



11月12日 | 第74届前端早早聊

前端工程化 工程演进 业务进阶

SSR

依赖管理

容器化

工程实践

架构演进



2023 全年程 ▶

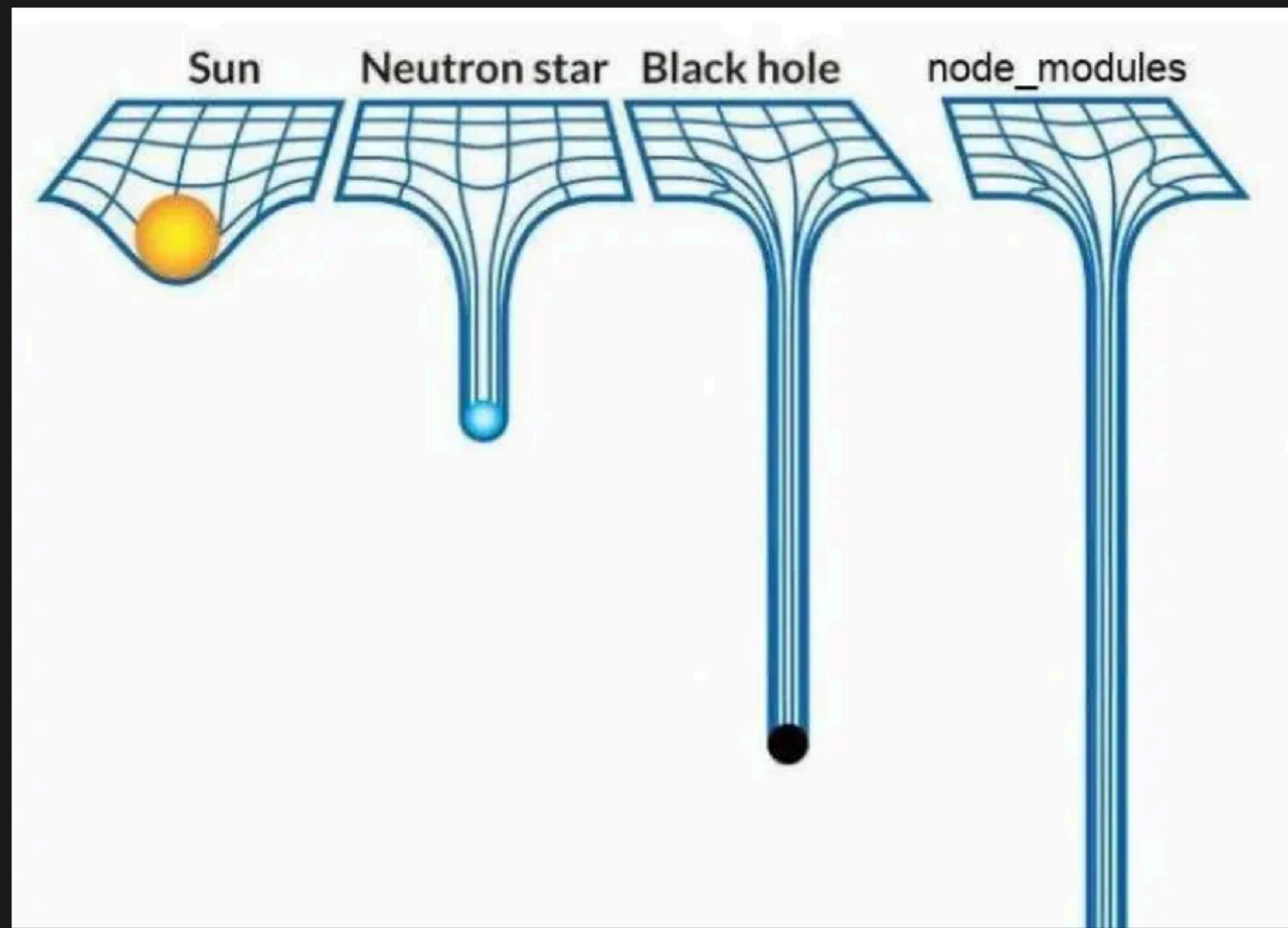
2/4	2/25	3/18	4/1	4/8	4/15	5/7	5/20	6/3	6/10	6/17	7/15	8/12	9/2	9/23	10/28	10/29	11/4	11/12	11/19	11/25	12/2	12/9	12/16	12/23
健康指南	前端搞规划	前端搞插件	Node.js	杭州 GPT 沙龙	低代码无代码	前端搞管理	前端就业新起点	上海 GPT 沙龙	武汉 GPT 沙龙	前端性能优化	Night Of AI	AI Chatbot	前端搞构建	前端搞 AI	AI 线下沙龙	AI 商业与赛道	前端跨端方案	前端工程化	前端搞 Remote	玩转 Prompt	前端搞前端	表格与表单	前端搞微前端	前端质量保障

关于 NPM 依赖管理 的复杂性

- 🧑 范文杰，前端，字节跳动；
- 🏠 某业务线 Monorepo 大仓基建负责人；
- 😊 热爱前端工程化；
- 📖 《Webpack5 核心原理与实践应用》作者；
- ✒ 公众号「Tecvan」作者；
- 😋 爱好：🦀️ 🦐️ 🦐️ 🦐️



这是一个很少被关注的话题：依赖管理



TOC

- 何谓“**依赖**”；
- Node 依赖管理方案可能存在的**问题**；
- 如何保持依赖 **健康度**？

代码片段 => NODE PACKAGE MANAGE => 跨组织分发、安装、共享消费

- 🧑 1700w Contributors
- 📦 7500w Packages
- 应用框架，如：React、Vue、Koa，以及相应的全家桶；
- 工程化工具，如：ESLint、Webpack、Babel、RSPack；
- 编码工具，如：axios、lodash、ramda 等；
- 还有很多基于 NPM Package 实现的企业内部闭源软件包；



先明确一下什么是依赖

广义上，支撑起软件运行所必要的软硬件资源都可称之为"依赖"：

- 从系统外部引入的 **代码片段**；
- 操作系统、网络、浏览器、系统时间、系统 IO。。。。
- 磁盘、内存、主板、GPU。。。
- 空气、水、电力，blabla

