

1. 初始化项目

1. 在终端运行以下的命令，初始化 vite 项目：

```
1 npm init vite-app todos
```

2. 使用 vscode 打开项目，并安装依赖项：

```
1 npm install
```

3. 安装 less 语法相关的依赖项：

```
1 npm i less -D
```

2. 梳理项目结构

1. 重置 index.css 中的全局样式如下：

```
1 :root {  
2   font-size: 12px;  
3 }  
4  
5 body {  
6   padding: 8px;  
7 }
```

2. 重置 App.vue 组件的代码结构如下：

```
1 <template>  
2   <div>  
3     <h1>App 根组件</h1>  
4   </div>  
5 </template>  
6  
7 <script>  
8   export default {  
9     name: 'MyApp',
```

```

10   data() {
11     return {
12       // 任务列表的数据
13       todolist: [
14         { id: 1, task: '周一早晨9点开会', done: false },
15         { id: 2, task: '周一晚上8点聚餐', done: false },
16         { id: 3, task: '准备周三上午的演讲稿', done: true },
17       ],
18     }
19   },
20 }
21 </script>
22
23 <style lang="less" scoped></style>

```

3. 删除 `components` 目录下的 `HelloWorld.vue` 组件。
4. 在终端运行以下的命令，把项目运行起来：

```

1  npm run dev

```

3. 封装 todo-list 组件

3.1 创建并注册 TodoList 组件

1. 在 `src/components/todo-list/` 目录下新建 `TodoList.vue` 组件：

```

1  <template>
2    <div>TodoList 组件</div>
3  </template>
4
5  <script>
6    export default {
7      name: 'TodoList',
8    }
9  </script>
10
11 <style lang="less" scoped></style>

```

2. 在 `App.vue` 组件中导入并注册 `TodoList.vue` 组件：



```

1  // 导入 TodoList 组件
2  import TodoList from './components/todo-list/TodoList.vue'
3
4  export default {
5    name: 'MyApp',
6    // 注册私有组件
7    components: {
8      TodoList,
9    },
10 }

```

3. 在 `App.vue` 的 `template` 模板结构中使用注册的 `TodoList` 组件:



```

1  <template>
2    <div>
3      <h1>App 根组件</h1>
4
5      <hr />
6
7      <!-- 使用 todo-list 组件 -->
8      <todo-list></todo-list>
9    </div>
10 </template>

```

3.2 基于 bootstrap 渲染列表组件

1. 将 `资料` 目录下的 `css` 文件夹拷贝到 `src/assets/` 静态资源目录中。
2. 在 `main.js` 入口文件中, 导入 `src/assets/css/bootstrap.css` 样式表:



```

1  import { createApp } from 'vue'
2  import App from './App.vue'
3
4  // 导入 bootstrap.css 样式表
5  import './assets/css/bootstrap.css'
6  import './index.css'
7
8  createApp(App).mount('#app')

```

3. 根据 bootstrap 提供的**列表组** (<https://v4.bootcss.com/docs/components/list-group/#with-badges>) 和**复选框** (<https://v4.bootcss.com/docs/components/forms/#checkboxes-and-radios-1>) 渲染列表组件的基本效果:

```

1 <template>
2   <ul class="list-group">
3     <li class="list-group-item d-flex justify-content-between
4       align-items-center">
5       <!-- 复选框 -->
6       <div class="custom-control custom-checkbox">
7         <input type="checkbox" class="custom-control-input"
8           id="customCheck1" />
9         <label class="custom-control-label"
10          for="customCheck1">Check this custom checkbox</label>
11       </div>
12       <!-- badge 效果 -->
13       <span class="badge badge-success badge-pill">完成</span>
14       <span class="badge badge-warning badge-pill">未完成</span>
15     </li>
16   </ul>
17 </template>

```

3.3 为 TodoList 声明 props 属性

1. 为了接受外界传递过来的列表数据，需要在 `TodoList` 组件中声明如下的 props 属性：

```

1 export default {
2   name: 'TodoList',
3   props: {
4     // 列表数据
5     list: {
6       type: Array,
7       required: true,
8       default: [],
9     },
10  },
11 }

```

2. 在 `App` 组件中通过 `list` 属性，将数据传递到 `TodoList` 组件之中：

```

1 <todo-list :list="todolist"></todo-list>

