JavaScript (http://www.sitepoint.com/javascript/)

Understanding module.exports and exports in Node.js

(http://www.sitepoint.com/author/ckim/)

Cho S. Kim (http://www.sitepoint.com/author/ckim/)

Published July 18, 2014

As developers, we often face situations where we need to use unfamiliar code. A question will arise during these moments. How much time should I invest in understanding the code that I'm about to use? A typical answer is learn enough to start coding; then explore that topic further when time permits. Well, the time has come to gain a better understanding of `module.exports` and `exports` in Node.js. Here's what I have learned.

What is a Module

A module encapsulates related code into a single unit of code. When creating a module, this can be interpreted as moving all related functions into a file. Let's illustrate this point with an example involving an application built with Node.js. Imagine that we created a file called `greetings.js` and it contains the following two functions:

```
1 // greetings.js
```

```
sayHelloInEnglish = function() {
  return "Hello";
};

sayHelloInSpanish = function() {
  return "Hola";
};
```

Exporting a Module

The utility of `greetings.js` increases when its encapsulated code can be utilized in other files. So let's refactor `greetings.js` to achieve this goal. To comprehend what is actually happening, we can follow a three-step process:

1) Imagine that this line of code exists as the first line of code in `greetings.js` :

```
// greetings.js
var exports = module.exports = {};
```

2) Assign any expression in `greetings.js` that we want to become available in other files to the `exports` object:

```
// greetings.js
// var exports = module.exports = {};

exports.sayHelloInEnglish = function() {
  return "HELLO";
};

exports.sayHelloInSpanish = function() {
  return "Hola";
};
```

In the code above, we could have replaced `exports` with `module.exports` and achieved the same result. If this seems confusing, remember that `exports` and `module.exports` reference the same object.

3) This is the current value of module.exports:

```
module.exports = {
  sayHelloInEnglish: function() {
    return "HELLO";
  },

  sayHelloInSpanish: function() {
    return "Hola";
  }
};
```

Importing a Module

Let's import the publicly available methods of `greetings.js` to a new file called `main.js`. This process can be described in three steps:

1) The keyword `require` is used in Node.js to import modules. Imagine that this is how `require` is defined:

```
var require = function(path) {

  // ...

  return module.exports;
};
```

2) Let's require `greetings.js` in `main.js`:

```
// main.js
var greetings = require("./greetings.js");
```

The above code is equivalent to this:

```
// main.js
var greetings = {
  sayHelloInEnglish: function() {
    return "HELLO";
  },
```

```
  sayHelloInSpanish: function() {
    return "Hola";
  }
};
```

3) We can now access the publicly available methods of `greetings.js` as a property of our `greetings` variable in `main.js`.

```
// main.js
var greetings = require("./greetings.js");

// "Hello"
greetings.sayHelloInEnglish();

// "Hola"
greetings.sayHelloInSpanish();
```

Salient Points

The keyword `require` returns an object, which references the value of `module.exports` for a given file. If a developer unintentionally or intentionally re-assigns `module.exports` to a different object or different data structure, then any properties added to the original `module.exports` object will be unaccessible.

An example will help elaborate this point:

```
// greetings.js
// var exports = module.exports = {};

exports.sayHelloInEnglish = function() {
  return "HELLO";
};

exports.sayHelloInSpanish = function() {
  return "Hola";
};

```

```
/*
 * this line of code re-assigns
 * module.exports
 */
module.exports = "Bonjour";
```

Now let's require `greetings.js` in `main.js`:

```
// main.js
var greetings = require("./greetings.js");
```

At this moment, nothing is different than before. We assign the variable `greetings` to any code that is publicly available in `greetings.js`.

