Javascript Module Syntax

Kevin Charles, 2020

github.com/kevinchar93/js_exec_model_and_module_types_presentation

Note some information in this talk / script has been removed for it to be used outside my workplace

in this talk i'll be covering javascript modules briefly and the different syntax available, because we can use acronyms like AMD/ES6 while we are discussing things regarding [redacted] and it's useful for others to picture what we mean

Modules in Javascript

- Modular code is good
- Easy to understand
- JavaScript didn't have built-in modules originally
- Support for modules essential as code grows
- Important for libraries and frameworks

so we all know that organising our code into modules is good practice because it makes things like code reuse and testing a lot easier

it's also just easier to understand a program when it separated into smaller parts rather than just one massive file

for a long time javascript didn't have a built in language feature for separating code into modules,

and this isn't a problem when scripts are small but as things grow, organising code becomes essential to be able to actually maintain the code going forward

this is particularly important when it comes to using libraries or frameworks, as you could end up with name collisions if there isn't some sort of standard practice for separating code into modules

Established Standards

- Community invented module standards
- AMD (asynchronous module definition)
- CommonJS
- ES6 (ECMA Script 6) now built in

As I mentioned earlier JavaScript didn't have a built-in language feature for separating modules, so the JavaScript community invented different ways of organizing code into modules

two of the most established standards are
* AMD (which stands for Asynchronous Module Definition)
* CommonJS

and
Javascript now has a built in module syntax called ES6 modules (ECMA Script 6 modules)

i'll give you some details on each of the standards

AMD (Asynchronous Module Definition)

- Asynnchronous module loading
- Define module using function
- Typically use library called Require JS

* so we have AMD , which as the name implies it allows you to define modules that can be loaded asynchronously, even if they depend on one another

* so this method uses a function where you define a modules dependencies, then return anything that module wants to make public

* typically when you're using AMD you will use a library called require.js to do the loading for you

CommonJS

- Used in Node JS
- Keywords are require & export

* common js is the module standard typically used in node js apps (so that's server side javascript)

* in this standard you make use of two keywords require & export
* require is used to synchronously load a module
* and export is used to expose anything that you want to make publicly available from a module

ES6 Modules (ECMA Script 6)

- Built in
- Introduced 2015
- Keywords import & export
- Designed to feel C style-like

* ES6 modules are built into the javascript language and were added to the standard in 2015

* it uses the keywords import to define what a module needs and export, to define what a module makes public

* it's designed to feel very similar to importing dependencies in other c style languages

Simple Example

- Demonstrate simple module
- 2 dependencies
  - sayHello
  - sayWorld
- Expose a function
  - sayHelloWorld

so in these examples i'm going to show you what each module syntax looks like with a really simple module

This module will use two dependencies , sayHello & sayWorld

and expose a function called sayHelloWorld

Simple Example - AMD

This is an amd module

First at arrow number 1 we have to define where the dependencies are with a path and the name of a file

Then at number 2 we need to define what the module will be called when it's imported , we keep the names the same as the file here (sayHello and sayWorld)

but you could change the names here

At arrow number 3 we then create an object to hold our module's exports

Then at number four we define the function sayHelloWorld

Finally at number five you return the module

Simple Example - CommonJS

This is the same module in Common JS syntax

At arrow number 1 we import the dependencies using the require keyword, here we don't have to use a file name because you usually have another file that defines where to find each module from the name you give to require

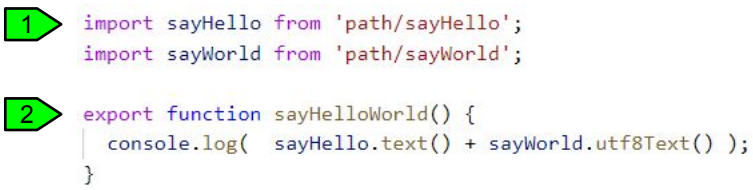
also notice we can name what the module is referred to (again we choose to keep the same names)

Then at number 2 we define a function myModule that will be our exported module

At number 3 we define the function sayHelloWorld

and finally at number 4 we set the exports for the module by setting it to the function myModule

Simple Example - ES6 Module

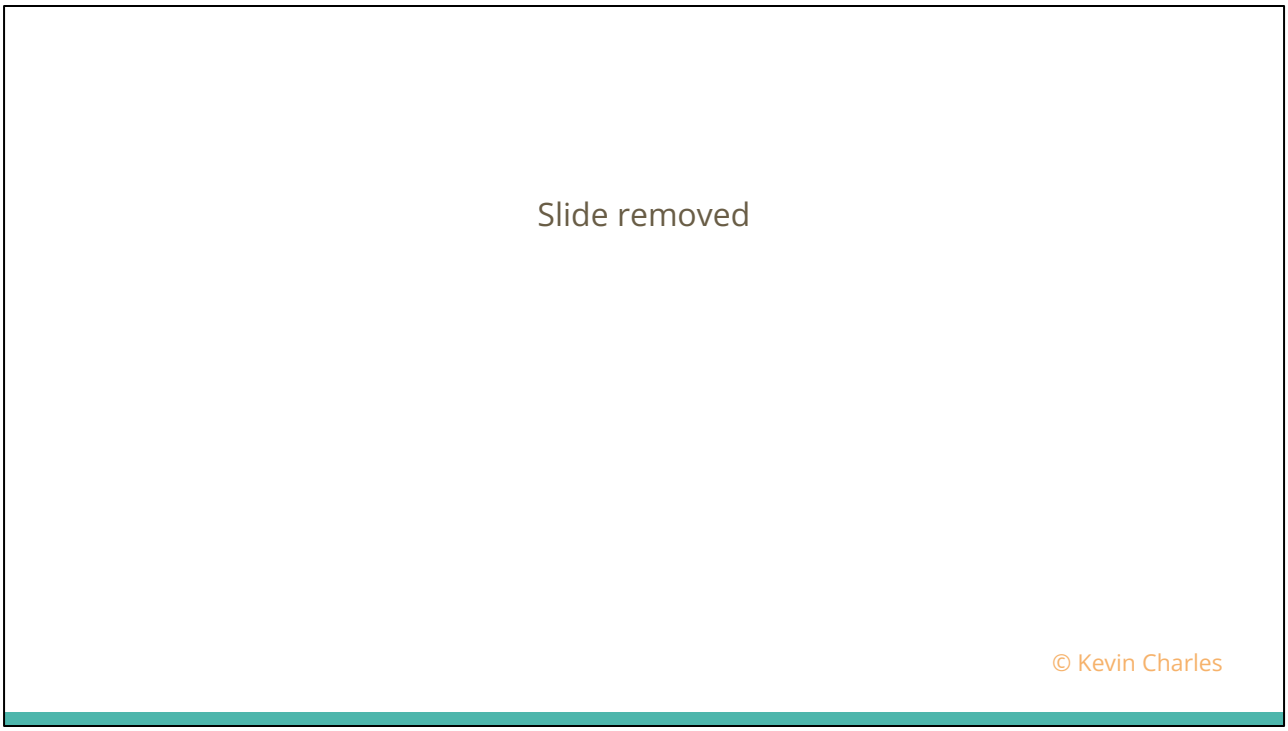


This is the module in ES6 syntax

again we start out at arrow number 1 by importing the dependencies using the import keyword

you give a path to the file name, and then name what you want the module to be referred to

Then at number 2 we export the function sayHelloWorld to make it publicly available




```
Slide removed
```

Thanks