简易本地前后端联通测试教程

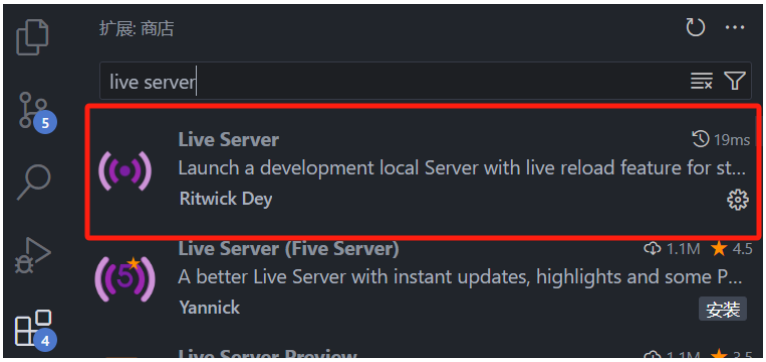
> Written by LCB, 2024.07.13

本文档将教学如何在开发时，在不将代码部署至服务器的情况下，测试前端是否能调用后端函数，往数据库内读/写数据。

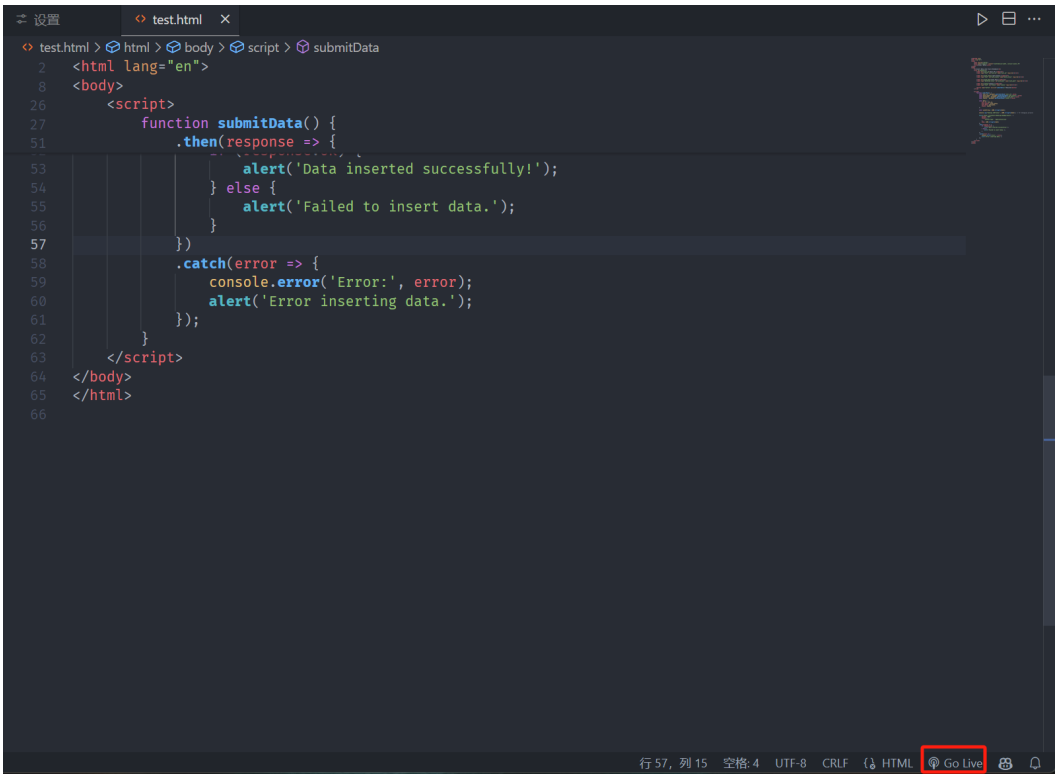
注意，本教程内提供的前端代码基于原生HTML，不考虑对vue的支持，如果大家想用已开发的vue前端进行测试，可以自行在本教程基础上进行探索。