The consequence of re-assigning `module.exports` to a data structure other than its default value is revealed when we attempt to invoke `sayHelloInEnglish` and `sayHelloInSpanish`:

```
// main.js
// var greetings = require("./greetings.js");

/*
 * TypeError: object Bonjour has no
 * method 'sayHelloInEnglish'
 */
greetings.sayHelloInEnglish();

/*
 * TypeError: object Bonjour has no
 * method 'sayHelloInSpanish'
 */
greetings.sayHelloInSpanish();
```

To understand why these errors are occuring, let's log the value of `greetings` to a console:

```
// "Bonjour"
console.log(greetings);
```

At this point, we are trying to access the methods `sayHelloInEnglish` and `sayHelloInSpanish` on the string "Bonjour." `module.exports`, in other words, is no longer referencing the default object that contain those methods.

Conclusion

Importing and exporting modules is a ubiqutous task in Node.js. I hope that the difference between `exports` and `module.exports` is clearer. Moreover, if you ever encounter an error in accessing publicly available methods in the future, then I hope that you have a better understanding of why those errors may occur.


(http://www.sitepoint.com/author/ckim/)

Cho S. Kim (http://www.sitepoint.com/author/ckim/)

Cho is a full-stack web-application developer. He dislikes mean people but likes the MEAN stack (MongoDB, Express.js, Angular.js, Node.js). During a typical week, he'll be coding in JavaScript, writing about JavaScript, or watching movies NOT about JavaScript. Here's his blog (http://www.choskim.me/).

(https://twitter.com/choskim)
(https://plus.google.com/110528360124637826614)

You might also like:

Understanding ES6 Modules (http://www.sitepoint.com/understanding-es6-modules/)

Book: Jump Start JavaScript

[(https://learnable.com/books/jump-start-javascript?utm_source=sitepoint&utm_medium=related-items&utm_content=js-javascript)](https://learnable.com/books/jump-start-javascript?utm_source=sitepoint&utm_medium=related-items&utm_content=js-javascript)

[Using npm link in Node.js (http://www.sitepoint.com/using-npm-link-node-js/)](http://www.sitepoint.com/using-npm-link-node-js/)

Dave McFarland • 8 months ago

I'm sorry I don't get it. What IS the difference between exports and module.exports? It sounds like they are the same thing: "In the code above, we could have replaced exports with module.exports and achieved the same result. If this seems confusing, remember that exports and module.exports reference the same object."

12 ^ | v • Reply • Share ›

Cho S. Kim ➚ Dave McFarland • 8 months ago

Dave,

Thanks for leaving a comment. Both module.exports and exports initially reference the same object:

var exports = module.exports = {};

But this could change if a developer re-assigns module.exports or exports to a different object (or entirely different data type). This usually occurs when developers use exports and module.exports interchangeably and incorrectly to make assignments in the same file.

Here's the difference between exports and module.exports: When a module is being required in a file, what's being returned from the require function is the value of module.exports. So this is worth repeating, the difference is that the require function returns only the value of module.exports, which initially references an object.

2 ^ | v • Reply • Share ›

nor ➚ Dave McFarland • 4 months ago

The principal difference is what module.exports replace the content of the exports only if exports previously has an value and this is only valid if you have the declaration exports first and then you write module.export. In summary use one of it not both.

1 ^ | v • Reply • Share ›

Andrew Hung • 6 days ago

Great explanation, Cho. Thanks for writing this. Definitely cleared things up for me.

^ | v • Reply • Share ›

Sergey Gulidov • 11 days ago

Thanks, it was useful for me)

^ | v • Reply • Share ›

Luke Swart • 14 days ago

Thanks for the post! We need more elegant explanations for ubiquitous topics such as module exports.

^ | v • Reply • Share ›

ConfusedCoder • a month ago

Thank you! Simple and clear, great tutorial!

^ | v • Reply • Share ›

Darnell Filliams • a month ago

Thank you very much. This was very clear and well-written.

• Reply • Share ›

Jahanzaib Aslam • 2 months ago

Simple and to the point. Thanks

• Reply • Share ›

Vengatesh TR • 3 months ago

wow. beautiful & detailed explanation. Thanks :)

• Reply • Share ›

Cho S. Kim ↗ Vengatesh TR • 2 months ago

Thanks for the positive feedback.

