Week 1

Environment construction

Install Node.js LTS.Create a new NodeProjects folder in a place you are familiar with, and create a new text and save it as hello.js.

```
console.log('Hello World!');
```

Save it and run it in terminal:

```
node hello.js
```

For easy development, it is recommended to install vscode and install the Node.js Extension Pack plug-in.

Modularity of Node

在计算机程序的开发过程中，随着写的代码越来越多，它变得越来越长，在 一个文件中的可维护性也越来越低。为了写出可维护的代码，我们将许多功能归 入独立的文件，使每个文件包含相对较少的代码，这也是许多编程语言组织代码 的方式。在 Node 环境中，一个.js 文件被称为一个模块。

使用模块的好处是什么?

最大的好处是大大提高了代码的可维护性。其次，你不需要从头开始写代码。 当一个模块写完后，它可以在其他地方被引用。当我们写程序时，我们经常引用 其他模块，包括 Node 中内置的模块和第三方的模块。使用模块还可以避免函数和变量名称的冲突。同名的函数和变量可以存在于 不同的模块中，所以我们在编写自己的模块时不必担心与其他模块的名称冲突。

在上一节中，我们写了一个 `hello.js` 文件。hello.js 文件是一个模块。模块的名称就是文件名（减去.js 的后缀），所以 hello.js 文件就是名为 hello 的模块。

1. https://www.liaoxuefeng.com/wiki/1022910821149312/1023027697415616
2. https://www.runoob.com/nodejs/nodejs-module-system.html

Task 1

Encapsulate hello as a function and accept the name variable passed in.export it, import it through require in another js file, and make a pass call, and watch the output.（将 hello 封装为一个函数，并接受传入的名称变量。导出它，在另一个 js 文 件中通过 require 导入它，并进行传入调用，观察输出）

Node.js模块系统

为了让Node.js的文件可以相互调用，Node.js提供了一个简单的模块系统。

模块是Node.js 应用程序的基本组成部分，文件和模块是一一对应的。换言之，一个 Node.js 文件就是一个模块，这个文件可能是JavaScript 代码、JSON 或者编译过的C/C++ 扩展。

> ## *引入模块*
>
> *在 Node.js 中，引入一个模块非常简单，如下我们创建一个* ***main.js*** *文件并引入 hello 模块，代码如下:*
>
> ```
> var hello = require('./hello');
> hello.world();
> ```
>
> *以上实例中，代码 require('./hello') 引入了当前目录下的 hello.js 文件（./ 为当前目录，node.js 默认后缀为 js）。*
>
> *Node.js 提供了 exports 和 require 两个对象，其中 exports 是模块公开的接口，require 用于从外部获取一个模块的接口，即所获取模块的 exports 对象。*
>
> *接下来我们就来创建 hello.js 文件，代码如下:*
>
> ```
> exports.world = function() {
>   console.log('Hello World');
> }
> ```
>
> *在以上示例中，hello.js 通过 exports 对象把 world 作为模块的访问接口，在 main.js 中通过 require('./hello') 加载这个模块，然后就可以直接访 问 hello.js 中 exports 对象的成员函数了。*
>
> *有时候我们只是想把一个对象封装到模块中，格式如下:*

```
module.exports = function() {
  // ...
}
```

例如:

```
//hello.js
function Hello() {
    var name;
    this.setName = function(thyName) {
        name = thyName;
    };
    this.sayHello = function() {
        console.log('Hello ' + name);
    };
};
module.exports = Hello;
```

这样就可以直接获得这个对象了:

```
//main.js
var Hello = require('./hello');
hello = new Hello();
hello.setName('BYVoid');
hello.sayHello();
```

模块接口的唯一变化是使用 module.exports = Hello 代替了exports.world = function() {}。 在外部引用该模块时，其接口对象就是要输出的 Hello 对象本身，而不是原先的 exports。

服务端的模块放在哪里

也许你已经注意到，我们已经在代码中使用了模块了。像这样:

