0. 实现步骤

① 搭建项目的基本结构

② 请求商品列表的数据

③ 封装 MyTable 组件

④ 实现删除功能

⑤ 实现添加标签的功能

1. 搭建项目基本结构

1.1 初始化项目

1. 在终端运行如下的命令，初始化 vite 项目：

```
npm init vite-app table-demo
```

2. cd 到项目根目录，安装依赖项：

```
npm install
```

3. 安装 less 依赖包：

```
npm i less -D
```

4. 使用 vscode 打开项目，并在 vscode 集成的终端下运行如下的命令，把项目运行起来：

```
npm run dev
```

1.2 梳理项目结构

1. 重置 App.vue 根组件的代码结构：

```vue
<template>
  <div>
    <h1>App 根组件</h1>
  </div>
</template>

<script>
export default {
  name: 'MyApp',
}
</script>

<style lang="less" scoped></style>
```

2. 删除 `components` 目录下的 `HelloWorld.vue` 组件
3. 重置 `index.css` 中的样式：

```css
:root {
  font-size: 12px;
}

body {
  padding: 8px;
}
```

4. 把**资料**目录下的 `css` 文件夹复制、粘贴到 `assets` 目录中，并在 `main.js` 入口文件中导入 `bootstrap.css`：

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

// 导入 bootstrap 样式表
import './assets/css/bootstrap.css'
import './index.css'

createApp(App).mount('#app')
```

2. 请求商品列表的数据

1. 运行如下的命令，安装 Ajax 的请求库：

```
npm install axios@0.21.0 -S
```

2. 在 `main.js` 入口模块中，导入并全局配置 axios：

```
// 1. 导入 axios
import axios from 'axios'

const app = createApp(App)

// 2. 将 axios 挂载到全局，今后，每个组件中，都可以直接通过 this.$http 代替 axios 发起 Ajax 请求
app.config.globalProperties.$http = axios
// 3. 配置请求的根路径
axios.defaults.baseURL = 'https://www.escook.cn'

app.mount('#app')
```

3. 在 `App.vue` 组件的 `data` 中声明 `goodslist` 商品列表数据：

```
data() {
    return {
        // 商品列表数据
        goodslist: []
    }
}
```

4. 在 `App.vue` 组件的 `methods` 中声明 `getGoodsList` 方法，用来从服务器请求商品列表的数据：

```
methods: {
    // 初始化商品列表的数据
    async getGoodsList() {
      // 发起 Ajax 请求
      const { data: res } = await this.$http.get('/api/goods')
      // 请求失败
      if (res.status !== 0) return console.log('获取商品列表失败！')
      // 请求成功
      this.goodslist = res.data
    }
}
```

5. 在 `App.vue` 组件中，声明 `created` 生命周期函数，并调用 `getGoodsList` 方法：

```
created() {
    this.getGoodsList()
}
```

3. 封装 MyTable 组件

3.0 MyTable 组件的封装要求

1. 用户通过名为 `data` 的 prop 属性，为 `MyTable.vue` 组件指定数据源
2. 在 `MyTable.vue` 组件中，预留名称为 `header` 的具名插槽
3. 在 `MyTable.vue` 组件中，预留名称为 `body` 的作用域插槽

3.1 创建并使用 MyTable 组件

1. 在 `components/my-table` 目录下新建 `MyTable.vue` 组件：

```
<template>
  <div>MyTable 组件</div>
</template>

<script>
export default {
  name: 'MyTable',
}
</script>

<style lang="less" scoped></style>
```

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

```vue
// 导入 MyTable 组件
import MyTable from './components/my-table/MyTable.vue'

export default {
  name: 'MyApp',
  // ... 省略其它代码

  // 注册 MyTable 组件
  components: {
    MyTable
  }
}
```

3. 在 `App.vue` 组件的 DOM 结构中使用 `MyTable.vue` 组件：

```vue
<template>
  <div>
    <h1>App 根组件</h1>

    <hr />

    <!-- 使用表格组件 -->
    <my-table></my-table>
  </div>
</template>
```


3.2 为表格声明 data 数据源

1. 在 `MyTable.vue` 组件的 props 节点中声明表格的 `data` 数据源：

```vue
export default {
  name: 'MyTable',
  props: {
    // 表格的数据源
    data: {
      type: Array,
      required: true,
      default: [],
    },
  },
}
```

2. 在 App.vue 组件中使用 MyTable.vue 组件时，通过**属性绑定的形式**为表格指定 data 数据源：

```
<!-- 使用表格组件 -->
<my-table :data="goodslist"></my-table>
```

3.3 封装 MyTable 组件的模板结构

1. 基于 bootstrap 提供的Tables(https://v4.bootcss.com/docs/content/tables/)，在 MyTable.vue 组件中渲染最基本的模板结构：

