

02-脚手架+指令

一、脚手架

1. Vue代码开发方式

1.1 思考

相比直接 script 引入 vue 源码, 有没有更好的方式编写vue代码呢?

1.2 两种开发方式对比

1.2.1 传统开发模式

基于html文件开发Vue, 类似jQuery的使用, `<script src="vue.js"></script>`

✓ VUE-TRADITION-WAY

```
# index.css
<> index.html
JS index.js
```

- 优点: 简单、上手快
- 缺点: 功能单一、开发体验差

1.2.2 工程化开发模式

在 构建工具(Vite/Webpack)环境下开发Vue, 这是最推荐的、也是企业采用的方式

✓ VUE-ENGINEERING-WAY

```
> node_modules
> public
✓ src
  > assets
  > components
  ▼ App.vue
  JS main.js
  # style.css
  .gitignore
  <> index.html
  {} package.json
  ⓘ README.md
  JS vite.config.js
  👤 yarn.lock
```

- 优点:

- a. 功能全面
- b. 开发体验好

...

- 缺点：

- a. 目录结构复杂
- b. 理解难度提升

1.3 总结

1. Vue代码有哪两种开发方式？

答：1. 传统开发方式，直接用script引入vue.js文件

2. 在工程化环境下，借助Vite对Vue代码进行打包 (推荐)

2. 工程化环境下开发Vue代码，相比传统方式，有哪些优势？

答：1. 功能全面

2. 开发体验好

2. 准备工程化环境

2.1 安装工具

2.1.1 安装Nodejs

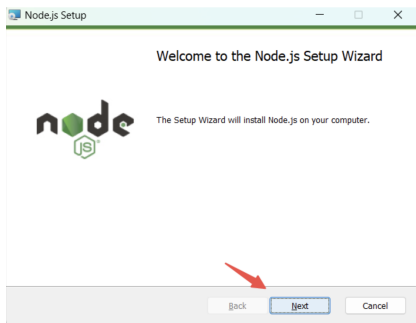
注意：安装18.3或更高版本的 Nodejs

官网: <https://nodejs.org/en/>

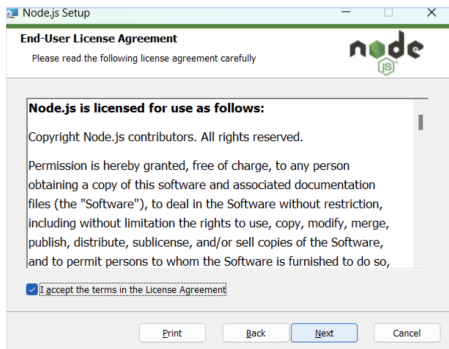
- 点击下载长期版



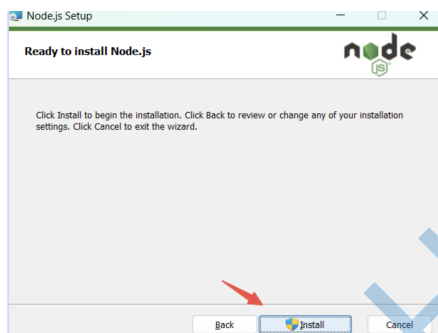
- 双击安装程序、并点击下一步



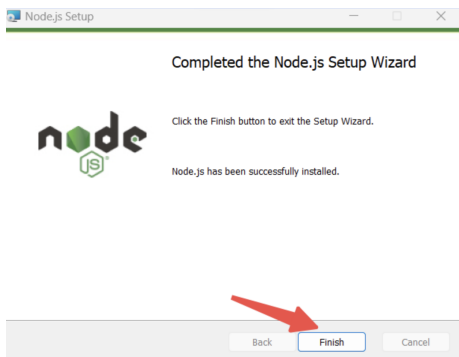
- 选择接受协议



- 点击下一步
- 点击install进行安装



- 安装完成点击finish



- 打开命令行窗口

```
1 node -v
2 npm -v
```

- npm换源

```
1 // 查看 npm 源
2 npm config get registry
3 // 默认是指向 https://registry.npmjs.org/, 也就是官方源
4
5 // 为了提高npm下载速度, 可以给npm换源
6 // 国内源有很多, 我这里用淘宝源吧。毕竟是大公司, 会比较稳定
7 npm config set registry https://registry.npmmirror.com
8
9 // 再一次查看 npm 源
10 npm config get registry
```

2.1.2 安装yarn或pnpm

yarn和pnpm、还有npm三者的功能类似, 都是包管理工具, 用来下载或删除模块包, 性能上yarn和pnpm优于npm

```
1 # windows系统
2 npm install yarn -g
3
4 npm install pnpm -g
5
6 -----
7
8 # mac系统
9 sudo npm install yarn -g
10
11 sudo npm install pnpm -g
```

- 检测是否安装成功

```
1 yarn -v
2
3 pnpm -v
```

2.2 总结

1. 准备工程化环境要安装哪些工具?

答：node / yarn | pnpm

3. 创建Vue工程化项目

3.1 脚手架

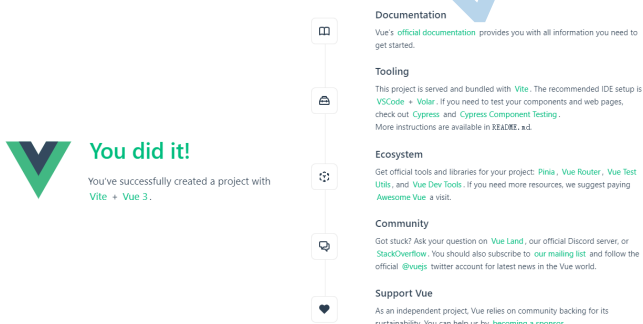
一个保证各项工作顺利开展的平台。好处就是拿来就用，零配置，基于它我们来学习Vue



3.2 创建步骤

1. 选定一个存放位置，比如选择桌面，根据自己情况，选择D盘或E盘等
2. 执行命令 `npm create vue@latest`，会安装并执行 `create-vue`，它是Vue 官方的项目脚手架工具
3. 进入项目根目录：`cd 项目名称`
4. 安装vue等模块依赖：`npm i`
5. 启动项目：`npm run dev`，会开启一个本地服务器
6. 浏览器网址栏输入：<http://localhost:5173>

最后呈现的界面效果如下，说明项目创建并启动成功了



3.3 总结

1. Vue脚手架是什么？有什么好处？

答：一个写Vue代码的环境(基础), 拿来就用, 零配置

2. 如何创建一个脚手架项目？

答: `npm create vue@latest`

3. 如何启动项目?