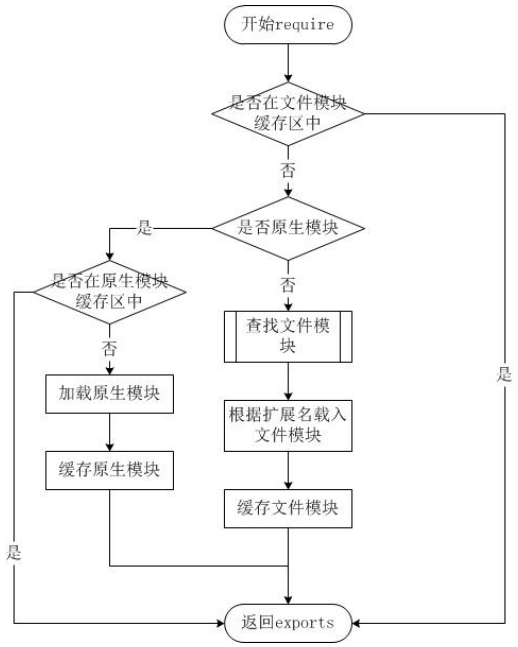
```
var http = require("http");

...

http.createServer(...);
```

Node.js 中自带了一个叫做 **http** *的模块，在我们的代码中请求它并把返回值赋给一个本地变量。*

这把我们的本地变量变成了一个拥有所有 http 模块所提供的公共方法的对象。

Node.js 的 require 方法中的文件查找策略如下：

由于 Node.js 中存在 4 类模块（原生模块和3种文件模块），尽管 require 方法极其简单，但是内部的加载却是十分复杂的，其加载优先级也各自不同。如下图所示：



从文件模块缓存中加载

尽管原生模块与文件模块的优先级不同，但是都会优先从文件模块的缓存中加载已经存在的模块。

从原生模块加载

原生模块的优先级仅次于文件模块缓存的优先级。require 方法在解析文件名之后，优先检查模块是否在原生模块列表中。以http模块为例，尽管在目录下存在一个 `http/http.js/http.node/http.json` 文件，require("http") 都不会从这些文件中加载，而是从原生模块中加载。

原生模块也有一个缓存区，同样也是优先从缓存区加载。如果缓存区没有被加载过，则调用原生模块的加载方式进行加载和执行。

从文件加载

当文件模块缓存中不存在，而且不是原生模块的时候，Node.js 会解析 require 方法传入的参数，并从文件系统中加载实际的文件，加载过程中的包装和编译细节在前一节中已经介绍过，这里我们将详细描述查找文件模块的过程，其中，也有一些细节值得知晓。

require方法接受以下几种参数的传递：

- http、fs、path等，原生模块。
- ./mod或../mod，相对路径的文件模块。
- /pathtomodule/mod，绝对路径的文件模块。
- mod，非原生模块的文件模块。

在路径 Y 下执行 require(X) 语句执行顺序：

```
1. 如果 X 是内置模块
   a. 返回内置模块
   b. 停止执行
2. 如果 X 以 '/' 开头
   a. 设置 Y 为文件根路径
3. 如果 X 以 './' 或 '/' or '../' 开头
   a. LOAD_AS_FILE(Y + X)
   b. LOAD_AS_DIRECTORY(Y + X)
4. LOAD_NODE_MODULES(X, dirname(Y))
5. 抛出异常 "not found"

LOAD_AS_FILE(X)
1. 如果 X 是一个文件, 将 X 作为 JavaScript 文本载入并停止执行。
2. 如果 X.js 是一个文件, 将 X.js 作为 JavaScript 文本载入并停止执行。
3. 如果 X.json 是一个文件, 解析 X.json 为 JavaScript 对象并停止执行。
4. 如果 X.node 是一个文件, 将 X.node 作为二进制插件载入并停止执行。

LOAD_INDEX(X)
1. 如果 X/index.js 是一个文件,  将 X/index.js 作为 JavaScript 文本载入并
停止执行。
2. 如果 X/index.json 是一个文件, 解析 X/index.json 为 JavaScript 对象并
停止执行。
3. 如果 X/index.node 是一个文件,  将 X/index.node 作为二进制插件载入并停止
执行。

LOAD_AS_DIRECTORY(X)
1. 如果 X/package.json 是一个文件,
   a. 解析 X/package.json, 并查找 "main" 字段。
```

```
   b. let M = X + (json main 字段)
   c. LOAD_AS_FILE(M)
   d. LOAD_INDEX(M)
2. LOAD_INDEX(X)

LOAD_NODE_MODULES(X, START)
1. let DIRS=NODE_MODULES_PATHS(START)
2. for each DIR in DIRS:
   a. LOAD_AS_FILE(DIR/X)
   b. LOAD_AS_DIRECTORY(DIR/X)

NODE_MODULES_PATHS(START)
1. let PARTS = path split(START)
2. let I = count of PARTS - 1
3. let DIRS = []
4. while I >= 0,
   a. if PARTS[I] = "node_modules" CONTINUE
   b. DIR = path join(PARTS[0 .. I] + "node_modules")
   c. DIRS = DIRS + DIR
   d. let I = I - 1
5. return DIRS
```

