

# 目录

## CONTENTS

- 01 | 基础篇：webpack 与构建发展简史
- 02 | 基础篇：webpack 基础用法
- 03 | 基础篇：webpack 进阶用法
- 04 | 进阶篇：编写可维护的 webpack 构建配置
- 05 | 进阶篇：webpack 构建速度和体积优化策略
- 06 | 原理篇：通过源码掌握 webpack 打包原理
- 07 | 原理篇：编写 Loader 和插件
- 08 | 实战篇：React 全家桶 和 webpack 开发商城项目



扫码试看/订阅  
《玩转webpack》

# 构建配置抽离成 npm 包的意义

## 通用性

- 业务开发者无需关注构建配置
- 统一团队构建脚本

## 可维护性

- 构建配置合理的拆分
- README 文档、ChangeLog 文档等

## 质量

- 冒烟测试、单元测试、测试覆盖率
- 持续集成

# 构建配置管理的可选方案

通过多个配置文件管理不同环境的构建，webpack `--config` 参数进行控制

将构建配置设计成一个库，比如：hjs-webpack、Neutrino、webpack-blocks

抽成一个工具进行管理，比如：create-react-app, kyt, nwb

将所有的配置放在一个文件，通过 `--env` 参数控制分支选择

# 构建配置包设计

通过多个配置文件管理不同环境的 webpack 配置

- 基础配置: webpack.base.js
- 开发环境: webpack.dev.js
- 生产环境: webpack.prod.js
- SSR环境: webpack.ssr.js
- .....

抽离成一个 npm 包统一管理

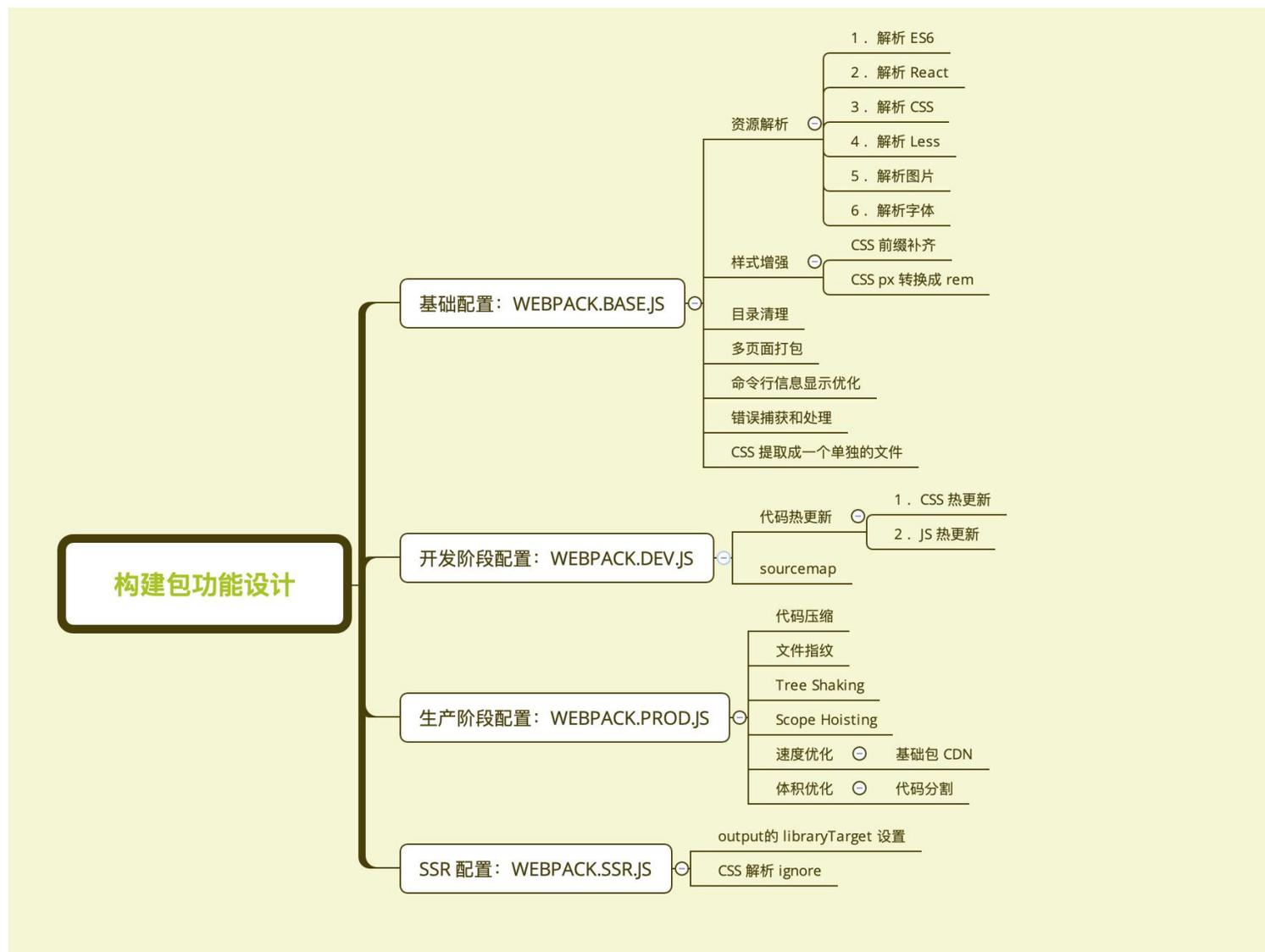
- 规范: Git commit日志、README、ESLint 规范、Semver 规范
- 质量: 冒烟测试、单元测试、测试覆盖率和 CI