答: 1、`npm dev` , 启动一个本地服务器

2、浏览器输入 <http://localhost:5173>

4. 认识脚手架目录及文件

4.1 目录和文件解读



4.2 总结

1. 脚手架项目中，几个主要的文件及作用?

答: 1、`node_modules` —— 第三方模块包

2、`package.json` —— 项目管理文件

3、`src/main.js` -- 整个项目打包的入口

4、`src/App.vue` -- Vue代码的入口(根组件)

5、`index.html` -- 项目入口网页

2. 我们今后Vue代码写哪个目录下?

答: `src` 目录, `src`下的所有代码会被 `vite` 打包 成 `css/js/img`,

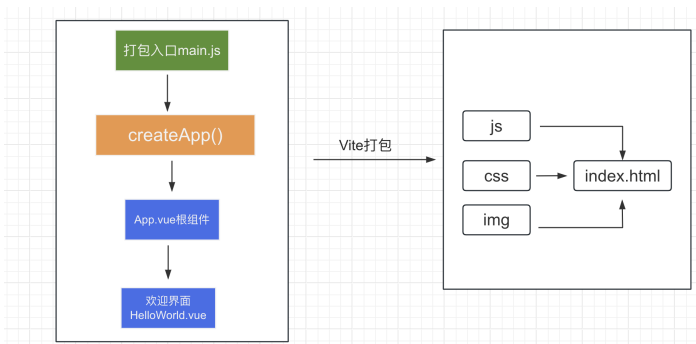
然后 交给`index.html` , 最终通过浏览器呈现在用户眼前

5. 分析3个入口文件的关系

5.1 目标

要知道3个入口文件的关系，以及我们编写的源代码是如何通过浏览器展示出来的

5.2 图解



5.3 总结

1. main.js、App.vue、index.html 三个文件的作用？

答：1、main.js — 项目打包的入口 — 创建应用

2、App.vue — Vue代码的入口(根组件)

3、index.html — 项目的入口网页

2. main.js、App.vue、index.html 三者的关系是什么？

答：1、App.vue(Vue入口) => main.js(项目打包入口) => index.html(浏览器入口)

2、main.js 是 Vue 代码通向网页代码的桥梁，非常关键

6. Vue单文件

6.1 思考

代码写一起、担心class类名、js变量名 重名冲突？Vue中如何避免呢？

6.2 vue单文件介绍

1. vue推荐采用 .vue 的文件来开发项目

2. 一个vue文件通常有3部分组成，script(JS) + template(HTML) + style(CSS)

3. 一个vue文件是 独立的模块，作用域互不影响

4. style配合 scoped 属性，保证样式只针对当前 template 内的标签生效

5. vue文件会被 vite打包成js、css 等，最终交给index.html 通过浏览器呈现效果

6.3 总结

1. vue文件的组成是什么？

答：三部分组成 = script(JS)+template(HTML)+style(CSS)

2. vue单文件的好处？

答：独立的作用域, 不用担心 JS 变量名、CSS 选择器名称冲突

7. 清理目录结构

7.1 目标

为了便于后续学习Vue代码，这里需要清楚默认不需要的目录结构

7.2 步骤

1. 删除 assets 文件夹
2. 删除 components 文件夹
3. 清除 App.vue 的内容
4. 清除 main.js 的内容

7.3 补充内容

App.vue

```
1 <script setup></script>
2 <template>
3   App根组件
4 </template>
5 <style></style>
```

main.js

```
1 import { createApp } from 'vue'
2 import App from './App.vue'
3 createApp(App).mount('#app')
```

8. setup简写+插值+响应式

8.1 完整写法

```
1 <script>
2   export default {
3     setup() {
4       // ...
5       const msg = 'Hello Vue3+Vite'
6
7       return {
```



```
8      msg
9    }
10  }
11  }
12 </script>
13 <template>
14   <h1>{{ msg }}</h1>
15 </template>
16 <style></style>
```

8.2 简化写法(推荐)

```
1 <script setup>
2   const msg = 'Hello Vue3+Vite'
3 </script>
4 <template>
5   <h1>{{ msg }}</h1>
6 </template>
7 <style></style>
```

8.3 练习代码示例

```
1 <script setup>
2   // 导入响应式函数
3   import { reactive, ref } from 'vue'
4
5   // 字符串
6   const msg = ref('Hello Vue3+Vite')
7
8   // 对象
9   const obj = reactive({
10     name: '小vue',
11     age: 9
12   })
13
14   // 函数
15   function fn() {
16     return 100
17   }
18 </script>
19
20 <template>
21   <!-- 1. 直接放变量 -->
```

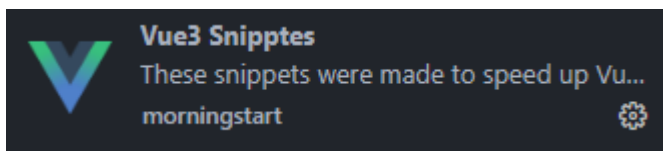
```

22     <h1>{{ msg }}</h1>
23     <!-- 2. 对象.属性 -->
24     <p>我叫 {{ obj.name }}，今年 {{ obj.age }} 岁</p>
25     <!-- 3. 三元表达式 -->
26     <p>{{ obj.age >= 18 ? '已成年' : '未成年' }}</p>
27     <!-- 3. 算数运算 -->
28     <p>来年我就 {{ obj.age + 1 }} 岁了</p>
29     <!-- 4. 函数的调用 -->
30     <p>fn的返回值是：{{ fn() }}</p>
31     <!-- 4. 方法的调用 -->
32     <p>{{ msg }} 中含有 {{ msg.split(' ').length }} 个单词</p>
33 </template>

```

8.4 安装VSCode插件

为了今后可以快速生成 vue 文件的三部分组成，建议安装VSCode的插件 Vue3 Snippets，然后在 vue 文件中输入 `vbase` 即可快速生成模版



