一、构建Vue3项目

vue create vue3-project

main.js文件的改变

```
import { createApp } from 'vue'
import App from './App.vue'
import router from './router'
const app = createApp(App);
app.use(router).mount('#app')
```

二、element-plus

1. 全部引入

在终端安转：

npm install element-plus --save

main.js文件引入：

```
import ElementPlus from 'element-plus'
import 'element-plus/dist/index.css'

const app=createApp(App)
app.use(VueElementPlus)
app.mount('#app')


```

使用：直接在组件里编写element代码即可

2. 按需引入

在终端安转：

npm install element-plus --save

在终端安装插件：

npm install -D unplugin-vue-components unplugin-auto-import

官网配置在 vite.config.ts文件引入

```
//引入部分
import AutoImport from 'unplugin-auto-import/vite'
import Components from 'unplugin-vue-components/vite'
import { ElementPlusResolver } from 'unplugin-vue-components/resolvers'

//plugins数组部分加入
  plugins: [
    AutoImport({
      resolvers: [ElementPlusResolver()],
    }),
    Components({
      resolvers: [ElementPlusResolver()],
    }),
  ],

```

使用：直接在组件里编写element代码即可

vue.config.js配置

```
const { defineConfig } = require('@vue/cli-service')

// 引入 element UI
const AutoImport = require('unplugin-auto-import/webpack')
const Components = require('unplugin-vue-components/webpack')
const { ElementPlusResolver } = require('unplugin-vue-components/resolvers')

module.exports = defineConfig({
  transpileDependencies: true,
  configureWebpack: {
    plugins: [
      AutoImport({
        resolvers: [ElementPlusResolver()],
```

```
      }),
      Components({
        resolvers: [ElementPlusResolver()],
      }),
    ],
  }
})
```

3. element-plus图标的引入和使用

- 在终端安转：

npm install element-plus --save

- 终端安转：

npm install @element-plus/icons-vue

- main.js引入：

```
import * as ElementPlusIconsVue from '@element-plus/icons-vue'

const app = createApp(App)

for (const [key, component] of Object.entries(ElementPlusIconsVue)) {
  app.component(key, component)
}

app.mount('#app')

```

1.静态引入图标

- 直接点击想要的图标图案，就可以复制相关代码

```
//例如加号图标
<el-icon><Plus /></el-icon>

```

2.动态引入图标

```
<!-- 遍历菜单栏-->
    <el-menu-item :index="item.path" v-for="item in noChildren()" :key="item.path">

        <!-- 根据遍历得到的item，动态引入图标 -->
       <component class="icons" :is="item.icon"></component>

    </el-menu-item>

```

单独安装less 引入less

终端引入less：npm install less-loader less --save-dev

5. 基础样式引入

在src/assets新建文件夹less

在less文件夹新建reset.less文件，这个是全局样式

在less文件夹新建index.less文件，里面只写@import './reset.less';

在main.js中引入index.js文件 ， import './assets/less/index.less'

三、vue3引入路由

1. 终端安装：脚手架创建已经自带router

```
npm install vue-router -S
```

2. 新建文件：src / router / index.js：文件配置如下

```
import { createRouter, createWebHistory } from 'vue-router'
import layout from '../views/layout'
import Login from "@/views/Login"

const routes = [
  {
    path: '/',
    component: layout,
    children:[
```

```
10
11      ]
12    },
13   {
14    path:'/login',
15    component:Login
16   }
17 ]
18
19 const router = createRouter({
20   history: createWebHistory(process.env.BASE_URL),
21   routes
22 })
23
24 export default router
25
```

1. main.js文件引入

```
1 import router from './router'
2
3 const app=createApp(App)
4 app.use(router)
5 app.mount('#app')
6
```

1. 使用：在具体的组件中需要导入

```
1 import {useRouter} from 'vue-router' //导入1
2 export default {
3   setup() {
4      let router=useRouter()//声明2
5         //下面可以配置方法进行路由跳转
6      }
7 }
```