# 通过 webpack-merge 组合配置

```
> merge = require("webpack-merge")
...
> merge(
... { a: [1], b: 5, c: 20 },
... { a: [2], b: 10, d: 421 }
... )
{ a: [ 1, 2 ], b: 10, c: 20, d: 421 }
```

合并配置：`module.exports = merge(baseConfig, devConfig);`

# 功能模块设计



# 目录结构设计

lib 放置源代码

test 放置测试代码

```
+ |- /test
+ |- /lib
+   |- webpack.dev.js
+   |- webpack.prod.js
+   |- webpack.ssr.js
+   |- webpack.base.js
+ |- README.md
+ |- CHANGELOG.md
+ |- .eslinrc.js
+ |- package.json
+ |- index.js
```

# 使用 ESLint 规范构建脚本

使用 `eslint-config-airbnb-base`

`eslint --fix` 可以自动处理空格

```
module.exports = {
  "parser": "babel-eslint",
  "extends": "airbnb-base",
  "env": {
    "browser": true,
    "node": true
  }
};
```

# 冒烟测试 (smoke testing)

冒烟测试是指对提交测试的软件在进行详细深入的测试之前而进行的预测试，这种预测试的主要目的是暴露导致软件需重新发布的基本功能失效等严重问题。

# 冒烟测试执行

构建是否成功

每次构建完成 build 目录是否有内容输出

- 是否有 JS、CSS 等静态资源文件
- 是否有 HTML 文件

# 判断构建是否成功

在示例项目里面运行构建，看看是否有报错

```
const path = require('path');
const webpack = require('webpack');
const rimraf = require('rimraf');
const Mocha = require('mocha');

const mocha = new Mocha({
  timeout: '10000ms',
});
process.chdir(__dirname);

rimraf('./dist', () => {
  const prodConfig = require('../lib/webpack.prod');
  webpack(prodConfig, (err, stats) => {
    if (err) {
      console.error(err);
      return;
    }

    console.log(stats.toString({
      colors: true,
      modules: false,
      children: false,
      chunks: false,
      chunkModules: false
    }));

    console.log('\n' + 'Compiler success, begin mocha test');
  })
});
```

