

14 | 现代化React：现代工程化技术下的React项目

2022-09-27 宋一玮 来自北京

天下无鱼
<https://shikey.com/>

《现代React Web开发实战》

课程介绍 >



讲述：宋一玮

时长 20:09 大小 18.41M



你好，我是宋一玮，欢迎回到 React 应用开发的学习。

在第 12、13 节课，我们学习了 React 的单向数据流，以及怎么用面向接口编程的思想指导组件设计开发。同时我们一起为 oh-my-kanban 做了一次大重构，实践了刚学到的概念和方法。可以说，我们在学习写 React 应用代码方面，已经获得了阶段性进展。

但也需要知道，写出来的源码毕竟还不能用来上线，还得经过 `npm run build` 打包构建出生产环境代码，然后才能上线。你可能会好奇，这个命令都做了什么？这个命令是 CRA，由 Create React App 脚手架工具提供，它的内部对开发者而言是个黑盒。要想了解它，我们得先把黑盒打开，或者，用更好的方式：自己搭一个白盒出来。

还记得在上节课末尾的预告吗？这节课我会带着你，不依赖 CRA，用现代的工程化技术重新搭建一个 React 项目，然后把 oh-my-kanban 的代码迁移过来，让它真正成为你自己的项目。

好的，现在开始这节课的内容。

CRA 为我们做了什么？

在🔗第 3 节课，我们用 FB 官方提供的 CRA 脚手架工具创建了 oh-my-kanban 项目，在这之后我们就一直专注于代码开发，再也没有关注过项目配置了。现在 oh-my-kanban 项目开发已经步入正轨，是时候回过头来看看 CRA 为我们做了哪些事情。

在项目根目录 package.json 文件的 scripts 节点下，有四个 NPM 命令。

最先接触的 `npm start`，你对它的使用应该已经比较熟悉了。这个命令启动了一个开发服务器（Dev Server），内置了开发环境构建（Development Build）、监听文件变化（Watch）、增量构建（Incremental Build）、模块热替换（Hot Module Replacement）等功能。其实这些功能你在前面的开发实践中都用到了。

与这个命令对应的还有生产环境构建。

生产环境构建

我想请你在 oh-my-kanban 项目根目录运行一遍 `npm run build`，它会打包构建出生产环境的代码。现在只看生成的 JS/CSS 文件：

📄 复制代码

```
1 build/static
2 |— css
3 |   |— main.9411d92b.css           1.2K
4 |   |— main.9411d92b.css.map
5 |— js
6 |   |— 787.4ea3479b.chunk.js      4.5K
7 |   |— 787.4ea3479b.chunk.js.map
8 |   |— main.7ed853e1.js           166K
9 |   |— main.7ed853e1.js.LICENSE.txt
10 |   |— main.7ed853e1.js.map
11 |— media
12 |   |— logo.6ce24c58023cc2f8fd88fe9d219db6c6.svg
```

如果你的项目源码是跟课程的代码仓库同步的，请你运行 `git checkout a70667e`，检出🔗第 3 节课刚初始化 CRA 项目时的代码，再跑一次 `npm run build`，你会发现构建结果的文件个数和大小都大同小异：

```
1 build/static
2   └─ css
3       └─ main.073c9b0a.css          1.0K
4       └─ main.073c9b0a.css.map
5   └─ js
6       └─ 787.4ea3479b.chunk.js      4.5K
7       └─ 787.4ea3479b.chunk.js.map
8       └─ main.be7e86b0.js          140K
9       └─ main.be7e86b0.js.LICENSE.txt
10      └─ main.be7e86b0.js.map
11 └─ media
12     └─ logo.6ce24c58023cc2f8fd88fe9d219db6c6.svg
```

你从🔗第 3 节课到🔗第 13 节课写的代码，为生产环境代码增加了 26.2KB，这包括了运行时依赖项 **emotion**。这些生产环境代码是可以用于上线的。

下一个是 `npm test`，用于执行 **Jest** 自动化测试。我们会在后面的第 20~22 节详细介绍 **React** 自动化测试，这里暂时先跳过。

从 CRA 下车

最后来到 `npm run eject`。相信你已经把之前的代码都提交到代码仓库了吧？那放心执行它，遇到确认提示直接敲回车，直到你看到 **Ejected successfully!** 就成功了。你发现项目突然多了十来个新文件，纳闷地问这个命令是什么意思？

Eject 的字面意思是弹出，比如飞行员从战斗机中紧急弹出就是这个词。执行了这个命令，就代表你从 **CRA** 下车了：这个项目不再依赖 **CRA**，**CRA** 封装的各种工程化功能，都被打散加入到这个项目的代码中，你可以根据需要做深度定制。

根据打散出来的文件，可以看到 **CRA** 包含的基本功能：

- 基于 **Webpack** 的开发服务器和生产环境构建；
- 用 **Babel** 做代码转译（Transpile）；
- 基于 **Jest** 和 `@testing-library/react` 的自动化测试框架；
- **ESLint** 代码静态检查；
- 以 **PostCSS** 为首的 **CSS** 后处理器。

前端框架与脚手架工具之间是相辅相成的关系，一般而言后者比前者更有**倾向性**（Opinionated）。工具（或框架）具有倾向性，意味着它**对你的使用场景做了假设和限定，为你提供了它认为是最有效或是最佳实践的默认配置**。

当你和这样的工具一拍即合时，它会简化你需要解决的问题，提升你的开发效率；但当你有深度自定义的需求时，它能提供的灵活性往往是有限的，这时你就需要重新考虑是否仍要采用这个工具了。

其实到目前我们对 **CRA** 还没有什么不满。不过出于学习目的，我们**暂时从 CRA 下车**，然后开始自己搭建一套现代化的 **React** 项目。

搭建一个新项目

既然决定不依赖脚手架工具，那么就需要自己一边做技术选型，一边分步骤搭建一个新项目。我们已经确定的技术栈包括：

- Node.js v16 LTS;
- NPM v8 包管理器;
- React v18.2.0;
- Emotion CSS-in-JS 库;
- 浏览器 Web 技术。

至于其他技术栈，我们一边搭建一边引入。

创建前端项目没什么需要注意的，先起个新的项目名吧，yeah-my-kanban 怎么样：

 复制代码

```
1 mkdir yeah-my-kanban && cd yeah-my-kanban
2 git init
3 npm init -y
```

接着，在刚创建的 `package.json` 里加入一行 `"private": true`，，预防不小心把项目作为 NPM 包发布出去。

然后，在项目根目录加入 `.nvmrc` 文件用于约定 Node.js 版本。`fnm`、`nvm` 工具都能可以识别这个文件名，文件内容只有一行：

```
1 16.17.1
```