1. 需要显示路由组件的地方添加

<router-view></router-view>

四、echarts使用 按需加载

1. 安装

npm i echarts -S

1.定义echarts

```
// 引入 echarts 核心模块，核心模块提供了 echarts 使用必须要的接口。
import * as echarts from 'echarts/core';
// 引入柱状图图表，图表后缀都为 Chart
import { BarChart,LineChart } from 'echarts/charts';
// 引入提示框，标题，直角坐标系，数据集，内置数据转换器组件，组件后缀都为 Component
import {
  TitleComponent,
  TooltipComponent,
  GridComponent,
  DatasetComponent,
  TransformComponent
} from 'echarts/components';

// 标签自动布局，全局过渡动画等特性
import { LabelLayout, UniversalTransition } from 'echarts/features';
// 引入 Canvas 渲染器，注意引入 CanvasRenderer 或者 SVGRenderer 是必须的一步
import { CanvasRenderer } from 'echarts/renderers';

// 注册必须的组件
echarts.use([
  TitleComponent,
  TooltipComponent,
  GridComponent,
  DatasetComponent,
  TransformComponent,
  BarChart,
  LineChart,
  LabelLayout,
  UniversalTransition,
  CanvasRenderer
]);
```

```
32
33
34 //导出
35 export default echarts
36
37 // // 接下来的使用就跟之前一样，初始化图表，设置配置项
38 // var myChart = echarts.init(document.getElementById('main'));
39 // myChart.setOption({
40 //   // ...
41 // });
```

3. 把自己创建好的echarts.js文件引入全局main.js中

```
1 import App from './App'
2 // 引入echarts
3 import echarts from './common/js/echarts.js'
4
5 import {createSSRApp} from 'vue'
6 let app = createSSRApp(App)
7
8 // 挂载到vue实例中
9 // Vue.prototype.$echarts = echarts;//vue2的挂载方式
10 app.config.globalProperties.$echarts = echarts;//vue3的挂载方式
11 //provide使用数据
12 app.provide('$echarts',echarts)
13 export function createApp() {
14  return {app}
15 }
16
17 //调用的时候就是 :   this.$echarts.init()
```

2.组件使用

```
1 <template>
2   <h2>home</h2>
3   <div id="myEChartsBar" style="width: 500px; height: 300px"></div>
4 </template>
5
```

```vue
<script>
import { inject, onMounted } from "@vue/runtime-core";
import { useRoute } from "vue-router";
export default {
  setup(props, content) {
    //获取echarts
    const echarts = inject("$echarts");
    console.log("echarts----", echarts);

    //绘制矩形
    const bar = () => {
      const myEChart = echarts.init(document.getElementById("myEChartsBar"));
      const option = {
        xAxis: {
          data: ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"],
        },
        yAxis: {},
        series: [
          {
            type: "line",
            data: [23, 24, 18, 25, 27, 28, 25],
          },
        ],
      };
      myEChart.setOption(option);
    };
    //生命周期函数-------------
    onMounted(() => {
      bar()
    });
  },
};
</script>

<style>
</style>
```

五、vue3注册添加全局实例属性的方法，如何在setup函数中调用

1.第一种方式

在 Vue 2 中， Vue.prototype 通常用于添加所有组件都能访问的 property。 在 Vue 3 等同于config.globalProperties。这些 property 将被复制到应用中作为实例化组件的一部分。

```
// 之前 - Vue 2
 Vue.prototype.$http = () => {}

// 之后 - Vue 3
const app = createApp({})
app.config.globalProperties.$http = () => {} ```

```

上面是官方给得说明

下面是我的具体用法：

在vue3中注册一个全局实例方法

```
import {createApp} from 'vue'
import App from './App.vue'

const app = createApp(App)
        app.config.globalProperties.$echarts = echarts;//vue3的挂载方式

```

在vue实例的setup()函数中没有this对象，如何调用实例方法$translate呢

```
import {getCurrentInstance} from "vue";

        const internalInstance = getCurrentInstance();
        const request =
        internalInstance.appContext.config.globalProperties.$echarts

```

2.第二种方式：通过provide注入

与在 2.x 根实例中使用 provide 选项类似，Vue 3 应用实例也提供了可被应用内任意组件注入的依赖项：

```
                //入口处全局导入
        app.provide("dayjs", dayjs)
        app.provide("request",request)

```

```
<script>
import {toRef, nextTick, inject, ref, computed} from "vue"
export default {
 //普通用法
  inject: ['dayjs'],
  setup(props, {emit}) {
  //setup中的用法
    const dayjs = inject("dayjs")

  }
}
</script>