8.5 总结

1. 推荐使用 `setup` 简化写法

```

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

```

二、指令

1. Vue中的常用指令

1.1 概念

指令（Directives）是 Vue 提供的带有 v- 前缀的特殊标签属性，用来增强标签的能力

1.2 作用

提高标签数据渲染的能力

1.3 分类

vue3 中的指令按照不同的用途可以分为如下 6 大类：

- 内容渲染指令 (v-html、v-text)
- 属性绑定指令 (v-bind)
- 事件绑定指令 (v-on)
- 条件渲染指令 (v-show、v-if、v-else、v-else-if)
- 列表渲染指令 (v-for)
- 双向绑定指令 (v-model)

指令是 vue 开发中最基础、最常用、最简单的知识点。

2. 内容渲染指令

2.1 目标

知道两个指令的作用和区别

2.2 作用

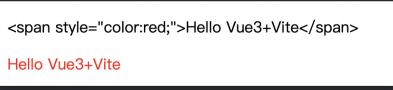
内容渲染指令用来辅助开发者渲染 DOM 元素的文本内容。

常用的内容渲染指令有如下2个：

- v-text (类似innerText)
 - 使用语法： `<p v-text="表达式"></p>`，意思是将表达式的值渲染到 p 标签中
 - 类似 innerText，使用该语法，会覆盖 p 标签原有内容
- v-html (类似innerHTML)
 - 使用语法： `<p v-html="表达式"></p>`，意思是将表达式的值渲染到 p 标签中
 - 类似 innerHTML，使用该语法，会覆盖 p 标签原有内容
 - 类似 innerHTML，使用该语法，能够将HTML标签的样式呈现出来。

```
<script setup>
import { ref } from 'vue' 556.4k (gzipped: 178.4k)
const str = ref('<span style="color:red;">Hello Vue3+Vite</span>')
</script>

<template>
  <div>
    <p v-text="str"></p>
    <p v-html="str"></p>
  </div>
</template>
```



2.3 代码示例

```
1 <script setup>
2   import { ref } from 'vue'
3   const str = ref('<span style="color:red;">Hello Vue3+Vite</span>')
```

```
4 </script>
5
6 <template>
7   <div>
8     <p v-text="str"></p>
9     <p v-html="str"></p>
10  </div>
11 </template>
12
13 <style scoped></style>
```

2.4 总结

1. v-text和v-html有什么作用？

答：把表达式的值展示在双标签中

2. 二者有何区别？

答：v-text不解析标签、v-html解析标签

3. 属性绑定指令

3.1 目标

掌握vue属性绑定的语法

3.2 作用

把 表达式的结果 与标签的 属性动态绑定

3.3 语法

- v-bind:属性名="表达式"
- 可简写成: 属性名="表达式"

```
<!-- v-bind: 标签属性的动态绑定 -->
<a v-bind:href="url">百度一下</a>
<!-- 简写 -->
<a :href="url">百度一下</a>
```

比如，有一个图片，它的 `src` 属性值，是一个图片地址，这个地址是ref定义的
则可以这样设置属性值：

- ``
- `` (v-bind可以省略)

3.4 代码示例

```

1 <script setup>
2   import { ref } from 'vue'
3   const url = ref('http://www.baidu.com')
4 </script>
5
6 <template>
7   <div>
8     <!-- v-bind: 标签属性动态绑定 -->
9     <a v-bind:href="url">百度一下</a>
10    <!-- 简写 -->
11    <a :href="url">百度一下</a>
12  </div>
13 </template>

```

3.5 总结

1. 如何给DOM标签的属性，绑定Vue表达式？

答：1、借助 `v-bind` 指令的简写, 语法 `:属性名="表达式"`

2、注意 属性名前的 冒号不能丢

4. 事件绑定指令

4.1 目标

掌握vue事件绑定语法

4.2 作用

给元素绑定事件

4.3 语法(3种写法)

以给 `button` 添加点击事件为例

- 1 `<button v-on:事件名="内联语句">按钮</button>`
- 2 `<button v-on:事件名="处理函数">按钮</button>`
- 3 `<button v-on:事件名="处理函数(实参列表)">按钮</button>`

```

<script setup>
import { ref } from 'vue'

const count = ref(0)

const increase = () => {
  count.value++
}

const add = (n) => {
  count.value += n
}
</script>

```

```

<div>
<p>{{ count }}</p>
<!-- 1. 内联语句 -->
<button v-on:click="count++">+1</button>
<!-- 2. 调用无参函数 -->
<button v-on:click="increase">+1</button>
<!-- 3. 调用有参函数 -->
<button v-on:click="add(3)">+3</button>
<button v-on:click="add(5)">+5</button>
<!-- 4. 简写 @click -->
<button @click="add(5)">+5</button>
</div>

```

- `v-on:` 可以简写为 `@`
- 函数需要在 `script` 下声明

4.4 代码示例

```
1 <script setup>
2   import { ref } from 'vue'
3
4   // 响应式数据 计数器
5   const count = ref(0)
6
7   // 无参函数
8   const increase = () => {
9     count.value++
10  }
11
12  // 有参函数
13  const add = (n) => {
14    count.value += n
15  }
16 </script>
17
18 <template>
19   <div>
20     <p>{{ count }}</p>
21     <!-- 1. 内联语句 -->
22     <button v-on:click="count++">+1</button>
23     <!-- 2. 调用无参函数 -->
24     <button v-on:click="increase">+1</button>
25     <!-- 3. 调用有参函数 -->
26     <button v-on:click="add(3)">+3</button>
27     <button v-on:click="add(5)">+5</button>
28     <!-- 4. 简写 @click -->
29     <button @click="add(5)">+5</button>
30   </div>
31 </template>
```

4.5 总结

1. 如何给DOM标签绑定事件?

答: 借助 `v-on` 指令的简写, 语法 `@事件名="函数"`

2. 如何给事件函数传参数?

