axios及前后端交互

学习目标：

1.了解axios是什么？什么作用？

2.了解什么是请求和响应？

3.**理解axios用法** -在前后端交互的时候（这部分代码因为要用到后端的python代码，我的要求是python的程序运行就行，不需要看。主要理解axios。这个部分后面会深入讲）

一、什么是axios

Axios是一个基于Promise的**JavaScript库**（*Promise是JavaScript中用于处理异步操作的一种机制，它表示一个异步操作的最终结果。Promise可以有三种状态：pending（进行中）、fulfilled（已成功）和rejected（已失败）。*）

Axios用于发送HTTP请求。它可以在浏览器和Node.js环境中使用，并且具有许多功能，使得在客户端和服务器端进行数据通信变得更加简单和方便。

使用Axios可以更好地管理和处理HTTP请求，包括处理错误、设置请求超时、发送请求数据、接收响应数据等。它已经成为开发人员中广泛使用的HTTP请求库之一，并且在许多前端和后端项目中得到了广泛的应用。

什么是请求和响应?

GET请求和POST请求的区别：

区别方式	GET请求	POST请求
请求参数	请求参数在请求行中。 例：`/brand/findAll?name=OPPO&status=1`	请求参数在请求体中
请求参数长度	请求参数长度有限制(浏览器不同限制也不同)	请求参数长度没有限制
安全性	安全性低。原因：请求参数暴露在浏览器地址栏中。	安全性相对高
常用于	常用于从服务器获取数据，参数通过URL的查询字符串传递。	常用于向服务器提交数据，数据传递在请求的主体中。

另外两种请求方式：

1. **PUT**：
    - **作用**：用于向指定资源位置上传更新后的内容。PUT 请求通常用于更新已存在的资源，如果资源不存在，则会创建一个新的资源。
    - **特点**：PUT 请求中的数据完整替换原始资源，因此需要提供完整的资源内容。
2. **DELETE**：
    - **作用**：用于请求删除指定的资源。
    - **特点**：DELETE 请求用于删除指定的资源，如果资源不存在，则返回404（Not Found）。

API（Application Programming Interface）是指应用程序接口，它定义了软件系统或应用程序提供的功能和服务的一组规范。API可以允许不同的软件系统、应用程序或服务之间进行交互和通信。通过API，开发人员可以访问特定软件系统或服务的功能，并在自己的应用程序中集成或利用这些功能，而无需了解其内部实现细节

二、使用场景

通常**用于网络请求的交互**，它**适用于前端和后端开发**，并且在许多不同的项目中都有应用。

发送HTTP请求：Axios可以用来发送各种类型的HTTP请求，包括GET、POST、PUT、DELETE等。无论是向服务器获取数据还是提交表单数据，Axios都可以处理。

> 拓展：
>
> 客户端 ： 提供给用户的操作界面。
>
> 当我们在客户端点击某一个操作按钮，其实会触发一个请求（发送请求），请求会顺着网络传递给服务器，服务器会去接受这个请求并处理（处理请求），处理完后进行返回请求结果（响应结果）。
>
> 服务器 ： 提供服务的机器。用来运行项目的电脑。
>
> 目前我们进行的axios操作就是实现，我们前端去向后端发送请求，并让后端响应结果，拿到后端响应的结果后，可以进行操作。

三、引入方式

1. 通过axios本地包导入（先要将工具包放到写代码的文件夹中）

```
<script src="axios.js"></script>
```

四、简单使用

- 步骤一、创建请求对象
- 步骤二、请求方式
- 步骤三、请求接口地址
- 步骤四、请求数据
- 步骤五、请求结果处理

```
<script>
    //传递一个配置对象作为参数，来创建一个Axios请求对象。该配置对象包含了请求的方法、URL以及其他可选的配置项
    axios({
      method: 'get',         // 发送 get 请求
      url: 'http://43.136.104.16:3002/xx接口',
    }).then((res) => {
      // 处理成功的结果
```

```
    }).catch((error) => {
      // 处理错误
    })
   /*    这段代码使用Axios发送了一个GET请求到http://43.136.104.16:3002/xx接口，并在控制台中打印响应结果。
    .then()方法用于处理成功状态。
    .catch()方法用于处理失败状态。                          */
</script>
```

