[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			
lerna-changelog	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	
standard-version	基于两个tag之间commit记录生成changelog	• 不能联动升级依赖模块版本 • 自定义 changelog 困难，仓库缺少维护	适合单项目	nuxt、vite	
changesets	• 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 • 交互式命令行，使用和理解方便	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：Eden Monorepo 自动化发包方案

> 💡 **结论：使用 changesets 作为版本管理工具**

changesets

Changesets 主要提供两个功能：

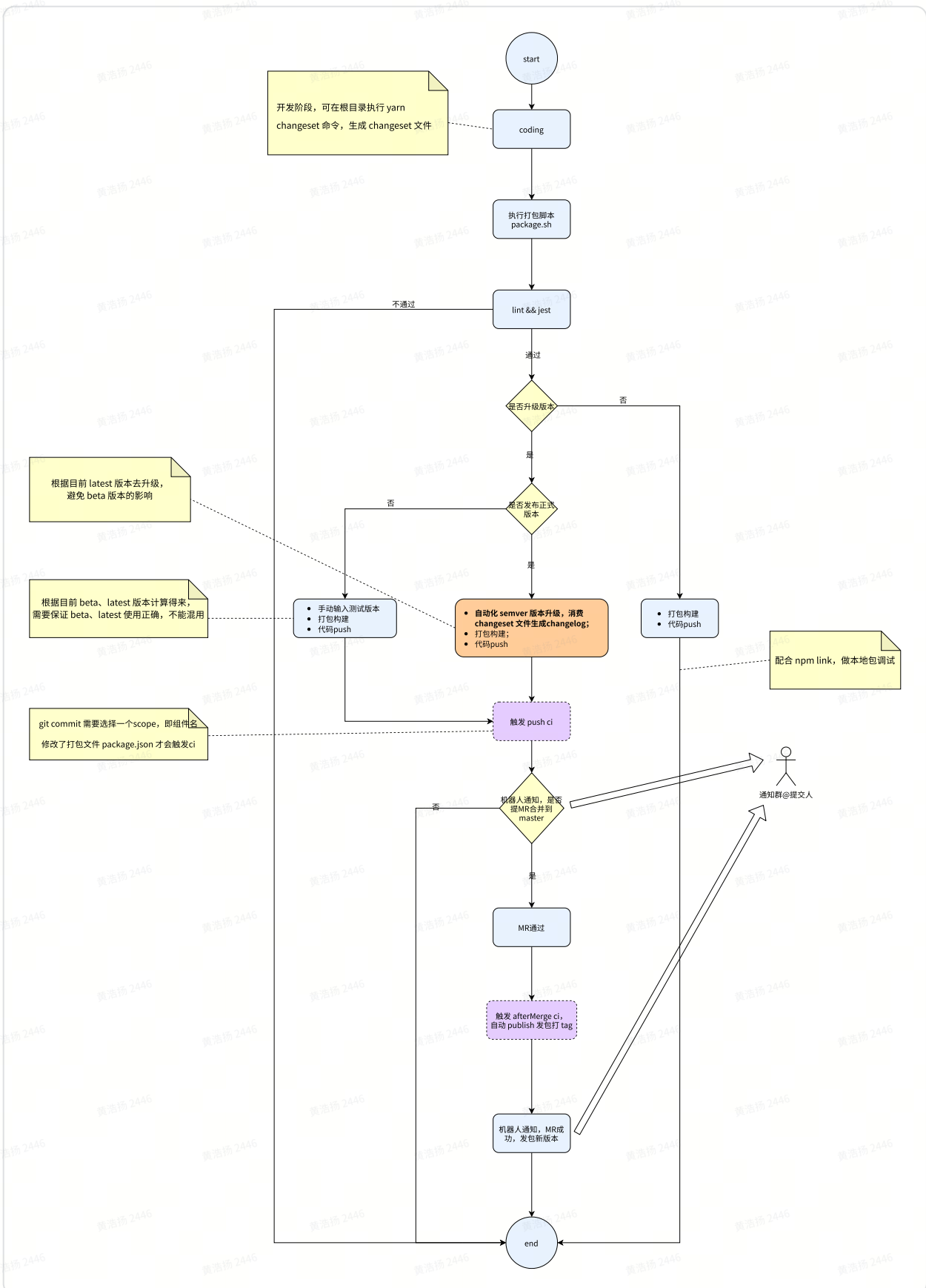
1. 按照版本规范升级子模块，同时扫描所有子模块依赖，联动升级依赖模块
2. 生成 changeset 文件，在升级时候消费 changeset 文件整合生成 changelog 文件