答: `@事件名="函数(实参列表)"`

3. 函数在哪声明？

答：script 标签下直接声明

5. 条件渲染指令

5.1 目标

掌握条件判断指令，用来辅助开发者控制 DOM 的显示与隐藏。

5.2 区别

1. v-show

- 作用：控制元素显示隐藏
- 语法：v-show = "布尔表达式" 表达式值为 true 显示，false 隐藏
- 原理：切换 display:none 控制显示隐藏
- 场景：频繁切换显示隐藏的场景

2. v-if

- 作用：控制元素显示隐藏（条件渲染）
- 语法：v-if = "布尔表达式" 表达式值 true 显示，false 隐藏
- 原理：基于条件判断，创建 或 移除 元素
- 场景：要么显示，要么隐藏，不频繁切换的场景

3. v-else 和 v-else-if

- 作用：辅助v-if进行判断渲染
- 语法：v-else v-else-if="表达式"
- 需要紧接着v-if使用



5.3 代码示例

```
1 <script setup>
2   import { ref } from 'vue'
3   // 是否可见
4   const visible = ref(true)
```

```

5  // 是否登录
6  const isLogin = ref(true)
7  // 成绩
8  const mark = ref(100)
9  </script>
10
11 <template>
12   <!-- v-show -->
13   <div class="red" v-show="visible"></div>
14   <!-- v-if -->
15   <div class="green" v-if="visible"></div>
16
17   <hr>
18
19   <!-- 双分支的条件渲染 -->
20   <div v-if="isLogin">xxx, 欢迎回来</div>
21   <div v-else>你好, 请登录</div>
22
23   <hr>
24
25   <!--
26   多分支的条件渲染:
27   1. 90及其以上优秀
28   2. 70到90之间良好
29   3. 其他的差
30   -->
31   <div v-if="mark >= 90">优秀</div>
32   <div v-else-if="mark >= 70">良好</div>
33   <div v-else>差</div>
34 </template>
35
36 <style scoped>
37   .red, .green {
38     width: 200px;
39     height: 200px;
40   }
41   .red {
42     background: red;
43   }
44
45   .green {
46     background: green;
47   }
48 </style>

```

5.4 总结

1. vue中如何控制标签显示/隐藏?

答: 借助 `v-show` 和 `v-if` 指令关联布尔表达式, true显示; false隐藏

2. 二者区别是什么?

答: 1、v-show是通过控制css的display属性

2、v-if直接从DOM树上移除或创建; 并且v-if还可以配合v-else-if/v-else进行多分支条件渲染

6. 案例比特人的学习之旅

6.1 效果



6.2 需求

默认展示数组中的第一张图片, 点击上一页下一页来回切换数组中的图片

6.3 实现思路

1. 数组存储图片路径 ['url1','url2','url3', ...]
2. 准备下标index 去数组中取图片地址。
3. 通过v-bind给src绑定当前的图片地址
4. 点击上一页下一页只需要修改下标的值即可
5. 当展示第一张的时候, 上一页按钮应该隐藏。展示最后一张的时候, 下一页按钮应该隐藏

6.4 静态模版代码(可复制)

```
1 <script setup>
2   // 图片列表
3   const imgList = [
4     'https://cxk-1305128831.cos.ap-beijing.myqcloud.com/11-00.gif',
5     'https://cxk-1305128831.cos.ap-beijing.myqcloud.com/11-01.gif',
6     'https://cxk-1305128831.cos.ap-beijing.myqcloud.com/11-02.gif',
7     'https://cxk-1305128831.cos.ap-beijing.myqcloud.com/11-03.gif',
8     'https://cxk-1305128831.cos.ap-beijing.myqcloud.com/11-04.png',
9     'https://cxk-1305128831.cos.ap-beijing.myqcloud.com/11-05.png'
10  ]
11 </script>
```

```

12 <template>
13   <div>
14     <button>上一页</button>
15   </div>
16   
19   <div>
20     <button>下一页</button>
21   </div>
22 </template>
23 <style>
24   #app {
25     display: flex;
26     width: 500px;
27     height: 240px;
28   }
29
30   img {
31     width: 240px;
32     height: 240px;
33   }
34
35   #app div {
36     flex: 1;
37     display: flex;
38     justify-content: center;
39     align-items: center;
40   }
41 </style>

```

6.5 完整代码

```

1 <script setup>
2   import { ref } from 'vue'
3
4   // 图片列表
5   const imgList = [
6     'https://cxk-1305128831.cos.ap-beijing.myqcloud.com/11-00.gif',
7     'https://cxk-1305128831.cos.ap-beijing.myqcloud.com/11-01.gif',
8     'https://cxk-1305128831.cos.ap-beijing.myqcloud.com/11-02.gif',
9     'https://cxk-1305128831.cos.ap-beijing.myqcloud.com/11-03.gif',
10    'https://cxk-1305128831.cos.ap-beijing.myqcloud.com/11-04.png',
11    'https://cxk-1305128831.cos.ap-beijing.myqcloud.com/11-05.png'
12  ]

```

```
13 // 当前下标, 默认值 0
14 const i = ref(0)
15
16 </script>
17 <template>
18   <div>
19     <button
20       v-if="i >= 1"
21       @click="i--">
22       上一页
23     </button>
24   </div>
25   
28   <div>
29     <button
30       v-if="i < imgList.length - 1"
31       @click="i++">
32       下一页
33     </button>
34   </div>
35 </template>
36 <style>
37   #app {
38     display: flex;
39     width: 500px;
40     height: 240px;
41   }
42
43   img {
44     width: 240px;
45     height: 240px;
46   }
47
48   #app div {
49     flex: 1;
50     display: flex;
51     justify-content: center;
52     align-items: center;
53   }
54 </style>
```

