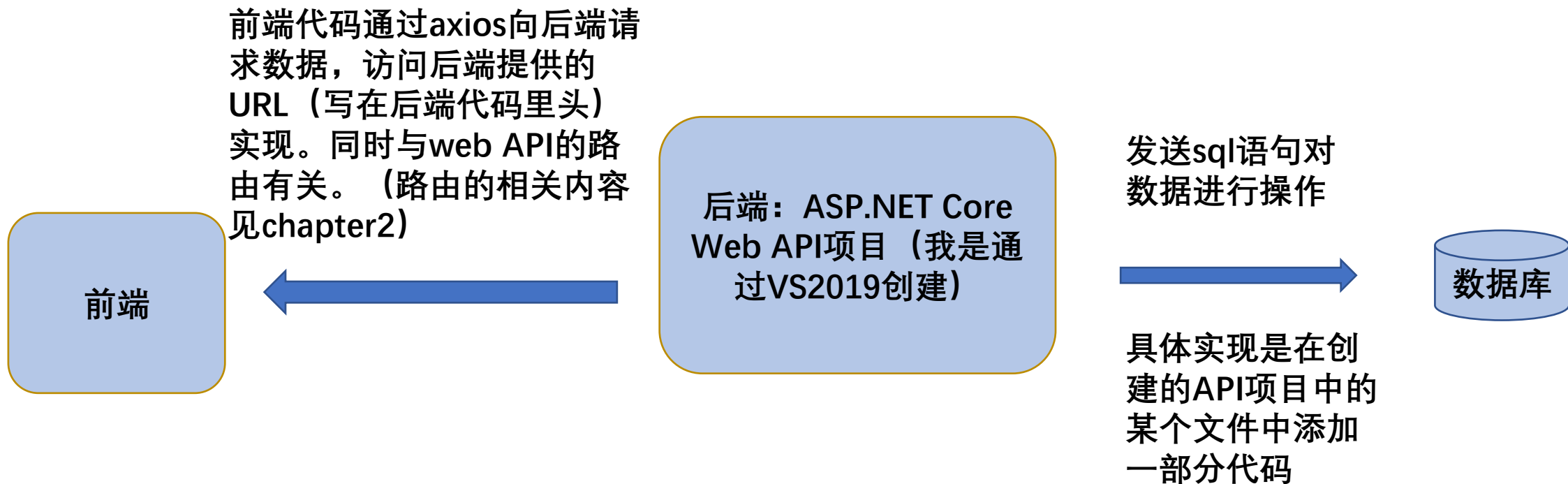


后端项目的大致原理（个人理解，有误可指正哈~）



目录

- Chapter1 API项目的大致结构
- Chapter2 创建API&&连接数据库
- Chapter3 对数据库的修改
- Chapter4 axios介绍

Chapter1: API项目的大致结构

1、在VS上下载并安装ASP.NET的包：



正在修改 - Visual Studio Community 2019 - 16.7.3

[工作负荷](#) 单个组件 语言包 安装位置

Web 和云 (4)



ASP.NET 和 Web 开发

使用 ASP.NET Core、ASP.NET、HTML/JavaScript 和包括 Docker 支持的容器生成 Web 应用程序。



Python 开发

对 Python 进行编辑、调试、交互式开发和源代码管理。



桌面应用和移动应用 (5)

2、创建项目，我这里出现了一个问题：找不到ASP.NET Core Web API的模板，只能先尝试创建应用程序。否则直接创建ASP.NET Core Web API就好了。



发现**API**的创建在应用程序里面！！

创建新的 ASP.NET Core Web 应用程序

.NET Core ASP.NET Core 3.1

 **空**
用于创建 ASP.NET Core 应用程序的空项目模板。此模板中没有任何内容。

 **API**
用于创建包含 RESTful HTTP 服务示例控制器的 ASP.NET Core 应用程序的项目模板。此模板还可以用于 ASP.NET Core MVC 视图和控制器。

 **Web 应用程序**
用于创建包含示例 ASP.NET Core Razor 页面内容的 ASP.NET Core 应用程序的项目模板。

 **Web 应用程序(模型视图控制器)**
用于创建包含示例 ASP.NET Core MVC 视图和控制器的 ASP.NET Core 应用程序的项目模板。此模板还可以用于

3、B站上教程对**controllers**的讲解

The image shows a Visual Studio editor window with the code for `WeatherForecastController` in the `YiXunWebAPI` project. The code is annotated with red text and arrows explaining key concepts:

