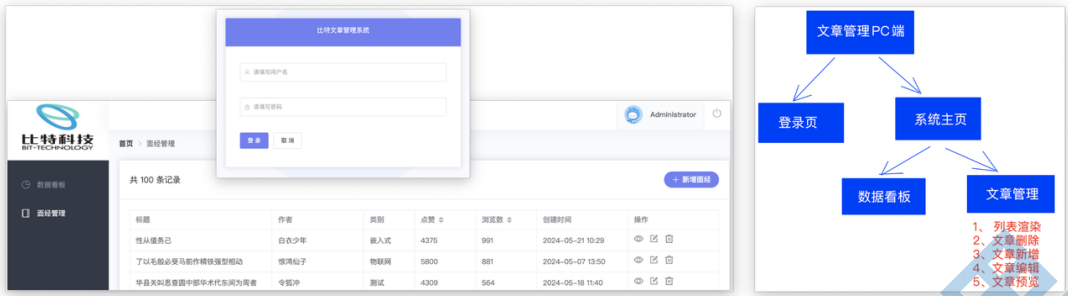


11-Vue3全家桶+ElementPlus实战

一、项目演示及收获

1. 效果演示



2. 项目收获

脚手架自定义项目构建	组件库 elementUI (全部&按需导入)	elementUI 主题色定制
request & storage & api 模块封装	嵌套路由 & 导航守卫 & 拦截器	深度作用选择器 - 定制样式
elementUI 表单校验	Pinia 分模块存储	模块化开发、组件化开发
elementUI 表格 & 分页渲染	富文本编辑器编辑 & 预览	文章管理 (增删改查)
抽离封装常量模块	优秀的编码思路	优秀的代码风格

...

3. 项目技术栈

vue3全家桶(vue3 + vue-router + pinia) + vite + element-plus + axios + es6+(体系) + scss ...

二、创建项目

1. 选定项目创建位置

2. 打开终端(命令行)窗口

按下图操作:

```
C:\Users\13522\Desktop>npm create vue@latest
> npx
> create-vue

Vue.js - The Progressive JavaScript Framework

/ 请输入项目名称: ... bit-article-sys
/ 是否使用 TypeScript 语法? ... 否 / 是
/ 是否启用 JSX 支持? ... 否 / 是
/ 是否引入 Vue Router 进行单页面应用开发? ... 否 / 是
/ 是否引入 Pinia 用于状态管理? ... 否 / 是
/ 是否引入 Vitest 用于单元测试? ... 否 / 是
/ 是否要引入一放端到端 (End to End) 测试工具? ... 不需要
/ 是否引入 ESLint 用于代码质量检测? ... 否 / 是
/ 是否引入 Vue DevTools 7 扩展用于调试? (试验阶段) ... 否 / 是

正在初始化项目 C:\Users\13522\Desktop\bit-article-sys...

项目初始化完成, 可执行以下命令:

cd bit-article-sys
npm install
npm run dev
```

3. 运行项目

3.1 进入项目

```
1 cd bit-article-sys
```

3.2 安装依赖

```
1 npm i
```

3.3 启动项目

```
1 npm run dev
```

4. 预览

浏览器打开: <http://localhost:5173>

三、认识并调整目录

脚手架默认创建的目录结构有些不满足我们的开发需求, 需要做一些自定义改动。主要有:

1. 删除初始化的默认文件
2. 修改剩余代码内容
3. 新增调整我们需要的目录结构

1. 删除文件

- components 下的所有文件和目录

- views 下的所有文件
- assets 下的所有文件
- stores 下的 counter.js

2. 修改内容

src/router/index.js

```
1 import { createRouter, createWebHistory } from 'vue-router'
2
3 const router = createRouter({
4   // import.meta.env.BASE_URL: 指明路由启动后的基础路径，默认值是 /
5   history: createWebHistory(import.meta.env.BASE_URL),
6   // 配置路由规则
7   routes: [
8
9   ]
10 })
11
12 export default router
```

src/main.js

```
1 import { createApp } from 'vue'
2 import { createPinia } from 'pinia'
3
4 import App from './App.vue'
5 import router from './router'
6
7 const app = createApp(App)
8
9 app.use(createPinia())
10 app.use(router)
11
12 app.mount('#app')
```

src/App.vue

```
1 <script setup></script>
2
3 <template>
```

```
4 <!-- 一级路由出口 -->
5 <router-view />
6 </template>
7
8 <style scoped></style>
```

3. 新增目录

在 src 目录下中新建以下目录：

- /api： 存储请求函数模块
- /styles: 样式文件模块
- /utils: 工具函数模块



- 将需要的图片放置在 `assets` 目录下

四、引入Element组件库

1. 全部引入(推荐)

如果你对打包后的文件大小不是很在乎，那么使用完整导入会更方便。

1、安装

```
1 # npm
2 npm i element-plus
3
4 # yarn
5 yarn add element-plus
6
7 # pnpm
8 pnpm i element-plus
```

2、导入并注册

main.js

```
1 // 导入组件库
2 import ElementPlus from 'element-plus'
3 // 导入组件样式
4 import 'element-plus/dist/index.css'
5 // 全局注册所有组件
6 app.use(ElementPlus)
```

2. 按需引入

1、安装

```
1 # npm
2 npm i element-plus
3
4 # yarn
5 yarn add element-plus
6
7 # pnpm
8 pnpm i element-plus
```

2、安装两款插件

```
1 # npm
2 npm i unplugin-vue-components unplugin-auto-import -D
3
4 # yarn
5 yarn add unplugin-vue-components unplugin-auto-import -D
6
7 # pnpm
8 pnpm i unplugin-vue-components unplugin-auto-import -D
```

3、加入Vite配置

vite.config.js

```
1 import AutoImport from 'unplugin-auto-import/vite'
2 import Components from 'unplugin-vue-components/vite'
3 import { ElementPlusResolver } from 'unplugin-vue-components/resolvers'
4
```

```
5 export default defineConfig({// ...
6   plugins: [
7
8     // ...
9     AutoImport({
10       resolvers: [
11         ElementPlusResolver()
12       ]
13     }),
14     Components({
15       resolvers: [
16         ElementPlusResolver()
17       ]
18     })
19   ]
20 })
```

3. 主题色定制

自定义主题: <https://element-plus.org/zh-CN/guide/theming.html>

1、新建 styles/index.scss

```
1 body {
2   margin: 0;
3   background: #fafbfe;
4 }
5
6 :root {
7   // 修改 ElementPlus 样式颜色值
8   --el-color-primary: rgba(114, 124, 245, 1);
9 }
```

