現代JavaScript

使用模塊管理器(npm、yarn)來自動下載第三方模塊、使用模塊打包工具(Webpack)來創建單個腳本文件、使用轉譯器(TypeScript)來使用未來的JavaScript功能，以及使用任務執行器(npm scripts)來自動化構建過程的不同部分

構建一個由多個JavaScript文件組成的Web應用程序，可以通過腳本標籤將 JavaScript文件添加到html中，但每個文件都需要一個單獨的http請求

<html>

<script src="/src/foo.js"></script>

<script src="/src/bar.js"></script>

</html>

CommonJS或ES6模塊提供了一種方式來指定依賴文件、使用接口(模組化)

// CommonJS

const foo = require('./foo');

module.exports = bar;

// ES Modules

import foo from './foo';

export default bar;

前端不支持模組化，需要用module bundler模塊打包工具幫助我們將多個JavaScript模塊組合成1個JavaScript文件

CommonJS

為JavaScript在網頁瀏覽器之外建立模組約定(將代碼從一個文件導入另一個文件的方法)的一個專案，允許JavaScript像大多數編程語言一樣跨文件導入和導出代CommonJS require/exports模組化語法

導出

exports.fs = fs; //單個特性導出，可導出多個

module.exports = fs; //整個模塊導出，每个模塊只包含一個

引入

const fs = require('fs'); //引入整個模塊

const \_ = require('<package-name>'); //引入npm模塊

Node.js

能夠在伺服器端(後端)運行JavaScript開放原始碼的跨平台執行環境，可以直接在 JavaScript文件中加載其他.js文件代碼，而不是使用HTML腳本標籤加載所有內容，所有建構工具都是基於Node.js平台運行，因此模塊化默認採用CommonJS