 复制代码

同时把 `oh-my-kanban` 的 `.gitignore` 文件直接拷贝过来，这个文件可以避免把不必要的文件提交到 `git` 代码仓库中。

在开始迁移 `oh-my-kanban` 源码之前，需要先为项目配置构建工具。

安装构建工具 Vite

在直播时我们曾讨论过，无论是软件工程化还是前端工程化，都是为了解决在开发中存在的痛点，提升开发效率效果。**构建**（Build）也是前端工程化领域最重要的话题之一。

`Webpack` 是前端领域最流行的**静态模块打包器**（Bundler），前面的 `CRA` 脚手架选用 `Webpack` 作为基础，以插件的形式加入代码转译、`CSS` 后处理、整合图片资源等功能，这样就可以支持完整的前端构建过程了。

`Bundler+` 插件之所以能成为前端构建工具的主流，很大程度上是因为浏览器技术的限制。现代 `JS` 应用开发动辄数十个依赖项、上百个源文件、上万行源代码，而传统浏览器由于 `JS` 引擎功能和网络性能等限制，无法直接消费这些 `JS`，所以需要 `Bundler` 来打包并优化交付给浏览器的产物。这其实也是一种对 `JS` 开发过程和浏览器环境适配的**关注点分离**（Separation Of Concerns）。

然而这个限制正在慢慢被放宽，现代浏览器开始支持 `HTTP/2`、`ECMAScript Modules` 标准，一些新兴的前端构建工具已经开始利用这些新功能。我们基于这个趋势，选择了 `Vite`（[官网](#)）作为 `yeah-my-kanban` 的构建工具。

`Vite` 为开发环境和生产环境提供了不同的方案，在开发时，`Vite` 提供了基于 `ESBuild` 的开发服务器，平均构建速度远超 `Webpack`；为生产环境，`Vite` 提供了基于 `Rollup` 的构建命令和预设配置集，构建出的产物，能达到与 `Webpack` 相当的优化程度和兼容性。

Vite 官方也提供了 `create-vite` 脚手架工具，但我们很倔强地不用，直接安装 Vite:

[📄 复制代码](#)

```
1 npm install -D vite
```

在 `package.json` 里加入两个新的命令:

[📄 复制代码](#)

```
1 "scripts": {  
2   "start": "vite dev --open",  
3   "build": "vite build",
```

再在项目根目录添加一个入口 HTML 文件 `index.html`:

[📄 复制代码](#)

```
1 <!DOCTYPE html>  
2 <html lang="en">  
3   <head>  
4     <meta charset="utf-8" />  
5     <meta name="viewport" content="width=device-width, initial-scale=1" />  
6     <title>React App</title>  
7   </head>  
8   <body>  
9     <noscript>You need to enable JavaScript to run this app.</noscript>  
10    <div id="root"></div>  
11  </body>  
12 </html>
```

运行 `npm start`，好的，浏览器自动打开页面，虽然里面什么内容都没有。

配置 React 插件

安装 `react`，顺便安装 Vite 的 React 插件:

[📄 复制代码](#)

```
1 npm install react react-dom  
2 npm install -D @vitejs/plugin-react
```

加入一个配置文件 `vite.config.js` :

复制代码

```
1 import { defineConfig } from 'vite';
2 import react from '@vitejs/plugin-react';
3
4 export default defineConfig({
5   plugins: [react()]
6 });
```

运行 `npm start` :

A terminal window titled 'npm (esbuild)' showing the output of running 'npm start'. The output indicates that Vite v3.1.3 is ready in 1063 ms and is serving the application on http://localhost:5173/. The terminal also shows the local and network addresses. The terminal window has a dark background with green and white text. The status bar at the bottom shows the current directory as ~/d/p/yeah-my-kanban, the command as esbuild node sh, and the battery level as 16%.

把 `oh-my-kanban` 的 `src/index.js` 文件拷过来, 改名为 `src/index.jsx`, 暂时注释掉一部分内容:

复制代码

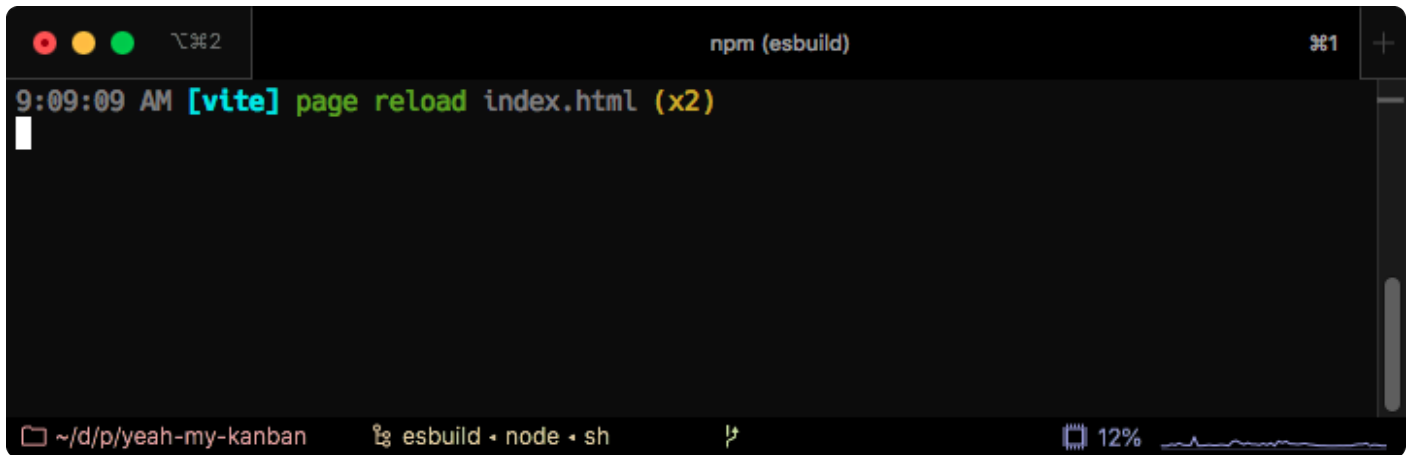
```
1 import React from 'react';
2 import ReactDOM from 'react-dom/client';
3 // import './index.css';
4 // import App from './App';
5
6 const root = ReactDOM.createRoot(document.getElementById('root'));
7 root.render(
8   <React.StrictMode>
9     { /* <App /> */}
10    <div>Yeah My Kanban</div>
11  </React.StrictMode>
12 );
13
```