2、main.js 引入

```
1 import '@styles/index.scss'
```

五、公共模块封装

1. 封装request请求模块

1.1 为什么封装

我们会使用 axios 来请求后端接口, 一般会对 axios 进行一些统一配置 (比如: 基础地址等)
项目开发中, 都会对 axios 进行基本的二次封装, 单独封装到一个模块中, 便于使用

1.2 步骤

1、安装

```
1 npm i axios
```

2、新建 `utils/request.js` 请求模块

[Axios中文网](#)

```
1  /* 封装 axios 用于发送请求 */
2
3
4  // 导入 axios
5  import axios from 'axios'
6
7  // 创建一个新的axios实例
8  const instance = axios.create({
9    // 请求基础地址
10   baseURL: 'http://localhost:4000/api',
11   // 请求超时时间
12   timeout: 5000
13 })
14
15 // 请求拦截器
16 instance.interceptors.request.use((config) => {
17   // 在发送请求之前做些什么
18   return config
19 }, (error) => {
20   // 对请求错误做些什么
21   return Promise.reject(error)
22 })
23
24 // 响应拦截器
25 instance.interceptors.response.use((response) => {
26   // 对响应数据做点什么
27   return response
28 }, (error) => {
29   // 对响应错误做点什么
30   return Promise.reject(error)
31 })
```

```
32
33 // 默认导出请求实例
34 export default instance
```

2. 封装token本地管理模块

2.1 为什么封装

`token` 是用户登录的凭证, 非常重要。需要同时通过 `Pinia`和本地存储 共同保存

- 当用户登录成功后, 我们要保存 `token`;
- 当用户退出登录时, 要删除 `token`, 并且还需要通过
- 当刷新页面时, 我们要从本地获取`token`

为了便于后期使用方便, 我们封装一个对 `token` 进行本地存、取、删的公共模块

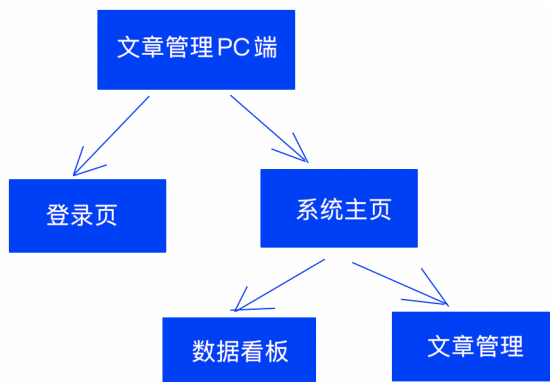
2.2 步骤

```
1 // 本地存储token的键名
2 const TOKEN_KEY = 'BIT_TOKEN_KEY'
3
4 // 存
5 export const setToken = (token) => {
6   localStorage.setItem(TOKEN_KEY, token)
7 }
8
9 // 取
10 export const getToken = () => {
11   return localStorage.getItem(TOKEN_KEY)
12 }
13
14 // 删
15 export const removeToken = () => {
16   localStorage.removeItem(TOKEN_KEY)
17 }
```

六、路由设计

1. 分析

凡是单个页面、独立展示的, 都是一级路由



整理思路: 新建页面 -> 配置规则 -> 路由出口

路由设计:

- 登录页 (一级) Login
- 首页架子 (一级) Layout
 - 数据看板 (二级) Dashboard
 - 文章管理 (二级) Article

2. 新建页面



3. 配置规则

router/index.js

```
1 import { createRouter, createWebHistory } from 'vue-router'
2
3 const router = createRouter({
4   // import.meta.env.BASE_URL: 指明路由启动后的基础路径, 默认值是 /
5   history: createWebHistory(import.meta.env.BASE_URL),
6   // 路由规则表
7   routes: [
8     {
9       // 根路径
10      path: '/',
11      // 路由懒加载: 当页面被访问的时候, 才去加载对应的文件, 从而提高首屏的加载速度
12      component: () => import('@/views/Layout/index.vue'),
```

```

13 // 重定向：防止访问根路径，页面空白
14 redirect: '/dashboard',
15 // 路由嵌套
16 children: [
17   {
18     // 数据看板
19     path: 'dashboard',
20     component: () => import('@views/Dashboard/index.vue')
21   },
22   {
23     // 文章管理
24     path: 'article',
25     component: () => import('@views/Article/index.vue')
26   }
27 ]
28 },
29 {
30   // 登录
31   path: '/login',
32   component: () => import('@views/Login/index.vue')
33 }
34 ]
35 })
36
37 export default router

```

4. 路由出口

App.vue

```

1 <script setup></script>
2 <template>
3   <!-- 一级路由出口 -->
4   <router-view />
5 </template>
6 <style lang="scss" scoped></style>

```

views/Layout/index.vue

```

1 <script setup></script>
2 <template>
3   <div>
4     <div>左侧</div>

```

```

5     <div>
6       <header>头部</header>
7       <main>
8         <!-- 二级路由出口 -->
9         <router-view />
10      </section>
11    </div>
12  </div>
13 </template>
14 <style lang="scss" scoped>
15
16 </style>

```

七、登录模块

1. 需求



2. 分析



3. 代码示例

3.1 静态标签结构

```

1 <script setup></script>
2 <template>
3   <div class="login-box">
4     <!-- 卡片组件 -->

```

```

5   <el-card>
6     <!-- 头部 -->
7     <template #header>
8       <div class="card-header">
9         <h2>比特文章管理系统</h2>
10      </div>
11    </template>
12    <!-- 主体: 表单 -->
13    <el-form>
14      <el-form-item>
15        <el-input
16          type="text"
17          clearable
18          placeholder="请填写用户名"
19          prefix-icon="User" />
20      </el-form-item>
21      <el-form-item>
22        <el-input
23          type="password"
24          clearable
25          show-password
26          placeholder="请填写密码"
27          prefix-icon="Lock" />
28      </el-form-item>
29      <el-form-item>
30        <el-button type="primary">登 录</el-button>
31        <el-button>取 消</el-button>
32      </el-form-item>
33    </el-form>
34  </el-card>
35 </div>
36 </template>

```

3.2 样式美化

```

1  <style scoped lang="scss">
2    .login-box {
3      display: flex;
4      justify-content: center;
5      align-items: center;
6      width: 100vw;
7      height: 100vh;
8      background: url('@/assets/login-bg.svg') center/cover;
9      .el-card {
10        width: 40vw;

