

7. Vue项目实战（一）项目评审与架构设计

1. 课程资料

<https://github.com/encode-studio-fe/encode-byelide>



byelide.zip

1.64MB



vue-action.zip

244.57KB



课程源码，重点看这几个分支：

- features/base-config
- features/base-app-layer

使用工具：sourcetree

没怎么用过 git 的同学，用 sourcetree、vscode 自带的

2. 课程目标




- 初级：
 - 了解无代码平台核心业务模型
 - 了解可视化相关架构设计思想
 - 掌握 Vue3 项目基础架构
- 中级：
 - 掌握 Vue3 项目工程化思维，包括打包构建流程
 - 掌握 Vue3 开发核心 API 及组件与模块化设计思想
 - 了解项目规范与团队管理办法
- 高级：
 - 掌握前端企业及项目架构设计思想
 - 掌握从零到一设计架构一个产品

- 掌握技术选型与评估，并能独立设计开发满足业务的组件以及封装库

注意：因为是实战课，除了看懂项目架构与基础开发以外，还要着重培养项目相关描述，在面试过程中这点非常重要！

本节课我们主要从业务层面出发，让大家深度参与一个项目从需求阶段到落地的整个实施过程。

 特别是之前很多同学没有参与过大型项目的，主要写后台相关 CRUD 的同学，一定一定要完全掌握本次实战课程的内容！这会直接决定你的编程思维与薪资涨幅！

3. 课程大纲

1. 项目需求分析评审（入料、编排、渲染、出码）
2. Vue 项目基础架构设计，基于 Vite、Vue3、Pinia、Vue-Router
3. 构建基础框架（导航、物料、配置）
4. 编排的选型与实现（流式、画布式、Grid）

 目录导航

4. 项目需求分析评审（入料、编排、渲染、出码）

4.1 背景介绍

近年来，无代码平台与可视化相关平台日益火热。

无代码允许开发人员低成本通过拖拉拽的方式快速构建企业内部系统或落地页，常见的无代码平台能力包括：审批流、应用构建等。其在企业开发提效与运营方面取得了不错的成绩。例如国外的：

<https://retool.com/>（宇宙最强低代码）、Coda、Airtable、Notion、

<https://www.unrealengine.com/zh-CN> 等产品，例如国内的：明道云、极简云、ClickPaaS 等产品。

构建无代码平台是一件有挑战性的事情，这节实战课会从业务与技术架构入手，带你彻底看懂无代码平台研发。

可视化平台允许数据分析人员通过拖拉拽的方式快速组建仪表盘或数据分析大盘，其在企业运营方面发挥重要作用。例如国外的：PowerBI、Monday、Coda 等产品，国内的：观远数据、BDP、FineBI 等产品。构建可视化 SaaS 是一件非常有趣的事情，这节实战课会从业务与技术架构出发，带你从零完成可视化图表数据协议设计、图表渲染器开发、图表编排等功能。

当无代码遇上可视化，相信会有更多火花，没错，我们课程将这两大火热业务方向进行融合，带你真正体验大厂企业级项目的设计、开发与流程管控。

开发过程中，我们会严格按照大厂开发流程，从项目搭建、规范约定、技术选型与评估、需求分析到开发，每个流程细节我们都不会放过。技术选型方面，我们会选用 Vue3 作为技术栈，状态管理选用 Pinia，编辑器开发我们会基于 tiptap 来展开，Echarts 开发图表相关内容，除此之外，我们会向大家展示如何更优雅更松散处理复杂表单的场景。相信你学完后，一定会感慨自己就这样又更上了一层楼！