```

3.4 渲染列表的 DOM 结构

1. 通过 v-for 指令，循环渲染列表的 DOM 结构：

```
1 <template>
2   <ul class="list-group">
3     <li class="list-group-item d-flex justify-content-between
4       align-items-center" v-for="item in list" :key="item.id">
5       <!-- 复选框 -->
6       <div class="custom-control custom-checkbox">
7         <input type="checkbox" class="custom-control-input"
8           :id="item.id" />
9         <label class="custom-control-label" :for="item.id">{{
10          item.task }}</label>
11       </div>
12       <!-- badge 效果 -->
13       <span class="badge badge-success badge-pill">完成</span>
14       <span class="badge badge-warning badge-pill">未完成</span>
15     </li>
16   </ul>
17 </template>
```

2. 通过 v-if 和 v-else 指令，按需渲染 badge 效果：

```
1 <!-- badge 效果 -->
2 <span class="badge badge-success badge-pill" v-if="item.done">完成
3   </span>
4 <span class="badge badge-warning badge-pill" v-else>未完成</span>
```

3. 通过 v-model 指令，双向绑定任务的完成状态：

```
1 <!-- 复选框 -->
2 <input type="checkbox" class="custom-control-input" :id="item.id"
3   v-model="item.done" />
4 <!-- 注意：App 父组件通过 props 传递过来的 list 是“引用类型”的数
5   据， -->
6 <!-- 这里 v-model 双向绑定的结果是：用户的操作修改的是 App 组件中数
7   据的状态 -->
```

4. 通过 v-bind 属性绑定，动态切换元素的 class 类名：

```
1 <label class="custom-control-label" :class="item.done ? 'delete' :  
  ''" :for="item.id">{{ item.task }}</label>
```

在 `TodoList` 组件中声明如下的样式，美化当前组件的 UI 结构：

```
1 // 为列表设置固定宽度  
2 .list-group {  
3   width: 400px;  
4 }  
5  
6 // 删除效果  
7 .delete {  
8   text-decoration: line-through;  
9 }
```

4. 封装 todo-input 组件

4.1 创建并注册 TodoInput 组件

1. 在 `src/components/todo-input/` 目录下新建 `TodoInput.vue` 组件：

```
1 <template>  
2   <div>TodoInput 组件</div>  
3 </template>  
4  
5 <script>  
6   export default {  
7     name: 'TodoInput',  
8   }  
9 </script>  
10  
11 <style lang="less" scoped></style>
```

2. 在 `App.vue` 组件中导入并注册 `TodoInput.vue` 组件：

```

1 // 导入 TodoList 组件
2 import TodoList from './components/todo-list/TodoList.vue'
3 // 导入 TodoInput 组件
4 import TodoInput from './components/todo-input/TodoInput.vue'
5
6 export default {
7   name: 'MyApp',
8   // 注册私有组件
9   components: {
10     TodoList,
11     TodoInput,
12   },
13 }

```

3. 在 `App.vue` 的 `template` 模板结构中使用注册的 `TodoInput` 组件:

```

1 <template>
2   <div>
3     <h1>App 根组件</h1>
4
5     <hr />
6
7     <!-- 使用 TodoInput 组件 -->
8     <todo-input></todo-input>
9     <!-- 使用 TodoList 组件 -->
10    <todo-list :list="todolist" class="mt-2"></todo-list>
11  </div>
12 </template>

```

4.2 基于 bootstrap 渲染组件结构

1. 根据 bootstrap 提供的 `inline-forms` (<https://v4.bootcss.com/docs/components/forms/#inline-forms>) 渲染 `TodoInput` 组件的基本结构。
2. 在 `TodoInput` 组件中渲染如下的 DOM 结构:

```

1 <template>
2   <!-- form 表单 -->
3   <form class="form-inline">
4     <div class="input-group mb-2 mr-sm-2">
5       <!-- 输入框的前缀 -->
6       <div class="input-group-prepend">

```

```

7       <div class="input-group-text">任务</div>
8     </div>
9     <!-- 文本输入框 -->
10    <input type="text" class="form-control" placeholder="请填写任
    务信息" style="width: 356px;" />
11  </div>
12
13    <!-- 添加按钮 -->
14    <button type="submit" class="btn btn-primary mb-2">添加新任务
    </button>
15  </form>
16 </template>

```

4.3 通过自定义事件向外传递数据

1. 在 `TodoList` 组件的 `data` 中声明如下的数据：

```

1  data() {
2    return {
3      // 新任务的名称
4      taskname: '',
5    }
6  }

```

2. 为 `input` 输入框进行 `v-model` 的双向数据绑定：

```

1  <input type="text" class="form-control" placeholder="请填写任务信
    息" style="width: 356px" v-model.trim="taskname" />