```

```

11     height: 40vh;
12     // :deep(选择器): scss 样式穿透的语法, 什么时候需要用到样式穿透:
13     // 1. 当前组件的 style 添加了 scoped
14     // 2. 修改的是第三方组件内部元素的样式
15     :deep(.el-card__header) {
16         display: flex;
17         justify-content: center;
18         align-items: center;
19         height: 70px;
20         // 注意: var(css变量), 好处就是利于今后的代码维护和修改
21         background: var(--el-color-primary);
22         color: #fff;
23         h2 {
24             font-size: 16px;
25             font-weight: 400;
26         }
27     }
28     .el-form {
29         padding: 15px 25px;
30         .el-form-item:nth-child(2) {
31             margin: 40px 0;
32         }
33         .el-input {
34             height: 45px;
35         }
36         .el-button {
37             width: 65px;
38             height: 35px;
39         }
40     }
41 }
42 }
43 </style>

```

4. 样式控制

默认情况下, 写在 `scoped` 中的样式, 只会影响到当前组件中的元素; 在添加了 `scoped` 的情况下, 如果希望样式向下影响到(组件内部)子元素, 那么需要借助 `深层作用选择器`, 即可让样式穿透, 从而作用在子元素上

在 `scss` 中, 深层作用选择器的语法:

```

1 :deep(当前选择器) {
2   // some style code ...
3 }

```

```
1 :deep(.el-card__header) {
2   display: flex;
3   justify-content: center;
4   align-items: center;
5   height: 70px;
6   // 注意: var(css变量), 好处就是利于今后的代码维护和修改
7   background: var(--el-color-primary);
8   color: #fff;
9 }
```

5. 输入框添加小图标

5.1 安装字体模块

```
1 npm i @element-plus/icons-vue
```

5.2 导入并注册字体组件

```
1 // main.js
2
3 // 按需导入
4 import {
5   User,
6   Lock
7 } from '@element-plus/icons-vue'
8
9 // 全局注册
10 app.component(User.name, User)
11 app.component(Lock.name, Lock)
```

5.3 input 添加 `prefix-icon` 属性

```
1 <el-input prefix-icon="User" />
2
3 <el-input prefix-icon="Lock" />
```

八、表单校验

1. 说明

在向后端发请求、调用接口之前, 我们需要对表单进行验证, 把用户的错误扼杀在摇篮之中。

2. 步骤

Form 表单组件提供了数据验证的功能

Form 表单组件的校验:

1. el-form: 绑定 `model` 表单对象 和 `rules` 验证规则
2. el-form-item: 绑定 `prop` 属性
3. el-input: 绑定 `v-model`

2.1 定义 model 和 rules 并绑定

```
1 import { reactive } from 'vue'
2
3 // 登录表单
4 const loginForm = reactive({
5   username: '', // 用户名
6   password: '' // 密码
7 })
8
9 // 校验规则
10 const loginFormRules = {
11   username: [
12     { required: true, message: '用户名不能为空', trigger: 'blur' },
13     { min: 3, max: 10, message: '长度在 3 到 10 个字符', trigger: 'blur' }
14   ],
15   password: [
16     { required: true, message: '密码不能为空', trigger: 'blur' },
17     { min: 5, max: 20, message: '长度在 5 到 20 个字符', trigger: 'blur' }
18   ]
19 }
```

```
1 <el-form
2   :model="loginForm"
3   :rules="loginFormRules"
4 >
5
6 </el-form>
```

2.2 form-item 绑定 prop 属性

将 form-item 的 prop 属性设置为需校验的字

```
1 <el-form-item prop="username"></el-form-item>
2
3 <el-form-item prop="password"></el-form-item>
```

2.3 input 进行 v-model 双向绑定

```
1 <el-input v-model="loginForm.username" />
2
3 <el-input v-model="loginForm.password" />
```

九、提交表单校验 和 重置

1. 说明

兜底校验: 每次点击登录按钮, 进行 ajax 请求前, 应该先对整个表单内容校验, 不然会发送很多无效的请求!!!

要保证通过校验了, 才能发送请求!!!

2. 步骤

- 校验: 获取 Form 组件实例, 调用 validate() 方法
- 重置: 获取 Form 组件实例, 调用 resetFields() 方法

2.1 准备 ref 并绑定

```
1 <script setup>
2   import { ref } from 'vue'
3
4   const loginFormRef = ref(null)
5 </script>
6
7 <template>
8   <el-form ref="loginFormRef">
9     ...
10  </el-form>
11 </template>
```


2.2 按钮绑定事件并处理

```
1 <script setup>
2   // 登录
3   const onLogin = () => {
4     // 整体校验
5     loginFormRef.value.validate((valid) => {
6       if (!valid) return
7       console.log('OK')
8     })
9   }
10
11  // 重置
12  const onCancel = () => {
13    loginFormRef.value.resetFields()
14  }
15 </script>
16 <template>
17   <el-button type="primary" @click="onLogin">登 录</el-button>
18   <el-button @click="onCancel">取 消</el-button>
19 </template>
```

十、封装登录请求

1. 封装接口并导出

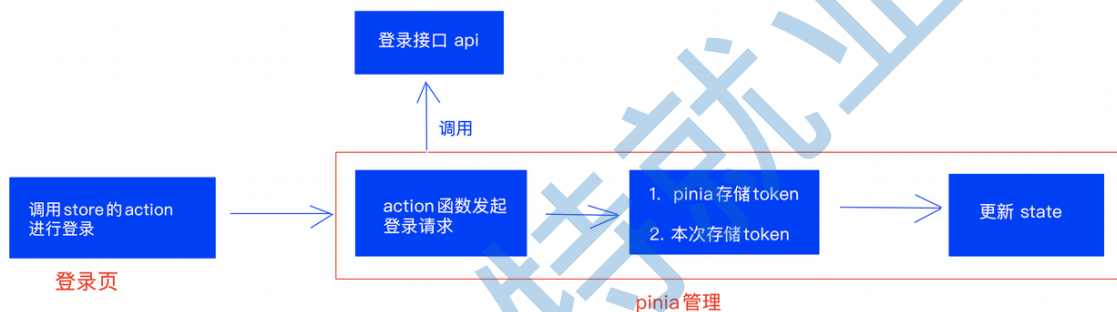
新建 `api/user.js` 模块, 提供登录接口

```
1 // 导入公共请求函数
2 import request from '@/utils/request'
3
4 // 定义并导出登录接口函数
5 export function loginApi(loginForm) {
6   // 注意: 这里 return 不能丢
7   return request({
8     method: 'post',
9     url: '/login',
10    data: loginForm
11  })
12 }
```