exports 和 module.exports 的使用

*如果要对外暴露属性或方法，就用 **exports** 就行，要暴露对象(类似class，包含了很多属性和方法)，就用 **module.exports**。*

Task2

Wrap multiple simple functions with module.export and see how the different functions are referenced separately in another js file.

Package Manager NPM Getting Started & Exploring JavaScript features

https://www.runoob.com/nodejs/nodejs-npm.html

Here, we recommend that domestic users replace the mirror source of npm to greatly improve the development speed.You can use the nrm tool to quickly change the image source. Open terminal and type:

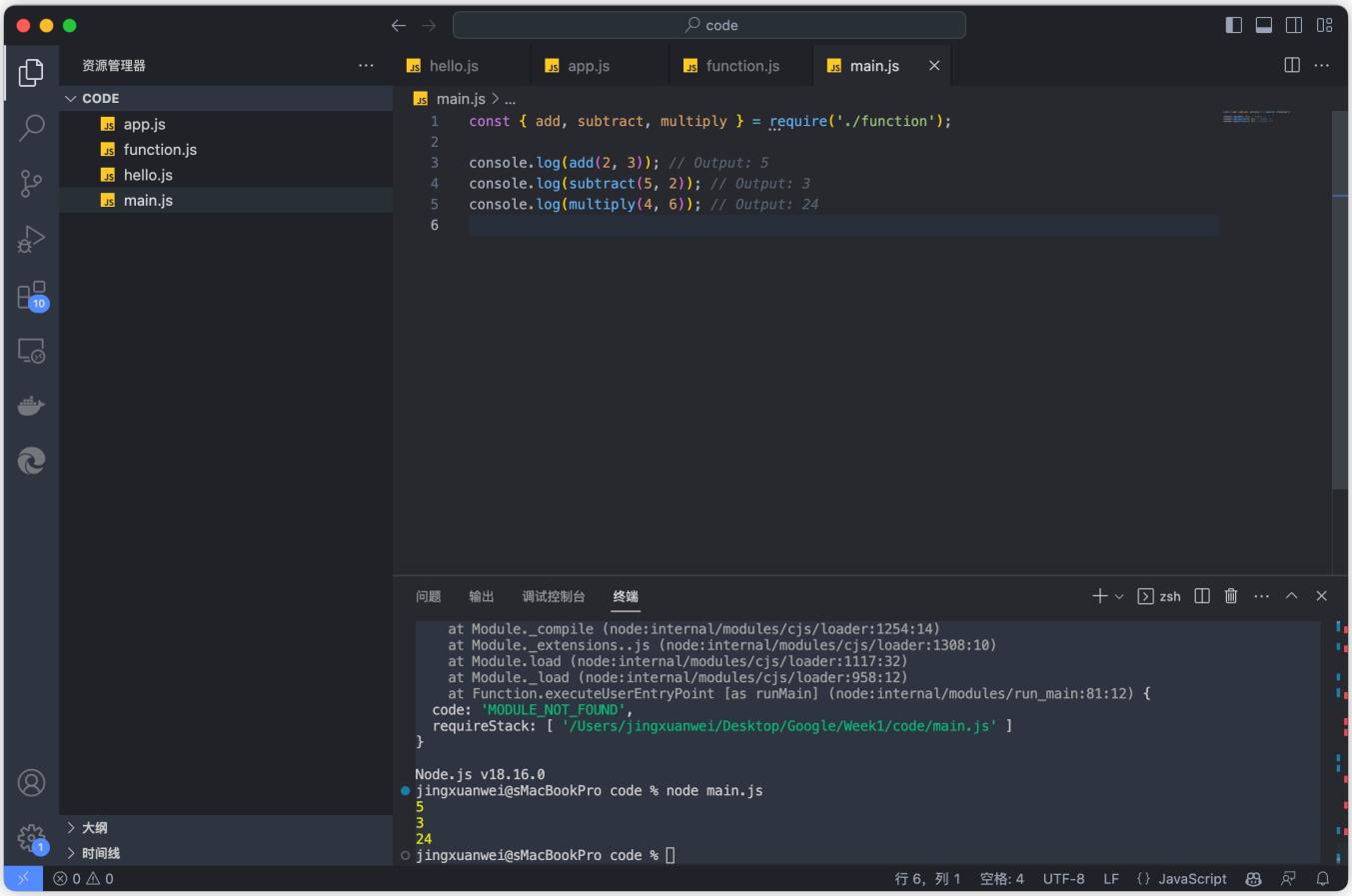
```
npm install -g nrm

nrm use taobao
```