代码共享 => 工业化、产业化、规模化



看似完美，但。。

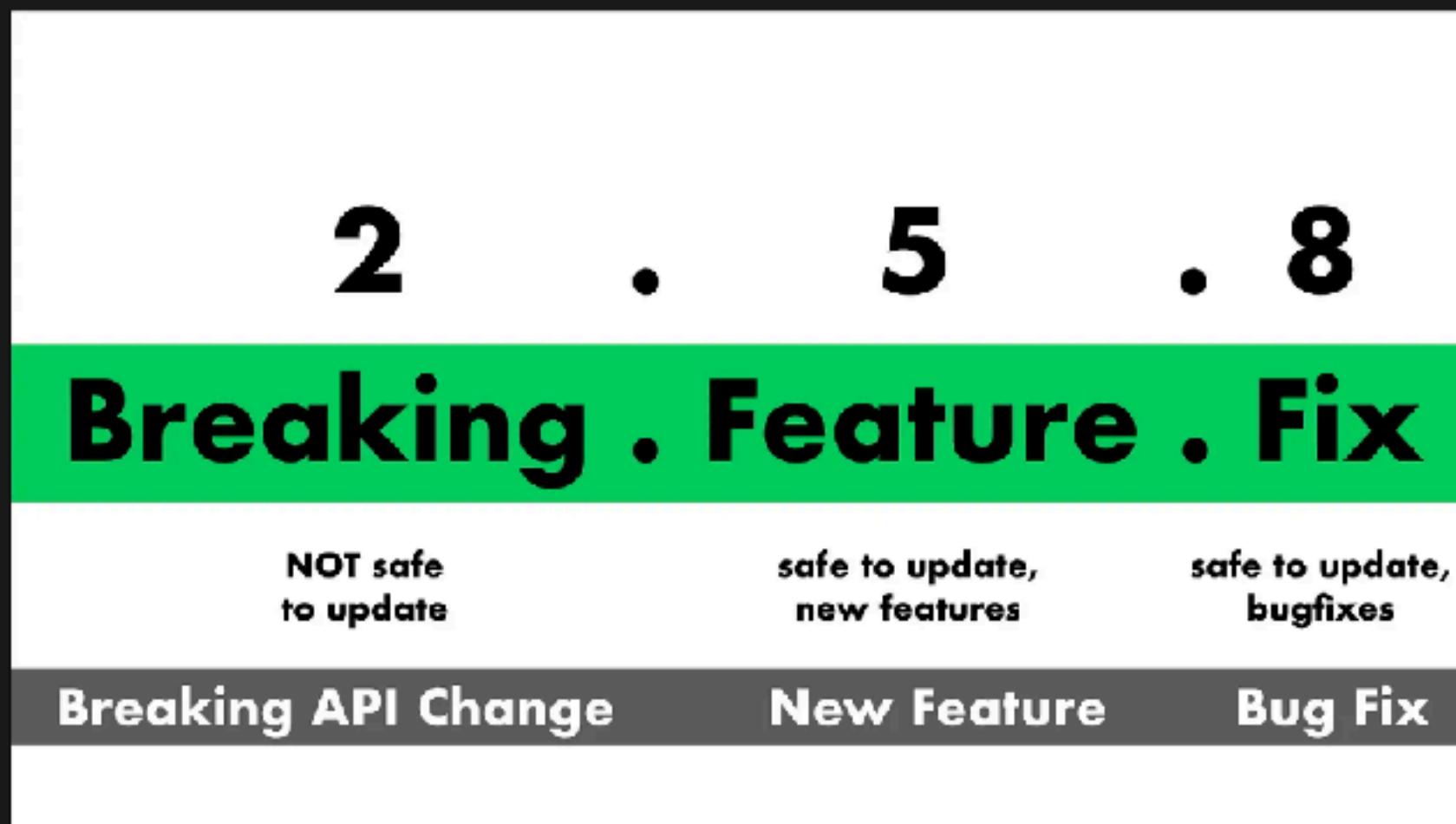
依赖管理 中可能存在许多问题



PROBLEMS: SEMVER 并不能保证稳定性



海勒姆定律：当系统用户足够多时，任何可观测行为都将被依赖



- **Patch** 版本，但修改了日志输出方式？
- **Major** 修改了接口 foo，但我们实际只依赖了 bar？
- **Minor** 版本如何保证只新增，不变更现有代码？
- 还有一大堆不遵守 semver 规则的开发者！

PROBLEMS: SEMVER 并不能保证稳定性



🔒 静态版本？

- 新版本总会带来更好的特性：性能优化、修复安全漏洞、修复 bug 等；
- 固定版本号，无法实现健康的可持续的版本更新；
- 版本累加，量变最终必然引发质变，最终需要做更 Breaking Change 的大更新；

🔒 范围版本？

- 你也不知道你的使用方式在新版本中是否 Breaking；
- 包管理器会隐式做好版本更新，可能在某次 `npm install` 之后代码就无法执行了；

PROBLEMS: 复杂的依赖类型



- **dependencies**: 生产依赖，应用程序或模块的核心组成部分，当你部署到生产或测试环境时，这些依赖项都需要被安装消费；
- **devDependencies**: 开发依赖，仅在开发过程中需要使用的依赖项，通常包括测试框架、构建工具、代码检查器、TS 类型库等，不需要在生产环境安装；
- **peerDependencies**: 对等依赖，用于指定当前 package 希望宿主环境提供的依赖，这个有点绕；
- **optionalDependencies**: 可选依赖，当满足特定条件时可以选择性安装的依赖，且即使安装失败，安装命令也不会中断。可选依赖项通常用于提供额外的功能或优化，并不是必需的；
- **bundledDependencies**: 捆绑依赖，用于指定需要一同打包发布的依赖项，用的比较少。

PROBLEMS: 复杂的依赖类型

那么，你是否清楚了解，什么场景下应该使用什么类型？可能引发什么后果？

- Web Application
- Node Application
- Webpack/ESLint/... Plugin
- Component Library
- SDK
- ...
- 工程化工具，如：Babel
- UI 框架，如：React
- TS @types/xxx
- 编码工具，如：lodash
- ...