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

```
#!/bin/bash
set -e
echo "node 版本: $(node -v)"

# 当前目录名
CURRENT_DIR=$(pwd)

# 版本信息
PKG_NAME=$(node -p "require('./package.json').name")
PKG_ORI_VERSION=$(node -p "require('./package.json').version")
PKG_LATEST_VERSION=$(npm v ${PKG_NAME} version)
echo "PKG_NAME: ${PKG_NAME}, PKG_ORI_VERSION: ${PKG_ORI_VERSION},
PKG_LATEST_VERSION: ${PKG_LATEST_VERSION}"

echo "lint && jest"
yarn test

# 版本升级
read -p "是否需要升级版本？（请输入Y/N，默认Y）" is_publish
is_publish=${is_publish:-Y} # 默认值
if [[ $is_publish == "Y" || $is_publish == "y" ]]
then
  read -p "是否发布正式版本？（请输入Y/N，默认Y）" is_release
  is_release=${is_release:-Y} # 默认值
  if [[ $is_release == "Y" || $is_release == "y" ]]
  then
    echo "发布正式版本"
    npm version $PKG_LATEST_VERSION --allow-same-version

    cd ..
    yarn changeset add
    yarn changeset version
  else
    read -p "发布测试版本，测试版本号：" beta_version
    npm version $beta_version
  fi
else
  echo "跳过升级版本"
fi

# 新版本
cd ${CURRENT_DIR}
PKG_NEW_VERSION=$(node -p "require('./package.json').version")
echo "组件版本：${PKG_NAME}@${PKG_NEW_VERSION}"

# 打包部署
echo "打包部署"
BUILD_DIR="package"
rm -rf ${BUILD_DIR}
mkdir ${BUILD_DIR}
yarn build