2. 登录页测试接口

```
1 import { loginApi } from '@api/user'
2
3 // 登录
4 const onLogin = () => {
5   loginFormRef.value.validate(async (valid) => {
6     if (!valid) return
7
8     // 调用接口
9     const resp = await loginApi(loginForm)
10    console.log(resp)
11  })
12 }
```

十一、Pinia 管理 token



1. 新建 userStore 模块

stores/user.js

```
1 import { defineStore } from 'pinia'
2 import { loginApi } from '@api/user'
3 import { getToken, setToken } from '@utils/token'
4
5 // 定义并导出用户 store
6 export const useUserStore = defineStore('user', {
7   state: () => ({
8     // token
9     token: getToken() || '',
10    // 用户信息
11    userInfo: {}
12  }),
13   actions: {
14     setToken(token) {
```

```

15     this.token = token
16     // token 存本地
17     setToken(token)
18   },
19   // 登录
20   async loginAction(loginForm) {
21     // 发起登录请求(调用登录接口)
22     // await 只能获取成功态的 Promise 实例结果
23     const resp = await loginApi(loginForm)
24     // 解构出 token
25     const { token } = resp
26     // 对 token 赋值
27     this.setToken(token)
28   }
29 }
30 })

```

2. 调用登录 action

Login/index.vue

```

1 <script setup>
2 // 导入用户仓库模块
3 import { useUserStore } from '@store/user'
4 // 得到用户仓库
5 const userStore = useUserStore()
6
7 // 登录
8 const onLogin = () => {
9   loginFormRef.value.validate((valid) => {
10     if (!valid) return
11     // 调用登录 action
12     userStore.loginAction(loginForm)
13   })
14 }

```

3. 登录后续操作

```

1 <script setup>
2 import { useRouter } from 'vue-router'
3 import { ElMessage } from 'element-plus'
4 const router = useRouter()
5

```

```

6 // 登录
7 const onLogin = () => {
8   loginFormRef.value.validate(async (valid) => {
9     if (!valid) return
10    try {
11      // 这里必须添加 await 的原因是 loginAction 是异步的,
12      // 需要等该函数执行完, 才能确定的登录成功还是失败
13      await userStore.loginAction(loginForm)
14      // 成功提示
15      ElMessage.success('登录成功')
16      // 跳转至首页
17      router.push('/')
18    } catch (e) {
19      // 错误提示
20      ElMessage.error(e.message)
21    }
22  })
23 }
24 </script>

```

十二、axios 响应成功拦截

1. 说明

为了后续请求获取数据方便, 因为所有的响应成功了, 都会经过 axios 响应拦截器的成功回调, 因此可以在这里对成功的数据做统一处理

2. 代码示例

utils/request.js

```

1 // 自定义响应状态码映射对象
2 const RESPONSE_CODE = {
3   success: 10000, // 成功
4 }
5 // 响应拦截器
6 instance.interceptors.response.use(
7   (response) => {
8     // 对响应成功做公共处理, 响应状态码以2开头
9
10    // 解构出 data
11    const { data: resp } = response
12    // 解构出 code, data, message
13    const { code, data, message } = resp
14    // 响应成功

```

```
15     if (code === RESPONSE_CODE.success) {
16         // 直接返回纯粹的数据
17         return data
18     }
19     // 失败
20     // 返回失败态的Promise实例，并携带错误信息
21     return Promise.reject(new Error(message))
22 }, (error) => {
23     return Promise.reject(error)
24 })
```

十三、登录访问拦截

1. 说明

在没有登录的情况下, 如果要访问的不是登录页, 则先去登录; 否则放行

2. 代码示例

router/index.js

```
1 // 导入用户仓库
2 import useUserStore from '@/store/user'
3
4 // 白名单: 无需登录就可以访问的路由路径
5 const whiteList = ['/login']
6
7 // 全局前置守卫
8 router.beforeEach((to, from) => {
9     const userStore = useUserStore()
10    // 从 pinia 中获取 token
11    const { token } = userStore
12    if (token) {
13        // 登录了
14        // 还想去登录
15        if (to.path === '/login') {
16            // 直接重定向到首页
17            return '/'
18        } else {
19            // 放行
20            return true
21        }
22    } else {
23        // 未登录
24        if (whiteList.includes(to.path)) {
```

```

25     // 放行
26     return true
27   } else {
28     // 重定向到登录，并且携带回跳地址
29     return `/login?redirectUrl=${to.path}`
30   }
31 }
32 })

```

3. 完善登录跳转

views/Login/index.vue

```

1  import { useRoute } from 'vue-router'
2  const route = useRoute()
3
4  // 登录
5  const onLogin = () => {
6    // 整体校验
7    loginFormRef.value.validate(async (valid) => {
8      if (!valid) return ElMessage.error('用户名或密码不对')
9      try {
10        // 发起登录请求
11        await userStore.loginAction(loginForm)
12        // 成功的提示
13        ElMessage.success('登录成功')
14        // 登录成功后：优先跳转到回跳地址；否则跳转到首页
15        router.push(route.query.redirectUrl || '/')
16      } catch (e) {
17        // 错误消息提示
18        ElMessage.error(e.message)
19      }
20    })
21  }

```

十四、Layout布局及请求token拦截

1. 整体流程



2. 静态代码

```
1 <script setup>
2   import { useRoute } from 'vue-router'
3
4   const route = useRoute()
5 </script>
6
7 <template>
8   <el-container class="layout-box">
9     <el-aside width="200px">
10       <div class="logo">
11         
16       </div>
17       <el-menu
18         router
19         :default-active="route.path"
20         background-color="#313a46"
21         active-text-color="#fff"
22         text-color="#8391a2">
23         <el-menu-item index="/dashboard">
24           <el-icon><PieChart /></el-icon>
25           <span>数据看板</span>
26         </el-menu-item>
27         <el-menu-item index="/article">
28           <el-icon><Notebook /></el-icon>
29           <span>文章管理</span>
30         </el-menu-item>
31       </el-menu>
32     </el-aside>
33     <el-container>
34       <el-header>
35         <div class="user">
36           <el-avatar :size="36" />
37           <el-link :underline="false"> xxx </el-link>
38         </div>
39         <div class="logout">
40           <el-icon>
41             <SwitchButton />
42           </el-icon>
43         </div>
44       </el-header>
45       <el-main>
46         <router-view />
```

```
47     </el-main>
48   </el-container>
49 </el-container>
50 </template>
51
52 <style scoped lang="scss">
53   .layout-box {
54     .el-aside {
55       height: 100vh;
56       background: #313a46;
57       .el-menu {
58         border-right: none;
59         .el-menu-item {
60           display: flex;
61           justify-content: center;
62         }
63       }
64     }
65     .el-header {
66       display: flex;
67       justify-content: flex-end;
68       align-items: center;
69       box-shadow: 0 0 35px 0 rgba(151, 161, 171, 0.2);
70       background: #fff;
71       .user {
72         display: flex;
73         align-items: center;
74         height: 100%;
75         background: #fafbfc;
76         padding: 0 15px;
77         border: 1px solid #f1f3f5;
78         .el-avatar {
79           margin-right: 15px;
80         }
81       }
82       .logout {
83         padding: 0 15px;
84         font-size: 20px;
85         color: #999;
86       }
87     }
88   }
89 </style>
```