PROBLEMS: 复杂的依赖类型

错误的类型声明可能导致：

- `dependencies => devDependencies` => CI/test/production 环境缺失依赖；
- `devDependencies => dependencies` => 性能问题；
- `optionalDependencies => dependencies` => 环境冲突、不必要的性能损耗；
- `peerDependencies =>` 依赖重复，甚至出现冲突；

PROBLEMS: 失控的依赖结构 🌋

vue create xxx =>

```
<span class="hljs-punctuation">>{</span>
  <span class="hljs-attr">"name"</span><span class="hljs-punctuation">>:</span> <span class="hljs-string">"test-vue"</span><span class="hljs-punctuation">>,</span>
  <span class="hljs-attr">"dependencies"</span><span class="hljs-punctuation">>:</span> <span class="hljs-punctuation">>{</span>
    <span class="hljs-attr">"core-js"</span><span class="hljs-punctuation">>:</span> <span class="hljs-string">"^3.8.3"</span><span class="hljs-punctuation">>,</span>
    <span class="hljs-attr">"vue"</span><span class="hljs-punctuation">>:</span> <span class="hljs-string">"^3.2.13"</span>
  <span class="hljs-punctuation">>}</span><span class="hljs-punctuation">>,</span>
  <span class="hljs-attr">"devDependencies"</span><span class="hljs-punctuation">>:</span> <span class="hljs-punctuation">>{</span>
    <span class="hljs-attr">"@babel/core"</span><span class="hljs-punctuation">>:</span> <span class="hljs-string">"^7.12.16"</span><span class="hljs-punctuation">>
    <span class="hljs-attr">"@babel/eslint-parser"</span><span class="hljs-punctuation">>:</span> <span class="hljs-string">"^7.12.16"</span><span class="hljs-punc
    <span class="hljs-attr">"@vue/cli-plugin-babel"</span><span class="hljs-punctuation">>:</span> <span class="hljs-string">"~5.0.0"</span><span class="hljs-punc
    <span class="hljs-attr">"@vue/cli-plugin-eslint"</span><span class="hljs-punctuation">>:</span> <span class="hljs-string">"~5.0.0"</span><span class="hljs-punc
    <span class="hljs-attr">"@vue/cli-service"</span><span class="hljs-punctuation">>:</span> <span class="hljs-string">"~5.0.0"</span><span class="hljs-punctua
    <span class="hljs-attr">"eslint"</span><span class="hljs-punctuation">>:</span> <span class="hljs-string">"^7.32.0"</span><span class="hljs-punctuation">>,</span>
    <span class="hljs-attr">"eslint-plugin-vue"</span><span class="hljs-punctuation">>:</span> <span class="hljs-string">"^8.0.3"</span>
  <span class="hljs-punctuation">>}</span><span class="hljs-punctuation">>,</span>
  ...
<span class="hljs-punctuation">>}</span>
```

PROBLEMS: 失控的依赖结构 🌋

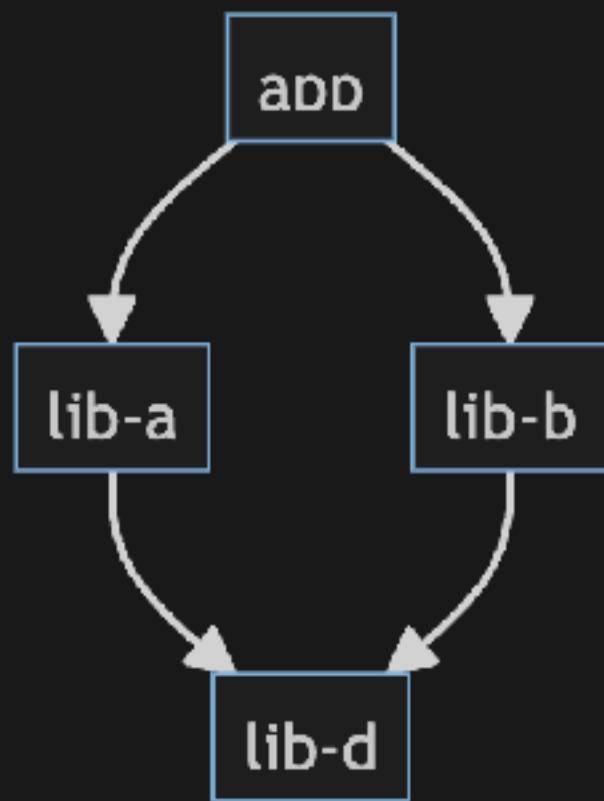
这部分次级依赖并不受你控制！而且极容易引发次生灾害：

- CPU([Version-SAT](#)) + IO 密集，初始安装、更新性能都比较差；
- 依赖网络可能存在冲突，轻则导致重复安装，或重复打包，严重时可能导致 Package 执行逻辑与预期不符，引入一些非常难以定位的 bug；
- 当依赖项之间有多层嵌套的依赖关系时，管理和解决依赖关系变得困难。添加、删除或更新依赖项可能会引发级联的变动和调整；
- 等等吧；