7. 案例 - 可折叠面板

7.1 效果



7.2 静态模版代码(可复制)

```
1 <script setup></script>
2 <template>
3   <h3>可折叠面板</h3>
4   <div class="panel">
5     <div class="title">
6       <h4>自由与爱情</h4>
7       <span class="btn"> 收起 </span>
8     </div>
9     <div class="container">
10      <p>生命诚可贵,</p>
11      <p>爱情价更高.</p>
12      <p>若为自由故,</p>
13      <p>两者皆可抛.</p>
14    </div>
15  </div>
16 </template>
17
18 <style lang="scss">
19   body {
20     background-color: #ccc;
21   }
22
23   #app {
24     width: 400px;
25     margin: 20px auto;
26     padding: 1em 2em 2em;
27     border: 4px solid green;
28     border-radius: 1em;
29     box-shadow: 3px 3px 3px rgba(0, 0, 0, 0.5);
30     background-color: #fff;
31   }
32   #app h3 {
33     text-align: center;
34   }
35
36   .panel {
```

```

37     .title {
38         display: flex;
39         justify-content: space-between;
40         align-items: center;
41         padding: 0 1em;
42         border: 1px solid #ccc;
43     }
44     .title h4 {
45         line-height: 2;
46         margin: 0;
47     }
48     .container {
49         border: 1px solid #ccc;
50         padding: 0 1em;
51     }
52     .btn {
53         /* 鼠标改成手的形状 */
54         cursor: pointer;
55     }
56 }
57 </style>
58

```

7.3 完整代码

```

1 <script setup>
2   import { ref } from 'vue'
3
4   // 是否可见, 默认值 false, 表示不可见
5   const visible = ref(false)
6 </script>
7 <template>
8   <h3>可折叠面板</h3>
9   <div class="panel">
10     <div class="title">
11       <h4>自由与爱情</h4>
12       <span
13         class="btn"
14         @click="visible = !visible">
15         {{ visible ? '收起' : '展开' }}
16       </span>
17     </div>
18     <div
19       class="container"
20       v-show="visible">

```

```
21     <p>生命诚可贵,</p>
22     <p>爱情价更高。</p>
23     <p>若为自由故,</p>
24     <p>两者皆可抛。</p>
25 </div>
26 </div>
27 </template>
28
29 <style lang="scss">
30   body {
31     background-color: #ccc;
32   }
33
34   #app {
35     width: 400px;
36     margin: 20px auto;
37     padding: 1em 2em 2em;
38     border: 4px solid green;
39     border-radius: 1em;
40     box-shadow: 3px 3px 3px rgba(0, 0, 0, 0.5);
41     background-color: #fff;
42   }
43   #app h3 {
44     text-align: center;
45   }
46
47   .panel {
48     .title {
49       display: flex;
50       justify-content: space-between;
51       align-items: center;
52       border: 1px solid #ccc;
53       padding: 0 1em;
54     }
55     .title h4 {
56       line-height: 2;
57       margin: 0;
58     }
59     .container {
60       border: 1px solid #ccc;
61       padding: 0 1em;
62     }
63     .btn {
64       /* 鼠标改成手的形状 */
65       cursor: pointer;
66     }
67   }
```

```
68 </style>
```

```
69
```

8. 列表渲染指令

8.1 目标

基于数组进行 v-for 列表渲染

8.2 语法

v-for 指令需要使用 `(item, index) in 目标结构` 形式的特殊语法，其中：

- item: 数组中的每一项
- index: 每一项的索引，不需要可以省略
- 目标结构: 被遍历的 数组/对象/数字

```
1 <script setup>
2   import { ref } from 'vue'
3   // 数字数组
4   const nums = ref([11, 22, 33, 44])
5
6   // 商品列表
7   const goodsList = ref([
8     { id: 1, name: '篮球', price: 299 },
9     { id: 2, name: '足球', price: 99 },
10    { id: 3, name: '排球', price: 199 }
11  ])
12
13  // 准备对象
14  const obj = {
15    id: 10001,
16    name: 'bit',
17    age: 9
18  }
19 </script>
20 <template>
21   <div>
22     <ul>
23       <!-- 遍历数字数组 -->
24       <li v-for="(item, index) in nums">{{ item }} => {{ index }}</li>
25     </ul>
26     <div class="goods-list">
27       <!-- 遍历对象数组 -->
28     </div>
```

```

29     class="goods-item"
30     v-for="item in goodsList">
31     <p>id = {{ item.id }}</p>
32     <p>name = {{ item.name }}</p>
33     <p>price = {{ item.price }}</p>
34 </div>
35 <ul>
36     <!-- 遍历对象 -->
37     <li v-for="(value, key, index) in obj">
38         {{ value }} => {{ key }} => {{ index }}
39     </li>
40 </ul>
41
42 <ul>
43     <!-- 遍历数字 -->
44     <li v-for="(item, index) in 5">{{ item }} => {{ index }}</li>
45 </ul>
46 </div>
47 </div>
48 </template>
49
50 <style scoped></style>

```

8.3 总结

1. v-for如何循环生成列表？

答：1、想循环谁就把 v-for 添加给谁
2、语法: v-for="(item, index) in 目标数据"

2. v-for可以遍历的哪些目标数据？

答：数组 / 对象 / 数字

9. 案例比特人的书架

9.1 需求

- 1.根据左侧数据渲染出右侧列表（v-for）
- 2.点击删除按钮时，应该把当前行从列表中删除（获取当前行的index，利用splice删除）

比特人的书架

《红楼梦》	曹雪芹	删除
《西游记》	吴承恩	删除
《水浒传》	施耐庵	删除
《三国演义》	罗贯中	删除

9.2 静态模版代码(可复制)

```
1 <script setup>
2   // 图书列表
3   const bookList = [
4     { id: 1, name: '《红楼梦》', author: '曹雪芹' },
5     { id: 2, name: '《西游记》', author: '吴承恩' },
6     { id: 3, name: '《水浒传》', author: '施耐庵' },
7     { id: 4, name: '《三国演义》', author: '罗贯中' }
8   ]
9 </script>
10
11 <template>
12   <h3>比特人的书架</h3>
13   <ul>
14     <li>
15       <span>《红楼梦》</span>
16       <span>曹雪芹</span>
17       <button>删除</button>
18     </li>
19   </ul>
20 </template>
21
22 <style>
23   #app {
24     width: 400px;
25     margin: 100px auto;
26   }
27
28   ul li {
29     display: flex;
30     justify-content: space-around;
31     padding: 10px 0;
32     border-bottom: 1px solid #ccc;
33   }
34 </style>
```

