# Vue 核心技术与实战

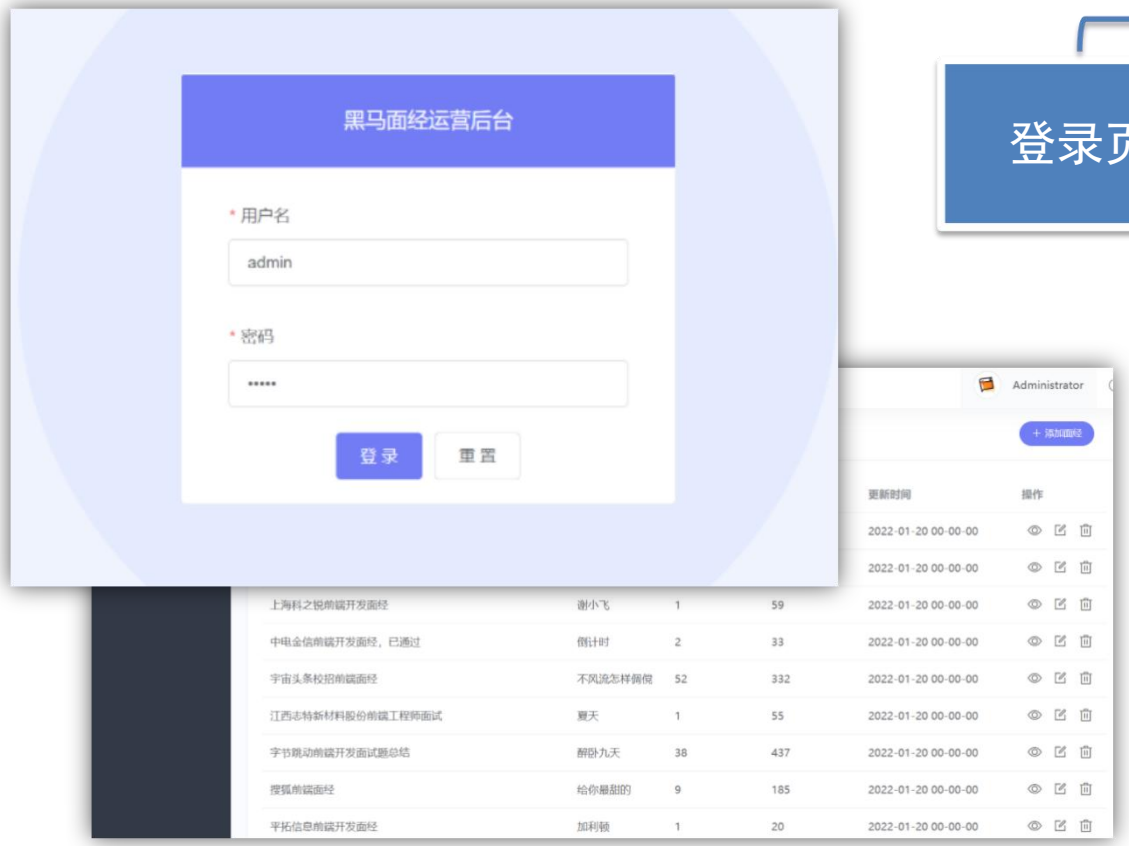
## 面经 PC 项目

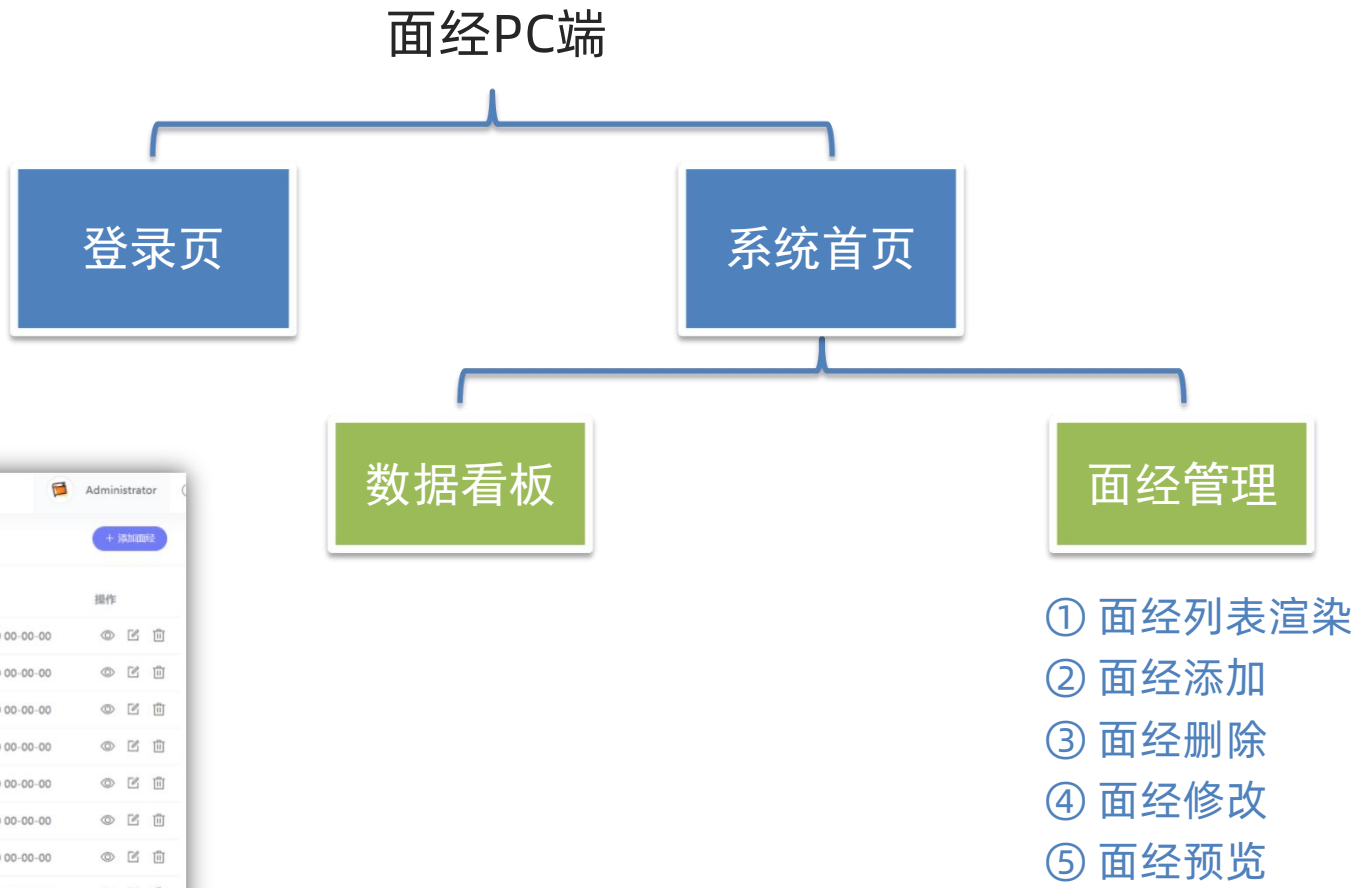
黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌

# 项目演示

**目标：查看项目效果，明确功能模块**



# 项目收获

**目标：明确做完本项目，能够收获哪些内容**

| | | |
|---|---|---|
| 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 | vuex 分模块存储 | echarts 数据可视化 |
| elementUI 表格 & 分页渲染 | 富文本编辑器编辑 & 预览 | 面经文章管理 (增删改查) |

# 创建项目

**目标：基于 VueCli 自定义创建项目架子 & 认识 sass 基础语法**


vue create hm-element-pc

- Babel / Router / Vuex / CSS / Linter
- Vue2.x
- VueRouter hash模式
- CSS预处理 Sass
- ESlint: Standard & Lint on Save
- 配置文件 dedicated config files

# 创建项目

**目标：基于 VueCli 自定义创建项目架子 & 认识 sass 基础语法**

sass的语法有两个 sass (旧) 和 scss (新)

1. scss 语法： 和 less 语法类似，支持嵌套，支持变量...

   scss: $变量名

   less: @变量名

2. sass 语法：和 stylus 语法很像，要求省略；和 {}

```
.box
  .son1
    width: 100px
    height: 100px
    margin: 20px
    background-color: pink
  .son2
    width: 100px
    height: 100px
    margin: 20px
    background-color: orange
```

```
$color: pink;

.box {
  .son1 {
    width: 100px;
    height: 100px;
    margin: 20px;
    background-color: $color;
  }
```