PROBLEMS: 依赖冲突

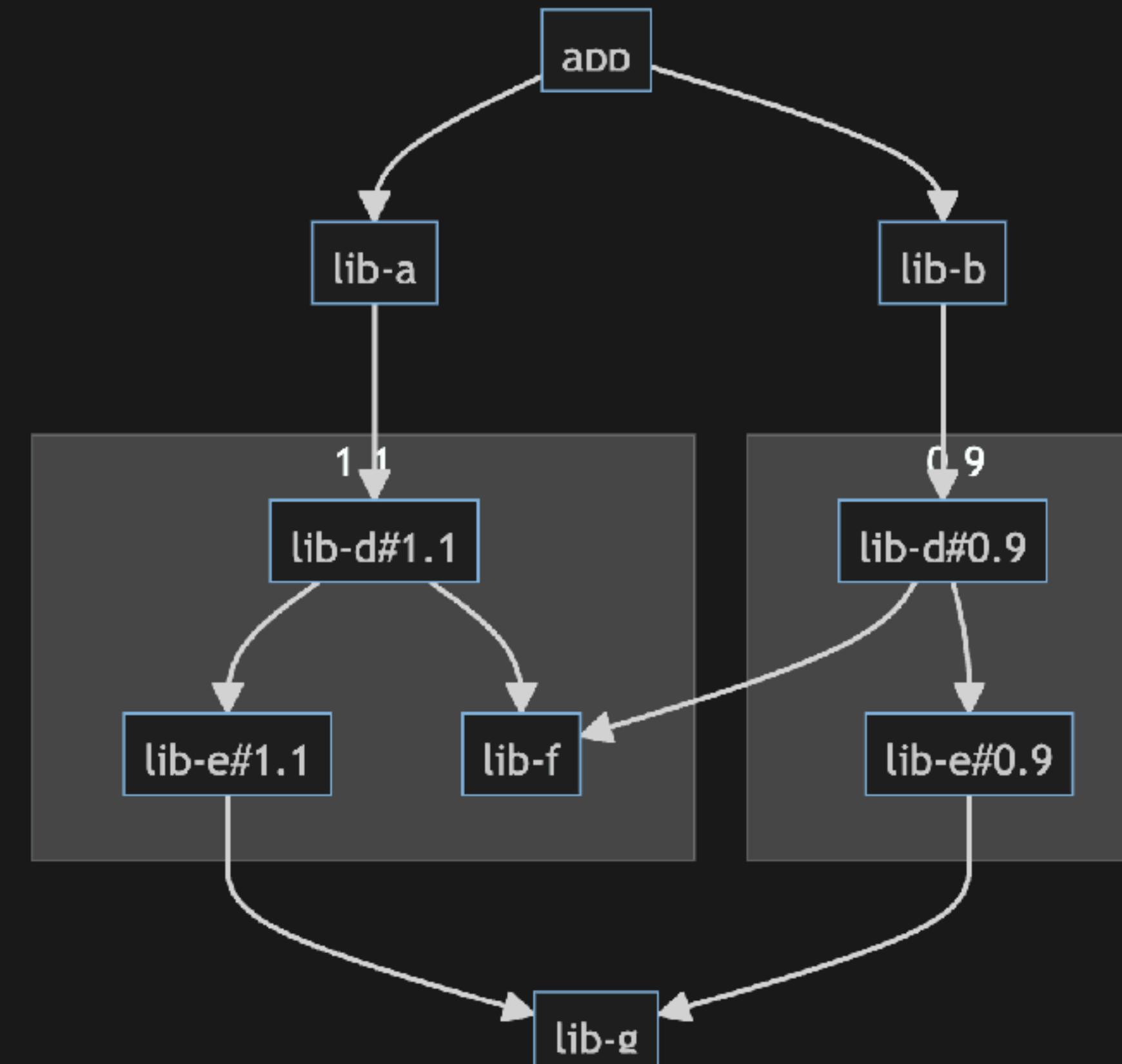
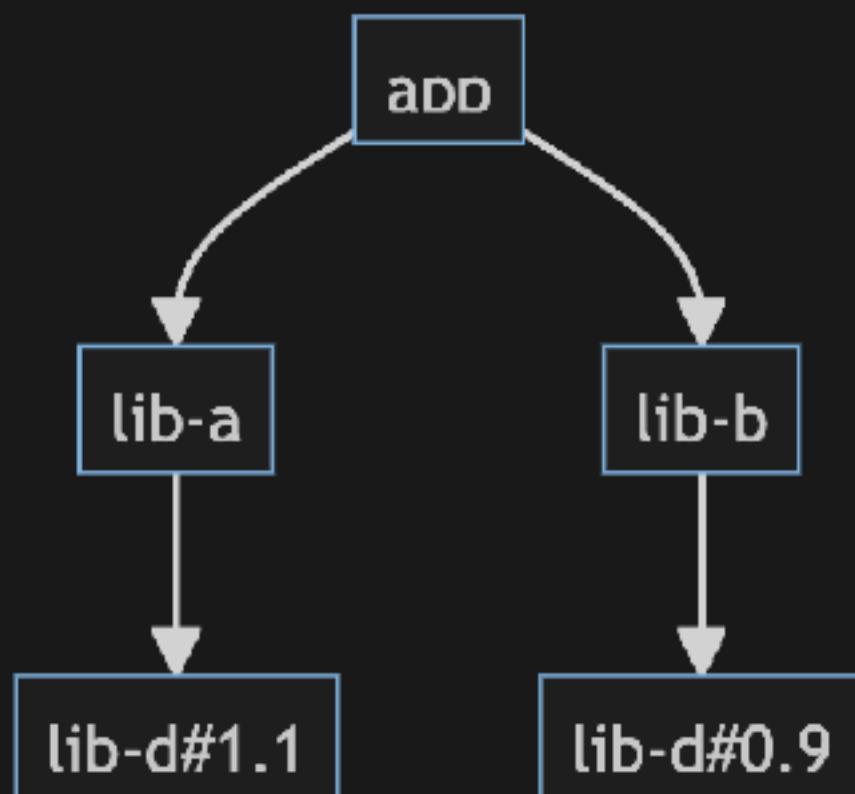


菱形依赖:

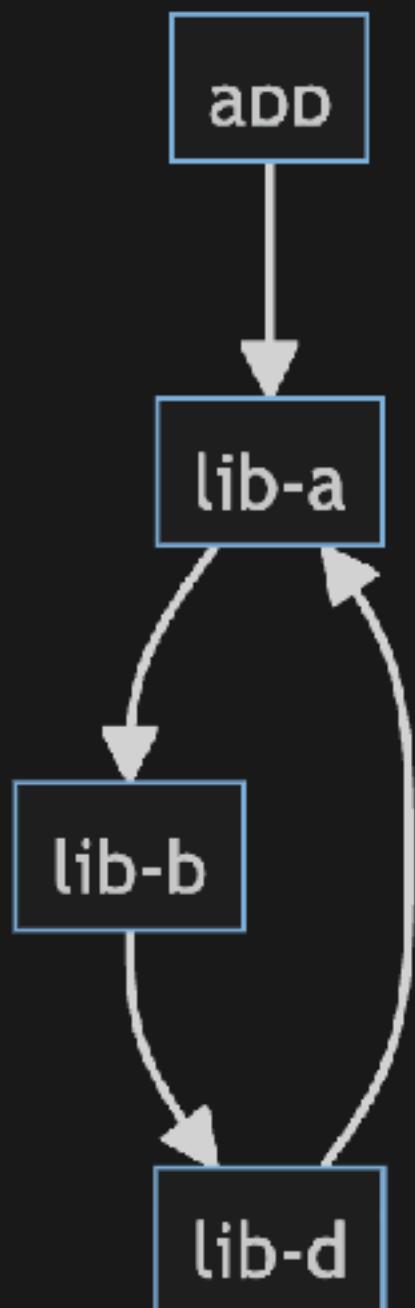


- 重复打包；
- 全局状态冲突(例如引入多次 React)；
- 重复安装；
- 复杂的 依赖网络结构：

若 **lib-a > 1.0 & lib-b < 1.0**



PROBLEMS: 循环依赖 ∞



- **编译错误**: 循环依赖会导致编译器无法确定模块的加载顺序；
- **运行时错误**: 当一个模块试图加载其依赖项时，如果依赖项又依赖于该模块，就会形成一个无限循环，使程序无法正常执行；
- **可维护性下降**: 循环依赖使代码难以理解、调试和维护；

