← Javascript call and apply

node.js events →

10
2011.11

ategory
ascript
node.js

Tags
allback
ascript
node.js
tutorial

Javascript callbacks

What is a callback? A callback is a function to be executed after another function is executed. Sounds tongue-twisted? Normally if you want to call function `do_b` after function `do_a` the code looks something like

```
01 function do_a(){
02   console.log( ``do_a`: this comes out first');
03 }
04
05 function do_b(){
06   console.log( ``do_b`: this comes out later' );
07 }
08
09 do_a();
10 do_b();
```

Result

```
1 `do_a`: this comes out first
2 `do_b`: this comes out later
```

However javascript is an event driven language. If `do_a` takes longer than `do_b`, the result of `do_b` comes out first than `do_a`;

```
01 function do_a(){
02   // simulate a time consuming function
03   setTimeout( function(){
04     console.log( ``do_a`: this takes longer than `do_b`' );
05   }, 1000 );
06 }
07
08 function do_b(){
09   console.log( ``do_b`: this is supposed to come out after `do_a` but it comes out before `
10 }
11
12 do_a();
13 do_b();
```

Result

```
`do_b`: this is supposed to come out after `do_a` but it comes out before `do_a`
`do_a`: this takes longer than `do_b`
```

So how do we make sure `do_b` comes out after `do_a` in that situation? This is where callbacks comes in handy.

```
function do_a( callback ){
  setTimeout( function(){
    // simulate a time consuming function
    console.log( '`do_a`: this takes longer than `do_b`' );

    // if callback exist execute it
    callback && callback();
  }, 3000 );
}

function do_b(){
  console.log( '`do_b`: now we can make sure `do_b` comes out after `do_a`' );
}

do_a( function(){
  do_b();
});
```

> Result

```
`do_a`: this takes longer than `do_b`
`do_b`: now we can make sure `do_b` comes out after `do_a`
```

> Different ways of applying a callback

```
function basic( callback ){
  console.log( 'do something here' );

  var result = 'i am the result of `do something` to be past to the callback';

  // if callback exist execute it
  callback && callback( result );
}

function callbacks_with_call( arg1, arg2, callback ){
  console.log( 'do something here' );

  var result1 = arg1.replace( 'argument', 'result' ),
      result2 = arg2.replace( 'argument', 'result' );

  this.data = 'i am some data that can be use for the callback function with `this` key wor

  // if callback exist execute it
  callback && callback.call( this, result1, result2 );
}

// this is similar to `callbacks_with_call`
// the only difference is we use `apply` instead of `call`
// so we need to pass arguments as an array
function callbacks_with_apply( arg1, arg2, callback ){
  console.log( 'do something here' );

  var result1 = arg1.replace( 'argument', 'result' ),
      result2 = arg2.replace( 'argument', 'result' );

  this.data = 'i am some data that can be use for the callback function with `this` key wor

  // if callback exist execute it
  callback && callback.apply( this, [ result1, result2 ]);
}

basic( function( result ){
  console.log( 'this callback is going to print out the result from the function `basic`' )
  console.log( result );
});

console.log( '------------------------------------------------------------------------

( function(){
  var arg1 = 'i am argument1',
      arg2 = 'i am argument2';

```

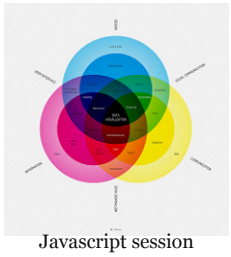
```
  callbacks_with_call( arg1, arg2, function( result1, result2 ){
    console.log( 'this callback is going to print out the results from the function `callba
    console.log( 'result1: ' + result1 );
    console.log( 'result2: ' + result2 );
    console.log( 'data from `callbacks_with_call`: ' + this.data );
  });
})();

console.log( '-------------------------------------------------------------------------------

( function(){
  var arg1 = 'i am argument1',
      arg2 = 'i am argument2';

  callbacks_with_apply( arg1, arg2, function( result1, result2 ){
    console.log( 'this callback is going to print out the result from the function `callbac
    console.log( 'result1: ' + result1 );
    console.log( 'result2: ' + result2 );
    console.log( 'data from `callbacks_with_apply`: ' + this.data );
  });
})();
```

Result

```
do something here
this callback is going to print out the result from the function `basic`
i am the result of `do something` to be past to the callback
-------------------------------------------------------------------------------------
do something here
this callback is going to print out the results from the function `callbacks_with_call`
result1: i am result1
result2: i am result2
data from `callbacks_with_call`: i am some data that can be use for the callback function w
-------------------------------------------------------------------------------------
do something here
this callback is going to print out the result from the function `callbacks_with_apply`
result1: i am result1
result2: i am result2
data from `callbacks_with_apply`: i am some data that can be use for the callback function
```

Now we know how to **execute node.js code, using** `require` **and** `exports`, the importance of **function scopes and closures** and callbacks. Next we are going to take a look at the most important thing in node.js – **Events**.

