林寻 Lv3
2019年04月15日 阅读 1244

关注

package.json和package-lock.json

前言：前几天接到一个小任务，需要给一个老项目增加一个小功能。这本来是个简单的活儿，但是由于是老项目，我本地没有，于是从仓库clone下来，一顿操作之后，`npm start`却跑不起来，报的错是依赖的版本有问题，于是开始了漫长的踩坑之路。

- 首先是删除`node_modules`重新`npm install`，这似乎跟重启一样好用，但还是报错，卒
- 然后固定依赖的版本，仍然报错，原来是第三方依赖中引用的依赖版本有问题，卒
- 通过问老大，查资料等方式了解到`package-lock.json`文件可以控制版本，但是仓库里并没有这个文件，好在同事有，满心欢喜再次`npm install`，成功！！！

> 其实还有个小插曲，我之前一直用的cnpm，所以并不会根据`package-lock.json`安装依赖，查阅资料后决定弃用cnpm，配置镜像后再次启用npm。建议大家还是使用官方的npm，配置镜像之后速度也还可以接受，第三方的可能会很多坑。

虽然看起来好像没多少技术含量，但是一开始由于思考的方向有点偏差，导致浪费了很多时间。仔细衡量了下还是记录下来。以下是一些笔记：

版本控制规则

> a.b.c：a表示主版本号，b表示次版本号，c表示补丁更新。当你只是简单的修复了bug， 当没有添加任何新功能，或者修改旧功能时， 就更新补丁号；当添加了新的功能， 但没有破坏原有的功能，就更新次版本号；当做了重大修改，导致新版本不兼容旧代码时，就更新主版本号。

package.json版本控制

1. 指定版本：比如 "antd": "3.10.7"或"antd": "=3.10.7"，表示安装3.10.7的版本
2. ~：比如 "antd": "~3.10.7"，表示安装3.10.x的最新版本（不低于3.10.7），但是不安装3.11.x，也就是说安装时不改变大版本号和次要版本号

3. ^：比如 "antd": "^3.10.7"，表示安装3.10.7及以上的版本，但是不安装4.0.0，也就是说安装时不改变大版本号。

package-lock.json

> `package-lock.json` 的作用就是用来保证我们的应用程序依赖之间的关系是一致的，兼容的；适合多人协作开发时保证每个人的依赖版本是一致的

当项目中不存在 `package-lock.json` 文件时，使用 `npm install` 时会自动生成这个文件。当 `package-lock.json` 存在时，则会安装文件中指定版本的依赖，并且安装速度会比不存在此文件时快很多。因为 `package-lock.json` 中已经存在依赖的版本，下载地址和整个 `node_modules` 的文件结构等信息。

> 使用yarn同样也会自动生成 `package-lock.json` 文件，但是cnpm不会自动生成，并且也不会读取 `package-lock.json` 文件，只根据 `package.json` 下载依赖。

更新依赖

> 在npm 5.x之前，我们可以直接更改 `package.json` 中的版本号，再 `npm install` 就可以直接更新了，但是5.x之后由于是根据 `package-lock.json` 安装依赖，所以我们只能使用 `npm install xxx@x.x.x` 去更新依赖，这样 `package-lock.json` 也会同步更新。

> **关注下面的标签，发现更多相似文章**

NPM

林寻 Lv3 前端

获得点赞 693 · 获得阅读 33,467

关注