



100  
TOP100 CASE STUDIES OF THE YEAR

100 主办方 msup®  
TOP100 CASE STUDIES OF THE YEAR

# 前端 Monorepo 在字节跳动的实践

林宜丙 字节跳动 前端工程师

# 讲师简介



林宜丙  
前端工程师

“

字节跳动-Web Infra-前端架构工程师，多年前端工程化经验，致力于帮助前端工程师更好地管理和治理工程，目前负责字节跳动前端 Monorepo 解决方案的设计及其业务落地工作，有丰富的 Monorepo 实践和治理经验。

”

- 1 现代前端工程开发
- 2 Monorepo 简介
- 3 问题及其实践
- 4 整体落地情况
- 5 总结与展望

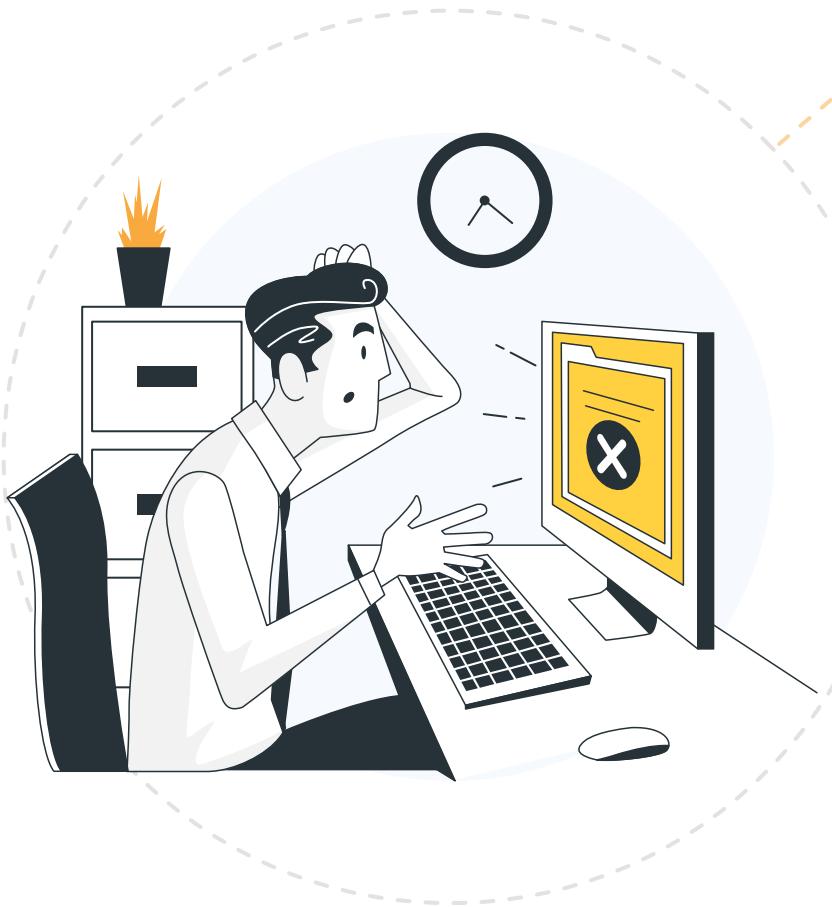
# 现代前端工程开发-前端工种趋势



# 现代前端工程开发-代码项目趋势



# 现代前端工程开发-代码研发痛点



## 项目基建重复

每次新增项目都需要重复配置



## 代码复用困难

跨项目的代码复用和调试繁琐



## 工作流程割裂

一个功能多次开发、合码、上线、验证

# 那么，如何应对呢？

# Monorepo! ! !

- 1 现代前端工程开发
- 2 Monorepo 简介
- 3 问题及其实践
- 4 整体落地情况
- 5 总结与展望

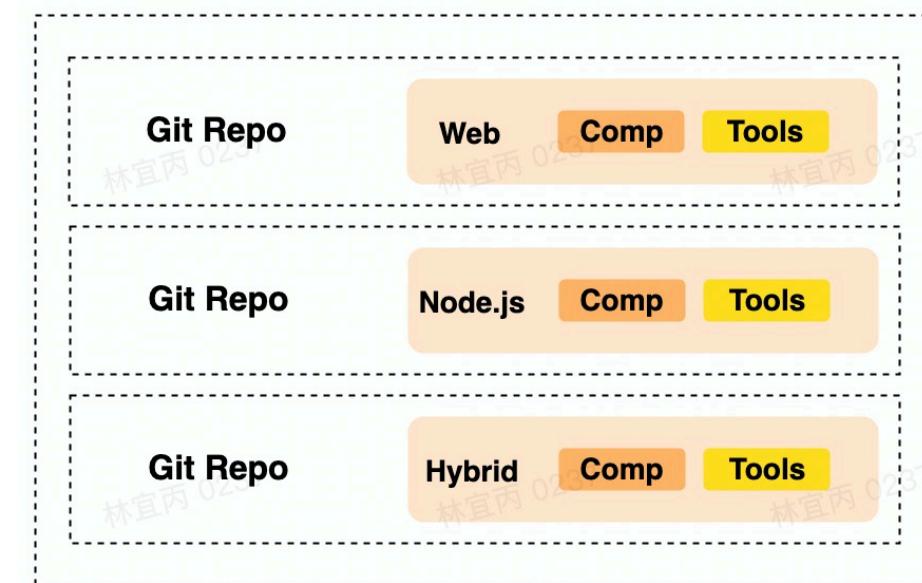
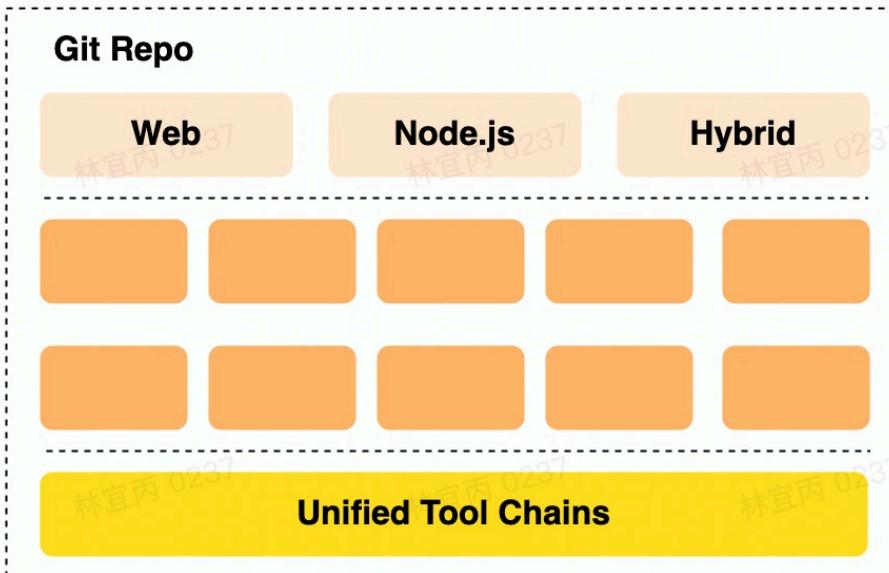
## Monorepo 是什么？

