

加餐 | 基础篇思考题答疑

2022-12-07 杨文坚 来自北京



天下无鱼

<https://shikey.com/>

《Vue 3 企业级项目实战课》

[课程介绍 >](#)



讲述：杨文坚

时长 07:19 大小 6.69M



你好，我是杨文坚。

首先要感谢你参与课程的学习，有几位同学 @风太大太大、@鱼吖鱼。、@都市夜归人、@雪黎... 讨论很积极，特别表扬，今天我们会对基础篇做一次统一的答疑，如果还有疑问也欢迎你留言。

01

课程：🔗 [01 编译和非编译模式](#)

提问：Vue.js 3 非编译场景与 Vue.js 的 JSX 写法有什么联系吗？

简单说 JSX 是一种语法糖，可以通过编译工具生成 Vue.js3 “非编译模式”的代码。

这里我想顺便展开聊聊 DOM Base Template（Vue.js3 的推荐语法）和 JSX 的差别。

DOM Base Template 受控，易于静态分析，逻辑引入需要特定语法支持：



- 受控：开发者难以实现无法预计的能力。
- 易于静态分析：因为本身是 HTML 语法，非常易于静态分析。
- 逻辑引入需要特定语法支持：例如需要分支逻辑，需要引入新概念 v-if 等。

JSX 则自由，难以静态分析，引入极少的概念：

- 自由：开发者可以任意使用，实现自己的任意逻辑。
- 难以静态分析：由于过于自由，所以 JSX 难以静态分析。
- 引入极少的概念：因为本身 Javascript 就是各种逻辑语法，所以 JSX 无需引入更多概念去实现各种逻辑。

02

课程：🔗 [02 Webpack 编译搭建，用 Webpack 初构建你的 Vue 3 项目](#)

提问：Webpack 从诞生到现在这么久，核心也迭代了很多大版本，那不同版本在打包构建上有什么差异吗？

这个思考题是想引导你查看官方文档，去了解每个版本 Webpack 都优化什么，为什么需要优化，怎么优化？因为官方博客有详细描述，这里就不做赘述了，有兴趣可以看官方文档的中文翻译内容：

- 🔗 [Webpack 5 发布，发布意味着什么？](#)
- 🔗 [Webpack 4 发布了](#)

03

课程：🔗 [03 用 Rollup 构建你的 Vue 3 项目](#)

提问：Vite 开发模式下的 esbuild 的按需打包编译，真的是绝对比 Rollup 的 Bundle 打包方式快吗？如果不是，能否推演一下有哪些可能的打包慢场景？



Vite 开发模式下，如果是 esbuild 单纯将 JS/TS 编译成 ES Module，很大程度是比 Rollup 快的，这是语言层面的运行能力决定的，利用了 Go 语言多线程的能力进行处理，多线程具备支持共享内存等优势。

但是 Rollup 是纯粹基于 JavaScript 来执行代码，JavaScript 是单线程的，即使 Node.js 环境能用多线程进行提效，但不能共享内存。所以暂时还没发现比 Rollup 慢的极端场景。

但 Vite 开发模式下比较“慢”，不是 esbuild 导致的，而是 ES Module 的模块多层依赖导致的。每个 ES Module 就是一个 HTTP 请求，如果一个 ESM 的 JS 背后 import 的 JS 有多级依赖，就会产生多个 HTTP 请求，而且 ESM 格式下，JS 的执行是会被依赖加载过程阻塞的。

顺便提一下，esbuild 其实也不是万能的，主要是对 ES6+ 的语法处理比较有优势，但是对于 ES5 语法处理就存在一些缺陷，例如压缩 ES5 代码时，就不一定能完全支持所有语法。

04

课程：🔗 [04 模板语法和 JSX 语法](#)

提问：前端开发组件经常会遇到组件的“递归使用”，也就是组件内部也循环使用了组件自己，那么，如何用模板语法和 JSX 语法处理组件的“自我递归使用”呢？

- JSX 语法里，组件的递归使用就是方法的递归使用。可以用函数式组件来实现。
- 模板语法里，组件的递归使用就是组件自己用 options API 方式定义组件名称，然后在组件模板使用组件自己。

我还看到有同学问了一个关于 Dialog 的疑问，这里也回答一下。

Dialog 动态场景，模板语法也能实现的，只不过没 JSX 那么“灵活”。这里说的“灵活”，很多体现在 JSX 语法不需要依赖独立的 Vue 文件。模板语法实现需要把组件写在 Vue 文件里，但是 JSX 语法同个文件就能实现功能。

用 JSX，说到底就是对 JSX 的熟悉程度和灵活度的理解。熟悉什么语法，或者认为什么用起来灵活，就选择最高性价比的方式。



05

课程：[🔗 05 理解和使用 Vue 3 的响应式数据](#)

提问：这节课都是基于组合式 API（Composition API）的开发方式来进行响应式操作，那么如果换成选项式 API（Options API）的开发方式，功能的实现应该怎么操作？