PROBLEMS: 幽灵依赖 🕷️

《剖析 npm、yarn 与 pnpm 依赖管理逻辑》

```
<span class="hljs-punctuation">>{</span>
  <span class="hljs-attr">"name"</span><span class="hljs-punctuation">>:</span> <span class="hljs-string">"app"</span><span class="hljs-punctu
  <span class="hljs-attr">"dependencies"</span><span class="hljs-punctuation">>:</span> <span class="hljs-punctuation">>{</span>
    <span class="hljs-attr">"A"</span><span class="hljs-punctuation">>:</span> <span class="hljs-string">"xxx"</span><span class="hljs-punctua
    <span class="hljs-attr">"B"</span><span class="hljs-punctuation">>:</span> <span class="hljs-string">"xxx"</span>
  <span class="hljs-punctuation">>}</span>
<span class="hljs-punctuation">>}</span>
```

NPM2

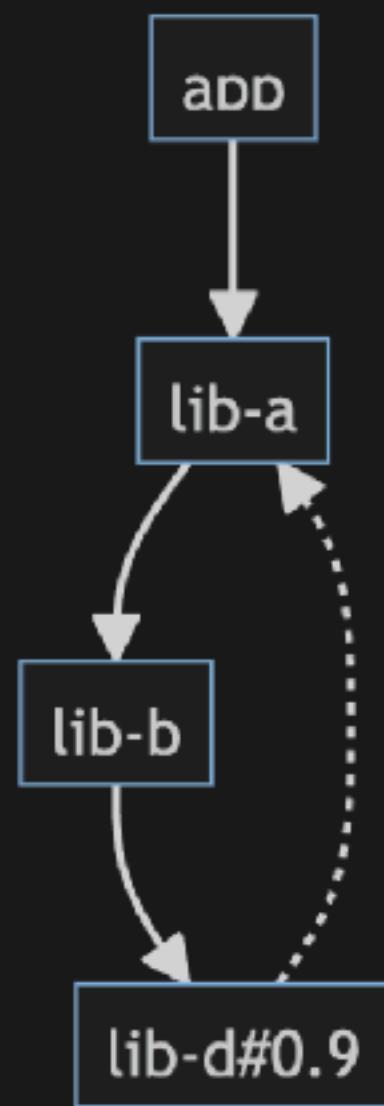
```
<span class="hljs-string">|-- node_modules</span>
  <span class="hljs-string">|-- A</span>
    <span class="hljs-string">|-- node_modules</span>
      <span class="hljs-string">|-- BaseA</span>
        <span class="hljs-string">|-- BaseB</span>
  <span class="hljs-string">|-- B</span>
    <span class="hljs-string">|-- node_modules</span>
      <span class="hljs-string">|-- BaseA</span>
        <span class="hljs-string">|-- BaseB</span>
```

PROBLEMS: 依赖传递链长 🐍

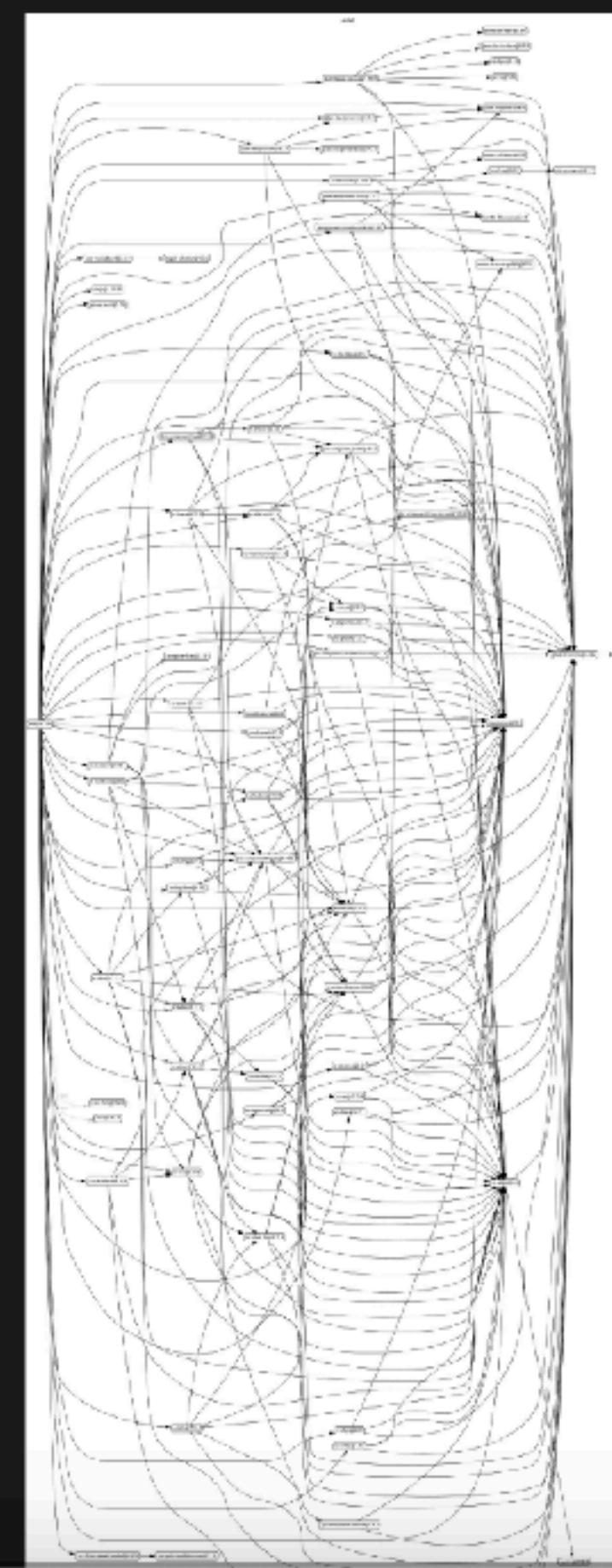
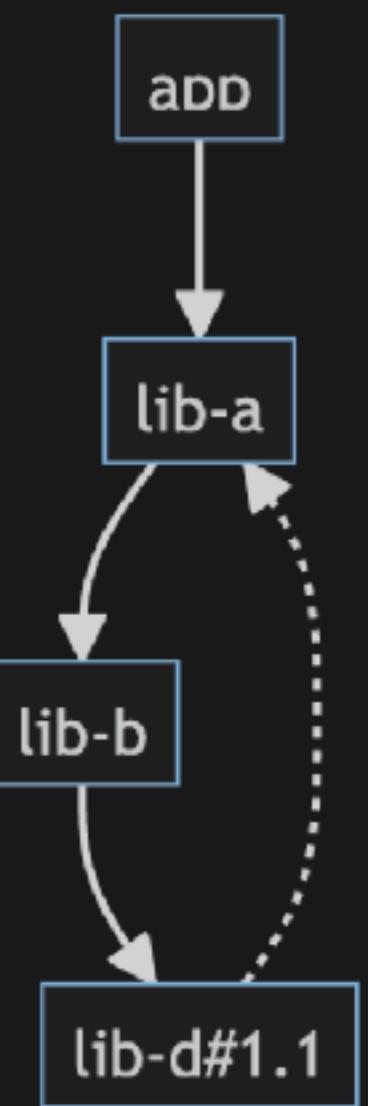
lib-d 存在异常，已经在新版本修复，但 lib-b 未及时更新；