# 判断基本功能是否正常

## 编写 mocha 测试用例

- 是否有 JS、CSS 等静态资源文件
- 是否有 HTML 文件

```
const glob = require('glob-all');

describe('checking generated file exists', function() {
  it('should generate html files', function(done) {
    const files = glob.sync(
      [
        './dist/index.html',
        './dist/search.html'
      ]
    );
    if (files.length > 0) {
      done();
    } else {
      throw new Error("No html files found");
    }
  });
  it('should generate js & css files', function(done) {
    const files = glob.sync(
      [
        './dist/index_*.js',
        './dist/search_*.js',
        './dist/index_*.css',
        './dist/search_*.css',
      ]
    );
    if (files.length > 0) {
      done();
    } else {
      throw new Error("No files found");
    }
  });
});
```

# 单元测试与测试覆盖率



单纯的测试框架，需要断言库

- chai
- should.js
- expect
- better-assert



集成框架，开箱即用



极简 API

# 编写单元测试用例



技术选型: Mocha + Chai

测试代码: describe, it, expect

测试命令: mocha add.test.js

```
add.test.js
```

```
const expect = require('chai').expect;
```

```
const add = require('../src/add');
```

```
describe('use expect: src/add.js', () => {  
  it('add(1, 2) === 3', () => {  
    expect(add(1, 2).to.equal(3));  
  });  
});
```