```

3. 监听 `form` 表单的 `submit` 事件，阻止默认提交行为并指定事件处理函数：

```

1  <form class="form-inline" @submit.prevent="onFormSubmit"></form>

```

4. 在 `methods` 中声明 `onFormSubmit` 事件处理函数如下：


```
1  methods: {
2    // 表单提交的事件处理函数
3    onFormSubmit() {
4      // 1. 判断任务名称是否为空
5      if (!this.taskname) return alert('任务名称不能为空! ')
6
7      // 2. 触发自定义的 add 事件，并向外界传递数据
8      // 3. 清空文本框
9    },
10 }
```

5. 声明自定义事件如下:

```
1  export default {
2    name: 'TodoInput',
3    // 声明自定义事件
4    emits: ['add'],
5  }
```

6. 进一步完善 `onFormSubmit` 事件处理函数如下:

```
1  methods: {
2    // 表单提交的事件处理函数
3    onFormSubmit() {
4      // 1. 判断任务名称是否为空
5      if (!this.taskname) return alert('任务名称不能为空! ')
6
7      // 2. 触发自定义的 add 事件，并向外界传递数据
8      this.$emit('add', this.taskname)
9      // 3. 清空文本框
10     this.taskname = ''
11   },
12 },
```

4.4 实现添加任务的功能

1. 在 `App.vue` 组件中监听 `TodoInput` 组件自定义的 `add` 事件:

```
1 <!-- 使用 TodoInput 组件 -->
2 <!-- 监听 TodoInput 的 add 自定义事件 -->
3 <todo-input @add="onAddNewTask"></todo-input>
```

2. 在 `App.vue` 组件的 `data` 中声明 `nextId` 来模拟 `id` 自增 +1 的操作:

```
1 data() {
2   return {
3     // 任务列表的数据
4     todolist: [
5       { id: 1, task: '周一早晨9点开会', done: false },
6       { id: 2, task: '周一晚上8点聚餐', done: false },
7       { id: 3, task: '准备周三上午的演讲稿', done: true },
8     ],
9     // 下一个可用的 Id 值
10    nextId: 4,
11  }
12 },
```

3. 在 `App.vue` 组件的 `methods` 中声明 `onAddNewTask` 事件处理函数如下:

```
1 methods: {
2   // TodoInput 组件 add 事件的处理函数
3   onAddNewTask(taskname) {
4     // 1. 向任务列表中新增任务信息
5     this.todolist.push({
6       id: this.nextId,
7       task: taskname,
8       done: false, // 完成状态默认为 false
9     })
10
11     // 2. 让 nextId 自增+1
12     this.nextId++
13   },
14 },
```

5. 封装 todo-button 组件

5.1 创建并注册 TodoButton 组件

1. 在 `src/components/todo-button/` 目录下新建 `TodoButton.vue` 组件:


```
1 <template>
2   <div>TodoButton 组件</div>
3 </template>
4
5 <script>
6   export default {
7     name: 'TodoButton',
8   }
9 </script>
10
11 <style lang="less" scoped></style>
12
```

2. 在 `App.vue` 组件中导入并注册 `TodoButton.vue` 组件:

```
1 // 导入 TodoList 组件
2 import TodoList from './components/todo-list/TodoList.vue'
3 // 导入 TodoInput 组件
4 import TodoInput from './components/todo-input/TodoInput.vue'
5 // 导入 TodoButton 组件
6 import TodoButton from './components/todo-button/TodoButton.vue'
7
8 export default {
9   name: 'MyApp',
10   // 注册私有组件
11   components: {
12     TodoList,
13     TodoInput,
14     TodoButton
15   },
16 }
```

5.2 基于 bootstrap 渲染组件结构

1. 根据 bootstrap 提供的 Button group (<https://v4.bootcss.com/docs/components/button-group/>) 渲染 Todobutton 组件的基本结构。
2. 在 `TodoButton` 组件中渲染如下的 DOM 结构:



```

1 <template>
2   <div class="button-container mt-3">
3     <div class="btn-group">
4       <button type="button" class="btn btn-primary">全部</button>
5       <button type="button" class="btn btn-secondary">已完成
6     </button>
7       <button type="button" class="btn btn-secondary">未完成
8     </button>
9   </div>
10  </div>
11 </template>

```

3. 并通过 `button-container` 类名美化组件的样式:



```

1 .button-container {
2   // 添加固定宽度
3   width: 400px;
4   // 文本居中效果
5   text-align: center;
6 }