Monorepo

子项目组织到一个仓库中统一管理

Polyrepo

子项目分布到不同的仓库中管理



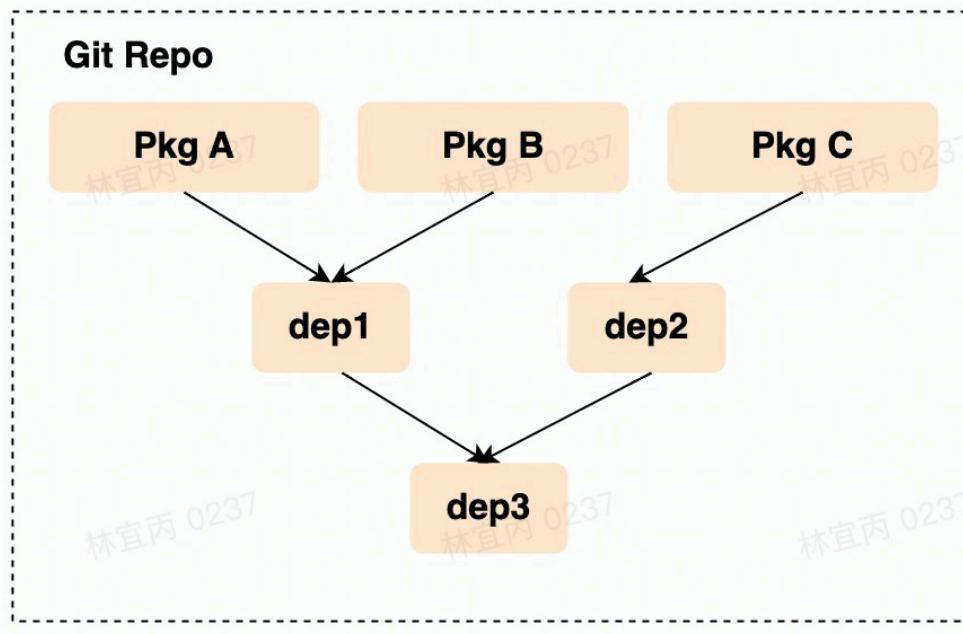
Monorepo = 单个仓库 + 多个独立项目

# Monorepo 简介

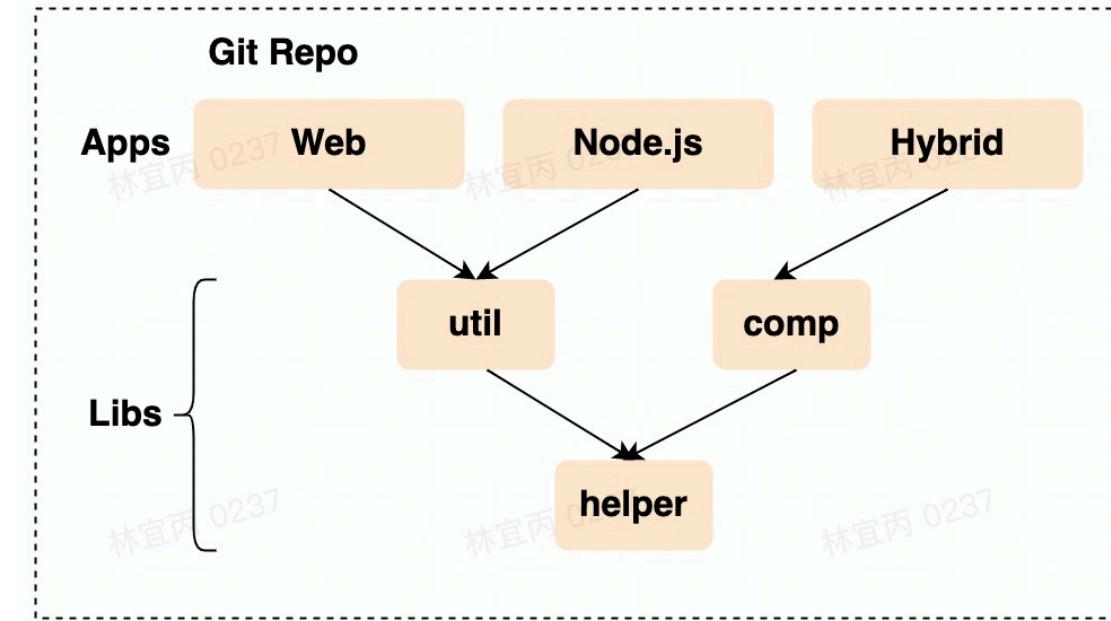
## “Lib” 型 Monorepo

VS

## “App” 型 Monorepo



- 包含多个 package，一般不需要部署
- packages 需要发布到 registry



- 包含多个业务项目，业务项目独立部署
- 包含多个业务项目共享的组件、工具函数等 Lib
- 业务项目和 Lib 一般不发布到 registry

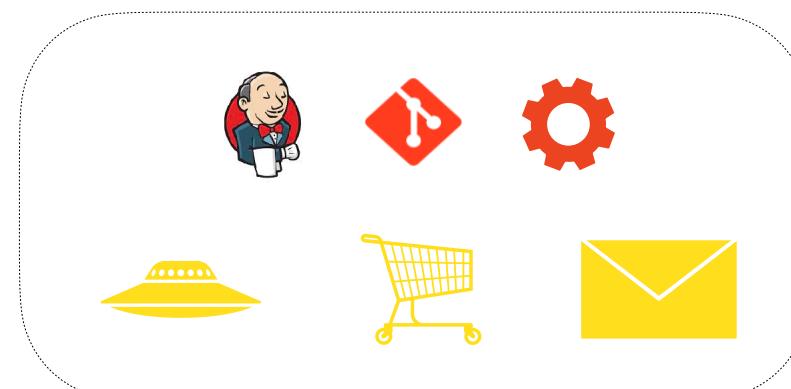
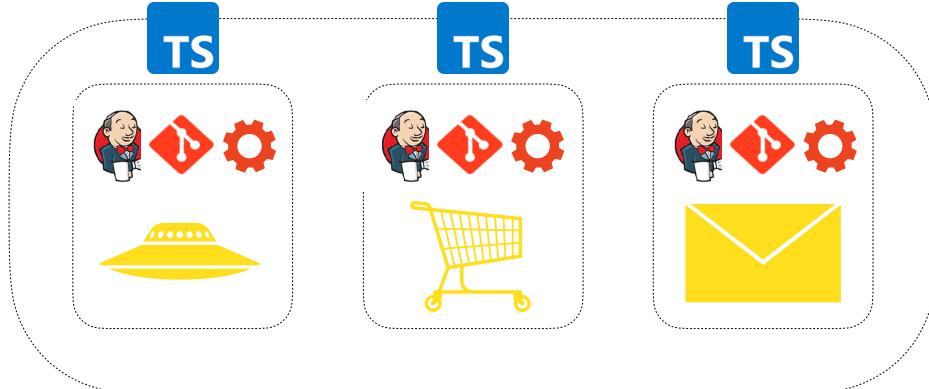
# Monorepo 是如何解决痛点的？