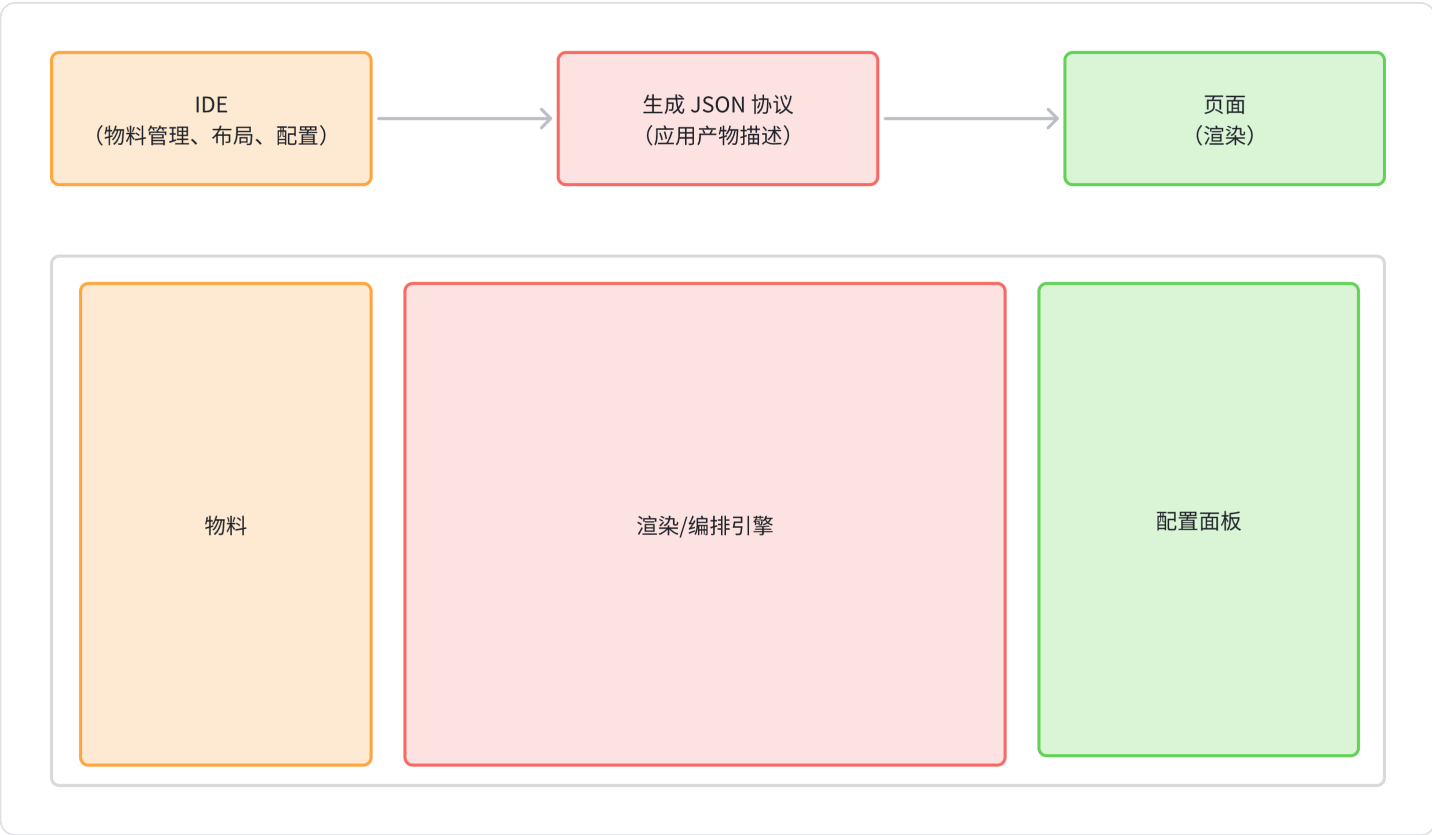
💡 请注意无代码与低代码的主要区别：

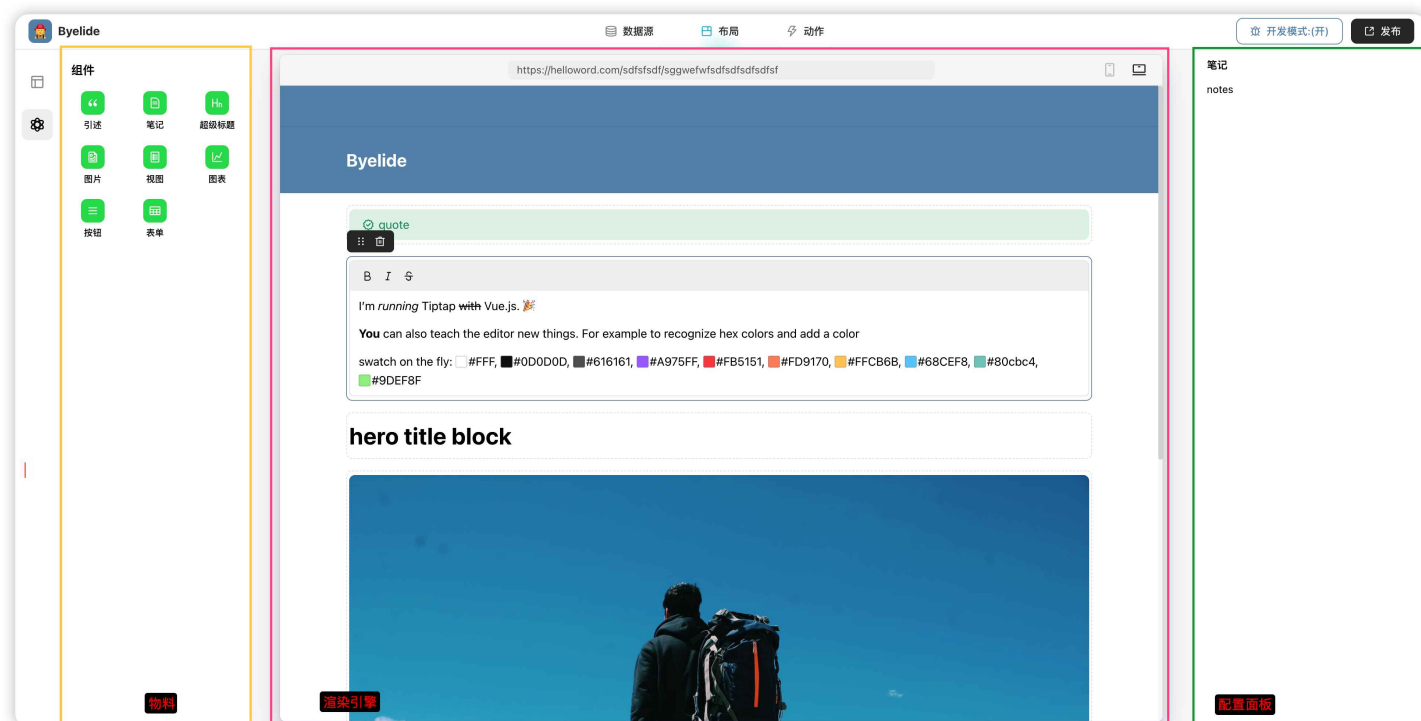
1. 无代码一般面向普通用户，可以没有编程经验，可以以最简单形式搭建应用。而低代码需要用户具备一定的编程能力。
2. 在实现上，低代码比无代码更复杂，低代码还会存在例如 DSL 以及代码植入的能力，低代码的代码片段植入思路接近微前端。