```

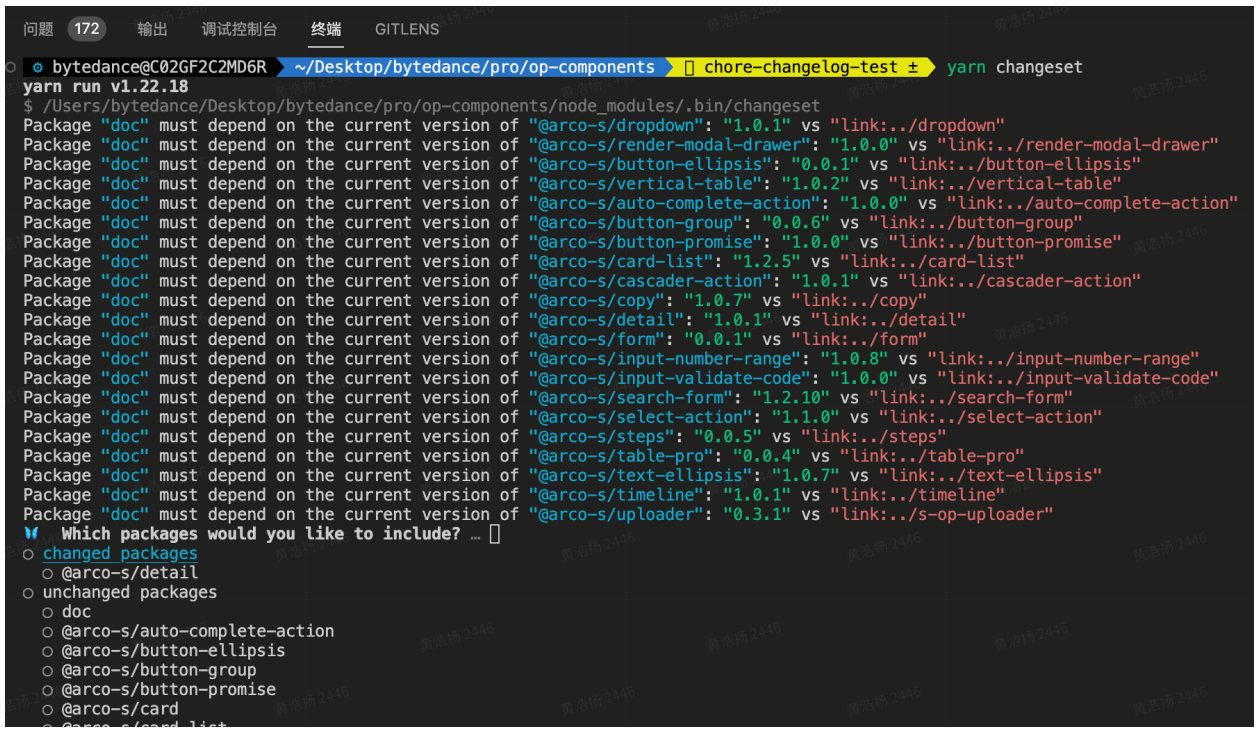
```
cp src/index.d.ts ${BUILD_DIR}
cp package.json ${BUILD_DIR}
cp CHANGELOG.md ${BUILD_DIR}
# cp README.md ${BUILD_DIR}

read -p "是否需要打包文档？（请输入Y/N，默认N）" is_doc
is_doc=${is_doc:-N} # 默认值
if [[ $is_doc == "Y" || $is_doc == "y" ]]
then
