

# Vue Note

## 教程地址



<https://www.bilibili.com/video/BV17RcMzrE4W>

## Vue 环境安装

安装 `vue` 脚手架



```
npm intall -g @vue/cli  
vue create vue-demo
```

之后就可以在控制台内选择默认模板或者手动选择特性，一般使用手动选择特性

使用空格选择，只需要选中Babel 和 PWA Support就行，学习阶段取消 Linter 和 Formatter

选择版本，之后选择配置文件位置，在哪里都可以

一般不保存为未来项目的预制

## 模板语法

### 文本插值与 `v-html` 指令

使用文本插值 `{{var}}` 来进行文本的动态控制，使用 `v-html` 来传入html字符串



```
<template>  
  <div>{{message}}</div>  
  <div v-html = "rawhtml"></div>  
</template>
```

```
<script>  
  export default{  
    data(){  
      return{
```

```
        message:"hello",
        rawHtml:<a>world</a>
    }
}
</script>
```

## || 动态属性绑定

这样可以动态绑定属性, `v-bind` 可以简写成 `:`



```
<template>
  <div v-bind:class="className"></div>
  <div :class="className"></div>
</template>

<script>
  data(){
    return{
      className:"myClass",
    }
  }
</script>
```

### js表达式支持

动态内容中的是js表达式, 而不是单纯的变量

但是注意是表达式而不是语句, 流程控制也不会生效, 需要用三目运算符



# 条件渲染

即根据条件不同来渲染不同的页面内容

## || `v-if` 指令用于条件性地渲染不同的内容



```
<template>
  <div v-if="nowRender === true">我是nowRender为true时渲染出来的</div>
  <div v-else>我是nowRender为False的时候渲染出来的</div>
  <button @click="nowRender = !nowRender">切换</button>
</template>
```

```
<script>

  export default {
    data(){
      return{
        nowRender:true,
      }
    }
  }

</script>
```

💡 `v-show` 标签也可以判断是否显示，用法和 `v-if` 基本相同，但是：  
`v-if` 是真正控制dom的渲染的，但是 `v-show` 是进行css层面的可见和不可见

## 列表渲染

💡 即通过 `v-for` 来循环渲染一个列表，其基于一个数组

基本使用语法如下

```
<ul>
  <li v-for="item in items">{{item.message}}</li>
</ul>
```

✍ 注意Key的使用

在 `v-for` 循环生成的元素中需要有一个 `key` 值，这个值一般使用对象内容中的 `id`

```
<template>
  <tr v-for="i in peopleList" :key = "i.id">
    <td>{{i.id}}</td>
    <td>{{i.name}}</td>
    <td>{{i.tel}}</td>
  </tr>
</template>
<script>
  export default{
    data(){
      return{
```

```
    peopleList:[
        {id:1,name:"xxx",tel:123},
        {id:2,name:"yyy",tel:1234}
    ]
}
}

</script>
```

## || 遍历的时候实际上 `v-for` 指令中有两个参数



```
<li v-for = "(item,index) in newsList">
```

上面的指令中的 `index` 是指的数组下标

# 事件处理

## || 监听事件

使用 `v-on` 指令来添加一个事件，通常缩写为 `@` 符号，来监听dom事件



```
<button @click = "counter += 1">Add 1</button>
<script>
    return{
        counter : 0;
    }
</script>
```

## || 事件处理方法

可以在method中写方法，然后给 `v-on` 指令去调用方法名称



```
<template>
    <div @click="test()"></div>
</template>

<script>
    data(){
        return{

```

```
        }
    },
methods:{
    test(event){
        console.log(event);
        //这里的事件对象就是原生的事件对象
    }
}
</script>
```

## || 事件传递参数

我们所写的methods是可以有参数传递的

```
<template>
    <div @click = "say('hi')">点我</div>
</template>
<script>
methods:{
    say(text){
        console.log(text);
    }
}
</script>
```

# 表单输入绑定

可以用 `v-model` 指令在表单 `<input> <textarea> <select>` 等元素上创建双向数据绑定，会根据控件类型自动选取正确的方法来更新元素

## || 基本用法

```
<template>
    <input v-model = "message">
    <p>message is : {{Message}}</p>
</template>

<script>
data(){
    return{
        message:"",
    }
}
```

```
    }
</script>
```

## || 修饰符

.lazy ::