# 单元测试接入

## 1. 安装 mocha + chai

```
npm i mocha chai -D
```

## 2. 新建 test 目录，并增加 xxx.test.js 测试文件

## 3. 在 package.json 中的 scripts 字段增加 test 命令

```
"scripts": {  
  "test": "node_modules/mocha/bin/_mocha"  
},
```

## 4. 执行测试命令

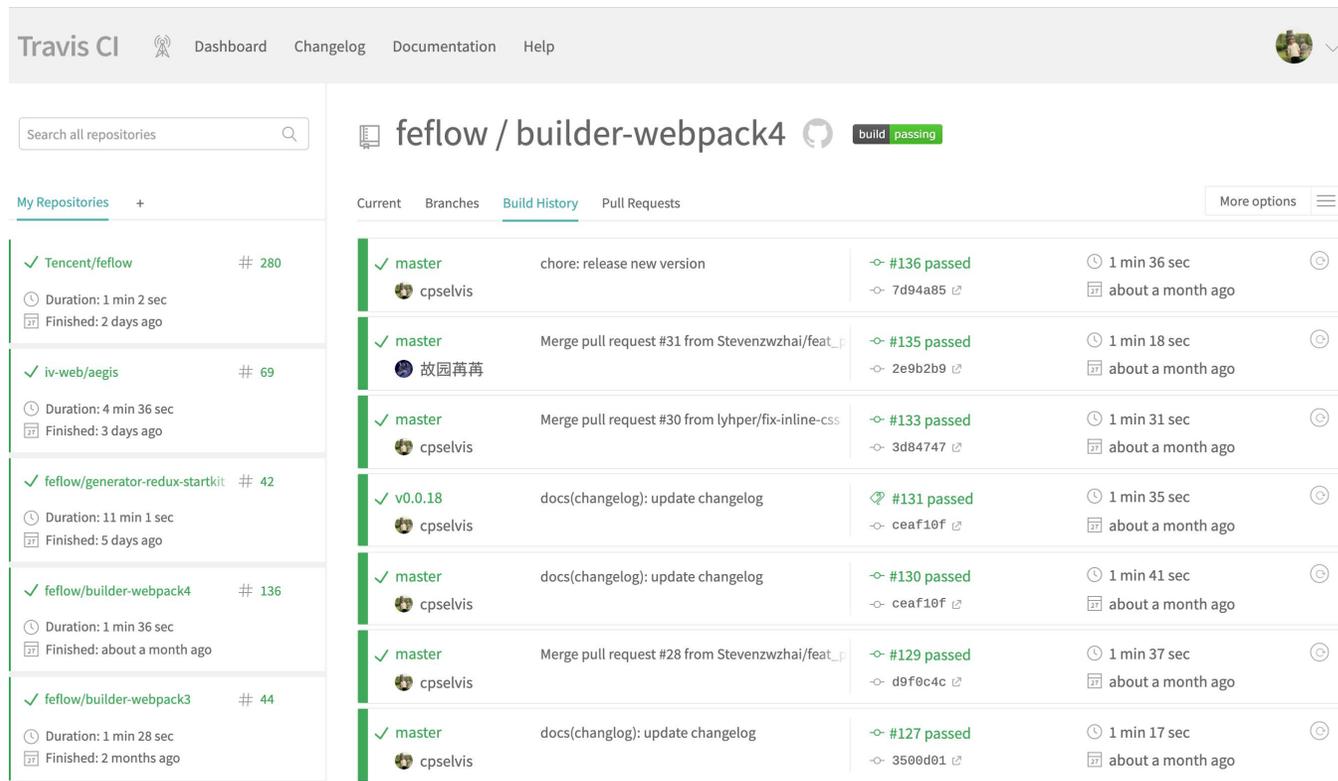
```
npm run test
```

# 持续集成的作用

## 优点:

- 快速发现错误
- 防止分支大幅偏离主干

核心措施是，代码集成到主干之前，必须通过自动化测试。只要有一个测试用例失败，就不能集成。



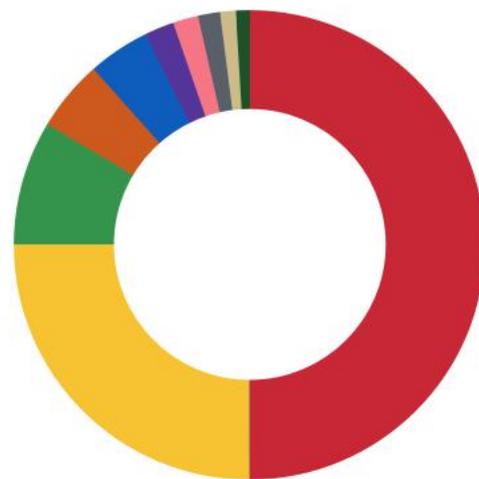
The screenshot displays the Travis CI interface for the repository 'feflow / builder-webpack4'. The top navigation bar includes 'Travis CI', 'Dashboard', 'Changelog', 'Documentation', and 'Help'. A search bar is present for repositories. The main content area shows a list of recent builds under the 'Build History' tab. Each build entry includes a status (e.g., 'passed'), a commit hash, a duration, and a completion time. The builds are ordered chronologically, with the most recent at the top.

Repository	Branch	Commit	Status	Duration	Finished
Tencent/feflow	master	7d94a85	#136 passed	1 min 36 sec	2 days ago
iv-web/aegis	master	2e9b2b9	#135 passed	4 min 36 sec	3 days ago
feflow/generator-redux-startkit	master	3d84747	#133 passed	11 min 1 sec	5 days ago
feflow/builder-webpack4	master	ceaf10f	#130 passed	1 min 36 sec	about a month ago
feflow/builder-webpack3	master	3500d01	#127 passed	1 min 28 sec	2 months ago

# Github 最流行的 CI

## Top 10 CI systems used with GitHub.com

(based on most used commit status contexts)



# 接入 Travis CI

1. <https://travis-ci.org/> 使用 GitHub 账号登录
2. 在 <https://travis-ci.org/account/repositories> 为项目开启
3. 项目根目录下新增 `.travis.yml`

# travis.yml 文件内容

install 安装项目依赖

script 运行测试用例

```
language: node_js

sudo: false

cache:
  apt: true
  directories:
    - node_modules

node_js: stable #设置相应的版本

install:
  - npm install -D #安装构建器依赖
  - cd ./test/template-project
  - npm install -D #安装模板项目依赖

script:
  - npm test
```

