

# [WIP]构建发布自动化

## 背景

### [技术需求]

目前 arcoS 组件库构建发布较繁琐不规范，版本升级需要自行输入，发包需要自行执行命令，且缺少组件 changelog。

通过引入版本管理工具，以及接入 ci/cd 来实现构建发布自动化。

## 名词解释

名词	解释	备注
版本规范	npm 包版本管理规范 SemVer	
changelog	记录 npm 包新版本新增 Feature，或修复 bug 等信息	
ci/cd	持续集成/持续部署，结合 codebase pipeline，自动化构建打包发布	

## 技术调研

方案	分析		适用场景	业界用户	其他
	优势	不足			
<a href="#">lerna-changelogs</a>	基于两个tag之间 commit记录生成 changelog	仅适用于github，需要依赖 github token，适合开源项目	适合 github 开源项目	nest、x-render、midway	
<a href="#">standard-version</a>	基于两个tag之间 commit记录生成 changelog	<ul style="list-style-type: none"><li>不能联动升级依赖模块版本</li><li>自定义 changelog 困难，仓库缺少维护</li></ul>	适合单项目	nuxt、vite	
<a href="#">changesets</a>	<ul style="list-style-type: none"><li>灵活选择 monorepo 中多个子应用的升级发布，联动升级依赖模块版本</li><li>交互式命令行，使用和理解方便</li></ul>	不能根据 commit 生成 changelog	公司内部使用案例较多：Jupiter、modern.js、eden	rrweb、react-router、plop、redux-saga、modern.js	参考： <a href="#">📖 Eden Monorepo 自动化发包方案</a>



结论：使用 changesets 作为版本管理工具

## changesets

Changesets 主要提供两个功能：

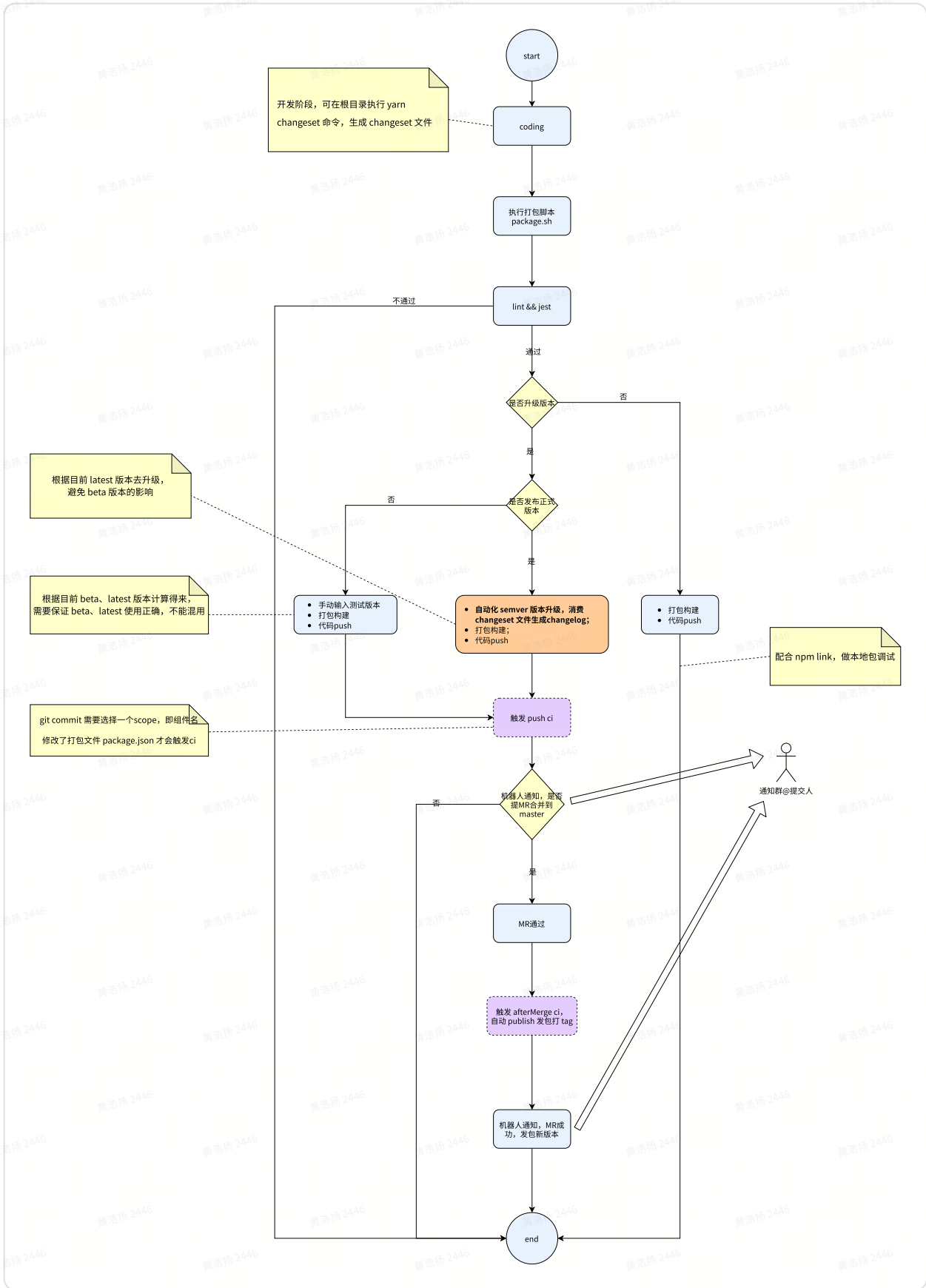
1. 按照版本规范升级子模块，同时扫描所有子模块依赖，联动升级依赖模块
2. 生成 changeset 文件，在升级时候消费 changeset 文件整合生成 changelog 文件