在默认情况下，`v-model` 在每次输入之后进行输入框的值与数据的值的同步，通过该修饰符可以转化为在 `change` 事件之后进行同步，即为失去焦点或者回车的时候进行获取，修饰符的用法如下

```
● ● ●

<input v-model.lazy="message" />

<script>
  data(){
    return{
      message:"",
    }
  }
</script>
```

.trim ::

该修饰符可以自动过滤用户输入的首尾空白字符

```
<input v-model.trim="message" />
```

# 组件基础

## || 单文件组件

Vue 单文件组件，后缀名为 `.vue` 是一种特殊的文件格式，允许将vue组件的模板、逻辑与样式封装在单个文件中

```
● ● ●

<template>
  <h3>基本的单文件组件</h3>
</template>

<script>
  export default{
    name:"MyComponent", //组件的名字
  }
</script>

<style>

</style>
```

## || 组件的使用

### 1. 引入组件

```
import MyComponent from "./xxx/xxx.vue"
```

### 2. 注册组件

```
<script>
  export default{
    name:"App",
    components:{
      MyComponent,
    },
  }
</script>
```

### 3. 使用组件

```
<template>
  <MyComponent />
  <MyComponent></MyComponent>
</template>
```

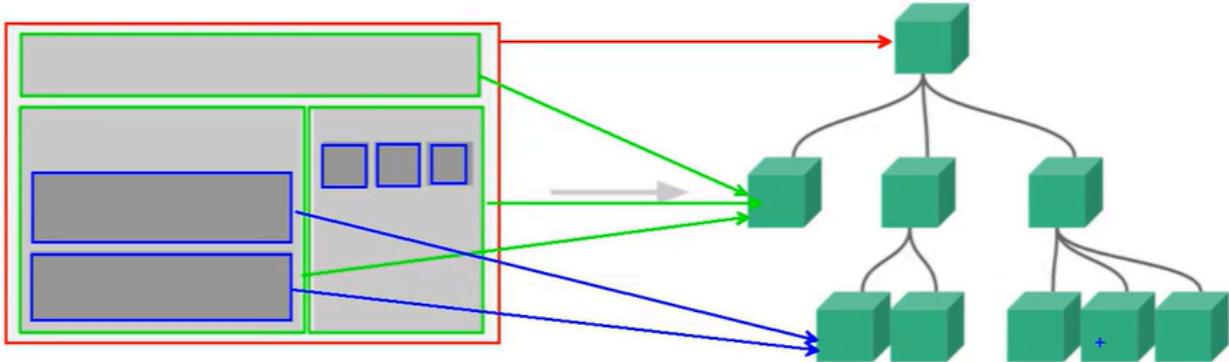
## || `<style>` 标签的 `scoped` 属性

```
<style scoped></style>
```

这个属性可以让css样式只在该组件中生效

## || 组件的组织

组件通常会以一个组件树的方式来组织



## Props 属性传递组件交互

`props` 是在组件上定义的一些自定义的属性,这样的传递是没有类型限制的

1. 在父组件上设置数据· `message`



```
<script>
  data(){
    return{
      message:"hello Props",
    }
  }
</script>
```

2. 在子组件上设置一个自定义属性 `props`



```
<script>
  props:{
    getMessage:{
      type:String,
      default:"",
    }
  }
</script>
```

3. 在父组件使用这个属性来向子组件传递数据



```
<template>
  <xxx :getMessage="message">
</template>

<script>
  imports xxx from "xxx.vue"
  components: {
    xxx,
  },
  data(){
    return{
      message:"hello Props",
    }
  }
</script>
```

#### 4. 在子组件使用文本插值来使用数值



```
<template>
  <div>{{getMessage}}</div>
</template>
```

特别注意：如果默认值是数组或者函数，那么必须需要使用一个函数进行返回



```
<script>
  props:{
    needArray:{
      default:() => {
        return []
      }
    }
  }
<]/script>
```

## 自定义事件的组件交互

自定义事件可以在组件中反向传递数据，可以利用自定义事件实现 `%emit`

#### 1. 子组件中有一个数据message需要传递到父组件中：

```
<script>
  data(){
    return{
      message:"我是子组件的消息"
    }
  }
</script>
```

2. 在子组件中定义一个方法来抛出事件给父组件，其中采用 `$emit("name", "value")` 的方式来创建自定义事件

```
<template>
  <button @click = "sendEmitMessage">点我发送数据</button>
</template>

<script>
  data(){
    return{
      message:"我是子组件的消息"
    }
  },
  methods:{
    sendEmitMessage(){
      this.$emit("onMessSend" , this.message);
    }
  }
</script>
```

