



M Azher
3 years ago

Great article. Brilliant piece to demystify "this".



eng
3 years ago

Your explanation is impressive. You have really helped me in my difficult transition from C++ to Javascript. Thanks



Monika
3 years ago

Excellent!!
Really the best blog on "this" object I have ever read.
Thank you 😊



Alejandro
3 years ago

One thing, It may have been mentioned before.
On the "Solution for fixing this when borrowing methods" you say:
"// 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 avg() method doesn't receive any parameters, so it should really be:
appController.avg.apply (gameController);



Bah Djibril
3 years ago

Thanks for this article. Best explanation.

In the last section, since appController.avg uses this.scores (instead of the arguments) I think there is no need to pass in the array as the second parameter.

* The code below

// 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); // Old line

appController.avg.apply (gameController); or appController.avg.call (gameController);



Mikael

3 years ago

I loved this! THANK YOU! Very clear, you are a good teacher!



Joselie Castañeda

3 years ago

thank you, your explanation on the topic helps me a lot.



Motty

3 years ago

Thanks so much for a wonderful article; I have never seen this explained so well!



Ankush

3 years ago

Great Article.. I have been reading your articles from few days and to be true they are so damn good with such clear understanding . Is there any book of yours I am sure it would be an excellent source of knowledge



Madan Neelapu

3 years ago

Awesome Man! Great Article. Thank you.
Your Blog is on the top of my JavaScript reference List.



Pri Seja

3 years ago

Wow! Very helpful article. You did a great job, the best way to learn about "this" word. Thank you so much! 😊 Excellent!



Niyati
3 years ago

genius....
Thank u so :-O much



Alexdev
3 years ago

The article is Great! thank you for the examples and detailed explanations!



Raman Gutyan
3 years ago

Thanks for such an awesome post on JavaScript's this keyword!



Steven Winston
3 years ago

In the last solution why did you pass in gameController.scores to the apply() method?

I found without it it works fine as the avg function takes no arguments.

<http://jsbin.com/naqaqonati/edit?html,console>

Did I miss something here in my understanding?

PS Great explanations, I'll use your blog again – best out there!



Ruchi Gupta
3 years ago

```
var gameController = {  
  scores :[20, 34, 55, 46, 77],  
  avgScore:null,  
  players :[  
    {name:"Tomy", 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
// The avg method "this" keyword will not refer to the gameController object, it will refer to the
appController object because it is being invoked on appController
// Don't run this code, we want the appController.avgScore to remain null

appController.avg.bind(gameController, gameController.scores);//null

// 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

console.log (appController.avgScore);
```

Hi Richard, In case of bind method, its showing null. Can you please explain this?
Great work Richard!



Sam
3 years ago

great article!

1 question though:

in the last code example, why did you have to pass 2 arguments (appController.avg.apply (gameController, gameController.scores);) when only 1 is enough to get the correct answer (gameController) ?

also i just would like to mention that the problem with jQuery button `\$('button').click(someFunc)` can be also solved using `.on()` instead of `.click()`



Ganesh
2 years ago

Very nice article.Thanks lot for your time.



Felipe
2 years ago

Greta article! Wish people would explain things as clear as you do! Thanks a lot



Han
2 years ago

Very useful stuff!



Ninad
2 years ago

Now that ES6 is here, no need of the that-this hack.
We can just use the fat arrow function and it will have access to surrounding 'this'
Note- Fat arrow functions have lexical scope!



hany
1 year ago

Thanks Man , best article i ever read about this
Thank you again



Abdifatah
1 year ago

I don't know the reason, but none of your examples is working for me. I don't know if the text editor has something to do with that; I am using sublime text editor on OSX, with chrome browser javascript enabled. By the way it shows this message when ever I retyped or copy/paste your examples:

app.js:5 Uncaught SyntaxError: Invalid or unexpected token

But, its working with other codes I typed by myself or even copied from other source including w3schools. So, any suggestion please.



Prits
1 year ago

The only article that I could find that explained this concept so thoroughly yet in such simple language. Excellent examples to follow up the theory. Thank you!



Murali
1 year ago

Thala Richard, thank you for this, it is very well explained and really helped me. By the way 'thala' is a word of endearment in my local language 'tamil' and means Leader or Boss. 😊



Haider
1 year ago

Could you make the indentation on the code samples accurate? It's a bit hard to read. The sans-serif font is not as much as annoying as the indentation. Anyways, great article. I learning a lot! Thanks



Pedro
1 year ago

Great article.
You should keep post more articles.
Great work



Jasper Verelst
1 year ago

This is a good read. As a (Belgian) junior (more like a student actually) developer, I'm reading slowly through this article and looking things up now and then. So besides the use of 'this', I'm learning a lot of additional things as well here. And I like the expression "Things get a touch hairy" 😊 never heard of it before. Guess what I'm trying to say is thanks man, that's the least I can do 😊



Richard
1 year ago

Thanks for such a nice tutorial. The explanation for "this" keyword is clear and thorough.



Suhas Mallesh

1 year ago

One of the best tutorials ever. Really appreciate the effort and hard work.



jayanth

1 year ago

best article ever in javascript



Gilbert

1 year ago

Excellent article. Thank you. Well written.