npm Node Package Manager軟件包管理器

用JavaScript編寫的Node.js預設的軟體套件管理系統，可以管理本地專案的所需模組並自動維護依賴情況，也可以管理全域安裝的JavaScript工具

package.json檔案

包含模塊依賴項等資訊

name應用程序或模塊名稱

version當前版本

description應用程序或模塊描述

main應用程序入口點

dependencies產品依賴安裝的npm模塊列表

devDependencies開發依賴安裝的npm模塊列表

scripts 定義node腳本

private如果為true，可以防止應用程序或模塊被發佈到npm

engines 設置此應用程序或模塊在哪個版本的Node.js上運行

browserslist 用於告知要支持哪些瀏覽器和瀏覽器版本

在package.json中，可以使用semver設置版本更新規則

主版號MAJOR:當你做了不相容的API修改

次版號MINOR:當你做了向下相容的功能性新增

修訂號PATCH:當你做了向下相容的問題修正

MAJOR.MINOR.PATCH

安裝此版本

^ MAJOR.MINOR.PATCH

保持MAJOR，MINOR、PATCH安裝最新版本

~ MAJOR.MINOR.PATCH

保持MAJOR、MINOR，PATCH安裝最新版本

> MAJOR.MINOR.PATCH

高於指定版本

>= MAJOR.MINOR.PATCH

高於、等於指定版本

<= MAJOR.MINOR.PATCH

低於、等於指定版本

< MAJOR.MINOR.PATCH

低於指定版本

major.minor.patch - MAJOR.MINOR.PATCH

接受major.minor.patch至MAJOR.MINOR.PATCH的版本

major.minor.patch||MAJOR.MINOR.PATCH

接受major.minor.patch或MAJOR.MINOR.PATCH的版本

node\_modules文件夾

包含下載模塊的檔案

本地模塊安裝在運行npm install <package-name>的目錄中，並且放置在此目錄下的node\_modules文件夾中

全局模塊安裝在系統中的單獨位置，無論在何處運行npm install -g <package-name>

package-lock.json檔案

跟踪被安裝的每個軟件包的確切版本，以便產品可以被完美複製

npm CLI

設置npm

npm init

在專案目錄中執行，創建package.json

–y

全部採用默認設置

package name應用程序或模塊名稱，默認為專案目錄名稱

version當前版本，默認為1.0.0

description應用程序或模塊描述

entry point應用程序入口點(如果要執行專案，應該執行的檔案)，默認為index.js

test command專案測試指令

git repository專案原始碼的版本控管位置

keywoards專案關鍵字

author專案作者

license專案版權，默認為ISC

查找資料

npm –v

顯示npm版本

npm list

查看所有已安裝(包括模塊的依賴項模塊)npm模塊版本

-g

查看全局安裝npm模塊版本

--depth=n

查看已安裝npm模塊版本至第n層依賴項模塊

npm list <package-name>

查看指定npm模塊版本(包括模塊的依賴項模塊)

npm view <package-name> version

查看模塊在npm倉庫的最新版本

npm view <package-name > versions

查看模塊在npm倉庫的所有版本

npm outdated

列出過時模塊列表

npm root –g

顯示全局位置

安裝、卸載、更新模塊

npm install或i

下載、安裝package-lock.json中的模塊

--production

只安裝dependencies

npm install <package-name>

下載、安裝指定模塊，並將模塊添加到dependencies中

--save-prod|-P

將模塊添加到dependencies中，為默認參數

--save-dev|-D

將模塊添加到devDependencies中

-g

全局安裝(不會將模塊安裝到node\_modules，而是安裝到全局位置)

npm install <package>@<version>

下載、安裝指定版本的指定模塊

npm uninstall或un <package-name>

卸載指定模塊

-S或--save

將模塊從dependencies中刪除

-D或--save-dev

將模塊從devDependencies中刪除

-g

全局卸載

npm update

更新所有模塊

npm update <package-name>

更新指定模塊

npm scripts腳本 任務執行器

npm允許在package.json裡，使用scripts對象定義npm腳本

npm <script>

執行指定內置npm腳本，例如start、test等npm run

查看所有npm腳本列表

npm run <script>

執行指定自定義npm腳本

npm run <script1> && npm <script2>

串聯運行多個腳本

npm run <script1> & npm <script2>

並聯運行多個腳本

npx

在npm v5.2.0之後的內建CLI，可以臨時性的安裝非全局性必要的套件(使用命令時，會安裝臨時包，等到命令完成後就刪除)，省下許多安裝、使用流程、磁碟空間，避免長期汙染

node\_modules/.bi目錄

存儲依賴項模塊的二進製文件(執行檔)

NPM使所有依賴項的二進製文件都可以在腳本中使用，可以直接通過腳本名稱來調用當前目錄的node\_modules / .bin子目錄中的所有腳本，而無需添加路徑

"test": "mocha test"

等同

"test": "./node\_modules/.bin/mocha test"

如果退出代碼不是0，則npm假定腳本執行失敗，將中止進程

Shell(提供使用者介面的軟體，例如bash)通配符

"lint": "jshint \*.js"

\*表示任何文件名

"lint": "jshint \*\*/\*.js"

\*\*表示任何子目錄

npm腳本有兩個鉤子(被觸發的程序)，pre(之前)和post(之後)

"scripts": {

"prebuild": "echo I run before the build script",

"build": "cross-env NODE\_ENV=production webpack",

"postbuild": "echo I run after the build script

}

npm\_lifecycle\_event環境變量被設置為正在執行的腳本名稱，因此可以為npm scripts同一腳本文件中的不同命令編寫代碼

const TARGET = process.env.npm\_lifecycle\_event;

if (TARGET === 'test') {

console.log('Running the test task!');

}

if (TARGET === 'pretest') {

console.log('Running the pretest task!');

}

npm腳本可以獲取package.json帶有npm\_package\_前綴的字段

{

"name": "foo",

"scripts": {

"install": "foo.js"

}

}

npm\_package\_name等於foo，npm\_package\_scripts\_ install等於foo.js