## 降低多项目维护成本

Polyrepo

VS

Monorepo



- 每个项目都需要有同学创建维护
- 基建重复，调整多个项目配置成本高
- 开发多项目时需要来回切换仓库和环境



- 1-2 个同学负责所有项目的公共架构维护
- 所有子项目复用 Monorepo 的一套基建
- 一键启动多个子项目的调试、构建

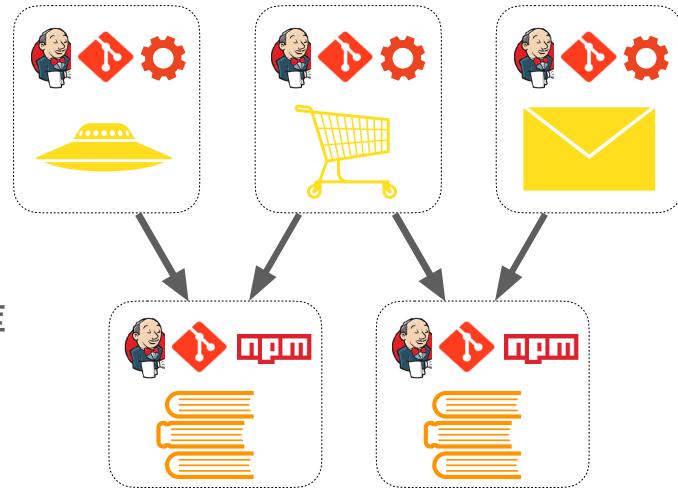
## 便捷的代码共享

### Polyrepo

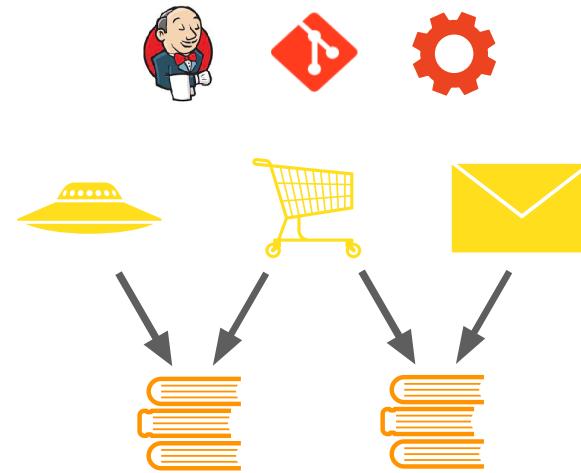
vs

### Monorepo

为每个模块设置单独的仓库



一键生成公用模块



- 公用代码的维护成本高，调试、升级繁琐
- 代码复用率低

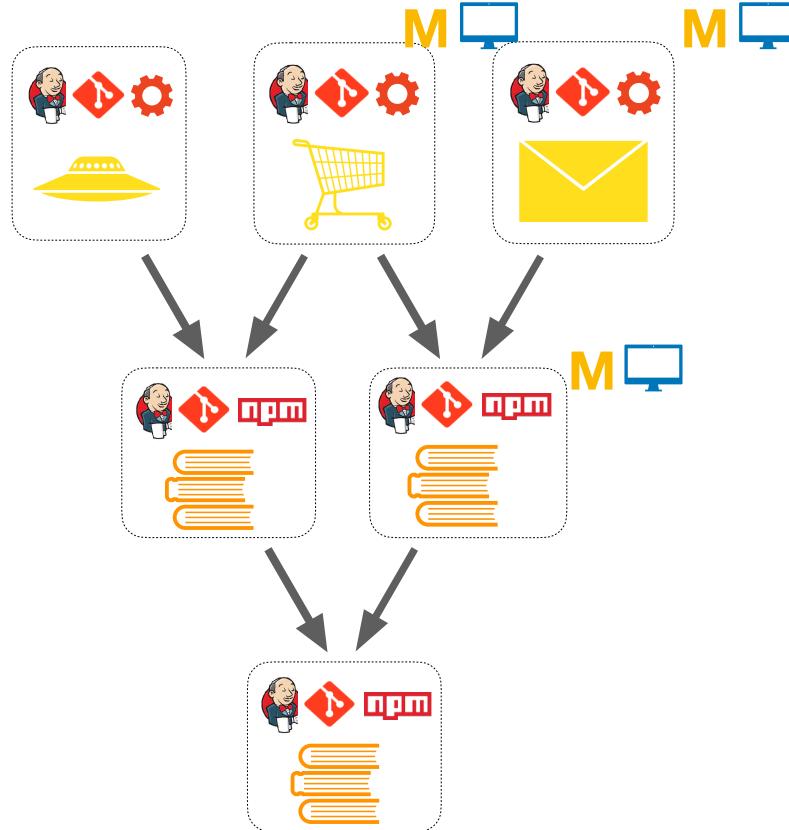
- 便捷的代码共享，一键引入、调试公用模块
- 代码复用率高

## 自动化的多项目工作流

业务需求可能涉及多个项目

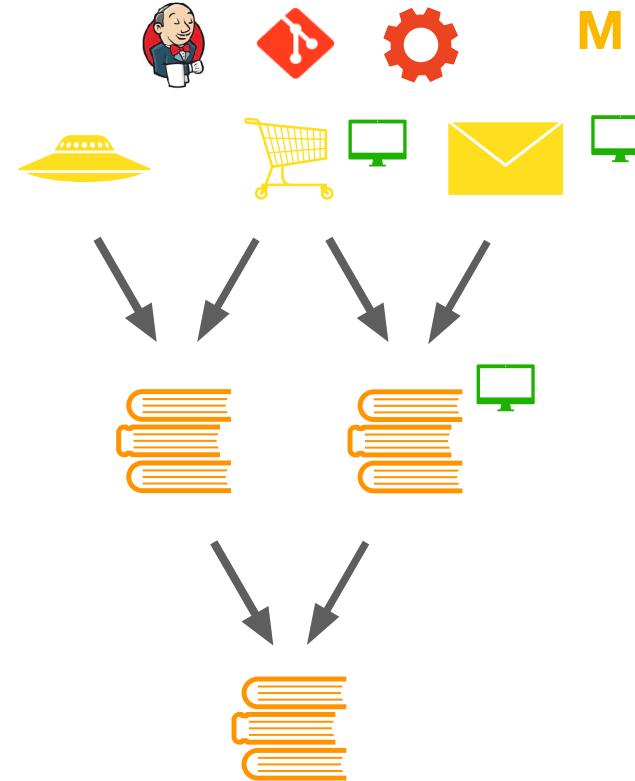
多项目工作流不连续

### Polyrepo



vs

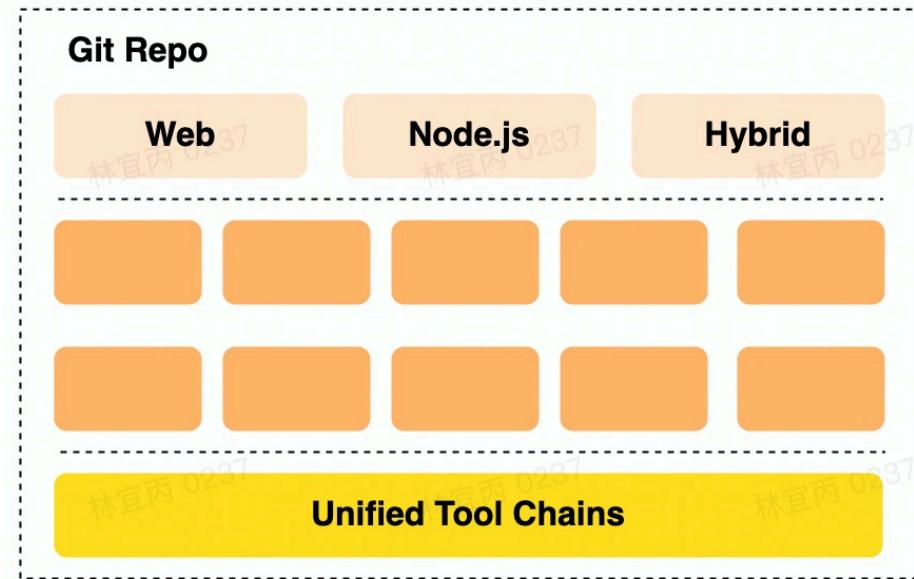
### Monorepo



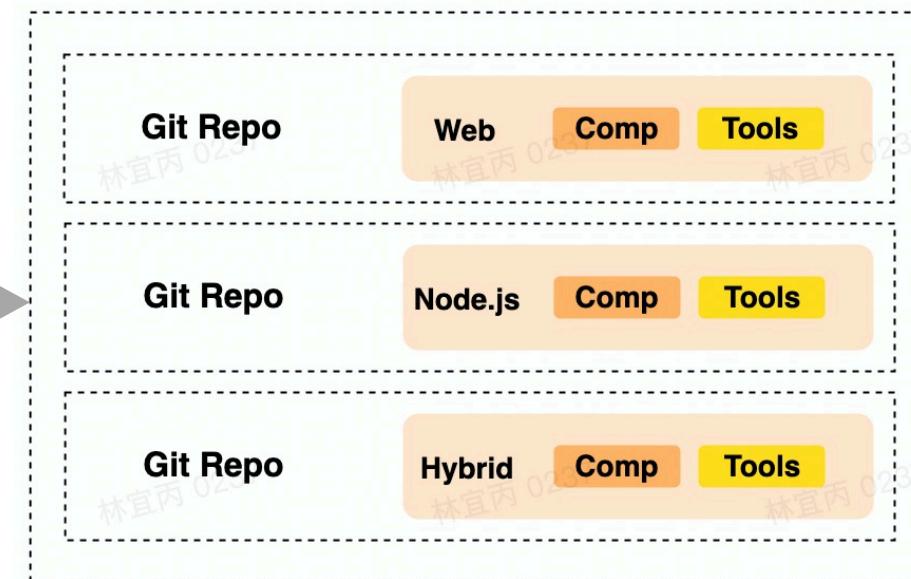
一次性调整并提交多个项目

自动、连续的 CI 流程

## Polyrepo



## Monorepo



工作流程割裂

流程规范统一

代码复用困难

代码共享便捷