```

5.3 通过 props 指定默认激活的按钮

1. 在 `ToggleButton` 组件中声明如下的 props, 用来指定默认激活的按钮的索引:



```

1 export default {
2   name: 'ToggleButton',
3   props: {
4     // 激活项的索引值
5     active: {
6       type: Number,
7       required: true,
8       // 默认激活索引值为 0 的按钮 (全部: 0, 已完成: 1, 未完成: 2)
9       default: 0,
10    },
11  },
12 }

```

2. 通过 **动态绑定** `class` **类名** 的方式控制按钮的激活状态:

```

1 <template>
2   <div class="button-container mt-3">
3     <div class="btn-group">
4       <button type="button" class="btn" :class="active === 0 ?
'btn-primary' : 'btn-secondary'">全部</button>
5       <button type="button" class="btn" :class="active === 1 ?
'btn-primary' : 'btn-secondary'">已完成</button>
6       <button type="button" class="btn" :class="active === 2 ?
'btn-primary' : 'btn-secondary'">未完成</button>
7     </div>
8   </div>
9 </template>

```

3. 在 `App` 组件中声明默认激活项的索引，并通过属性绑定的方式传递给 `TodoButton` 组件：

```

1 data() {
2   return {
3     // 激活的按钮的索引
4     activeBtnIndex: 0
5   }
6 }
7
8 <!-- 使用 TodoButton 组件 -->
9 <todo-button :active="activeBtnIndex"></todo-button>

```

5.4 通过 v-model 更新激活项的索引


需求分析：

父 -> 子 通过 props 传递了激活项的索引 (active)

子 -> 父 需要更新父组件中激活项的索引

这种场景下适合在组件上使用 `v-model` 指令，维护组件内外数据的同步。

1. 为 `TodoButton` 组件中的三个按钮分别绑定 `click` 事件处理函数如下：



```

1 <button type="button" class="btn" :class="active === 0 ? 'btn-
  primary' : 'btn-secondary'" @click="onBtnClick(0)">
2   全部
3 </button>
4 <button type="button" class="btn" :class="active === 1 ? 'btn-
  primary' : 'btn-secondary'" @click="onBtnClick(1)">
5   已完成
6 </button>
7 <button type="button" class="btn" :class="active === 2 ? 'btn-
  primary' : 'btn-secondary'" @click="onBtnClick(2)">
8   未完成
9 </button>

```

2. 在 `TodoButton` 组件中声明如下的自定义事件，用来更新父组件通过 `v-model` 指令传递过来的 `props` 数据：



```

1 export default {
2   name: 'TodoButton',
3   // 声明和 v-model 相关的自定义事件
4   emits: ['update:active'],
5   props: {
6     // 激活项的索引值
7     active: {
8       type: Number,
9       required: true,
10      default: 0,
11    },
12  },
13 }

```

3. 在 `TodoButton` 组件的 `methods` 节点中声明 `onBtnClick` 事件处理函数如下：



```

1 methods: {
2   // 按钮的点击事件处理函数
3   onBtnClick(index) {
4     // 1. 如果当前点击的按钮的索引值，等于 props 传递过来的索引值，
      则没必要触发 update:active 自定义事件
5     if (index === this.active) return
6     // 2. 通过 this.$emit() 方法触发自定义事件
7     this.$emit('update:active', index)
8   },
9 }

```

5.5 通过计算属性动态切换列表的数据

需求分析：

点击不同的按钮，切换显示不同的列表数据。此时可以根据当前激活按钮的索引，动态计算出要显示的列表数据并返回即可！

1. 在 `App` 根组件中声明如下的计算属性：

```
1  computed: {  
2    // 根据激活按钮的索引值，动态计算要展示的列表数据  
3    tasklist() {  
4      // 对“源数据”进行 switch...case 的匹配，并返回“计算之后的结果”  
5      switch (this.activeBtnIndex) {  
6        case 0: // 全部  
7          return this.todolist  
8        case 1: // 已完成  
9          return this.todolist.filter(x => x.done)  
10       case 2: // 未完成  
11         return this.todolist.filter(x => !x.done)  
12       }  
13     },  
14   },
```

2. 在 `App` 根组件的 DOM 结构中，将 `TodoList` 组件的 `:list="todolist"` 修改为：

```
1  <!-- 使用 TodoList 组件 -->  
2  <todo-list :list="tasklist" class="mt-2"></todo-list>
```