3. 封装接口

api/user.js

```
1 export function getUserInfoApi() {
2   return request({
3     method: 'get',
4     url: '/profile'
5   })
6 }
```

4. 请求拦截器携带token

utils/request.js

```
1 import useUserStore from '@store/user'
2 // 请求拦截器
3 instance.interceptors.request.use(
4   (config) => {
5     // config: 包含了所有请求信息的对象, 比如: method/url/headers...
6
7     // 对请求做公共处理
8     const userStore = useUserStore()
9     // 获取 token
10    const { token } = userStore
11    // token 有值
12    if (token) {
13      // 在请求头中统一携带 token
14      config.headers.Authorization = `Bearer ${token}`
15    }
16    // 给服务器返回请求对象
17    return config
18  },
19   (error) => Promise.reject(error)
20 )
```

5. Pinia 管理用户信息

stores/user.js

```
1 import { getUserInfoApi } from '@api/user'
2
3 export const useUserStore = defineStore('user', {
4
```

```

5  actions: {
6    setUserInfo(userInfo) {
7      this.userInfo = userInfo
8    },
9    // 登录 action
10   async loginAction(loginForm) {
11     // 发请求
12     const resp = await loginApi(loginForm)
13     // 存储数据
14     const { token } = resp
15     this.setToken(token)
16   },
17
18   // 获取用户信息
19   async getUserInfoAction() {
20     try {
21       // 发请求 (调用接口函数)
22       const userInfo = await getUserInfoApi()
23       // 设置 userInfo
24       this.setUserInfo(userInfo)
25     } catch (e) {}
26   }
27 }
28 })

```

6. 进入 Layout 调用 action

```

1  <script setup>
2  import useUserStore from '@store/user'
3  const userStore = useUserStore()
4
5  // 获取用户信息
6  userStore.getUserInfoAction()
7  </script>

```

7. Layout 头部渲染用户信息

```

1  <div class="user">
2    <el-avatar :size="36" :src="userStore.userInfo.avatar" />
3    <el-link :underline="false">{{ userStore.userInfo.name ||
4      userStore.userInfo.username }}</el-link>
5  </div>

```

十五、退出登录

1. Pinia 封装 action

```
1 actions: {  
2   // 退出  
3   logout() {  
4     this.token = ''  
5     removeToken()  
6  
7     this.userInfo = {}  
8   }  
9 }
```

2. 监听确认事件

```
1 <script setup>  
2   // 退出  
3   const onLogout = () => {  
4  
5   }  
6 </script>  
7  
8 <template>  
9   <el-popconfirm @confirm="onLogout">  
10     <template #reference>  
11       <el-icon><SwitchButton /></el-icon>  
12     </template>  
13   </el-popconfirm>  
14 </template>
```

3. 调用 action 并回到登录

```
1 <script setup>  
2   import { useRouter } from 'vue-router'  
3   const router = useRouter()  
4   // 退出登录  
5   const onLogout = () => {  
6     // 调用 userStore 的 logout 函数  
7     userStore.logout()
```

```
8 // 跳转至登录页
9 router.push('/login')
10 }
11 </script>
```

十六、处理 token 过期

1. 整理流程



2. 代码示例

```
1 import router from '@router'
2
3 const RESPONSE_CODE = {
4   unauthorized: 401 // 未认证
5 }
6 // 响应拦截器
7 instance.interceptors.response.use(
8   (response) => {
9
10  },
11   async (error) => {
12     const { status, data: { message } } = error.response
13     // 统一错误提示
14     ElMessage.error(message)
15     // token 过期
16     if (status === RESPONSE_CODE.unauthorized) {
17       const userStore = useUserStore()
18       userStore.logout()
19       router.push(`/login?redirectUrl=${router.currentRoute.value.path}`)
20     }
21     // 给组件返回错误信息
22     return Promise.reject(new Error(message))
23   }
24 )
```

十七、明确文章管理模块需求

1. 整体效果

共 100 条记录

+ 新增文章

标题	作者	类别	点赞	浏览数	创建时间	操作
构压目何农太较情精感压打业论相区	袁承志	大数据	6029	748	2024-04-17 15:09	
派比外这然种候四科同京特见备	风清扬	云计算	6293	826	2024-02-07 12:40	
件律数闻各来千门话列代专或出派下种	风清扬	算法	4366	948	2024-05-18 11:40	
技龙现地酸道律响它听元火	admin	测试	7111	961	2024-04-17 15:09	
或原群多月天始这车得交并消走更法定	山村码农	测试	9002	716	2024-03-11 14:50	
参商里说带复持采只新立万断系位京	仗剑走天涯	后端	3688	779	2024-04-06 18:10	
斗记劳根情周及非前白以	仗剑走天涯	数据库	6695	135	2024-02-26 09:17	
达住知好放往	白衣少年	人工智能	2522	184	2024-03-23 16:31	
性题用方看严和物	井川里予	嵌入式	9646	667	2024-05-21 10:29	
林如件亲样属使根声制派正即积作式	admin	人工智能	7209	304	2024-05-07 13:50	

<

1

2

3

4

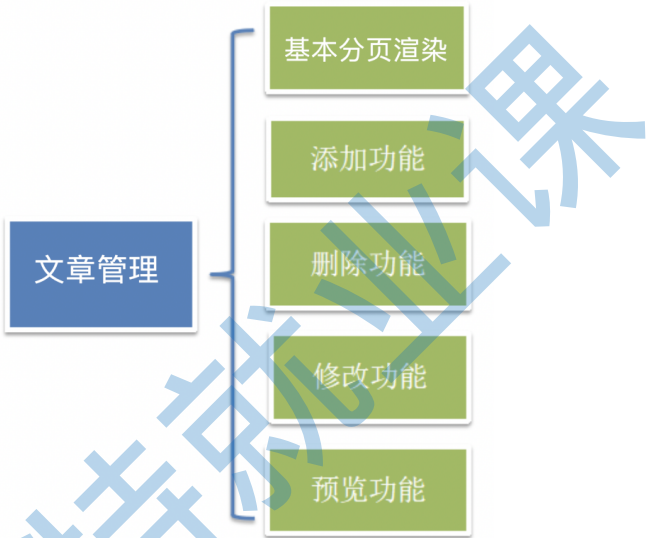
5

6

...