回到 Vite 的入口文件 `index.html`，在 `<body>` 封闭标签最后加入一行特殊的 `<script>` 标签：

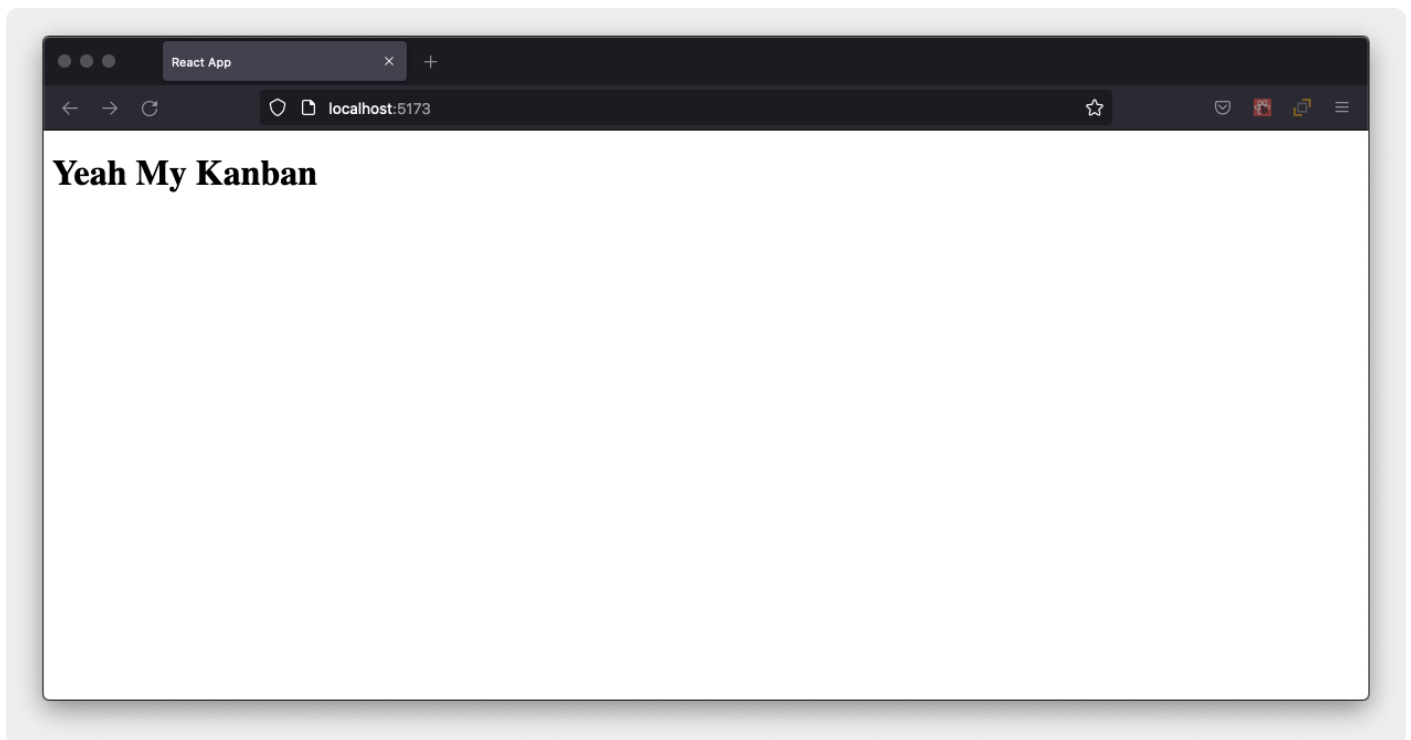
```
1 <script type="module" src="./src/index.jsx"></script>
```

复制代码

Vite 自动构建：



浏览器页面自动更新，显示出“Yeah My Kanban”字样了：



配置 Emotion

在 Vite 里配置 `emotion` 会稍微啰嗦些。安装 `emotion` 时需要额外安装一个开发依赖项：


```
1 npm install @emotion/react
2 npm install -D @emotion/babel-plugin
```

修改配置文件 `vite.config.js`，利用 `emotion` 的 Babel 插件为 JSX 加入 `css` 属性，这样也不需要在每个 JSX 文件开头写 JSX Pragma 了：

```
1 export default defineConfig({
2   plugins: [
3     react({
4       jsxImportSource: '@emotion/react',
5       babel: {
6         plugins: ['@emotion/babel-plugin'],
7       },
8     }),
9   ],
10 });
```

好了，准备工作完成，可以开始把 `oh-my-kanban` 的源码迁移至 `yeah-my-kanban` 了。

迁移项目源码

首先，把除了 `oh-my-kanban/src/index.js` 的组件文件、样式文件和 `context` 文件，一股脑地拷贝到 `yeah-my-kanban/src/components` 下。

再把里面的 `context` 文件移动到 `src/context/AdminContext.js`，这时 `VSCode` 会提示是否更新它在其他文件中的导入路径，选择“是”。然后把所有的组件文件扩展名改成 `.jsx`，否则 `Vite` 不认。目前 `yeah-my-kanban` 的源码应该是这样的：

```
1 src
2 |— components
3 |   |— App.css
4 |   |— App.jsx
5 |   |— KanbanBoard.jsx
6 |   |— KanbanCard.jsx
7 |   |— KanbanColumn.jsx
8 |   |— KanbanNewCard.jsx
9 |   |— logo.svg
```

```
10 |   context
11 |     └── AdminContext.js
12 |   index.css
13 |   index.jsx
```

所有.jsx 文件第一行的 `/** @jsxImportSource @emotion/react */` 可以删掉了。

把 `yeah-my-kanban/src/index.jsx` 的注释代码还原，注意 `App` 的导入路径变了：

复制代码

```
1 import React from 'react';
2 import ReactDOM from 'react-dom/client';
3 import './index.css';
4 import App from './components/App';
5
6 const root = ReactDOM.createRoot(document.getElementById('root'));
7 root.render(
8   <React.StrictMode>
9     <App />
10  </React.StrictMode>
11 );
```

完成。运行 `npm start`，浏览器中出现了熟悉的页面：



这时你也会发现，**Vite** 的开发服务器启动和初次构建都明显比 **Webpack** 快。对于yeah-my-kanban 这样体量很小的项目，这种速度提升不算明显。不过随着项目规模提升，Vite 构建的速度优势就体现出来了。

好了，迁移完成！也许你原本以为还需要很多步骤，但其实到这里我们的源码迁移已经成功完成了。你可以把yeah-my-kanban 项目的源码也提交到代码仓库里。

为编写代码保驾护航

接下来是与源码开发相关的工程化实践，包括代码自动补全、代码静态检查、单元测试、Git Hook。其中单元测试，我们留到后面第 20～22 节课详细介绍，这里暂时先跳过。