9.3 完整代码

```
1 <!-- @format -->
2 <script setup>
3   import { ref } from 'vue'
4
5   // 图书列表
6   const bookList = ref([
7     { id: 1, name: '《红楼梦》', author: '曹雪芹' },
8     { id: 2, name: '《西游记》', author: '吴承恩' },
9     { id: 3, name: '《三国演义》', author: '罗贯中' },
10    { id: 4, name: '《水浒传》', author: '施耐庵' }
11  ])
12  // 删除
13  const onDel = (i) => {
14    // i: 当前点击的下标
15
16    // 删除前先确认
17    if (window.confirm('确定删除么?')) {
18      // 调用 splice 进行删除
19      bookList.value.splice(i, 1)
20    }
21  }
22 </script>
23
24 <template>
25   <h3>比特人的书架</h3>
26   <ul>
27     <li v-for="(item, index) in bookList">
28       <span>{{ item.name }}</span>
29       <span>{{ item.author }}</span>
30       <button @click="onDel(index)">删除</button>
31     </li>
32   </ul>
33 </template>
34
35 <style>
36   #app {
37     width: 400px;
38     margin: 100px auto;
39   }
40   ul {
41     list-style: none;
42   }
43   ul li {
44     display: flex;
```

```
45     justify-content: space-around;
46     padding: 10px 0;
47     border-bottom: 1px solid #ccc;
48   }
49 </style>
```

10. v-for中的key

10.1 目标

知道为什么要给v-for添加key

10.2 语法

:key="唯一值"

10.3 作用

给列表项添加的唯一标识, 便于Vue进行列表项的正确排序复用, 因为Vue 的默认行为会尝试原地修改元素 (就地复用)

10.4 代码示例

```
1 <!-- @format -->
2 <script setup>
3   import { ref } from 'vue'
4
5   // 图书列表
6   const bookList = ref([
7     { id: 1, name: '《红楼梦》', author: '曹雪芹' },
8     { id: 2, name: '《西游记》', author: '吴承恩' },
9     { id: 3, name: '《三国演义》', author: '罗贯中' },
10    { id: 4, name: '《水浒传》', author: '施耐庵' }
11  ])
12  // 删除
13  const onDel = (i) => {
14    // i: 当前点击的下标
15
16    // 删除前先确认
17    if (window.confirm('确定删除么?')) {
18      // 调用 splice 进行删除
19      bookList.value.splice(i, 1)
20    }
21  }
22 </script>
23
```

```

24 <template>
25   <h3>比特人的书架</h3>
26   <ul>
27     <!-- 无key -->
28     <li v-for="(item, index) in bookList">
29       <span>{{ item.name }}</span>
30       <span>{{ item.author }}</span>
31       <button @click="onDel(index)">删除</button>
32     </li>
33   </ul>
34
35   <ul>
36     <!-- 有key且为id -->
37     <li
38       v-for="(item, index) in bookList"
39       :key="item.id">
40       <span>{{ item.name }}</span>
41       <span>{{ item.author }}</span>
42       <button @click="onDel(index)">删除</button>
43     </li>
44   </ul>
45 </template>
46
47 <style>
48   #app {
49     width: 400px;
50     margin: 100px auto;
51   }
52   ul {
53     list-style: none;
54   }
55   ul li {
56     display: flex;
57     justify-content: space-around;
58     padding: 10px 0;
59     border-bottom: 1px solid #ccc;
60   }
61 </style>

```

10.5 注意

1. key 的类型只能是 数字 或 字符串
2. key 的值必须 唯一，不能重复
3. 推荐用 id 作为 key（因为id唯一），不推荐用 index 作为 key（会变化）

10.6 总结

1. 为什么要给v-for所在的元素添加key?

答: 最大限度的复用DOM、从而提高DOM的更新性能

2. key的类型?

答: 数字 / 字符串

3. key的选择?

答: 首选每次循环的 id、其次是下标

11. 双向绑定指令

11.1 介绍

所谓双向绑定就是:

1. 数据变了 -> 视图的变化
2. 视图变了 -> 数据的变化

11.2 作用

作用在 **表单元素** (input、select、radio、checkbox) 上, 实现数据双向绑定, 从而可以快速 **获取** 或 **设置** 表单元素的值

11.3 语法

```
v-model="响应式数据"
```

11.4 需求

使用双向绑定实现以下需求:

1. 点击登录按钮获取表单中的内容
2. 点击重置按钮清空表单中的内容



11.5 完整代码

```
1 <script setup>
2 // 导入 reactive 响应式函数
```

```

3   import { reactive } from 'vue'
4
5   // 登录表单对象
6   const loginForm = reactive({
7     username: '',
8     password: ''
9   })
10
11 </script>
12 <template>
13   <div class="login-box">
14     账户: <input type="text" v-model="loginForm.username"/><br /><br />
15     密码: <input type="password" v-model="loginForm.password"/><br /><br />
16     <button>登 录</button>
17     <button>重 置</button>
18   </div>
19 </template>

```

12. 案例比特人的记事本

12.1 静态模版代码(可复制)

新建 styles/index.css

```

1  html,
2  body {
3    margin: 0;
4    padding: 0;
5  }
6  body {
7    background: #fff;
8  }
9  button {
10   margin: 0;
11   padding: 0;
12   border: 0;
13   background: none;
14   font-size: 100%;
15   vertical-align: baseline;
16   font-family: inherit;
17   font-weight: inherit;
18   color: inherit;
19   -webkit-appearance: none;
20   appearance: none;
21   -webkit-font-smoothing: antialiased;