4.2 核心能力

- 可视化配置面板
- 具有可拓展能力：组件、模板、逻辑复用
- 生命周期管理：开发管理、页面管理、部署管理

4.3 架构设计





4.3.1 物料堆

包含可供使用的物料，通常物料的组织与消费是插件化（微内核）的，还有很多产品将物料组织发布到插件市场，可供用户自行拓展，这都是跟插件化思想与 **js 沙盒** 设计离不开的。

<https://juejin.cn/post/6981374562877308936>

4.3.2 编排

- 微内核思想：操作的是DSL（json树）， $f(\text{state}) \rightarrow \text{view}$
 - 组合和渲染层隔离
 - $\text{render runtime}(\text{渲染引擎sdk}) + \text{dsl}(\text{json}) = \text{页面}$
- 事件：DND拖拽、Mouse Event
- 画布分层技术：借鉴浏览器渲染模型，使用冒泡机制，走到所有层
 - 第一层：div，负责渲染， $\text{render}(\text{dsl}, \text{document.querySelector}(\text{"\#root"}))$
 - 第二层：加一层div，只负责处理右键事件
 - 第三层：加一层div，只处理快捷键
 - event bus：所有层可通过 event bus 进行通信
- 画布功能及拓展：
 - 简单编排
 - 物料均为块级，操作简便，不涉及复杂布局问题
 - 画布灵活编排
 - 无限画布、引导线（衫格）、吸附对齐、旋转、快捷键、右键、缩放

- 无限画布：监听滚动事件，每次给画布加宽带
- 引导线：使用div画线（高度和宽带为1px）、绝对定位可拖动，下方
- 吸附对齐：计算想尽的dom节点，定6个点和3个线，距离相近时，设置距离为0

4.3.3 配置面板

配置面板就是对物料进行配置，我们需要将数据、视图、操作解耦

- 功能：撤销、重做、预览、提测、发布
 - 重做（undo/redo）：使用队列，指针移动
 - 预览：render(dsl: {type, key, props {}, animate {}, actions: {}, attrs: {}, children []})
- 中间层：权限控制、数据操作（转换），暴露部分api

这块比较考验大家对于复杂表单的设计实现

4.3.4 渲染器与出码

通常平台会提供 schema 渲染页面的功能，同时也可将所有配置输出 JSON Schema，我们称为**出码**功能

4.3.5 扩展功能与进阶（这部分内容）

历史记录和版本、模版、分享、主题

进阶：协同编辑、定时任务、微前端（集成到其他系统的能力）、混合开发

- 混合开发：组件（ts、flow）和 json 混合开发
- 逻辑的配置：使用流程图（flow），最后生成业务逻辑
- 协同：crdt算法，yjs、ot 等方案

5. Vue 项目基础架构设计，基于 Vite、Pinia、Vue-Router

5.1 技术选型

- 包管理：pnpm、npm、yarn、cnpm
- 工程化相关
 - vite
 - lint-staged
 - cspell
 - commitizen
 - cz-git

- husky
- zx
- tsno
- lint规范：
 - commitlint;
 - stylelint;
 - prettier;
 - eslint;
 - editorconfig;
- Vue CLI、Vue3
- Pinia
- Vue-Router
- 拖拽库：基于原生的 smooth-dnd 封装用于 Vue3 的拖拽组件（实战课录播第二节）
- 编辑器：tiptap for vue3, <https://tiptap.dev/installation/vue3>
- UI 库：Arco Design, <https://arco.design/vue/docs/start>
- 表单校验：vee-validate, <https://vee-validate.logaretm.com/v4/>
- 流程编排：@vue-flow/core, <https://vueflow.dev/>
- 图表：echarts

5.2 项目初始化

我们使用 `create-vue` 初始化基础结构，在此基础上设计整个项目架构。

```
npm init vue@latest  
  
# 推荐  
pnpm create vue@latest
```