```
<template>
  <table class="table table-bordered table-striped">
    <!-- 表格的标题区域 -->
    <thead>
      <tr>
        <th>#</th>
        <th>商品名称</th>
        <th>价格</th>
        <th>标签</th>
        <th>操作</th>
      </tr>
    </thead>
    <!-- 表格的主体区域 -->
    <tbody></tbody>
  </table>
</template>
```

2. 为了提高组件的复用性，最好把表格的 标题区域 预留为 <slot> **具名插槽**，方便使用者自定义表格的标题：

```vue
<template>
  <table class="table table-bordered table-striped">
    <!-- 表格的标题区域 -->
    <thead>
      <tr>
        <!-- 命名插槽 -->
        <slot name="header"></slot>
      </tr>
    </thead>
    <!-- 表格的主体区域 -->
    <tbody></tbody>
  </table>
</template>
```

3. 在 `App.vue` 组件中，通过**具名插槽**的形式，为 `MyTable.vue` 组件指定标题名称：

```vue
<!-- 使用表格组件 -->
<my-table :data="goodslist">
  <!-- 表格的标题 -->
  <template v-slot:header>
    <th>#</th>
    <th>商品名称</th>
    <th>价格</th>
    <th>标签</th>
    <th>操作</th>
  </template>
</my-table>
```


3.4 预留名称为 body 的作用域插槽

1. 在 `MyTable.vue` 组件中，通过 `v-for` 指令循环渲染表格的数据行：

```vue
<template>
  <table class="table table-bordered table-striped">
    <thead>
      <tr>
        <slot name="header"></slot>
      </tr>
    </thead>
    <!-- 表格的主体区域 -->
    <tbody>
```

```
      <!-- 使用 v-for 指令，循环渲染表格的数据行 -->
      <tr v-for="(item, index) in data" :key="item.id"></tr>
    </tbody>
  </table>
</template>
```

2. 为了提高 `MyTable.vue` 组件的复用性，最好把表格数据行里面的 `td` 单元格预留为 `<slot>` 具名插槽。示例代码如下：

```
<template>
  <table class="table table-bordered table-striped">
    <thead>
      <tr>
        <slot name="header"></slot>
      </tr>
    </thead>
    <!-- 表格的主体区域 -->
    <tbody>
      <!-- 使用 v-for 指令，循环渲染表格的数据行 -->
      <tr v-for="(item, index) in data" :key="item.id">
          <!-- 为数据行的 td 预留的插槽 -->
          <slot name="body"></slot>
      </tr>
    </tbody>
  </table>
</template>
```

3. 为了让组件的使用者在提供 `body` 插槽的内容时，能够自定义内容的渲染方式，需要把 `body` **具名插槽**升级为 `作用域插槽`：

```
<template>
  <table class="table table-bordered table-striped">
    <thead>
      <tr>
        <slot name="header"></slot>
      </tr>
    </thead>
    <!-- 表格的主体区域 -->
    <tbody>
      <!-- 使用 v-for 指令，循环渲染表格的数据行 -->
      <tr v-for="(item, index) in data" :key="item.id">
          <!-- 为数据行的 td 预留的“作用域插槽” -->
          <slot name="body" :row="item" :index="index"></slot>
      </tr>
```

```
    </tbody>
  </table>
</template>
```

4. 在 `App.vue` 组件中，基于**作用域插槽**的方式渲染表格的数据：

```
<!-- 使用表格组件 -->
<my-table :data="goodslist">

  <!-- 表格的标题 -->
  <template v-slot:header>
    <th>#</th>
    <th>商品名称</th>
    <th>价格</th>
    <th>标签</th>
    <th>操作</th>
  </template>

  <!-- 表格每行的单元格 -->
  <template v-slot:body="{ row, index }">
    <td>{{ index + 1 }}</td>
    <td>{{ row.goods_name }}</td>
    <td>￥{{ row.goods_price }}</td>
    <td>{{ row.tags }}</td>
    <td>
      <button type="button" class="btn btn-danger btn-sm">删除</button>
    </td>
  </template>

</my-table>
```

4. 实现删除功能

1. 为删除按钮绑定 `click` 事件处理函数：

```
<td>
  <button type="button" class="btn btn-danger btn-sm" @click="onRemove(row.id)">删除</button>
</td>
```

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