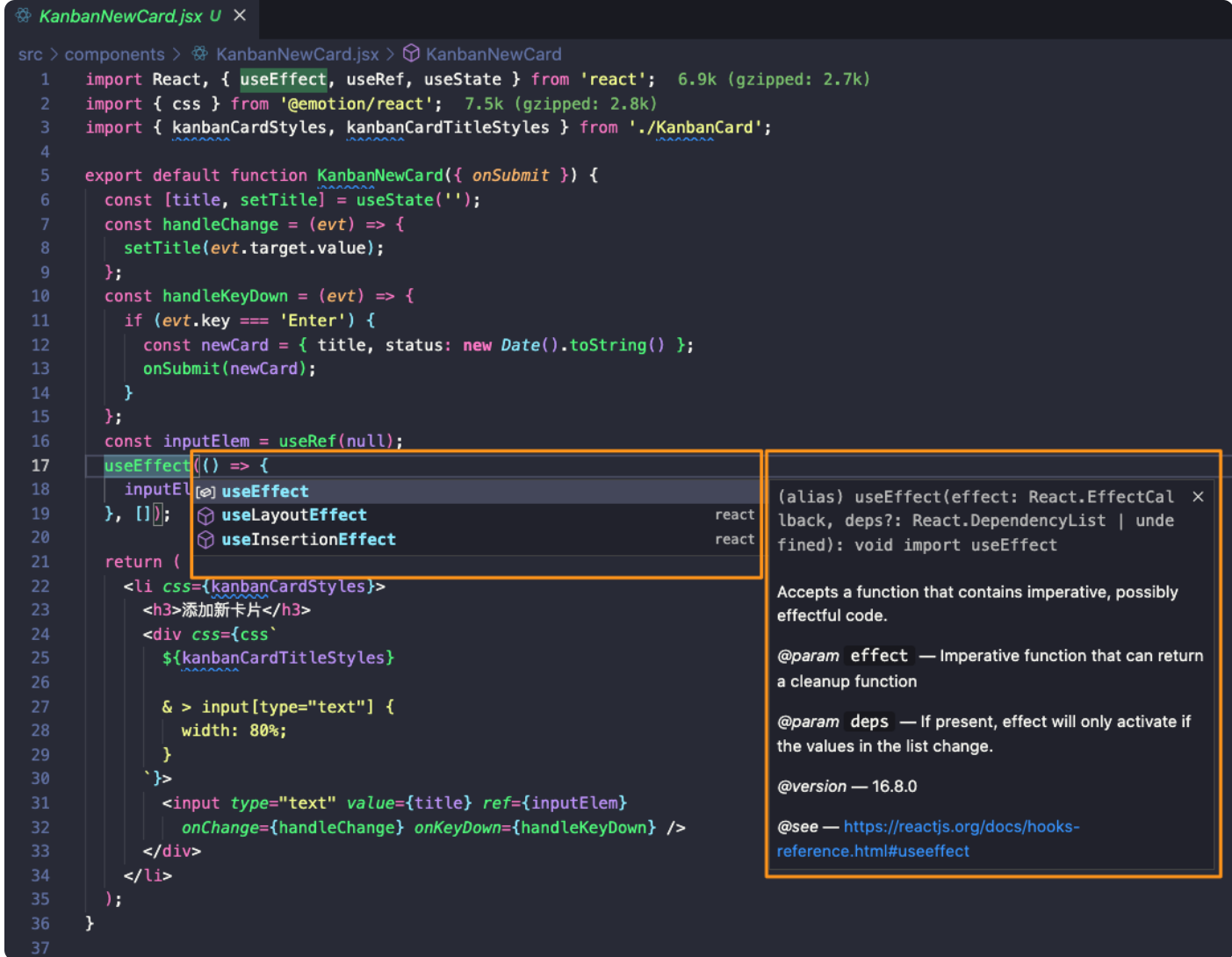
代码自动补全

现代 JS 开发是很幸福的，自从有了 TypeScript 生态，基本上常用的开源库都会以 `*.d.ts` 形式提供 Types 定义，IDE 读取这些定义，可以提供精准的代码自动补全列表；有不少库还同时提供了丰富的 JSDoc 或 TSDoc 文档，IDE 可以在代码提示中内嵌展示出来。

可以在安装 React 的 Types:

```
1 npm install -D @types/react @types/react-dom
```

 复制代码



如果你在 VSCode 中发现你什么都还没做，就能有 React API 的代码自动补全，那是因为它已经提前内置了。

代码静态检查

代码毕竟还是人编写的，人一定会犯错，这点不用避讳。**代码静态检查（Linting）**是通过静态代码分析，为开发者指出代码中可能的编程错误和代码风格问题，并提出修改建议，达到提升代码质量的目的。因此代码静态检查器（Linter），就是开发者的好伙伴。

在 JS 生态中，目前最强大使用最广泛的是 **ESLint**（[官网](#)）。

安装 ESLint:

复制代码

```
1 npm init @eslint/config -y
```

安装命令会依次问几个问题，大部分用默认值就行。其中需要注意，对于“你打算怎样使用 ESLint？”这个问题，请选择第三项“检查语法，寻找错误，规范代码风格”：

 复制代码


```
1 ? How would you like to use ESLint? ...
2   To check syntax only
3   To check syntax and find problems
4 > To check syntax, find problems, and enforce code style
```

后面还有一个问题，“你打算怎样定义项目的代码风格？”，请选择第一项“选择一个流行的代码风格指南”，随后我推荐选择 Airbnb 的代码风格：

 复制代码

```
1 ? How would you like to define a style for your project? ...
2 > Use a popular style guide
3   Answer questions about your style
4 ? Which style guide do you want to follow? ...
5 > Airbnb: https://github.com/airbnb/javascript
6   Standard: https://github.com/standard/standard
7   Google: https://github.com/google/eslint-config-google
8   XO: https://github.com/xojs/eslint-config-xo
```

运行完命令行会提示：

 复制代码

```
1 ✓ How would you like to use ESLint? · style
2 ✓ What type of modules does your project use? · esm
3 ✓ Which framework does your project use? · react
4 ✓ Does your project use TypeScript? · No / Yes
5 ✓ Where does your code run? · browser
6 ✓ How would you like to define a style for your project? · guide
7 ✓ Which style guide do you want to follow? · airbnb
8 ✓ What format do you want your config file to be in? · JavaScript
9 Checking peerDependencies of eslint-config-airbnb@latest
10 Local ESLint installation not found.
11 The config that you've selected requires the following dependencies:
12
13 eslint-plugin-react@^7.28.0 eslint-config-airbnb@latest eslint@^7.32.0 || ^8.2.
14 ✓ Would you like to install them now? · No / Yes
15 ✓ Which package manager do you want to use? · npm
16
17 Installing eslint-plugin-react@^7.28.0, eslint-config-airbnb@latest, eslint@^7.
```