```

使用 provide 在编写插件时非常有用，可以替代 globalProperties。
在应用之间共享配置
上面也是官方推荐得方式

六、Element-plus 分页组件英文转为中文

方法1：全局引入

1. 在main.js或者app.vue中添加如下的代码

```
import ElementPlus from 'element-plus';
import locale from "element-plus/lib/locale/lang/zh-cn";//需要新加的代码
app.use(ElementPlus, {locale});//需要改变的地方，加入locale
```

2. 完整代码演示

```
<template>
```

```
  <el-pagination
    v-model:current-page="currentPage"
    v-model:page-size="props.pageSize"
    layout="total, prev, pager, next, jumper"
    :total="props.total"
    @current-change="handleCurrentChange"
  />
</template>

<script setup>
import { ref,reactive } from "vue";
const emit = defineEmits(['getCurrentPage'])
const props = defineProps(['total','pageSize'])
const currentPage = ref(1);
const handleCurrentChange = (val) => {
    console.log(`current page: ${val}`);
    emit('getCurrentPage',val)
};
</script>
```

如何上述报错 解决办法：在vite.config.js里添加

```
optimizeDeps:{
    include: ['element-plus/lib/locale/lang/zh-cn']
}
```

方法2：按需引入

app.vue 组件或者是使用的组件里面配置

```
<template>
  <el-config-provider :locale="locale">
    <router-view/>
  </el-config-provider>
</template>
<script setup>
import { ElConfigProvider } from 'element-plus'
```

```
import zhCn from 'element-plus/lib/locale/lang/zh-cn';
import {reactive} from 'vue'

const { locale } = reactive({
  locale: zhCn,
});

</script>

```

七、setup函数写async await

1. 第一种方法 使用suspense 包裹你的组件 感觉不太好

```
<template>
 <suspense>
        <router-view></router-view>
</suspense>
</template>

export default {
  async setup() {
    // 在 `setup` 内部使用 `await` 需要非常小心
    // 因为大多数组合式 API 函数只会在
    // 第一个 `await` 之前工作
    const data = await loadData()

    // 它隐性地包裹在一个 Promise 内
    // 因为函数是 `async` 的
    return {
      // ...
    }
  }
}
```

2. 第二种方法 使用生命周期钩子

```
setup() {
    const users = ref([]);
    onBeforeMount(async () => {
      const res = await axios.get("https://jsonplaceholder.typicode.com/users");
      users.value = res.data;
      console.log(res);
    });

    return {
      users,
    };
  },
```

或者普通函数

```
<script setup>
const loadNode =async (node, resolve) => {
  // console.log('node----', node);
  if (node.level === 0) {
    //自动加载--进入页面执行
    let result =await goodsItemCategory(1);
    console.log('result',result);
    return resolve(result);
  }
  if (node.level >= 1) {
    //点击tree 获取点击的层级级别 请求对应的数据
    // console.log('请求子级菜单',node.data);
    let result =await  goodsItemCategory(node.data.cid);
    return resolve(result);
  }
};
//获取tree数据-----------------------------------
const goodsItemCategory =async (type) => {
 let res = await api.goodsItemCategory({ type })
    console.log('tree-----', res.data);
    if (res.data.status === 200) {
      return res.data.result;
    } else {
```

```
        return [];
    }
};
</script>
```

八、props emit

1. props

在使用 <script setup> 的单文件组件中，props 可以使用 defineProps() 宏来声明

```
<script setup>
const props = defineProps(['foo'])
console.log(props.foo)
</script>
```

2. 声明触发的事件

组件要触发的事件可以显式地通过
defineEmits() 宏来声明：

```
<template>
  <el-tree :props="props" :load="loadNode" lazy show-checkbox @node-click="clickTree"
/>
</template>
<script setup>
const emit = defineEmits(['changeTree', 'submit'])

const clickTree=(data,node)=>{
            // console.log('点击了tree',data,node);
            emit('changeTree',data)
        }
</script>

```

九、Pinia使用

1. 安装和配置

（1）安装

```
npm i pinia
```

（2）在main.js中挂载pinia

```
import { createapp } from 'vue'
import app from './app.vue'
import { createpinia } from 'pinia'
const pinia = createpinia()
createapp(app).use(pinia).mount('#app')
```

在src文件下创建一个store文件夹，并添加index.js

（3）新建文件store/counter.js

```
import { defineStore  } from 'pinia'
// 创建store,命名规则： usexxxxstore
// 参数1：store的唯一标识
// 参数2：对象，可以提供state actions getters
const usecounterstore = defineStore ('counter', {
  state: () => {
    return {
      count: 0,
    }
  },
  getters: {
  },
  actions: {
  },
})
export default usecounterstore
```