- 1、Controllers: 存放对外接口的类** (1. Controllers: Class for storing external interfaces) - Points to the `WeatherForecastController` class.
- 2、两个特性:** (2. Two characteristics:)
 - `[ApiController]` is annotated as **自动推断的特性 (初学者不用管)** (Automatic inference characteristic (beginners don't need to worry)).
 - `[Route("[controller]")]` is annotated as **路由规则** (Routing rule).
- 3、这里的Controller是一个命名规范, 如果使用Control作为接口, 类必须以Controller作为结尾** (3. Here, Controller is a naming convention. If Control is used as an interface, the class must end with Controller) - Points to the `Controller` part of the class name.

The code in the editor is as follows:

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Threading.Tasks;
5 using Microsoft.AspNetCore.Mvc;
6 using Microsoft.Extensions.Logging;
7 // 自动推断的特性 (初学者不用管)
8 namespace YiXunWebAPI.Controllers
9 {
10     [ApiController]
11     [Route("[controller]")]
12     // 路由规则
13     public class WeatherForecastController : ControllerBase
14     {
15         private static readonly string[] Summaries = new[]
16         {
17             "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm"
18         };
19         private readonly ILogger<WeatherForecastController> _logger
20         // 0 个引用
21         public WeatherForecastController(ILogger<WeatherForecastCon
22         {
```

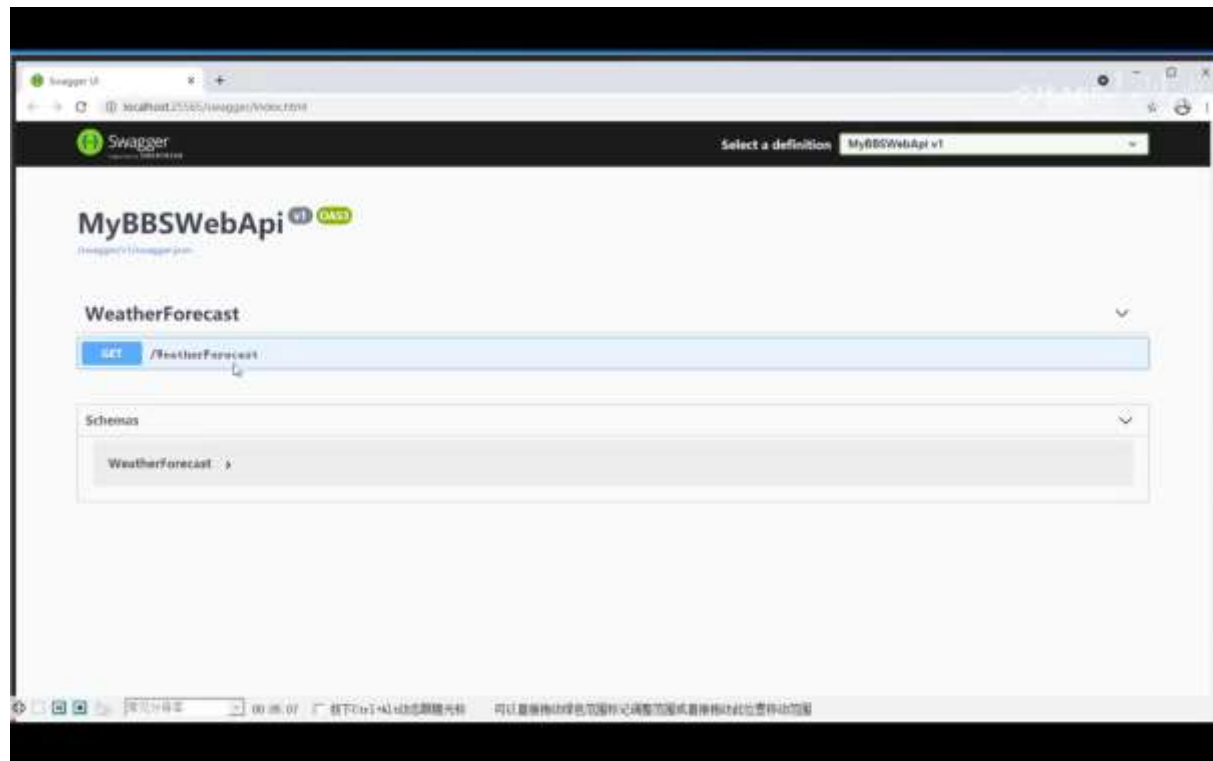
The Solution Explorer on the right shows the project structure:

- 解决方案"YiXunWebAPI"(1 个项目/共 1 个)
 - YiXunWebAPI
 - Connected Services
 - Properties
 - 依赖项
 - Controllers
 - WeatherForecastController.cs
 - WeatherForecastController
 - appsettings.json
 - Program.cs
 - Startup.cs
 - WeatherForecast.cs

4、这时出现了一个新问题：**B**站上教程调试后得到的是一个有一定图形界面的网页，但是我的只有一串**JSON**字符？