```

```
22 -moz-osx-font-smoothing: grayscale;
23 }
24
25 body {
26   font: 14px 'Helvetica Neue', Helvetica, Arial, sans-serif;
27   line-height: 1.4em;
28   background: #f5f5f5;
29   color: #4d4d4d;
30   min-width: 230px;
31   max-width: 550px;
32   margin: 0 auto;
33   -webkit-font-smoothing: antialiased;
34   -moz-osx-font-smoothing: grayscale;
35   font-weight: 300;
36 }
37
38 :focus {
39   outline: 0;
40 }
41
42 .hidden {
43   display: none;
44 }
45
46 #app {
47   background: #fff;
48   margin: 180px 0 40px 0;
49   padding: 15px;
50   position: relative;
51   box-shadow: 0 2px 4px 0 rgba(0, 0, 0, 0.2), 0 25px 50px 0 rgba(0, 0, 0, 0.1);
52 }
53 #app .header input {
54   border: 2px solid rgba(175, 47, 47, 0.8);
55   border-radius: 10px;
56 }
57 #app .add {
58   position: absolute;
59   right: 15px;
60   top: 15px;
61   height: 68px;
62   width: 140px;
63   text-align: center;
64   background-color: rgba(175, 47, 47, 0.8);
65   color: #fff;
66   cursor: pointer;
67   font-size: 18px;
68   border-radius: 0 10px 10px 0;
```

```
69 }
70
71 #app input::-webkit-input-placeholder {
72   font-style: italic;
73   font-weight: 300;
74   color: #e6e6e6;
75 }
76
77 #app input::-moz-placeholder {
78   font-style: italic;
79   font-weight: 300;
80   color: #e6e6e6;
81 }
82
83 #app input::input-placeholder {
84   font-style: italic;
85   font-weight: 300;
86   color: gray;
87 }
88
89 #app h1 {
90   position: absolute;
91   top: -120px;
92   width: 100%;
93   left: 50%;
94   transform: translateX(-50%);
95   font-size: 60px;
96   font-weight: 100;
97   text-align: center;
98   color: rgba(175, 47, 47, 0.8);
99   -webkit-text-rendering: optimizeLegibility;
100   -moz-text-rendering: optimizeLegibility;
101   text-rendering: optimizeLegibility;
102 }
103
104 .new-todo,
105 .edit {
106   position: relative;
107   margin: 0;
108   width: 100%;
109   font-size: 24px;
110   font-family: inherit;
111   font-weight: inherit;
112   line-height: 1.4em;
113   border: 0;
114   color: inherit;
115   padding: 6px;
```



```
116 box-shadow: inset 0 -1px 5px 0 rgba(0, 0, 0, 0.2);
117 box-sizing: border-box;
118 -webkit-font-smoothing: antialiased;
119 -moz-osx-font-smoothing: grayscale;
120 }
121
122 .new-todo {
123   padding: 16px;
124   border: none;
125   background: rgba(0, 0, 0, 0.003);
126   box-shadow: inset 0 -2px 1px rgba(0, 0, 0, 0.03);
127 }
128
129 .main {
130   position: relative;
131   z-index: 2;
132 }
133
134 .todo-list {
135   margin: 0;
136   padding: 0;
137   list-style: none;
138   overflow: hidden;
139 }
140
141 .todo-list li {
142   position: relative;
143   font-size: 24px;
144   height: 60px;
145   box-sizing: border-box;
146   border-bottom: 1px solid #e6e6e6;
147 }
148
149 .todo-list li:last-child {
150   border-bottom: none;
151 }
152
153 .todo-list .view .index {
154   position: absolute;
155   color: gray;
156   left: 10px;
157   top: 20px;
158   font-size: 22px;
159 }
160
161 .todo-list li .toggle {
162   text-align: center;
```

```

163 width: 40px;
164 /* auto, since non-WebKit browsers doesn't support input styling */
165 height: auto;
166 position: absolute;
167 top: 0;
168 bottom: 0;
169 margin: auto 0;
170 border: none; /* Mobile Safari */
171 -webkit-appearance: none;
172 appearance: none;
173 }
174
175 .todo-list li .toggle {
176   opacity: 0;
177 }
178
179 .todo-list li .toggle + label {
180   /*
181    Firefox requires `#` to be escaped -
182    https://bugzilla.mozilla.org/show_bug.cgi?id=922433
183    IE and Edge requires *everything* to be escaped to render, so we do that
184    instead of just the `#` - https://developer.microsoft.com/en-us/microsoft-
185    edge/platform/issues/7157459/
186    */
187   background-image:
188     url('data:image/svg+xml;utf8,%3Csvg%20xmlns%3D%22http%3A%2F%2Fwww.w3.org%2F2000%2Fsvg%2
189     2%20width%3D%2240%22%20height%3D%2240%22%20viewBox%3D%22-10%20-
190     18%20100%20135%22%3E%3Ccircle%20cx%3D%2250%22%20cy%3D%2250%22%20r%3D%2250%22%20
191     fill%3D%22none%22%20stroke%3D%22%23ededed%22%20stroke-
192     width%3D%223%22%2F%3E%3C%2Fsvg%3E%27%3E%3C/svg%3E');
193   background-repeat: no-repeat;
194   background-position: center left;
195 }
196
197 .todo-list li .toggle:checked + label {
198   background-image:
199     url('data:image/svg+xml;utf8,%3Csvg%20xmlns%3D%22http%3A%2F%2Fwww.w3.org%2F2000%2Fsvg%2
200     2%20width%3D%2240%22%20height%3D%2240%22%20viewBox%3D%22-10%20-
201     18%20100%20135%22%3E%3Ccircle%20cx%3D%2250%22%20cy%3D%2250%22%20r%3D%2250%22%20
202     fill%3D%22none%22%20stroke%3D%22%23bddad5%22%20stroke-
203     width%3D%223%22%2F%3E%3Cpath%20fill%3D%22%235dc2af%22%20d%3D%22M72%2025L42%2071%2
204     027%2056l-4%204%2020%2020%2034-52z%22%2F%3E%3C%2Fsvg%3E%27%3E%3C/svg%3E');
205 }
206
207 .todo-list li label {
208   word-break: break-all;
209   padding: 15px 15px 15px 60px;