function.js

```
function add(a, b) {
    return a + b;
  }

  function subtract(a, b) {
    return a - b;
  }

  function multiply(a, b) {
    return a * b;
  }

  module.exports = {
    add,
    subtract,
    multiply
  };
```

Main.js

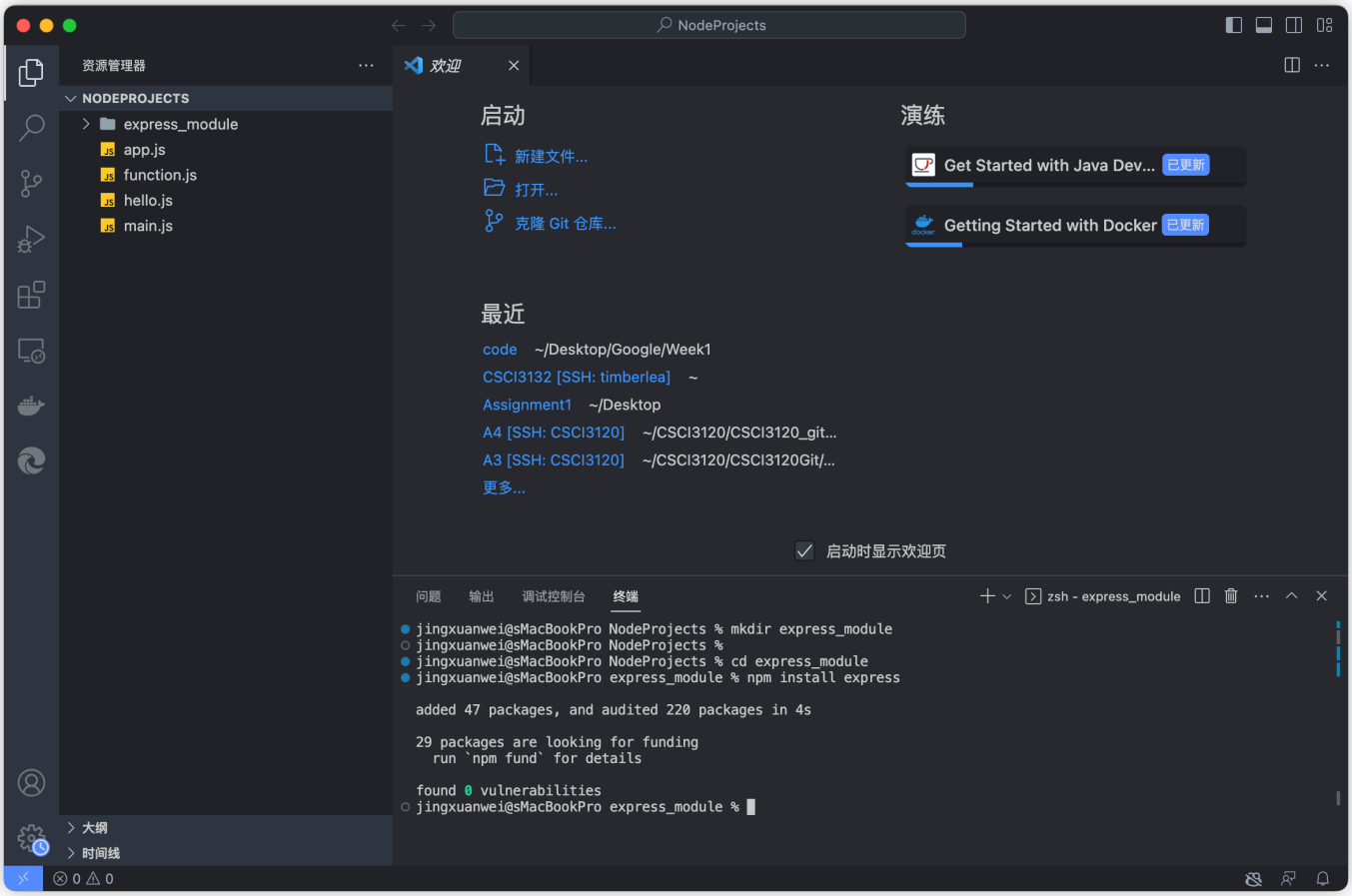
```
const { add, subtract, multiply } = require('./functions');

console.log(add(2, 3)); // Output: 5
console.log(subtract(5, 2)); // Output: 3
console.log(multiply(4, 6)); // Output: 24
```