  # 部署文档
  echo "部署文档"
  cd ../_doc_
  yarn build-doc -n $CURRENT_DIR -v $PKG_VERSION
else
  echo "跳过文档部署"
fi

echo "git push"
cd ..
git add . && yarn cz && git push
```

版本管理和 changelog

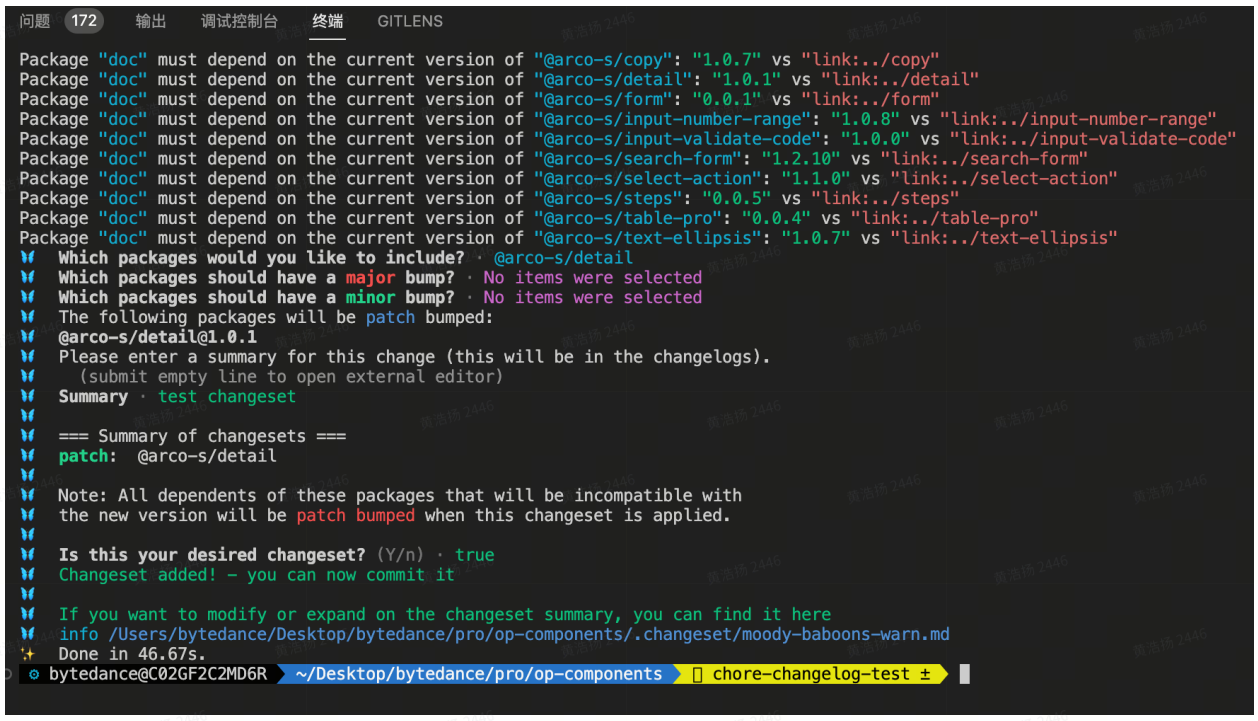
通过 changesets 进行版本管理和生成 changelog。changesets 配置相关文件在 `/.changeset` 目录

