npm scripts 使用指南

作者： 阮一峰

日期： 2016年10月11日

Node 开发离不开 npm，而脚本功能是 npm 最强大、最常用的功能之一。

本文介绍如何使用 npm 脚本（npm scripts）。

一、什么是 npm 脚本？

npm 允许在 package.json 文件里面，使用 scripts 字段定义脚本命令。

```
{
  // ...
  "scripts": {
    "build": "node build.js"
  }
}
```

上面代码是 package.json 文件的一个片段，里面的 scripts 字段是一个对象。它的每一个属性，对应一段脚本。比如， build 命令对应的脚本是 node build.js 。

命令行下使用 npm run 命令，就可以执行这段脚本。

```
$ npm run build
# 等同于执行
$ node build.js
```

这些定义在 package.json 里面的脚本，就称为 npm 脚本。它的优点很多。

- 项目的相关脚本，可以集中在一个地方。
- 不同项目的脚本命令，只要功能相同，就可以有同样的对外接口。用户不需要知道怎么测试你的项目，只要运行npm run test即可。
- 可以利用 npm 提供的很多辅助功能。

查看当前项目的所有 npm 脚本命令，可以使用不带任何参数的 npm run 命令。

```
$ npm run
```

二、原理

npm 脚本的原理非常简单。每当执行 npm run ，就会自动新建一个 Shell，在这个 Shell 里面执行指定的脚本命令。因此，只要是 Shell（一般是 Bash）可以运行的命令，就可以写在 npm 脚本里面。

比较特别的是， npm run 新建的这个 Shell，会将当前目录的 node_modules/.bin 子目录加入 PATH 变量，执行结束后，再将 PATH 变量恢复原样。

这意味着，当前目录的 node_modules/.bin 子目录里面的所有脚本，都可以直接用脚本名调用，而不必加上路径。比如，当前项目的依赖里面有 Mocha，只要直接写 mocha test 就可以了。

```
"test": "mocha test"
```

而不用写成下面这样。

```
"test": "./node_modules/.bin/mocha test"
```

由于 npm 脚本的唯一要求就是可以在 Shell 执行，因此它不一定是 Node 脚本，任何可执行文件都可以写在里面。

npm 脚本的退出码，也遵守 Shell 脚本规则。如果退出码不是 0 ，npm 就认为这个脚本执行失败。

三、通配符

由于 npm 脚本就是 Shell 脚本，因为可以使用 Shell 通配符。

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

上面代码中， * 表示任意文件名， ** 表示任意一层子目录。

如果要将通配符传入原始命令，防止被 Shell 转义，要将星号转义。

```
"test": "tap test/\*.js"
```

四、传参

向 npm 脚本传入参数，要使用 -- 标明。

```
"lint": "jshint **.js"
```

向上面的 npm run lint 命令传入参数，必须写成下面这样。

```
$ npm run lint --  --reporter checkstyle > checkstyle.xml
```

也可以在 package.json 里面再封装一个命令。

```
"lint": "jshint **.js",
"lint:checkstyle": "npm run lint -- --reporter checkstyle > checkstyle.xml"
```

五、执行顺序

如果 npm 脚本里面需要执行多个任务，那么需要明确它们的执行顺序。

如果是并行执行（即同时的平行执行），可以使用 & 符号。

```
$ npm run script1.js & npm run script2.js
```

如果是继发执行（即只有前一个任务成功，才执行下一个任务），可以使用 && 符号。

```
$ npm run script1.js && npm run script2.js
```

这两个符号是 Bash 的功能。此外，还可以使用 node 的任务管理模块：script-runner、npm-run-all、redrun。

六、默认值

一般来说，npm 脚本由用户提供。但是，npm 对两个脚本提供了默认值。也就是说，这两个脚本不用定义，就可以直接使用。

```
"start": "node server.js",
"install": "node-gyp rebuild"
```

上面代码中， npm run start 的默认值是 node server.js ，前提是项目根目录下有 server.js 这个脚本； npm run install 的默认值是 node-gyp rebuild ，前提是项目根目录下有 binding.gyp 文件。

七、钩子

npm 脚本有 pre 和 post 两个钩子。举例来说， build 脚本命令的钩子就是 prebuild 和 postbuild 。

```
"prebuild": "echo I run before the build script",
"build": "cross-env NODE_ENV=production webpack",
"postbuild": "echo I run after the build script"
```

用户执行 npm run build 的时候，会自动按照下面的顺序执行。

```
npm run prebuild && npm run build && npm run postbuild
```

因此，可以在这两个钩子里面，完成一些准备工作和清理工作。下面是一个例子。

```
"clean": "rimraf ./dist && mkdir dist",
```

```
"prebuild": "npm run clean",
"build": "cross-env NODE_ENV=production webpack"
```

npm 默认提供下面这些钩子。

- prepublish，postpublish
- preinstall，postinstall
- preuninstall，postuninstall
- preversion，postversion
- pretest，posttest
- prestop，poststop
- prestart，poststart
- prerestart，postrestart