```

```
196 display: block;
197 line-height: 1.2;
198 transition: color 0.4s;
199 }
200
201 .todo-list li.completed label {
202 color: #d9d9d9;
203 text-decoration: line-through;
204 }
205
206 .todo-list li .destroy {
207 display: none;
208 position: absolute;
209 top: 0;
210 right: 10px;
211 bottom: 0;
212 width: 40px;
213 height: 40px;
214 margin: auto 0;
215 font-size: 30px;
216 color: #cc9a9a;
217 margin-bottom: 11px;
218 transition: color 0.2s ease-out;
219 }
220
221 .todo-list li .destroy:hover {
222 color: #af5b5e;
223 }
224
225 .todo-list li .destroy:after {
226 content: 'x';
227 }
228
229 .todo-list li:hover .destroy {
230 display: block;
231 }
232
233 .todo-list li .edit {
234 display: none;
235 }
236
237 .todo-list li.editing:last-child {
238 margin-bottom: -1px;
239 }
240
241 .footer {
242 color: #777;
```

```
243 padding: 10px 15px;
244 height: 20px;
245 text-align: center;
246 border-top: 1px solid #e6e6e6;
247 }
248
249 .footer:before {
250   content: '';
251   position: absolute;
252   right: 0;
253   bottom: 0;
254   left: 0;
255   height: 50px;
256   overflow: hidden;
257   box-shadow: 0 1px 1px rgba(0, 0, 0, 0.2), 0 8px 0 -3px #f6f6f6,
258             0 9px 1px -3px rgba(0, 0, 0, 0.2), 0 16px 0 -6px #f6f6f6,
259             0 17px 2px -6px rgba(0, 0, 0, 0.2);
260 }
261
262 .todo-count {
263   float: left;
264   text-align: left;
265 }
266
267 .todo-count strong {
268   font-weight: 300;
269 }
270
271 .filters {
272   margin: 0;
273   padding: 0;
274   list-style: none;
275   position: absolute;
276   right: 0;
277   left: 0;
278 }
279
280 .filters li {
281   display: inline;
282 }
283
284 .filters li a {
285   color: inherit;
286   margin: 3px;
287   padding: 3px 7px;
288   text-decoration: none;
289   border: 1px solid transparent;
```

```
290     border-radius: 3px;
291 }
292
293 .filters li a:hover {
294     border-color: rgba(175, 47, 47, 0.1);
295 }
296
297 .filters li a.selected {
298     border-color: rgba(175, 47, 47, 0.2);
299 }
300
301 .clear-completed,
302 html .clear-completed:active {
303     float: right;
304     position: relative;
305     line-height: 20px;
306     text-decoration: none;
307     cursor: pointer;
308 }
309
310 .clear-completed:hover {
311     text-decoration: underline;
312 }
313
314 .info {
315     margin: 50px auto 0;
316     color: #bfbfbf;
317     font-size: 15px;
318     text-shadow: 0 1px 0 rgba(255, 255, 255, 0.5);
319     text-align: center;
320 }
321
322 .info p {
323     line-height: 1;
324 }
325
326 .info a {
327     color: inherit;
328     text-decoration: none;
329     font-weight: 400;
330 }
331
332 .info a:hover {
333     text-decoration: underline;
334 }
335
336 /*
```

```

337  Hack to remove background from Mobile Safari.
338  Can't use it globally since it destroys checkboxes in Firefox
339  */
340  @media screen and (-webkit-min-device-pixel-ratio: 0) {
341    .toggle-all,
342    .todo-list li .toggle {
343      background: none;
344    }
345
346    .todo-list li .toggle {
347      height: 40px;
348    }
349  }
350
351  @media (max-width: 430px) {
352    .footer {
353      height: 50px;
354    }
355
356    .filters {
357      bottom: 10px;
358    }
359  }

```

App.vue

```

1  <script setup>
2    // 导入 todo 样式
3    import './styles/index.css'
4
5    // 代办任务列表
6    const todoList = [
7      { id: 321, name: '吃饭', finished: false },
8      { id: 666, name: '睡觉', finished: true },
9      { id: 195, name: '打豆豆', finished: false }
10   ]
11
12  </script>
13
14  <template>
15    <section class="todoapp">
16      <header class="header">
17        <h1>比特人记事本</h1>
18        <input
19          placeholder="请输入任务"

```

```

20     class="new-todo" />
21     <button class="add">添加任务</button>
22 </header>
23 <section class="main">
24     <ul class="todo-list">
25         <li class="todo">
26             <div class="view">
27                 <span class="index">1.</span> <label>吃饭饭</label>
28                 <button class="destroy"></button>
29             </div>
30         </li>
31     </ul>
32 </section>
33 <footer class="footer">
34     <span class="todo-count">合 计: <strong> 0 </strong></span>
35     <button class="clear-completed">清空任务</button>
36 </footer>
37 </section>
38 </template>

```

12.2 功能需求

1. 列表渲染

```

1
2 <script setup>
3   import { ref } from 'vue'
4
5   // 代办任务列表
6   const todoList = ref([
7     { id: 321, name: '吃饭饭', finished: false },
8     { id: 666, name: '睡觉觉', finished: true },
9     { id: 195, name: '打豆豆', finished: false }
10  ])
11 </script>
12
13 <ul class="todo-list">
14   <li
15     class="todo"
16     v-for="(item, index) in todoList"
17     :key="item.id">
18     <div class="view">
19       <span class="index">{{ index + 1 }}.</span>
20       <label>{{ item.name }}</label>
21       <button class="destroy"></button>
22     </div>

```

```
23   </li>
24 </ul>
```

2. 添加功能

```
1   <script setup>
2
3   // 任务名称
4   const title = ref('')
5
6   // 添加
7   const onAdd = () => {
8     // 去除首尾空格
9     const name = title.value.trim()
10    // 非空校验
11    if (!name) return alert('名称不能为空')
12    // 添加至数组
13    todoList.value.push({
14      name,
15      id: Date.now(), // 时间戳
16      finished: false
17    })
18    // 清空输入框
19    title.value = ''
20  }
21 </script>
22
23
24 <header class="header">
25   <h1>比特人记事本</h1>
26   <input
27     v-model="title"
28     placeholder="请输入任务"
29     class="new-todo"
30     @keydown.enter="onAdd" />
31   <button
32     class="add"
33     @click="onAdd">
34     添加任务
35   </button>
36 </header>
```

3. 删除功能


```
1 <script setup>
2   // 删除
3   const onDel = (index) => {
4     if (window.confirm('确认删除么?')) {
5       todoList.value.splice(index, 1)
6     }
7   }
8 </script>
9
10
11 <button
12   class="destroy"
13   @click="onDel(index)"></button>
```

4. 底部统计和清空

```
1 <script setup>
2   // 清空
3   const onClear = () => {
4     todoList.value = []
5   }
6 </script>
7
8 <footer class="footer">
9   <span class="todo-count">
10     合 计:
11     <strong> {{ todoList.length }} </strong>
12   </span>
13   <button
14     class="clear-completed"
15     @click="onClear">
16     清空任务
17   </button>
18 </footer>
```