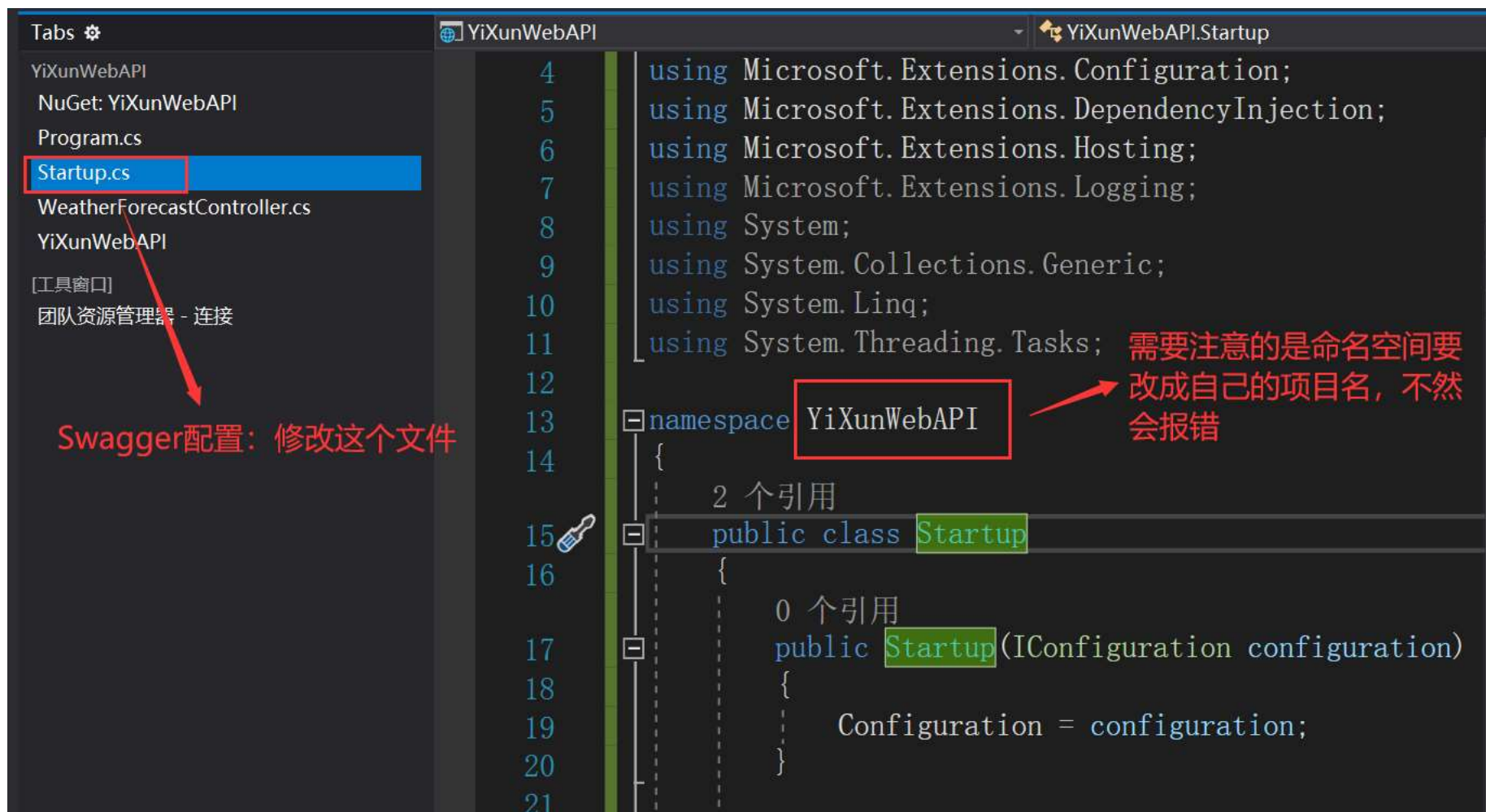
```
[{"date": "2022-07-27T16:14:12.4423835+08:00", "temperatureC": 50, "temperatureF": 121, "summary": "Scorching"}, {"date": "2022-07-28T16:14:12.4449376+08:00", "temperatureC": 28, "temperatureF": 82, "summary": "Chilly"}, {"date": "2022-07-29T16:14:12.4449473+08:00", "temperatureC": 7, "temperatureF": 44, "summary": "Cool"}, {"date": "2022-07-30T16:14:12.4449479+08:00", "temperatureC": -4, "temperatureF": 25, "summary": "Cool"}, {"date": "2022-07-31T16:14:12.4449482+08:00", "temperatureC": 24, "temperatureF": 75, "summary": "Warm"}]
```

B站上的运行截图：

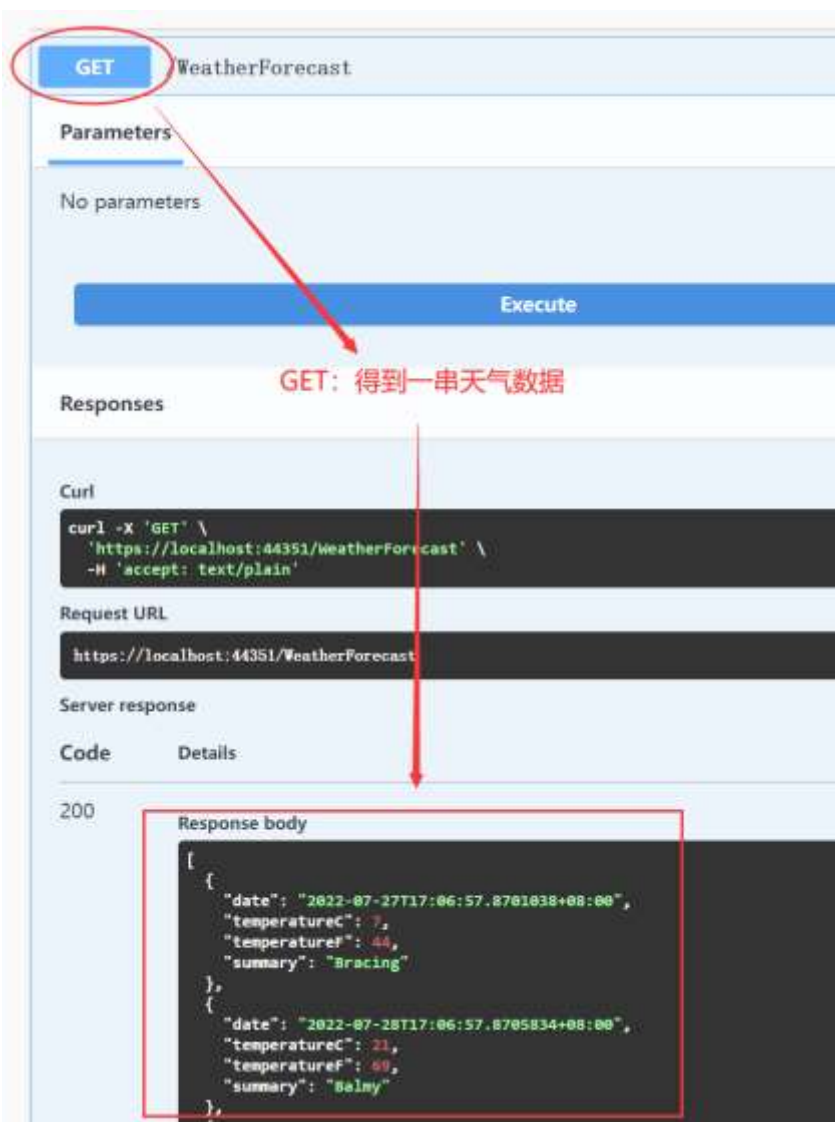


解决方案：[swagger配置](#) ([Web 视图](#))

5、对解决方案的一点说明：将教程上的代码全选复制粘贴到**Startup.cs**文件里就**OK**了，但需要注意的是：



6、运行后在网页上：Get → Try it out → Execute



GET WeatherForecast

Parameters

No parameters

Execute

Responses

Curl