自定义的脚本命令也可以加上 pre 和 post 钩子。比如， myscript 这个脚本命令，也有 premyscript 和 postmyscript 钩子。不过，双重的 pre 和 post 无效，比如 prepretest 和 postposttest 是无效的。

npm 提供一个 npm_lifecycle_event 变量，返回当前正在运行的脚本名称，比如 pretest 、 test 、 posttest 等等。所以，可以利用这个变量，在同一个脚本文件里面，为不同的 npm scripts 命令编写代码。请看下面的例子。

```
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!`);
}

if (TARGET === 'posttest') {
  console.log(`Running the posttest task!`);
}
```

注意， prepublish 这个钩子不仅会在 npm publish 命令之前运行，还会在 npm install （不带任何参数）命令之前运行。这种行为很容易让用户感到困惑，所以 npm 4 引入了一个新的钩子 prepare ，行为等同于 prepublish ，而从 npm 5 开始， prepublish 将只在 npm publish 命令之前运行。

八、简写形式

四个常用的 npm 脚本有简写形式。

- npm start是npm run start
- npm stop是npm run stop的简写
- npm test是npm run test的简写
- npm restart是npm run stop && npm run restart && npm run start的简写

npm start 、 npm stop 和 npm restart 都比较好理解，而 npm restart 是一个复合命令，实际上会执行三个脚本命令： stop 、 restart 、 start 。具体的执行顺序如下。

1. prerestart
2. prestop
3. stop
4. poststop
5. restart
6. prestart
7. start
8. poststart
9. postrestart

九、变量

npm 脚本有一个非常强大的功能，就是可以使用 npm 的内部变量。

首先，通过 npm_package_ 前缀，npm 脚本可以拿到 package.json 里面的字段。比如，下面是一个 package.json 。

```
{
  "name": "foo",
  "version": "1.2.5",
  "scripts": {
    "view": "node view.js"
  }
}
```

那么，变量 npm_package_name 返回 foo ，变量 npm_package_version 返回 1.2.5 。

```
// view.js
console.log(process.env.npm_package_name); // foo
console.log(process.env.npm_package_version); // 1.2.5
```

上面代码中，我们通过环境变量 process.env 对象，拿到 package.json 的字段值。如果是 Bash 脚本，可以用 $npm_package_name 和 $npm_package_version 取到这两个值。

npm_package_ 前缀也支持嵌套的 package.json 字段。

```
  "repository": {
    "type": "git",
    "url": "xxx"
  },
  scripts: {
    "view": "echo $npm_package_repository_type"
  }
```

上面代码中， repository 字段的 type 属性，可以通过 npm_package_repository_type 取到。

下面是另外一个例子。

```
"scripts": {
  "install": "foo.js"
}
```

上面代码中， npm_package_scripts_install 变量的值等于 foo.js 。

然后，npm 脚本还可以通过 npm_config_ 前缀，拿到 npm 的配置变量，即 npm config get xxx 命令返回的值。比如，当前模块的发行标签，可以通过 npm_config_tag 取到。

```
"view": "echo $npm_config_tag",
```

注意， package.json 里面的 config 对象，可以被环境变量覆盖。

```
{
  "name" : "foo",
  "config" : { "port" : "8080" },
  "scripts" : { "start" : "node server.js" }
}
```

上面代码中， npm_package_config_port 变量返回的是 8080 。这个值可以用下面的方法覆盖。

```
$ npm config set foo:port 80
```

最后， env 命令可以列出所有环境变量。

```
"env": "env"
```

十、常用脚本示例

```
// 删除目录
"clean": "rimraf dist/*",

// 本地搭建一个 HTTP 服务
"serve": "http-server -p 9090 dist/",

// 打开浏览器
"open:dev": "opener http://localhost:9090",

// 实时刷新
 "livereload": "live-reload --port 9091 dist/",

// 构建 HTML 文件
"build:html": "jade index.jade > dist/index.html",

// 只要 CSS 文件有变动，就重新执行构建
"watch:css": "watch 'npm run build:css' assets/styles/",

// 只要 HTML 文件有变动，就重新执行构建
"watch:html": "watch 'npm run build:html' assets/html",

// 部署到 Amazon S3
"deploy:prod": "s3-cli sync ./dist/ s3://example-com/prod-site/",

// 构建 favicon
"build:favicon": "node scripts/favicon.js",
```

十一、参考链接

- How to Use npm as a Build Tool, by Keith Cirkel
- Awesome npm scripts, by Ryan Zimmerman

（完）

文档信息

- 版权声明：自由转载-非商用-非衍生-保持署名（创意共享3.0许可证）
- 发表日期： 2016年10月11日

相关文章

- **2021.01.20:** 剪贴板操作 Clipboard API 教程

  > 一、简介 浏览器允许 JavaScript 脚本读写剪贴板，自动复制或粘贴内容。

- **2020.12.28:** Fetch API 教程

  > fetch()是 XMLHttpRequest 的升级版，用于在 JavaScript 脚本里面发出 HTTP 请求。

- **2020.09.15:** 轻松学会 React 钩子：以 useEffect() 为例

  > 五年多前，我写过 React 系列教程。不用说，内容已经有些过时了。

- **2020.08.20:** Node.js 如何处理 ES6 模块

  > 学习 JavaScript 语言，你会发现它有两种格式的模块。

Weibo | Twitter | GitHub

Email: yifeng.ruan@gmail.com