Understand JavaScript Callback Functions and Use Them

MARCH. 4 2013 180

(Learn JavaScript Higher-order Functions, aka Callback Functions)

In JavaScript, functions are first-class objects; that is, functions are of the type *Object* and they can be used in a first-class manner like any other object (String, Array, Number, etc.) since they are in fact objects themselves. They can be "stored in variables, passed as arguments to functions, created within functions, and returned from functions"[1].

Because functions are first-class objects, we can pass a function as an argument in another function and later execute that passed-in function or even return it to be executed later. This is the essence of using callback functions in JavaScript. In the rest of this article we will learn everything about

Table of Contents

JavaScript callback functions. Callback functions are probably the most widely used functional programming technique in JavaScript, and you can find them in just

about every piece of JavaScript and jQuery code, yet they remain mysterious to many JavaScript developers. The mystery will be no more, by the time you finish reading this article.

Callback functions are derived from a programming paradigm known as functional programming. At a fundamental level, functional programming specifies the use of functions as arguments. Functional programming was—and still is, though to a much lesser extent today—seen as an esoteric technique of specially trained, master programmers.

Fortunately, the techniques of functional programming have been elucidated so that mere mortals like you and me can understand and use them with ease. One of the chief techniques in functional programming happens to be *callback functions*. As you will read shortly, implementing callback functions is as easy as passing regular variables as arguments. This technique is so simple that I wonder why it is mostly covered in advanced JavaScript topics.

What is a Callback or Higher-order Function?

A callback function, also known as a higher-order function, is a function that is passed to another function (let's call this other function "otherFunction") as a parameter, and the callback function is called (or executed) inside the otherFunction. A callback function is essentially a pattern (an established solution to a common problem), and therefore, the use of a callback function is also known as a callback pattern.

Consider this common use of a callback function in jQuery:

```
//Note that the item in the click method's parameter is a function, not a variable.
//The item is a callback function
$("#btn_1").click(function() {
```

```
  alert("Btn 1 Clicked");
});
```

As you see in the preceding example, we pass a function as a parameter to the *click* method. And the click method will call (or execute) the callback function we passed to it. This example illustrates a typical use of callback functions in JavaScript, and one widely used in jQuery.

Ruminate on this other classic example of callback functions in basic JavaScript:

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

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

Again, note the way we pass an anonymous function (a function without a name) to the forEach method as a parameter.

So far we have passed anonymous functions as a parameter to other functions or methods. Lets now understand how callbacks work before we look at more concrete examples and start making our own callback functions.

How Callback Functions Work?

We can pass functions around like variables and return them in functions and use them in other functions. When we pass a callback function as an argument to another function, we are only passing the function definition. We are not executing the function in the parameter. In other words, we aren't passing the function with the trailing pair of executing parenthesis () like we do when we are executing a function.

And since the containing function has the callback function in its parameter as a function definition, it can execute the callback anytime.

Note that the callback function is not executed immediately. It is "called back" (hence the name) at some specified point inside the containing function's body. So, even though the first jQuery example looked like this:

```
//The anonymous function is not being executed there in the parameter.
//The item is a callback function
```

```
$("#btn_1").click(function() {
  alert("Btn 1 Clicked");
});
```

the anonymous function will be called later inside the function body. Even without a name, it can still be accessed later via the *arguments* object by the containing function.

Callback Functions Are Closures

When we pass a callback function as an argument to another function, the callback is executed at some point inside the containing function's body just as if the callback were defined in the containing function. This means the callback is a closure. Read my post, Understand JavaScript Closures With Ease for more on closures. As we know, closures have access to the containing function's scope, so the callback function can access the containing functions' variables, and even the variables from the global scope.

Basic Principles when Implementing Callback Functions

While uncomplicated, callback functions have a few noteworthy principles we should be familiar with when implementing them.

Use Named OR Anonymous Functions as Callbacks

In the earlier jQuery and forEach examples, we used anonymous functions that were defined in the parameter of the containing function. That is one of the common patterns for using callback functions. Another popular pattern is to declare a named function and pass the name of that function to the parameter. Consider this:

```
// global variable
var allUserData = [];

// generic logStuff function that prints to console
function logStuff (userData) {
    if ( typeof userData === "string")
    {
        console.log(userData);
    }
    else if ( typeof userData === "object")
    {
        for (var item in userData) {
            console.log(item + ": " + userData[item]);
        }
```

```
    }

}

// A function that takes two parameters, the last one a callback function
function getInput (options, callback) {
    allUserData.push (options);
    callback (options);

}

// When we call the getInput function, we pass logStuff as a parameter.
// So logStuff will be the function that will called back (or executed) inside the g
getInput ({name:"Rich", speciality:"JavaScript"}, logStuff);
//  name: Rich
// speciality: JavaScript
```

Pass Parameters to Callback Functions

Since the callback function is just a normal function when it is executed, we can pass parameters to it. We can pass any of the containing function's properties (or global properties) as parameters to the callback function. In the preceding example, we pass *options* as a parameter to the callback function. Let's pass a global variable and a local variable:

```
//Global variable
var generalLastName = "Clinton";

function getInput (options, callback) {
    allUserData.push (options);
// Pass the global variable generalLastName to the callback function
    callback (generalLastName, options);
}
```

Make Sure Callback is a Function Before Executing It

It is always wise to check that the callback function passed in the parameter is indeed a function before calling it. Also, it is good practice to make the callback function optional.

Let's refactor the getInput function from the previous example to ensure these checks are in place.

```
function getInput(options, callback) {
    allUserData.push(options);
```

```
    // Make sure the callback is a function
    if (typeof callback === "function") {
    // Call it, since we have confirmed it is callable
        callback(options);
    }
}
```

Without the check in place, if the getInput function is called either without the callback function as a parameter or in place of a function a non-function is passed, our code will result in a runtime error.