Task 3

Create a new folder in the `NodeProjects` folder, `express_tutorial`, as your project learning directory, and install the express framework locally.

```
mkdir express_module

cd express_module

npm install express
```



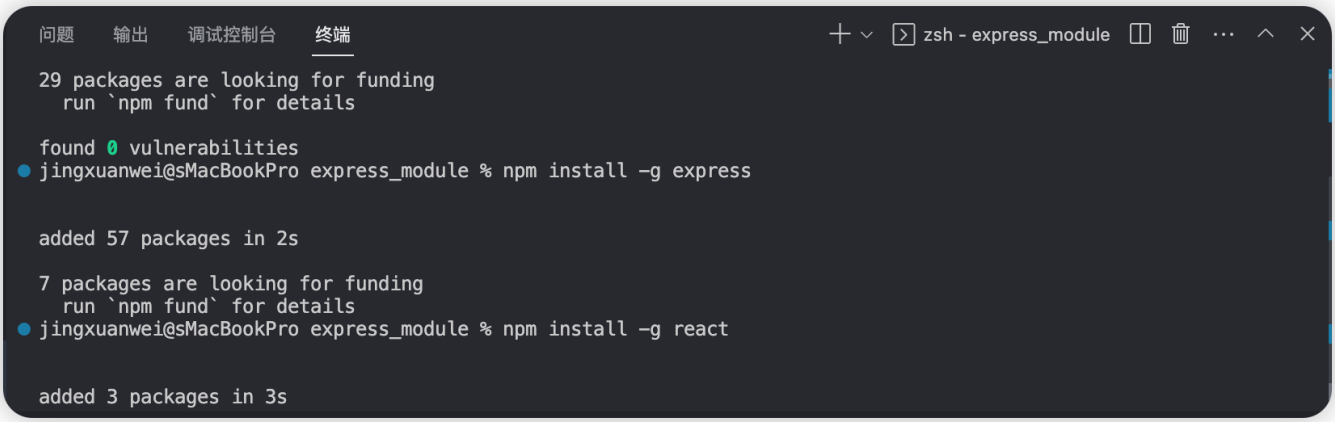
Task 4

Install express and react globally (although you probably won't use react in this project) 在全局范围内安装Express和react（尽管你可能不会在这个项目中使用react）。

```
npm install -g express
```

```
npm install -g react
```

Install the `express` and `react` package



```
jingxuanwei@sMacBookPro NodeProjects % npm ls express
jingxuanwei@ /Users/jingxuanwei
└── express@4.18.2
```

```
jingxuanwei@sMacBookPro NodeProjects % npm ls react
jingxuanwei@ /Users/jingxuanwei
├─┬ @emotion/react@11.9.0
│ └── react@18.1.0
├─┬ @emotion/styled@11.8.1
│ └── react@18.1.0 deduped
├─┬ @mui/icons-material@5.8.2
│ └── react@18.1.0 deduped
├─┬ @mui/material@5.8.2
│ ├─┬ @mui/base@5.0.0-alpha.83
│ │ └── react@18.1.0 deduped
│ ├─┬ @mui/system@5.8.2
│ │ ├─┬ @mui/private-theming@5.8.0
│ │ │ └── react@18.1.0 deduped
│ │ ├─┬ @mui/styled-engine@5.8.0
```