## 整体设计

### 时序图

[🔗 PlantUML \[该内容不支持导出查看\]](#)

### 流程图



# 详细设计

## 打包流程

详见 [\[时序图\]](#)，打包脚本如下

```
1  #!/bin/bash
2  set -e
3  echo "node 版本: $(node -v)"
4
5  # 当前目录名
6  CURRENT_DIR=$(pwd)
7
8  # 版本信息
9  PKG_NAME=$(node -p "require('./package.json').name")
10 PKG_ORI_VERSION=$(node -p "require('./package.json').version")
11 PKG_LATEST_VERSION=$(npm v ${PKG_NAME} version)
12 echo "PKG_NAME: ${PKG_NAME}, PKG_ORI_VERSION: ${PKG_ORI_VERSION},
13    PKG_LATEST_VERSION: ${PKG_LATEST_VERSION}"
14
15 echo "lint && jest"
16 yarn test
17
18 # 版本升级
19 read -p "是否需要升级版本? (请输入Y/N, 默认Y) " is_publish
20 is_publish=${is_publish:-Y} # 默认值
21 if [[ $is_publish == "Y" || $is_publish == "y" ]]
22 then
23     read -p "是否发布正式版本? (请输入Y/N, 默认Y) " is_release
24     is_release=${is_release:-Y} # 默认值
25     if [[ $is_release == "Y" || $is_release == "y" ]]
26     then
27         echo "发布正式版本"
28         npm version $PKG_LATEST_VERSION --allow-same-version
29
30         cd ..
31         yarn changeset add
32         yarn changeset version
33     else
34         read -p "发布测试版本, 测试版本号: " beta_version
35         npm version $beta_version
36     fi
37 else
38     echo "跳过升级版本"
39 fi
40
41 # 新版本
42 cd ${CURRENT_DIR}
43 PKG_NEW_VERSION=$(node -p "require('./package.json').version")
44 echo "组件版本: ${PKG_NAME}@${PKG_NEW_VERSION}"
45
46 # 打包部署
47 echo "打包部署"
48 BUILD_DIR="package"
49 rm -rf ${BUILD_DIR}
50 mkdir ${BUILD_DIR}
51 yarn build
```