# 创建项目

**目标：构建面经PC的目录**

默认生成的目录结构不满足我们的开发需求，所以这里需要做一些自定义改动。

主要是4个工作：

① 删除初始化的默认文件

② 修改剩余代码内容 (router/index.js & App.vue)

③ 新增调整我们需要的目录结构

④ 将项目需要的图片资源放置 *assets 文件夹* 中

# element ui 组件库 - 全部 & 按需导入

**目标：element ui 组件库 - 全部导入**

官方文档: https://element.eleme.io/#/zh-CN

全部引入: 会导入所有的组件，方便但代码包体积会变大

① 安装 element-ui

```
yarn add element-ui
```

② main.js 中注册

```
import ElementUI from 'element-ui';
import 'element-ui/lib/theme-chalk/index.css';
Vue.use(ElementUI);
```

③ 使用测试

```
<el-button type="primary">主要按钮</el-button>
```


# element ui 组件库 - 全部 & 按需导入

**目标：element ui 组件库 - 按需引入**

① 安装 element-ui (已安装)

```
yarn add element-ui
```

② **安装插件**

```
yarn add babel-plugin-component -D
```

③ babel.config.js 中配置

```
module.exports = {
  presets: [
    '@vue/cli-plugin-babel/preset'
  ],
  plugins: [
    [
      'component',
      {
        libraryName: 'element-ui',
        styleLibraryName: 'theme-chalk'
      }
    ]
  ]
}
```

④ main.js 按需导入注册

```
import { Button } from 'element-ui'
Vue.use(Button)
```

⑤ 测试使用

```
<el-button type="primary">主要按钮</el-button>
```

⑥ 提取到 element-ui.js中，main.js 导入

```
// 导入按需导入的配置文件
import '@/utils/element-ui.js '
```

# element-ui 主题色配置

**目标：阅读文档，配置element-ui主题色**

官方配置

① 新建 styles/index.scss

```scss
// 修改主题色
$--color-primary: rgba(114,124,245,1);
$--font-path: '~element-ui/lib/theme-chalk/fonts';
@import "~element-ui/packages/theme-chalk/src/index";

body {
  margin: 0;
  padding: 0;
  background: #fafbfe;
}
```

② main.js 导入

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



# element-ui 主题色配置

**目标：request模块 & storage模块 封装**

① request 模块封装：

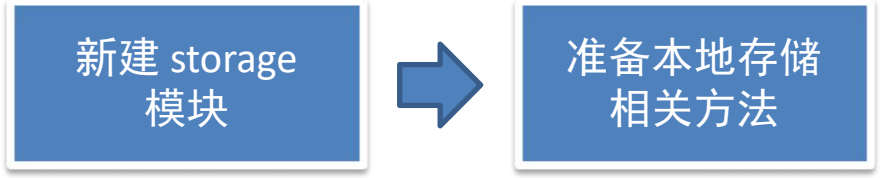
我们会使用 axios 来请求后端接口, 一般都会对 axios 进行一些配置 (比如: 配置基础地址等)