项目基建重复

项目基建复用

Monorepo 将多个项目作为一个工程整体，当需要同时开发维护多个项目的时候，推荐使用 Monorepo

- 1 现代前端工程开发
- 2 Monorepo 简介
- 3 问题及其实践
- 4 整体落地情况
- 5 总结与展望

# 问题及其实践



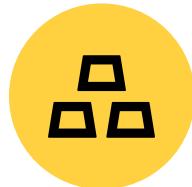
## 用户教育，上手门槛

紧密贴合字节基础设施



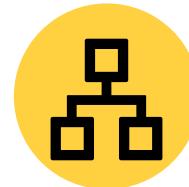
## 规模增大，性能下降

多种方式提高构建效率



## 代码屎山，持续劣化

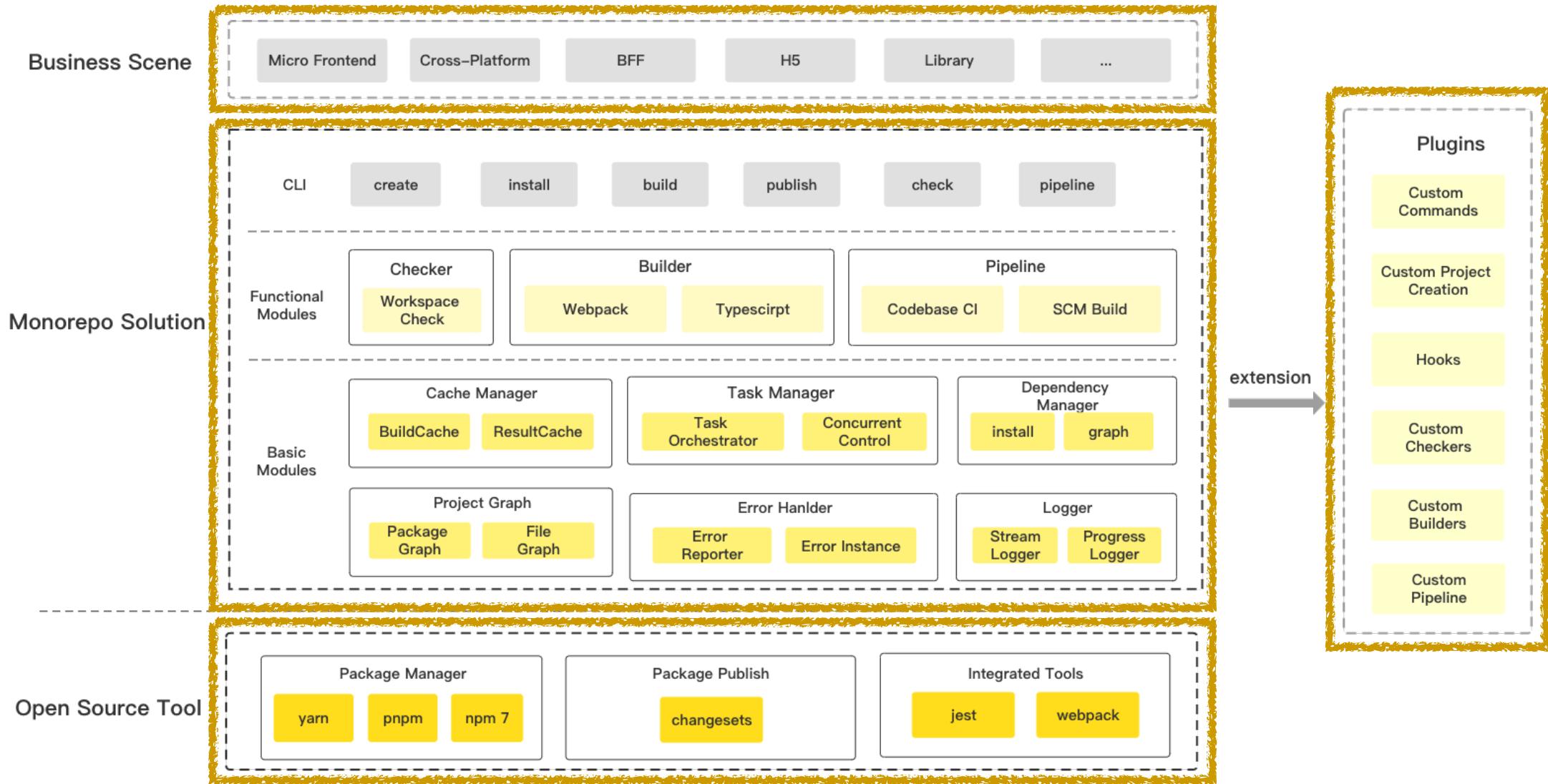
Checker 机制规范检查



## 业务各异，难以扩展

支持丰富的扩展能力

# 自研方案的整体架构



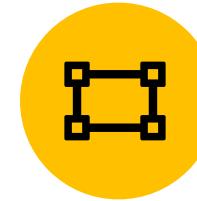
# 自研方案的实践-用户教育，上手门槛

## 紧密贴合字节 基础设施



### 内置脚手架

一键创建字节常见应用类型



### CI/CD 能力

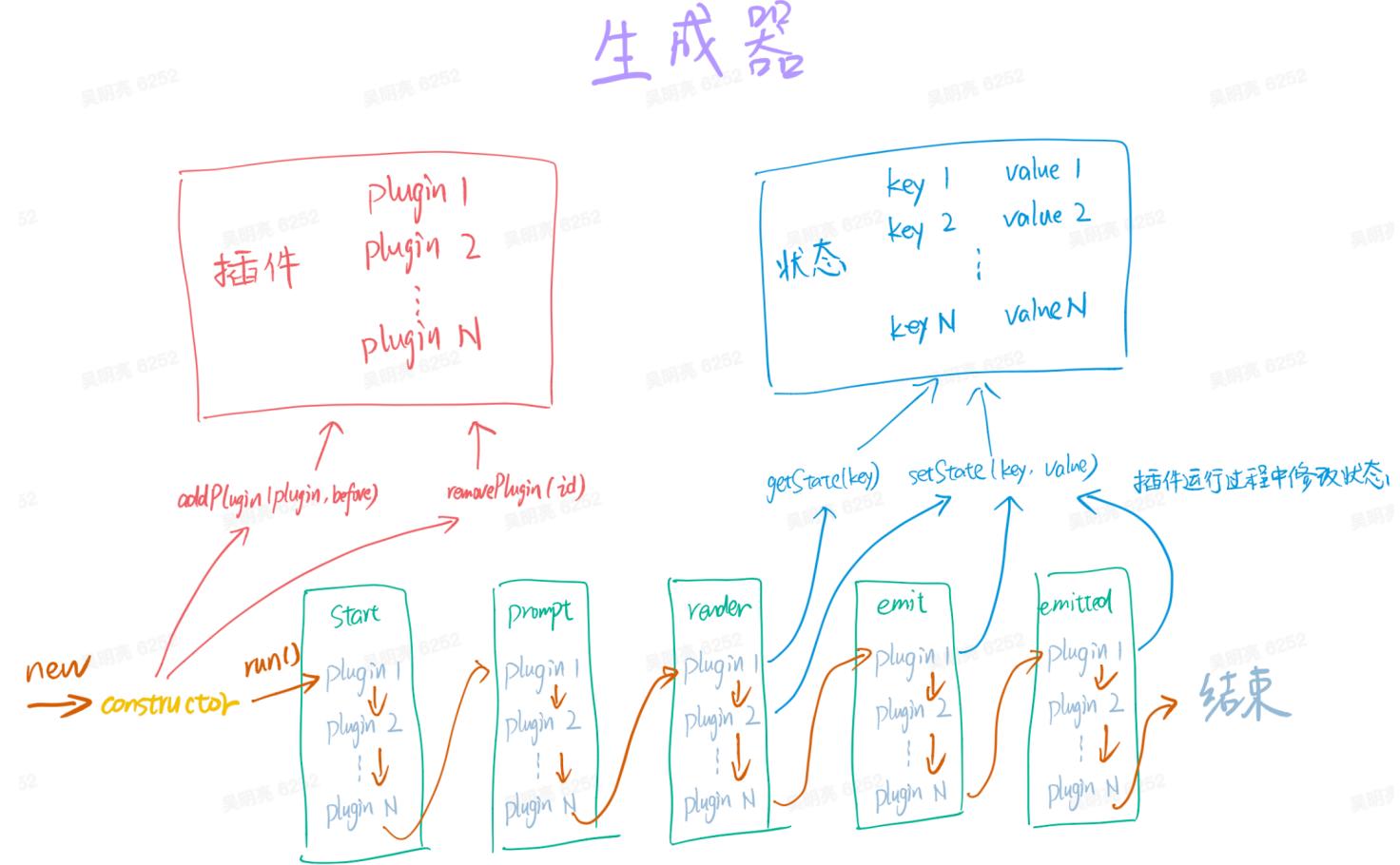
适配字节基建的发包和构建流水线



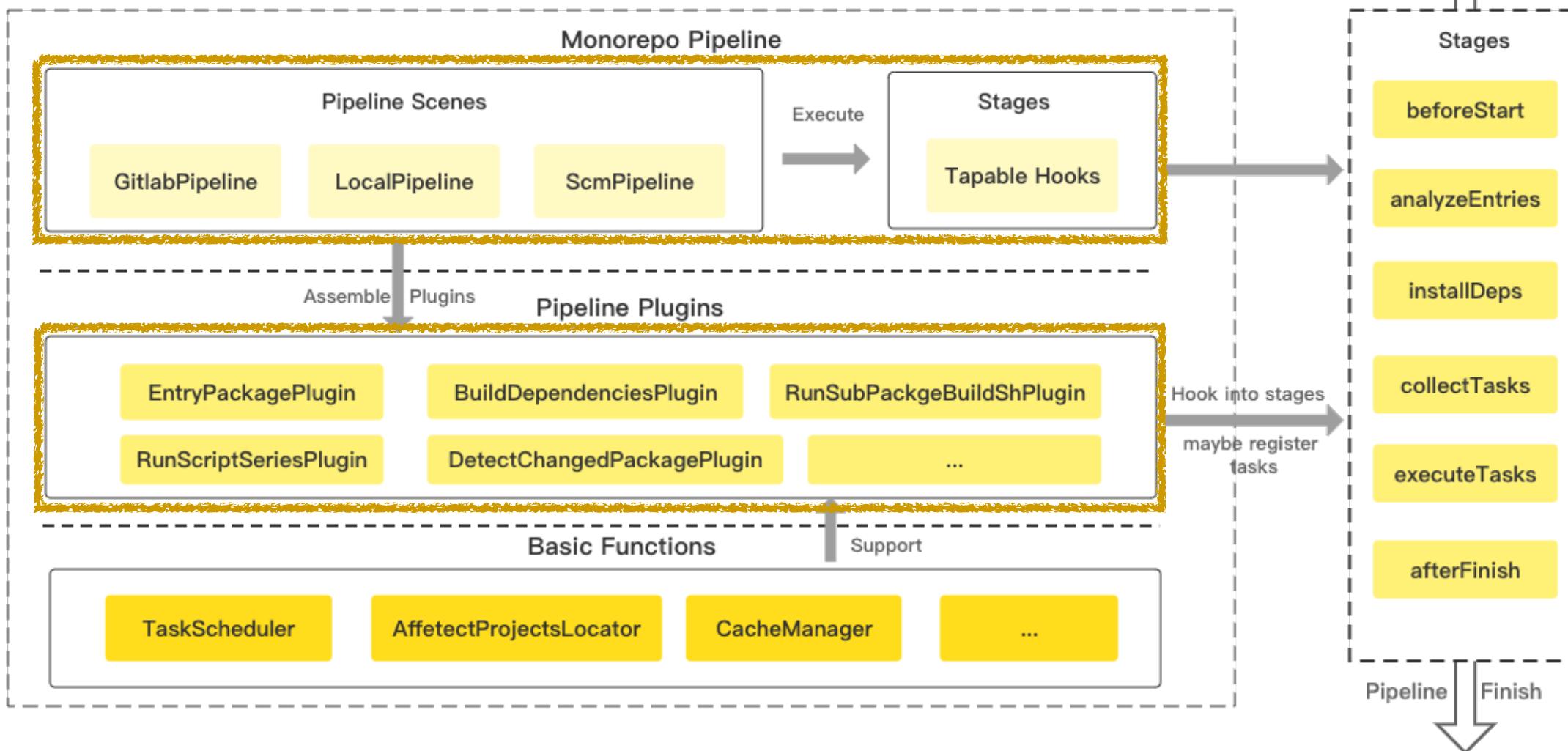
### 可视化扩展

通过界面教育和引导用户使用

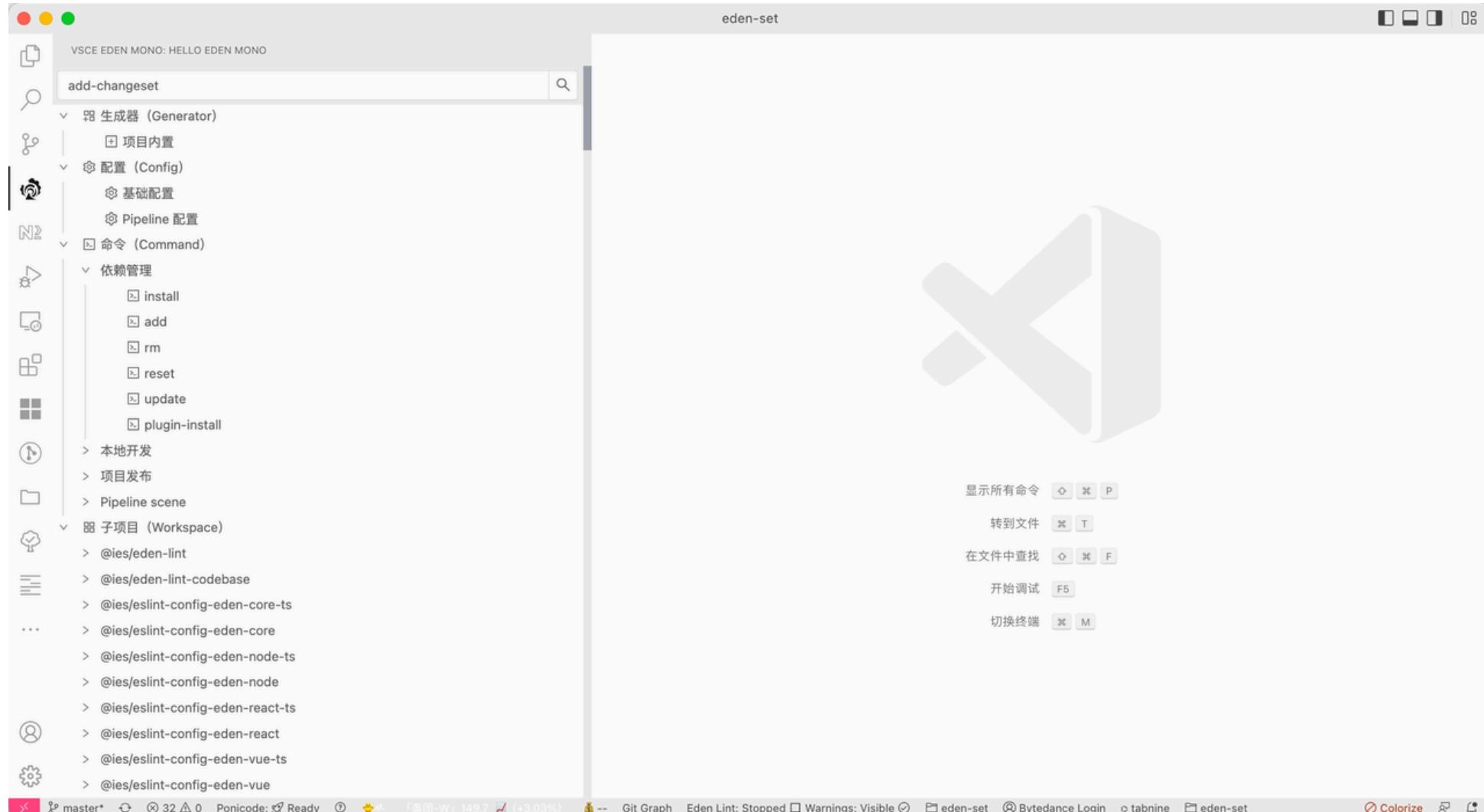