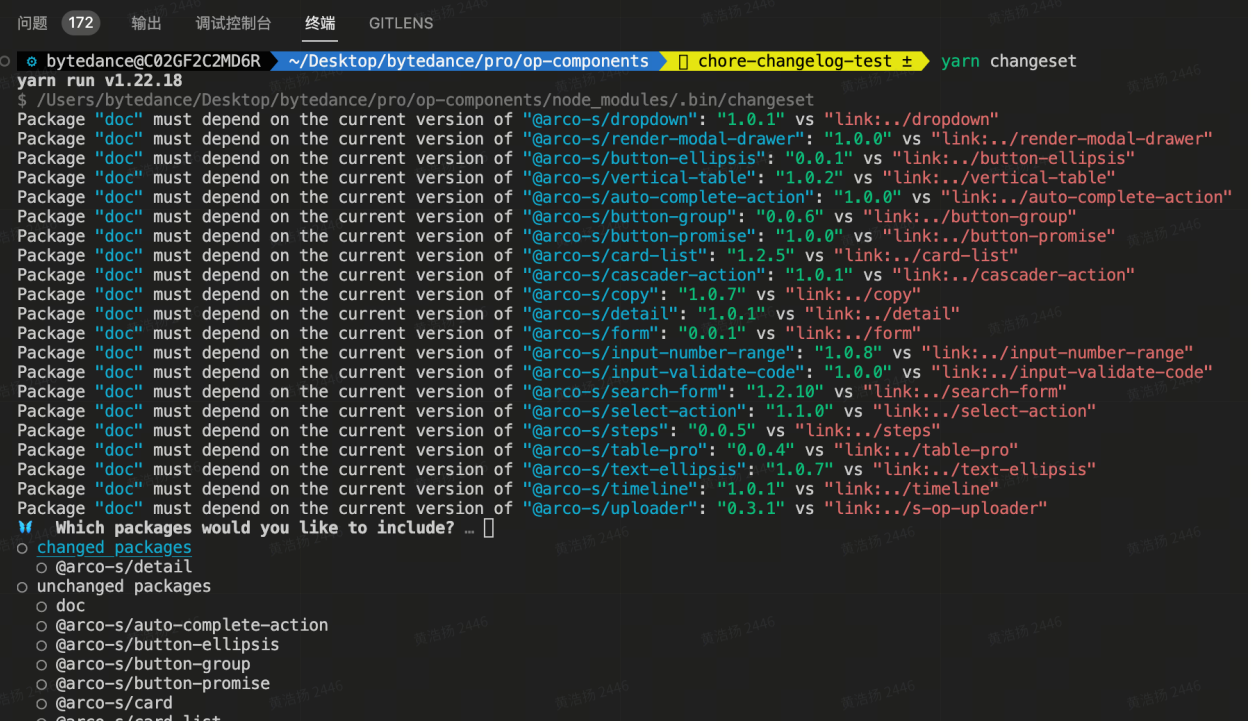
```
52 cp src/index.d.ts ${BUILD_DIR}
53 cp package.json ${BUILD_DIR}
54 cp CHANGELOG.md ${BUILD_DIR}
55 # cp README.md ${BUILD_DIR}
56
57 read -p "是否需要打包文档? (请输入Y/N, 默认N) " is_doc
58 is_doc=${is_doc:-N} # 默认值
59 if [[ $is_doc == "Y" || $is_doc == "y" ]]
60 then
61     # 部署文档
62     echo "部署文档"
63     cd ../_doc_
64     yarn build-doc -n $CURRENT_DIR -v $PKG_VERSION
65 else
66     echo "跳过文档部署"
67 fi
68
69 echo "git push"
70 cd ..
71 git add . && yarn cz && git push
```