一般项目开发中, 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


② storage 本地存储模块封装：

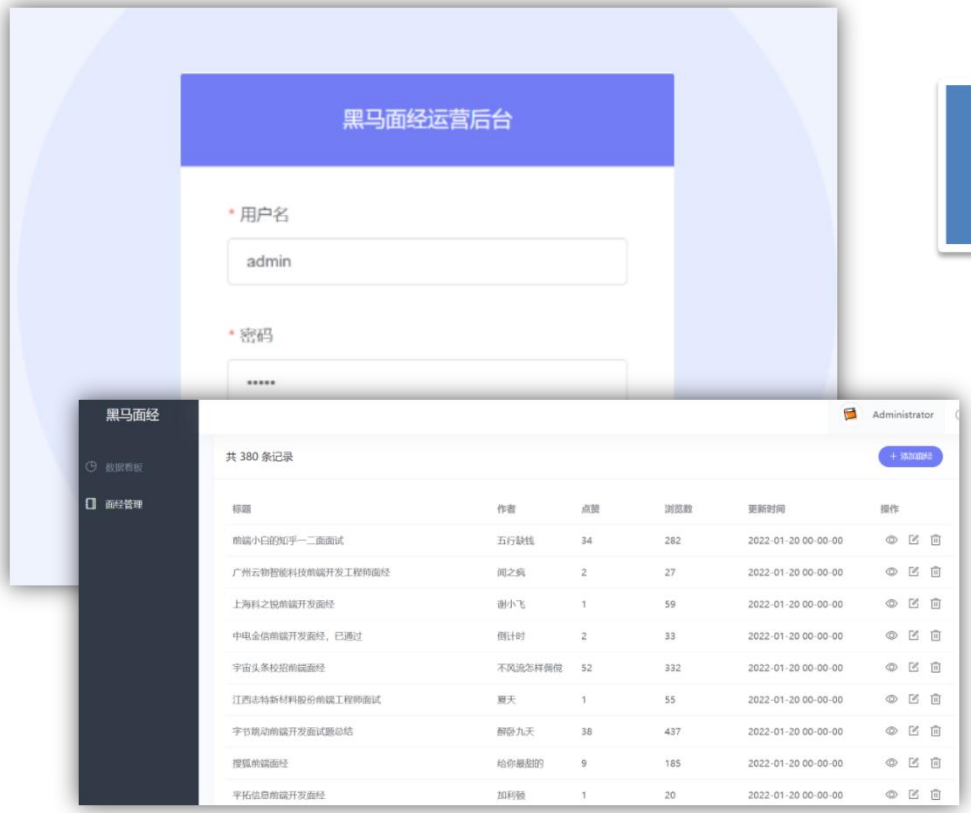
本地存储字段名，为了防止重名，通常会起的很长，直接不封装使用，就很容易出错

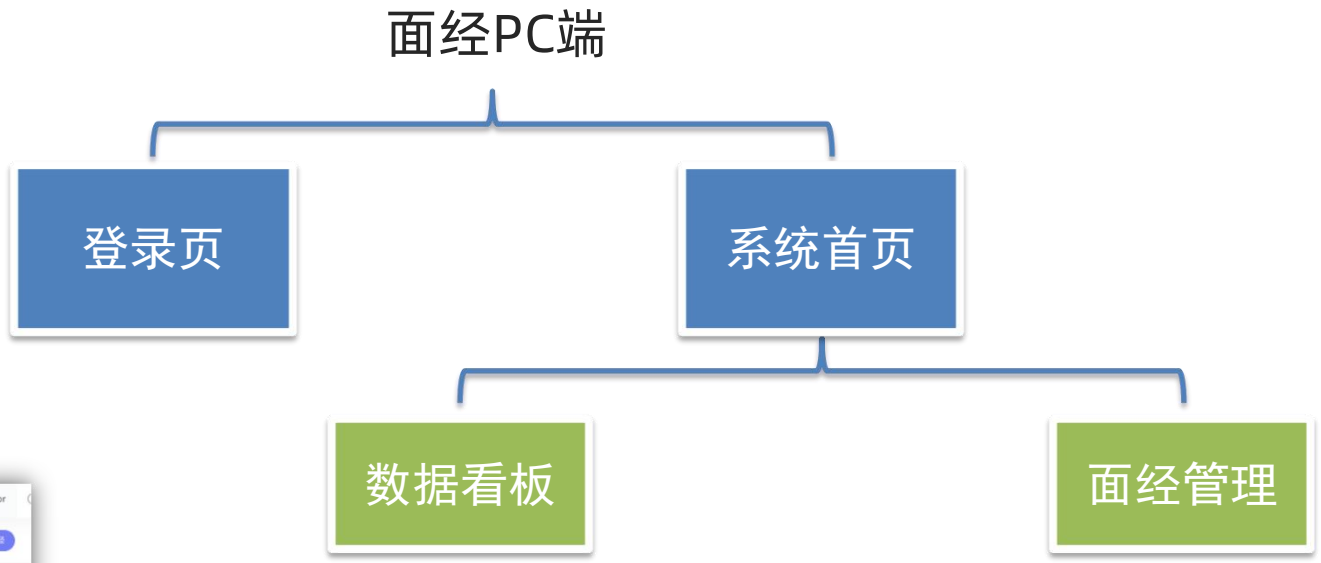
一般项目中，都会对本地存储的操作进行封装，提高可维护性



# 路由设计配置

**目标：分析项目页面，设计路由，配置路由**

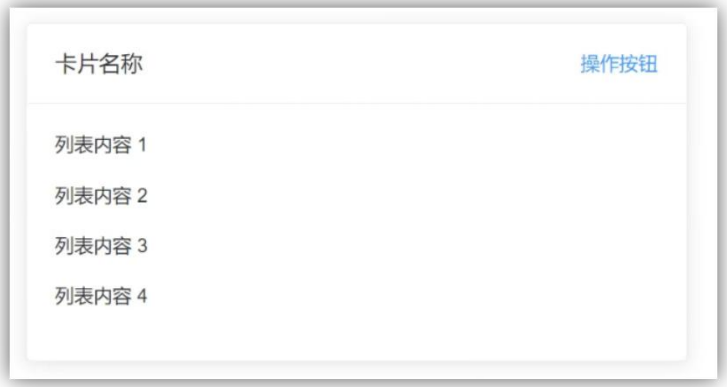



# 登录模块

**目标：搭建登录基本架子 & 样式定制**


- el-card 卡片组件
- el-form表单组件
- el-form-item列表项组件
- el-input 输入框组件
- el-button 按钮组件



# 登录模块

**目标：搭建登录基本架子 & 样式定制**

1. 控制组件的样式

① 给组件加上类名，添加的类名，会直接渲染到组件根元素上

<el-button class="myBtn">

② 直接通过组件标签名，作为类名控制样式

规范：组件的根元素，有一个和组件名同名的类名

<el-button> => .el-button


2. 深度作用选择器

默认scoped只会影响到当前模板范围内的组件

如果希望样式向下深度渗透，→ 深度作用选择器

① scss -> ::v-deep

② less -> /deep/

# 登录模块

**目标：构建基本的表单结构 & 样式美化**


- el-card 卡片组件
- el-form表单组件
- el-form-item列表项组件
- el-input 输入框组件
- el-button 按钮组件

```
<el-form>
  <el-form-item label="用户名: ">
    <el-input placeholder="请输入用户名"></el-input>
  </el-form-item>
  <el-form-item label="密码: ">
    <el-input type="password" placeholder="请输入用户密码">
  </el-form-item>
  <el-form-item class="tc">
    <el-button type="primary">登录</el-button>
    <el-button>重置</el-button>
  </el-form-item>
</el-form>
```

# 登录模块

**目标：掌握 element-ui 基本校验**

说明：在向后端发请求，调用接口之前，我们需要对所要传递的参数进行验证，把用户的错误扼杀在摇篮之中。

element-ui的校验

① el-form：`model` 属性，`rules` 规则

② el-form-item：绑定 `prop` 属性

③ el-input： 绑定` v-model`

## 登录模块

**目标：掌握 element-ui 正则校验**

下面是常用内置的基本验证规则：其余校验规则参见 [async-validator]

| 规则 | 说明 |
| --- | --- |
| required | 必须的，例如校验内容是否非空 |
| pattern | 正则表达式，例如校验手机号码格式、校验邮箱格式 |

```
password: [
  { required: true, message: '请输入密码', trigger: ['blur', 'change'] },
  { pattern: /^\w{5,11}$/, message: '请输入 5 到 11 位的密码', trigger: ['blur', 'change'] }
]
```

上述已经可以完成大部分需求，如果需要更复杂业务校验需求，可以自定义校验~（后面的一个项目课程会进一步讲解）

# 登录模块

**目标：提交表单 和 重置**

说明：每次点击按钮, 进行ajax登录前, 应该先对整个表单内容校验，要通过校验了, 才发送请求!

1. 通过 ref 和 $refs 获取 form 组件

```
<el-form ref="form" :model="form" :rules="rules" autocomplete="off">
...
<el-button @click="login" type="primary">登 录</el-button>
  </div>
</template>
```


2. 调用 form 组件方法，validate / resetFields

```
methods: {
  async login () {
    try {
      const valid = await this.$refs.form.validate()
      console.log(valid)
    } catch (e) {
      console.log(e)
    }
  }
}
```

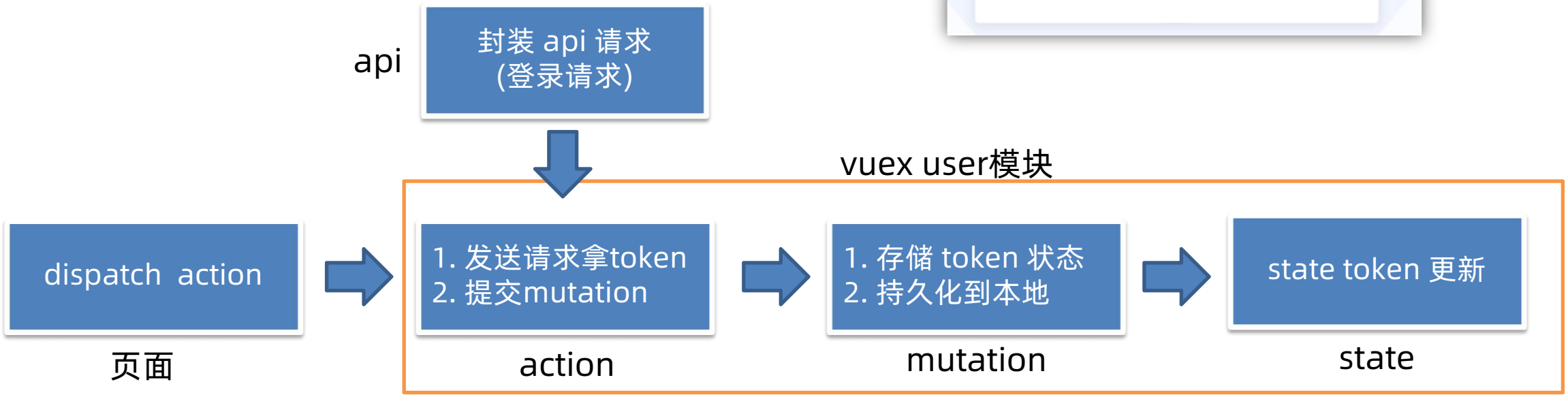
```
methods: {
  reset () {
    this.$refs.form.resetFields()
  }
}
```

# 登录模块

**目标：封装 API 登录请求模块，vuex 构建 user 模块存 token**

补充说明：

1. token 存入 vuex 的好处，易获取，响应式
2. vuex 需要分模块 => user 模块
3. vuex 刷新会重新初始化，缓存的数据会丢失 => 本地存储也要存