当依赖链条很长的时候：

当前依赖网络：



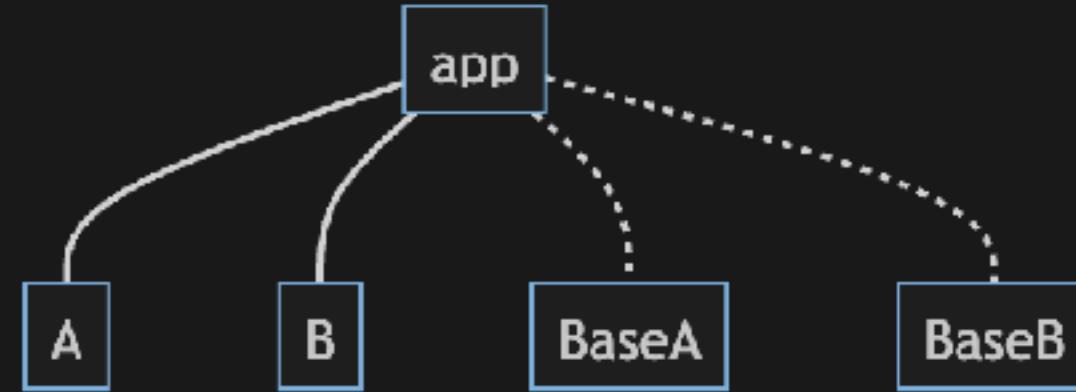
理想的依赖网络：



PROBLEMS: 幽灵依赖 😬

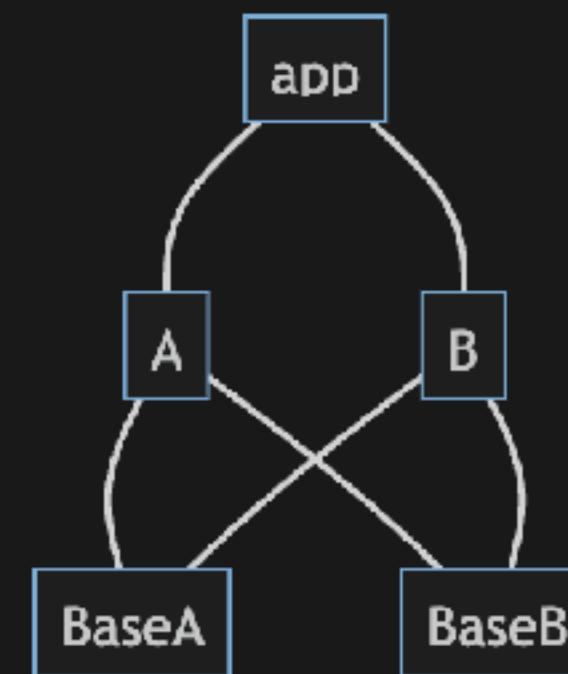
NPM3 & Yarn: 扁平化

```
<span class="hljs-string">>|-- node_modules</span>
<span class="hljs-string">>|-- A</span>
<span class="hljs-string">>|-- B</span>
<span class="hljs-string">>|-- BaseA</span>
<span class="hljs-string">>|-- BaseB</span>
```



PNPM

```
<span class="hljs-string">>|-- node_modules</span>
<span class="hljs-string">>|-- .pnpm</span>
<span class="hljs-string">>|-- BaseA</span>
<span class="hljs-string">>|-- node_modules</span>
<span class="hljs-string">>|-- Symlink_to_sub_modules</span>
<span class="hljs-string">>|-- BaseB</span>
<span class="hljs-string">>|-- A</span>
<span class="hljs-string">>|-- node_modules</span>
<span class="hljs-string">>|-- Symlink_BaseA</span>
<span class="hljs-string">>|-- Symlink_BaseB</span>
<span class="hljs-string">>|-- B</span>
<span class="hljs-string">>|-- node_modules</span>
<span class="hljs-string">>|-- Symlink_BaseA</span>
<span class="hljs-string">>|-- Symlink_BaseB</span>
```



- **Problems**: semver 并不能保证稳定性
- **Problems**: 复杂的依赖类型
- **Problems**: 失控的依赖结构
- **Problems**: 依赖冲突
- **Problems**: 循环依赖
- **Problems**: 依赖传递链长
- **Problems**: 幽灵依赖
- **Problems**: 循环依赖
- **Problems**: ...