5.3 规范约定

如果你作为前端 Leader 第一需要考虑的就是开发实现成本，但是往往每个成员开发的方式各异，因此我们在项目早期就一定需要做足规范方面的约束。

通常规范约束包含：

- 代码相关：js/ts/style 编码规范

- 命名与单词拼写规范
- git 提交规范等

基于此，你需要设计完整的规范约定，并在编码和代码提交时做检测，检测不通过不允许代码提交。

5.4 规范约束

代码相关检查我们使用 eslint 约束 ts 编码，stylelint 约束样式相关。

eslint 项目脚手架已经集成我们不做过多介绍，stylelint 的使用比较简单：

5.4.1 安装 stylelint

修改 `package.json`

```
"script": {
  "lint:style": "stylelint **/*.{vue,css}",
},
"devDependencies": {
  "stylelint": "15.10.2",
  "stylelint-config-standard": "34.0.0",
  "stylelint-config-prettier": "9.0.5",
  "stylelint-config-html": "1.1.0",
  "stylelint-config-vue": "1.0.0",
}
```

5.4.2 安装 cspell

```
"script": {
  "spellcheck": "cspell lint --dot --gitignore --color --cache --show-suggestions \"src/**/*.@(html|js|cjs|mjs|ts|tsx|css|scss|md|vue)\",
},
"devDependencies": {
  "cspell": "6.31.2",
}
```

配置 cspell，需要在项目根目录创建 `cspell.json`

```
{
  "import": ["@cspell/dict-lorem-ipsum/cspell-ext.json"],
  "caseSensitive": false,
  "dictionaries": ["custom-words"],
  "dictionaryDefinitions": [
```

```
{
  "name": "custom-words",
  "path": "./.cspell/custom-words.txt",
  "addWords": true
},
"ignorePaths": ["**/node_modules/**", "**/dist/**", "**/lib/**",
"**/docs/**", "**/stats.html"]
}
```

并在项目根目录创建 `.cspell/custom-words.txt`，把那些你主观认为是对的单词放进去，比如：

```
behaviour
Byelide
commitlint
conventionalcommits
optimizelegibility
pinia
tiptap
vuedraggable
vuejs
```

最后执行 `pnpm i`

这时，你就可以在本地运行对应 script 进行检测了，例如：

`pnpm spellcheck` 、 `pnpm lint:style`

5.4.3 commit 检测

5.4.3.1 husky 相关

为了在提交时进行代码检测，我们就需要使用 git 提交钩子进行处理，方便起见我们一般使用 `husky`，同时我们需要使用 `commitlint` 以及 `lint-stage` 相关来配置如何检测。

在 `package.json` 中配置相关依赖

```
{
  "script": {
    "prepare": "husky install",
    "lint:stage": "lint-staged",
    "commit": "cz-git",
    "commitlint": "commitlint --edit"
  },
}
```

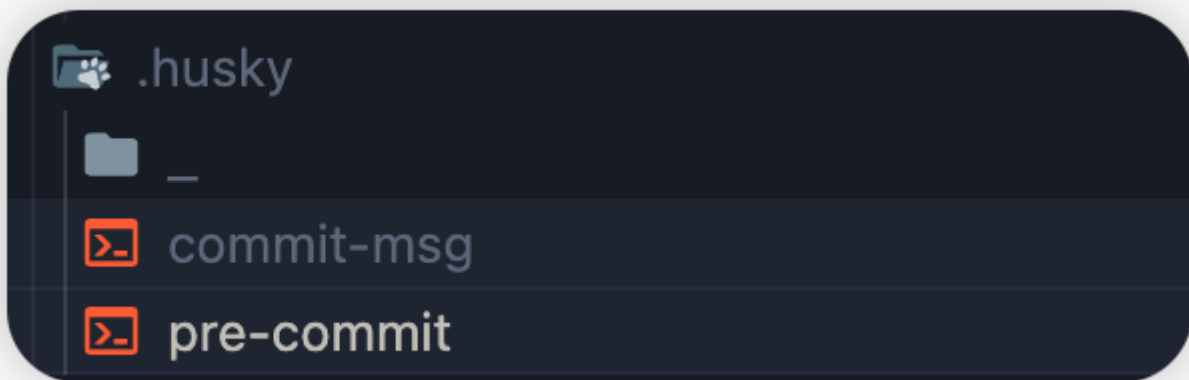


```

"lint-staged": {
  "*": "prettier --write",
  "*.{vue,ts}": "eslint --fix",
  "*.{html,vue,css,sass,scss}": "stylelint --fix"
},
"config": {
  "commitizen": {
    "path": "node_modules/cz-git"
  }
},
"devDependencies": {
  "@commitlint/cli": "17.6.7",
  "@commitlint/config-conventional": "17.6.7",
  "@commitlint/cz-commitlint": "17.6.7",
  "husky": "8.0.3",
  "lint-staged": "13.2.3",
}
}

```

执行 `pnpm prepare` 就会在根目录生成 `.husky`，钩子相关的内容写在这里。



为了方便大家学习，我选用了最简单让大家写脚本的方案，我们都知道一般我们会选择使用 shell 脚本，但是很多同学没有学习过，我们可以选择使用 `zx` 它能让我们写 js 代码以执行脚本相关内容，比如上图看到的 `pre-commit`，我们是这样：

```

#!/usr/bin/env sh
. "$(dirname "$0")/_/husky.sh"

pnpm exec tsno run ./scripts/pre-commit.ts