0x10 安装VsCode插件

1. 在VsCode的拓展商店中，搜索并安装Live Server插件：

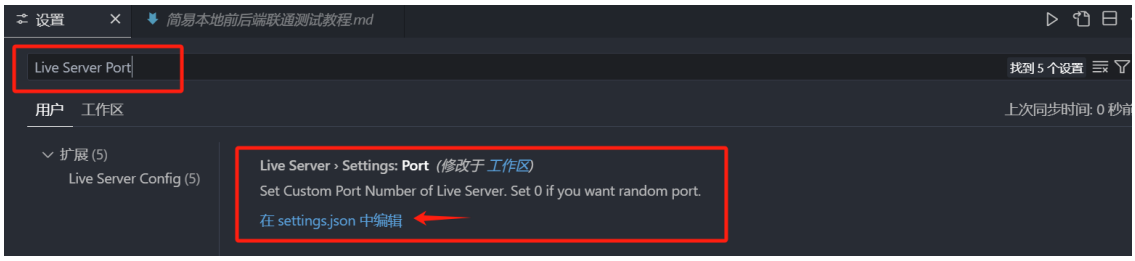


2. 安装完毕后，新建一个HTML文件，即可在窗口右下角看到`Go Live`图标：



3. 修改Live Server默认端口：
    - Live Server默认运行在5500端口上，但是这个端口很容易被占用，故我们需要修改默认端口
    - 进入VsCode设置：

- 在上方搜索栏中输入 `Live Server Port` 看到如下设置
- 点击 `在 settings.json 中编辑`



- 将此条中原先的0，修改为`7692`，并按CTRL+S保存即可

> 此端口可以任意，但是为了与后续的后端代码修改保持一致，笔者此处修改为7692

0x20 前端代码部署

此处作者提供一个非常非常简单的前端代码进行操作，仅支持对数据库表的查询及插入操作

0x21 测试插入.html

代码文件见本教程所在文件夹中附件，代码分为两部分，一部分是前端输入框，一部分是前端调用后端API接口函数

输入框:

```
<form id="dataForm">
        <label for="user_id">User ID:</label><br>
        <input type="text" id="user_id" name="user_id" required><br><br>

        <label for="auth_status">Auth Status:</label><br>
```

```
        <input type="text" id="auth_status" name="auth_status" required><br><br>

        <label for="auth_date">Auth Date:</label><br>
        <input type="datetime-local" id="auth_date" name="auth_date" required>
<br><br>

        <label for="status">Status:</label><br>
        <input type="text" id="status" name="status" required><br><br>

        <button type="button" onclick="submitData()">Execute</button>
    </form>
```

该文件默认向`Auth_Info`表中插入数据，故此处为4条输入内容，如果大家需要修改为对自己的表进行插入测试，修改相应的代码即可

```
如：我要向xx表内插入一个int类型的属性，属性名叫Stu_ID,修改代码为：
<label for="stu_id">Stu_ID:</label><br>
<input type="int" id="stu_id" name="stu_id" required><br><br>
注：html属性内大小写不敏感，但是在下方建立数据的时候大小写敏感，后面会详细说明
```

API调用

API调用部分主要就是在点击`Execute`按钮后，收集用户输入，构建JSON字符串，调用API发送给后端

1. 收集数据部分

```
const user_id = document.getElementById('user_id').value;
const auth_status = document.getElementById('auth_status').value;
const auth_date = document.getElementById('auth_date').value;
const status = document.getElementById('status').value;
```

注意，这里的变量名（const后的xxx）可以随意，但是`getElementById()`中的变量名，一定要是你上方在表格中定义的`<label for="stu_id">`及`<input type="int" id="stu_id" name="stu_id" required><br><br>`里面的变量名一致，也就是说，下图中的三个地方变量名需要一致：

比如说，我刚刚建立了了一条

```
<label for="stu_id">Stu_ID:</label><br>
<input type="int" id="stu_id" name="stu_id" required><br><br>
```

那么我此处收集信息时就应该是：

```
<script>
    function submitData() {
        ...
        const stu_id = document.getElementById('stu_id').value;
        ...
    }
</script>
```

其实笔者建议只要是同一个变量的所有名称都一致，省的出现问题

2. 构建JSON字符串

```
const data = {
     User_ID: user_id,
     Auth_Status: auth_status,
     Auth_Date: auth_date,
     Status: status
};
console.log("Sending JSON data:", JSON.stringify(data)); // For debugging
purposes
```