解题的关键：

需要一种能安全引入外部依赖，且随 **时间 & 项目规模** 变化，能够持续保持依赖网络健康的方案。

SOLUTION: MONOREPO

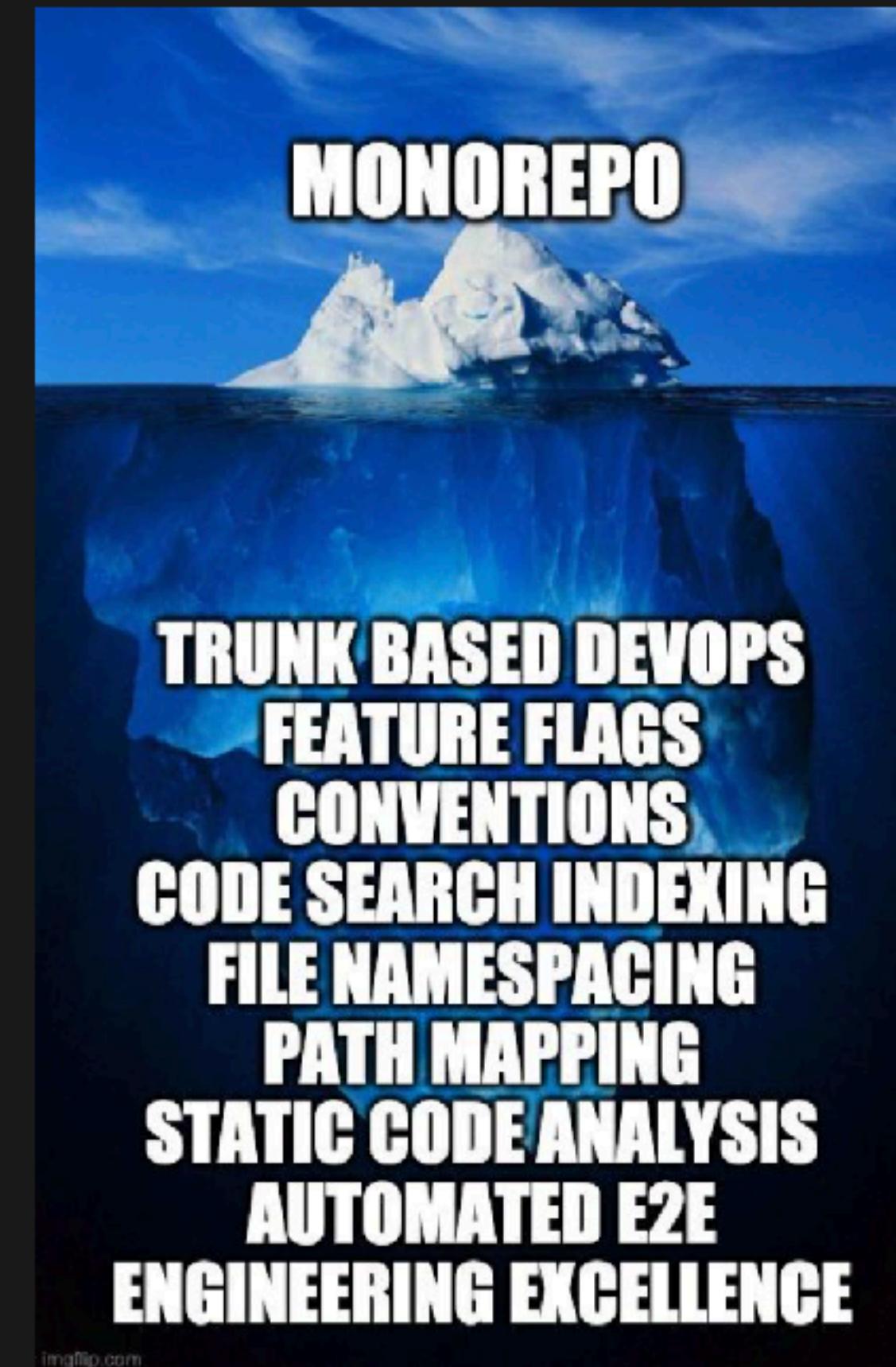
从依赖管理转化为“源码管理”

😊 PROS

- 生产、消费永远一致的版本；
- 配合单测，能够自动化确保向下兼容；
- 只有一份源码，保持“**唯一真实**”；
- 传递链条短，变更立即生效，无时间差；

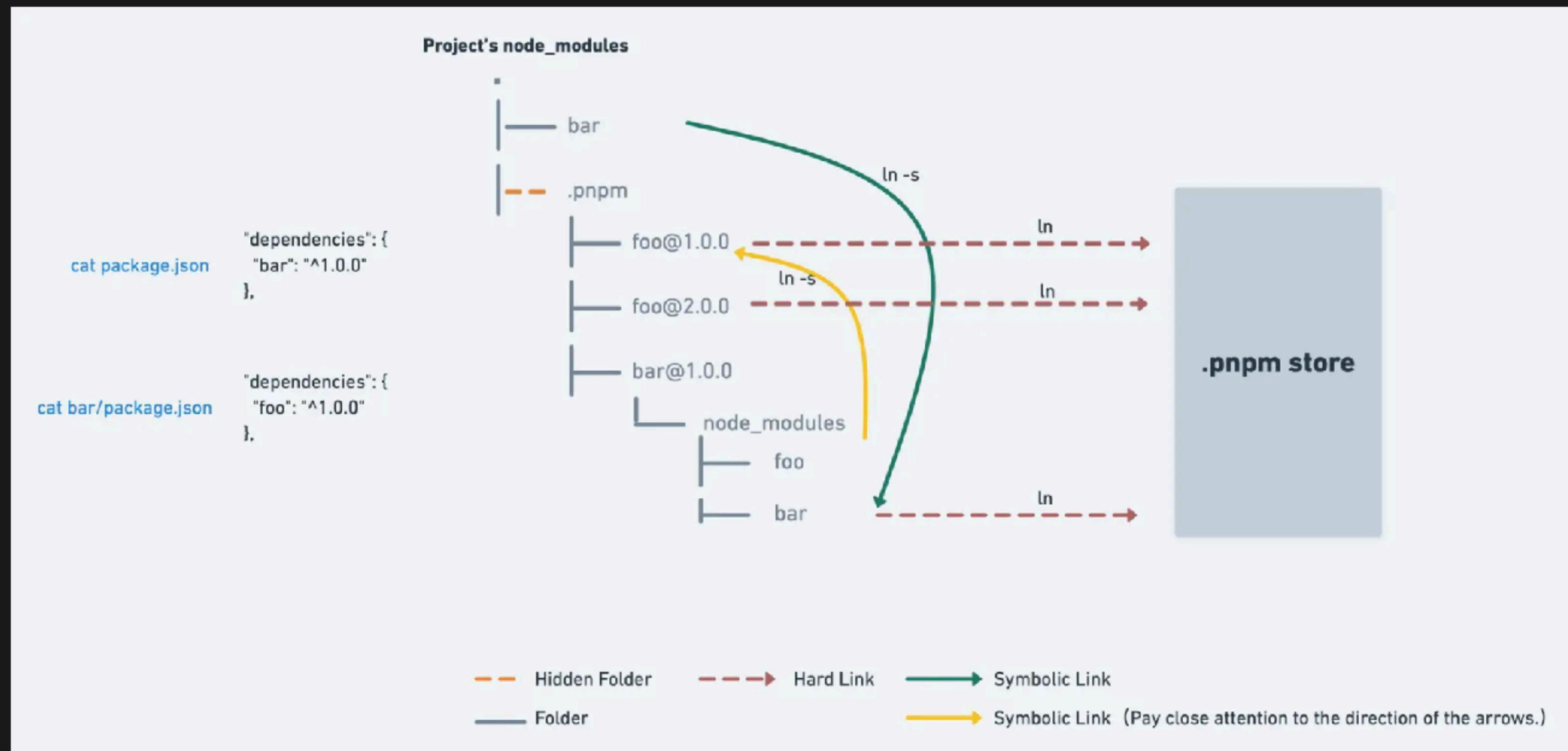
😢 CONS

- 工程技术要求更高：单测、FG、CI/CD、代码静态分析检查等；
- 容错率低，单点可能触发全局错误；
- 变更成本高，更需要回归测试；
- 只适用于组织内源码；
- **对人、团队、技术要求都非常高**；



PRACTICE: 使用 PNPM

Performance Node Package Manage



PRACTICE: 使用 PNPM

Performance Node Package Manage

- **单一包存储**: 无论多少项目用到相同包, PNPM 只安装一次包。这消除了在不同项目之间冗余复制包的情况, 减少了磁盘空间的使用和安装时间。
- **符号链接**: PNPM 使用符号链接在项目之间共享包的安装, 不需要硬复制 node_module 文件, 提升安装性能;
- **并行安装**: PNPM 可实现并行包安装;
- ...