1. **GET**：用于请求从服务器获取数据。GET请求应该只用于获取数据，并不会引起服务器上的数据变化。在Flask中，可以通过`@app.route()`装饰器指定`methods=['GET']`来处理GET请求。
2. **POST**：用于向服务器提交新的数据，通常会导致服务器上的数据变化，如创建新的资源。在Flask中，可以通过`@app.route()`装饰器指定`methods=['POST']`来处理POST请求。POST请求经常用于表单提交。
3. **PUT**：用于更新服务器上的现有资源。PUT请求应该包含待更新资源的所有信息，服务器会用这些信息完全替换现有资源。在Flask中，处理PUT请求的方式与POST类似，也是通过`methods=['PUT']`指定。
4. **DELETE**：用于删除服务器上的资源。DELETE请求不包含请求体，仅指定要删除的资源的URI。在Flask中，处理DELETE请求也是通过`methods=['DELETE']`指定。

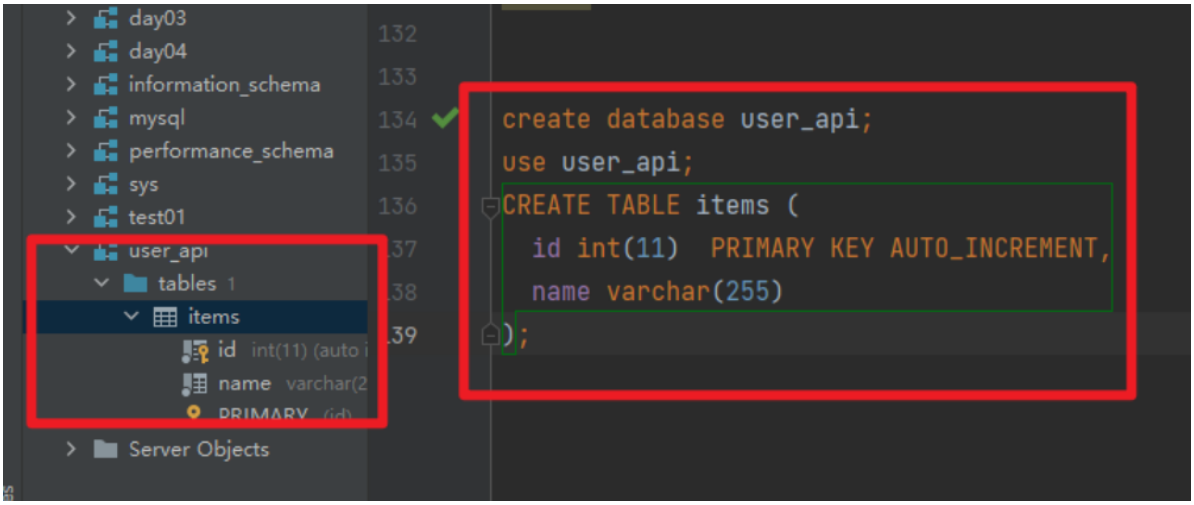
五、请求参数

参数	作用
method	请求方式 默认为GET
url	请求地址
data	**data是作为请求主体被发送的数据** 一般用于post
params	是即将与请求一起发送的 URL 参数 适用`get`请求
timeout	请求时间
.then()	回调函数 用于获取服务器返回的数据
.catch()	用于处理失败状态