10

>



十八、准备文章管理结构

1. 静态结构

```
1 <script setup>
2 </script>
3 <template>
4   <div class="article-box">
5     <el-breadcrumb separator-icon="arrow-right">
6       <el-breadcrumb-item :to="{ path: '/' }">首页</el-breadcrumb-item>
7       <el-breadcrumb-item>文章管理</el-breadcrumb-item>
8     </el-breadcrumb>
9     <el-card>
10      <!-- 头部 -->
11      <template #header>
12        <div class="card-header">
13          <span>共 xxx 条记录</span>
14          <el-button type="primary" round>
```

```

15         <el-icon><Plus /></el-icon>
16         <span>新增文章</span>
17     </el-button>
18 </div>
19 </template>
20 </el-card>
21 </div>
22 </template>
23
24 <style lang="scss" scoped>
25     .article-box {
26         .el-card {
27             margin-top: 20px;
28             .card-header {
29                 display: flex;
30                 justify-content: space-between;
31                 align-items: center;
32                 height: 40px;
33             }
34             :deep(.el-card__footer) {
35                 margin-top: -15px;
36                 border-top: none;
37             }
38             .el-table {
39                 .el-icon:nth-child(2) {
40                     margin: 0 15px;
41                 }
42             }
43             .el-pagination {
44                 display: flex;
45                 justify-content: center;
46             }
47         }
48     }
49 </style>

```

十九、渲染功能-定义接口并调用

1. 定义接口

api/article.js

```

1 export const getArticleListApi = (query) => {
2     return request({
3         method: 'get',

```

```
4     url: '/article',
5     params: query
6   })
7 }
```

2. 调用接口

views/Article/index.vue

```
1 import { reactive, ref } from 'vue'
2 // 文章列表
3 const articleList = ref([])
4 // 总数量
5 const totalCount = ref(0)
6
7 // 请求查询参数
8 const query = reactive({
9   current: 1, // 当前页
10  pageSize: 10 // 每页条数
11 })
12
13 getArticleList()
14
15 // 获取并保存文章列表
16 async function getArticleList() {
17   // 发请求
18   const { rows, total } = await getArticleListApi(query)
19   // 保存列表数据
20   articleList.value = rows
21   // 保存数据总条数
22   totalCount.value = total
23 }
```

二十、渲染功能-动态渲染表格

1. 数据渲染

views/Article/index.vue

```
1 <el-table :data="articleList" border striped>
2   <el-table-column prop="stem" label="标题" width="280" />
3   <el-table-column prop="creator" label="作者" width="180" />
4   <el-table-column prop="category" label="类别" width="100" />
```

```

5 <el-table-column prop="likeCount" label="点赞" width="120" sortable />
6 <el-table-column prop="views" label="浏览数" width="120" sortable />
7 <el-table-column prop="createdAt" label="创建时间" />
8 <el-table-column label="操作"></el-table-column>
9 </el-table>

```

其中 `sortable` 表示该列可排序

2. 插槽自定义操作列

```

1 <el-table-column label="操作">
2   <template>
3     <el-tooltip placement="bottom" content="预览">
4       <el-icon :size="18"><View /></el-icon>
5     </el-tooltip>
6     <el-tooltip placement="bottom" content="编辑">
7       <el-icon :size="18"><Edit /></el-icon>
8     </el-tooltip>
9     <el-tooltip placement="bottom" content="删除">
10      <el-icon :size="18"><Delete /></el-icon>
11    </el-tooltip>
12  </template>
13 </el-table-column>

```

3. main.js导入并注册图标组件

```

1 import {
2   View,
3   Edit,
4   Delete
5 } from '@element-plus/icons-vue'
6
7 app.component(View.name, View)
8 app.component(Edit.name, Edit)
9 app.component>Delete.name, Delete)

```

二十一、渲染功能-分页渲染

1. 效果

2. 认识分页组件

Pagination分页组件 | Element Plus

当数据量过多时，使用分页分解数据。

3. 代码示例

```
1 <script setup>
2   // 当前页码改变时
3   const onCurrentChange = (val) => {
4     // 更新页码
5     query.current = val
6     // 请求列表
7     getArticleList()
8   }
9 </script>
10 <template #footer>
11   <!-- 分页组件 -->
12   <el-pagination
13     @current-change="onCurrentChange"
14     v-model:current-page="query.current"
15     :total="totalCount"
16     background
17     layout="prev, pager, next"
18   />
19 </template>
```

二十二、删除文章

1. 定义接口

api/article.js

```
1 /**
2  * 删除文章
3  */
4 export const delArticleApi = (articleId) => {
5   return request({
6     method: 'delete',
7     url: `/article/${articleId}`
```

```
8   })
9 }
```

2. 点击删除

2.1 添加点击事件

```
1 <el-icon :size="18" @click="onDel(row.id)"><Delete /></el-icon>
```

2.2 删除确认

```
1 import { ElMessage, ElMessageBox } from 'element-plus'
2
3 // 删除
4 const onDel = async (articleId) => {
5   try {
6     // 确认了
7     await ElMessageBox.confirm('此操作不可逆、确认删除么?', 'Warning')
8     // ...
9   } catch (e) {
10    // 取消了
11    console.log(e)
12   }
13 }
```

2.3 完成删除

```
1 import { delArticleApi } from '@/api/article'
2
3 // 删除
4 const onDel = async (articleId) => {
5   try {
6     await ElMessageBox.confirm('此操作不可逆、确认删除么?', '友情提示')
7     // 删除请求
8     await delArticleApi(articleId)
9     // 刷新列表
10    getArticleList()
11   } catch (e) {
12     console.log(e)
13   }
14 }
```

