**05-Working with package.json**

管理本地安装的npm packages的最好的方式是通过创建**package.json文件。**

一个**package.json**文件：

●包含项目依赖的packages

●在文件中可以指定项目中依赖的package的版本号

●使我们的构建更具有可重塑性，并且更加容易与其他开发者共享。

## **Requirements**

**一个package.json必须包括：**

●name：其值必须小写，不准有空格，可以包含“-“和“\_”

●version:其值的格式为\*.\*.\*

例如：

|  |
| --- |
| {  "name": "my-awesome-package",  "version": "1.0.0"  } |

## **Creating a package.json**

有两种方式可以创建一个**package.json**文件

1. 通过命令行，使用一种类似于问卷的方式创建

|  |
| --- |
| > npm init |

这个命令将在你所在的目录下创建一个**package.json**文件。执行命令后，会通过问卷方式，让你输入一些信息，这些信息便会出现在生成的package.json文件中

1. 创建一个默认的package.json文件（不需要通过文件方式输入信息。直接形成一个还有默认信息的文件）

|  |
| --- |
| > npm init --yes 或者 >npm init -y |

这种方式将会使用当前所在目录的一些信息创建一个package.json文件，例如

|  |
| --- |
| Wrote to /home/ag\_dubs/my\_package/package.json:  {  "name": "my\_package",  "description": "",  "version": "1.0.0",  "main": "index.js",  "scripts": {  "test": "echo \"Error: no test specified\" && exit 1"  },  "repository": {  "type": "git",  "url": "https://github.com/ashleygwilliams/my\_package.git"  },  "keywords": [],  "author": "",  "license": "ISC",  "bugs": {  "url": "https://github.com/ashleygwilliams/my\_package/issues"  },  "homepage": "https://github.com/ashleygwilliams/my\_package"  } |

●name：所在目录的文件夹名称

●version：默认总是1.0.0

●description：默认内容来自于readme或者是一个空字符串

●main：默认为index.js

●scripts：默认为一段test script

●keywords：默认为空

●author：默认为空

●license：默认为ISC

●bugs：

●homepage：

你也可以在执行init命令之前，设置一些配置项，例如：

|  |
| --- |
| > npm set init.author.email "wombat@npmjs.com"  > npm set init.author.name "ag\_dubs"  > npm set init.license "MIT" |

注意：

如果在package.json文件中没有description 这个部分，npm将是使用readme.md的第一行代替。

description 帮助人们在npm页面中更加容易的查询到你的package。

### How to Customize the package.json questionnaire

如果你期望创建多个package.json文件，而且每个文件中都包含一些各自不同的其他内容。那么我们可以通过在当前用户目录下通过创建一个**.npm-init.js（~/.npm-init.js）文件。**

**例如文件的内容如下：**

|  |
| --- |
| module.exports = {  customField: 'Custom Field',  otherCustomField: 'This field is really cool'  } |

那么当我们执行npm init命令的时候，会发现，package.json文件中会多出大括号中的内容。

更多的相关高级的自定义操作，可以参考 [**init-package-json**](https://github.com/npm/init-package-json)

## Specifying Dependencies(指定项目需要的依赖)

要想指定项目需要哪些第三方package的依赖，需要在package.json文件中添加下面两个内容。

●dependencies：生产环境中所需要的第三方依赖放在这个里面

●devDependencies：开发环境下或者测试需要的第三方依赖放在这个里面

### Manually editing your package.json（手动编辑package.json）

通过手动编辑，我们在package.json文件中，添加一个属性为dependencies的对象，对象中为项目需要依赖的第三方package。如果依赖的第三方package只是在开发和测试情况下需要，那么就添加一个devDependencies属性的对象，将需要的第三方package放到里面便可。

例如，一个项目，再生产环境下需要依赖一个名为my\_dep的第三方package，在开发环境下需要一个my\_test\_framework的第三方package，那么写法如下：

|  |
| --- |
| {  "name": "my\_package",  "version": "1.0.0",  "dependencies": {  "my\_dep": "^1.0.0"  },  "devDependencies" : {  "my\_test\_framework": "^3.1.0"  }  } |

### The --save and --save-dev install flags

如果想添加一个生产环境下依赖的第三方package时，使用

|  |
| --- |
| npm install <package\_name> --save |

如果想添加一个开发环境或者测试环境下依赖的第三方package时，使用

|  |
| --- |
| npm install <package\_name> --save-dev |

## Managing dependency versions

## 当运行**npm install** 命令时，如果所在目录下有package.json文件，npm会先在package.json文件中的dependencies中查找对应的第三方package，然后下载其指定的版本。如果没有则下载最新的版本。

# 06-How to Update Local Packages

如何定期的更新项目中依赖的第三方package？按照这样做：

1. 在package.json所在目录运行 **npm update** 命令
2. 然后运行 **npm outdated**命令

# 07-How to Uninstall Local Packages

如果要从node\_modules文件夹下删除一个apckage，使用如下命令：

npm uninstall <package>:

|  |
| --- |
| npm uninstall lodash |

如果要从package.json文件的dependencies中删除它，那么需要使用save flag；

|  |
| --- |
| npm uninstall - -save lodash |

如果要从package.json文件的devDependencies中删除它，那么需要使用- -save-dev flag：

|  |
| --- |
| npm uninstall - -save-dev lodash |

#### Test:

如何确定npm install命令正确的被执行，你可以到node\_modules文件夹下查看，只要文件夹下不再包含你删除的那个package，那么就算是成功删除了。

除此之外，你也可以通过运行下面的命令来删除package：

●ls node\_modules(linux 命令)

●dir node\_modules(window下)

