

02Node模块化

为什么要模块化

简介

在Node.js中，以模块为单位划分所有功能，并且提供了一个完整的模块加载机制，这时的我们可以将应用程序划分为各个不同的部分。狭义的说，每一个JavaScript文件都是一个模块；而多个JavaScript文件之间可以相互require，他们共同实现了一个功能，他们整体对外，又称为一个广义上的模块。

导出

Node.js中，一个JavaScript文件中定义的变量、函数，都只在这个文件内部有效。当需要从此JS文件外部引用这些变量、函数时，必须使用exports对象进行暴露。使用者要用require()命令引用这个JS文件。

foo.js文件中的代码：

```
var msg = "你好";  
exports.msg = msg;
```

msg这个变量，是一个js文件内部才有作用域的变量。如果别人想用这个变量，那么就要用exports进行暴露。

使用者：

```
var foo = require("./test/foo.js");  
console.log(foo.msg);
```

使用者用foo来接收exports对象，也就是说，这里的foo变量，就是文件中的exports变量。

一个JavaScript文件，可以向外exports无数个变量、函数。但是require的时候，仅仅需要require这个JS文件一次。使用的它的变量、函数的时候，用点语法即可。所以，无形之中，增加了一个顶层命名空间。

js文件中，可以用exports暴露很多东西，比如函数、变量。

```
var msg = "你好";
var info = "呵呵";
function showInfo(){
    console.log(info);
}
exports.msg = msg;
exports.info = info;
exports.showInfo = showInfo;
```

在使用者中，只需要require一次。

```
var foo = require("./test/foo.js");
```

相当于增加了顶层变量。所有的函数、变量都要从这个顶层变量走：

```
console.log(foo.msg);
console.log(foo.info);
foo.showInfo();
```

注意

Node中，js文件和js文件，就是被一个个exports和require构建成为网状的。不是靠html文件统一在一起的。

可以将一个JavaScript文件中，描述一个类。用 module.export = 构造函数名; 的方式向外暴露一个类。

也就是说，js文件和js文件之间有两种合作的模式：1) 某一个js文件中，提供了函数，供别人使用。只需要暴露函数就行了； exports.msg=msg; 2) 某一个js文件，描述了一个类。 module.exports = People;

引用

如果在require命令中，这么写：

```
var foo = require("foo.js"); //没有写./， 所以不是一个相对路径。是一个特殊的路径
```

那么Node.js将该文件视为node_modules目录下的一个文件

node_modules文件夹并不一定在同级目录里面，在任何直接祖先级目录中，都可以。甚至可以放到NODE_PATH环境变量的文件夹中。这样做的好处稍后你将知道：分享项目的时候，不需要带着modules一起给别人。

我们可以使用文件夹来管理模块，比如

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

那么Node.js将会去寻找node_modules目录下的bar文件夹中的index.js去执行。

require()别的js文件的时候，将执行那个js文件。

注意：

require()中的路径，是从当前这个js文件出发，找到别人。而fs是从命令提示符找到别人。所以，桌面上有一个a.js，test文件夹中有b.js、c.js、1.txt a要引用b：

```
var b = require("../test/b.js");
```

b要引用c：

```
var c = require("../c.js");
```

但是，fs等其他的模块用到路径的时候，都是相对于cmd命令光标所在位置。所以，在b.js中想读1.txt文件，推荐用绝对路径：

```
fs.readFile(__dirname + "/1.txt",function(err,data){
    if(err) { throw err; }
    console.log(data.toString());
});
```

每一个模块文件夹中，推荐都写一个package.json文件，这个文件的名称不能改。node将自动读取里面的配置。有一个main项，就是入口文件：

```
{
  "name": "kaoladebar",
  "version": "1.0.1",
  "main" : "app.js"
}
```

package.json文件，要放到模块文件夹的根目录去。

npm

Node.js 的包管理器 npm，是全球最大的开源库生态系统。

我们刚才学习了，模块就是一些功能的封装，所以一些成熟的、经常使用的功能，都有人封装成为了模块。并且放到了社区中，供人免费下载。这个伟大的社区，叫做npm。也是一个工具名字 node package management <https://www.npmjs.com/>

去社区搜索需求，然后点进去，看api。如果要配置一个模块，那么直接在cmd使用 `npm install` 模块名字 就可以安装。模块名字全球唯一。安装的时候，要注意，命令提示符的所在位置。

1. 我们的依赖包，可能在随时更新，我们永远想保持更新，或者某持某一个版本；
2. 项目越来越大的时候，给别人看的时候，没有必要再次共享我们引用的第三方模块。

我们可以用package.json来管理依赖。在cmd中，使用`npm init`可以初始化一个package.json文件，用回答问题的方式生成一个新的package.json文件。使用 `npm install` 将能安装所有依赖。npm也有文档，这是package.json的介绍：<https://docs.npmjs.com/files/package.json>