```

在项目根目录下创建 `scripts/xxx` 具体脚本的内容我们在 `scripts/pre-commit.ts` 中编写。

```
#!/usr/bin/env zx

import { $ } from 'zx'

console.log('开始执行代码质量评估...\n')

await import('./check').catch((out) => {
  throw new Error('代码质量评估失败, 请检查代码')
})

console.log('printf "检测通过, 创建 commit 中...\n')

await $`git add .`
```

`./check.ts`

```
#!/usr/bin/env zx

import type { ProcessOutput } from 'zx'
import { $ } from 'zx'
import { printObject } from './utils'

await $`pnpm spellcheck`.catch((out: ProcessOutput) => {
  console.log(out)

  throw new Error(out.stdout)
})

// await Promise.all([$`pnpm type-check`, $`pnpm lint`]).catch((out:
ProcessOutput) => {
//   printObject(out)
//   throw new Error(out.stdout)
// })

// check type and stage
await Promise.all([$`pnpm type-check`, $`pnpm lint:stage`]).catch((out:
ProcessOutput) => {
  printObject(out)
  throw new Error(out.stdout)
})
```

./utils

```
import { ProcessOutput } from 'zx/core'

export function printObject(
  object: Record<string, unknown> | ProcessOutput,
  method: 'log' | 'warn' | 'error' = 'log'
) {
  for (const [key, value] of Object.entries(object)) {
    // eslint-disable-next-line no-console
    console[method](`${key}: \n${value} \n`)
  }
}
```

5.4.3.2 commitlint 相关

另外, commitlint 及 git-cz 相关的配置我们需要在根目录创建 `commitlint.config.js`

```
// module.exports = { extends: ['@commitlint/config-conventional'] }

// eslint-disable-next-line no-undef
module.exports = {
  extends: ['@commitlint/config-conventional'], // extends can be nested
  parserPreset: 'conventional-changelog-conventionalcommits',
  prompt: {
    settings: {},
    messages: {
      skip: ':skip',
      max: 'upper %d chars',
      min: '%d chars at least',
      emptyWarning: 'can not be empty',
      upperLimitWarning: 'over limit',
      lowerLimitWarning: 'below limit'
    },
    types: [
      { value: 'feat', name: 'feat: ✨ A new feature', emoji: '✨' },
      { value: 'fix', name: 'fix: 🐛 A bug fix', emoji: '🐛' },
      { value: 'docs', name: 'docs: 📖 Documentation only changes', emoji: '📖' },
      {
        value: 'style',

```

```

    name: 'style:  Changes that do not affect the meaning of the
code',
    emoji: ' '
  },
  {
    value: 'refactor',
    name: 'refactor:  A code change that neither fixes a bug nor adds
a feature',
    emoji: ' '
  },
  {
    value: 'perf',
    name: 'perf:  A code change that improves performance',
    emoji: ' '
  },
  {
    value: 'test',
    name: 'test:  Adding missing tests or correcting existing
tests',
    emoji: ' '
  },
  {
    value: 'build',
    name: 'build:  Changes that affect the build system or external
dependencies',
    emoji: ' '
  },
  {
    value: 'ci',
    name: 'ci:  Changes to our CI configuration files and
scripts',
    emoji: ' '
  },
  {
    value: 'chore',
    name: 'chore:  Other changes that don't modify src or test
files",
    emoji: ' '
  },
  { value: 'revert', name: 'revert:  Reverts a previous commit',
    emoji: ':rewind:' }
  ],
  useEmoji: true,
  confirmColorize: true,
  emojiAlign: 'center',
  questions: {
    scope: {

```

```

    description: 'What is the scope of this change (e.g. component or file
name)'
  },
  subject: {
    description: 'Write a short, imperative tense description of the
change'
  },
  body: {
    description: 'Provide a longer description of the change'
  },
  isBreaking: {
    description: 'Are there any breaking changes?'
  },
  breakingBody: {
    description:
      'A BREAKING CHANGE commit requires a body. Please enter a longer
description of the commit itself'
  },
  breaking: {
    description: 'Describe the breaking changes'
  },
  isIssueAffected: {
    description: 'Does this change affect any open issues?'
  },
  issuesBody: {
    description:
      'If issues are closed, the commit requires a body. Please enter a
longer description of the commit itself'
  },
  issues: {
    description: 'Add issue references (e.g. "fix #123", "re #123".)'
  }
}
}
}