Problem When Using Methods With The *this* Object as Callbacks

When the callback function is a method that uses the *this* object, we have to modify how we execute the callback function to preserve the *this* object context. Or else the *this* object will either point to the global window object (in the browser), if callback was passed to a global function. Or it will point to the object of the containing method.
Let's explore this in code:

```
// Define an object with some properties and a method
// We will later pass the method as a callback function to another function
var clientData = {
    id: 094545,
    fullName: "Not Set",
    // setUserName is a method on the clientData object
    setUserName: function (firstName, lastName)  {
        // this refers to the fullName property in this object
      this.fullName = firstName + " " + lastName;
    }
}

function getUserInput(firstName, lastName, callback)  {
    // Do other stuff to validate firstName/lastName here

    // Now save the names
    callback (firstName, lastName);
}
```

In the following code example, when clientData.setUserName is executed, this.fullName will not set the fullName property on the clientData object. Instead, it will set fullName on the window object, since getUserInput is a global function. This happens because the *this* object in the global function points to the window object.

```
getUserInput ("Barack", "Obama", clientData.setUserName);

console.log (clientData.fullName);// Not Set

// The fullName property was initialized on the window object
console.log (window.fullName); // Barack Obama
```

Use the Call or Apply Function To Preserve *this*

We can fix the preceding problem by using the *Call* or *Apply* function (we will discuss these in a full blog post later). For now, know that every function in JavaScript has two methods: Call and Apply. And these methods are used to set the *this* object inside the function and to pass arguments to the functions.

Call takes the value to be used as the *this* object inside the function as the first parameter, and the remaining arguments to be passed to the function are passed individually (separated by commas of course). The Apply function's first parameter is also the value to be used as the *this* object inside the function, while the last parameter is an array of values (or the *arguments* object) to pass to the function.

This sounds complex, but lets see how easy it is to use Apply or Call. To fix the problem in the previous example, we will use the Apply function thus:

```
//Note that we have added an extra parameter for the callback object, called "callb
function getUserInput(firstName, lastName, callback, callbackObj)  {
    // Do other stuff to validate name here

    // The use of the Apply function below will set the this object to be callbackOb
    callback.apply (callbackObj, [firstName, lastName]);
}
```

With the *Apply* function setting the *this* object correctly, we can now correctly execute the callback and have it set the fullName property correctly on the clientData object:

```
// We pass the clientData.setUserName method and the clientData object as parameter
getUserInput ("Barack", "Obama", clientData.setUserName, clientData);

// the fullName property on the clientData was correctly set
console.log (clientData.fullName); // Barack Obama
```

We would have also used the *Call* function, but in this case we used the *Apply* function.

Multiple Callback Functions Allowed

We can pass more than one callback functions into the parameter of a function, just like we can pass more than one variable. Here is a classic example with jQuery's AJAX function:

```
function successCallback() {
    // Do stuff before send
}

function successCallback() {
    // Do stuff if success message received
}

function completeCallback() {
    // Do stuff upon completion
}

function errorCallback() {
    // Do stuff if error received
}

$.ajax({
    url:"http://fiddle.jshell.net/favicon.png",
    success:successCallback,
    complete:completeCallback,
    error:errorCallback

});
```

"Callback Hell" Problem And Solution

In asynchronous code execution, which is simply execution of code in any order, sometimes it is common to have numerous levels of callback functions to the extent that you have code that looks like the following. The messy code below is called callback hell because of the difficulty of following the code due to the many callbacks. I took this example from the node-mongodb-native, a MongoDB driver for Node.js. [2]. The example code below is just for demonstration:

```
var p_client = new Db('integration_tests_20', new Server("127.0.0.1", 27017, {}), {
p_client.open(function(err, p_client) {
    p_client.dropDatabase(function(err, done) {
        p_client.createCollection('test_custom_key', function(err, collectior ⌟
            collection.insert({'a':1}, function(err, docs) {
```

```
                collection.find({'_id':new ObjectID("aaaaaaaaaaaa")}, function(err,
                    cursor.toArray(function(err, items) {
                        test.assertEquals(1, items.length);

                        // Let's close the db
                        p_client.close();
                    });
                });
            });
        });
    });
});
```

You are not likely to encounter this problem often in your code, but when you do—and you will from time to time—here are two solutions to this problem. [3]

1. Name your functions and declare them and pass just the name of the function as the callback, instead of defining an anonymous function in the parameter of the main function.
2. Modularity: Separate your code into modules, so you can export a section of code that does a particular job. Then you can import that module into your larger application.

Make Your Own Callback Functions

Now that you completely (I think you do; if not it is a quick reread :)) understand everything about JavaScript callback functions and you have seen that using callback functions are rather simple yet powerful, you should look at your own code for opportunities to use callback functions, for they will allow you to:

Do not repeat code (DRY—Do Not Repeat Yourself)

Implement better abstraction where you can have more generic functions that are versatile (can handle all sorts of functionalities)

Have better maintainability

Have more readable code

Have more specialized functions.

It is rather easy to make your own callback functions. In the following example, I could have created one function to do all the work: retrieve the user data, create a generic poem with the data, and greet the user. This would have been a messy function with much if/else statements and, even still, it would have been very limited and incapable of carrying out other functionalities the application might need with the user data.

Instead, I left the implementation for added functionality up to the callback functions, so that the main function that retrieves the user data can perform virtually any task with the user data by simply passing the user's full name and gender as parameters to the callback function and then executing the callback function.

In short, the getUserInput function is versatile: it can execute all sorts of callback functions with myriad of functionalities.

```
// First, setup the generic poem creator function; it will be the callback function
function genericPoemMaker(name, gender) {
    console.log(name + " is finer than fine wine.");
    console.log("Altruistic and noble for the modern time.");
    console.log("Always admirably adorned with the latest style.");
    console.log("A " + gender + " of unfortunate tragedies who still manages a perpe
}

//The callback, which is the last item in the parameter, will be our genericPoemMake
function getUserInput(firstName, lastName, gender, callback) {
    var fullName = firstName + " " + lastName;

    // Make sure the callback is a function
    if (typeof callback === "function") {
    // Execute the callback function and pass the parameters to it
    callback(fullName, gender);
    }
}
```

Call the getUserInput function and pass the genericPoemMaker function as a callback:

```
getUserInput("Michael", "Fassbender", "Man", genericPoemMaker);
// Output
/* Michael Fassbender is finer than fine wine.
Altruistic and noble for the modern time.
Always admirably adorned with the latest style.
A Man of unfortunate tragedies who still manages a perpetual smile.
*/
```

Because the getUserInput function is only handling the retrieving of data, we can pass any callback to it. For example, we can pass a greetUser function like this:

```
function greetUser(customerName, sex)  {
   var salutation  = sex && sex === "Man" ? "Mr." : "Ms.";
  console.log("Hello, " + salutation + " " + customerName);
}

// Pass the greetUser function as a callback to getUserInput
getUserInput("Bill", "Gates", "Man", greetUser);

// And this is the output
Hello, Mr. Bill Gates
```

We called the same getUserInput function as we did before, but this time it performed a completely different task.