## 版本管理和 changelog

通过 changesets 进行版本管理和生成 changelog。changesets 配置相关文件在 `/.changeset` 目录

使用流程：

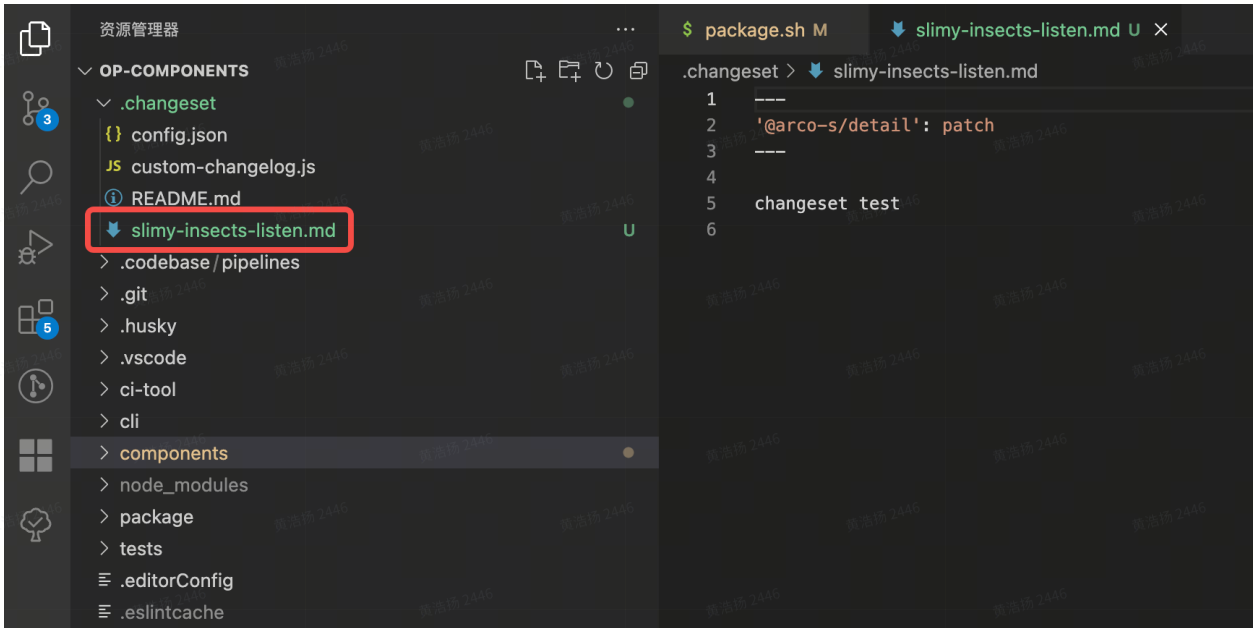
1. 执行 `yarn changeset`，会有交互式命令行提示，选择对应的子模块：