```
curl -X 'GET' \
  'https://localhost:44351/WeatherForecast' \
  -H 'accept: text/plain'
```

Request URL

```
https://localhost:44351/WeatherForecast
```

Server response

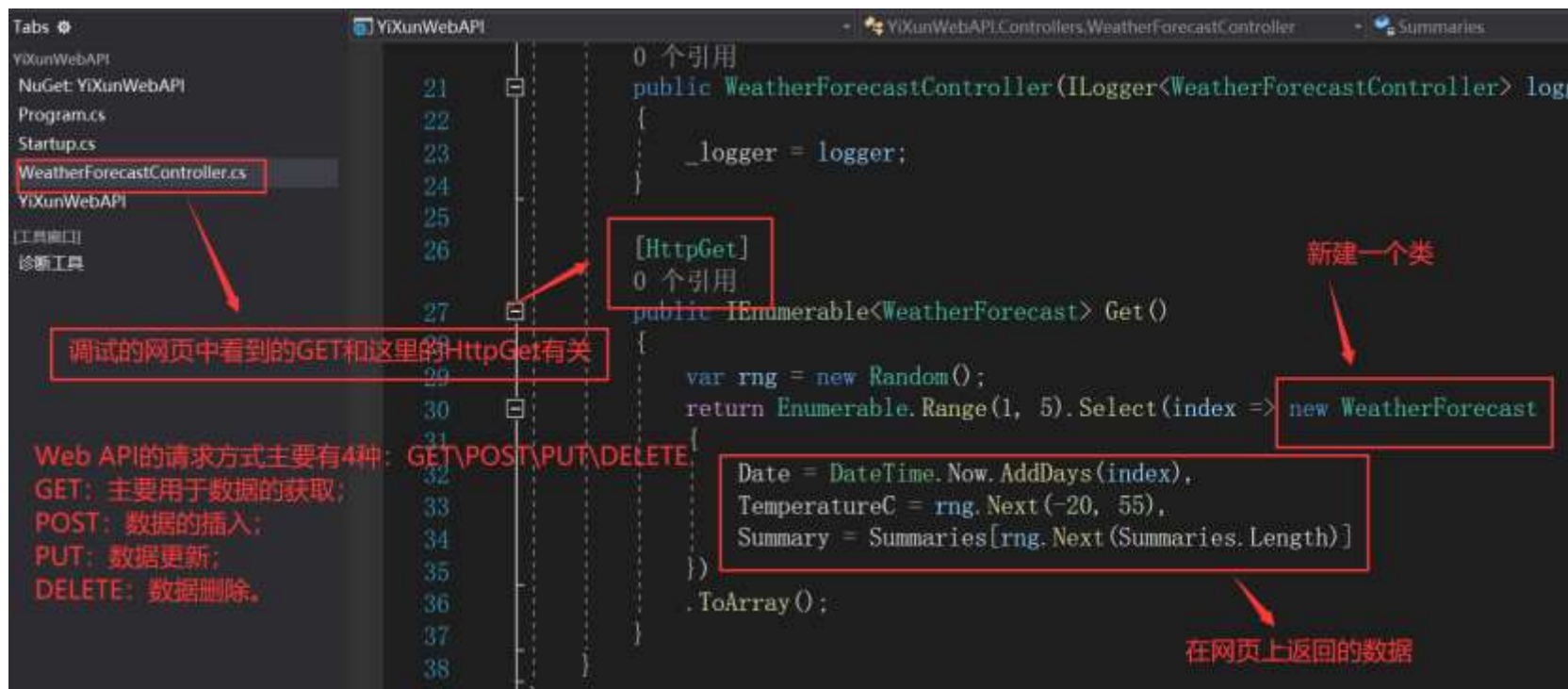
Code Details

200

Response body

```
[
  {
    "date": "2022-07-27T17:06:57.8761038+08:00",
    "temperatureC": 7,
    "temperatureF": 44,
    "summary": "Bracing"
  },
  {
    "date": "2022-07-28T17:06:57.8705834+08:00",
    "temperatureC": 21,
    "temperatureF": 69,
    "summary": "Sunny"
  }
]
```

GET: 得到一串天气数据



YiXunWebAPI

YiXunWebAPI

NuGet: YiXunWebAPI

Program.cs

Startup.cs

WeatherForecastController.cs

YiXunWebAPI

[工具窗口]

诊断工具

0 个引用