# 自研方案的实践-内置脚手架



# 自研方案的实践-CI/CD 能力



# 自研方案的实践-可视化扩展



| 自研方案的实践-规模增大，性能下降

# 多种方式提高 构建效率



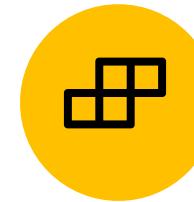
## 任务并行能力

采用最大限度的批量任务并行加速



## 多级缓存能力

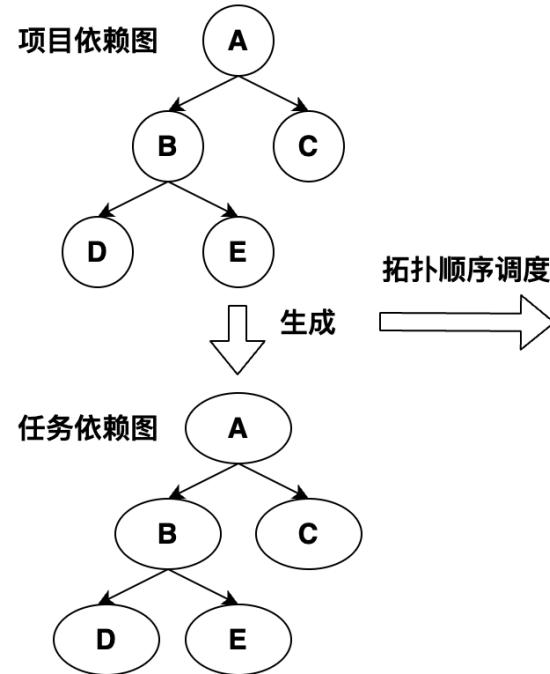
对依赖安装、构建产物、测试结果等实现了多级缓存



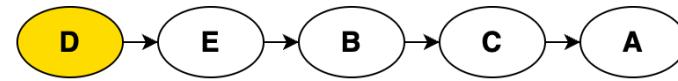
## 按需构建能力

根据依赖图以及代码更改的影响面来进行构建、测试

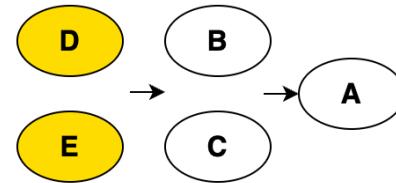
# 自研方案的实践-任务并行能力



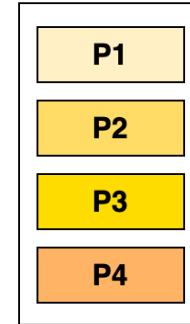
方式1：串行



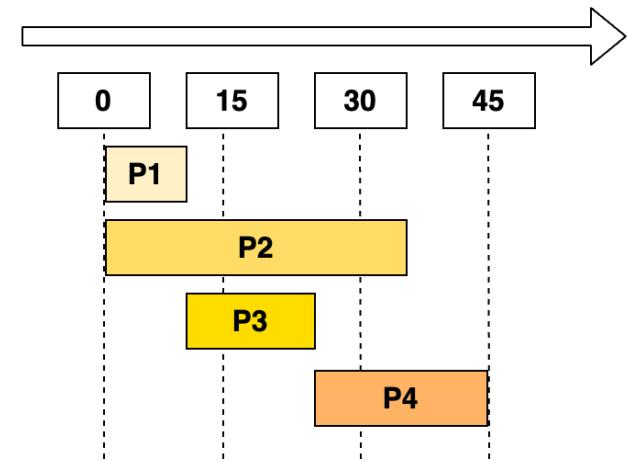
方式2：并行



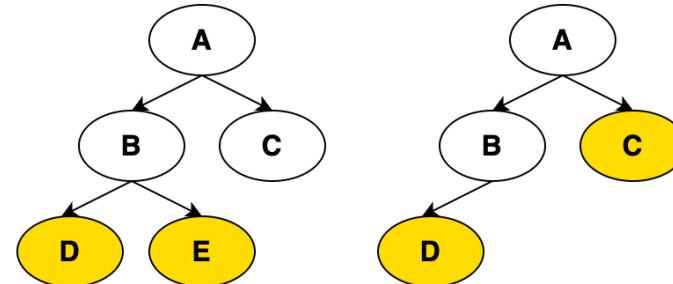
并发数：2



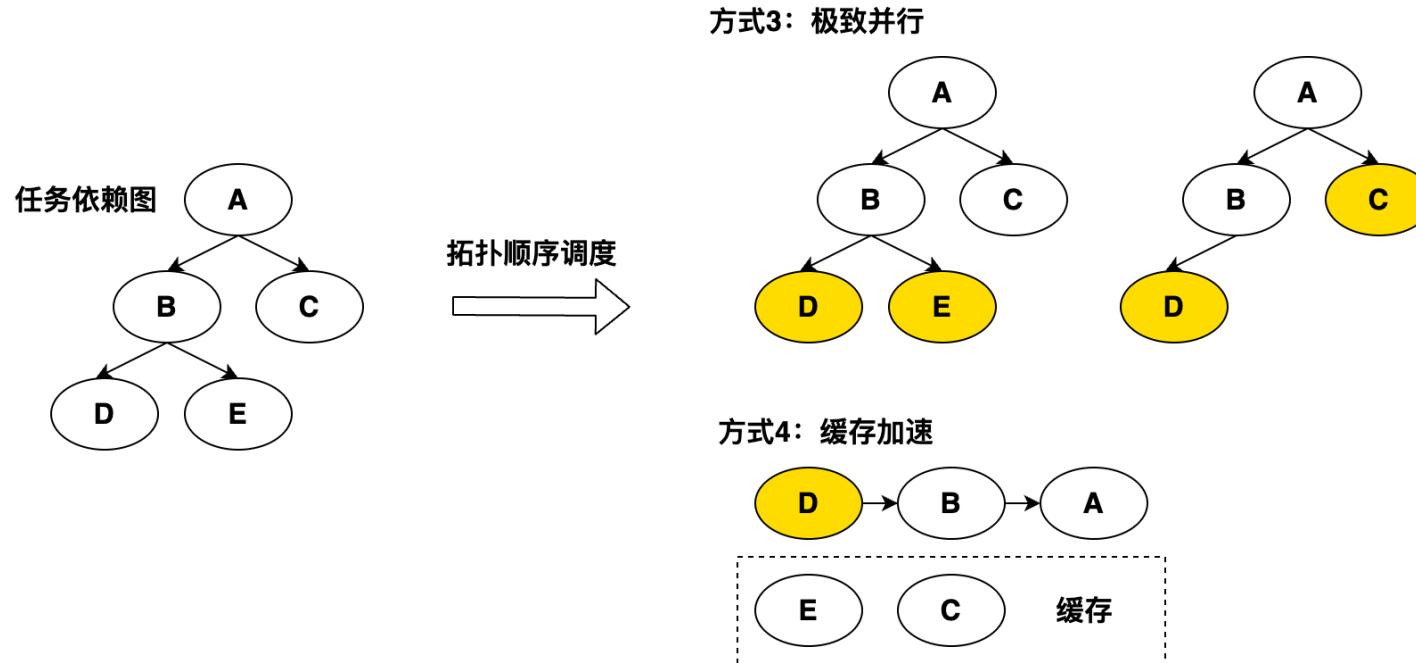
并发数：2



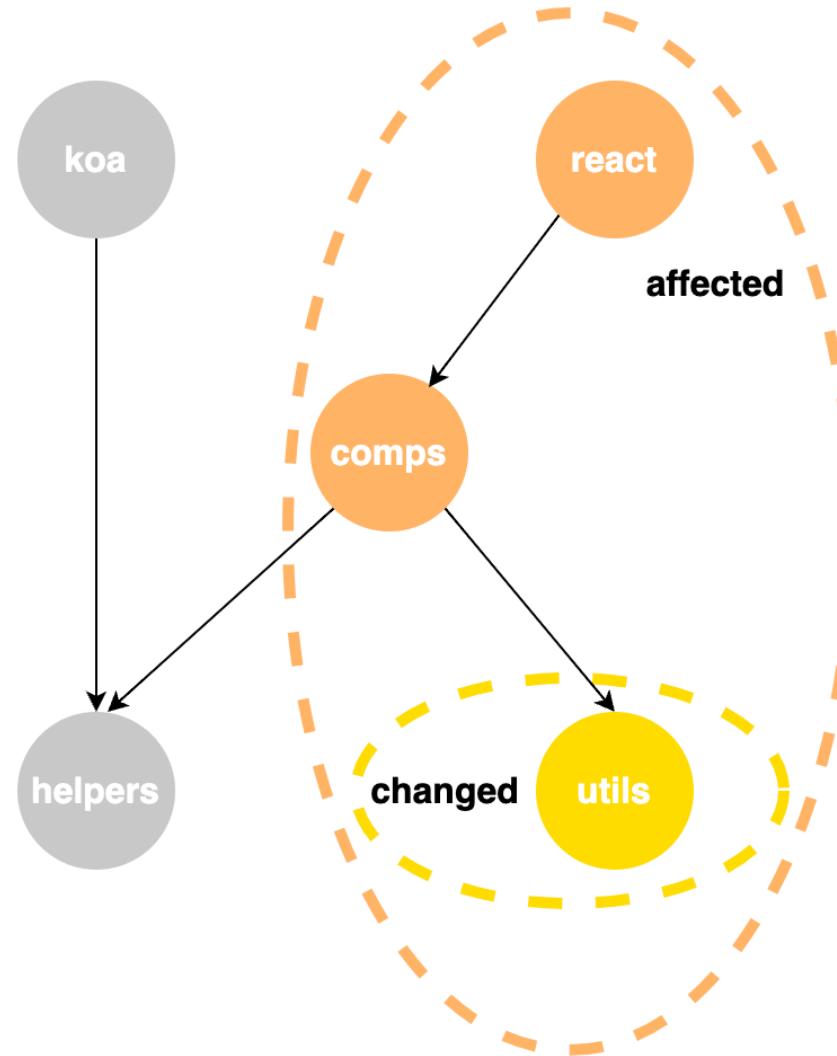
方式3：极致并行



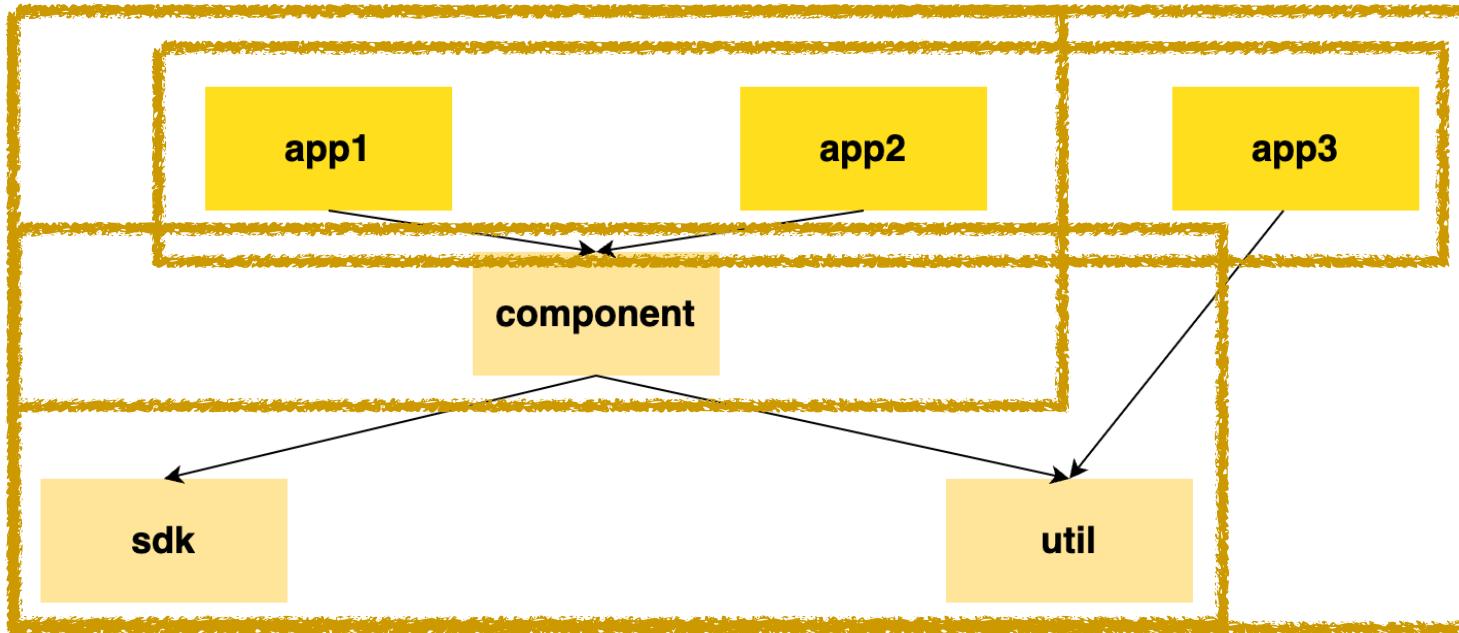
# |自研方案的实践-多级缓存能力



# |自研方案的实践-按需构建能力

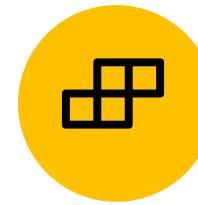