```

5.4.3.3 pnpm commit

通过上面配置，以后的提交都通过这个命令就搞定了。

6. 构建基础框架（导航、物料、配置）

Vue3 项目基础框架我们可以像如下形式组织，不过可以根据自己项目实际情况调整。

byelide

```
├─.DS_Store
├─.cspell
│   └─custom-words.txt
├─.cspellcache
├─.eslintrc.cjs
├─.husky
│   └─
│       └─husky.sh
│       └─commit-msg
│       └─pre-commit
├─.prettierignore
├─.prettierrc.json
├─.vite
├─.vscode
│   └─extensions.json
├─README.md
├─commitlint.config.js
├─cspell.json
├─cypress
├─cypress.config.ts
├─env.d.ts
├─index.html
├─package.json
├─pnpm-lock.yaml
├─public
│   └─favicon.ico
├─scripts
│   └─check.ts
│   └─pre-commit.ts
│   └─utils.ts
├─src
│   └─.DS_Store
│   └─App.vue
│   └─assets
│       └─base.css
│       └─logo.svg
│       └─main.css
│       └─reset.css
│       └─variable.css
│   └─blocks
│   └─components
│   └─constants
│       └─blocksBaseMeta.ts
│   └─main.ts
│   └─mocks
│       └─blocks.ts
```

```
|   |├router
|   |   └index.ts
|   |├setup.ts
|   |├stores
|   |   └appEditor.ts
|   |       └debug.ts
|   |├test.ts
|   |├types
|   |   └block.ts
|   |       └protocal
|   |├utils
|   |   └array.ts
|   └views
├stylelint.config.js
├tsconfig.app.json
├tsconfig.json
├tsconfig.node.json
├tsconfig.vitest.json
├vite.config.ts
└vitest.config.ts
```

以上代码有 etree 生成，还不知道的同学可以试试

6.1 主体框架

因为我们是单体应用，所以项目主题内容我们可以从路由 **router** 出发，路由决定了页面入口，路由分发到页面，通常与路由对应的页面我们存放在 **views** 中，组件相关内容在 **components** 内。

细心的同学可以发现，我们整个布局其实有很多细节，比如为什么页面外层容器需要 `position: fixed` 定位？

为什么边框我们选择使用 `box-shadow` 而不是直接使用 `border` ？

6.2 导航

```
<script setup lang="ts">
import { useEnvStore } from '@stores/debug'
import { Data, LayoutThree, Lightning, Share, Bug } from '@icon-park/vue-next'
import { computed, defineComponent, h } from 'vue'
import { useRoute } from 'vue-router'

const linkItems = [
  {
    value: 'dataSource',
```

```

    label: '数据源',
    bg: `radial-gradient(50% 50% at 50% 100%, rgba(0, 196, 83, 0.2) 0%,
    rgba(0, 196, 83, 0) 100%)`,
    color: 'rgb(0, 196, 83)',
    borderColor: 'radial-gradient(50% 50%, rgb(0, 196, 83) 0%, rgba(0, 196,
    83, 0) 100%)'
  },
  {
    value: 'layout',
    label: '布局',
    bg: `radial-gradient(50% 50% at 50% 100%, rgba(24, 190, 212, 0.2) 0%,
    rgba(24, 190, 212, 0) 100%)`,
    color: 'rgb(24, 190, 212)',
    borderColor: 'radial-gradient(50% 50%, rgb(24, 190, 212) 0%, rgba(24, 190,
    212, 0) 100%)'
  },
  {
    value: 'actions',
    label: '动作',
    bg: `radial-gradient(50% 50% at 50% 100%, rgba(241, 60, 11, 0.2) 0%,
    rgba(241, 60, 11, 0) 100%)`,
    color: 'rgb(241, 60, 11)',
    borderColor: 'radial-gradient(50% 50%, rgb(241, 60, 11) 0%, rgba(241, 60,
    11, 0) 100%)'
  }
]

```

```

defineProps<{
  msg: string
}>()

```

```

// 千万不要这样!
// const { debug, toggle } = useEnvStore()
const envStore = useEnvStore()

```

```

const route = useRoute()
console.log('🚀 ~ file: AppNavigator.vue:35 ~ route:',
JSON.parse(JSON.stringify(route)))

```

```

const activeLink = computed(() => route.name)

```

```

// 等价于 computed
// const activeLink = ref(route.path.slice(1))
// watch(
//   () => route.path,
//   (path) => {
//     console.log('🚀 ~ file: AppNavigator.vue:20 ~ path:', path)
//   }
// )

```



```
//    activeLink.value = path.slice(1)
//  }
// )
```

```
const Icon = defineComponent({
  setup(props) {
    // 千万不能这样子写!!!
    // const { type } = props
    switch (props.type) {
      case 'dataSource':
        return () => h(Data, { size: 16 })
      case 'layout':
        return () => h(LayoutThree, { size: 16 })
      case 'actions':
        return () => h(Lightning, { size: 16 })

      default:
        return () => h('div')
    }
  },
  props: {
    type: {
      type: String,
      required: true
    }
  }
})
</script>
```

```
<template>
  <div class="app-navigator">
    <div class="app-info-wrapper">
      <div class="app-logo">
        
      </div>
      <h1 class="app-name">Byelide</h1>
    </div>
    <div class="app-navigator-link-wrapper">
      <RouterLink
        class="app-navigator-link-item"
        v-for="item in linkItems"
        :key="item.value"
        :style="activeLink === item.value && { background: item.bg }">
```

```

      :to="item.value"
    >
      <!-- defineComponent + h 代替条件渲染 -->
      <!-- <div v-if="item.value === 'dataSource'"><Data /></div>
      <div v-if="item.value === 'layout'"><LayoutThree /></div>
      <div v-if="item.value === 'action'"><Lightning /></div> -->
      <div
        :style="{
          lineHeight: 0.7,
          color: activeLink === item.value ? item.color : 'var(--color-gray-
700)'
        }"
      >
        <Icon :type="item.value" :active="activeLink === item.value" />
      </div>
      <span class="item-title">
        {{ item.label }}
      </span>
      <div
        class="item-border"
        :style="activeLink === item.value ? { background: item.borderColor }
: {}"
      ></div>
    </RouterLink>
  </div>
  <div class="app-setting-wrapper">
    <div class="common-btn debug-btn" :class="{ debug: envStore.debug }"
@click="envStore.toggle">
      <Bug />
      开发模式:({{ envStore.debug ? '开' : '关' }})
    </div>
    <div class="common-btn">
      <Share />
      发布
    </div>
  </div>
</div>
</template>

<style scoped>
.app-navigator {
  position: relative;
  z-index: 5;
  display: flex;
  justify-content: space-between;
  align-items: center;
  height: 48px;