```
public WeatherForecastController (ILogger<WeatherForecastController> logger)
{
    _logger = logger;
}
```

[HttpGet]

0 个引用

```
public IEnumerable<WeatherForecast> Get()
{
    var rng = new Random();
    return Enumerable.Range(1, 5).Select(index => new WeatherForecast
    {
        Date = DateTime.Now.AddDays(index),
        TemperatureC = rng.Next(-20, 55),
        Summary = Summaries[rng.Next(Summaries.Length)]
    })
    .ToArray();
}
```

新建一个类

调试的网页中看到的GET和这里的HttpGet有关

Web API的请求方式主要有4种：GET/POST/PUT/DELETE

GET: 主要用于数据的获取;

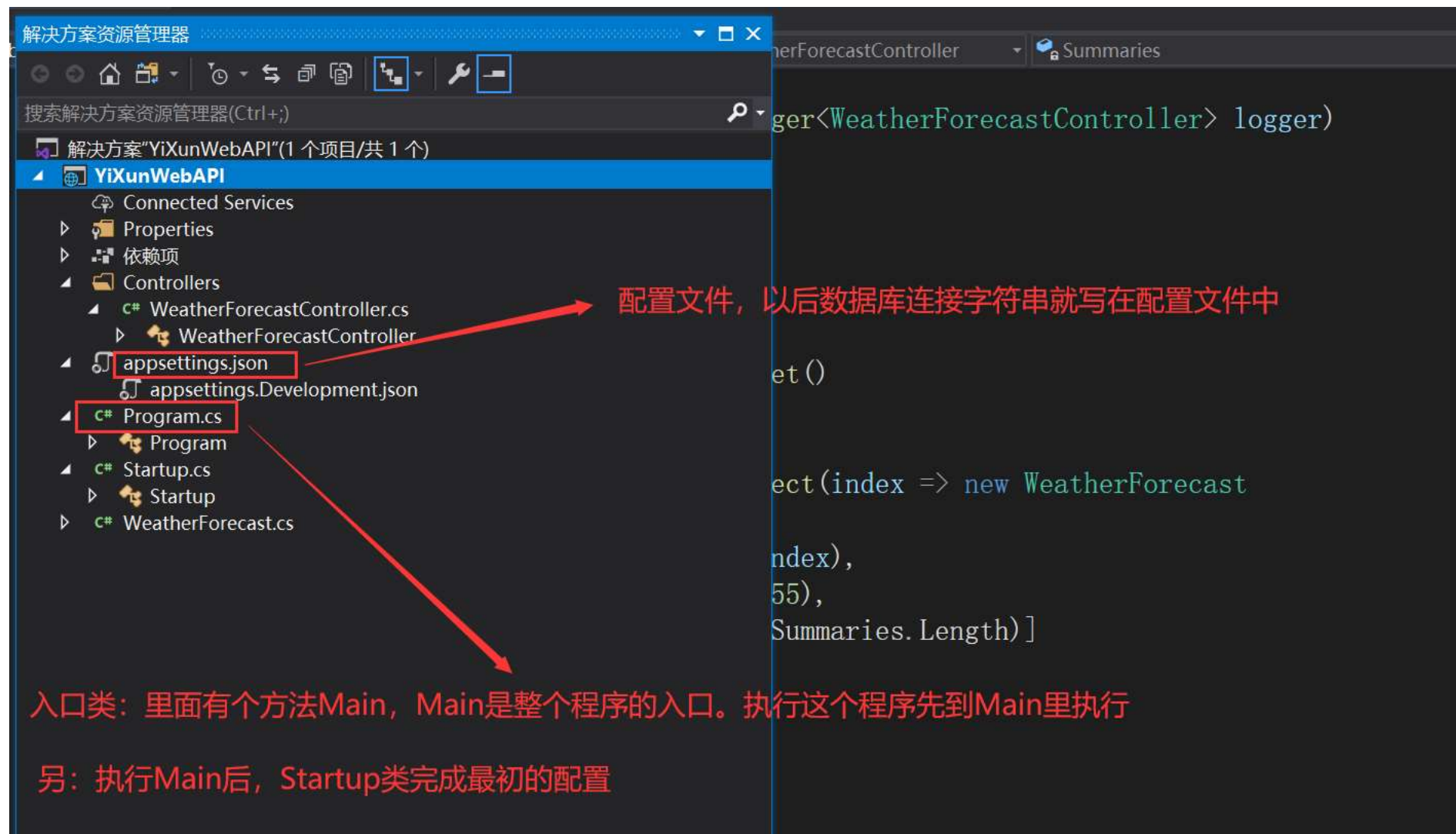
POST: 数据的插入;

PUT: 数据更新;

DELETE: 数据删除。