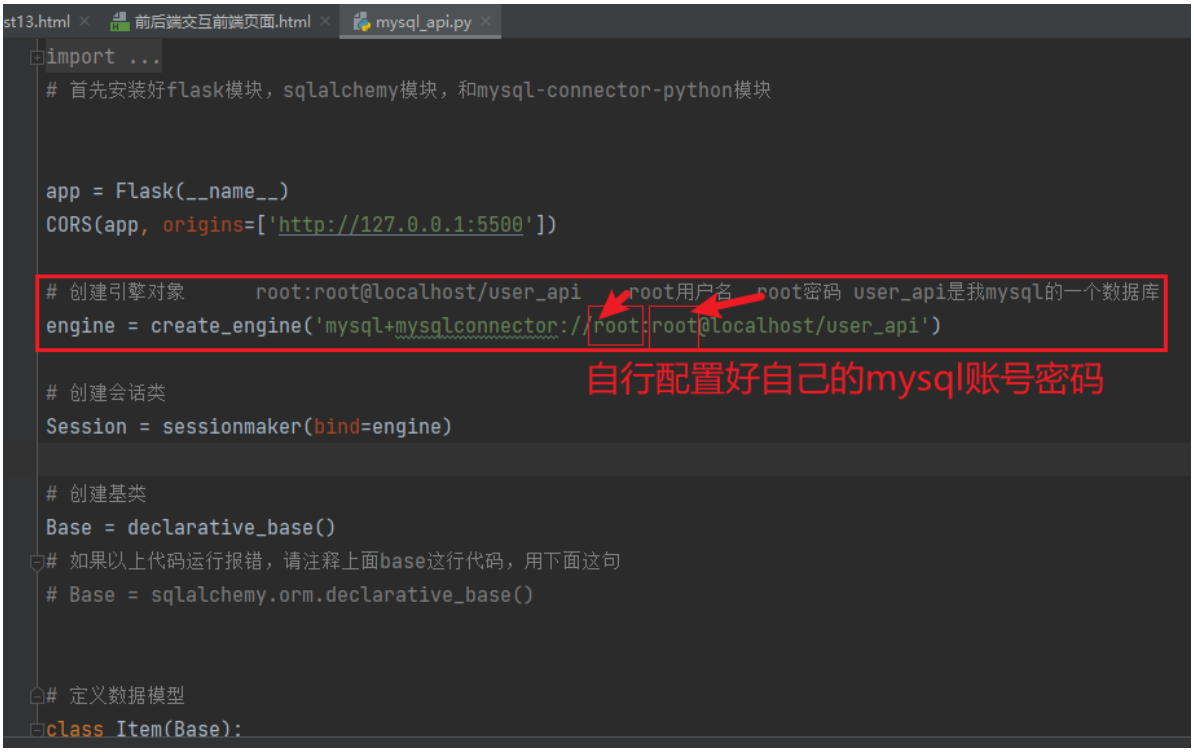
六、使用接口对数据进行增删改查(本地服务器)

准备工作：

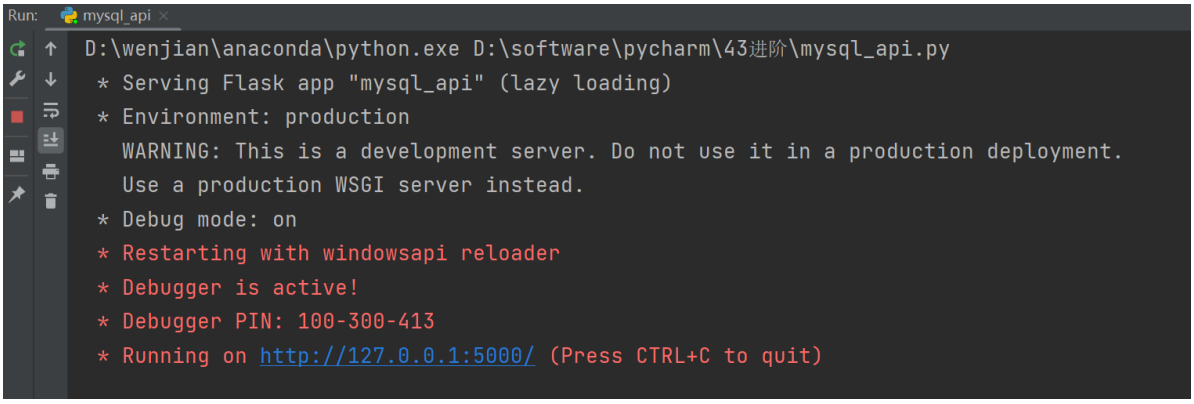
启动小皮创建好**数据库user_api**和**items**表



1、下载好mysql_api.py这个python文件，并且配置好自己的mysql的信息，运行他（这个是我写的后端接口）



2、右击运行这个文件




```
http://localhost:5000/api/items
```

通过http协议，访问ip为127.0.0.1的服务器，中端口为5000的服务（程序），api/items 是这个程序的一个功能。

查操作

```
fetchItems() {
    // 发送 GET 请求获取项目列表数据
    axios({
        method: 'get',
        url: "http://localhost:5000/api/items"
    }).then(response => {
        this.items = response.data; // 将返回的项目列表数据存储到 items 数组中
    }).catch(error => {
        console.error('获取数据出错: ', error); // 捕获错误并输出到控制台
    });
},
```

增加操作

```
addItem() {
    // 发送 POST 请求添加新项目
    axios({
        method: 'post', // post请求
        url: 'http://localhost:5000/api/items',
        data: { name: this.newItem } // 请求的数据为新项目的名称
    }).then(response => {
        this.items.push(response.data); // 将新添加的项目数据添加到 items 数组中
        this.newItem = ''; // 清空输入框
    }).catch(error => {
        console.error('添加项目出错: ', error); // 捕获错误并输出到控制台
    });
},
```