安装完成，项目根目录多了一个 `.eslintrc.js` 配置文件。

你已经等不急要体验一下 ESLint 的功能了。在 `package.json` 中增加一个 NPM 命令：

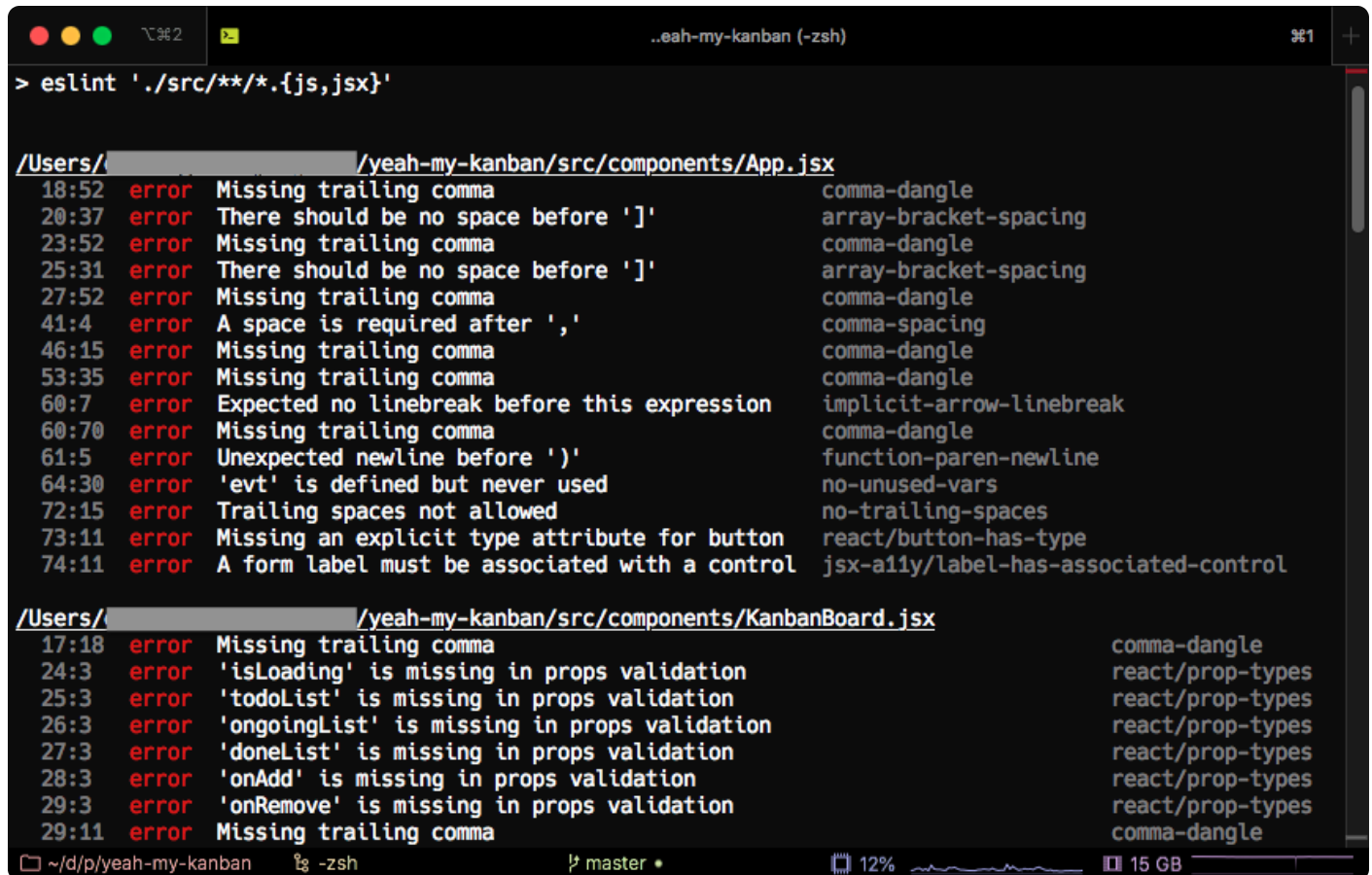
```
1  "scripts": {
2    "start": "vite dev --open",
3    "build": "vite build",
4    + "lint": "eslint './src/**/*.{js,jsx}'",
```

为了避免误伤，在 `vite.config.js` 顶部插入一行：

```
1  /* eslint-disable import/no-extraneous-dependencies */
```

复制代码

好了，执行 `npm run lint`，结果如下：



```
> eslint './src/**/*.{js,jsx}'

/Users/.../yeah-my-kanban/src/components/App.jsx
18:52  error  Missing trailing comma                comma-dangle
20:37  error  There should be no space before ']'  array-bracket-spacing
23:52  error  Missing trailing comma                comma-dangle
25:31  error  There should be no space before ']'  array-bracket-spacing
27:52  error  Missing trailing comma                comma-dangle
41:4   error  A space is required after ','         comma-spacing
46:15  error  Missing trailing comma                comma-dangle
53:35  error  Missing trailing comma                comma-dangle
60:7   error  Expected no linebreak before this expression  implicit-arrow-linebreak
60:70  error  Missing trailing comma                comma-dangle
61:5   error  Unexpected newline before ')'         function-paren-newline
64:30  error  'evt' is defined but never used       no-unused-vars
72:15  error  Trailing spaces not allowed           no-trailing-spaces
73:11  error  Missing an explicit type attribute for button  react/button-has-type
74:11  error  A form label must be associated with a control  jsx-a11y/label-has-associated-control

/Users/.../yeah-my-kanban/src/components/KanbanBoard.jsx
17:18  error  Missing trailing comma                comma-dangle
24:3   error  'isLoading' is missing in props validation  react/prop-types
25:3   error  'todoList' is missing in props validation  react/prop-types
26:3   error  'ongoingList' is missing in props validation  react/prop-types
27:3   error  'doneList' is missing in props validation  react/prop-types
28:3   error  'onAdd' is missing in props validation  react/prop-types
29:3   error  'onRemove' is missing in props validation  react/prop-types
29:11  error  Missing trailing comma                comma-dangle
```

