为什么我不使用 shrinkwrap（lock）

死马
招 Node.js / 前端（busi.hyy@antfin.com）
关注他

最近 Facebook 发布的 yarn 又引发了前端（node）界的海啸，yarn 除了速度比 npm 更快、支持离线安装等特性外，对 npm 体系最大的一个冲击是 yarn 默认提供了 lock 功能。也就是说在通过 yarn 安装了一次依赖之后，如果不执行 yarn upgrade，删除后再重新安装模块的版本不会发生变化。其实在 npm 中也有提供一个类似的功能：shrinkwrap，但是他需要用户执行 npm shrinkwrap 来手动锁定版本。

在 yarn 出来之后，许多开发者都在讨论锁定版本的功能对于线上代码的稳定性提升有很大帮助，而在维护 cnpm / npminstall 的时候，由于我们对 install 的过程做了优化而不再继续支持 shrinkwrap 也带来了非常大的咨询量。因此也想借这个机会来聊聊为什么我不使用 shrinkwrap（lock）。

在进入这个问题的探讨之前，我们先预设一个前提条件：***一定要选择靠谱的开源模块***，否则无论是否采用 shrinkwrap 都无法拯救你的项目。一般来说，靠谱的模块一般都有下面几个特征：

1. 严格遵循 semver 语义化版本的原则来进行版本发布。对不遵循 semver 发布的模块请敬而远之。
2. npm 上有较大的下载量，当遇到模块问题时，波及范围越广，修复的速度越快。
3. github 上问题反馈迅速，或者是一些知名开发者维护。
4. patch 位变更的发布不多（说明 bug fix 不多）。

npm 和其他语言（java，ruby）的模块管理器不太一样，是通过 nested dependency 形式进行依赖处理的，每个模块对自身的依赖负责，导致一个项目虽然只直接依赖了十来个模块，但最终却间

于是 npm 引入了 semver 语义化版本的机制来帮助开发者管理依赖，开发者可以在 package.json 中通过 ^1.1.0 或者 ~1.0.0 的方式来引入模块，如果开发者信任他们依赖的模块，开发者可以通过 ^ 来锁定一个模块的大版本，这样在每次重新安装依赖或者打包的时候，都能够享受到这个包所有的新增功能和 bug 修复。而这个模块如果遵循 semver 原则，也不用担心它会引入一些不兼容变更导致项目出现一些未知异常。最终开发者需要关心的其实只有直接依赖的这些模块是否足够靠谱。

我们在公司内部（阿里巴巴 / 蚂蚁金服）维护着一个非常复杂的 node web 框架，它的依赖模块多达 932 个，分散在不同的插件和基础库中，被不同的同学维护着。我们通过最宽松的依赖方式（^）来管理依赖，任何一个插件或者底层模块提供了新功能或者 bug fix，不需要我们主动去推动业务方升级，业务方只需要信任我们的框架，指定框架的大版本，在下一次重新安装依赖或者打包的时候就会自动更新。

作为框架维护者，可能我们都不知道到底有多少线上、线下的业务在依赖我们的代码，如果所有的业务开发用 shrinkwrap 锁定住版本号，有一些隐藏的 bug 可能很难被修复。而大部分的业务开发同学其实对 npm 整体理解并没那么深入，想要靠所有的业务同学频繁更新 shrinkwrap 并不现实。而且在保障业务优先的前提下，更可能的是在业务开发过程中，能不升级的情况下就不去动任何依赖的版本，这样是看起来『最稳定』的。殊不知这样反而留下了更大的隐患。

举一个例子，前段时间一个项目中使用到了 mongodb，由于内部的一次断网演习触发了 node-mongodb-native 的底层依赖库 mongodb-core 的一个bug 导致内存泄露 OOM 了。如果是正常的通过 semver 依赖，在下一次安装模块的时候这个 bug 就默默的被修复了，但是如果你通过 shrinkwrap 锁定了依赖，很可能就是这个 bug 的下一个受害者。

通过 semver 依赖机制，在一个***良性的环境中***，可以快速的正向传递新功能和 bug fix，而 shrinkwrap 就是这个传递路径上的一道墙，虽然它也许可以挡住一些新 bug 的引入，但其实是得不偿失的。我们真正要做的是提供一个更好的环境（依赖可靠的模块）。

不使用 shrinkwrap 会有一些问题，而这些问题也许都是可以解决的：

> 如果不使用 shrinkwrap，可能出现线上运行的代码包中打包的依赖和开发、测试时不一致。

如果担忧出现这个情况，其实更多的是应该思考一下整个运维发布体系如何进行优化了。最佳方案应该是从提测开始时进行一次打包，然后对这个代码包进行测试、回归直到上线，保证发布上线的代码（包括所有的依赖）是经过完善测试的。否则就算通过 shrinkwrap 锁定了版本，也并不意味着你发布上线的代码和开发、测试阶段的代码一模一样。也许是因为依赖引入了一些没有在 npm registry 上的模块（git、remote url），也许是因为构建的环境不一致导致，都可能引起线上运行代码和测试时不一致。

> 测试毕竟无法完全覆盖所有场景，如果发到线上还是发生了一些由于依赖的底层模块升级导致的故障怎么办?