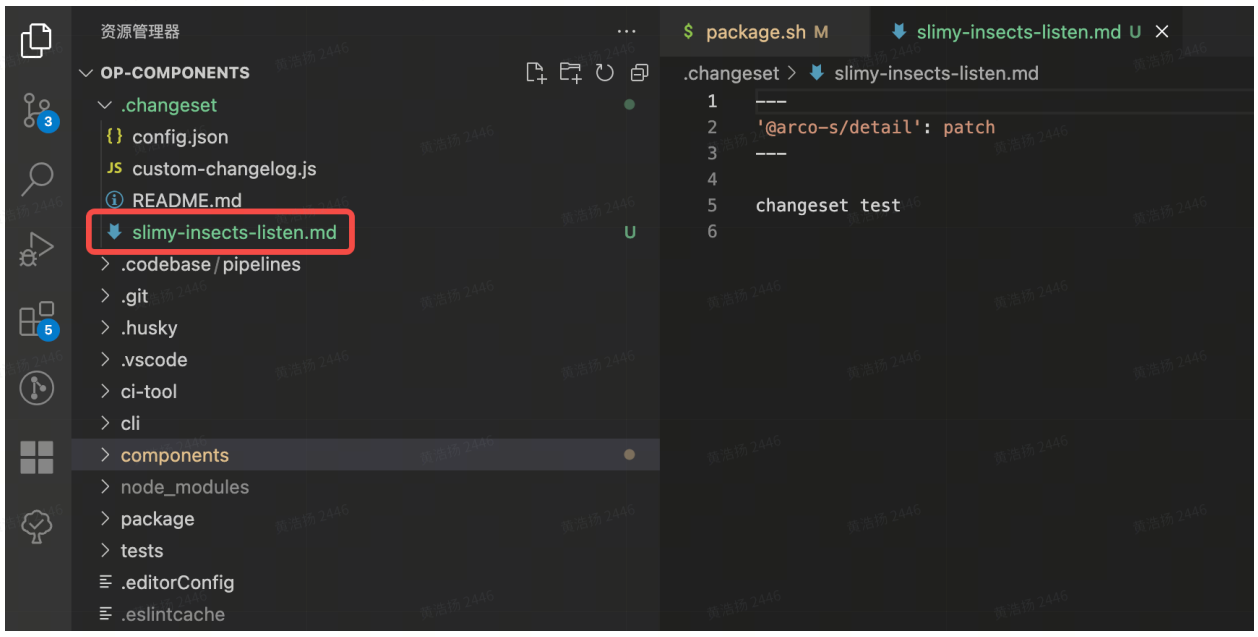
使用流程：

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

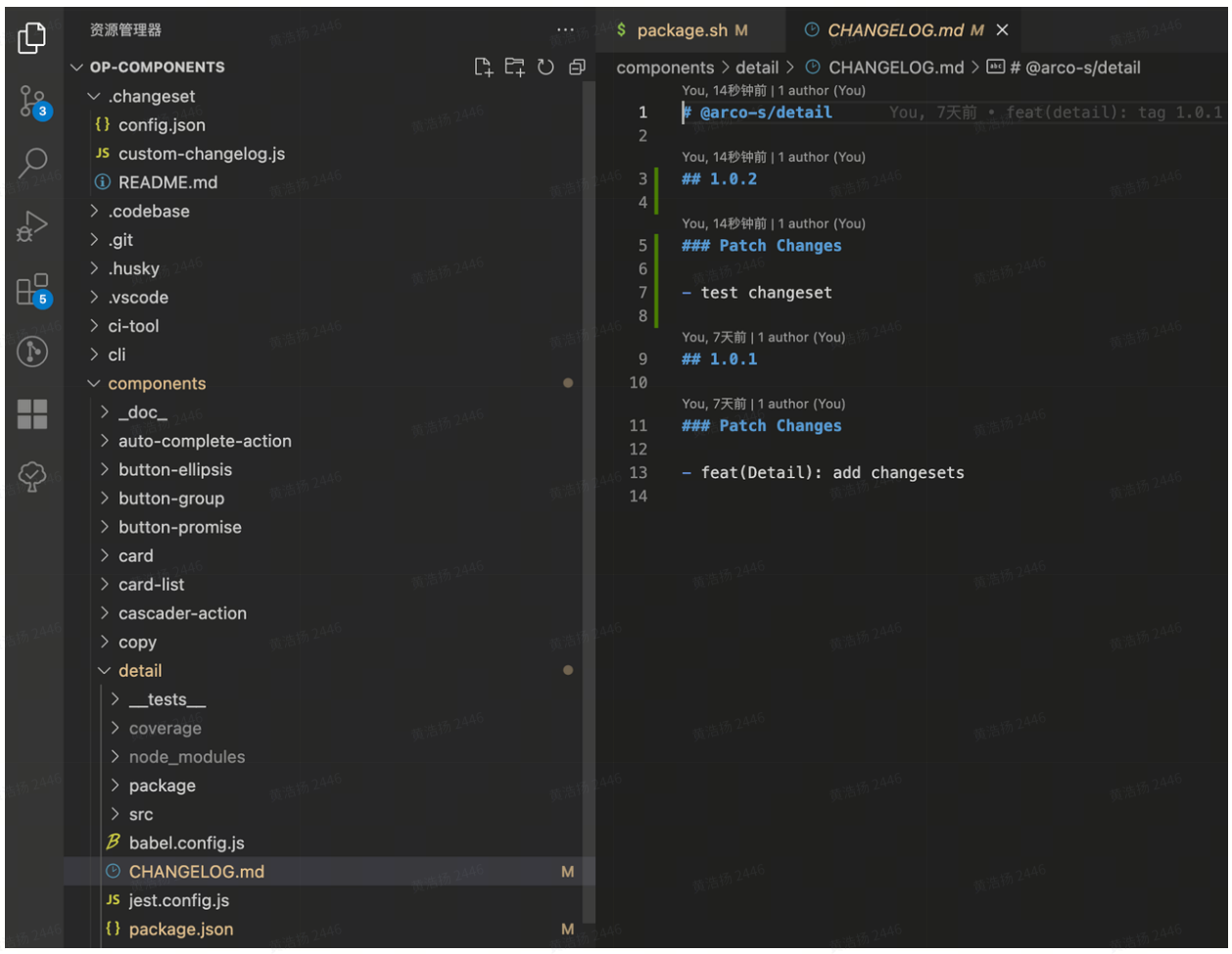


2. 选择升级类型（major/minor/patch），执行完成会生成 changeset md 文件，记录此次操作信息