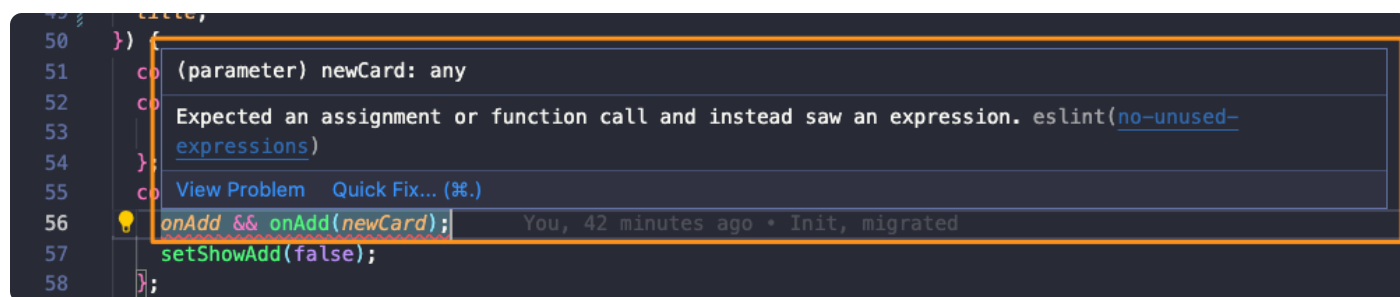
复制代码

```
✖ 86 problems (86 errors, 0 warnings)
37 errors and 0 warnings potentially fixable with the `--fix` option.
```

居然检查出这么多错误？别担心，大部分都是代码格式的报错，反正代码提交过了，我们可以放心使用自动修正功能。运行 `npm run lint -- --fix`，ESLint 自动修正了一部分错误，还剩 50 多个错误。接下来，让我们看看还剩下哪些典型的错误。

Lint 规则：禁止不被使用的表达式

对应的规则说明在这里：<https://eslint.org/docs/latest/rules/no-unused-expressions>。



上面代码中的表达式，在 JS 中有个专门的称呼：**短路表达式**（Short-Circuit Expression），在前端开发中还是很常用的。我们在 `.eslintrc.js` 的 `rules` 字段中加入如下一行规则，为它开个绿灯：

```
1 rules: {
2   'no-unused-expressions': ['error', { allowShortCircuit: true }],
3 }
```

复制代码

Lint 规则：禁止在函数内部修改函数参数

对应的规则说明在这里：<https://eslint.org/docs/latest/rules/no-param-reassign>。


```
60 return {
61   <sec (parameter) evt: React.DragEvent<HTMLDivElement>
62   on
63   Assignment to property of function parameter 'evt'. eslint(no-param-reassign)
64   View Problem Quick Fix... (%)
65   evt.dataTransfer.dropEffect = 'move';
66   setIsDragTarget(true);
67 }
68 onDragLeave={() => {
69   evt.preventDefault();
70   evt.dataTransfer.dropEffect = 'none';
71   setIsDragTarget(false);
72 }}
```

这个规则是非常有用的，可以避免很多编程问题。但 `dropEffect` 算是特例，我们加条规则排除掉它：

复制代码

```
1 'no-param-reassign': ['error', { props: true, ignorePropertyModificationsFor: [
```

Lint 规则：React 组件的 props 需要定义 PropTypes

对应的规则说明在这里：<https://github.com/jsx-eslint/eslint-plugin-react/blob/master/docs/rules/prop-types.md>。

```
22 const MINUTE = 60 * 1000;
23 const HOUR = 60 * MINUTE;
24 const DAY = 24 * HOUR;
25 const UPDATE_INTERVAL = MINUTE
26
27 import default function Kanban
28 title, status, onDragStart, onRemove, You, 1 minute ago * Uncommitted changes
29 ) {
30   const [displayTime, setDisplayTime] = useState(status);
31   useEffect(() => {
32     const updateDisplayTime = () => {
33       const timePassed = new Date() - new Date(status);
```

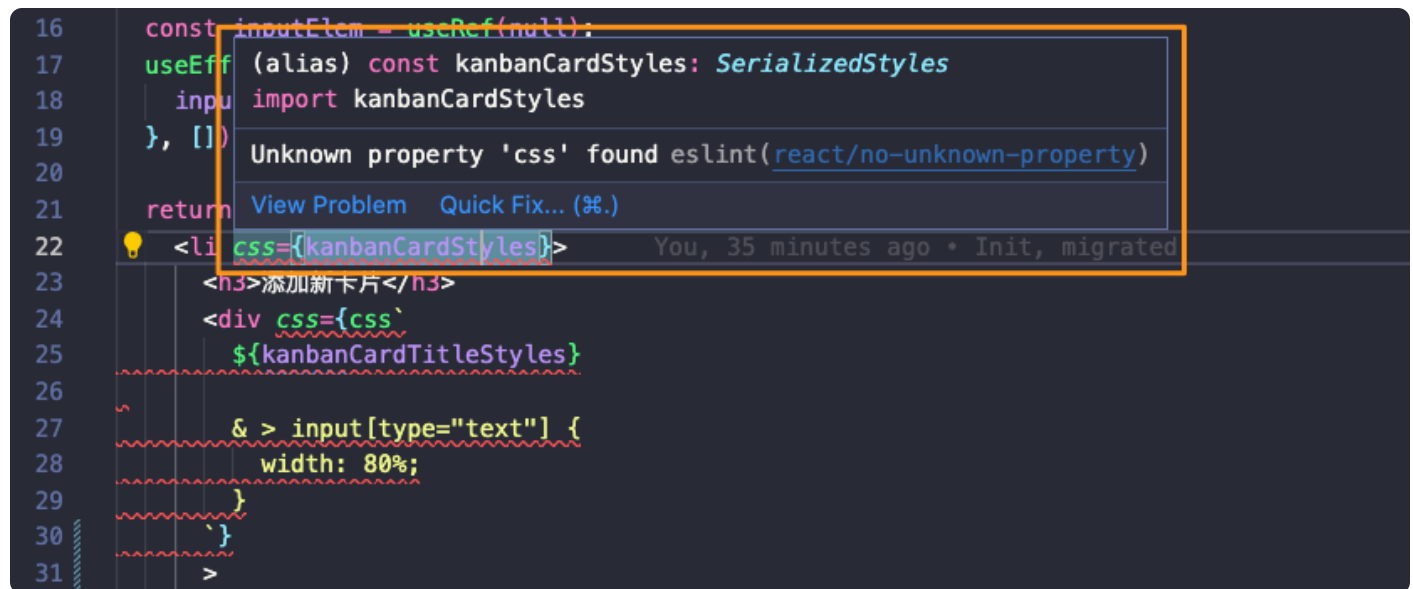
我们后面第 17 节课会讲到 `PropTypes`，所以现在先无视它。在 `.eslintrc.js` 的 `rules` 字段中加入如下一行规则，以覆盖 `plugin:react/recommended` 规则集中的默认值：

