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

JULY. 5 2013 172

menu

(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:

Table of Contents

```
    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 "h 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 r
// "this" will have the value of the person object because the person object will invoke showFullNam
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 la
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 in
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.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 sho
// 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 () metho
// "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.showFullN

```

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 th
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

this.data.forEach (function (person) {
// But here inside the anonymous function (that we pass to the forEach method), "this" no longe
// 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 va
// 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
//
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
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 () 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 appControlle

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

// The avgScore property was successfully set on the gameController object, even though we bor
console.log (gameController.avgScore); // 46.4

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

```
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.

Posted in: 16 Important JavaScript Concepts, JavaScript / Tagged: Beginner JavaScript, Intermediate Javascript, javaScript this, JavaScript Tips

Richard

Thanks for your time; please come back soon. Email me here: javascriptissexy at gmail email, or use the contact form.

172 Comments

Malwat

July 10, 2013 at 2:54 am / Reply

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

Richard Bovell (Author)

July 11, 2013 at 8:07 pm / Reply

Great. I am happy to hear that.

JF

July 10, 2013 at 9:44 am / Reply

Nice article, very well exposed.
Thank you !

JCh

July 13, 2013 at 3:26 am / Reply

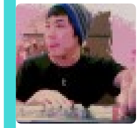
Very nice article, your examples are straightforward, thanks for helping me understand JS better.

PS the site's overall design also makes it a breeze to read.

Richard Bovell (Author)

July 17, 2013 at 3:08 pm / Reply

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



Jian

July 19, 2013 at 5:46 am / Reply

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

Richard Bovell (Author)

July 31, 2013 at 12:43 am / Reply

Sweet!

Julian Kigwana

July 22, 2013 at 12:37 pm / Reply

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)

July 31, 2013 at 1:11 am / Reply

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

July 28, 2013 at 6:47 pm / Reply

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

Richard Bovell (Author)

July 31, 2013 at 1:15 am / Reply

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

Mike

August 2, 2013 at 11:43 am / Reply

thanks Richard. This article is just awesome.

MuthuGanapathyNathan

August 3, 2013 at 3:14 am / Reply

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)

August 4, 2013 at 9:32 pm / Reply

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

August 8, 2013 at 7:30 am / Reply

Thanks for the Post! Very interesting and gr8 article!

Nick

August 9, 2013 at 9:18 pm / Reply

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

April 24, 2014 at 2:12 pm / Reply

Thanks for pointing this out. Makes more sense now.

Santosh Marigowda

August 13, 2013 at 5:36 pm / Reply

Very good article Thank you for your time and effort.

Richard Bovell (Author)

August 15, 2013 at 2:21 pm / Reply

You got it, Santosh. Thanks.

Sam

August 15, 2013 at 2:56 am / Reply

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)

August 15, 2013 at 3:08 pm / Reply

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

May 15, 2015 at 5:37 pm / Reply

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

Jason

August 25, 2013 at 5:31 am / Reply

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)

August 26, 2013 at 5:39 pm / Reply

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

Jason

August 26, 2013 at 10:30 pm / Reply

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)

August 28, 2013 at 12:53 pm / Reply

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
September 2, 2013 at 3:47 am / Reply

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)
September 4, 2013 at 9:21 pm / Reply

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

Abhijit Mallik
September 3, 2013 at 10:09 am / Reply

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)
September 4, 2013 at 9:35 pm / Reply

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

September 5, 2013 at 11:47 pm / Reply

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)

September 8, 2013 at 2:14 pm / Reply

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

September 10, 2013 at 10:21 pm / Reply

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

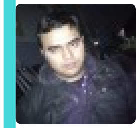
September 14, 2013 at 3:06 am / Reply

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

Richard Bovell (Author)

September 16, 2013 at 4:41 am / Reply

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



Subhan

September 25, 2013 at 2:04 am / Reply

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

Hari

October 11, 2013 at 3:22 am / Reply

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

Sean

October 12, 2013 at 5:04 pm / Reply

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)
October 16, 2013 at 1:57 am / Reply

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
November 24, 2013 at 6:52 am / Reply

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
November 27, 2013 at 9:38 am / Reply

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

November 27, 2013 at 12:18 pm / Reply

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 caal 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

December 10, 2013 at 7:06 am / Reply

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

January 12, 2014 at 10:25 pm / Reply

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

Chad

January 15, 2014 at 10:34 am / Reply

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 appre the time, effort, and your talent in explaining this previously frustrating topic.

zack

January 29, 2014 at 10:16 pm / Reply

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

February 3, 2014 at 2:49 am / Reply

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
$("buttom").click(fun …..this….) this is the buttom

both are pretty similar in structure…why in one ref to global and in the other refers to buttom

thanks

Laker

February 12, 2015 at 3:42 am / Reply

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.

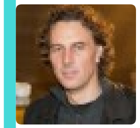
In the second example you are using jquery, which automatically assigns this to jquery object, in this case the button.

Hope it helps… a year later … :/

tc

February 11, 2014 at 12:46 am / Reply

awesome, thanks! just what i needed



Colin

March 5, 2014 at 3:13 pm / Reply

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 paramters, as some other people mentioned.

Binh Thanh Nguyen

April 27, 2014 at 11:26 pm / Reply

Thanks, nice post

Tarak

May 5, 2014 at 3:50 am / Reply

Hats Off to the great teacher "Richard"

froi

May 11, 2014 at 5:48 am / Reply

Thank you! Great Article.. very helpful! 🙂

Johan

May 14, 2014 at 3:54 am / Reply

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

Sunil Gautam

May 15, 2014 at 6:37 am / Reply

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

Bill

May 19, 2014 at 9:25 pm / Reply