3. 父组件中也需要一个方法来接受子组件传来的数据，传来的数据会到事件参数 `event` 中去

```
<template>
</template>

<script>
  data(){
    return{
      gottenMessage:"",
    }
  },
  methods:{
```

```
getEmitMessage(event){  
    this.gottenMessage = event;  
}  
}  
</script>
```

#### 4. 需要使用那个组件来监听相应事件

```
<template>  
    <xxx @onMessSend="getEmitMessage" />  
    <div> {{gottenMessage}} </div>  
</template>  
  
<script>  
    import xxx from "./xxx"  
    components:{  
        xxx  
    }  
    data(){  
        return{  
            gottenMessage:"",  
        }  
    },  
    methods:{  
        getEmitMessage(event){  
            this.gottenMessage = event;  
        }  
    }  
</script>
```

## 网络请求封装

安装

Axios

```
npm install -save axios
```

具体操作见代码：

```
utils/request.js  
api/path.js
```

```
api/index.js  
components/axios_network.vue
```

# 网络请求跨域解决方案

`js` 采取的是同源策略，浏览器只允许js代码请求和当前服务器

域名、端口、协议相同的数据接口上的数据，这就是同源策略

前台通过 `Proxy` 来解决跨域问题：

在 `vue.config.js` 中去添加如下代码：

```
devServer:{  
  proxy:{  
    'api':{  
      target : 'url',//这里添加根域名  
      changeOrigin:true  
    }  
  }  
}
```

以上代码通过配置vue开发服务器的方式来在开发环境下解决跨域问题

解决完跨域配置之后要重新启动服务器

# Vue 引入路由配置

在vue中，我们可以使用 `vue-router` 路由管理页面之间的关系

是 `Vue.js` 的官方路由，让用 `Vue.js` 来实现单页面应用更为简单

## || 创建路由

1. 安装 `vue-router`

```
npm install --save vue-router
```

2. 创建一个新的 `router` 文件夹，放在src文件夹中，新建 `index.js` 是路由的配置文件

3. 创建页面，创建一个新的 `views` 文件夹，在其中创建页面

4. 引入页面，在 `index.js` 中引入页面和需要的资源

```
// index.js  
import {createRouter , createWebHashHistory} from "vue-router";  
import HomeView from "../views/HomeView.vue";  
import AboutView from "../views/AboutView.vue";
```

## 5. 配置 `routes` 数组来定义页面之间的关系

```
// index.js
const routes = [
  {
    path: "/", //path为一个单斜杠代表根页面
    component: HomeView,
  },
  {
    path: "/about",
    component: AboutView,
  }
]
```

更推荐使用以下的写法:

优势在于异步加载和写法更加简单清晰，但是注意首页一般仍然用上面的加载方式，这有助于提升首页的加载速度

```
const routes = [
  {
    path: "/",
    component: () => import("../views/HomeView.vue"),
  },
  {
    path: "/about",
    component: () => import("../views/AboutView.vue"),
  },
  {
    path: "/news",
    component: () => import("../views/NewsView.vue"),
  }
]
```

## 6. 创建路由

```
// index.js
const router = createRouter({
  history: createWebHashHistory(),
  routes
})

export default router; //导出路由
```

7. 在主入口文件 `main.js` 引入路由并作如下修改：通过 `.use` 来明确安装路由功能

```
//main.js
import router from './router'
createApp(App).use(router).mount('#app');
```

8. 在 `app.vue` 中插入路由的显示入口

```
<template>
  <!--路由的显示入口-->
  <router-view></router-view>
</template>
```

## || 路由跳转 `router-link`

在 `App.vue` 的模板中加入 `<router-link>` 标签即可实现路由跳转

```
<!--App.vue-->
<template>
  <router-link to="/">首页</router-link>
  <router-link to="/about">关于</router-link>
  <router-view></router-view>
</template>
```

## || 配置中的 `history` 的不同配置

有两种常用的不同配置 `createWebHistory` 和 `createWebHashHistory`

他们的区别是：hash的路径有 `\#\`` 和 `\#\`about` 而没有Hash的方法没有 `\#\``

```
[bili-class-router](http://localhost:8080/#/about)
```

上面的方法的原理是的锚点链接

```
[bili-class-router](http://localhost:8080/about)
```

上面的方法的原理是H5的pushState()

并且 `createWebHistory` 需要后端配合做重定向，否则会出现404问题，而 `createWebHashHistory` 不需要

# 路由传递参数

1. 在路由配置中指定参数key

```
//router/index.js
const routes = [
{
  path: "/news/details/:name", //这里后面的冒号就是key，代表要传参数
  component: () => import("../views/newsDetails.vue"),
},
]
```