致命，然后加上及时的回滚止血，在很大程度上降低了此类问题出现的影响范围。

> 就算及时止血了，如果问题模块的作者一直不修复这个 bug，我们也没有办法控制底层模块不依赖到这个问题版本。

166

如果真的遇到这类问题且会造成巨大影响的时候，我相信你们的开发团队已经比较壮大了，是到了建立一个团队内的私有源的时候了。私有源 完全是团队可控的， 可以通过删除有问题的模块版本，或者是通过重新设置 latest tag 的方式让有问题的模块不再被安装到。通过 cnpm，可以很低成本的在内部搭建一个完全受控的 npm 私有源。

> 我们并没有私有源，无法在内部进行控制，去有问题的模块提交 issue 也半天没有人处理怎么办?

npm 上几十万个模块，其实真正热门且一直都有被良好维护的模块并不多。如果我们要在一些重要业务上使用 node，就务必要精心挑选社区热门的模块，最好是对自己使用的模块有充分的了解。通常这些热门模块发生此类问题的几率相对较低，而且一旦出现问题也会很快被修复。要相信社区优秀模块的自我修复能力是非常强的。以我们遇到的为数不多经验来看，基本都可以在半小时内得到修复。如果真的发生了这样的问题，就要回过头来想想当时为什么选择了一个这么不靠谱的模块了。

如果对上面的观点仍然有疑问，欢迎来阿里/蚂蚁金服体验没有 shrinkwrap 的 js / node 世界是怎么运行的。:)

Q&A：

> 「如果是正常的通过 semver 依赖，在下一次安装模块的时候这个 bug 就默默的被修复了，但是如果你通过 shrinkwrap 锁定了依赖，很可能就是这个 bug 的下一个受害者。」但如果这个是线上问题的话，仍然需要应用负责人空发一次上线，所以仍然是需要通知到每一个依赖方的，这对于简化流程并没有帮助，并不能证明宽松的 semver 优于 lock。

如果是我们主动修复的重大 bug，肯定是需要发通知重新发布的，但是如果没有 shrinkwrap 的情况下还是有很大的不同的：

1. 如果用了shrinkwrap，可能业务的版本和现在的版本相差很大（甚至是跨大版本），一次升级版本变更非常大，升级痛苦。
2. 如果是一些社区修复的 bug，我们无法立刻感知到，如果不用 shrinkwrap ，在下次发布就更新了，用了 shrinkwrap 永远也不知道。

> 『一定要选择靠谱的开源模块』这个前提在实际业务开发中就很难成立

如果不成立，那就别去考虑 shrinkwrap 的问题，先解决『如何选择靠谱的开源模块』这个问题。事情有轻重缓急，shrinkwrap 不致命，但是引入了不靠谱的开源模块绝对是致命的。

166

> 个人认为lock之后风险更低。这样经过完整测试之后至少能保证当前包没有问题，而如果线上的依赖包有问题，不管你是否lock都需要再重新打包更新发布一次。而lock的好处就是，在之后进行小更新发布的时候依然能确保依赖包的稳定性。如果将这种稳定性寄托于理想化的semver和靠谱的开源模块，这本身就是不靠谱的。一个大型项目可能会直接依赖几十个npm包，这些npm包又会依赖其他npm包，最后加起来可能上万，根本无法保证这些包完全靠谱。说不定有些人一怒之下就把自己的包全删了（逃

首先，一怒之下删包的情况，通过 shrinkwrap 并不能解决。 其次，你不需要保证所有的这些包完全靠谱，你只要保证你直接依赖的包靠谱就可以了，一个靠谱的模块它自身就会谨慎选择他的依赖，用的人多也同样会将其中的很多问题给提前暴露出来。 最后，当线上的依赖包真的有问题，第一操作一定是回滚，后面再修复。在修复的时候，反而是之前 lock 了版本会更麻烦，因为一次 lock 更新之后的版本差别比不 lock 的时候大太多了，你要回归的范围会更大。

编辑于 2018-03-28

Node.js　npm　yarnpkg

文章被以下专栏收录

Node.js
在 eggjs 团队的日常协作中，遵循「基于 GitLab 的硬盘式异步协作模式」。 先通过 i...

关注专栏

推荐阅读