(4) 在组件中使用

```
<template>
  <div class="about">
```

```
    <h1>This is an about page</h1>
    <p>num:{{counter.count}}</p>
  </div>
</template>
<script setup>
import usecounterstore from '../store/counter'
const counter = usecounterstore()
</script>
```

2. pinia数据持久化

目标： 通过 pinia 插件快速实现持久化存储。

1.安装

```
yarn add pinia-plugin-persistedstate
or
npm i  pinia-plugin-persistedstate
```

使用插件 在main.js中注册

```
import { createapp } from "vue";
import app from "./app.vue";
import piniapluginpersistedstate from 'pinia-plugin-persistedstate'

const pinia = createpinia();
pinia.use(piniapluginpersistedstate);
createapp(app).use(pinia);
```

2. 模块开启持久化

```
const usehomestore = definestore("home",{
  // 开启数据持久化
  persist: true
  // ...省略
});
```

十、wangEditor

1. 安装

```
yarn add @wangeditor/editor
# 或者 npm install @wangeditor/editor --save

yarn add @wangeditor/editor-for-vue@next
# 或者 npm install @wangeditor/editor-for-vue@next --save
```

2. 使用

```
<template>
    <div style="border: 1px solid #ccc">
      <Toolbar
        style="border-bottom: 1px solid #ccc"
        :editor="editorRef"
        :defaultConfig="toolbarConfig"
        :mode="mode"
      />
      <Editor
        style="height: 500px; overflow-y: hidden;"
        v-model="valueHtml"
        :defaultConfig="editorConfig"
        :mode="mode"
        @onCreated="handleCreated"
      />
    </div>
</template>
```

script

```
<script>
import '@wangeditor/editor/dist/css/style.css' // 引入 css

import { onBeforeUnmount, ref, shallowRef, onMounted } from 'vue'
import { Editor, Toolbar } from '@wangeditor/editor-for-vue'
```

```
6
7  export default {
8    components: { Editor, Toolbar },
9    setup() {
10     // 编辑器实例，必须用 shallowRef
11     const editorRef = shallowRef()
12
13     // 内容 HTML
14     const valueHtml = ref('<p>hello</p>')
15
16     // 模拟 ajax 异步获取内容
17     onMounted(() => {
18         setTimeout(() => {
19             valueHtml.value = '<p>模拟 Ajax 异步设置内容</p>'
20         }, 1500)
21     })
22
23     const toolbarConfig = {}
24     const editorConfig = { placeholder: '请输入内容...' }
25
26     // 组件销毁时，也及时销毁编辑器
27     onBeforeUnmount(() => {
28         const editor = editorRef.value
29         if (editor == null) return
30         editor.destroy()
31     })
32
33     const handleCreated = (editor) => {
34       editorRef.value = editor // 记录 editor 实例，重要!
35     }
36
37     return {
38       editorRef,
39       valueHtml,
40       mode: 'default', // 或 'simple'
41       toolbarConfig,
42       editorConfig,
43       handleCreated
44     };
45   }
```

```
}
</script>
```

十一、导出Excel

1. 安装模块

```
npm install xlsx file-saver -S
```

2. 导入文件

```
import * as XLXS from "xlsx";
import FileSaver from "file-saver";
```

3. 模板使用

```
    <download-excel
         style="display:inline-block;margin-left:10px;"
         :data="DetailsForm"
         :fields="json_fields"
         :header="title"
         :name="title">
          <el-button class="order-btn" type="warning" size="small" @click="download">导出</el-button>
        </download-excel>
```

4. 事件

```
//导出表格=---------------
    const download = () => {
        // 导出文件名
```

```
        // const filename = state.excelTitle
        // 通过id，获取导出的表格数据
        const wb = XLXS.utils.table_to_book(document.getElementById("table"), {
            raw: true,
        });
        const wbout = XLXS.write(wb, {
            bookType: "xlsx",
            bookSST: true,
            type: "array",
        });
        try {
            FileSaver.saveAs(new Blob([wbout], {
                    // 定义文件格式流
                    type: "application/octet-stream",
                }),
                name.value + ".xlsx"
            );
        } catch (e) {
            console.log(e);
        }
        return wbout;
    }

```