```
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

May 19, 2014 at 9:43 pm / Reply

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

May 22, 2014 at 3:14 pm / Reply

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

June 4, 2014 at 2:50 pm / Reply

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

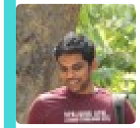
July 5, 2014 at 11:17 am / Reply

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

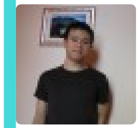
July 22, 2014 at 3:27 am / Reply

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

Yoni

August 6, 2014 at 5:07 pm / Reply

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



Ordirobo

August 11, 2014 at 10:04 pm / Reply

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

Seshu Vuggina

October 10, 2014 at 2:53 am / Reply

Great article! It really helps me alot

vivek pothagoni

October 15, 2014 at 5:47 pm / Reply

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

October 17, 2014 at 10:01 am / Reply

this is a great article. Thanks
By the way, ever considered teaching grammar to JavaScript devs? 🙂

mug896

October 19, 2014 at 8:09 am / Reply

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

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

appController.avg.bind(gameController)();

Michael

October 28, 2014 at 12:57 pm / Reply

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

October 29, 2014 at 9:52 am / Reply

Great article, good job Richard!

TylerF.

November 5, 2014 at 11:59 am / Reply

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

Richard (Author)

November 5, 2014 at 1:33 pm / Reply

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

Rafi

November 7, 2014 at 1:48 pm / Reply

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.

Huy Son

November 16, 2014 at 11:48 am / Reply

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

November 21, 2014 at 7:56 pm / Reply

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 REALY helping. So just wanted to say thanks for doing this. Big respect for sharing your knowledge

Muddassir

December 3, 2014 at 2:06 am / Reply

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

December 3, 2014 at 5:19 am / Reply

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

ritika

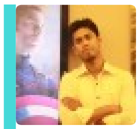
December 5, 2014 at 7:54 am / Reply

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

Joynal Abedin

January 3, 2015 at 1:15 pm / Reply

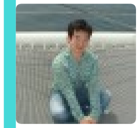
Awesome article i ever seen.



Joynal Abedin

January 3, 2015 at 1:15 pm / Reply

Awesome article i ever seen.



Ko

January 9, 2015 at 2:04 am / Reply

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

Mahavir

January 16, 2015 at 12:43 am / Reply

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

Leandro

January 22, 2015 at 3:56 pm / Reply

Great article and nice read. Thank you.