PRACTICE: 更严格的准入审查

需要设定一个清晰的准入审查清单

Quality & Stable

- 单测、单测覆盖率、是否通过；
- 活跃度如何，ISSUE 是否及时解决；
- 是否有 Benchmark，性能数据如何；
- 二级依赖结构如何，是否足够合理；
- 静态代码检测覆盖情况如何；
- 是否有更优选择；
- ...

Developer Experience

- 是否有完备的 README，是否对使用方法及基本原理都有完整介绍；
- 接口方法是否简明易懂，接口结构是否合理；
- 代码结构如何，是否足够清晰易懂，社区是否有足够讨论热度；
- ...

PRACTICE: 更严格的准入审查

🤖 自动化工具：

- **Blacklist**
- **node_modules** 结构健康度检查，重点：重复依赖、循环依赖、过期依赖、版本冲突等；
- **Package** 健康度检查，重点：二级包结构、体积、License、新版本检测等；

PRACTICE: 常态化更新

- dev 使用 npm install; ci/cd/test/prod 使用 npm ci;
- 保持常态化更新：十次小更新的成本小于一次大更新；
- [trick] 可借助 patch-packages、pnpm patch、resolution 等手段，更新次级依赖的版本 & 代码，但尽可能少用；
- [trick] 合理的代码冗余：对小代码片段使用 copy，会比使用 Dependencies 更容易管理；

PRACTICE: 自动化测试

- 理想情况下，任何变更，只要能跑通自动化测试，风险都会非常非常小；
- 单测覆盖率足够高之后，可以大胆做出依赖更新还是源码变更，重构不再是问题；
- 这事放在业务团队确实很难，但“长期正确”；

- **Problems**: semver 并不能保证稳定性
 - **Problems**: 复杂的依赖类型
 - **Problems**: 失控的依赖结构
 - **Problems**: 依赖冲突
 - **Problems**: 循环依赖
 - **Problems**: 依赖传递链长
 - **Problems**: 幽灵依赖
 - **Problems**: 循环依赖
 - **Problems**: ...
- **Solution**: Monorepo
 - **Practice**: 使用 PNPM
 - **Practice**: 更严格的准入审查
 - **Practice**: 常态化更新
 - **Practice**: 自动化测试
 - ...

THANKS FOR READING.



早

兔年跳个好团队

早早聊 跳槽营



职业路线辅导
跳槽套路指导
模拟面试摸底
题库讲解直播
面经直播分享
优质团队内推
优质简历分享
职业成长视频

八大服务 破解 七大跳槽难题

学历硬伤
跳槽高频
项目简单
底子薄弱
深度不足
业务不精
管理不通

这次,你将 **TOGETHER**

一起走满5年

早早聊天使票预售



三周年限定版天使年票

最后 200 张



解锁 2020 - 2024 年大会年票会员

2020-2024 年 800 场大会直播永久观看权

晋级职场高阶能力拓展营，40 位顶尖高手带你玩

额外的 50 场群内直播分享，剖析业务设计思路，现象级产品解读，复盘经典运营活动，分享成体系管理实践，全方位开商业天眼，加速技能破圈

获得 Scott 职业建议与指导 + 管家式服务

Scott 职业建议与指导 2 次，早早聊对你的商业作品评估和投资，跳槽服务折扣及专享招聘服务，优质人脉共享，助力技术成果变现...

拥有早早聊 6 年利润分红

成为早早聊天使会员，享有 2024 ~ 2029 年盈利分红权

扫码咨询
了解更多详情





跟我一起学刁吧！

每月挑一个周六，一起充电成长！



快扫码上车早早聊
500个录播等你看

2023

全年
行程

2022

全年
行程

前端
自由职业
2021
全年
行程

页面搭建专场
小程序组件化

2/4	2/25	3/18	4/1	4/8	4/15	5/7	5/20	6/3	6/10	6/17	7/15	8/12	9/23	10/14	10/15	10/21	10/22	10/28	11/4	11/11	11/25	12/9	12/16	12/23	
健康指南	前端搞规划	前端搞插件	杭州 GPT沙龙	Node.js	低代码无代码	前端搞管理	前端搞插件	上海 GPT沙龙	武汉 GPT沙龙	求职就业新起点	前端性能优化	AI Chatbot	Night Of AI	前端搞构建	前端搞 AI	技术人 AI 搞钱	表格与表单	前端跨端方案	前端工程化	玩转 Prompt	3D 互动	前端搞 Rust	前端搞浏览器	CI/CD	前端质量保障
1/29	2/26	3/19	4/9	4/23	5/14	5/25	5/28	6/18	6/25	7/16	7/17	8/6	8/27	9/3	9/17	10/22	10/29	11/26	11/27	12/3	12/11	12/24	小程序专场	远程工作专场	
前端职业前路	TS 技术专场	管理专场	跳槽面试专场	组件专场	监控专场	裁员仲裁	性能专场	成长与晋升	跨端专场	海外工作专场	一站式基建	工程化专场	低代码前端	音视频专场	低代码微前端	工程化跨端性能	框架专场	可视化专场	AntV 可视化	测试专场	远端工作专场	小程序专场	远程工作专场	小程序专场	
大一 招聘面试	前端搞 CI/CD	前端玩转算法	前端玩转晋升	前端玩转互动	前端搞 Flutter	前端 WebGL	前端搞可视化	前端玩转 BFF	前端 GraphQL 专场	前端质量专场	前端安全专场	前端搞 Vue	前端搞 IDE	前端搞 Node.js	前端转管理	前端搞搭建	前端搞监控	前端搞规划	前端转前端	前端大厂面试	在线文档专场	前端跨端跨栈	女生职业专场	职业成长与晋升	前端搞报表