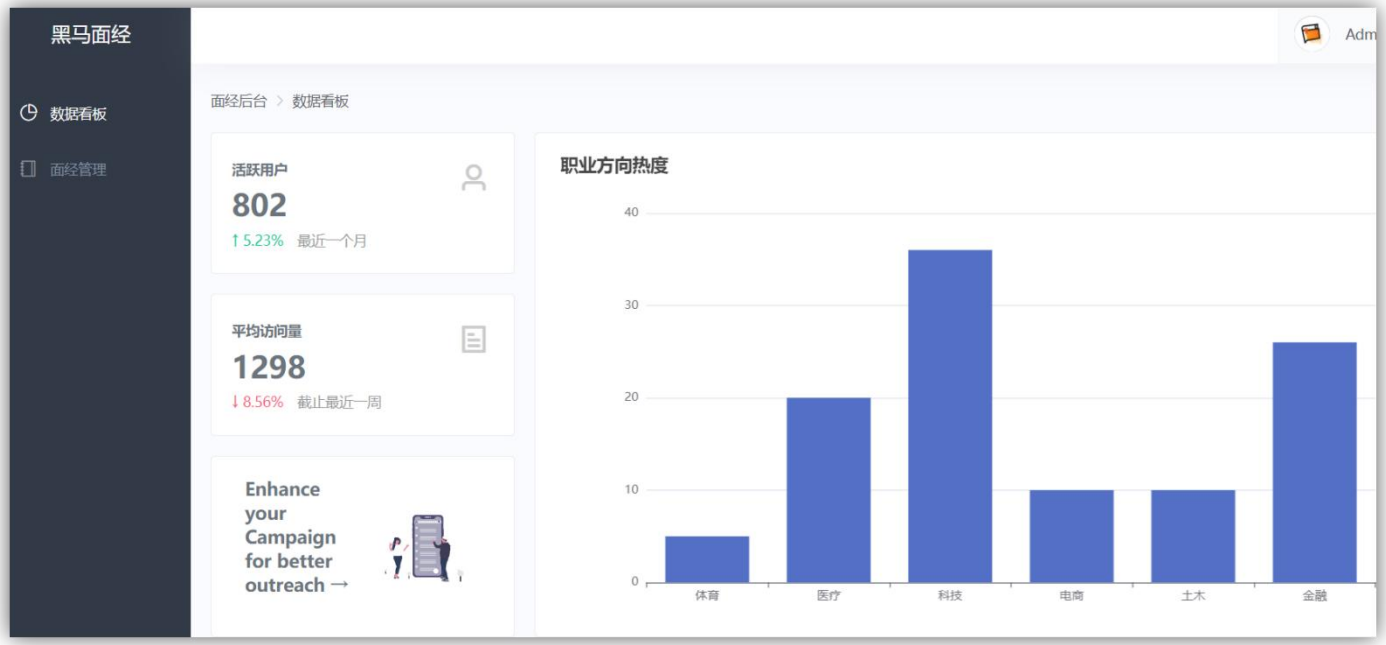



# 登录模块

**目标：登录提示 & 登录访问拦截处理**

说明：

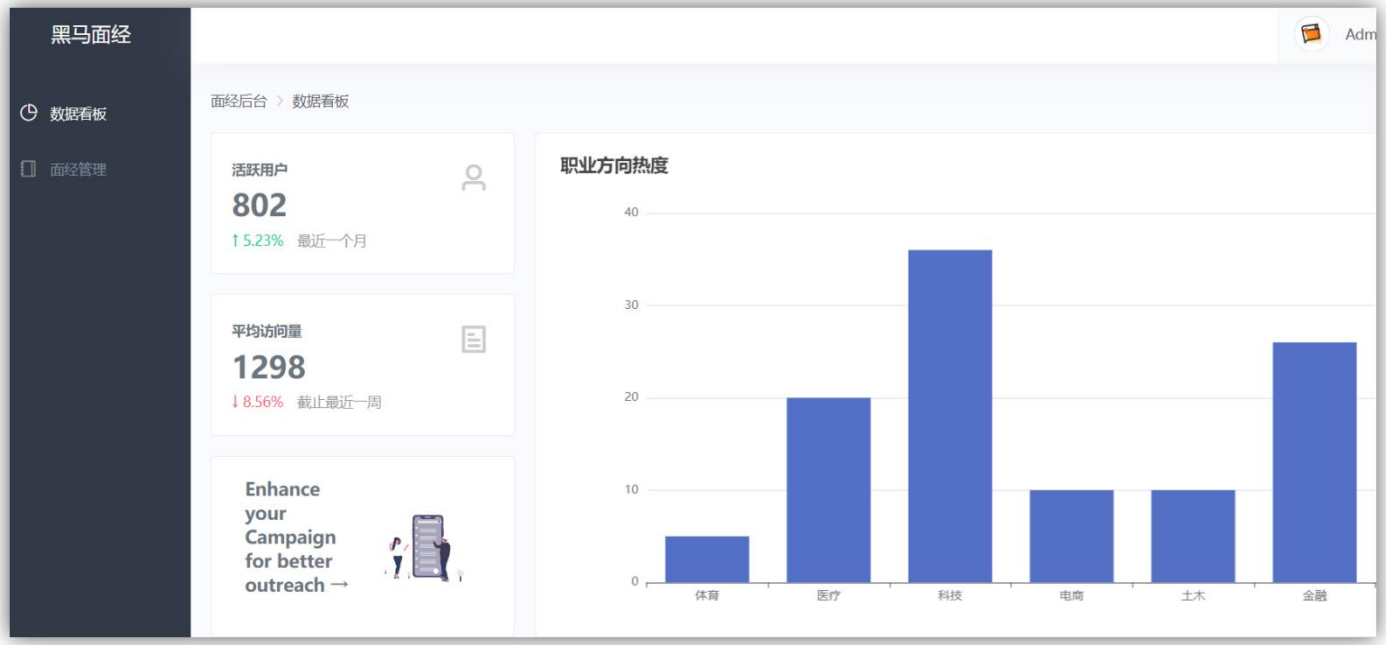
1. 登录成功，需要给个提示。错误提示可以通过 响应拦截器 统一处理。
2. 未登录的用户，不可以访问首页，需要 登录访问拦截（路由导航守卫）



# 首页架子 layout 模块

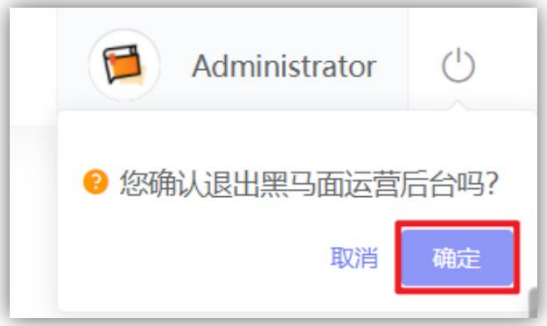
**目标：layout 布局 & 请求拦截器携带 token**

layout 基本布局 (快速) → 一进入页面请求个人信息 → 遇到 401 错误? → 请求拦截器统一携带 token



# 首页架子 layout 模块

**目标：分析结构 & 退出功能**




## el-popconfirm 气泡框组件

| 事件名称 | 说明 |
| --- | --- |
| confirm | 点击确认按钮时触发 |
| cancel | 点击取消按钮时触发 |

1. 提供处理函数，提交mutation

```
handleConfirm () {
  this.$store.commit('user/logout')
  this.$router.push('/login')
}
```

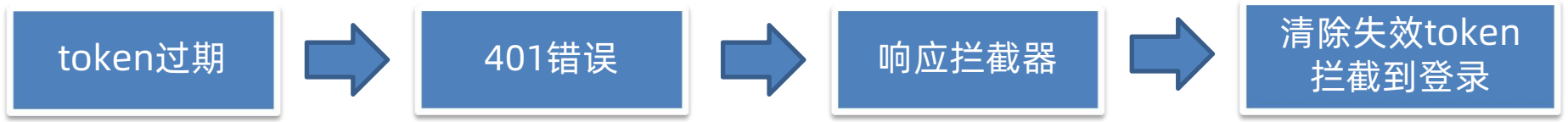
2. 提供mutaiton处理函数

```
mutations: {
  logout (state) {
    state.token = ''
    // 清除vuex的同时，需要将本地的也清空
    delToken()
  }
},
```

