Project 0 | Lecture Notes

Notes by : Milind Mishra

Module Pattern

Header Notes for Module Pattern

- Syntax like `module.exports = { }` is used to export the module. This is called module pattern.
- Other syntaxes like import {useEffect} from 'react' are also used to export the module which are a part of ES6 Modules syntax.
- In ES6 Modules syntax, we can export the module in two ways:
  - `export default` : This is used to export the module as default.
  - `export` : This is used to export the module as named export.
- In ES6 Modules syntax, we can import the module in two ways:
  - `import` : This is used to import the module as default.
  - `import { }` : This is used to import the module as named export.

Module Pattern in Node JS

- Module Pattern is a mechanism for splitting JS Program into separate managable chunks called as module, that may be imported when needed.
- Sidenote, just like in C++ header files, there are bunch of code written in a file, and we can import that file in another file and use the code written in that file. For example Stack in STL is implemented in a header file, and we can import that header file in our program and use the stack.
- In Node JS, we can use the module pattern to split our code into separate files and import them when needed.
- For illustration purposes, lets say we need to implement Stack using Linked List. We can split our code into two files, one for the implementation of Stack and another for the implementation of Linked List. We can then import the Linked List file in the Stack file and use the Linked List implementation in the Stack file. This is how we can use the module pattern in Node JS.
- In other programming languages, in java as well we prepare class, library and use them in other class.
- In the world of JavaScript we prepare packages. We prepare them in the form of modules. We can use them in other modules.
- Usecase : We can prepare a module for all the searching algorithms, one for sorting algorithms, one for data structures and use them in other modules.
- At interviews its common to be asked to prepare a module for a particular functionality and use it in other modules.

2 Mechanisms for module creation in JavaScript

- In JavaScript, we can create modules in two ways:
  - CJS (Common JS) : This is the default module system in Node JS.
  - ESM (ES6 Modules) : This is the default module system in the browser. This is also supported in Node JS from version 13.2.0.
- ESM stands for ECMAScript Modules.

- Adding `type: "module"` in package.json of a node.js application converts its module system to ESM.
- In ESM, we can export the module in two ways:
  - `export default` : This is used to export the module as default.
  - `export` : This is used to export the module as named export.
- Examples of ESM Syntax :

```
// Exporting a default function
export default function add(x, y) {
  return x + y;
}
// Importing from react library
import {StrictMode} from 'react';
import {createRoot} from 'react-dom/client';
import './styles.css';

import App from './App';
```

- Since React uses ESM we can import the module in two ways:
  - `import React from 'react'` : This is used to import the module as default.
  - `import {useEffect} from 'react'` : This is used to import the module as named export.

DRY Principle (Don't Repeat Yourself)

- DRY Principle stands for Don't Repeat Yourself.
- In programming, we should not repeat the code. We should write the code once and use it multiple times. This is called DRY Principle.
- Technically we write functions and use them multiple times. This is how we follow the DRY Principle.
- If something can be repeated, it should be written as a function and used multiple times.
- We can segregate the code into different files and use them in other files.

Example of Module Pattern

- Lets make a `searching.js` file that contains the implementation of Linear Search and Binary Search.

```
// Linear Search
const linearSearch = (arr, x) => {
  for (let i = 0; i < arr.length; i++) {
    if (arr[i] === x) {
      return i;
    }
  }
  return -1;
};

// Binary Search assuming array is sorted
const binarySearch = (arr, x) => {
  let start = 0;
  let end = arr.length - 1;
  while (start <= end) {
    let mid = Math.floor((start + end) / 2);
    if (arr[mid] === x) {
      return mid;
    } else if (arr[mid] < x) {
      start = mid + 1;
    } else {
      end = mid - 1;
    }
  }
  return -1;
};

// Exporting the module in Common JS Syntax
module.exports = {
  linearSearch: linearSearch,
  binarySearch: binarySearch
};

// shorthand syntax when thw key value pair is same
module.exports = {linearSearch, binarySearch};

// named export in Common JS Syntax
module.exports = {linear : linearSearch, binary : binarySearch};

// In ES6 Modules Syntax
export default {linearSearch, binarySearch};
```

> There exists a global object called `module` in Node JS. This object contains a property called `exports` which is an object. We can add properties to this object and export them. This is how we export the module in Common JS Syntax.

- Inorder to use this module in another file, we can import it using the following syntax:
- Using it in the file lets say `index.js` in the same level.

> The global require function is used to import the module in Node JS. This function takes the path of the module as an argument and returns the module.

```
// Importing the module in Common JS Syntax
const searchingFunctions = require('./searching');

console.log(searchingFunctions); // { linearSearch: [Function: linearSearch], binarySearch: [Function: binarySearch] }

// in order to use the functions we need to access them using the dot notation
console.log(searchingFunctions.linearSearch([1, 2, 3, 4, 5], 3)); // 2
console.log(searchingFunctions.binarySearch([1, 2, 3, 4, 5], 3)); // 2

const {linearSearch, binarySearch} = require('./searching'); // using object destructuring

// In ES6 Modules Syntax
// import {linearSearch, binarySearch} from './searching';

console.log(linearSearch([1, 2, 3, 4, 5], 3)); // 2
console.log(binarySearch([1, 2, 3, 4, 5], 3)); // 2
```

- This is how we are able to get the functions defined in another file and use them in our file.