1 • Reply • Share ›

Mustafa Gamal • 4 months ago

Thanks for the simplicity :)

• Reply • Share ›

Cho S. Kim ↗ Mustafa Gamal • 2 months ago

:).

• Reply • Share ›

Alex • 4 months ago

Thank so much for this very clear presentation. Very helpful to me.

• Reply • Share ›

Cho S. Kim ↗ Alex • 2 months ago

I'm happy that it was useful.

• Reply • Share ›

Quynh Le • 4 months ago

Thank you very much. It is clear now for me.

• Reply • Share ›

Cho S. Kim ↗ Quynh Le • 2 months ago

You're welcome :).

• Reply • Share ›

jokeyrhyme • 8 months ago

I've found it less confusing to just avoid the `exports` variable altogether.

• Reply • Share ›

Bushwazi ↗ jokeyrhyme • 7 months ago

Can you explain this more? I tried changing `exports.sayHelloInSpanish = function() {return "Hola";}` to `sayHelloInSpanish = function(){return "Hola";}` and I got an error. And

`console.log(greetings);` just returned and empty object {}. Do you have to structure greetings.js different to make it work this way?

• Reply • Share ›

Cho S. Kim ➚ Bushwazi • 7 months ago

Bushwazi,
I believe that jokeyrhyme's initial comment could have been misinterpreted for avoiding the use of both exports and module.exports. He clarified what he meant in his reply to your question (using module.exports instead of exports), but I'll try to answer your question, too.

Question 2 of 2: console.log(greetings);
Imagine this is the file we are talking about and the comments represent code that's implied:

```
// var module.exports = {};

sayHelloInSpanish = function() {
return "Hola";
};

// module.exports = {};
```

When you require this file in a different file, you're returning module.exports. In this case, you never added a method called sayHelloInSpanish to module.exports. So this explains why an empty object is being returned in the console.

Question 1 of 2: you received an error
This may be in reference to a syntactical error. I can confirm this if you provide more information, such as the code in the file you are using. Or you can tell me the type of error; however, seeing the file is probably the easiest approach.

I hope this helps.

• Reply • Share ›

Bushwazi ➚ Cho S. Kim • 7 months ago

Sorry, it's obvious now that I look again. jokeyrhyme's just saying he prefers this pattern (is this a constructor/ object initializer?):

```
module.exports = {

methodName: function(){}

...

}
```

and you were showing a different pattern (and this is "Dot syntax"?):

```
exports.mathodName = function(){};
```

to accomplish the same thing.

Let me know if I still don't get it...

• Reply • Share ›

Cho S. Kim ➚ Bushwazi • 7 months ago

Yes, he was using an object literal to add methods and I was using dot notation to add methods. Both syntax works, but I choose the later because adding a lot of methods to an object literal delays the end of the object literal (closing curly brace) for many lines of code.

I hope this helps.

• Reply • Share ›

Bushwazi ➚ Cho S. Kim • 7 months ago

Yeah man, this was great. Thank You!

• Reply • Share ›

jokeyrhyme ➚ Bushwazi • 7 months ago

This article looks at exports and module.exports.

Given the choice, I avoid exports, and only use module.exports.

• Reply • Share ›

Cho S. Kim ➚ jokeyrhyme • 8 months ago

That makes a lot of sense.

• Reply • Share ›

joe • a month ago

var module.exports = { } is a syntax error. You can't use member variables in var declaration.

• Reply • Share ›

Cho S. Kim ➚ joe • a month ago

Hey, Joe. I believe that you've misread the syntax:
var exports = module.exports = {};
Considering the syntax above, the pointers are assigned from right to left. So the code works like this:

About

About us (/about-us/)
Advertise (/advertising)
Press Room (/press)
Legals (/legals)
Feedback (mailto:feedback@sitepoint.com)
Write for Us (/write-for-us)

Our Sites

Learnable (https://learnable.com)

Connect

   (/feed) (/newsletter) (https://www.facebook.com/sitepoint)
  (http://twitter.com/sitepointdotcom)
(https://plus.google.com/+sitepoint)