**Chapter 2**

**Core Node JS**

**Node JS**

In this chapter, we will be looking at the key built-in modules and features of Node JS.

**Node.js File-Based Module System**

Kevin Dongaoor created CommonJS in 2009 with the goal to specify an ecosystem for JavaScript modules on the server. Node.js follows the CommonJS module specification. Following are a few main points of the module system:

• Each file is its own module.

• To export a current module, use module.exports.

• To import a module, use the require function

**Module.exports**

Let’s take a look at an example of how we can export a file as a module and use it in another file.

Consider the following code which is in a file formula.js

|  |
| --- |
| multiplyFunct = (x, y) => x \* y;  module.exports=multiplyFunct |

By using exports.module, we are able to export the function. Now in another file, if we want to use the function, we can import it using the require function as below:

|  |
| --- |
| var mulFn = require('./formula');  console.log(mulFn(2,3)); |

The above example shows how you can export a single object(which is a function), but what if we have multiple functions in a single file? The below example illustrates how we can export multiple objects:

|  |
| --- |
| multiplyFunct = (x, y) => x \* y;  addFunct = (x, y) => x + y;  module.exports.mulFn=multiplyFunct;  module.exports.addFn=addFunct; |

We would then import the file in the same way and call the corresponding function accordingly.

|  |
| --- |
| var formula = require('./formula');  console.log(“Multiply 2x3=”+formula.mulFn(2,3));  console.log(“Add 2+3=”+formula.addFn(2,3)); |

However what happens if we have defined many functions within one file? If we use the above method, there would be many module.exports statements which causes the code to be rather cumbersome.

One approach to solving this issue is to declare a single object which has many functions defined inside it. Then we export that one single object.

Let’s do this as an exercise now.

|  |
| --- |
|  |

**Require function**

The Node.js require function is the main way of importing a module into the current file.

When we make a require call with a relative path—for example, something like require('./filename'), Node.js runs the destination JavaScript file in a new scope and returns whatever was the final value for module.exports in that file.

The items that we intend to export from a module should be attached to the module.exports variable. It is important to note that module.exports is already defined to be a new empty object in every file. That is, module.exports = {} is implicitly present. By default, every module exports an empty object, in other words, {}.

When we want to import another file/module, take note that the file is referenced with a relative path. To have a good idea on path referencing, we take a look at another example.

Consider the below file structure:

|  |
| --- |
|  |

If we have code in Program.js that wants to imports in myFile.js, we can use the relative path “./myFile.js” since both files are in the same directory.

However if we have file0.js which wants to import myFile.js, then we need to use the relative path “./SubFolder/myFile.js”.

require(‘./SubFolder/myFIle’);

If we need to go one level up the directory, we can use “../”.

If say I want to import file0 into Program.js.

So in Program.js:

require(“../file0”);

**Modules, Callbacks and Error Handling**

Now that we have a better understanding of modules, callbacks and error handling in node js modules, let’s combine these 3 concepts and work on the formula example by extending some constraints.

Let’s assume the multiply function contains a “complex” I/O asynchronous operation requiring much time and also we do not allow any “infinity” input values for the function arguments, and any attempt to pass in “infinity” values should return an error to the caller.

Let us now modify the code to show how callbacks can be used in this scenario.

|  |
| --- |
| formula={  mulFn: (x, y,callback) => {  if (x == Infinity || y == Infinity)  setTimeout(() =>  callback(new Error("Input values cannot be infinity!"),  null),  2000);  else  setTimeout(() =>  callback(null, x\*y  ),  2000);  },  addFn: (x, y) => x + y  };  module.exports=formula; |

|  |
| --- |
| var formula = require('./formula');  formula.mulFn(2,3,(err,result)=>{  if (err)  console.log(err.message);  else  console.log(result);  }    );  console.log(“Waiting for result…”); |