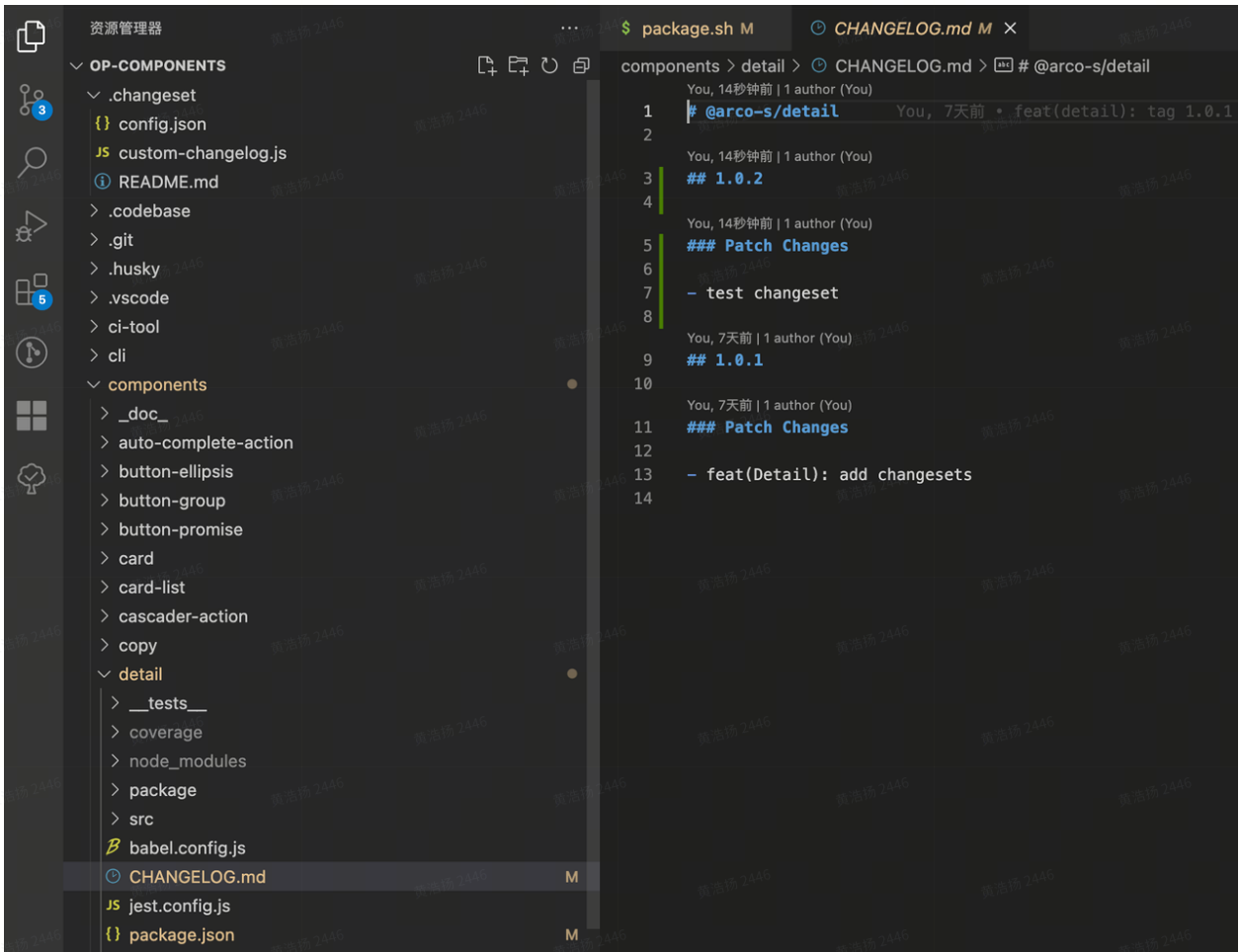
2. 选择升级类型（major/minor/patch），执行完成会生成 changeset md 文件，记录此次操作信息

```
问题 172 输出 调试控制台 终端 GITLENS

Package "doc" must depend on the current version of "@arco-s/copy": "1.0.7" vs "link:../copy"
Package "doc" must depend on the current version of "@arco-s/detail": "1.0.1" vs "link:../detail"
Package "doc" must depend on the current version of "@arco-s/form": "0.0.1" vs "link:../form"
Package "doc" must depend on the current version of "@arco-s/input-number-range": "1.0.8" vs "link:../input-number-range"
Package "doc" must depend on the current version of "@arco-s/input-validate-code": "1.0.0" vs "link:../input-validate-code"
Package "doc" must depend on the current version of "@arco-s/search-form": "1.2.10" vs "link:../search-form"
Package "doc" must depend on the current version of "@arco-s/select-action": "1.1.0" vs "link:../select-action"
Package "doc" must depend on the current version of "@arco-s/steps": "0.0.5" vs "link:../steps"
Package "doc" must depend on the current version of "@arco-s/table-pro": "0.0.4" vs "link:../table-pro"
Package "doc" must depend on the current version of "@arco-s/text-ellipsis": "1.0.7" vs "link:../text-ellipsis"
Which packages would you like to include? · @arco-s/detail
Which packages should have a major bump? · No items were selected
Which packages should have a minor bump? · No items were selected
The following packages will be patch bumped:
@arco-s/detail@1.0.1
Please enter a summary for this change (this will be in the changelogs).
(submit empty line to open external editor)
Summary · test changeset
=== Summary of changesets ===
patch: @arco-s/detail
Note: All dependents of these packages that will be incompatible with
the new version will be patch bumped when this changeset is applied.
Is this your desired changeset? (Y/n) · true
Changeset added! - you can now commit it
If you want to modify or expand on the changeset summary, you can find it here
info /Users/bytedance/Desktop/bytedance/pro/op-components/.changeset/moody-baboons-warn.md
Done in 46.67s.
bytedance@C02GF2C2MD6R ~/Desktop/bytedance/pro/op-components [ chore-changelog-test ]
```