复制代码

```
1 'react/prop-types': ['error', { skipUndeclared: true }],
```

Lint 规则：React 组件禁止使用未知的 DOM 属性

对应的规则说明在这里：<https://github.com/jsx-eslint/eslint-plugin-react/blob/master/docs/rules/no-unknown-property.md>。



```
16 const inputElem = useRef(null);
17 useEff (alias) const kanbanCardStyles: SerializedStyles
18   import kanbanCardStyles
19 }, [])
20   Unknown property 'css' found eslint(react/no-unknown-property)
21   View Problem Quick Fix... (⌘.)
22   <li css={kanbanCardStyles}> You, 35 minutes ago • Init, migrated
23   <h3>添加新卡片</h3>
24   <div css={css`
25     ${kanbanCardTitleStyles}
26
27     & > input[type="text"] {
28       width: 80%;
29     }
30   `}>
31   >
```

这个属于误伤，`plugin:react/recommended` 并不知道 `emotion` 框架的存在。加一行配置忽略它：

复制代码

```
1 'react/no-unknown-property': ['error', { ignore: ['css'] }],
```

再跑一次 `npm run lint`，还剩 11 个错误。你可以尝试自己修正或者忽略它们。

Lint 规则：检查 React Hooks 的使用规则

等一下还没完，请你回忆第 10 节课学习的 React Hooks 的使用规则，ESLint 能帮上忙吗？故意用错 Hooks 试试看。需要先修改 `.eslintrc.js`，启用 Airbnb 代码规则集里 Hooks 的部分：

```

1  module.exports = {
2    // ...
3    extends: [
4      'plugin:react/recommended',
5      'airbnb',
6 +   'airbnb/hooks',
7   ],

```

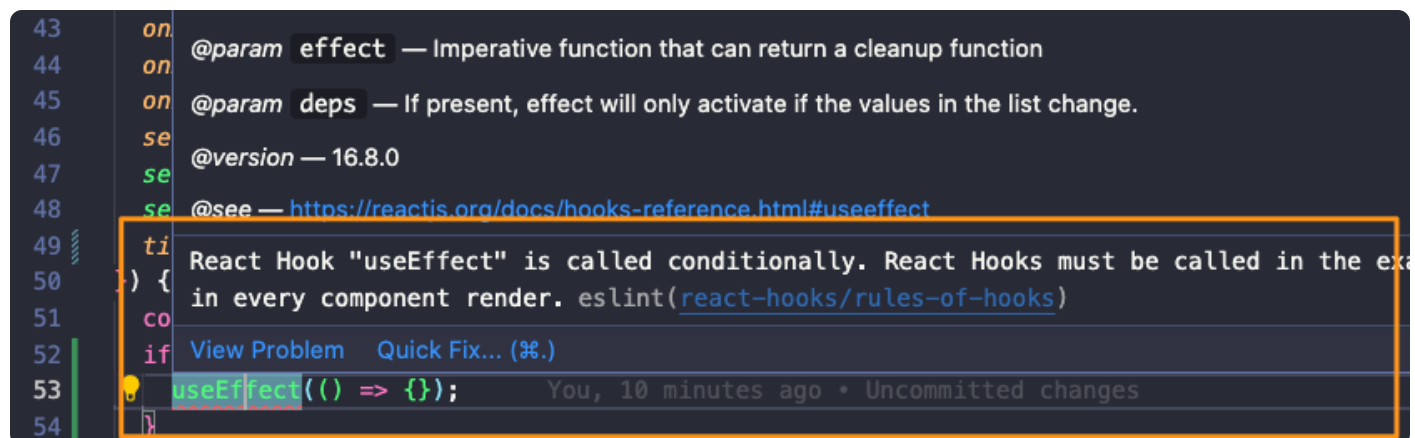
然后故意写个 Bug:

```

1  export default function KanbanColumn({
2    // ...
3  }) {
4    const [showAdd, setShowAdd] = useState(false);
5 +   if (showAdd) {
6 +     useEffect(() => {});
7 +   }

```

还没运行 lint 命令，VSCode 里就根据 ESLint 规则报错了：



对应的规则说明： <https://zh-hans.reactjs.org/docs/hooks-rules.html>。靠谱儿。

Git Hook

“今日事今日毕”。你开发工作忙碌一天，下班前最后一件事是什么？加班？不不，我是指提交本地代码到代码仓库，所谓“落袋为安”。就在这个提交代码过程中，你也有机会用更高标准要

求自己：今天新写的代码必须通过 **Lint** 和 **Test**，否则禁止提交。

第一步，安装 **Git Hook** 工具 **husky**：

```
1 npx husky-init && npm install
```

 复制代码


在 **package.json** 中额外加入一个 **lint-staged** 命令：

```
1   "scripts": {  
2     "prepare": "husky install",  
3   +   "lint-staged": "echo 'Pre-commit!'"  
4   },
```

在新加入的 **.husky/pre-commit** 中把默认的 **npm test** 改为 **npm run lint-staged**，这样之后加 **Git Hook** 只要改 **package.json** 就可以了。

来，测试一下，命令行打印 **Pre-commit!** 就成功了：

```
1 git add .  
2 git commit -m "Husky"
```