二十三、抽屉打开和关闭

1. 认识抽屉组件

Drawer抽屉组件 | Element Plus

```
1  <!--
2    title="我是标题"
3    visible="布尔值" 控制显示隐藏
4    direction="direction" 控制方向
5    size="60%" 窗体所占的区域多宽
6  -->
7
8  <script>
9    // 抽屉是否可见
10   const visible = ref(false)
11 </script>
12
13
14 <el-drawer
15   v-model="visible"
16   title="大标题"
17   direction="rtl"
18   size="45%"
19 >
20   <span>我来啦!</span>
21 </el-drawer>
```

2. 控制抽屉打开和关闭

添加、编辑、预览，都要打开抽屉, 可以复用

```
1  <script setup>
2    // 文章操作类型
3    const ARTICLE_OPS_TYPE = {
4      add: 'add', // 新增
5      edit: 'edit', // 编辑
6      preview: 'preview' // 预览
7    }
8    // 当前操作类型
9    const currentOpsType = ref('')
10
```

```

11 // 打开抽屉
12 const onOpen = () => {
13   visible.value = true
14 }
15 </script>
16
17 <template>
18   <el-button type="primary" round @click="onOpen">
19     <el-icon><Plus /></el-icon>
20     <span>新增文章</span>
21   </el-button>
22
23
24   <template #default="{ row }">
25     <el-tooltip placement="bottom" content="预览">
26       <el-icon :size="18" @click="onOpen">
27         <View />
28       </el-icon>
29     </el-tooltip>
30     <el-tooltip placement="bottom" content="编辑">
31       <el-icon :size="18" @click="onOpen">
32         <Edit />
33       </el-icon>
34     </el-tooltip>
35     <el-tooltip placement="bottom" content="删除">
36       <el-icon :size="18"><Delete /></el-icon>
37     </el-tooltip>
38   </template>
39 </template>

```

二十四、记录操作类型并控制标题

1. 记录操作类型

```

1 <script setup>
2   // 记录当前的操作类型
3   const currentOpsType = ref('')
4   // 记录当前操作的文章 id
5   const articleId = ref('')
6
7   // 打开抽屉
8   const onOpen = (opsType, id) => {
9     visible.value = true
10    // 保存当前操作类型
11    currentOpsType.value = opsType

```

```

12    // 保存当前的文章id
13    articleId.value = id
14  }
15 </script>
16 <template>
17   <el-button
18     type="primary"
19     round
20     @click="onOpen(ARTICLE_OPS_TYPE.add, '')">
21     <el-icon><Plus /></el-icon>
22     <span>新增文章</span>
23   </el-button>
24
25   <el-icon
26     size="18"
27     @click="onOpen(ARTICLE_OPS_TYPE.preview, row.id)"
28   ><View/></el-icon>
29
30   <el-icon
31     size="18"
32     @click="onOpen(ARTICLE_OPS_TYPE.edit, row.id)"
33   ><Edit/></el-icon>
34 </template>

```

2. 声明映射对象

```

1 // 标题的映射对象
2 const TITLE_MAP = {
3   add: '新增',
4   edit: '编辑',
5   preview: '预览'
6 }

```

3. 计算属性得到动态标题

```

1 // 当前展示的标题
2 const activeTitle = computed(() => {
3   return TITLE_MAP[currentOpsType.value] + '面板'
4 })

```

4. 抽屉组件绑定计算属性

```
1 <el-drawer :title="activeTitle">
2   ...
3 </el-drawer>
```

二十五、抽屉组件主体内容

1. 目标

新增、编辑、预览 共用一个组件

2. 新建组件

Article/components/ArticleOps.vue

```
1 <el-form label-width="55px">
2   <el-form-item label="标题">
3     <el-input placeholder="请输入文章标题" />
4   </el-form-item>
5   <el-form-item label="内容">
6     富文本编辑器
7   </el-form-item>
8   <el-form-item>
9     <el-button type="primary">确 认</el-button>
10    <el-button>取 消</el-button>
11  </el-form-item>
12 </el-form>
13
14 <style lang="scss" scoped>
15   .el-form {
16     .el-input {
17       height: 36px;
18     }
19   }
20 </style>
```

3. 使用组件

Article/index.vue

```
1 <script setup>
2   import ArticleOps from './components/ArticleOps.vue'
3 </script>
```

```
4
5 <template>
6   <el-drawer>
7     <ArticleOps />
8   </el-drawer>
9 </template>
```

二十六、集成富文本编辑器

Vuejs优秀插件集合 | [GitHub - vuejs/awesome-vue](#)

1. 安装 VueQuill

[VueQuill官方文档](#)

```
1 npm install @vueup/vue-quill@latest
2
3 # or
4
5 yarn add @vueup/vue-quill@latest
```

2. 导入并注册

```
1 // main.js
2
3 // 导入组件
4 import { QuillEditor } from '@vueup/vue-quill'
5 // 导入样式
6 import '@vueup/vue-quill/dist/vue-quill.snow.css'
7
8 // 全局注册组件
9 app.component('QuillEditor', QuillEditor)
```

3. 使用组件

在 `ArticleOps.vue` 中

```
1 <QuillEditor
2   theme="snow"
3   toolbar="full"
4   placeholder="请输入文章内容"
```

4. 样式美化

```
1 .el-form {
2   .el-input {
3     height: 36px;
4   }
5   :deep(.ql-container) {
6     width: 100%;
7     min-height: 180px;
8   }
9   .el-form-item:last-child {
10    margin-top: 100px;
11  }
12 }
```

二十七、文章操作组件区分操作类型

1. 分析

因为 新增、编辑、预览 共用一个组件, 后续会调用不同的接口, 为了清楚知道当前是何种操作, 进行父子通信

2. 父子通信

2.1 子接受

ArticleOps.vue

```
1 // 接受 props
2 const props = defineProps({
3   // 当前操作类型
4   opsType: {
5     type: String,
6     default: ''
7   },
8   // 当前文章 id
9   articleId: {
10    type: String,
11    default: ''
12  }
13 })
```


2.2 父传递

Article/index.vue

```
1 <article-ops
2   :opsType="currentOpsType"
3   :articleId="articleId"
4 />
```

二十八、预览文章

1. 分析

预览与新增、编辑不同,展示另一套界面,所以我们要新增一套预览的 HTML 结构

在 ArticleOps.vue 下

```
1 <template>
2   <!-- 预览 -->
3   <div class="preview">
4     <h4> xxx </h4>
5     <div> xxxxxxxxxxxx... </div>
6   </div>
7
8   <!-- 新增/编辑 -->
9   <el-form label-width="55px">
10
11   </el-form>
12 </template>
```