# 首页架子 layout 模块

**目标：响应拦截器，统一处理 token 过期**

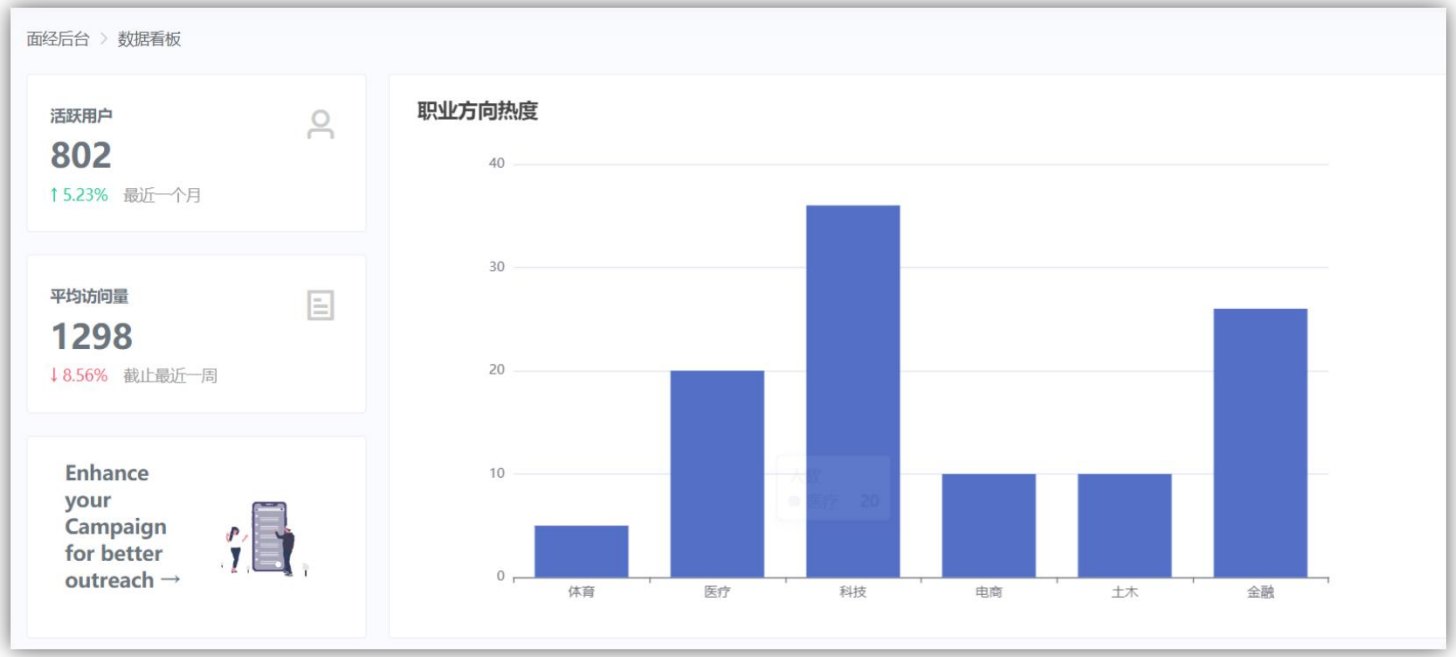
说明：token 是有过期时间的，一旦 token 过期失效了，应该怎么办？




# 数据看板模块【实战】

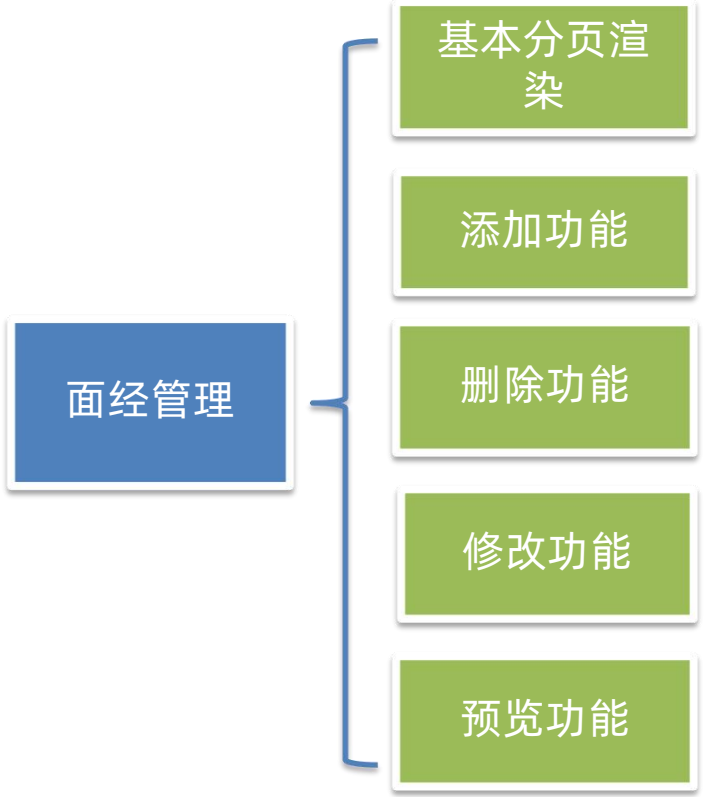
**目标：静态结构 & Vue 中 echarts 的使用**

静态结构 → 装包 → 导入 → 准备容器 → mounted初始化



# 面经管理

## 目标：明确面经管理的核心功能需求

# 面经管理 【核心功能 - 基本分页渲染】

## 目标：搭建基本的架子 & 认识 element-ui 表格组件

# 面经管理 【核心功能 - 基本分页渲染】

**目标：搭建基本的架子 & 认识 element-ui 表格组件**

准备结构 → 请求获取数据 → 结合数据渲染 → 分页效果

表格组件相关：

1. el-table 整个表格组件

   data 数据源

2. el-table-column 表格的列

   ① prop 设置数据源中对象中的键名，即可填入数据

   ② label 列名

   ③ width 列宽


```
<el-table :data="tableData" style="width: 100%">
  <el-table-column prop="date" label="日期" width="180"></el-table-column>
  <el-table-column prop="name" label="姓名" width="180"></el-table-column>
  <el-table-column prop="gender" label="性别"></el-table-column>
  <el-table-column prop="address" label="地址"> </el-table-column>
</el-table>
```

```
tableData: [{
  date: '2016-05-02',
  name: '王小虎',
  address: '上海市普陀区金沙江路 1518 弄'
}, {
  date: '2016-05-04',
  name: '王小虎',
  address: '上海市普陀区金沙江路 1517 弄'
}, {
```

# 面经管理 【核心功能 - 基本分页渲染】

**目标：请求获取数据**


1. 新建 `api/article.js` 封装接口

```
// 获取文章面经列表
export const getArticleList = ({ current, pageSize }) => {
  return request.get('/admin/interview/query', {
    params: {
      current,
      pageSize
    }
  })
}
```

2. `article/index.vue` created中发送初始化获取数据的请求

```
data () {
  return {
    tableData: [],
    current: 1, // 当前页
    pageSize: 10, // 每页条数
    total: 0 // 总条数
  }
},
```

```
created () {
  this.initData()
},
methods: {
  // 发送初始化请求的方法
  async initData () {
    const res = await getArticleList({
      current: this.current,
      pageSize: this.pageSize
    })
    console.log(res)
  }
}
```

# 面经管理 【核心功能 - 基本分页渲染】

**目标：结合数据渲染**

准备结构 → 请求获取数据 → 结合数据渲染 → 分页效果

1. 将数据存入 data 中

```
const { data } = await getArticleList({
  current: this.current,
  pageSize: this.pageSize
})
this.total = data.total
this.tableData = data.rows
```

2. 结合数据 和 表格语法渲染

```
  el-table-column 列组件
    prop 配置数据，配置对象中的属性名
    label 表格的列名
    width 表格的宽度
 -->
<el-table
  :data="tableData"
  style="width: 100%">
  <el-table-column prop="stem" label="标题" width="400"></el-table-column>
  <el-table-column prop="creator" label="作者"></el-table-column>
  <el-table-column prop="likeCount" label="点赞"></el-table-column>
  <el-table-column prop="views" label="浏览数"></el-table-column>
  <el-table-column prop="createdAt" label="更新时间"></el-table-column>
  <el-table-column label="编辑"></el-table-column>
</el-table>
```

| 标题 | 作者 | 点赞 | 浏览数 | 更新时间 |
| --- | --- | --- | --- | --- |
| 域起网络前端开发面经，已经通过 | 尼古拉特斯拉 | 1 | 41 | 2022-01-20 00-00-00 |
| 宇宙头条校招前端面经 | 不风流怎样倜傥 | 53 | 332 | 2022-01-20 00-00-00 |
| 21届秋招猿辅导前端面经笔试 | 多吉利奥 | 9 | 551 | 2022-01-20 00-00-00 |
| 字节跳动前端开发面试题总结 | 醉卧九天 | 38 | 437 | 2022-01-20 00-00-00 |
|  | admin | 0 | 0 | 2022-06-17 10-55-35 |
|  | 谢小飞 | 1 | 59 | 2022-01-20 00-00-00 |