 复制代码

第二步，安装 **lint-staged**，这个工具会保证只检查需要提交的文件，而不是所有文件：

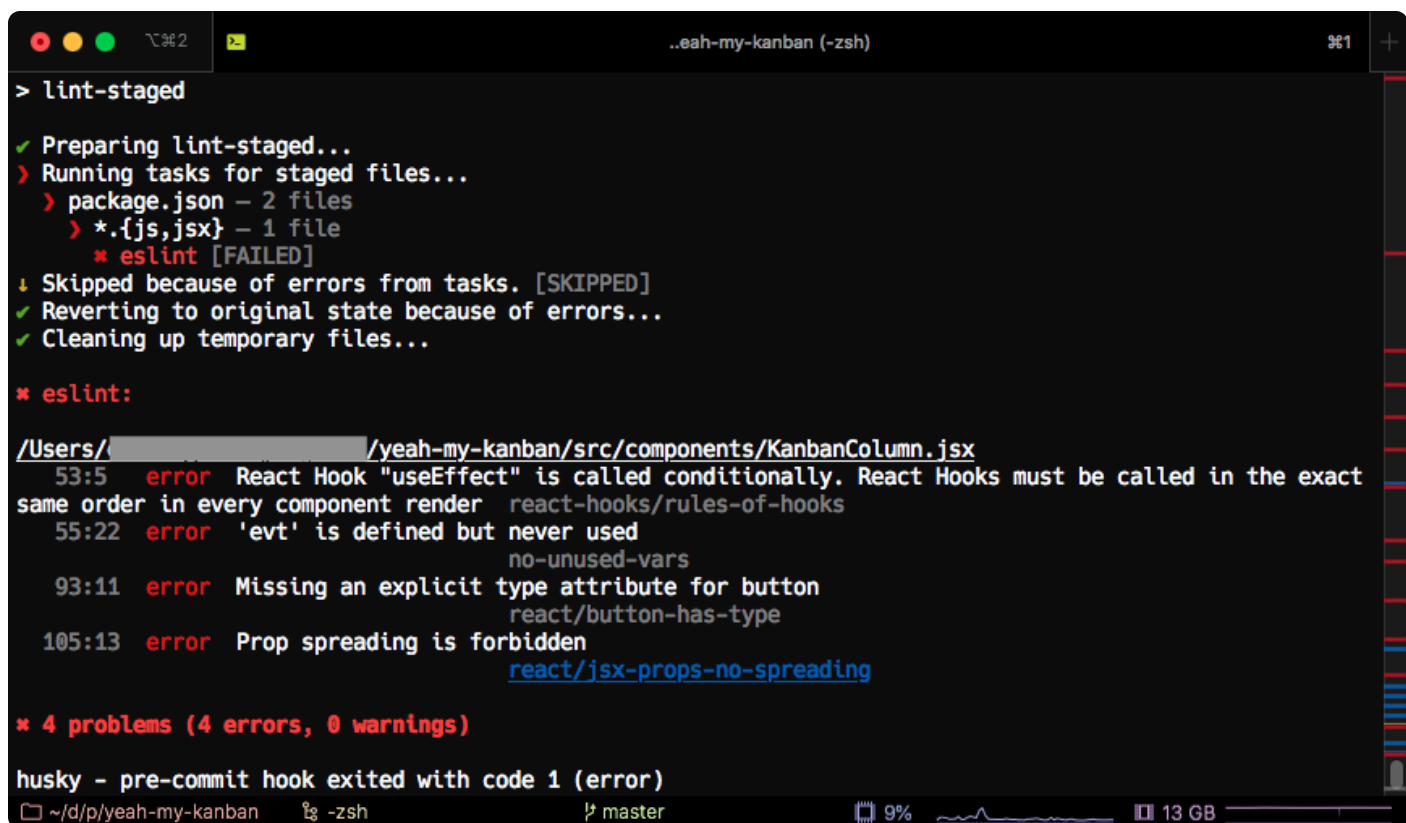
```
1 npm install -D lint-staged
```

 复制代码

在 **package.json** 中调用 **lint-staged**：

```
1   "scripts": {
2     "prepare": "husky install",
3 -   "lint-staged": "echo 'Pre-commit!'"
4 +   "lint-staged": "lint-staged"
5   },
6 +   "lint-staged": {
7 +     "*. {js,jsx}": "eslint"
8 +   },
```

随便在哪个 JSX 文件中加个空格，尝试提交，怎么样？你被拦住了吧（得意状）？



```
> lint-staged

✓ Preparing lint-staged...
> Running tasks for staged files...
  > package.json - 2 files
    > *. {js,jsx} - 1 file
      ✖ eslint [FAILED]
  ↓ Skipped because of errors from tasks. [SKIPPED]
✓ Reverting to original state because of errors...
✓ Cleaning up temporary files...

✖ eslint:

/Users/.../yeah-my-kanban/src/components/KanbanColumn.jsx
53:5   error  React Hook "useEffect" is called conditionally. React Hooks must be called in the exact same order in every component render  react-hooks/rules-of-hooks
55:22  error  'evt' is defined but never used        no-unused-vars
93:11  error  Missing an explicit type attribute for button  react/button-has-type
105:13 error  Prop spreading is forbidden             react/jsx-props-no-spreading

✖ 4 problems (4 errors, 0 warnings)

husky - pre-commit hook exited with code 1 (error)
```

也不用担心，只要修好就能提交成功了。我常说 lint-staged 是个“自律”工具，可以逼迫自己提高代码质量。

小结

这节课我们不再依赖 CRA，而是选用更高效的工程化工具 Vite，从零开始，亲手搭建了一个新的 React 项目 yeah-my-kanban。并且不费吹灰之力，把 oh-my-kanban 的代码迁移了过来，熟悉了与 React 应用代码直接相关的工程化概念和工具。其中我们也重点介绍了代码静态检查工具的用法和部分规则，以及 Git Hook 这种“自律”工具。

到此为止，你已经学习了 **React** 开发的基础内容，相信你已经有能力成为一位 **React** 开发的“独狼”工程师了。


从下节课开始，我们将进入新的模块，学习一些中型、大型 **React** 项目中会用到的技术和最佳实践，尤其是介绍当你融入一个前端开发团队时，需要的开发工作思路和方式的转变，这会帮你更从容地应对中大型 **React** 应用项目。

思考题

我曾强调过，前端工程化不是凭空出现的，而一定是为了解决在开发中存在的痛点，提升开发效率效果。你在前端开发过程中，尤其是第 3 节课到 13 节课期间的实践中，遇到过哪些痛点？你自己都是怎么解决的？你知道在前端技术社区有什么对应的工程化实践吗？

好了，这节课内容就是这些。“独狼”**React** 工程师，我们下节课不见不散！

分享给需要的人，Ta购买本课程，你将得 18 元

 生成海报并分享

 赞 2  提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 13 | 组件表与里（下）：用接口的思路设计开发**React**组件

下一篇 加餐01 | 留言区心愿单：真·子组件以及jsx-runtime

精选留言 (3)

 写留言



船长

2022-09-30 来自北京

记得想要 **Eslint** 生效要启动 **vscode** 中的 **eslint** 插件。。

作者回复: 你好，船长，感谢提醒，确实是需要的。

其他使用VSCode但还没有安装ESLint插件的同学，强烈推荐安装：<https://marketplace.visualstudio.com/items?itemName=dbaeumer.vscode-eslint>



DullSword

2022-09-28 来自北京

增加NPM 命令lint出错的小伙伴可以试试：

```

```
"lint": "eslint \"/src/**/*.{js,jsx}\"",
```

```

作者回复: 赞！



船长

2022-09-27 来自北京

3-13 痛点：

没有报错提醒

没有智能提示，比如在引入 `useEffect`，浏览器直接报错，原因是没有在顶部 `import`，这时候还需要手写去引入

作者回复: 你好，船长，确实，自动import真是太有用了。如果你使用的是VSCode，可以参考 <http://code.visualstudio.com/shortcuts/keyboard-shortcuts-macos.pdf> 里面有快捷键 `^Space, ⌘I` Trigger suggestion。当输入`useEffect`，打开代码提示，会有添加import的选项。