I would like to suggest a thought that may help. When you say that 'The "this" property is a variable with the value of the object that invokes the function where this is used.', you are implying something. You are implying that the object is invoking a function. That is: `myObj.myFunc()` is happening. The thing is, although that is true, when one calls a global function, one does not say `window.myGlobalFunc()`, one simply calls `myGlobalFunc()` and may not know that it is being invoked by the global object. Therefore, to some people this implication can cause confusion. For the sake of completeness, it might be helpful to add a note about that. That every function call is always invoked by an object, whether it be implicit, or explicitly called. What say you?



Marcello Scattolini

1 year ago

Thanks a lot for this useful article.

Many beginners in coding, as I am, face challenges to find straight explanations, and you totally helped.

Appreciate your dedication.



Moi

1 year ago

Thank you for your explanation, is very helpful and clear for not english people 😊



Vijay B. Kodare

1 year ago

Thanks dude, You are only one from which I got well understanding of Core Javascripts concepts like prototype, this, call, apply & bind.

In above one of the given example, there are my correction for below lines. I run the below code in JSFiddler & its gives output as 'undefined undefined'. What I understood from you is, 'this' is associated with the object instance. The direct attributes defined on object can be accessible through 'this'. In below lines firstName & lastName are not direct attributes defined on 'document' or 'window' object. These are a declared variables, so you can not access those on 'this'.

```
var firstName = "Peter", lastName = "Ally";  
function showFullName () {  
  console.log (this.firstName + " " + this.lastName);  
}
```

The above lines should be

```
this.firstName = "Peter";  
this.lastName = "Ally";  
function showFullName () {  
  console.log (this.firstName + " " + this.lastName);  
}
```



Mike

1 year ago

This is just what I needed!

Pingback: [node.js tutorial for beginners – waynes coding blog](#)



Jason

12 months ago

See, this is why I can't learn a language no matter how hard I try – it's everyone else's fault. Ok, maybe not really. But if people would explain things in a way that makes sense to me then of course I'd understand things. Consider, for example, on about page 2 of this article, which I immediately recognized as having been SPECIFICALLY written for me alone – thank you for that, there's this code example:

```
var person = {  
  firstName: "Penelope",  
  lastName: "Barrymore",
```

```
fullName: function () {  
  // some other ish
```

In the paragraph immediately following it, it seems you're talking about the code example just given. That would make all the sense in the world, since you already understand it and I do not. But then in the last sentence you say,

"It tells us that we are referring to the specific John at the beginning of the sentence."

Which, because you said "John" instead of "Penelope" completely undoes all that I almost understood because I now realize I don't even know to what you're referencing.

Could you have said "Penelope" there instead? If so, why on earth didn't you? Will you and everybody else willing to share these clandestine, occult secrets of the gods with us mere mortals please condescend to adhere to a "one example at a time" policy forthwith?

Thanking you!

Sincerely,

He-that-walks-in-JS-blindness-and-is-kinna-hungry-for-a-sandwich-right-now-and-has-ADHD-and...
what was I saying?

P.S. Good work though, exactly what I needed without realizing it.



Jason
12 months ago

ah God. This is so basic I'll probably be justifiably ignored and shunned and marked as an outcast so that nobody else will help me either. My problem is that when I use the Firefox Developer tool ScratchPad to enter the code (I know it's not meant as an exercise but I just wanted to):

```
var friends = ["Mike", "Stacy", "Andy", "Rick"];
```

```
friends.forEach(function (eachName, index){  
  console.log(index + 1 + ". " + eachName); // 1. Mike, 2. Stacy, 3. Andy, 4. Rick  
});
```

...ScratchPad gives me no feedback, and there's none in the Console window either. In the Console window there's a lot of irrelevant & incomprehensible code that apparently originates from the client/server interaction of the the page I happen to be on, but nothing that looks like a result of anything I'm actually trying to run like the code above.

I know it's something very trivial & basic, sorry.



Pooja
12 months ago

Excellent tutorial!



Balasubramani M

10 months ago

Finally, I understood, apply, bind and call methods in JavaScript. Thanks a lot.



Jagadish

7 months ago

In method to access the outer functions this inside the for loop, you can use Arrow function also. You can update the post with this new method.

I am sure you know this 😊 but just reminding you.

Nice Post. Keep Posting :).



Maharjun

6 months ago

good article, Well explained. In the last example we no need to pass the extra argument. It will work fine without that extra argument. as we are not using any parameter for the method 😊



Swagnik Dutta

4 months ago

Excellent article, thanks for the effort you have put in. Never before I have come across such a detailed explanation (with effective examples). I feel more confident now. 😊



Alexandar

4 months ago

This is probably the best article I've ever read on "this" topic! But one thing that is painful while reading is that code examples while reading. Some styles regarding code examples are welcome!



bhupesh

4 months ago

Stopped reading article the moment I saw "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.". Please update the article



Sunho

2 months ago

Thanks for nice article.

Clearly the best article on the net about damm this keyword in javascript.

Thanks again!



eric

2 months ago

By definition, "unambiguous" means that it doesn't take a lengthy article to explain the value to which it occurs. 🙄 To carry on the pronoun metaphor, `_this_` unambiguously refers the antecedent, or the printer, or the book it's in, or the window object of the website where the pronoun is written. 🙄

I really appreciate the article, and don't want to seem like I'm criticizing the author.

This was just a bad design decision for a language never intended to be used for full applications that we're still wrangling with almost 30 years later. There is really no reason that the magic value could not simply have been a parameter and leave `_this_` to the enclosing scope.

That said, I've largely abandoned the ``function`` keyword altogether in favor of the much more predictable behavior of arrow functions. It's... well, more ``function``al. 🙄