# 面经管理 【核心功能 - 基本分页渲染】

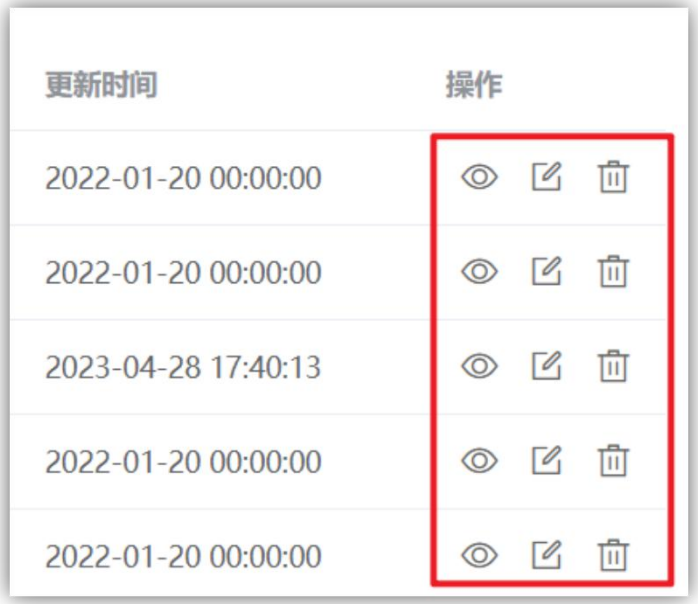
**目标：数据渲染操作列的展示**

说明：列的渲染的两种方式

① prop 渲染

② 作用域插槽渲染 (自定义列)


```
<el-table-column label="操作">
  <!-- obj中有两个常用属性 => $index下标，row一行的数据对象 （遍历时的一个item） -->
  <template #default="obj">
    <!-- {{ obj.$index }} -->
    <!-- {{ obj.row.id }} -->
    <div class="actions">
      <i class="el-icon-view"></i>
      <i class="el-icon-edit-outline"></i>
      <i class="el-icon-delete" @click="del(obj.row.id)"></i>
    </div>
  </template>
</el-table-column>
```



# 面经管理 【核心功能 - 基本分页渲染】

**目标：分页功能完成**


1. 阅读文档，配置分页组件

```
<el-pagination
  background
  @current-change="handleCurrentChange"
  :current-page="current"
  :page-size="pageSize"
  :total="total"
  layout="prev, pager, next"
>
</el-pagination>
```


```
@size-change 监听每页条数的变化
@current-change 监听当前页的变化
:page-sizes 可供选择的每页条数下拉菜单
:page-size 设置当前生效的【每页条数】
:current-page 设置当前生效的【当前页】
:total  设置【总条数】
background 按钮底色
```

2. 基于分页组件的处理函数，完成分页功能

```
// 处理当前页的变化
handleCurrentChange (val) {
  // console.log('当前页变化了', val)
  // 更新数据中的当前页
  this.current = val
  // 重新请求
  this.initData()
}
```

# 面经管理 【核心功能 - 添加功能】

**目标：点击添加，预览，编辑按钮，共用逻辑 → 将来都是显示抽屉**


明确：

1. 关于添加，预览，编辑功能弹出的容器，采用抽屉。 → 【抽屉 vs 对话框】

   抽屉：适合大块的操作区域，例如：表单很长，预览文章...

   对话框：适合中小块的操作区域，例如：简易的表单收集...

2. 点击添加，预览，编辑都要显示抽屉可以共用

```
<el-button @click="openDrawer('add') ...

<i class="el-icon-view" @click="openDrawer('preview', row.id, )"></i>
<i class="el-icon-edit-outline" @click="openDrawer('edit', row.id)"></i>
<i class="el-icon-delete" @click="del(row.id)"></i>

openDrawer (type, id) {
  console.log(type, id)
}
```

# 面经管理 【核心功能 - 添加功能】

**目标：显示抽屉**

1. 准备抽屉组件

```
<!-- 抽屉区域
    1. title="我是标题"
    2. :visible 控制显示隐藏
    3. :direction="direction"  控制方向
    4. :before-close="handleClose" 关闭抽屉前的处理逻辑
       (比如：询问客户是否真的要关闭？)
    5. size="60%" 窗体所占的区域多宽
-->
<el-drawer
  title="我是标题"
  :visible.sync="isShowDrawer"
  direction="rtl"
  :before-close="handleClose"
  size="60%"
>
  <span>我来啦!</span>
</el-drawer>
```

2. 提供抽屉显示隐藏的布尔值 和 处理函数

```
data () {
  return {
    tableData: [],
    current: 1, // 当前页
    pageSize: 10, // 每页条数
    total: 0, // 总条数
    isShowDrawer: false // 控制抽屉的展示
  }
},
```

```
// 共用的打开抽屉的方法
openDrawer (type, id) {
  console.log('触发了', type, id)
  this.isShowDrawer = true
},
handleClose (done) {
  // $confirm 可以弹出一个确认框，可确认.then  可取消.catch
  this.$confirm('你确认要关闭么？').then(() => {
    done() // done() 调用，就代表关闭抽屉
  }).catch(() => {
    console.log('取消')
  })
},
```

# 面经管理 【核心功能 - 添加功能】

**目标：计算属性控制标题**

说明：不同情况下的标题不同，可以根据不同状态，利用计算属性控制标题切换

1. 准备一个变量，drawerType 记录状态 (add / preview / edit)

```
data () {
  return {
    drawerType: 'add' // 默认认为是添加框，add preivew edit
  }
},
```

2. 基于状态，使用计算属性动态的生成一个标题

```
openDrawer (type, id) {
  console.log('触发了', type, id)
  this.drawerType = type
  this.isShowDrawer = true
},
```

```
computed: {
  drawerTitle () {
    let title = '标题'

    if (this.drawerType === 'add') title = '添加面经'
    if (this.drawerType === 'preview') title = '面经预览'
    if (this.drawerType === 'edit') title = '修改面经'

    return title
  }
},
```



# 面经管理 【核心功能 - 添加功能】

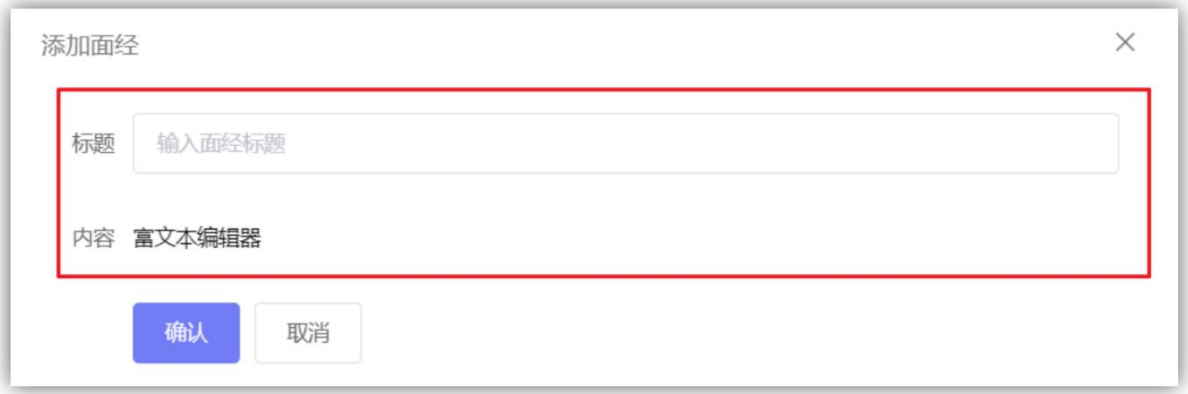
**目标：准备基础表单结构**

**核心：**

① el-form

② el-form-item

③ el-input



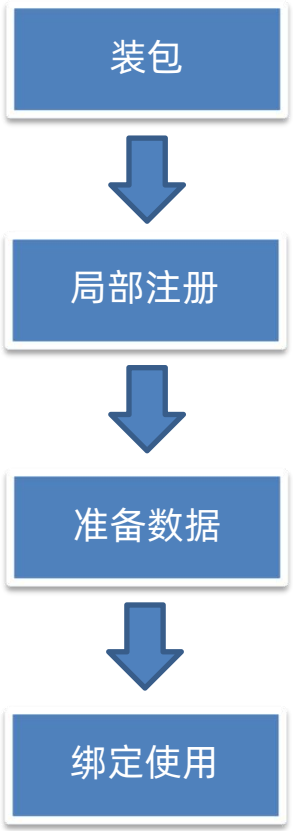

```
<el-form label-width="80px">
  <el-form-item label="标题">
    <el-input placeholder="输入面经标题"></el-input>
  </el-form-item>
  <el-form-item label="内容">
    富文本编辑器
  </el-form-item>
  <el-form-item>
    <el-button type="primary">确认</el-button>
    <el-button>取消</el-button>
  </el-form-item>
</el-form>
```

