# Nodejs

## package.json 依赖项版本号前面的 ~ ^ 有什么用？

* ~ 会匹配最近的小版本依赖包，比如~1.2.3会匹配所有1.2.x版本，但是不包括1.3.0
* ^ 会匹配最新的大版本依赖包，比如^1.2.3会匹配所有1.x.x的包，包括1.3.0，但是不包括2.0.0
* 不写前缀，只写版本号就不会有更新问题

## 项目中使用 package-lock.json 锁版本问题

* [项目中使用package-lock.json锁版本问题](https://www.cnblogs.com/yy95/p/10441727.html)
* [npm install 生成的package-lock.json是什么文件？有什么用？](https://www.zhihu.com/question/62331583)
* [npm依赖版本变动引发的惨案](https://segmentfault.com/a/1190000024520174)

## peerDependencies 有什么用

假设一个项目有如下依赖项：

{  
 "dependencies": {  
 "a": "1.0.0",  
 "b": "1.0.0",  
 "c": "1.0.0",  
 }  
}

并且 b 和 c 也各自有依赖项 a。那么下载依赖后的项目目录是这样的：

.  
├── project  
│ └── node\_modules  
│ ├── a  
│ ├── b  
│ │ └── nodule\_modules  
│ │ └── a  
│ └── c  
│ │ └── nodule\_modules  
│ │ └── a

这样会有一个问题，依赖 a 会被重复安装 3 次。这时可以使用 peerDependencies，它可以避免相同的依赖被重复安装。

现在只需要在 b 和 c 的 package.json 文件加上以下代码：

{  
 "peerDependencies": {  
 "a": "1.0.0"  
 }  
}

这样在安装时就可以避免重复安装依赖了。现在下载依赖后的目录为：

.  
├── helloWorld  
│ └── node\_modules  
│ ├── a  
│ ├── b  
│ └── c

* 如果用户显式依赖了核心库，则可以忽略各插件的 peerDependency 声明；
* 如果用户没有显式依赖核心库，则按照插件 peerDependencies 中声明的版本将库安装到项目根目录中；

注意，如果主项目的 a 依赖项和 b、c 安装的 a 依赖项版本不同，有可能会报错。

例如 b 和 c 是一个插件的时候，它们依赖于某个核心依赖才能工作。当 b c 的核心依赖版本和项目的核心依赖版本不同时，就会报错。这时使用 peerDependencies 就可以很好的解决这个问题。

参考资料： \* [一文搞懂peerDependencies](https://segmentfault.com/a/1190000022435060) \* [浅谈npm 的依赖与版本](https://github.com/SamHwang1990/blog/issues/7)

## 垃圾回收机制

[内存回收](https://github.com/woai3c/Notes-and-Labs/blob/master/notes/nodejs-srjc/docs/5.md)