定义一个data变量后，即可构建字符串，对于一个数据项`User_ID: user_id`，前者为数据表内属性

注意，此处大小写敏感，如果你的表里的属性叫ReTuRn_DATe，那么你此处构建的数据项也应该是`ReTuRn_DATe : return_date`，否则插入100%不成功！！！

后者即为上面收集用户输入时定义的变量名。

此处还贴心的添加了一条console.log()，可以按下F12，在弹出的开发者窗口中点击控制台即可查看构造的JSON数据串。

3. 调用后端API

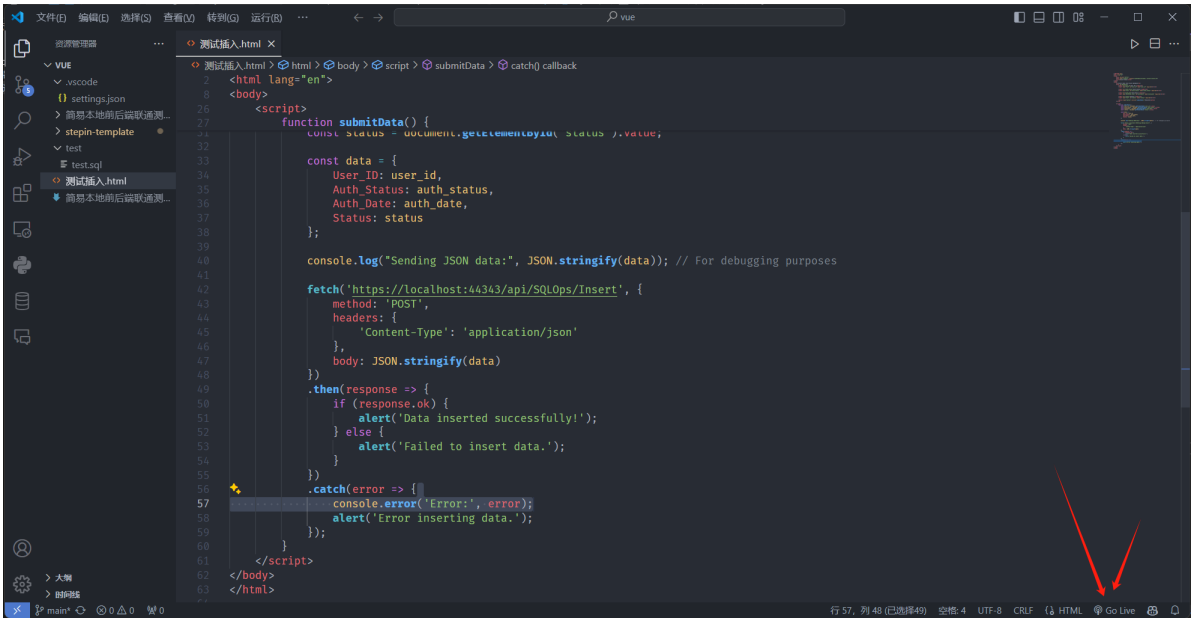
```
fetch('https://localhost:44343/api/SQLOps/Insert', {
    method: 'POST',
    headers: {
    'Content-Type': 'application/json'
    },
    body: JSON.stringify(data)
```