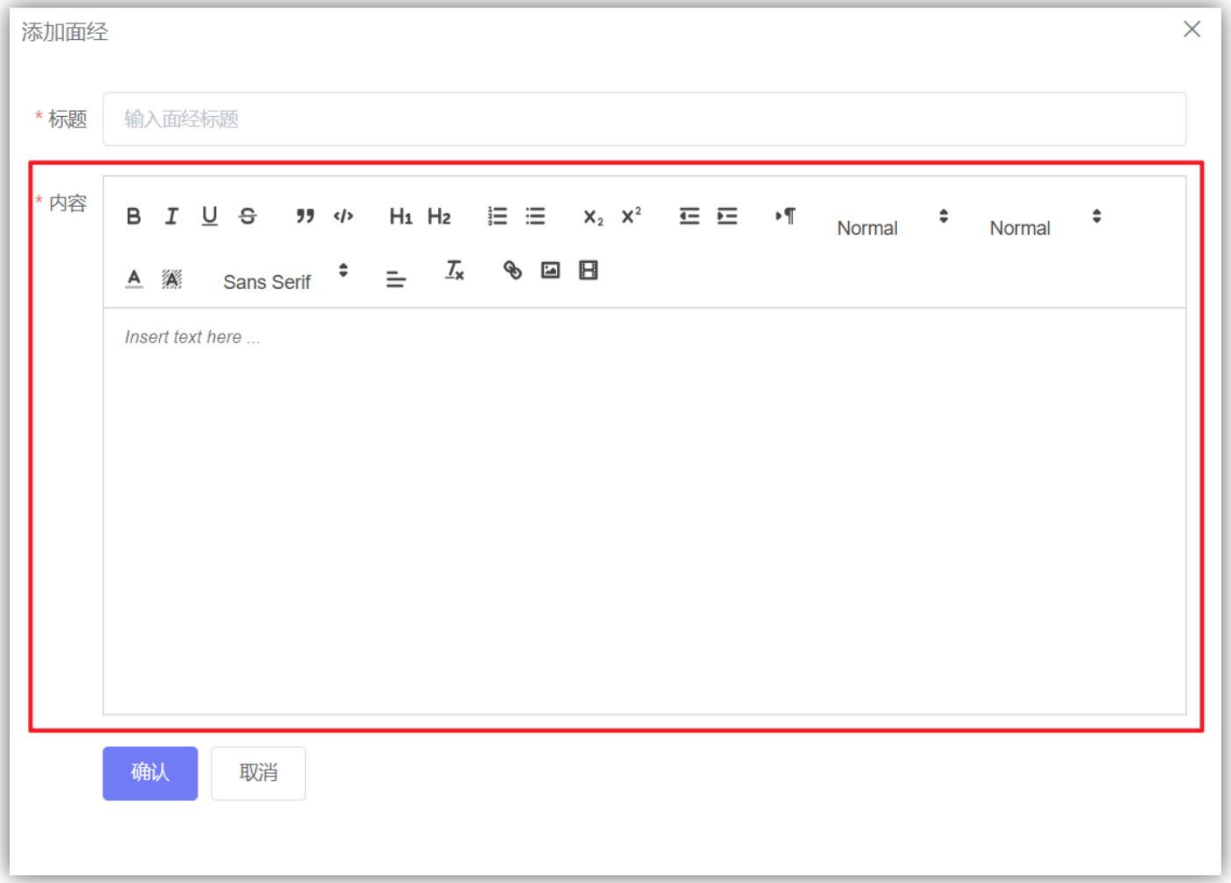
# 面经管理 【核心功能 - 添加功能】

**目标：准备富文本编辑器**

vue-quill-editor

官网



# 面经管理 【核心功能 - 添加功能】

**目标：非空校验**

**说明：**element-ui 自带的校验，对element-ui表单组件才会默认校验，其他组件标签，需要手动校验

1. element-ui 表单组件的校验

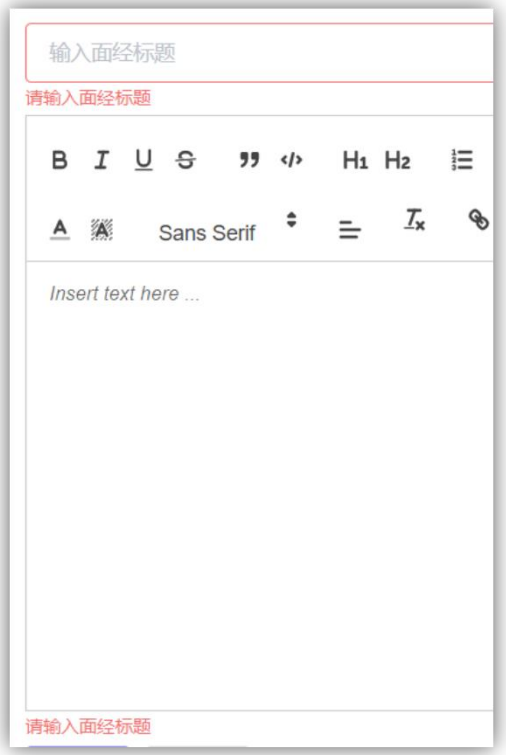
   el-form组件 → :model="form对象" :rules="rules规则"

   el-form-item组件 → prop 字段

   el-input组件 → v-model

2. 富文本编辑器的手动校验

   el-form-item组件 → prop 字段




监听失焦 → 手动校验

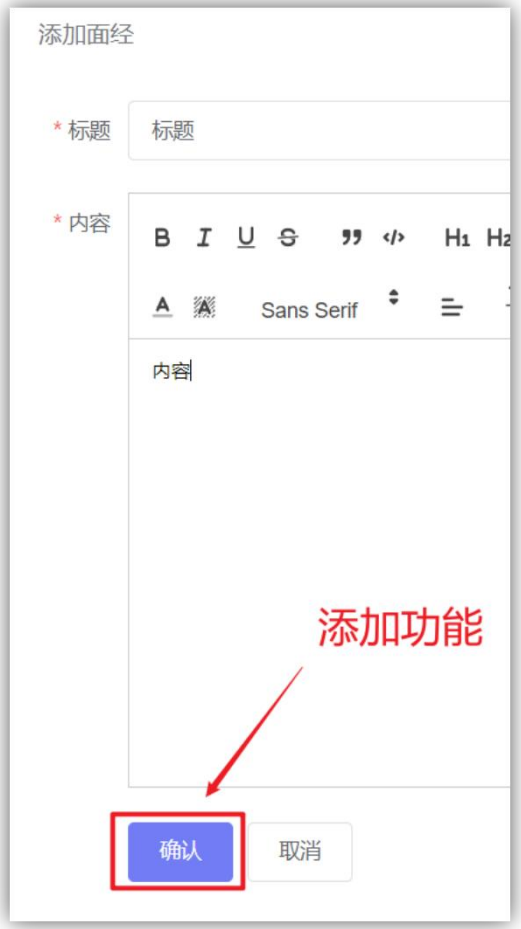
@blur　　this.$refs.form. validateField('字段名')

# 面经管理 【核心功能 - 添加功能】

**目标：封装添加接口，完成添加功能**

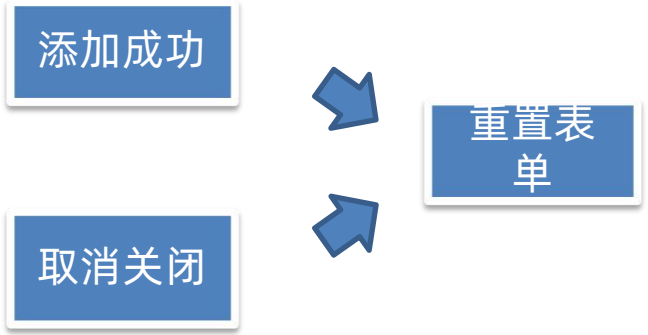
封装接口 → 校验&请求 → 添加提示 → 关闭抽屉 → 重新渲染

```
async submit () {
  // 1. 校验 (对整个表单校验)
  await this.$refs.form.validate()
  // 2. 请求
  await createArticle(this.form)
  // 3. 提示
  this.$message.success('恭喜添加成功')
  // 4. 关闭抽屉
  this.isShowDrawer = false
  // 5. 重新渲染 将当前页重置到第一页(新增的数据，新增到第一页去了)
  this.current = 1
  this.initData()
},
```



# 面经管理 【核心功能 - 添加功能】

**目标：重置表单 → 解决关闭抽屉后，重新打开，内容还在的问题**



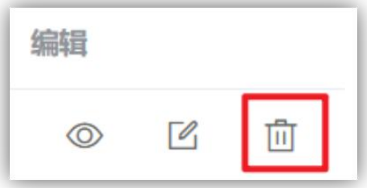

this.$refs.form.resetFields( )

```
closeDrawer () {
  this.$refs.form.resetFields() // 重置表单 和 状态
  this.isShowDrawer = false // 关闭抽屉
},
```