3. 执行 `yarn changeset version`，会扫描所有 changeset 文件，合并生成 changelog，并升级版本。



## 接入自动化 ci/cd

为了提效打包构建流程，接入 ci/cd 自动实现提 mr 合并 master，自动发包打 tag，并通知 lark 群。

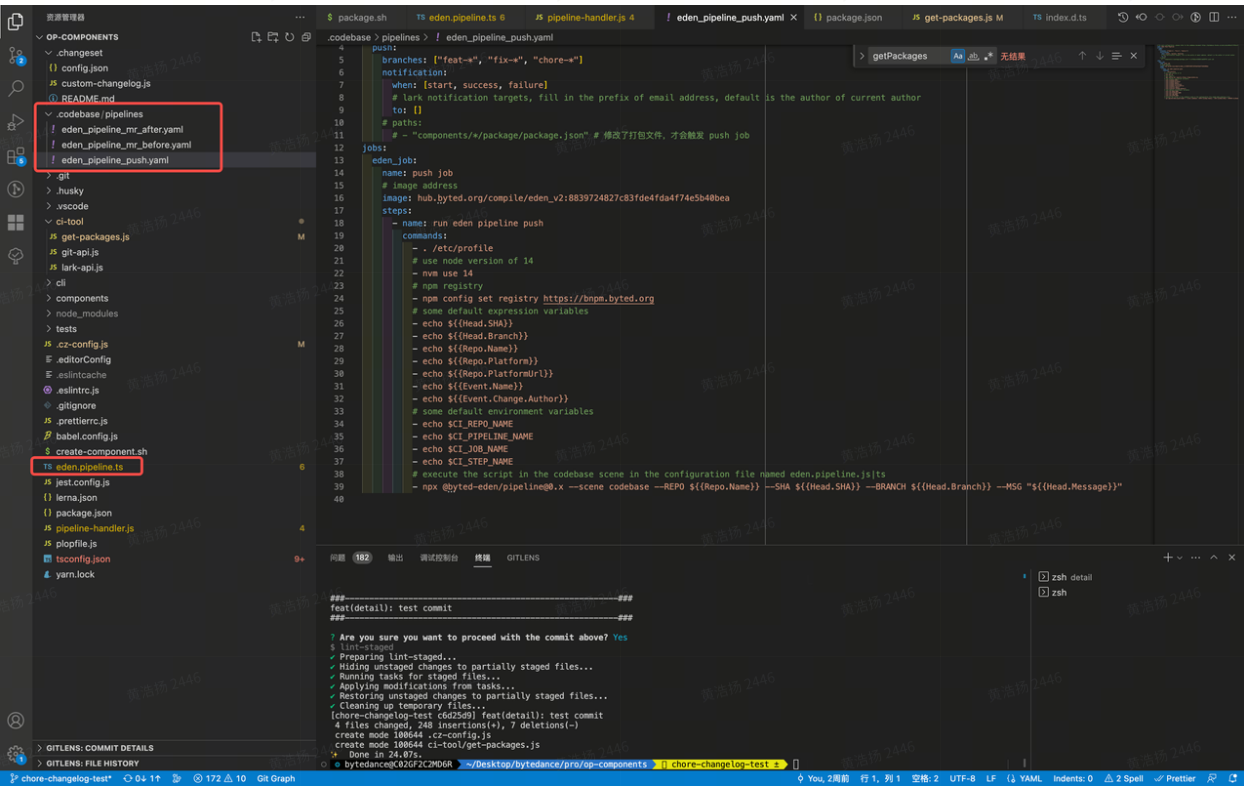


Eden ci

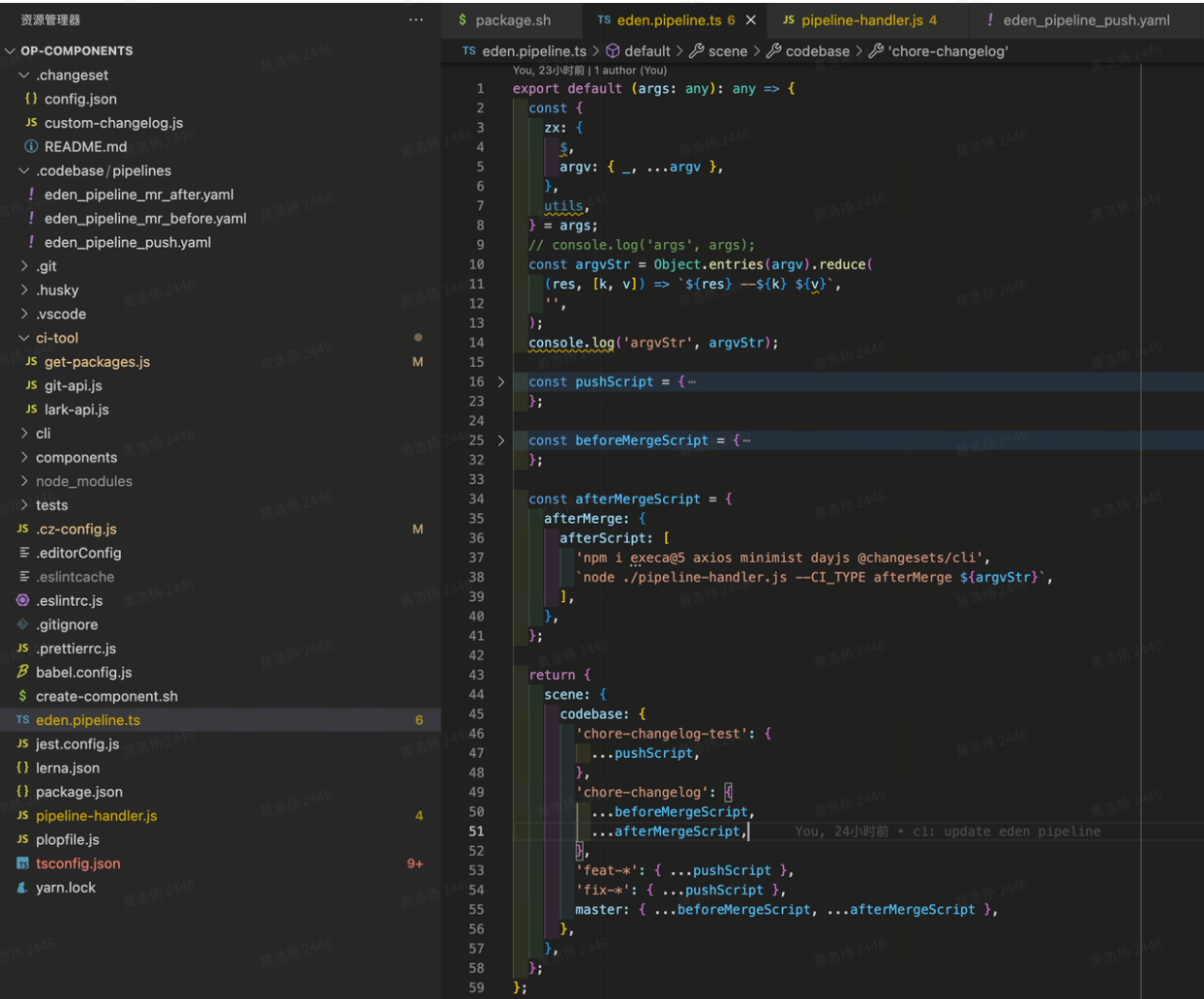
使用的 ci 是公司的 eden ci，主要考虑其：配置和使用简单；整个项目组还在频繁迭代维护；对 codebase ci 满足所求，甚至还提供了 scm 和 tce 相关的集成功能。