As you see, callback functions afford much versatility. And even though the preceding example is relatively simple, imagine how much work you can save yourself and how well abstracted your code will be if you start using callback functions. Go for it. Do it in the monings; do it in the evenings; do it when you are down; do it when you are k

Note the following ways we frequently use callback functions in JavaScript, especially in modern web application development, in libraries, and in frameworks:

- For asynchronous execution (such as reading files, and making HTTP requests)
- In Event Listeners/Handlers
- In setTimeout and setInterval methods
- For Generalization: code conciseness

Final Words

JavaScript callback functions are wonderful and powerful to use and they provide great benefits to your web applications and code. You should use them when the need arises; look for ways to refactor your code for Abstraction, Maintainability, and Readability with callback functions.

See you next time, and remember to keep coming back because JavaScriptIsSexy.com has much to teach you and you have much to learn.

Notes

1. http://c2.com/cgi/wiki?FirstClass
2. https://github.com/mongodb/node-mongodb-native
3. http://callbackhell.com/
4. JavaScript Patterns by Stoyan Stefanov (Sep 28, 2010)

Posted in: 16 Important JavaScript Concepts, Advanced javaScript, JavaScript / Tagged: Callback Functions, Higher-order Functions

Richard

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

180 Comments

Mikey

March 14, 2013 at 12:45 pm / Reply

Thank you or this I never took the time to learn callback functions.

Richard Bovell (Author)

March 15, 2013 at 10:34 pm / Reply

You are welcome, Mike.

Kyle

March 20, 2013 at 1:19 pm / Reply

Thanks for this! Callbacks have always been a little confusing for me to understand but you have laid it out so simply. I especially love the last two examples that demonstrate how callbacks can be simple and still very powerful to abstract out the code. I've been reading your other posts and you are doing a great job of communicating topics like OOP, closures, protot

inheritance, scope, etc. Not coming from a CS background and wanting to learn JS, these topics have been intimidating to understand and when I search for help on them the explanations are always too technical. I like how you use simple and concise examples like “Hello ” and “aFruit inherits from otherFruit”. So thank you very much and keep up the great work!

Richard Bovell (Author)

March 25, 2013 at 2:14 am / Reply

Thanks for the kind words, Kyle.

I am very happy to hear that my articles are helpful. And most importantly, I am glad the articles have helped you better understand some of JavaScript core concepts.

Keep learning and coding 🙂

Paula

April 4, 2013 at 9:52 pm / Reply

Love this blog post! It was very helpful to me.

Bobby

April 15, 2013 at 3:55 am / Reply

Great post! Thanks for writing it! I have a question regarding an example I read from the JavaScript: Definitive Guide.

One of their examples goes something like this:

```
function not(f) {
return function() {
var result = f.apply(this, arguments);
return !result;
};
}
```

My question is what does the “this” (first argument) refer to?
Thanks so much in advance! Great article and once again thank you for taking the time for teaching noobs like me.
-Bobby

Richard Bovell (Author)

April 16, 2013 at 2:58 pm / Reply

Bobby,
The “this” you are inquiring about is referring to the *window* object, which is the global window object that all the code is defined on.

In the *apply* function, the first argument sets the use of the “this” keyword. So this line:
f.apply(this, arguments);
is saying: call the f function and use the *window* object as the this keyword inside the f function. Also, pass to the f function the *arguments* passed into the *not* function.

Here is a slight modification of the code that shows reveals the detail:

```
var name = "Richard";
function not(f) {
    return function() {
        console.log("This: " + this); //This: [object Windo
        var result = f.apply(this, arguments);
        return result;
    };
}

function test () {
    var name = "Rob";
    return this.name; // Richard (not Rob, because "this" re
}

var news = not (test);
news ();
```

Mike

June 10, 2013 at 11:57 am / Reply

I am an experienced javascript developer and find your articles excellent as they cover much of the fine nuances and details of Javascript that are fundamental to its understanding and yet overlooked / ignored by so many (myself included)

Congratulations and keep up the good work (as i'll be coming back regularly 🙂

Richard Bovell (Author)

June 21, 2013 at 2:28 am / Reply

Thanks very much, Mike.

TPS

July 11, 2013 at 11:06 am / Reply

Thanks for writing this article.
Thanks 🙂

Richard Bovell (Author)

July 11, 2013 at 7:45 pm / Reply

Thank you, TPS.

Anton

August 1, 2013 at 6:07 am / Reply

Very nice website, and your explanations are pretty clear. However I'm sorry to inform you, what you are talking about are not true Callbacks. These are essentially call forwards, ie. telling one routine the name (or in C the pointer) to another routine to call. A true Callback is an asynchronous device used to tell the compiler where to route the response back to the requesting program.

In fact a true Callback is never explicitly called by the programmer, the compiler will actually engineer the connection.

Depending on your perspective, you could argue that the event function is an actual Callback routine since in reality, those routines are registered with the OS as event handler functions that need to be responded to by the OS Event Handler and thus the function is called in the event of that Event occurring.

In the event that you don't understand my description of Event Handling, try Googling "Event Handler". In any event, have a nice day!

Balázs Mária

August 1, 2013 at 10:15 am / Reply

Something seems to be wrong about the callbacks section.

In the example you modify the function getUserInput(firstName, lastName, callback).

That's all nice, but most of the time I write the callback function, and not the function which I pass the callback to.

So if I have a callback function, how do I preserve its this object? I'm not sure I can.

Richard Bovell (Author)

August 9, 2013 at 8:21 pm / Reply

I discussed this in the article. Read the section, "Problem When Using Methods With The 'this' Object as Callbacks."

Nick

August 8, 2013 at 9:22 pm / Reply

Richard,

Good series so far. Helps me review some of these core JS concepts in a straightforward, plain English manner. Makes me more comfortable with

concepts, so I appreciate your writing style. That being said, question about this bit of text:

“We would have used the Call function if we only needed to pass a single parameter to the callback function, but since we needed to pass firstName and lastName, we used the Apply function. This is the only difference between Call and Apply.”

This seems to imply that using Call was not an option when in fact you could have used “callback.call (callbackObj, firstName, lastName);” in place of “callback.apply (callbackObj, [firstName, lastName]);”. If you meant that your choice to use Apply was just a preference rather than the only option, do you mind clarifying that in the article for your readers? In any case, thanks for the short and intuitive articles.

Richard Bovell (Author)

August 9, 2013 at 8:17 pm / Reply

Nick,
You are absolutely correct: I have modified the sentence to make it more correct and precise.

We could have used either the Call or Apply methods.