```
methods: {
  // 根据 Id 删除商品信息
  onRemove(id) {
    this.goodslist = this.goodslist.filter(x => x.id !== id)
  },
}
```

5. 实现添加标签的功能

5.1 自定义渲染标签列

根据 bootstrap 提供的 Badge（https://v4.bootcss.com/docs/components/badge/#contextual-variations）效果，**循环渲染**商品的标签信息如下：

```
<td>
  <span class="badge badge-warning ml-2" v-for="item in row.tags" :key="item">{{tag}}</span>
</td>
```

5.2 实现 input 和 button 的按需展示

1. 使用 `v-if` 结合 `v-else` 指令，控制 input 和 button 的按需展示：

```
<td>
  <!-- 基于当前行的 inputVisible，来控制 input 和 button 的按需展示-->
  <input type="text" class="form-control form-control-sm ipt-tag" v-if="row.inputVisible">
  <button type="button" class="btn btn-primary btn-sm" v-else>+Tag</button>

  <span class="badge badge-warning ml-2" v-for="item in row.tags" :key="item">{{item}}</span>
</td>
```

2. 点击按钮，控制 input 和 button 的切换：

```
<td>
  <!-- 基于当前行的 inputVisible，来控制 input 和 button 的按需展示-->
  <input type="text" class="form-control form-control-sm ipt-tag" v-if="row.inputVisible" />
  <button type="button" class="btn btn-primary btn-sm" v-else @click="row.inputVisible = true">+Tag</button>

  <span class="badge badge-warning ml-2" v-for="item in row.tags" :key="item">{{item}}</span>
</td>
```

5.3 让 input 自动获取焦点

1. 在 `App.vue` 组件中，通过 `directives` 节点自定义 `v-focus` 指令如下：

```
directives: {
    // 封装自动获得焦点的指令
    focus(el) {
        el.focus()
    },
}
```

2. 为 input 输入框应用 `v-focus` 指令：

```
<input type="text" class="form-control ipt-tag form-control-sm" v-if="row.inputVisible" v-focus />
```

5.4 文本框失去焦点自动隐藏

1. 使用 `v-model` 指令把 input 输入框的值双向绑定到 `row.inputValue` 中：

```
<input
  type="text"
  class="form-control ipt-tag form-control-sm"
  v-if="row.inputVisible"
  v-focus
  v-model.trim="row.inputValue"
/>
```

2. 监听文本框的 `blur` 事件，在触发其事件处理函数时，把 `当前行的数据` 传递进去：

```
<input
  type="text"
  class="form-control ipt-tag form-control-sm"
  v-if="row.inputVisible"
  v-focus
  v-model.trim="row.inputValue"
  @blur="onInputConfirm(row)"
/>
```

3. 在 `App.vue` 组件的 `methods` 节点下声明 `onInputConfirm` 事件处理函数：

```
onInputConfirm(row) {
  // 1. 把用户在文本框中输入的值，预先转存到常量 val 中
  const val = row.inputValue
  // 2. 清空文本框的值
  row.inputValue = ''
  // 3. 隐藏文本框
  row.inputVisible = false
}
```


5.5 为商品添加新的 tag 标签

进一步修改 `onInputConfirm` 事件处理函数如下：

```
onInputConfirm(row) {
  // 把用户在文本框中输入的值，预先转存到常量 val 中
  const val = row.inputValue
  // 清空文本框的值
  row.inputValue = ''
  // 隐藏文本框
  row.inputVisible = false

  // 1.1 判断 val 的值是否为空，如果为空，则不进行添加
  // 1.2 判断 val 的值是否已存在于 tags 数组中，防止重复添加
  if (!val || row.tags.indexOf(val) !== -1) return
  // 2. 将用户输入的内容，作为新标签 push 到当前行的 tags 数组中
  row.tags.push(val)
}
```

5.6 响应文本框的回车按键

当用户在文本框中敲击了 回车键 的时候，也希望能够把当前输入的内容添加为 tag 标签。此时，可以为文本框绑定 keyup 事件如下：

```
<input
  type="text"
  class="form-control ipt-tag form-control-sm"
  v-if="row.inputVisible"
  v-focus
  v-model.trim="row.inputValue"
  @blur="onInputConfirm(row)"
  @keyup.enter="onInputConfirm(row)"
/>
```

5.7 响应文本框的 esc 按键

当用户在文本框中敲击了 esc 按键的时候，希望能够快速清空文本框的内容。此时，可以为文本框绑定 keyup 事件如下：

```
<input
  type="text"
  class="form-control ipt-tag form-control-sm"
  v-if="row.inputVisible"
  v-focus
  v-model.trim="row.inputValue"
  @blur="onInputConfirm(row)"
  @keyup.enter="onInputConfirm(row)"
  @keyup.esc="row.inputValue = ''"
/>
```