NPM的flatten, dedupe和shrinkwrap


柳正来 (/u/6e2e8c12ef0b) + 关注
2017.11.30 14:56* 字数 1594 阅读 37 评论 0 喜欢 0
(/u/6e2e8c12ef0b)

今天碰到了一些NPM相关的问题, 花了些时间搞清楚, 记录一下.

关于NPM包依赖的扁平化 (flatten)

接手的项目里面使用到了包flatten-packages (https://link.jianshu.com?t=https://www.npmjs.com/package/flatten-packages), 这个包可以将npm包的嵌套依赖压扁. 对这方面之前没了解过, 于是做了一下调研.

为什么需要扁平化

在早期, npm包是以下形式保存的:

```
.
|--app
    |--node_modules
       |--sub_module
          |--node_modules
             |--sub_sub_module
                |-- ... and so on
```

即你的项目依赖包A, 那么node_modules里面就有一个A的文件夹, 在这个文件夹中, 又有一个node_modules, 所有A的依赖都放这个node_modules中, 以此类推. npm的包依赖就会成为一个非常深的文件夹结构, 深到有时你在windows系统直接删除该文件夹会失败, 因为文件目录超过了windows限制的256个字符. 此时你可以打开Git Bash什么的 `rm -rf .`

这会导致一个问题: 不同的包(A和B)可能依赖同一个包(C), 这样 `npm install` 的时候, 安装A的时候会下载一次C, 安装B的时候又会下载一次C. 无形中多了很多没必要的下载, 导致 `npm install` 的速度变慢很多.

所以这也就有了扁平化包依赖的需求 -- 让重复的依赖尽量合并, 以加速包管理的速度.

扁平化的方法

flatten-packages

flatten-packages (https://link.jianshu.com?t=https://www.npmjs.com/package/flatten-packages)这个包可以解决这个问题. 跑一下 `flatten-packages` 即可.

但是, 新版本的npm会进行自动扁平化处理, 所以flatten-packages已经没有用了, 这个包也已经停止更新了. 而且在解决包冲突方面也不如 `npm dedupe` . 相关Issue (https://link.jianshu.com?t=https://github.com/arifsetiawan/flatten/issues/22)

npm3的自动扁平化

npm第3版会自动进行包的扁平化. 详见npm v3 (https://link.jianshu.com?t=https://github.com/npm/npm/releases/tag/v3.0.0), 搜索flat.

npm3会尽可能地扁平化包依赖, 绝大多数情况下你的依赖包都会直接存在于node_modules里. 唯一的例外是, 某两个包依赖互相冲突的时候.

为什么包依赖会有冲突?

简单的例子就是同一个包C, A依赖于C 1.0.0, B依赖于C 2.0.0. 这样如果你的app同时依赖于A和B, 那就没法直接在node_modules里面放一个版本C来同时满足A和B的依赖了. 这里的版本号是SemVer.

SemVer

SemVer (https://link.jianshu.com?t=https://semver.org/) (Semantic Versioning, 语义化版本).

SemVer最基本的结构是major.minor.patch, 如1.2.3.
其中,

- major为主版本号, 当有非向后兼容(即breaking change)的时候, 更新major.
- minor为次版本号, 当有向后兼容的时候, 更新minor.
- patch为补丁号, 当有向后兼容的bug fix时, 更新patch.

npm使用SemVer来标注包的版本, 这些配置写入到了package.json中.

除了指定固定的版本号, package.json中还可以指定版本号范围.

- `1.2.x` (`x` 也可以用 `*` 代替), 相当于 `>=1.2.0 <1.3.0`
- `1.x.x` , 相当于 `>=1.0.0 <2.0.0`
- 波浪线(Tilde), `~1.2.3` 相当于 `>=1.2.3 <1.3.0` , 即minor不能增加.
- 破折号(Caret), `^1.2.3` 相当于 `>=1.2.3 <2.0.0` , 即major不能增加.