Thanks.

yougen

August 19, 2013 at 8:17 am / Reply

Hi, you are doing an amazing job! I learn a lot from your posts. Is it possible to writing an Execution Context and Scope topic post using graphics to illustrate? Like this post style http://howtonode.org/object-graphs. As for as I know, I t will be easier to understand something by graphs. Thanks a lot.

Richard Bovell (Author)

August 19, 2013 at 2:56 pm / Reply

Why not. I will give it go. I am not sure when I will write about Execution Context, but I have written about JavaScript Scope here: http://javascriptissexy.com/javascript-variable-scope-and-hoisting-explained/

ben

August 23, 2013 at 8:27 am / Reply

thanks a lot, it it is very clear and it helped me a lot!

Richard Bovell (Author)

August 23, 2013 at 11:35 am / Reply

Great.

preeti jain

September 3, 2013 at 5:40 am / Reply

i an working on indexedDB and jaydata,
i want to return number of records in table “LogFile”

```
function checkLogFile()
{
var length;
offlinedb.LogFile.toArray(function (documents) { length=documents.length;
console.log(length); // here show 3 which is right
});
return length; // cant access length , shows 0
}
```

how it cab be done ?

Richard Bovell (Author)

September 4, 2013 at 10:02 pm / Reply

Which JavaScript library are you using for the “toArray” method? It is not a native JavaScript method. I am curious. And post the full example code.

BTW, using “length” as a variable name is not a good idea.

Pradeep Reddy

September 4, 2013 at 2:49 pm / Reply

This was really helpful,clearly explained and Simply Awesome.

Thank you Rich

Richard Bovell (Author)

September 4, 2013 at 9:39 pm / Reply

Lovely! I am very happy to hear that the article was clear to understand. You are welcome, and good luck.

vedran

September 5, 2013 at 6:42 am / Reply

Richard,

Thank you for an amazing place that holds bunch of articles which are very, very easy to follow and digest.

Richard Bovell (Author)

September 8, 2013 at 12:59 pm / Reply

You are welcome, Vedran.

Serge

October 14, 2013 at 9:24 am / Reply

Hi! Thanks for nice post. This is the first post from you blog I've read and I didn't yet read other post (but have subscribed), maybe you plan to write posts about "Default settings" and "Pub-sub" patterns (if not yet), I'm personally very interesting in)

Richard Of Stanley (Author)

October 16, 2013 at 1:59 am / Reply

Yes, I do plan to write about PubSub. I actually completed the PubSub post many weeks ago, but I have to spend a few hours formatting it and cleaning it up before I publish it.

What do you mean by "Default Settings"?

Gyan

October 23, 2013 at 12:13 pm / Reply

Very nice post… I have read many posts of your's here and every time I gained a lot.

You are helping me in reaching to the next level.

Thanks a lot.

Richard Of Stanley (Author)

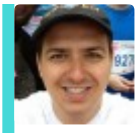
October 23, 2013 at 3:26 pm / Reply

Great to hear, Gyan. Keep up the good work learning and be come a great JS developer one day.

Karl Pokus

November 12, 2013 at 11:22 am / Reply

Great write-up! +1 for your patience.



Fabio Serragnoli

November 22, 2013 at 7:38 am / Reply

Well done Richard.
Excellent post.

jimmytung

November 28, 2013 at 9:48 pm / Reply

Great post.
Thanks for sharing your mind.

Alex Jacobs

December 3, 2013 at 8:11 pm / Reply

Great post.
Question about formatting of the ternary operator in the code block of the last example:

```
function greetUser(customerName, sex) {
var salutation = sex && sex === “Man” ? “Mr.” : “Ms.”;
console.log(“Hello, ” + salutation + ” ” + customerName);
}
```

It’s working in my browser to simply write it as:
`var salutation = sex === “Man” ? “Mr.” : “Ms.”;`

Is that proper syntax?

Alex Jacobs

December 3, 2013 at 8:32 pm / Reply

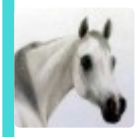
Is this alternate correct?

```
var salutation;
sex === “Man” ? ( salutation = “Mr.”) : ( salutation = “Ms.”);
```

Kiran

December 3, 2013 at 10:41 pm / Reply

Thank you for the explanation. Much informative.



Dan

December 26, 2013 at 6:03 am / Reply

Thanks for this, you saved me hours of banging my head against a wall due to my ignorance. I'm a do-it-youself developer. I wish Google had put your site higher sooner!

Richard Of Stanley (Author)

January 7, 2014 at 12:40 am / Reply

Thank you, Dan. And I know What you mean about the Google search results.

Some of the articles on this blog are ranked very high on Google search results, while some others are not: That is Google's algorithm at work. I am thankful that they send quite a bit of hits to the blog, though.

Harsha

December 26, 2013 at 7:31 am / Reply

Thanks for this great post. Always dreaded callbacks when I saw them, now I am looking forward to actually creating and using them extensively.

Thanks Again

Richard Of Stanley (Author)

January 7, 2014 at 12:38 am / Reply

I am happy to help, Harsha. Enjoy coding.

Mahmoud annaggar

December 30, 2013 at 3:27 am / Reply

Thanks a lot for this wonderful article, It helped me very much understanding callbak.

Richard Of Stanley (Author)

January 7, 2014 at 12:33 am / Reply

I am happy to hear that, Mahmoud.

srini

January 3, 2014 at 5:17 pm / Reply

thank you for the article, it helped to understand the callbacks and its pros &cons from your examples.

Filip

January 15, 2014 at 4:48 am / Reply

As far as I'm concerned these articles are the best way for learning javascript. This is my favourite javascript site 🙂
Keep it up with the good work!

Kyle

February 2, 2014 at 12:51 pm / Reply

You sir, know how to write a tut

Nikhil

February 3, 2014 at 3:42 am / Reply

Great article. Thanks you.

Sean

March 6, 2014 at 6:46 am / Reply

Thank you very much for this! I try to change some value in the example and it works~

Vincent

March 6, 2014 at 12:16 pm / Reply

Fantastic writeup! This concise article has really helped me understand how important callback functions are. I shall definitely be coming back here on a regular basis.

Esteban

March 6, 2014 at 2:14 pm / Reply

Hi all, I have a doubt about jquery function, I have my click function
$('#BtnBuscarb').click(function (param) {
}
And I want to pass a parameter here $("#BtnBuscarb").trigger("click");
how i can pass my param in this call?

alexdown

August 12, 2014 at 11:05 pm / Reply

use an array after the event type.