# 自研方案的实践-提升效果



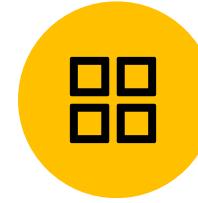
|     | App1   | App2   | App3   |
|-----|--------|--------|--------|
| 全量  |        | 17.72s |        |
| 按需  |        | 8.94s  |        |
| 无缓存 | 16.94s | 16.13s | 10.77s |
| 有缓存 | 9.74s  | 8.94s  | 7.55s  |

# Checker 机制 规范检查



## 内置多种 Checker

确保项目基本的可维护性



## 自定义 Checker

编写基于各自团队的规范检查

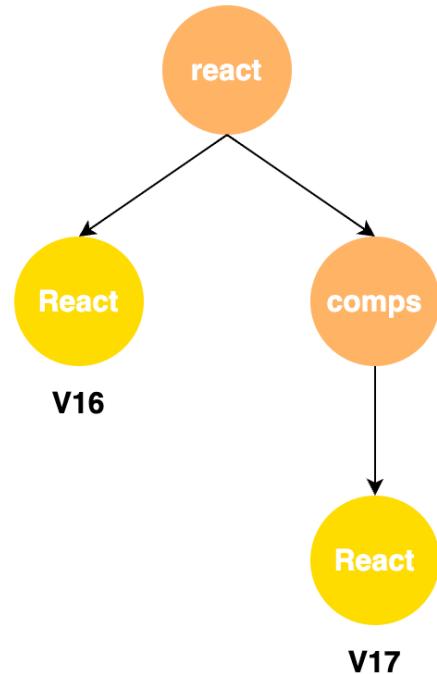


## 支持自动修复

自动修复不符合规范的代码或配置

# 自研方案的实践-内置多种 Checker

- 统一依赖版本? DependencyVersionChecker: 检查各项目的 package.json 下相同依赖版本是否一致
- 禁止 App 相互依赖? CategoryRelationChecker: 检查项目间的依赖关系. 默认 apps 之间不能相互依赖
- 限制依赖关系? TagRelationChecker: 检查项目间对 tag 的依赖
- 项目/文件级的循环依赖检查? CycleDependencyChecker: 支持项目级和文件级两种模式的规范检查