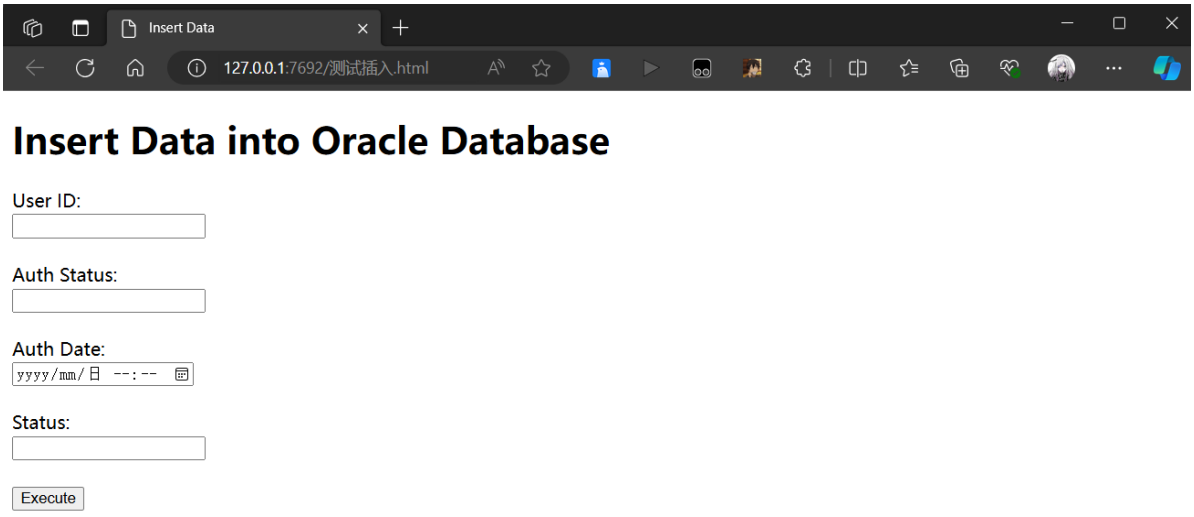
```
})
    .then(response => {
    if (response.ok) {
        alert('Data inserted successfully!');
    } else {
        alert('Failed to insert data.');
    }
})
.catch(error => {
    console.error('Error:', error);
    alert('Error inserting data.');
});
```

此处调用的API为`/api/SQLOps/Insert`，此api是定义在`WebAppTest-APITemplate-SQLOpsController.cs`中的，并且插入操作使用的是POST操作，定义如上，如果要测试自己的API，只要修改fetch 内部的内容以及method即可。

部署至网页

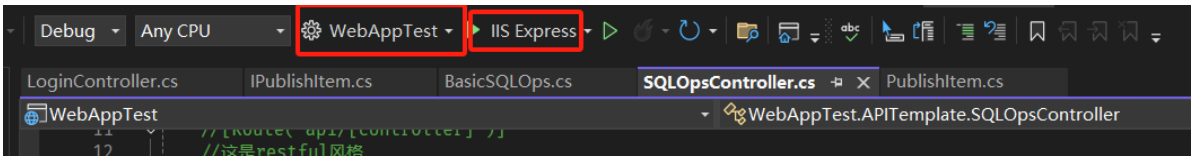
编写好html代码后，点击VsCode右下角的Go Live，即会自动部署，打开一个网页。





0x30 后端代码修改

安装好必备的运行库后，打开解决方案，选中WebAppTest，并点击右侧的IIS Express调试，既可以打开后端服务器（此处为模拟在服务器上操作）。



弹出网页后，无需操作，仍然在刚刚编写的前端网页上操作，不过由于是调试模式，可以在自己编写的代码上打断点进行debug。

0x31 解决跨域错

由于模拟服务器运行在44343端口上，我们的前端网页运行在7692端口上，故两个不同域的服务互相访问时，会出现跨域错，我们需要为后端添加白名单，允许7692端口访问，修改`WebAPPTest`中的`Program.cs`代码如下（此处不解释，真正部署至服务器端的代码不会有这个问题，无脑复制即可）

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at
https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

// 添加CORS服务
builder.Services.AddCors(options =>
{
    options.AddPolicy("AllowSpecificOrigin",
        builder =>
        {
            builder.WithOrigins("http://127.0.0.1:7692")
                   .AllowAnyHeader()
                   .AllowAnyMethod();
        });
});


var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

app.UseCors("AllowSpecificOrigin"); // Use the CORS policy

app.UseAuthorization();

app.MapControllers();
```

```
app.Run();
```

0x40 结语

本文档给出的只是一个非常简单的前后端测试代码，笔者还会提供一个自定义查询的html，如果大家需要将自己的vue进行测试，则方法类似，vue部署完毕后，假设运行在8888端口上，只需要在Program.cs中，`builder.WithOrigins("http://127.0.0.1:7692")`内的端口号改为8888即可。

0x41 测试查询ps

同样的，查询调用的函数也是定义在`WebAppTest-APITemplate-SQLOpsController.cs`下的，默认查询Auth_Info表中，id为400的数据，如需自定义查询，请自行修改。

0x42 版本控制ps

鉴于大家在运行测试前，备份一份源文件，以防版本出错。