.trigger(eventType [, extraParameters])

from http://api.jquery.com/trigger/

Jason

March 10, 2014 at 3:06 pm / Reply

thanks so much for writing this! it's an excellent article and has clarified callbacks significantly for me. i do have one very 'stupid' question. in the example that you gave (copied below), how does the the anonymous function

know what the variables ‘eachName’ and ‘index’ refer to since they weren’t declared anywhere?

var friends = [“Mike”, “Stacy”, “Andy”, “Rick”];

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

San

May 6, 2014 at 7:54 am / Reply

Hello Jason,
Each element and its index are passed to the callback function.
These element and index are copied to the callback arguments(eachName and index).
//San.

JP Balakrishna

March 17, 2014 at 3:08 am / Reply

It is a really great, short and worthy article……
Im searching for months….atlast you I found you
RICHARD it is really helpful for every javascript learner….
Im expecting many more javascript articles from you….

ALL 16 concepts are simply superb…thank you

mark twain

March 27, 2014 at 6:59 pm / Reply

In below code, how are “eachName” and “index” populated?

var friends = [“Mike”, “Stacy”, “Andy”, “Rick”];

friends.forEach(function (eachName, index){
console.log(index + 1 + “. ” + eachName); // 1. Mike, 2. Stacy, 3. Andy, 4

Rick
});

alexdown

August 12, 2014 at 11:02 pm / Reply

“callback is invoked with three arguments:
– the element value
– the element index
– the array being traversed
”

https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Globa

siv niz

May 13, 2014 at 3:25 pm / Reply

You definitely got one more follower…page bookmarked.

Phil

May 23, 2014 at 1:36 pm / Reply

function rocks(name) {
console.log(name + ” rocks and gives the best JS explainations!”);
}

function giveOpinion(firstName, lastName, callback) {
var fullName = firstName + ” ” + lastName;
if (typeof callback === “function”) {
callback(fullName);
}
}

giveOpinion(“Richard”, “Bovell”, rocks);

budyk

July 3, 2014 at 8:22 am / Reply

Thank you sir for the tutorial …

Kapil

July 6, 2014 at 7:49 am / Reply

One of the excellent posts about callbacks

Srle

July 19, 2014 at 7:33 pm / Reply

“As we know, closures have access to the containing function’s scope, so the callback function can access the containing functions variables”

This is not correct, callback function CAN’T access the containing function variables.

http://stackoverflow.com/questions/24119106/callback-function-closure-and-execution-context

Kevin

July 23, 2014 at 4:42 pm / Reply

Hi Richard,

Am I missing something or in the section “Multiple Callback Functions Allowed” do you use the same function name to define two different “successCallback” callback functions?

Thanks,
Kevin

alexdown

August 12, 2014 at 10:58 pm / Reply

Yep, looks he copy/pasted 4 times because he wanted to define a function for beforeSend too but then forgot to rename (and us

Ori

July 26, 2014 at 2:27 am / Reply

Great tutorial, thank you.
On question: what if I want to make a generic function that gets a callback function with unknown number of arguments that need to be passed to the callback function?

Sharath

September 7, 2014 at 7:30 am / Reply

you can store all your unnumbered arguments in an array or object and pass that, in simple words use apply.

Anish Gurung

August 1, 2014 at 12:40 am / Reply

Was finding difficulty in understanding callbacks used in javascript and node.js . Many thanks for the awesome tutorial.

Gerald LeRoy

August 18, 2014 at 11:15 am / Reply

Hello,

Thanks for your excellent article! I found it to be very helpful.

A couple of comments:
1. Srle is correct. In this case, the callback function can't access the containing function's variables. The callback function's parent is the global object.

2. "Also, pass to the f function the arguments passed into the not function."

In this case, the arguments parameter doesn't refer to arguments passed to the not function. The arguments parameter refers to arguments passed f

returned function, i.e. news()

Thanks again for taking the time to post this information.

Sharath

September 7, 2014 at 7:27 am / Reply

Thanks for your articles, i have one pretty simple and silly question though.

Most of the time when i call a function from a function, i just don't pass the calling function as parameter to the main or containing function. Does that mean i am using callback ?

budi

October 13, 2014 at 8:03 am / Reply

this is why i get new knowledge ..awesome…

SmilingSunrise

November 3, 2014 at 10:33 am / Reply

It's very perfect article!

Daniel

November 24, 2014 at 7:11 pm / Reply

Your explanations are so concise and clear. Keep up the good work! I like the your list for reasons of using callbacks. I had known of callbacks and how to use them but not many articles explained any of the reasons for using them.

w

December 18, 2014 at 5:03 am / Reply

wwwwwwwwwwwwwwww

Manickkam

December 26, 2014 at 7:28 am / Reply

Excellent write-up on Callback functions. Thanks a lot for enlightening the people around with this important stuff in Javascrupt.

Himanshu Pandey

December 27, 2014 at 11:36 pm / Reply

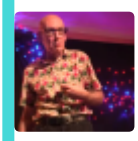
Hi
I am also web developer and now coding in asp.net mvc4.Today I have complete your articles on Callback and closure both article is very descriptive and today I can say that yes I have the knowledge about closure and callback.Great works by you.Thanks a lot to write such article.

Gregory Ledger

January 18, 2015 at 4:41 pm / Reply

Thanks for the post, but I'm having trouble with the concept.
in the code:
//Note that the item in the click method's parameter is a function, not a variable.
//The item is a callback function
$("#btn_1″).click(function() {
alert("Btn 1 Clicked");
});

is the callback function "alert("Btm 1 Clicked")' and outer function the "function() {…}" or is the callback function the "function(){…} and the outer function something else [perhaps a method of the window object]?



Danny Crossley

February 18, 2015 at 7:27 am / Reply

I found this explanation very clear and helpful. It helped me to get my head around it. As a result, I have produced some code that helps me understand it and should like to share it.

http://pastebin.com/embed_js.php?i=ShVatkEJ

sumit

March 24, 2015 at 4:25 am / Reply

very helpful article…i always used callback function but actual concept and it's working understood from your article only:)thanx again:)

Filip

April 20, 2015 at 6:58 am / Reply

Awesome tutorial. You guys have some of the best JavaScript tutorials out there.
Thanx!

Luciana

April 24, 2015 at 7:55 pm / Reply

Your posts are just great! Thanks for sharing them!