# 面经管理 【核心功能 - 删除功能】

**目标：实现删除功能**

封装接口 → 点击调用 → 添加提示 → 重新渲染 → 处理删除当前页最后一条




```
// 3. 删除文章
export const removeArticle = (id) => {
  return request.delete('/admin/interview/remove', {
    // params: {}, // 地址栏的查询参数传参
    data: { id } // 请求体传参body传参
  })
}
```

```
async del (id) {
  // 删除请求
  await removeArticle(id)
  // 添加成功的提示
  this.$message.success('删除成功')

  // 处理删除当前页的最后一条
  if (this.tableData.length === 1 && this.current > 1) {
    this.current--
  }
  // 重新渲染
  this.initData()
},
```

# 面经管理 【核心功能 - 修改功能】

**目标：修改回显**

说明：修改比添加多一层回显，显示弹框时，需要发送请求获取数据进行展示

1. 封装接口，获取对应 id 的文章详情

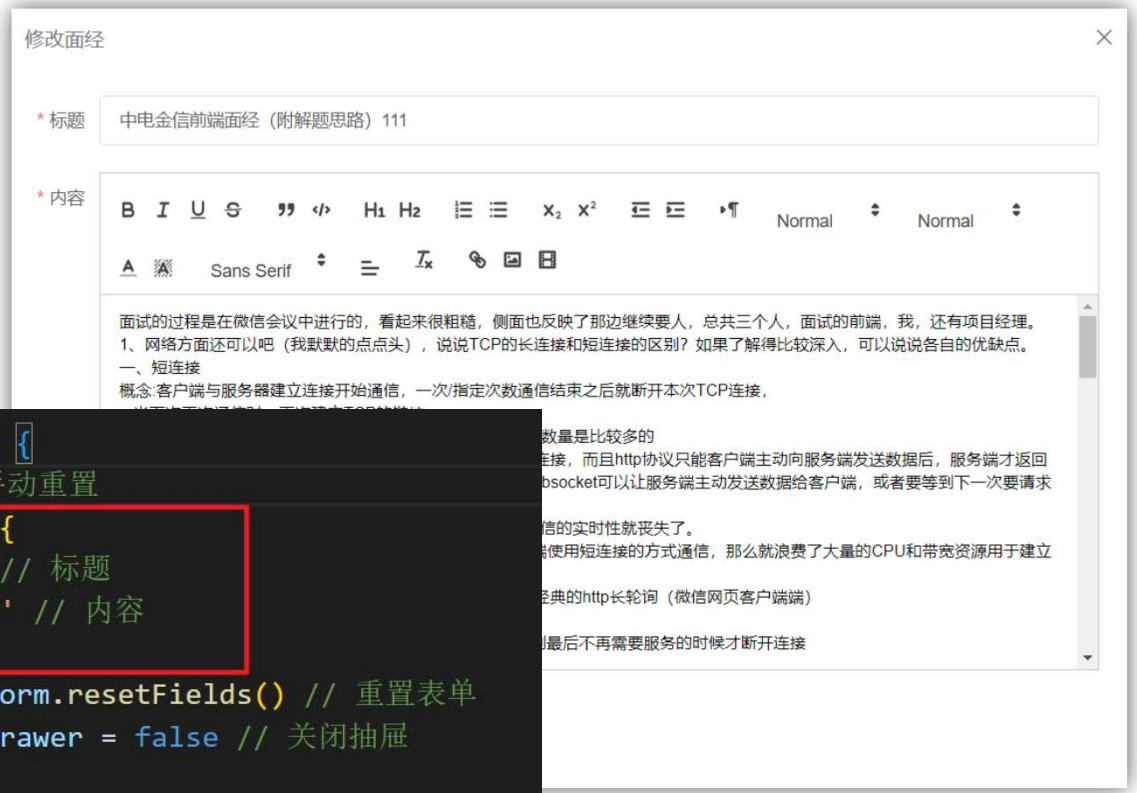
```
export const getArticleDetail = id => {
  return request.get('/admin/interview/show', {
    params: {
      id
    }
  })
}
```

2. 显示抽屉的同时，将数据存入 form 回显

```
async openDrawer (type, id) {
  this.drawerType = type
  this.isShowDrawer = true

  if (type !== 'add') {
    const res = await getArticleList(id)
    this.form = {
      ...res.data
    }
  }
},
```

```
closeDrawer () {
  // 将form也手动重置
  this.form = {
    stem: '', // 标题
    content: '' // 内容
  }
  this.$refs.form.resetFields() // 重置表单
  this.isShowDrawer = false // 关闭抽屉
},
```



# 面经管理 【核心功能 - 修改功能】

**目标：完成修改提交功能**

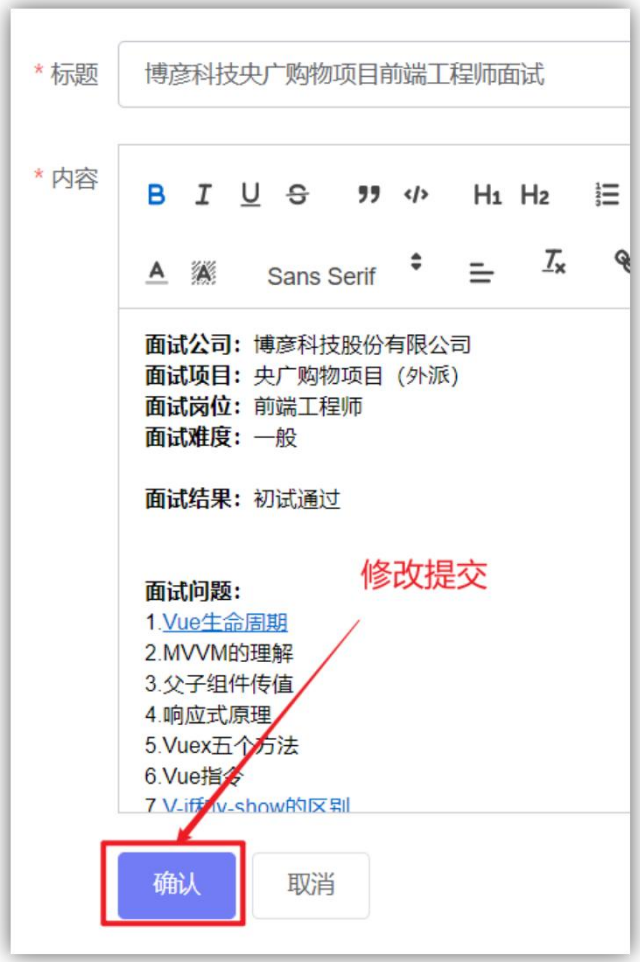
说明：修改 和 添加 共用同一个按钮，需要进行判断才能复用

1. 封装根据 id 进行提交修改的接口

```
// 5. 根据id提交修改文章
export const updateArticle = ({ id, stem, content }) => {
  return request.put('/admin/interview/update', { id, stem, content })
}
```

2. 按钮逻辑中添加判断，区分不同场景，调用接口完成功能

```
if (this.drawerType === 'add') {
  // 请求
  await createArticle(this.form)
  // 提示
  this.$message.success('恭喜添加成功')
}
if (this.drawerType === 'edit') {
  // 请求
  const { id, stem, content } = this.form
  await updateArticle({ id, stem, content })
  // 提示
  this.$message.success('恭喜修改成功')
}
```



# 面经管理 【核心功能 - 预览功能】

**目标：完成预览功能**

说明：其实预览已经完成了，也是进行回显，只是不是利用表单回显，而是直接回显文章内容

语法：v-html

```
<!-- 只有添加和编辑需要表单 / 预览不需要表单 -->
<div v-if="drawerType === 'preview'" class="article-preview">
  <h5>{{ form.stem }}</h5>
  <div v-html="form.content"></div>
</div>

<el-form v-else ref="form" :model="form" :rules="rules" label-width="80px">
```

```
closeDrawer () {
  // 将form也手动重置
  this.form = {
    stem: '', // 标题
    content: '' // 内容
  }
  // 只有非预览的情况(添加/修改) 需要重置
  if (this.drawerType !== 'preview') {
    this.$refs.form.resetFields() // 重置表单
  }
  this.isShowDrawer = false // 关闭抽屉
},
```


传智教育旗下高端IT教育品牌