Gytis

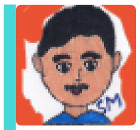
January 30, 2015 at 2:42 pm / Reply

this was a long read.. But very useful 🙂

kedar parikh

February 9, 2015 at 6:50 am / Reply

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

February 22, 2015 at 1:03 pm / Reply

Great article. thank you
Santosh

gheorghe

March 5, 2015 at 10:33 am / Reply

I`ve just wrote like that
appController.avg.apply(gameController);

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

How that?

Jason Dufair

March 6, 2015 at 11:03 pm / Reply

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

David

March 10, 2015 at 10:51 am / Reply

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

March 17, 2015 at 6:16 am / Reply

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

Rohit

March 28, 2015 at 10:09 am / Reply

Great Tutorial ! It's very helpful.

Thanks buddy

Bart Van Audenhove

April 13, 2015 at 3:49 am / Reply

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

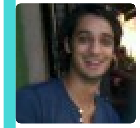
Maybe it would be nice to add this to the example, it would make it even clearer, I think…

Frank Topel

April 29, 2015 at 2:08 pm / Reply

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

May 3, 2015 at 12:22 pm / Reply

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.

Abhishek

May 14, 2015 at 5:58 am / Reply

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

RMD

May 16, 2015 at 9:54 pm / Reply

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

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

May 17, 2015 at 4:00 am / Reply

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.

snehashis

June 28, 2015 at 5:51 pm / Reply

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

Khushboo

July 7, 2015 at 9:05 am / Reply

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

Thank you so much.

Khushi

July 12, 2015 at 11:59 am / Reply

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

DY

July 14, 2015 at 4:19 pm / Reply

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

July 20, 2015 at 7:13 am / Reply

Great work!
Looking forward for new articles.

Cody

July 20, 2015 at 12:39 pm / Reply

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 🙂

Christronomy

July 23, 2015 at 10:06 am / Reply

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!

kevin

August 8, 2015 at 4:58 pm / Reply

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

Azizul Haque

August 10, 2015 at 2:40 am / Reply

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

programmer

August 10, 2015 at 6:38 am / Reply

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

August 14, 2015 at 6:47 pm / Reply

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

August 17, 2015 at 10:27 am / Reply

This is really an awesome article! Well done.

Bill Christo

August 17, 2015 at 11:23 am / Reply

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

August 19, 2015 at 3:24 pm / Reply

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

August 28, 2015 at 2:06 pm / Reply

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

August 28, 2015 at 2:06 pm / Reply

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

September 5, 2015 at 7:29 pm / Reply

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

September 7, 2015 at 6:55 am / Reply

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

Dana

September 12, 2015 at 2:11 pm / Reply

Great article! thanks 🙂

shaune

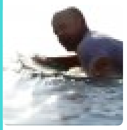
September 20, 2015 at 9:20 pm / Reply

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

hendrik

October 15, 2015 at 11:47 pm / Reply

Well done.
You are a natural good teacher!



Ian

October 22, 2015 at 4:54 pm / Reply

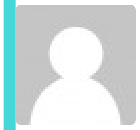
In example 4 you put:
// Don't run this code, for it is just for illustration; we want the appController.avgScore to remain null
gameController.avgScore = appController.avg();

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



AaronChen

October 23, 2015 at 5:11 am / Reply

Hi Richard,

Thanks for your post, and it is really helpful,and I got a question about the borrow part. I 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

November 11, 2015 at 6:14 am / Reply

Thanks for this tutorial. Very useful

Nad

November 13, 2015 at 3:41 am / Reply

Thank you. Great, clear and useful.

Janne

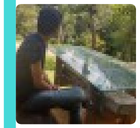
November 19, 2015 at 9:30 am / Reply

Your article was exactly what I needed, thanks.