## 2. 在跳转过程中携带参数

```
<li><router-link to="/news/details/百度">百度新闻</router-link></li>
<li><router-link to="/news/details/网易">网易新闻</router-link></li>
<li><router-link to="/news/details/头条">头条新闻</router-link></li>
```

## 3. 在详情页面读取路由携带的参数

```
<p>{{ $route.params.key }}</p> //这里的key指的是你上面起的key的名字
```

# 嵌套路由配置

操作	优惠券发放ID	优惠券名称	发放时间	发放场景	发放数量	领取数量	有效时间	状态	发放状态
编辑	停用	详情	2020-02-17 21:13:4	活动领取	100	0	永久有效	启用	发放中
编辑	启用	详情	2020-01-15 15:43:1	活动领取	1000	0	永久有效	停用	发放中
详情			2020-01-06 18:54:1	课程打开	2	2	永久有效	停用	已结束
编辑	启用	详情	2020-01-06 18:39:4	活动领取	100	2	永久有效	停用	发放中
编辑	启用	详情	2020-01-06 18:39:4	活动领取	100	2	永久有效	停用	发放中
编辑	启用	详情	2020-01-06 18:39:4	活动领取	100	4	永久有效	停用	发放中
编辑	停用	详情	2020-01-06 18:25:6	知药学Banner图	4	2	永久有效	启用	发放中
详情			2020-01-06 00:00:1	活动领取	11	1	永久有效	停用	已结束
详情			2020-01-05 00:00:1	消息推送	0	0	2020-01-05 - 2020-01-06	停用	已结束
详情			2020-01-03 00:00:1	消息推送	0	4	2020-01-03 - 2020-01-04	停用	已结束
编辑	停用	详情	2020-01-03 21:59:4	消息推送	10	4	2020-01-03 - 2020-01-04	启用	发放中
详情			2020-01-03 21:52:4	活动领取	4	4	永久有效	停用	已结束
编辑	启用	详情	2020-01-03 21:49:4	知药学Banner图	14	3	永久有效	停用	发放中
编辑	停用	详情	2020-01-03 21:02:4	活动领取	18	7	永久有效	停用	发放中
详情			2020-01-03 21:03:4	活动领取	4	4	永久有效	停用	已结束

这是一个常见的后台管理系统，有两个导航：第一个是左导航，在一个导航后面还有其他的子导航，适用于路由嵌套的效果进行实现。

## || 配置二级路由

这里展示配置 `about` 页面的二级路由，使用 `children:[]` 来进行配置，注意二级导航的路径不要加斜杠

```
//route/index.js
{
  path: "/about",
  component: () => import("../views/AboutView.vue"),
  children: [
    {
      //二级导航的路径不要加斜杠
      path: "us",
      component: () => import("../views/AboutSub/AboutUS.vue"),
    },
    {
      path: "info",
      component: () => import("../views/AboutSub/AboutInfo.vue"),
    },
  ],
}
```

在相应的页面加入相应标签,注意这里要写全路径

即：children的路径要配合父级来写

```
<template>
  <router-link to="/about/us">关于我们</router-link> |
  <router-link to="/about/info">关于信息</router-link>
  <router-view></router-view>
</template>
```

## || 配置重定向

重定向配置 `redirect` 可以默认配置进入某个页面

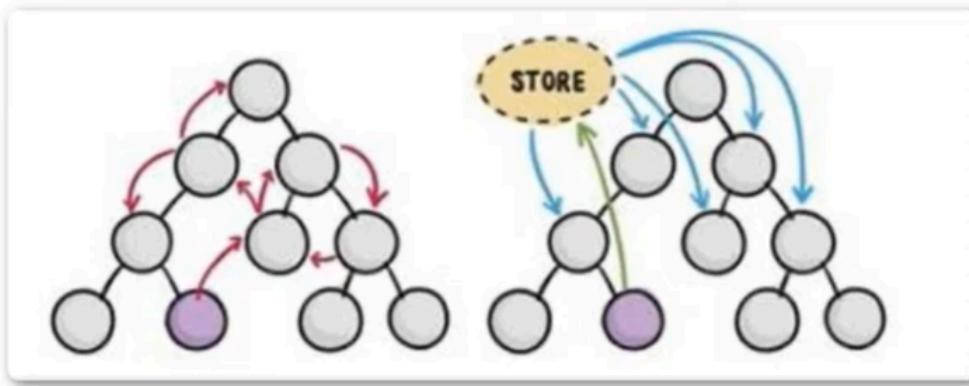
```
//route/index.js
{
  path: "/about",
  redirect: "/about/us",
  component: () => import("../views/AboutView.vue"),
  children: [
    {
      //二级导航的路径不要加斜杠
    },
  ],
}
```

```
        path: "us",
        component: () => import("../views/AboutSub/AboutUS.vue"),
    },
{
    path: "info",
    component: () => import("../views/AboutSub/AboutInfo.vue"),
},
]
},
```