Related posts



Centralize HTML DOM element with jQuery Center plugin

jQuery blockUI alternative with jQuery MSG plugin

Javascript session

How to setup a node.js development environment on Ubuntu 11.04

Javascript call and apply

← Javascript call and apply

node.js events →

Risk Taker • 2 months ago

thanks for the tutorial but it could be better. seeing that this is an intro to callback, the author should not include obscure syntax like - callback && callback(). it's distracting and should be replaced with if(callback) { callback(); }. at least i think that's what "callback && callback()" means. also in the second example of callback usage, it's too long and distracting, please keep your examples short and to the point. we don't have a good grasp of what callback is yet, please don't make us read through long, complicated examples. this makes the concept of what we're trying to learn seem more complicated than it actually is.

Reply • Share ›

Alex Margulis ➚ Risk Taker • 10 days ago

can not agree with you more!
But moving from "criticism" mode to some more productive level i will try to explain
Let say we have designed the function which takes 3 parameters
function anyFunction (arg1, arg2, arg3){
console.log ("Arg 1: " + arg1);
console.log ("Arg 2: " + arg2);
console.log ("Arg 3: " + arg3);
}
When this function is called the number of arguments theoretically may differ
There will be no any run-time error!!!
anyFunction (4,5) // Arg 3 remains undefined
anyFunction (4,5,6,9) // all additional args will be simply ignored

Back to the original line in question.
Logically it says:
"If there is a 3rd parameter provided and this parameter is a function, only then execute this function"
function anyFunction (a, b, callback) {
// some code here....
if (callback && typeof (callback) ==="function") {
callback();
}

Reply • Share ›

RizonBarns • 3 months ago

sometime i saw object like this:

```
function myOwn (param) {
param(num);
}

myOwn( (num) => {
alert(num + 2);
});
```

What symbol '=>' means?

Reply • Share ›

Susan • a year ago

There's an error in this case:

```
```

```
function do_a(){
// simulate a time consuming function
setTimeout( function(){
console.log( ``do_a`: this takes longer than `do_b`' );
}, 1000 );
}

function do_b(){
console.log( ``do_b`: this is supposed to come out after `do_a` but it comes out before `do_a`' );
}

do_a();
do_b();
```

The reason do_b finishes executing first is not because do_a takes longer. It is because do_a contains one of the few

see more

Reply • Share ›

Fabio Souza • 2 years ago

thanks!

Reply • Share ›

Jeff Hendricks • 2 years ago

Nice example until you got all out of sorts with you last example. Either change the background color or color the code. Otherwiese...it's good

Reply • Share ›

krmfla • 2 years ago

寫的很清楚, 太感謝了!

Reply • Share ›

Cyprien • 3 years ago

Great article and thanks! Quick note however. Towards the beginning, the real reason why the output of a function (ex. do_b) may come before the output of a more time consuming function (do_a) is because of the asynchronous nature of JavaScript. In other words function do_a will not 'block'.

Reply • Share ›

kevin • 3 years ago

font is bad for code. I stopped reading half way. Colors would help

Reply • Share ›

Natalia Dushkina(Smirnova) • 4 years ago

this is a very nice article, thank you.

Btw I think that you can simplify the code and instead of "do_a(function(){do_b()});" just write "do_a(do_b);"

Reply • Share ›

Nic • 4 years ago

Although JavaScript has event driven functions, it is definitely thread safe which means it always run in single thread. Any running functions block the execution of other functions, even for even driven functions. So the claim "Previous running function finishs later because JavaScript is event-driven" is definitely wrong. That is to say, two consecutive statements will not start running simultaneously.

Try this out:

```
function do_a(){
// 模擬一個需要長間的 function
var c=0;
for(var i=0;i<1000000;i++)
c++;
console.log( ``do_a`: 這個需要的時間比 `do_b` 長' );
}

function do_b(){
console.log( ``do_b`: 這本來應該出現在 `do_a` 之後但是卻先出現了' );
}
```

do_a();
do_b();
1 • Reply • Share ›

Jibran Javed • 4 years ago
I have read that Javascript interpretation is by default is blocking(synchronous). The problem is to make that Asynchronous non blocking and that's where callbacks functions are handy; your content seems delivering contrary or my understanding is still narrow ;-)
• Reply • Share ›

ben Mod ➚ Jibran Javed • 4 years ago
Try to run the examples and you'll see.
• Reply • Share ›

Dan • 5 years ago
Thanks for your example
I am at the moment trying to simulate a REAL JUKEBOX
in a realJuke box you used to drop coins, select a song and it started immediately then you could add other coins and select other songs which were played later once the first song was finished etc..
I am trying to do that with javascript . I have been advised to use a javascript queue . I didi it : I click the first song it starts
immediately in the queue, but when I click the second one , it starts immediately tooand stops the previous one. I unsderstand that using a call back could solve the problem : the inconvenient is that the callback is the same function : actually with 4 songs I click Playsong 1, then play song 2 etc which leads to a call to the same function but with a varying number ! I don't know how to use it
Have you any solution for me
Thanks
Dan www.bresseblues.com
• Reply • Share ›

ben Mod ➚ Dan • 5 years ago
The code would look something like this https://gist.github.com/365...
• Reply • Share ›

adison wu • 5 years ago
// 如果 callback 存在的話就執行他 callback &&
請問上面的callback &&是程式碼之一嗎?
• Reply • Share ›

ben Mod ➚ adison wu • 5 years ago
Yes, 他等同於 if(callback) callback();
• Reply • Share ›

Subscribe Add Disqus to your siteAdd DisqusAdd Privacy