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

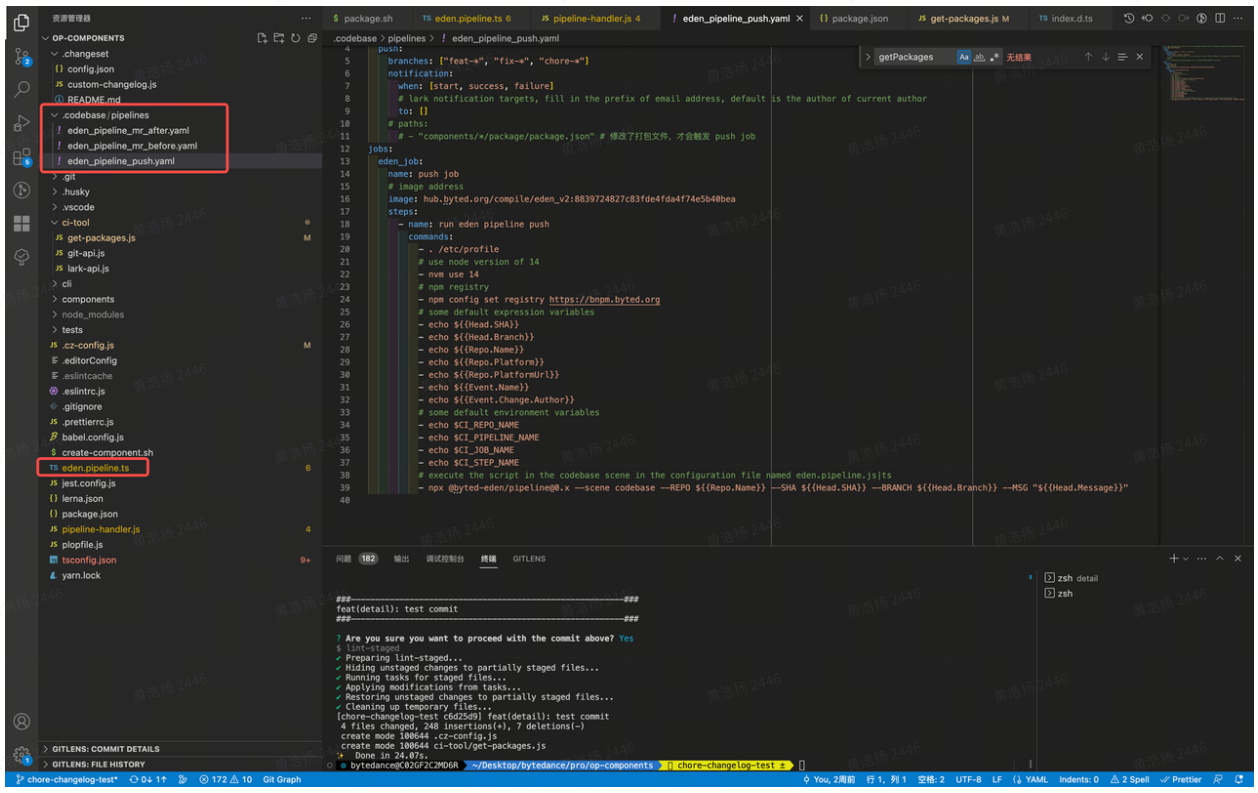
为了提效打包构建流程，接入 ci/cd 自动实现提 mr 合并 master，自动发包打 tag，并通知 lark 群。