常用命令

npm init

这个命令用于创建一个package.json `npm init --yes`或`npm init -y`:从当前目录中提取的信息生成默认的package.json。创建过程中不会提问。

```
Windows PowerShell
PS C:\Users\wbscxxy\Desktop\npm_init> npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

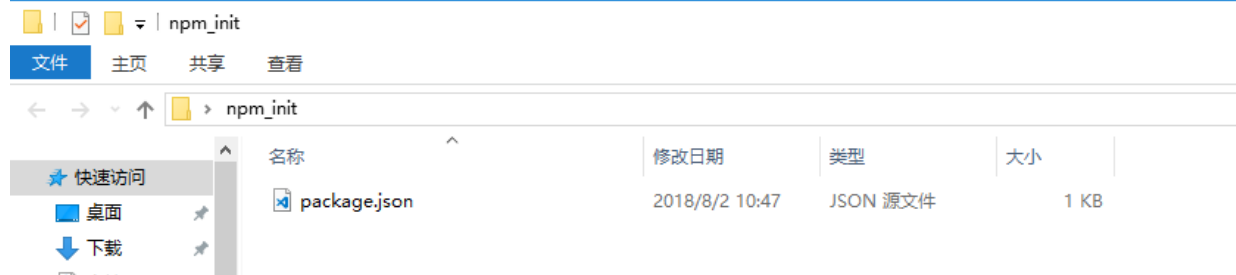
See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (npm_init)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to C:\Users\wbscxxy\Desktop\npm_init\package.json:

{
  "name": "npm_init",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}

Is this ok? (yes)
PS C:\Users\wbscxxy\Desktop\npm_init>
```



npm install

这个命令用于安装一个或多个npm包

本地安装

npm install 包名

1. 将安装包放在 ./node_modules 下（运行 npm 命令时所在的目录），如果没有 node_modules 目录，会在当前执行 npm 命令的目录下生成 node_modules 目录。
2. 可以通过 require() 来引入本地安装的包。

全局安装

npm install 包名 -g

1. 将安装包放在 /usr/local 下或者你 node 的安装目录。
2. 可以直接在命令行里使用。

- --save和- --save-dev

npm install 在安装 npm 包时，有两种命令参数可以把它们的信息写入 package.json 文件，一个是 npm install --save另一个是 npm install --save-dev，他们表面上的区别是--save 会把依赖包名称添加到 package.json 文件 dependencies 键下，--save-dev 则添加到 package.json 文件 devDependencies 键下，譬如： 一个完整的package.json文件

```
{
  "name": "vvv",
  "version": "1.0.0",
  "description": "A Vue.js project",
  "author": "wbscxy <709806757@qq.com>",
  "private": true,
  "scripts": {
    "dev": "webpack-dev-server --inline --progress --config build/webpack.dev.conf.js",
    "start": "npm run dev",
    "build": "node build/build.js"
  },
  "dependencies": {
    "vue": "^2.5.2",
    "vue-router": "^3.0.1"
  },
  "devDependencies": {
    "autoprefixer": "^7.1.2",
    "babel-core": "^6.22.1",
    "babel-helper-vue-jsx-merge-props": "^2.0.3",
    "babel-loader": "^7.1.1",
    "babel-plugin-syntax-jsx": "^6.18.0",
    "babel-plugin-transform-runtime": "^6.22.0",
    "babel-plugin-transform-vue-jsx": "^3.5.0",
    "babel-preset-env": "^1.3.2",
    "babel-preset-stage-2": "^6.22.0",
```

```

    "chalk": "^2.0.1",
    "copy-webpack-plugin": "^4.0.1",
    "css-loader": "^0.28.0",
    "extract-text-webpack-plugin": "^3.0.0",
    "file-loader": "^1.1.4",
    "friendly-errors-webpack-plugin": "^1.6.1",
    "html-webpack-plugin": "^2.30.1",
    "node-notifier": "^5.1.2",
    "optimize-css-assets-webpack-plugin": "^3.2.0",
    "ora": "^1.2.0",
    "portfinder": "^1.0.13",
    "postcss-import": "^11.0.0",
    "postcss-loader": "^2.0.8",
    "postcss-url": "^7.2.1",
    "rimraf": "^2.6.0",
    "semver": "^5.3.0",
    "shelljs": "^0.7.6",
    "uglifyjs-webpack-plugin": "^1.1.1",
    "url-loader": "^0.5.8",
    "vue-loader": "^13.3.0",
    "vue-style-loader": "^3.0.1",
    "vue-template-compiler": "^2.5.2",
    "webpack": "^3.6.0",
    "webpack-bundle-analyzer": "^2.9.0",
    "webpack-dev-server": "^2.9.1",
    "webpack-merge": "^4.1.0"
  },
  "engines": {
    "node": ">= 6.0.0",
    "npm": ">= 3.0.0"
  },
  "browserslist": [
    "> 1%",
    "last 2 versions",
    "not ie <= 8"
  ]
}

```

不过这只是它们的表面区别。它们真正的区别是，devDependencies 下列出的模块，是我们开发时用的，比如 我们安装 js 的压缩包gulp-uglify 时我们采用的是“npm install --save-dev gulp-uglify”命令安装，因为我们在发布后用不到它，而只是在我们开发才用到它。dependencies 下的模块，则是我们发布后还需要依赖的模块，譬如像jQuery库或者Vue框架类似的，我们在开发完后肯定还要依赖它们，否则就运行不了。

另外需要补充的是：正常使用npm install时，会下载dependencies和devDependencies中的模块，当使用npm install --production或者注明NODE_ENV变量值为production时，只会下载dependencies中的模块。

npm publish

这个命令用于上传当前包到npm服务器 当我们上传成功之后 所有人就可以用npm install下载下来了

npm unpublish

这个命令用于在npm服务器移除当前包

npm adduser

在当前电脑保存npm用户信息，第一次上传之前需先登录当前用户