Paul

May 19, 2015 at 1:47 pm / Reply

Just a quick note to those with "Callback Hell", otherwise known as the "Pyramid of Doom". You may want to check out another approach which has become popular. Promises: with the Q library, jQuery has also implemented this, and AngularJS with $q. Instead of callback after callback after callback, the function returns a promise and what is returned from the asynchronous function is passed the promise then() or catch() function if a problem arises. The syntax is kinda like this:

```
call1(function(first_callback) {
// do something
return Q.resolve(some_object);
}).then(function(some_object) {
// do something with some_object
```

}).catch(function(error) {
// there was an error do something
});

You can essentially chain callbacks in a fashion that marches down rather than to the right. I find it immensely helpful even without large chains.

Aziz

May 27, 2015 at 7:18 pm / Reply

This is a great blog post, I found it to be very useful

I would really like to understand how the parameters in the callback are working in the example you mentioned “eachName, index” these haven’t been specified in a function call but yet the get the values of the array?

var friends = [“Mike”, “Stacy”, “Andy”, “Rick”];

friends.forEach(function (eachName, index){
console.log(index + 1 + “. ” + eachName);
});

I would normally expect parameters of a function to be specified in a function call

function add (x, y) {
console.log (x + y);
}

add(2 , 3);

Thanks,
Aziz

Ram

June 17, 2015 at 2:46 am / Reply

Thank you

Mehul

June 17, 2015 at 3:10 am / Reply

As you mentioned in above ..

Callback Functions Are Closures

As we know, closures have access to the containing function's scope, so the callback function can access the containing functions' variables, and even the variables from the global scope.

But it's showing error in following code

```
function fun(firstParam,callback){
var funVariable = 10;
callback();
}

function myfun() {
console.log("firstParam value : "+ firstParam);
console.log("funVariable value : "+funVariable);
}

fun("11",myfun);
// error — firstParam is not defined
// error — funVariable is not defined
```

if function scope variable can access in callback function then it should work.

Help me to get out from this problem

Raghavan SK

June 19, 2015 at 2:59 am / Reply

This post was really helpful to me…Thanks a lot !!!

Jeffrey Wan

June 25, 2015 at 7:14 pm / Reply

Why is var allUserData = []; important in these examples?

Aaron

June 30, 2015 at 3:05 am / Reply

Well written, thanks. Turns out I have been using call back for ages without really understanding what they did.

Fauzia

July 1, 2015 at 3:52 am / Reply

Such a wonderful informative and helping article made me able to use Callback functions and do modifications.Before reading it I was just looking at the word “callback” in Javascript and could’nt work on it.

Pratibha Panchmukh

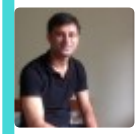
July 1, 2015 at 7:50 am / Reply

Well written post, helpful to us. Thank you !!!

Harshal

July 7, 2015 at 2:13 am / Reply

Thorough explanation..Very helpful article 🙂



Ash

July 7, 2015 at 5:13 am / Reply

Thanks for this brilliant post.
In the example you gave, I am curious to know how the anonymous function gets called by the arguments object of the containing function.

```
$(“#btn_1″).click(function() {
alert(“Btn 1 Clicked”);
});
```

RustyB

July 15, 2015 at 9:44 pm / Reply

What those ^^^ guys said. I have been struggling with callbacks for ages (too long). Now I get it.

Keshar Limbu

July 16, 2015 at 11:40 am / Reply

Thank you very much for so detailed information on callback functions.

Ivan

July 17, 2015 at 2:41 pm / Reply

Great Job in explaining callback functions essentials in simple and transparent way!

refschool

July 22, 2015 at 11:04 am / Reply

Thank you, at last I found an article that covered completely the UNDER THE HOOD aspect, now i can say i understand the theory about callback functions and most importantly how to implement them.

GEA

July 22, 2015 at 6:54 pm / Reply

Your definition of a callback is such that a callback is synonymous with a higher-order function. Are you sure?

From wikipedia:
In mathematics and computer science, a higher-order function (also functional form, functional or functor) is a function that does at least one of the following:

1. takes one or more functions as an input
2. outputs a function

This has nothing to do with said function being taken as parameter to another function. This confuses me a bit and I'd appreciate your thoughts. Thanks!

codeoverdose

August 4, 2015 at 7:21 pm / Reply

That threw me off too.

Haverbeke defines a higher order function as:

"Functions that operate on other functions, either by taking them as arguments or by returning them, are called higher-order functions"

Nagaraja T

August 5, 2015 at 11:15 pm / Reply

That's wonderful explanation. I was too much struggling to understand the code with callbacks and closure. Now I got some idea of how the flow works.

Thanks !!

fistvan

August 8, 2015 at 9:16 am / Reply

Hi,

you say that: "it will set fullName on the window object, since getUserInput is a global function". Ok,
but in the following example I have two local functions (as far as I know they are local) getUserInput2 and getUserInput1, and the result is the same:

```
function Test() {
this.clientData = {
fullName: "Not Set",
setUserName: function (firstName, lastName) {
```

```
this.fullName = firstName + " " + lastName;
},

getUserInput2: function (firstName, lastName, callback) {
callback(firstName, lastName);
}

};

this.getUserInput1 = function (firstName, lastName, callback) {
callback(firstName, lastName);
};

}

var test = new Test();
var userInput = new test.getUserInput1("Barack1", "Obama1",
test.clientData.setUserName);

console.log('test.clientData.fullName : ' + test.clientData.fullName); // Not Set
console.log('window.fullName : ' + window.fullName); // Barack1 Obama1

test.clientData.getUserInput2("Barack2", "Obama2",
test.clientData.setUserName);

console.log('test.clientData.fullName : ' + test.clientData.fullName); // Not Set
console.log('window.fullName : ' + window.fullName); // Barack2 Obama2
```

Could you please explain why these functions produce the same result since they are not global functions?

Nice post! Thank you!

Ankush Somani

August 13, 2015 at 10:51 am / Reply

This was brillient post. I was afraid of *callback function* before reading this blog. thanks a lot.

Hector

August 19, 2015 at 6:29 am / Reply

Hello!, thank you very much for this great explanation! I read all the nested articles as well. great way of explaining everything. Cheers!

Will

August 19, 2015 at 11:44 am / Reply

This was fantastic! This is a tough concept to grasp for a programming noob like myself.

Michael

August 23, 2015 at 10:16 am / Reply

Hello Richard. Can you explain why is the var allUserData = []; and the allUserData.push(options); are important? Thank you very much.