Eden ci

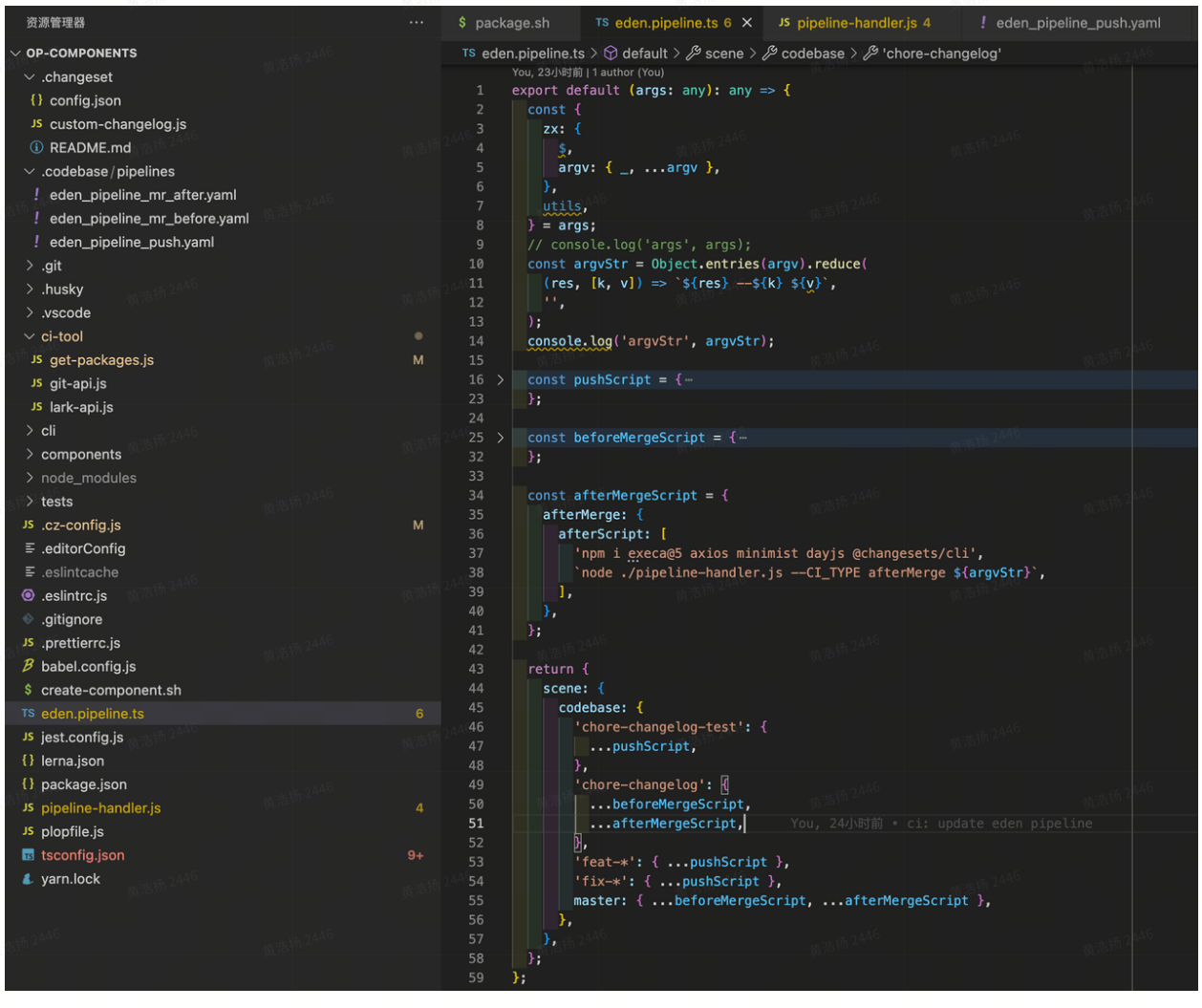
使用的 ci 是公司的 eden ci，主要考虑其：配置和使用简单；整个项目组还在频繁迭代维护；对 codebase ci 满足所求，甚至还提供了 scm 和 tce 相关的集成功能。

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

pipeline.config.js

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

```
const path = require('path');
const globby = require('globby');
const loadJsonFile = require('load-json-file');

// 获取所有子模块 package.json
const getPackages = () => {
  const allPackageJson = globby.sync(
    `${path.resolve(__dirname, '../')}/components/*/package.json`,
  );
  return allPackageJson.map(loadJsonFile.sync);
};

const allPackages = getPackages();

const packageNames = allPackages.map(v => v.name.replace(/@arco-s\//, ''));

module.exports = {
  allPackages,
  packageNames,
};
```

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

1. 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。
2. 自定义 changelog

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