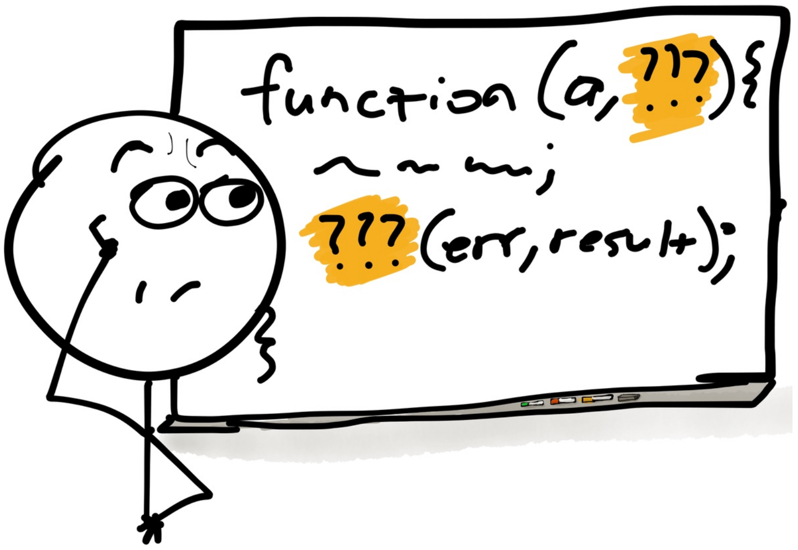
What Do You Name The Callback Function?

February 15, 2016 By derickbailey

JavaScript practically requires callback functions to do anything asynchronous. Unless you're talking about generators, you really can't get away from them. Callbacks are everywhere.



But someone recently asked a question about naming callback methods:

> *Do you use different / standard callback names in different scenarios? For example, "next" for Express middleware, maybe "cb" or "callback" for other methods in Node?*

Yes, I do use diferent names for the callback parameter, depending on the circumstance. I believe naming is important as it provides a set of expectations. If your code breaks the expectations of the name, people will be confused and it will cause problems.

Common Callback Names

Early in my Node days, I would name every callback "cb" – short for "callback". There are still times when I use this name, but it's not my default anymore. Rather, I have a short list of parameter names to go with the following scenarios:

"cb" (short for "callback")

This is a general purpose name to use when you are creating an API that does asynchronous work, and allows the consumer of the API to receive the results of your work.

In node, the standard callback method includes an error as the first argument. This requires the consumer of the API to handle error scenarios before they can get data or whatever result from the API call.

```
var someThing  = {

  // define the callback
  doSomethingAsync : function (cb){

    //simulate some async work
    setTimeout (function (){

      // use the generic callback method
      cb(undefined , "some result ");

    }, 10);
  }
};

// use the API
someThing .doSomethingAsync (function (err, result ){
  if (err) { ... }
  ...
});
```

1.js hosted with ❤ by **GitHub** **view raw**

Examples include loading data from a database, making an HTTP API call, sending a message to RabitMQ, etc. This is easily the most common name for callbacks, as most API development uses this kind of pattern.

“next”

Middleware implementations need a way to continue on to the next middleware method, when required. The most common name for this is “next”, and a single call to it will move the code forward.

Beyond the specific use case, there is a significant difference in the implementation compared to generic “cb” callback methods. The “next” function is usually supplied by the middleware API, for the API consumer to call, which is the opposite of the typical “cb” callback.

```
var express = require ("express ");
var router = new express.Router ();

// use the api that provies a "next" callback
router .get ("/", function (req, res, next ){
  // do some work, then move on to next middleware
  next ();
});
```

2.js hosted with ❤ by **GitHub** **view raw**

A common example of this is Express.js routers. I've also used the generic-middleware library to create my own Express-like middleware features in my Rabbus library. There are other implementations of middleware around, but the Express model is frequently used.

"done"

This method is similar to "next" in that it is provided by an API for the consumer code to call. Unlike the "next" method, though, it doesn't move on to the next chunk of code to execute. Rather, it signals the completion of the current code, allowing the API internals to clean up any initialized resources it holds.

```
describe ("some test ", function (){

  // use the API that provides a "done" callback
  beforeEach (function (done ){

    // do something async
    myStuff .doThings (function (){

      // tell the test we're done
      done ();
    });

  });

  it("...", function (){
    ...
  });

});
```

3.js hosted with ❤ by GitHub view raw

A common example is asynchronous unit tests, as shown above. Jasmine needs to know when the "beforeEach" and "it" assertion blocks have completed. By providing a "done" method as part of the API, the developer writing the test can tell Jasmine that the code is complete.

My Ocarina library uses this pattern to clean up Oracle cursors and connections, as another example.

Other Callback Names

The above are three common callback names that I use in my code, whether I am creating an API to be consumed, or consuming an API with my code.

This is not an exhaustive list of callback names, howeve.lr have written plenty of code that used other names, and I'm sure I will do so again.

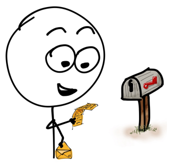
For example, my Rabbus libary provides an "actions" object for several parts of its API. This object is little more than a collection of callback methods that the API provides to the consumer code.

```
var Rabbus = require("rabbus");

var receiver = new Rabbus.Receiver({
  ...
});


// use the API which provides both a "next" callback
// and a series of other callbacks collected in "acti
receiver.use(function(message, properties, actions

  // what should we do?
  if (some condition){
    // use an action callback
    actions.ack();
  } else {
    // move on to the next middleware
    next();
  }

});
```

4.js hosted with ❤ by GitHub view raw

I’m sure there are dozens of other scenarios in which a new method name would be appropriate, as well.

So, what are your common callback method names, and in what scenarios are those named used?



Frustrated by JavaScript's "this"?

Get the FREE, 6 day email course packed with the secrets to the most notorious keyword in the JavaScript language.

Includes ES2015 Arrow Functions!

enter your email ad

SEND ME THE 6 RULES!

Tweet

RELATED POST

The Docker Management Cheatsheet

What Is RabbitMQ? What Does It Do For Me?

Callbacks First, Then Promises

Ending the Nested Tree of Doom with Chained Promis...

Does ES6 Mean The End Of Underscore / Lodash?

Filed Under: Callbacks, Code, Context, ExpressJS, Functions, JavaScript, Naming, NodeJS

About derickbailey

Derick Bailey is a developer, entrepreneur, author, speaker and technology leader in central Texas (north of Austin). He's been a professional developer since the late 90's, and has been writing code since the late 80's. In his spare time, he gets called a spamming marketer by people on Twitter, and blurts out all of the stupid / funny things he's ever done in his career on his email newsletter.

DERICK BAILEY AROUND THE WEB

Twitter: @derickbailey

Google+: DerickBailey

Screencasts: WatchMeCode.net

eBook: Building Backbone Plugins

Copyright © 2016 Muted Solutions, LLC. All Rights Reserved · Log in