使用流程：

1. 首先 codebase 的 ci yaml 文件放置在 `/.codebase` 目录，配置语法与传统的 github、gitlab一致。主要提供了 push 和 merge 触发时机，涵盖了代码提交所有流程。eden ci 比较好用的一点是，触发 yaml 后，会执行 `eden.pipeline.js`，将控制权转移到 js 脚本，能够做很多事情。



2. `pipeline.config.js` 返回一个函数，从传参可以拿到 zx 运行环境上下文信息，以及提供 scm、tce继承用的 api。例如 yaml 最终调用 pipeline 脚本，命令行参数可以从 `zx.argv` 获取



commitizen

配置

 黄浩扬 2023年3月3日

[CI Configuration Syntax 配置语法](#)

通过 `commitizen` 限制 commit scope，scope 通过扫描组件获得，即 `packageNames`

```
1  const path = require('path');
2  const globby = require('globby');
3  const loadJsonFile = require('load-json-file');
4
5  // 获取所有子模块 package.json
6  const getPackages = () => {
7    const allPackageJson = globby.sync(
8      `${path.resolve(__dirname, '..')}/components/*/package.json`,
9    );
10   return allPackageJson.map(loadJsonFile.sync);
11 };
12
13 const allPackages = getPackages();
14
15 const packageNames = allPackages.map(v => v.name.replace(/@arco-
16 s\\/, ''));
17
18 module.exports = {
19   allPackages,
20   packageNames,
21 };

```

## 机器人后台服务

自动提 MR 的操作由 [飞书机器人](#) 完成，需要有一个后台服务处理操作。使用了 goofy stack，方便搭建 NodeJS server --

<https://stack.bytedance.net/team/20816/app/64005530d740d80074312d6f/224567>

## 开发者使用流程

!

注意：升级版本时获取当前包的最新版本，是通过 `npm v xxx version` 获取 `latest` 版本，如果误发测试版本为 `latest`，需要执行更改命令：

`npm dist-tag add xxx@a.b.c latest`

`npm dist-tag add xxx@a.b.c-beta.x beta`

自动化完成后，组件开发者只需要在对应组件目录内，执行打包脚本 `package.sh`，就会自动做代码检测和版本升级和构建。触发 `push ci` 后，会收到机器人通知：**是否提 MR到master**。提交和通过MR后，会自动进行发包并通知。

## 后续

- 版本管理、changelog 生成、ci 自动发包、机器人通知这一流程，整体可以适用在大部分工具类项目，看有没抽成脚手架的价值。
- 自定义 changelog



黄浩扬 2023年3月3日  
<https://eden.bytedance.net/tols/pipeline/scene/codebase>





