**02 Node Module System**

**1) Global Object**:

Node.js global objects are global in nature and they are available in all modules. We do not need to include these objects in our application, rather we can use them directly.

**Example**:

1. \_\_dirname
2. \_\_filename
3. Console
4. Process
5. Buffer
6. setImmediate(callback[, arg][, ...])
7. setInterval(callback, delay[, arg][, ...])
8. setTimeout(callback, delay[, arg][, ...])
9. clearImmediate(immediateObject)
10. clearInterval(intervalObject)
11. clearTimeout(timeoutObject)

**2) Modules**:

When we declare a function or variable that is added to the global scope may causes a problem. In real world application we are working with a lot of function, variable or file name. It is possible that we have two files or variable or functions have the same name.

As the function is define in the global scope, when we declare the function in another file the previous definition will be overwrite with the new definition. This is the problem in the global scope.

For resolve this type of problem we use module.

Module in Node.js is a simple or complex functionality organized in single or multiple JavaScript files which can be reused throughout the Node.js application.

Each module in Node.js has its own context, so it cannot interfere with other modules or pollute global scope. Also, each module can be placed in a separate .js file under a separate folder.

Every thing we declare in a module is private. That is, we cannot access it outside the module.

**Node.js Module Types**:

Node.js includes three types of modules

1. Core Modules
2. Local Modules
3. Third Party Modules

**Core Module**:

Node.js is a light weight framework. The core modules include bare minimum functionalities of Node.js. These core modules are compiled into its binary distribution and load automatically when Node.js process starts. However, we need to import the core module first in order to use it in our application.

For example, http, url, path, fs, util etc.

**Local Module**:

Local modules are modules created locally in your Node.js application. These modules include different functionalities of your application in separate files and folders. You can also package it and distribute it via NPM, so that Node.js community can use it. For example, if you need to connect to MongoDB and fetch data then you can create a module for it, which can be reused in your application.

**Third Party Modules**:

"Third-party" means any module not written by the developer own or included in the standard library. we can use them to add functionality to our code without having to write it ourselves.

**3) Creating a Modules**:

You can create your own modules, and easily include them in your applications.

Module is an object. We can see the details of a module.

console.log(module);

*/\**

*Module {*

*id: '.',*

*exports: {},*

*parent: null,*

*filename: 'C:\\Users\\ruhul\\Desktop\\node-practice\\app.js',*

*loaded: false,*

*children: [],*

*paths:*

*[ 'C:\\Users\\ruhul\\Desktop\\node-practice\\node\_modules',*

*'C:\\Users\\ruhul\\Desktop\\node\_modules',*

*'C:\\Users\\ruhul\\node\_modules',*

*'C:\\Users\\node\_modules',*

*'C:\\node\_modules' ] }*

*\*/*

Suppose we are creating a module name "logger.js" for login messages. We want to use the module in various part of our application.

**loger.js**:

*//suppose we provide service in online*

var url = "http://mylogger.io/log";

function log(message) {

*//sent an http request*

console.log(message);

}

Now the variable "url" and function "log" both scopes only in the logger file. They are private. We cannot access them outside the file. But we want to access log function from app.js file.

Now in the Module object one of the properties is "exports". The property is set to an empty object. Anything that we add to this object will be exported from this module and this is available outside of this module.

*//suppose we provide service in online*

var url = "http://mylogger.io/log";

function log(message) {

*//sent an http request*

console.log(message);

}

*//adding a method to export object and setting the log function to it*

module.exports.log = log;

We also can change the name of the property that we can access from outside.

module.exports.myLog = log;

Now the log method is public and accessible from outside.

**4) Loading a Modules**:

To lode a module we need the "require()" function. This function takes one argument, that’s the name or path of the target module we want to load. This function returns the object that is exported from the target module. We can store the object in a variable.

var logger = require("./logger");

Now we can call the log function from app.js file.

**app.js**:

*//since app.js and logger.js locket in the same folder*

var logger = require("./logger");

logger.log("Ruhul"); *//Ruhul*

The best practice is, when loading a module using the require() function its better to store the result in a constant. The reason is we don’t override the value of the variable.

const logger = require("./logger");

**5) Module Wrapper Function**:

**6) Path Module**:

**7) 017 OS Module**:

**8) Path Module**:

**9) OS Module**:

**10) File Module**:

**11) Event Module**:

**12) Extending Event Emitter**:

**13) HTTP Module**:

02 Node Module System