删除操作

```
deleteItem(id) {
    // 发送 DELETE 请求删除指定 ID 的项目
    axios({
        method: 'delete',
        url: `http://localhost:5000/api/items/${id}`
    }).then(() => {
        // 重新调用查询方法
        this.fetchItems();
    }).catch(error => {
        console.error('删除项目出错: ', error); // 捕获错误并输出到控制台
    });
},
```

修改操作

```
updateItem(id) {
    var newName = this.updatedItem[id]; // 获取要更新的项目名称
    // 发送 PUT 请求更新指定 ID 的项目名称
    axios({
        method: 'put',
        url: `http://localhost:5000/api/items/${id}`,
        data: { name: newName } // 请求的数据为更新后的项目名称
    }).then(() => {
```

```
            this.updatedItem[id] = ''; // 清空输入框
            this.fetchItems(); // 更新项目列表
        }).catch(error => {
            console.error('修改项目出错: ', error); // 捕获错误并输出到控制台
        });
    }
```

综合实例完整代码：

```
<!DOCTYPE html>
<html lang="zh-CN">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vue MySQL CRUD</title>
    <script src="vue.js"></script>
    <!-- 导入axios包 -->
    <script src="axios.js"></script>
</head>

<body>
<div id="app">
    <h1>前后端交互对 MySQL 增删改查</h1>
    <!-- 输入框和按钮，用于添加新项目 -->
    <input type="text" v-model="newItem" placeholder="请输入新项目">
    <button @click="addItem">添加项目</button>
    <ul>
        <!-- 遍历显示项目列表 -->
        <li v-for="item in items" :key="item.id">
            {{ item.name }} <!-- 显示项目名称 -->
            <!-- 每个项目后面都有一个删除按钮，点击时触发 deleteItem 方法 -->
            <button @click="deleteItem(item.id)">删除</button>
            <!-- 输入框用于修改项目名称 -->
            <input type="text" v-model="updatedItem[item.id]" placeholder="修改项
目">
            <!-- 每个项目后面都有一个修改按钮，点击时触发 updateItem 方法 -->
            <button @click="updateItem(item.id)">修改</button>
        </li>
    </ul>
</div>

<script>
    new Vue({
        el: '#app', // 指定 Vue 实例挂载的元素
        data: {
            newItem: '', // 新增项目的名称
            items: [], // 存储所有项目的数组
            updatedItem: {} // 存储修改后项目的名称
        },
        mounted() {
            this.fetchItems(); // 在 Vue 实例挂载后调用 fetchItems 方法获取项目列表
        },
        methods: {
            fetchItems() {
                // 发送 GET 请求获取项目列表数据
                axios({
```

```
                    method: 'get',
                    url: "http://localhost:5000/api/items"
                }).then(response => {
                    console.log(response)
                    this.items = response.data; // 将返回的项目列表数据存储到 items
数组中
                }).catch(error => {
                    console.error('获取数据出错：', error); // 捕获错误并输出到控制台
                });
            },
            addItem() {
                // 发送 POST 请求添加新项目
                axios({
                    method: 'post',
                    url: 'http://localhost:5000/api/items',
                    data: { name: this.newItem } // 请求的数据为新项目的名称
                }).then(response => {
                    this.items.push(response.data); // 将新添加的项目数据添加到 items
数组中
                    this.newItem = ''; // 清空输入框
                }).catch(error => {
                    console.error('添加项目出错：', error); // 捕获错误并输出到控制台
                });
            },
            deleteItem(id) {
                // 发送 DELETE 请求删除指定 ID 的项目
                axios({
                    method: 'delete',
                    url: `http://localhost:5000/api/items/${id}`
                }).then(() => {
                    // 重新调用查询方法
                    this.fetchItems();
                }).catch(error => {
                    console.error('删除项目出错：', error); // 捕获错误并输出到控制台
                });
            },
            updateItem(id) {
                var newName = this.updatedItem[id]; // 获取要更新的项目名称
                // 发送 PUT 请求更新指定 ID 的项目名称
                axios({
                    method: 'put',
                    url: `http://localhost:5000/api/items/${id}`,
                    data: { name: newName } // 请求的数据为更新后的项目名称
                }).then(() => {
                    this.updatedItem[id] = ''; // 清空输入框
                    this.fetchItems(); // 更新项目列表
                }).catch(error => {
                    console.error('修改项目出错：', error); // 捕获错误并输出到控制台
                });
            }
        }
    });
</script>
</body>
</html>
```

作业完成和理解好这个案例

截图或者录屏你实现的效果