2. 条件控制渲染

```
1 <template>
2   <!-- 预览 -->
3   <div class="preview" v-if="props.opsType === 'preview'">
4
5   </div>
6
7   <!-- 新增/编辑 -->
8   <el-form label-width="55px" v-else>
9
```

```
10     </el-form>
11 </template>
```

3. 封装常量模块

便于 `ARTICLE_OPS_TYPE` 复用和可维护性, 建议将其抽离封装到常量模块

新建 `src/constant/article.js` 模块

```
1 // 文章操作类型
2 export const ARTICLE_OPS_TYPE = {
3   add: 'add', // 新增
4   edit: 'edit', // 编辑
5   preview: 'preview' // 预览
6 }
```

4. 重构代码

在 `Article/index.vue` 和 `ArticleOps.vue` 下

```
1 // 1、导入文章操作常量
2 import { ARTICLE_OPS_TYPE } from '@/constant/article'
3
4 // 2、删除之前定义好的 ARTICLE_OPS_TYPE
```

`ArticleOps.vue` 修改判断条件

```
1 <!-- 预览结构 -->
2 <div
3   class="preview"
4   v-if="props.opsType === ARTICLE_OPS_TYPE.preview">
5   ...
6 </div>
```

5. 定义获取文章详情接口

`api/article.js`

```
1 /**
2  * 获取文章详情
```

```

3  */
4  export const getArticleDetailApi = (articleId) => {
5    return request({
6      method: 'get',
7      url: `/article/${articleId}`
8    })
9  }

```

6. 请求并渲染文章详情

```

1  <script setup>
2    // 导入详情接口
3    import { getArticleDetailApi } from '@api/article'
4
5    // 当前文章表单对象
6    const articleForm = reactive({
7      id: '', // 文章 id
8      stem: '', // 文章标题
9      content: '' // 文章内容
10   })
11
12   getArticleDetail()
13
14   // 获取文章详情
15   async function getArticleDetail() {
16     // 判断是否存在文章 id
17     if (!props.articleId) return
18     // 请求文章详情
19     const detail = await getArticleDetailApi(props.articleId)
20     // 给表单字段赋值
21     Object.keys(articleForm).forEach((key) => {
22       articleForm[key] = detail[key]
23     })
24   }
25 </script>
26 <template>
27   <!-- 预览结构 -->
28   <div
29     class="preview"
30     v-if="props.opsType === ARTICLE_OPS_TYPE.preview">
31     <h4>{{ articleForm.stem }}</h4>
32     <div v-html="articleForm.content"></div>
33   </div>
34 </template>

```

7. 解决数据重复

在 `Article/index.vue` 下

`el-drawer` 默认不会每次创建一个新的组件, 而是会复用之前的组件, 这里就会导致数据重复

解决办法: 给组件添加一个 不重复的 `key`, 这样因为每次key值不同, `el-drawer` 就会 重新创建组件, 不去复用之前的, 从而保证了每次数据的不重复

```
1 <ArticleOps
2   :key="articleId"
3 />
```

二十九、新增文章

1. 定义接口

```
1 /**
2  * 新增文章
3  */
4 export const postArticleApi = (articleForm) => {
5   return request({
6     method: 'post',
7     url: '/article',
8     data: articleForm
9   })
10 }
```

2. 表单控件双向数据绑定

在 `ArticleOps.vue` 下

```
1 <template>
2   <el-form label-width="55px">
3     <el-form-item label="标题">
4       <el-input v-model="articleForm.stem" />
5     </el-form-item>
6     <el-form-item label="内容">
7       <QuillEditor
8         contentType="html"
9         v-model:content="articleForm.content"
10      />
11     </el-form-item>
```

```
12   </el-form>
13 </template>
```

3. 发起新增请求

在 `ArticleOps.vue` 下

```
1 <script setup>
2   import { postArticleApi } from '@api/article'
3
4   // 确认
5   const onConfirm = async () => {
6     if (props.articleId) {
7       // 编辑
8
9     } else {
10      // 新增
11
12      // 解构
13      const { stem, content } = articleForm
14      // 调接口
15      await postArticleApi({
16        stem,
17        content
18      })
19    }
20    // 成功提示
21    ElMessage.success(props.articleId ? '编辑成功' : '新增成功')
22    // 通过父组件 (1.关闭抽屉 2.获取文章列表)
23    emit('finish')
24  }
25 </script>
26
27 <template>
28   <el-button
29     type="primary"
30     @click="onConfirm"
31     >确 认</el-button>
32 </template>
```

4. 父组件关闭抽屉+获取列表

```
1 <script setup>
```

```

2    // 关闭抽屉
3    const onClose = () => {
4      visible.value = false
5    }
6
7    const onFinish = () => {
8      // 关闭抽屉
9      onClose()
10     // 重新获取文章列表
11     getArticleList()
12   }
13 </script>
14 <template>
15   <el-drawer>
16     <!-- 文章操作组件 -->
17     <ArticleOps @finish="onFinish" />
18   </el-drawer>
19 </template>

```

三十、编辑文章

1. 定义接口

```

1 /**
2  * 编辑文章
3  */
4 export function putArticleApi(articleForm) {
5   return request({
6     method: 'put',
7     url: `/article/${articleForm.id}`,
8     data: articleForm
9   })
10 }

```

2. 绑定点击事件并处理

在 `ArticleOps.vue` 下

```

1 <script setup>
2   import { putArticleApi } from '@api/article'
3
4   import { ElMessage } from 'element-plus'
5

```

```

6 // emit events
7 const emit = defineEmits()
8
9 // 确认
10 const onConfirm = async () => {
11   if (props.articleId) {
12     // 编辑
13     await putArticleApi(articleForm)
14
15   } else {
16     // 新增
17
18     // ...
19   }
20 }
21 </script>

```

三十一、取消和清空表单

1. 点击取消关闭抽屉

```

1 <script setup>
2 // 取消
3 const onCancel = () => {
4   // 通知父组件：关闭抽屉组件
5   emit('close')
6 }
7 </script>
8
9 <el-button @click="onCancel">取 消</el-button>

```

2. 父组件关闭抽屉

```

1 <ArticleOps @close="onClose" />

```

3. 新增完毕清空表单

```

1 // 确认
2 const onConfirm = async () => {
3   if (props.articleId) {

```

```
4      // 编辑
5
6    } else {
7      // 新增
8
9      // 解构
10     const { stem, content } = articleForm
11     // 调接口
12     await postArticleApi({
13       stem,
14       content
15     })
16
17     // 清空表单
18     Object.keys(articleForm).forEach((key) => {
19       articleForm[key] = ''
20     })
21
22   }
23   // ...
24 }
```