Robert

November 23, 2015 at 10:08 pm / Reply

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

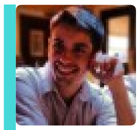
November 27, 2015 at 1:13 am / Reply

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

Noopur

November 30, 2015 at 4:24 am / Reply

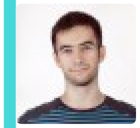
Very well written..Great Expanation!



Tom

December 31, 2015 at 3:59 pm / Reply

Very helpful and well written, thanks!



Artem

January 7, 2016 at 1:24 pm / Reply

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.

Nalinda

January 23, 2016 at 2:47 am / Reply

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

January 30, 2016 at 4:05 pm / Reply

Great article. Brilliant piece to demystify "this".

eng

March 5, 2016 at 8:18 am / Reply

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

Monika

March 11, 2016 at 12:15 pm / Reply

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

Alejandro

March 11, 2016 at 1:50 pm / Reply

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

March 15, 2016 at 11:42 pm / Reply

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

March 20, 2016 at 12:31 pm / Reply

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

Joselie Castañeda

March 26, 2016 at 3:25 am / Reply

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

Motty

March 29, 2016 at 3:49 pm / Reply

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

Ankush

April 1, 2016 at 5:26 am / Reply

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

April 4, 2016 at 9:57 pm / Reply

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

Pri Seja

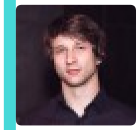
April 7, 2016 at 4:40 am / Reply

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

Niyati

April 8, 2016 at 7:33 am / Reply

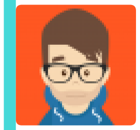
genius….
Thank u so :-O much



Alexdev

April 11, 2016 at 8:38 am / Reply

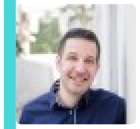
The article is Great! thank you for the examples and detailed explanations!



Raman Gutyan

April 13, 2016 at 2:25 am / Reply

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



Steven Winston

April 17, 2016 at 11:51 am / Reply

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

May 5, 2016 at 2:55 am / Reply

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

May 6, 2016 at 4:34 am / Reply

great article!
1 question though:
in the last code example, why did you had 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

November 5, 2016 at 5:15 pm / Reply

Very nice article.Thanks lot for your time.

Felipe

November 16, 2016 at 3:36 pm / Reply

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

Trackbacks for this post

1. Apply, Call, and Bind Methods are Essential for JavaScript Professionals; Understand Them Well | JavaScript is Sexy
2. Understand JavaScript's "this&rdquo...
3. Understand JavaScript's "this&rdquo...
4. 16 JavaScript Concepts JavaScript Professionals Must Know Well | JavaScript is Sexy
5. Beautiful JavaScript: Easily Create Chainable (Cascading) Methods for Expressiveness | JavaScript is Sexy
6. this context bind apply call | johngslater
7. How to Learn JavaScript Properly | JavaScript is Sexy

8. Anonymous
9. [FRAGE] Aufruf von Function klappt nicht
10. How to: JavaScript "this" keyword | SevenNet
11. Solution: JavaScript "this" keyword #dev #it #computers | Technical information for you
12. Fixed JavaScript "this" keyword #dev #it #asnwer | Good Answer
13. Javascript this 关键字 – 剑客|关注科技互联网
14. JavaScript Concepts we should know well |
15. Program Outline | Tilly Codes
16. 译文：理解并掌握 JavaScript 中 this 的用法 – 热前端
17. Javascript | Pearltrees
18. JS interesting articles | Pearltrees
19. Preparing Before Hack Reactor Begins | bash $ cat bitchblog
20. How to know what the keyword "this" refers to at all times | Hera Kim
21. Javascript's This | Brian Mesa
22. Understanding 'this' in Javascript – Jagdeep Singh Bisht
23. HackReactor's Admissions Challenge | FunStackDeveloper.com
24. JavaScript's "this" Foils an OO Guy | My name is Ozymandias
25. angularjs - When is this needed in JavaScript / Angular - CSS PHP

Leave a Reply

Name

Email (hidden)

Website

Comment:

Submit Comment

☐ Notify me of follow-up comments by email.

☐ Notify me of new posts by email.