# 08-How to Install Global Packages

安装npm packages分为局部安装和全局安装，选择哪一种取决于想如何使用这些package。

●如果你想把一个package当作一个命令行工具来使用，那么进行全局安装。这种方式，使用时就无需考虑当前是在那个文件夹下。

●如果你想依赖只在自己的项目中使用，那么使用局部安装。

如果全局安装的话，使用命令npm install –g <package>,如：

|  |
| --- |
| npm install –g jshint |

# 09-How to Update Global Packages

Requires version2.6.1 or greater。

要更新全局的packages，使用命令：

**npm update –g <package>**

例如，更新jshint package，需要输入

|  |
| --- |
| npm update –g jshint |

如果想要知道哪个package需要更新，使用命令：

**npm outdated –g - -depth=0**

如果要更新所有的package,使用命令：

**npm update –g**

如果使用的npm版本是2.6或者更早的版本的话，使用[this script](https://gist.github.com/othiym23/4ac31155da23962afd0e)来更新全局的packages

无论如何，最好你还是考虑升级npm到最新版本。如下：

**npm install npm@lastest –g**

# 10-How to uninstall global packages

删除一个全局的package，使用命令：

**Npm uninstall –g <package>**

例如：

|  |
| --- |
| npm uninstall –g jshint |

# 11-How to Create Node.js Modules

Node.js模块是可以发布到npm中的包的其中一种。创建一个新的模块首先要创建一个package.json文件

使用npm init命令创建package.json，它将提示你要输入一些内容。其中有两个字段的内容是必须要输入的。那就是name和version。另外你也需要设置main字段的值，当然你不设置会默认使用index.js。关系详细描述参考05.

一旦创建完package.json后，你需要在创建一个文件，这个文件会被作为一个入口文件，当你的模块被其他项目所依赖时，需要从这个入口文件开始执行。这个文件的默认名字为index.js

在这个入口文件中添加一个函数作为exports对象的一个属性。这样当其他的文件require这个入口文件后，这个函数就可以在其他代码中使用了。如下：

|  |
| --- |
| exports.printMsg = function() {  console.log("This is a message from the demo package");  } |

Test：

1. 发布你的package到npm上
2. 在你的项目外，新建一个文件夹
3. 切换到这个新建的文件中（cd）
4. 运行npm install<package>
5. 创建一个test.js，然后使用require引入，然后调用printMsg这个方法
6. 运行node test.js,控制台成功打印出了那段话

# 12-How to Publish & Update a Package

# 13-How to use Semantic Versioning

何为Semantic Versioning可参考：<https://semver.org/lang/zh-TW/>

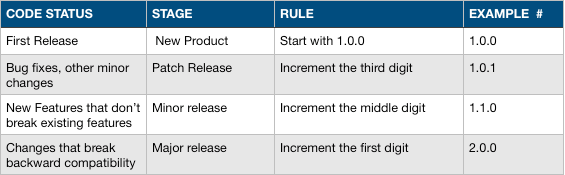
这里先放一个关于版本格式的一个小说明：

|  |
| --- |
| 版本格式：主版號.次版號.修訂號，版號遞增規則如下：   1. 主版號：當你做了不相容的 API 修改， 2. 次版號：當你做了向下相容的功能性新增， 3. 修訂號：當你做了向下相容的問題修正。   先行版號及版本編譯資訊可以加到「主版號.次版號.修訂號」的後面，作為延伸。 |

## Semver for Publishers

如果一个项目打算分享给他人使用，那么一般都会版版本号设置为1.0.0

随后，将会根据项目内容不同的改变，来修改版本号，如下：



## Semver for Consumers

作为使用者，你可以在package.json文件中指定你的项目需要哪个版本。

# 14-How to Work with Scoped Packages

Reuqireds npm version 2 or greater

Scopes用于把相关的一些包组织在一起，然后创建一个命名空间。

如果一个package的名字以@开头，那么他就是一个scoped package。这个scope就是位于@和/之间的那个内容。如

|  |
| --- |
| @scope/project-name |

每个npm用户都已有一个他们自己独自的scope

|  |
| --- |
| @user-name/project-name |

npm Orgs 也有scopes

@orgname/project-name

你可以在 [CLI documentation](https://docs.npmjs.com/misc/scope#publishing-public-scoped-packages-to-the-public-npm-registry)找到更多相关的讲

## How to Initialize a Scoped Package

创建一个scoped package，只需要简单的使用下面的方式

|  |
| --- |
|  |

如果你使用npm init命令，你可以在命令后添加参数即可，如下：

|  |
| --- |
| Npm init - -scope=username |

如果你想在所有情况下，都是用同一个scope，那么可以将这个选项设置到.npmrc文件中，命令如下：

|  |
| --- |
| npm config set scope username |

## Publishing a Scoped Package

默认情况下，scoped package是私有的，要发布一个私有的模块，你需要去付费成为 [private modules](https://www.npmjs.com/private-modules)用户

Public scoped modules是免费的，不需要付费。去发布一个public scoped modules，设置一下访问选项即可。

|  |
| --- |
| npm publish - -access = public |

## Using a Scoped Package

使用一个scoped package，只需要将需要使用的package通过name包含到package.json。如下：

|  |
| --- |
| {  "dependencies": {  "@username/project-name": "^1.0.0"  }  } |

如果想通过命令行的模式来添加使用，则如下：

|  |
| --- |
| npm install @username/project-name --save |

在调用声明时，如下写法：

|  |
| --- |
| var projectName = require("@username/project-name") |

相关更多的信息，参考 [npmjs.com/private-modules](https://www.npmjs.com/private-modules).