# 发布到 npm

添加用户: `npm adduser`

## 升级版本

升级补丁版本号: `npm version patch`

升级小版本号: `npm version minor`

升级大版本号: `npm version major`

发布版本: `npm publish`

# Git 规范和 Changelog 生成

良好的 Git commit 规范优势：

- 加快 Code Review 的流程
- 根据 Git Commit 的元数据生成 Changelog
- 后续维护者可以知道 Feature 被修改的原因

## 技术方案



1 Git提交格式

- ▶ 统一团队 Git commit 日志标准，便于后续代码 review 和版本发布
  - 使用 angular 的 Git commit 日志作为基本规范
    - 提交类型限制为：feat, fix, docs, style, refactor, perf, test, chore, revert 等
    - 提交信息分为两部分，标题（首字母不大写，末尾不要标点）、主体内容（正常的描述信息即可）
  - 日志提交时友好的类型选择提示
    - 使用 commitize 工具
  - 不符合要求格式的日志拒绝提交的保障机制
    - 使用 validate-commit-msg 工具
    - 需要同时在客户端、gitlab server hook 做
  - 统一 changelog 文档信息生成
    - 使用 conventional-changelog-cli 工具

# 提交格式要求

```
<type>(<scope>): <subject>  
<BLANK LINE>  
<body>  
<BLANK LINE>  
<footer>
```

对格式的说明如下：

- type代表某次提交的类型，比如是修复一个bug还是增加一个新的feature。所有的type类型如下：
- feat：新增feature
- fix: 修复bug
- docs: 仅仅修改了文档，比如README, CHANGELOG, CONTRIBUTE等等
- style: 仅仅修改了空格、格式缩进、都好等等，不改变代码逻辑
- refactor: 代码重构，没有加新功能或者修复bug
- perf: 优化相关，比如提升性能、体验
- test: 测试用例，包括单元测试、集成测试等
- chore: 改变构建流程、或者增加依赖库、工具等
- revert: 回滚到上一个版本

# 本地开发阶段增加 precommit 钩子

安装 husky

```
npm install husky --save-dev
```

通过 commitmsg 钩子校验信息

```
"scripts": {  
  "commitmsg": "validate-commit-msg",  
  "changelog": "conventional-changelog -p  
angular -i CHANGELOG.md -s -r 0"  
},  
"devDependencies": {  
  "validate-commit-msg": "^2.11.1",  
  "conventional-changelog-cli": "^1.2.0",  
  "husky": "^0.13.1"  
}
```

# Changelog 生成

Commits on Sep 10, 2017

 docs: add plugin usage section  
cpselvis committed on GitHub 2 days ago ✓

 docs: update init project step.  
cpselvis committed on GitHub 2 days ago ✓

 style: pkg version  
cpselvis committed 2 days ago ✓

 fix: fix deps  
cpselvis committed 2 days ago ✓

 docs: changelog update  
cpselvis committed 2 days ago ✓

 fix: npm registry and proxy can be set when initialized.  
cpselvis committed 2 days ago ✓

 feat: init client before load external plugins  
cpselvis committed 2 days ago ✓

Commits on Sep 9, 2017

 feat: use yaml to store local config  
cpselvis committed 3 days ago ✗

 feat: check node engine requirements  
cpselvis committed 3 days ago ✓



## 0.10.1 (2017-09-10)

### Bug Fixes

- npm registry and proxy can be set when initialized. ([a9a8119](#))
- support yeoman generator app/index.js folder ([bedfaa9](#))

### Features

- check node engine requirements ([499358d](#))
- init client before load external plugins ([2982f3d](#))
- print banner when no args ([3fff39c](#))
- use yaml to store local config ([c7d3241](#))

## 0.10.0 (2017-08-21)

### Features

- add uninstall plugin command. ([84de19a](#))
- command register mechanism. ([febe1c3](#))
- Initial commit ([626def2](#))
- support -v and --version command. ([fa14d9e](#))