这主要是考察你对 Options API 的熟悉程度，文章里有许多例子就不一一演示了，我们把最后一段代码改写成 Options API 作为演示：

复制代码

```
1 <template>
2   <form>
3     <textarea v-model="state.text" placeholder="信息" />
4     <div>中文字数: {{ state.zhCount }}</div>
5   </form>
6 </template>
7
8 <script>
9 const defaultVal = '今天是2022年01月01日'
10 // 计算文本中文个数函数
11 function countZhText(txt) {
12   const zhRegExp = /[\\u4e00-\\u9fa5]{1,}/g;
13   const zhList = txt.match(zhRegExp);
14   let count = 0;
15   zhList.forEach((item) => {
16     count += item.length;
17   });
18   return count;
19 }
20
21 export default {
22   data: function () {
23     return {
24       state: {
25         text: defaultVal,
26         zhCount: countZhText(defaultVal)
27       }
28     }
29   },
30   watch: {
31     'state.text': {
32       handler(val, oldVal) {
```

```
33     this.state.zhCount = countZhText(val)
34   }
35 }
36 }
37 }
38 </script>
```



06

课程：🔗 [06 常见的组件间数据通信方式](#)

提问：这节课主要讲解多种跨组件的数据通信方式，每种方式虽然有其适用的场景，但若使用不规范，都会存在响应式数据被污染的隐患，那么如何更好地保护响应式数据，在跨组件通信过程中得到规范使用呢？

无论是哪种组件通信方式，都会有传递 JSON 格式（Object 类型，包括 JSON 和 Array）的响应式数据的场景。JSON 数据传递的只是个“引用”，任何修改这个响应式 JSON 的属性内容，都会触发响应式行为。但有些时候，业务操作是想使用传递拿到的响应式 JSON 数据，这时候就需要用 `toRaw` 进行解除响应式影响，再进行修改 JSON 数据。

这只是一个场景举例，实际开发过程中，需要团队人为来 Code Review 代码，避免出现这里提到的 JSON 响应式数据的修改，导致不必要的视图变化。

团队在操作 JSON 响应式数据时候要考虑影响面，如果确定要影响依赖到数据的视图，就可以修改。如果只是临时修改数据，不想影响视图，就要解除数据的响应式，甚至还要考虑是否需要深拷贝。

这里提到的 JSON 都是 JSON 对象字面量（JSON Object Literals）

07

课程：🔗 [07 掌握项目代码规范，成为一名合格的团队协作工程师](#)

提问：在实际项目中，如何结合 git 流程，在提交代码（git commit）或者推送代码（git push）时候进行自动化的 ESLint 代码质量检查？

如果打开项目根目录下的 .git 文件夹，你会发现在 hooks 文件夹有这些文件：

✓ .GIT



天下无鱼

<https://shike.com/>

✓ hooks

```
$ applypatch-msg.sample
≡ commit-msg.sample
≡ fsmonitor-watchman.sample
≡ post-update.sample
≡ pre-applypatch.sample
$ pre-commit.sample
≡ pre-merge-commit.sample
≡ pre-push.sample
≡ pre-rebase.sample
≡ pre-receive.sample
≡ prepare-commit-msg.sample
≡ push-to-checkout.sample
≡ update.sample
```

这是 git 的 hooks 案例，如果我们想在 commit 前或者 push 前做一些操作，就可以通过创建 pre-commit 或 pre-push 来达到效果，比如我们在创建一个 pre-commit，并配置：

复制代码

```
1 STAGE_FILES=$(git diff --cached --name-only --diff-filter=ACM -- '*.vue' '*.js')
2 if test ${#STAGE_FILES} -gt 0
3 then
4     echo '开始eslint检查'
```

```

5   which eslint &> /dev/null
6   if [[ "$?" == 1 ]]; then
7       echo '没安装eslint'
8       exit 1
9   fi
10
11  PASS=true
12
13  for FILE in $STAGE_FILES
14  do
15      eslint $FILE
16      if [[ "$?" == 1 ]]; then
17          PASS=false
18      fi
19  done
20
21  if ! $PASS; then
22      echo "eslint检查没通过！"
23      exit 1
24  else
25      echo "eslint检查完毕"
26  fi
27
28  else
29      echo '没有js文件需要检查'
30  fi
31
32  exit 0
33

```




天下无鱼
<https://shikey.com/>

每次提交前，都会触发对提交的 .js 和 .vue 文件做 eslint 检查（记得配置 eslint 配置文件），只有 eslint 通过了才能提交成功。

当然，实际项目中，可能很少人会直接配置 git hooks，更多人会 husky 等工具简化配置，有兴趣你可以查看 husky 的官方文档 <https://typicode.github.io/husky/#/>。

下一讲我们进入进阶篇，学习大厂里企业级项目常用的技术知识，包括 Vue.js 的生态知识和前端领域的相关概念，从基础组件，到动态渲染组件、布局组件，一步步地完成自研组件库的搭建。下一讲见。

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



上一篇 07 | 项目代码规范：如何成为一名合格的团队协作工程师？

下一篇 08 | 如何从零搭建自研的Vue组件库？

精选留言

💬 写留言

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。