Episode-05 → Diving into the Node.js github repo

Q.> Whenever we are requiring any module, what happens behind the scene?

And

If we have a variable, that variable remains in private space of module, how this happens, how Node.js is able to do this ?

A.> In JS if we create a function, all the variable and function inside the function (what is there inside the function), it is privately scoped.

```
function x() {
    const a=10;
  function y() {
  console.log ("b");
  }
}
```

- console.log (a); → we won't be able to access outside the function.
- function has it's own scope.

- Whenever we are wrapping some piece of code inside a function, now the scope of that variable and function can only be accessed inside, not outside the function.

- Modules work the same way in JS, in Node.js when we create a new module all that code we write inside module is wrapped inside a function and then executed.

- That's why we can't access variable and function outside the module until we export it.

- When we require the module, what happens is that the Node.js takes the code from this file, wraps it into a function and then execute it, so when it is wrapped inside a function it will not interfere with the other things.
When we call "require" method and gives it a path, it just wraps all the code and then run it.

- And when it wraps the code inside a function, now this function is not a normal function. This function is special function and that function is known as IIFE.

- IIFE - Immediately Invoked Function Expression.

- It is an anonyms function and which we invoking immediately.

-
```
(function () {

})();
```

- After calling require method, Node.js will take all the code, it will wrap inside an IIFE and then it will give it to V8.

Q.) Why do we need an IIFE ?

A.) It solves the multiple purposes.

- It immediately invokes the code.
- It keeps variable & functions safe and private, it doesn't interfere.

ex → Variable with same name can exist with an IIFE and also outside of it. Both code (variable) are independent from eachother.

Q.) How are variables and functions are private in different module? Why can't we access them ?

A.) Because of IIFE & 'require' method.

- 'require' statement wraps our code in IIFE.

Q.) How do we get access to "module.exports" ? Where does the "module.exports" come from ?

A.) Because when our code is wrapped inside a function, that function has a parameter "module.exports" over there.

So when we invoke it, there is a module being passed, it's an empty object.

module.exports is an empty object.

ex →

```
( function (module) {

        // code

    module.exports = {calculateMultiply};

}) (module.exports = { });
```

- Suppose we require any file, this require is also passed over there.

- (function (module, require) {

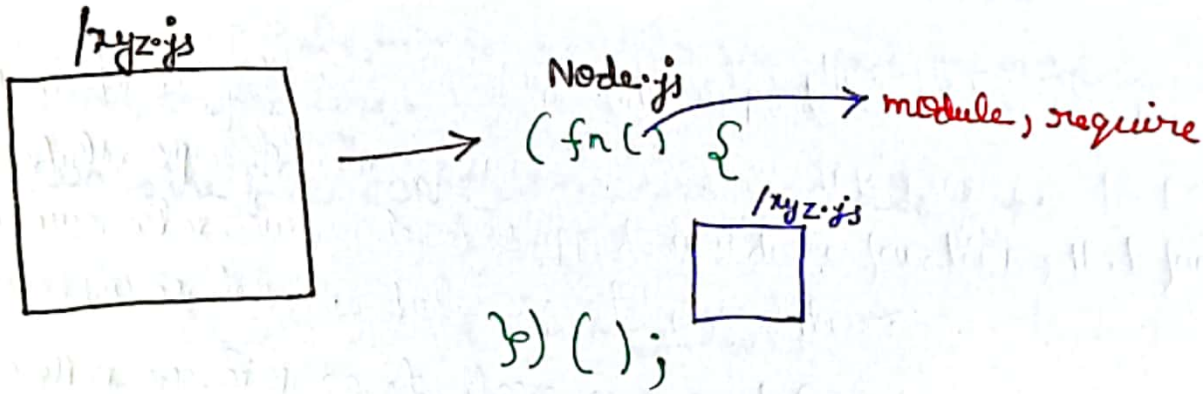
  require (/path)

  =

  }

- So, at the end of the day, the code that we write will be wrapped inside a function have these parameters - module & require

- These 'module' & 'require' are given by Node.js.



Then code is passed to the V8 Engine.

- 5 Steps mechanism of "require" →

- Wherever we do a require of some path, what happens behind the scene.

- There are 5 steps happen to get the module & execute the code.

1. Resolving the module → That means, it checks whether the module is 'local path' or this is a 'json path' or this is 'node colon' internal module, there are different types of require statement.

So, every require needs to do something different, every different path need different logic.

ex → ./local path

.json

node: module

- So, in first step it seas that what type of data it is coming, accordingly it resolves the module.

2. Loading the module → So how do we load the module, basically it depends on the type.

It loads the file content according to file type.

3. Wraps inside an IIFE → compile

4. code Evaluation → In this step "module.exports" happens, "module.exports" is returned.

5. Caching → The module is cached

- This is very important step.

- Suppose if there are multiple modules, if app.js requires - ./xyz.js and now sum.js also requires the same - require("../xyz.js")

  multiply.js also requires ./xyz.js, so what will happen that there are multiple files requiring multiple modules, so node.js caches the require. So basically, what happens is the code of xyz.js will run only once.

- Suppose if next time sum.js is requiring xyz.js, it will just return from the cache, it will not do the 5 steps again, it will just return from the cache.

- In real world projects there are hundreds of files which are requiring 100 of modules and there are so many modules that are required by so many files, so if we keep loading the same file once again in every module then it will become a mess, application will become very slow.
  So, Node.js makes it very efficient by caching it up.

- This is behind the scenes how require works.

- Open Source Node.js repository →
- go to Node.js repository.
- then go to 'lib' folder (it's a library folder)
  ↳ all the majority JS code is in Lib folder
- In 'lib' folder we can find all sorts of JS files, functions and everything we can use.

- 'setTimeout' lies in 'timers' files of 'lib' folder.

- timer.js is also a module

- lib → internal → modules

  we find two types of module there - CJS & ESM

- ESM → helper.js

'helpers.js' is the most important file, here we will find the actual implementation of the require method.

→ "makeRequireFunction" → This function in helper.js is building the require function.

- This 'makeRequireFunction' returns our most used require function.

- In the code of "makeRequireFunction" we can see our 5 steps mechanism of require.

- Cjs → loader.js → code of require function

- Note → whenever we are inside the github code, after pressing the '.' (fullstop / dot) button, it takes us to the code Editor, basically it makes our github into dev mode, Over there it will load all the file inside the code editor.

For more PDF, Follow

gitHub → rajeshjha2000