# 开源项目版本信息案例

软件的版本通常由三位组成，形如：  
X.Y.Z

版本是严格递增的，此处是：16.2.0 –  
> 16.3.0 –> 16.3.1

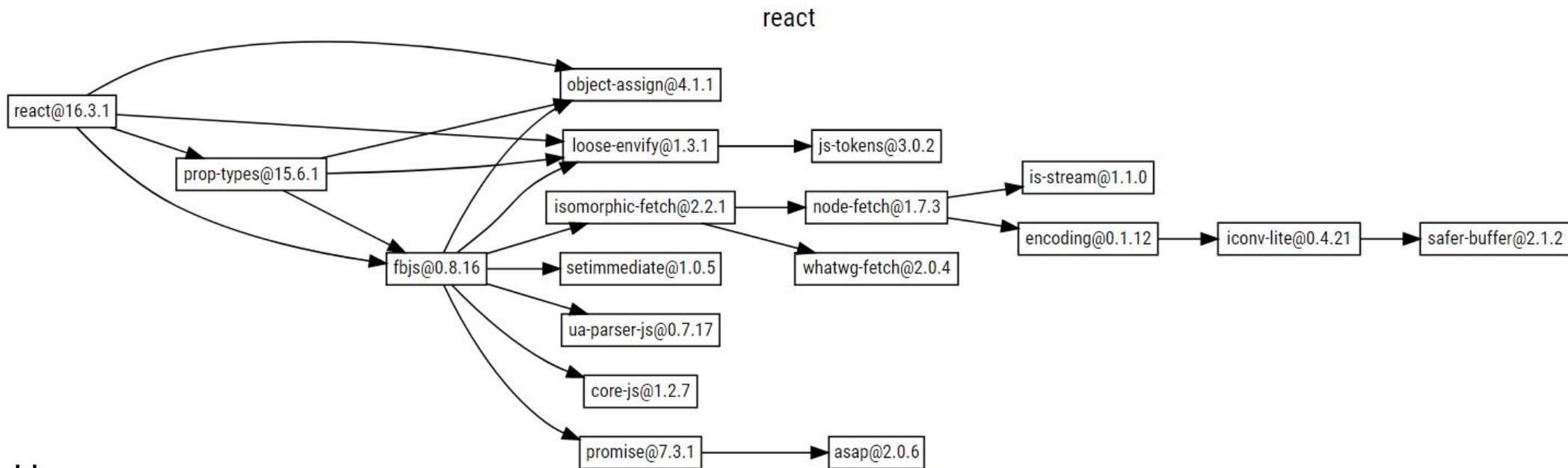
在发布重要版本时，可以发布alpha, rc  
等先行版本

alpha和rc等修饰版本的关键字后面可  
以带上次数和meta信息

16.4.0-alpha.3174632	a year ago
16.4.0-alpha.7926752	a year ago
16.3.0-alpha.1	a year ago
16.3.0-alpha.0	a year ago
16.2.0	2 years ago
16.1.1	2 years ago
16.1.0	2 years ago
16.1.0-rc	2 years ago
16.1.0-beta.1	2 years ago
16.1.0-beta	2 years ago
16.0.0	2 years ago
15.6.2	2 years ago
16.0.0-rc.3	2 years ago
16.0.0-rc.2	2 years ago
16.0.0-rc.1	2 years ago
16.0.0-beta.5	2 years ago
16.0.0-beta.4	2 years ago

React 版本信息

# 遵守 semver 规范的优势



优势：

- 避免出现循环依赖
- 依赖冲突减少

# 语义化版本 (Semantic Versioning) 规范格式 极客时间

主版本号：当你做了不兼容的 API 修改，

次版本号：当你做了向下兼容的功能性新增，

修订号：当你做了向下兼容的问题修正。

# 先行版本号

先行版本号可以作为发布正式版之前的版本，格式是在修订版本号后面加上一个连接号（-），再加上一连串以点（.）分割的标识符，标识符可以由英文、数字和连接号（[0-9A-Za-z-]）组成。

- alpha：是内部测试版，一般不向外部发布，会有很多 Bug。一般只有测试人员使用。
- beta：也是测试版，这个阶段的版本会一直加入新的功能。在 Alpha 版之后推出
- rc：Release Candidate) 系统平台上就是发行候选版本。RC 版不会再加入新的功能了，主要着重于除错。



扫码试看/订阅  
《玩转webpack》