```

```
    box-shadow: rgb(0 0 0 / 8%) 0 1px 0;
}

.app-info-wrapper {
  width: 1000px;
  display: flex;
  place-items: center;
}

.app-logo {
  width: 32px;
  height: 32px;
  margin: 10px 8px 10px 18px;
  border-radius: 8px;
  background-color: var(--color-primary);
}

.app-logo img {
  width: 100%;
  height: 100%;
  padding: 6px;
}

.app-name {
  color: var(--color-gray-900);
  font-weight: var(--font-weight-bolder);
  font-size: var(--font-size-large);
}

.app-navigator-link-wrapper {
  display: flex;
  flex-shrink: 0;
  justify-content: space-between;
  align-self: stretch;
}

.app-navigator-link-item {
  position: relative;
  display: flex;
  align-items: center;
  padding: 0 8px;
  margin: 0 24px;
  align-self: stretch;
  background-color: var(--color-white);
  color: var(--color-text);
  font-size: var(--font-size-normal);
  font-weight: var(--font-weight-bold);
}
```

```
}

.item-title {
  margin-left: 8px;
  color: var(--color-black);
}

.item-border {
  position: absolute;
  bottom: -1px;
  left: 0;
  right: 0;
  height: 1px;
}

.app-setting-wrapper {
  width: 1000px;
  display: flex;
  place-items: center;
  justify-content: flex-end;
  margin-right: 18px;
  gap: 12px;
}

.common-btn {
  display: flex;
  align-items: center;
  gap: 8px;
  padding: 6px 18px;
  border-radius: 8px;
  background-color: var(--color-black);
  color: var(--color-white);
  font-size: var(--font-size-normal);
  font-weight: var(--font-weight-bold);
  cursor: pointer;
  user-select: none;
}

.debug-btn {
  box-shadow: var(--color-gray-300) 0 0 0 1px;
  background-color: var(--color-white);
  color: var(--color-black);
}

.debug-btn.debug {
  color: var(--color-primary);
  box-shadow: var(--color-primary) 0 0 0 1px;
```

```
}  
</style>
```

这块对于 `h` 运用以及如何将 Vue 组件策略渲染外部组件需要大家重点学习掌握。

6.3 物料

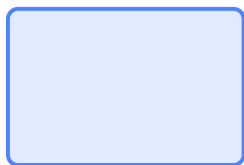
物料的处理主要是涉及到了布局以及拖拽，具体实现我们会在下次课为大家讲解拖拽库的封装实现。

7. 编排的选型与实现（流式、画布式、Grid）

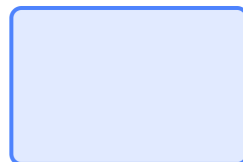
编排的实现方式有很多，我们可以选择基于画布式的自由拖拽、可以选择流式布局或者 grid，我们本次实战课选用的是流式。

这三种方式布局会考虑的技术给大家列举一下，可作为参考。

- 流式
 - smooth-dnd (<https://github.com/kutlugsahin/vue-smooth-dnd?tab=readme-ov-file>)
 - vue.dragabble.next (<https://sortablejs.github.io/vue.draggable.next/#/simple>)
- 自由拖拽 (<https://kirillmurashov.com/vue-drag-resize/>) **刻意练习**
- grid (<https://jbaysolutions.github.io/vue-grid-layout/>)



阿斯顿法师法



7.1 需求层面需要评判出来的优劣势

流式布局

- 没有脱离文档流，**自适应宽高**
- 缺点也很明显：不是平铺模式，后期可能存在比较严重的性能问题，另外对于内层数据的处理不方便。

```
{
  type: 'text',
  children: [
    {
      type: 'text',
      children: [
        type: 'text'
      ]
    }
  ]
}
```

自由拖拽，

- 脱离文档流了，布局更灵活
- 无法自适应宽高，如果实在需要自适应宽高，需要做非常复杂的处理

Grid

- 比较折中的办法，有一定的灵活性有一定的规范
- 无法自适应宽高

通过上面这些思考，我们最终确定了使用**流式布局**，接下来，问题又来了

- smooth-dnd (<https://vueflow.dev/>)
- vue.dragabble.next (<https://sortablejs.github.io/vue.draggable.next/#/simple>)