Hussain AlQatari

August 31, 2015 at 10:20 pm / Reply

Hi Richard Of Stanley,

“(Learn JavaScript Higher-order Functions, aka Callback Functions)”
“A callback function, also known as a higher-order function, is a function that is passed to another function (let’s call this other function “otherFunction”) ”

Actually, the enclosing function, what you are calling otherFuntion, is the higher order function. The callback function is never called higher order. Check it yourself.

Jack

September 9, 2015 at 10:23 pm / Reply

Thanks for writing this, It’s really going to help with all those nested ajax callbacks!

I feel like I can actually write a few normal functions now and get some re-usability in my javascript!

GreenRaccoon23

September 10, 2015 at 12:46 am / Reply

This is solid gold, Richard. I'm new to JavaScript and you explained this so well that even noobs like me could grasp it. I'm so glad you included the 'Use the Call or Apply Function To Preserve this' section too. That would have for sure messed me up somewhere down the road and it would've taken me forever to figure out why the callbacks weren't working.

CZ

September 10, 2015 at 9:42 pm / Reply

Great article, solid explanation. Thanks!

Nidhi

September 24, 2015 at 2:34 am / Reply

Great explanation ! Understand callback nicely!

NL

September 24, 2015 at 1:03 pm / Reply

One of the best articles about callbacks, congratulations. Thanks a lot.

alakhya

October 3, 2015 at 2:45 pm / Reply

I just gone through the example :- you have not defined any function with name "callback(options);" any where in your example.

is "callback" is a keyword ??

alakhya

October 3, 2015 at 2:45 pm / Reply

I just gone through the example :- you have not defined any function with name “callback(options);” any where in your example.

is “callback” is a keyword ??

alakhya

October 3, 2015 at 2:46 pm / Reply

I just gone through the example :- you have not defined any function with name “callback(options);” any where in your example.

is “callback” is a keyword ?? Could you please answer my question

Mukesh

October 6, 2015 at 6:15 am / Reply

Great explanation,
Very very thanks for explanation in a very simple way.

Sudheesh MS

October 10, 2015 at 10:20 am / Reply

Great explanation. I don’t think a concept can be explained more simpler. Thanks

Toby

November 3, 2015 at 4:00 pm / Reply

Thanks for the detailed explanation and helpful examples. Very helpful.

Armin

November 6, 2015 at 5:51 pm / Reply

a quite valuable article!

Chan

November 7, 2015 at 1:07 pm / Reply

you make it more complex it for me to understand the complex. don't write those if you didn't understand the concept. you have google some articles and try to make it as your own one.

why you didn't explain this.
riends.forEach(function (eachName, index){
console.log(index + 1 + ". " + eachName); // 1. Mike, 2. Stacy, 3. Andy, 4. Rick
});

this is much better then this shit.

http://recurial.com/programming/understanding-callback-functions-in-javascript/

Chan

November 7, 2015 at 1:10 pm / Reply

you make it more hard for me to understand the logic behind this. don't write those if you didn't understand the concept. you have google some articles and try to make it as your own one.

why you didn't explain this.

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

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

below article is much better then this shit.
http://recurial.com/programming/understanding-callback-functions-in-javascript/

Armen

November 8, 2015 at 5:03 pm / Reply

Hi There,

I just wanted to tell you that “You just made my day today!” Believe me you made my day, today!
I am Java developer and all of a sudden are project requires Javascript knowledge. My JS skills are near zero.
Found JS very hard to learn and understand. Now, i am on the 5th tutorial and enjoying what i am reading/learning from you Awesome explanations.
You are direct to the point and easy to understand. Please
keep the great going..

Thanks again,

–Armen

Nagesh

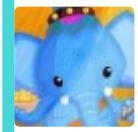
November 13, 2015 at 3:46 am / Reply

Great explanation ! Understand callback easily

Saran

November 19, 2015 at 12:30 pm / Reply

In the example callback function is not defined but its used. i am not able to find the callback() function code. am i missing anything?



San

November 27, 2015 at 3:23 am / Reply

Hello Richard,
Its really wonderful article. I have a question.

How can I assign a value from a call back function to a variable in the global scope?

The following code is DrupalGap, but no one answering there.

```
function my_first_module_menu_page() {
var content = {};
var user_count;

my_first_module_get_user_count({
success: function(result) {
user_count = result[0];

console.log('There are ' + user_count + ' registered user(s)!'); // Works here
}
});

// variable user_count is not accessable here(undefined)
content['my_intro_text'] = {
markup: 'User Count :' + user_count + ''
};

return content;
}

function my_first_module_get_user_count(options) {
try {
options.method = 'GET';
options.path = 'my_first_module_resources.json';
options.service = 'my_first_module';
options.resource = 'get_user_count';
Drupal.services.call(options);
}
catch (error) { console.log('my_first_module_get_user_count – ' + error); }
}
```

When I call my_first_module_menu_page(), I want to set the variable 'user_count' with the result from the my_first_module_get_user_count().

SOF link: http://stackoverflow.com/questions/33917091/how-to-access-the-value-outside-the-function-in-drupalgap

Russell

December 17, 2015 at 6:01 pm / Reply

Great Post!

lokesh kalyanam

December 19, 2015 at 9:19 am / Reply

Thank you for great explanation Richard Bovell.

Lucky

December 19, 2015 at 12:32 pm / Reply

Hello Richard!
Thank you so much for what you have been teaching us. I have read a couple of your articles and I find them extremely helpful. Your explanation is very clear and easy to understand. WOW I'm glad I have found this site. This encourages me to hold on to Javascript. Someone told me to give up. He told me "your code is so bad, so unclear and duplicated, you shouldn't be doing this". I was sad and depressed and then I came to realize I have to prove him that he's so dead wrong about me. I can write clean, maintainable and sexy code. I would be a excellent front-end dev.
Thank you again I wish you best of luck…
Regards,

Glen

January 1, 2016 at 8:04 am / Reply

Hi, in your tutorial "JavaScript Objects in Detail" you have shown an example where mango as native of Central America. It is not. Its a native of (also national fruit) of INDIA. Also the name "mango" comes from an Indian language. This needs to be corrected.

Aaron

January 30, 2016 at 12:40 pm / Reply

Thanks man!

yogesh

February 3, 2016 at 7:43 am / Reply

Hi Bro, It was awesome tutorial for beginners, Thanks For this. 🙂

Dane

February 21, 2016 at 7:27 pm / Reply

I noticed you used the phrase: “imagine how much work you can save yourself and how well abstracted your code will be if you start using callback functions”

Just a heads up, abstracted means: “showing a lack of concentration on what is happening around one, oblivious, inattentive.” I think a better word may suffice unless the phrase is referencing something I am unaware of.