- 检查 phantom dependencies 问题? ExternalDependencyChecker: 检查使用了未声明的依赖/项目中声明了未使用的依赖
- 正确配置 project reference? ProjectReferenceChecker: 检查依赖的项目, 是否正确设置 tsconfig project reference



# 自研方案的实践-自定义 Checker



```
class CustomChecker extends Checker {
    async check(ctx: Context): Promise<boolean> {
        console.log('check fail');
        // 获取配置
        console.log(ctx.config);
        // 获取依赖图
        console.log(ctx.projectGraph);
        return false;
    }
    async autofix(ctx: Context): Promise<boolean> {
        console.log('autofix success');
        return true;
    }
}
```

# |自研方案的实践-支持自动修复



> **monorepo check**

[ Monorepo ] Version: v2.21.4

[ Monorepo ] Building full project graph, this may take seconds...

[ Monorepo ] Checking whether workspaces satisfy following rules:

**tagRelationCheck**

**dependencyVersionCheck**

**externalDependencyCheck**

**categoryRelationCheck**

**cycleDependencyCheck**

**projectReferenceCheck**

**Checking Passed!**

| 自研方案的实践-业务各异，难以扩展

支持丰富的扩  
展能力



- 1 现代前端工程开发
- 2 Monorepo 简介
- 3 问题及其实践
- 4 整体落地情况
- 5 总结与展望

# 落地情况



25%  
代码仓库占比



47%  
构建月活占比



140000+  
BNPM 周下载量

- 1 现代前端工程开发
- 2 Monorepo 简介
- 3 问题及其实践
- 4 整体落地情况
- 5 总结与展望

性能

任务调度性能

更快速的安装和调度性能

缓存

缓存共享能力

更细粒度的缓存共享能力

依赖

依赖管理能力

更智能化的依赖管理能力

**TOP100**®

**100**  
TOP100 CASE STUDIES OF THE YEAR

**100** 主办方 **msup®**  
TOP100 CASE STUDIES OF THE YEAR



微信官方公众号：壹佰案例  
关注查看更多年度实践案例