7.2 基于 smooth-dnd 的封装

因为 `smooth-dnd` 是原生js库，我们需要进行一定封装才能在 Vue3 项目中使用。

我们创建一个 SmoothDnd 组件，通常拖拽实现需要实现两个组件，分别为：Container 和 DraggableItem，我们分别定义两个组件，命名为 `SmoothDndContainer.ts` 和 `SmoothDndDraggable.ts`。

`SmoothDndContainer.ts`

```

import { defineComponent, h } from 'vue'
import { smoothDnD, dropHandlers } from 'smooth-dnd'
import type { SmoothDnD } from 'smooth-dnd'
import { getTagProps, validateTagProp } from './utils'

smoothDnD.dropHandler = dropHandlers.reactDropHandler().handler
smoothDnD.wrapChild = false

type EventKey = 'drag-start' | 'drag-end' | 'drop' | 'drag-enter' | 'drag-leave' | 'drop-ready'

const eventEmitterMap: Record<EventKey, string> = {
  'drag-start': 'onDragStart',
  'drag-end': 'onDragEnd',
  drop: 'onDrop',
  'drag-enter': 'onDragEnter',
  'drag-leave': 'onDragLeave',
  'drop-ready': 'onDropReady'
}

export const SmoothDndContainer = defineComponent({
  name: 'SmoothDndContainer',
  setup() {
    return {
      container: null as SmoothDnD | null
    }
  },
  mounted() {
    // emit events
    const options: any = Object.assign({}, this.$props)
    for (const key in eventEmitterMap) {
      const eventKey = key as EventKey
      options[eventEmitterMap[eventKey]] = (props: any) => {
        this.$emit(eventKey, props)
      }
    }
    const containerElement = this.$refs.container || this.$el
    this.container = smoothDnD(containerElement, options)
  },
  unmounted() {
    if (this.container) {
      try {
        this.container.dispose()
      } catch {
        // ignore
      }
    }
  }
})

```

```

    }
  },
  emits: ['drop', 'drag-start', 'drag-end', 'drag-enter', 'drag-leave', 'drop-
ready'],
  props: {
    orientation: { type: String, default: 'vertical' },
    removeOnDropOut: { type: Boolean, default: false },
    autoScrollEnabled: { type: Boolean, default: true },
    animationDuration: { type: Number, default: 250 },
    behaviour: String,
    groupName: String,
    dragHandleSelector: String,
    nonDragAreaSelector: String,
    lockAxis: String,
    dragClass: String,
    dropClass: String,
    dragBeginDelay: Number,
    getChildPayload: Function,
    shouldAnimateDrop: Function,
    shouldAcceptDrop: Function,
    getGhostParent: Function,
    dropPlaceholder: [Object, Boolean],
    tag: {
      validator: validateTagProp,
      default: 'div'
    }
  },
  render() {
    const tagProps = getTagProps(this)
    return h(
      tagProps.value,
      Object.assign({}, { ref: 'container' }, tagProps.props),
      this.$slots.default?.( )
    )
  }
})

```

SmoothDndDraggable.ts

```

import { defineComponent, h } from 'vue'
import { constants } from 'smooth-dnd'
import { getTagProps, validateTagProp } from './utils'

export const SmoothDndDraggable = defineComponent({

```



```
name: 'SmoothDndDraggable',
props: {
  tag: {
    validator: validateTagProp,
    default: 'div'
  }
},
render: function () {
  //wrap child
  const tagProps = getTagProps(this, constants.wrapperClass)
  return h(tagProps.value, Object.assign({}, tagProps.props),
    this.$slots?.default?.())
}
```

8. 课程总结

本次课我们重点为大家介绍了无代码平台相关需求、Vue3 项目从零到一的架构思想，另外在技术选型方面我们都需要如何思考，最后带大家整体实现了无代码平台基础框架包含导航、物料以及相关配置。

希望大家课后，反复多学习思考，重点学习项目设计的思想，下次换个其他项目你也有能力完全主导设计开发，这样你的核心竞争力才能提高。

共勉！

其他资料

 <https://form.making.link/>

可视化低代码表单设计器 - FormMaking

基于 Vue 的可视化低代码表单设计器，赋能企业实现低代码开发模式；帮助开发者从传统枯燥的表单代码中解放出来，更多关注业务，快速提高效率，节省研发成本。在OA系统、考试系统、报表系统、流程管理等系统中应用广泛。



<https://github.com/alibaba/lowcode-engine>

GitHub - alibaba/lowcode-engine: An enterprise-class low-code technology stack with scale-out design

An enterprise-class low-code technology stack with scale-out design / 一套面向...

sahadev/**vue-component-creator-ui**

拖拽式Vue组件代码生成编辑器（VCC）



<https://github.com/sahadev/vue-component-creator-ui>

GitHub - sahadev/vue-component-creator-ui: 拖拽式Vue组件代码生成编辑器（VCC）

2 Contributors 5 Issues 2 Discussions 665 Stars 195 Forks

拖拽式Vue组件代码生成编辑器（VCC） . Contribute to sahadev/vue-component-...