Toodles.

d0t

February 25, 2016 at 11:48 pm / Reply

Neat and clear.

Mohammad Irshad

February 26, 2016 at 12:18 am / Reply

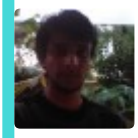
Thank you Richard!
I found it useful and it is very very good explanation of callback.

Andrés

March 18, 2016 at 6:28 pm / Reply

My mind blow after this well redacted, structured and showed javascript language future.

Thanks so much 🙂



vikrant singh

March 21, 2016 at 8:19 am / Reply

why is the apply not working in the below code:

```
function Test() {
this.clientData = {
fullName: “Not Set”,
setUserName: function (firstName, lastName) {
this.fullName = firstName + ” ” + lastName;
},
getUserInput2: function (firstName, lastName, callback) {
callback(firstName, lastName);
}
};
this.getUserInput1 = function (firstName, lastName, callback,obj) {
callback.apply(obj,[firstName,lastName]);
//callback(firstName, lastName);
};
}
var test = new Test();
var userInput = new test.getUserInput1(“Barack1″, “Obama1″,
test.clientData.setUserName,Test.clientData);
console.log(‘test.clientData.fullName : ‘ + test.clientData.fullName); // Not Set
console.log(‘window.fullName : ‘ + window.fullName); // Barack1 Obama1
test.clientData.getUserInput2(“Barack2″, “Obama2″,
test.clientData.setUserName);
console.log(‘test.clientData.fullName : ‘ + test.clientData.fullName); // Not Set
console.log(‘window.fullName : ‘ + window.fullName); // Barack2 Obama2
```

George

March 28, 2016 at 9:13 pm / Reply

Great posts and a great site! However, I'm confused by this example. How is it that apply is a method of the parameter f? What am I missing?

ani

April 14, 2016 at 1:17 pm / Reply

Excellent post! Thanks!

FlyingGambit

April 21, 2016 at 6:51 am / Reply

Is callback function also a closure ?

Raj

April 26, 2016 at 8:43 am / Reply

Instead of passing the function definition as an argument and using it as callback, why can't we directly call the function inside at last line like below.

```
var clientData = {
id: 094545,
fullName: "Not Set",
setUserName: function (firstName, lastName)
{
this.fullName = firstName + " " + lastName;
}
}

function getUserInput(firstName, lastName, callback)
{
callback (firstName, lastName);
}

getUserInput ("Barack", "Obama", clientData.setUserName);

console.log (clientData.fullName);
```

console.log (window.fullName);

_____

In the above example, instead of passing clientData.setUserName as parameter and getting it as callback in getUserInput, why can't we directly use that function inside the getUserInput function like below

function getUserInput(firstName, lastName,)
{
clientData.setUserName(firstName,lastName)
}

Thanks,
Raj C

Mauricio

April 27, 2016 at 11:03 am / Reply

Great post. I have been using callbacks sporadically following examples from other sources, but I never really understood the theory and the power of callbacks. Thanks for the post

Seth

May 5, 2016 at 6:39 pm / Reply

Why not just use .bind instead of apply or call?

Dirk

May 7, 2016 at 7:17 pm / Reply

That's 'distracted' …

Munish

October 5, 2016 at 6:32 am / Reply

Great Post!!

Does Call back is somewhat related to late binding or virtual functions.

Jonathan

November 1, 2016 at 6:24 pm / Reply

Great post!!! Thank you very much.

Thomas Thai

November 11, 2016 at 6:56 pm / Reply

Thanks you for a well written tutorial! It was easy to follow with examples to demonstrate the concept.

Trackbacks for this post

1. JavaScript Medley | M's Web Dev
2. Apply, Call, and Bind Methods are Essential for JavaScript Professionals | JavaScript is Sexy
3. 16 JavaScript Concepts JavaScript Professionals Must Know Well | JavaScript is Sexy
4. Things you may find confusing when you shift from Java to JavaScript | Adi's Blog
5. ساخت یک وب سرور ساده به وسیله Node.js | A Geek Notes
6. Helping You Study for Exam 70-480 HTML5, CSS3 and JavaScript | eric.w.bryant
7. sexy.com | JavaScript is Sexy
8. L'importance, pour un développeur web, de connaître le JavaScript | French Coding
9. D3js take data from an array instead of a file | Ngoding
10. Javascript | Pearltrees
11. How can I get the return value from an ajax event? [duplicate] | Zager Answers
12. JavaScript: Creating a function with a callback - csskarma
13. Day 42 – Current Weather | Stacey Learns to Code
14. Fun with Callbacks | Layers of Curiosity
15. The Odin Project Program Outline | Tilly Codes
16. Understand JavaScript Callback Functions and Use Them | Dinesh Ram Kali.
17. JavaScript Callback | J.-H. Kwon
18. Nodejs related resources | notes
19. In jQuery UI Accordion, where does the `response` function is implemented? - BlogoSfera
20. Understand JavaScript Callback Functions and Use Them | Sai

21. Preparing Before Hack Reactor Begins | bash $ cat bitchblog
22. JS- Callback | anxtech
23. 理解与使用Javascript中的回调函数 | Cmgine
24. Do you see the client server round trip always? |
25. Hack Reactor’s Technical Interview | FunStackDeveloper.com
26. AJAX For The Masses | Strange Dog Blog
27. How do callbacks work in JavaScript? - HTML CODE
28. nodejs callback don't understand how callback result come through argument - CSS PHP
29. Day 21 -23 , (Jan 10 – 12) | Mandy's road to code
30. javascript - Using the response from an asynchronous call with the Google Maps API - CSS PHP
31. javascript - What do you call this programming pattern? - javascript
32. Sauvegarde des News… – Nicolas Huleux codes
33. Going for the Dream full throttle Geek-Style | Designz by Leslie
34. Hack Reactor Remote-Getting Here | elizabeth sciortino
35. Solving callback pyramid with generators | Leslie -- Updates every Thursday
36. Dev Console showing TypeError on forEach (js)
37. Gerenciando o fluxo assíncrono de operações em NodeJS - Caderno de estudo TI
38. Gerenciando o fluxo assíncrono de operações em NodeJS -
39. Promises (& chains) – j-scripture

Leave a Reply

Comment:

Submit Comment

2.2

Current ye@r *

Notify me of follow-up comments by email.

Notify me of new posts by email.

 JavaScript is Sexy