## Vuex 状态管理

是一个状态管理模式库，集中式存储管理应用的所有组件的状态，并以相应的规则以一种可预测的方式发生变化。

简单来说：就是可以理解成更方便地管理组件之间的数据交互，任何组件都可以按照指定的方式进行读取和改变数据。



可以简化组件之间的一层一层的传递关系，而是把数据集中管理起来

### I | Vuex 的安装与使用

#### 1. 安装 vuex



```
npm install --save vuex
```

#### 2. \src 文件夹下新建 \store 目录，其中新建 index.js，其中的 state 用来存放数据



```
//index.js
import {createStore} from "vuex";

//vuex的核心作用就是帮我们管理组件之间的状态的
export default createStore({
    //所有的数据都放在这里
    state:{
```

```
        counter: 0,  
    }  
})
```

### 3. 在 `main.js` 中引用



```
//main.js  
import store from './store'  
createApp(App).use(store).mount('#app')
```

### 4. 在组件中读取数据



```
<p>{{$store.state.couter}}</p>
```



```
<template>  
  <p>{{counter}}</p>  
</template>  
  
<script>  
  import {mapState} from "Vuex"  
  
  computed:{  
    ...mapState(["counter"])  
  }  
</script>
```

## I `Vuex` 的核心概念

包括 `State`, `getter`, `Mutation`, `Action`

I `Getter` ::

`Getter` 的作用是对Vuex中的数据进行过滤



```
export default createStore({  
  //所有的数据都放在这里  
  state:{  
    counter: 0,  
  },  
  getters:{  
    getCounter(state){
```

```
        return state.counter > 0 ? state.counter : "counter 数据异常";
    }
}

})
```

使用方式如下：

```
<template>
  <div>{{$store.getters.getCounter}}</div>
  <div>{{getCounter}}</div>
</template>

<script>
export default{
  import {mapGetters} from "vuex";
  computed:{
    ...mapGetters(["getCounter"]);
  }
}
</script>
```

### Mutation

..

更改 `Vuex` 的 `Store` 中的状态的唯一方法是提交 `Mutation`

`Vuex` 中的 `Mutation` 非常类似于事件：每个 `mutation` 都有一个字符串的事件类型和一个回调函数。这个回调函数就是我们实际进行状态更改的地方，他会接受 `State` 作为第一个参数

1. 创建一个 `Mutation`：

```
<script>
export default createStore({
  //所有的数据都放在这里
  state:{
    counter:0,
  },
  getters:{
    getCounter(state){
      return state.counter > 0 ? state.counter : "counter 数据异常";
    }
  },
  mutations:{
    addCounter(state , num){
      state.counter += num;
    }
  }
})
```

```
        state.counter++;
    }
}

</script>
```

## 2. 通过规定的方法使用 `Mutation` 中的方法

```
<template>
  <div @click = "test">点击调用方法</div>
</template>
<script>
  methods:{
    test(){
      //传入函数名来调用函数
      this.$store.commit("addCounter");
    }
  }
</script>
```

也可以传参数：

```
mutations:{
  addCounter(state , num){
    state.counter += num;
  }
}
```

```
methods:{
  test(){
    //传入函数名来调用函数
    this.$store.commit("addCounter",15); //参数放在这个函数名的后面
  }
}
```

也可以引用 `MapMutation` 来实现方便的调用

注意 `MapMutation` 写在 `methods` 中

```
import {mapMutations} from "vuex";
```

```
export default:{  
  methods:{  
    ... mapMutations(["addCounter"]);  
    this.addCounter(20); //直接作为函数传递  
  }  
}
```

### Action ..

Action 允许异步操作，而 Mutation 只允许同步操作。示例：



```
actions:{  
  asyncAddCounter({commit}){  
    axios.get("url")  
    .then((res) => {  
      commit("addCounter", res.data[0]); //假设获取的数据是一个数组  
    })  
  }  
}
```

