package.json文件

来自《JavaScript 标准参考教程（alpha）》 (/)，by 阮一峰

目录

1. 概述

每个项目的根目录下面，一般都有一个 package.json 文件，定义了这个项目所需要的各种模块，以及项目的配置信息（比如名称、版本、许可证等元数据）。 npm install 命令根据这个配置文件，自动下载所需的模块，也就是配置项目所需的运行和开发环境。

下面是一个最简单的package.json文件，只定义两项元数据：项目名称和项目版本。

```
{
  "name" : "xxx",
  "version" : "0.0.0",
}
```

上面代码说明， package.json 文件内部就是一个JSON对象，该对象的每一个成员就是当前项目的一项设置。比如 name 就是项目名称， version 是版本（遵守“大版本.次要版本.小版本”的格式）。

下面是一个更完整的package.json文件。

```
{
        "name": "Hello World",
        "version": "0.0.1",
        "author": "张三",
        "description": "第一个node.js程序",
        "keywords":["node.js","javascript"],
        "repository": {
                "type": "git",
                "url": "https://path/to/url"
        },
        "license":"MIT",
        "engines": {"node": "0.10.x"},
        "bugs":{"url":"http://path/to/bug","email":"bug@example
        "contributors":[{"name":"李四","email":"lisi@example.co
        "scripts": {
                "start": "node index.js"
        },
        "dependencies": {
                "express": "latest",
                "mongoose": "~3.8.3",
                "handlebars-runtime": "~1.0.12",
                "express3-handlebars": "~0.5.0",
                "MD5": "~1.2.0"
        },
        "devDependencies": {
                "bower": "~1.2.8",
                "grunt": "~0.4.1",
                "grunt-contrib-concat": "~0.3.0",
                "grunt-contrib-jshint": "~0.7.2",
                "grunt-contrib-uglify": "~0.2.7",
                "grunt-contrib-clean": "~0.5.0",
                "browserify": "2.36.1",
                "grunt-browserify": "~1.3.0",
        }
}
```

下面详细解释package.json文件的各个字段。

2. scripts字段

scripts 指定了运行脚本命令的npm命令行缩写，比如start指定了运行 npm run start 时，所要执行的命令。

下面的设置指定了 npm run preinstall 、 npm run postinstall 、 npm run start 、 npm run test 时，所要执行的命令。

```
"scripts": {
    "preinstall": "echo here it comes!",
    "postinstall": "echo there it goes!",
    "start": "node index.js",
    "test": "tap test/*.js"
}
```

3. dependencies字段，devDependencies字段

dependencies 字段指定了项目运行所依赖的模块， devDependencies 指定项目开发所需要的模块。

它们都指向一个对象。该对象的各个成员，分别由模块名和对应的版本要求组成，表示依赖的模块及其版本范围。

```
{
  "devDependencies": {
    "browserify": "~13.0.0",
    "karma-browserify": "~5.0.1"
  }
}
```

对应的版本可以加上各种限定，主要有以下几种：

- **指定版本**：比如 1.2.2 ，遵循“大版本.次要版本.小版本”的格式规定，安装时只安装指定版本。
- **波浪号（tilde）+指定版本**：比如 ~1.2.2 ，表示安装1.2.x的最新版本（不低于1.2.2），但是不安装1.3.x，也就是说安装时不改变大版本号和次要版本

号。

- **插入号（caret）+指定版本**：比如^1.2.2，表示安装1.x.x的最新版本（不低于1.2.2），但是不安装2.x.x，也就是说安装时不改变大版本号。需要注意的是，如果大版本号为0，则插入号的行为与波浪号相同，这是因为此时处于开发阶段，即使是次要版本号变动，也可能带来程序的不兼容。
- **latest**：安装最新版本。

package.json文件可以手工编写，也可以使用 `npm init` 命令自动生成。

```
$ npm init
```

这个命令采用互动方式，要求用户回答一些问题，然后在当前目录生成一个基本的package.json文件。所有问题之中，只有项目名称（name）和项目版本（version）是必填的，其他都是选填的。

有了package.json文件，直接使用npm install命令，就会在当前目录中安装所需要的模块。

```
$ npm install
```

如果一个模块不在 `package.json` 文件之中，可以单独安装这个模块，并使用相应的参数，将其写入 `package.json` 文件之中。

```
$ npm install express --save
$ npm install express --save-dev
```

上面代码表示单独安装express模块， `--save` 参数表示将该模块写入 `dependencies` 属性， `--save-dev` 表示将该模块写入 `devDependencies` 属性。

4. peerDependencies

有时，你的项目和所依赖的模块，都会同时依赖另一个模块，但是所依赖的版本不一样。比如，你的项目依赖A模块和B模块的1.0版，而A模块本身又依赖B模块的2.0版。

大多数情况下，这不构成问题，B模块的两个版本可以并存，同时运行。但是，有一种情况，会出现问题，就是这种依赖关系将暴露给用户。

最典型的场景就是插件，比如A模块是B模块的插件。用户安装的B模块是1.0版本，但是A插件只能和2.0版本的B模块一起使用。这时，用户要是将1.0版本的B的实例传给A，就会出现问题。因此，需要一种机制，在模板安装的时候提醒用户，如果A和B一起安装，那么B必须是2.0模块。

peerDependencies 字段，就是用来供插件指定其所需要的主工具的版本。

```
{
  "name": "chai-as-promised",
  "peerDependencies": {
    "chai": "1.x"
  }
}
```

上面代码指定，安装 chai-as-promised 模块时，主程序 chai 必须一起安装，而且 chai 的版本必须是 1.x 。如果你的项目指定的依赖是 chai 的2.0版本，就会报错。

注意，从npm 3.0版开始， peerDependencies 不再会默认安装了。

5. bin字段

bin项用来指定各个内部命令对应的可执行文件的位置。

```
"bin": {
  "someTool": "./bin/someTool.js"
}
```

上面代码指定，someTool 命令对应的可执行文件为 bin 子目录下的 someTool.js。Npm会寻找这个文件，在 node_modules/.bin/ 目录下建立符号链接。在上面的例子中，someTool.js会建立符号链接 npm_modules/.bin/someTool 。由于 node_modules/.bin/ 目录会在运行时加入系统的PATH变量，因此在运行npm时，就可以不带路径，直接通过命令来调用这些脚本。

因此，像下面这样的写法可以采用简写。