包去重 (dedupe)

想要去重只需运行 `npm dedupe`

既然npm3自动进行了扁平化, 为什么还需要去重? 这个npm用了一篇文章npm3 Duplication and Deduplication (https://link.jianshu.com?t=https://docs.npmjs.com/how-npm-works/npm3-dupe)进行讲解, 我就不翻译了. 简单来说就是:

1. 由于历史原因, 某两个包A和B依赖于同一个包C的不同版本1.0.0和2.0.0, 这个冲突导致C的依赖无法被合并.
2. 后来A和B的某次更新使得他们依赖于同一个版本的C2.0.0, 但是仍然由于你的app直接依赖C1.5.0, 导致A和B依赖的C2.0.0依然无法被合并, 只能各自存放在A和B的目录下.
   (问题: 如果这时候运行 `npm dedup` 是什么效果?)
3. 后来你把直接依赖C1.5.0更新成了C2.0.0, 这会导致你有三个重复的C2.0.0, 需要运行 `npm dedupe` 去重.
   (问题: 为什么不能更新版本的时候自动运行dedupe呢?)

包依赖锁定(shrinkwrap/lock)

package.json中的依赖包的版本号可能是版本范围, 或者依赖包的依赖可能使用了版本范围, 这会导致一个问题: 你今天 `npm install` 安装了包A1.0.0, 一段时间后你的同事运行 `npm install` , 可能就会安装A1.2.0, 导致你们的运行环境不完全一样. 想要将包依赖的版本号完全锁定住, 就需要shrinkwrap/lock.

package-lock.json

npm5引入了package-lock.json, 即在你运行 `npm install` 的时候自动会生成package-lock.json文件, 这是一个描述依赖树的文件, 它的好处是锁定了所有依赖的版本甚至下载地址, 而且结构清晰人能读懂(相对于错综复杂的node_modules目录结构).

示例package-lock.json

```
{
  "name": "A",
  "version": "0.1.0",
  ...metadata fields...
  "dependencies": {
    "B": {
      "version": "0.0.1",
      "resolved": "https://registry.npmjs.org/B/-/B-0.0.1.tgz",
      "integrity": "sha512-DeAdb33F+"
      "dependencies": {
        "C": {
          "version": "git://github.com/org/C.git#5c380ae319fc4efe9e7f2d9c78b0faa588
        }
      }
    }
  }
}
```

如果有package-lock.json, 安装过程会变成:

1. 按照package-lock.json重建依赖包的树形结构. 如果有"resolved"字段则使用该字段指向的文件下载文件, 否则使用"version".
2. 若最后还有缺失的依赖包, 则使用普通的package.json安装方法.

注意: `npm install`, `npm rm`, `npm update` 都会自动更新package-lock.json. 如果你不想更新, 可以使用以下命令行参数:
`--no-save`: 不更新package.json也不更新package-lock.json
`--no-shrinkwrap`: 更新package.json, 不更新package-lock.json和npm-shrinkwrap.json.

npm非常建议 (https://link.jianshu.com?t=https://docs.npmjs.com/files/package-locks)将package-lock.json存入版本控制, 以确保组内所有成员, 持续集成(Continuous Integration, CI)和部署环境是用完全一致的依赖包.[package-locks]

npm-shrinkwrap.json

你可以使用 `npm shrinkwrap` (https://link.jianshu.com?t=https://docs.npmjs.com/cli/shrinkwrap)指令, 在package-lock.json的基础上生成一个名为npm-shrinkwrap.json的文件.

package-lock.json和npm-shrinkwrap.json的内容**完全一样**, 唯一区别是: 当发布包的时候, package-lock.json不会被包含在内, 但是npm-shrinkwrap.json会被一同发布. 如果两者同时存在, package-lock.json会被完全忽略.

参考

1. Semver: A Primer (https://link.jianshu.com?t=http://nodesource.com/blog/semver-a-primer/)
2. 为什么我不使用 shrinkwrap（lock） (https://link.jianshu.com?t=https://zhuanlan.zhihu.com/p/22934066)
3. package-locks (https://link.jianshu.com?t=https://docs.npmjs.com/files/package-locks)