调用方式：



```
methods:{  
  addAsyncClickHandle(){  
    this.$store.dispatch("asyncAddCounter");  
  }  
}
```

或者采用这样的方式：



```
import {mapActions} from "vuex"  
  
methods:{  
  ... mapActions(["asyncAddCounter"]),  
  addAsyncClickHandle(){  
    this.asyncAddCounter();  
  }  
}
```

# Vue3 的新特性

## 组合式API

### 组合式API的基本结构 ::

组合式API实际上就相当于把 `data(){} , methods:{}` 等内容可以统一写到一个 `setup()` 中去：

```
setup(ctx){  
    //setup中没有this，可以用提供的ctx参数获取当前实例对象  
}
```

### 响应式数据的创建 ::

组合式API采用 `ref` 和 `reactive` 来创建响应式数据

```
setup(){  
    const message = ref("I'm Message");  
    const myArray = reactive({  
        list: ["a", "b", "c"];  
    })  
  
    return {  
        message, myArray  
    }  
}
```

注意事项：

1. 使用 `reactive` 接收一个对象类型的数据，例如上面的 `myArray`
2. 所有创建的数据如果要在外面访问那么你就需要 `return` 他们
3. 例如使用上述的 `myArray`，那么就需要使用 `myArray.list`
4. 如果想要修改数据的值，那么就需要使用 `var.value` 进行修改

```
setup(){  
    const message = ref("I'm Message");  
  
    message.value = "changed Message";  
  
    const myArray = reactive({  
        list: ["a", "b", "c"];  
    })  
}
```

```
    return {
      message, myArray
    }
}
```

## 方法的创建 ::

可以直接在 `setup()` 中正常定义函数并且 `return` 出去以供使用

```
setup(){
  const message = ref("I'm message");
  const changeMsg = (msg) => {
    message.value = msg;
  }

  return {
    message, changeMsg
  }
}
```

## Props 的使用 ::

`Props` 仍然需要先创建属性，然后才能使用 `setup()` 中的 `props` 参数来进行获取和处理

```
props: {
  message: String,
}

setup(props){
  const msg = props.message;
  return {
    msg
  }
}
```

## 在 `setup` 中使用生命周期函数 ::

现在需要使用图中的第二列的生命周期函数了，而且需要先引入

Options API	Hook inside setup
beforeCreate	Not needed*
created	Not needed*
beforeMount	onBeforeMount
mounted	onMounted
beforeUpdate	onBeforeUpdate
updated	onUpdated
beforeUnmount	onBeforeUnmount
unmounted	onUnmounted



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

setup() {
    //给生命周期函数中传入一个函数
    onMounted(() => {
        console.log("is mounted1!");
    })

    //现在可以存在多个
    onMounted(() => {
        console.log("is mounted2!");
    })
}
```

## | `Provide` 和 `inject`

用作组件之间的数据传递

注意事项：

1. `provide()` 和 `inject()` 可以实现嵌套组件之间的数据传递
2. 这两个函数只能在 `setup()` 函数中使用
3. 父组件中使用 `provide()` 向下传递参数
4. 子组件中采用 `inject()` 获取上层传递过来的数据
5. 只能父组件到子组件，不能反向传递



```
//父组件
```

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

setup() {
```

```
provide("key" , "message");
//key是这个消息的名字, message是这个消息的内容
}
```

```
//子组件
import {inject} from "vue";

setup(){
  const message = provide("key");//这样获取到数据
  return {
    message,
  }
}
```

## Element UI

### 安装

```
npm install element-plus --save
```

### 引用

#### 完整引入 ..

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

//引入js和index.css, 然后通过use的方式来加载它
const app = createApp(App).use(ElementPlus).mount("#app");
```

#### 按需导入 ..

1. 在main.js中不做任何改动和引入
2. 安装两款插件

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

### 3. 修改配置文件

插入以下代码在 `vue.config.js` 中

```
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()],
      }),
    ],
  },
})
```

## 加载字体图标

### 安装字体图标

```
npm install @element-plus/icons-vue
```

### 全局注册字体图标

在项目根目录下创建一个 `plugins` 文件夹，创建 `icons.js` 文件

```
//icons.js
import * as components from "@element-plus/icons-vue";
export default{
  install:(app) => {
    for (const key in components){
      const componentConfig = components[key];
      app.component(componentConfig.name,componentConfig);
    }
  },
}
```

### 在 `main.js` 中引入 `icons.js` 文件



```
//main.js
import elementIcon from "./plugins/icons";
app.use(elementIcon)
```