在网页上返回的数据

7、其他一些文件的主要用途概览



解决方案资源管理器

搜索解决方案资源管理器(Ctrl+;)

解决方案"YiXunWebAPI"(1 个项目/共 1 个)

- YiXunWebAPI
 - Connected Services
 - Properties
 - 依赖项
 - Controllers
 - WeatherForecastController.cs
 - WeatherForecastController
 - appsettings.json
 - appsettings.Development.json
 - Program.cs
 - Startup.cs
 - Startup
 - WeatherForecast.cs

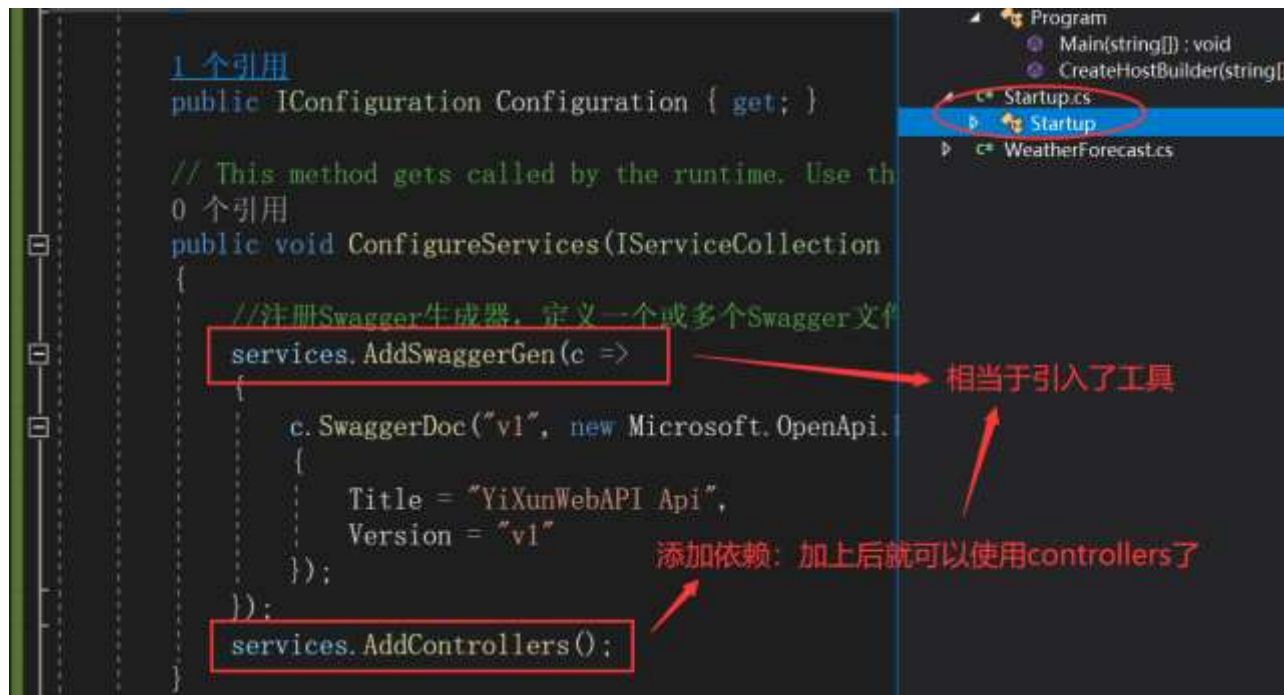
配置文件，以后数据库连接字符串就写在配置文件中

入口类：里面有个方法Main，Main是整个程序的入口。执行这个程序先到Main里执行

另：执行Main后，Startup类完成最初的配置

```
ger<WeatherForecastController> logger)  
  
et()  
  
ect(index => new WeatherForecast  
  
ndex),  
55),  
Summaries.Length)]
```

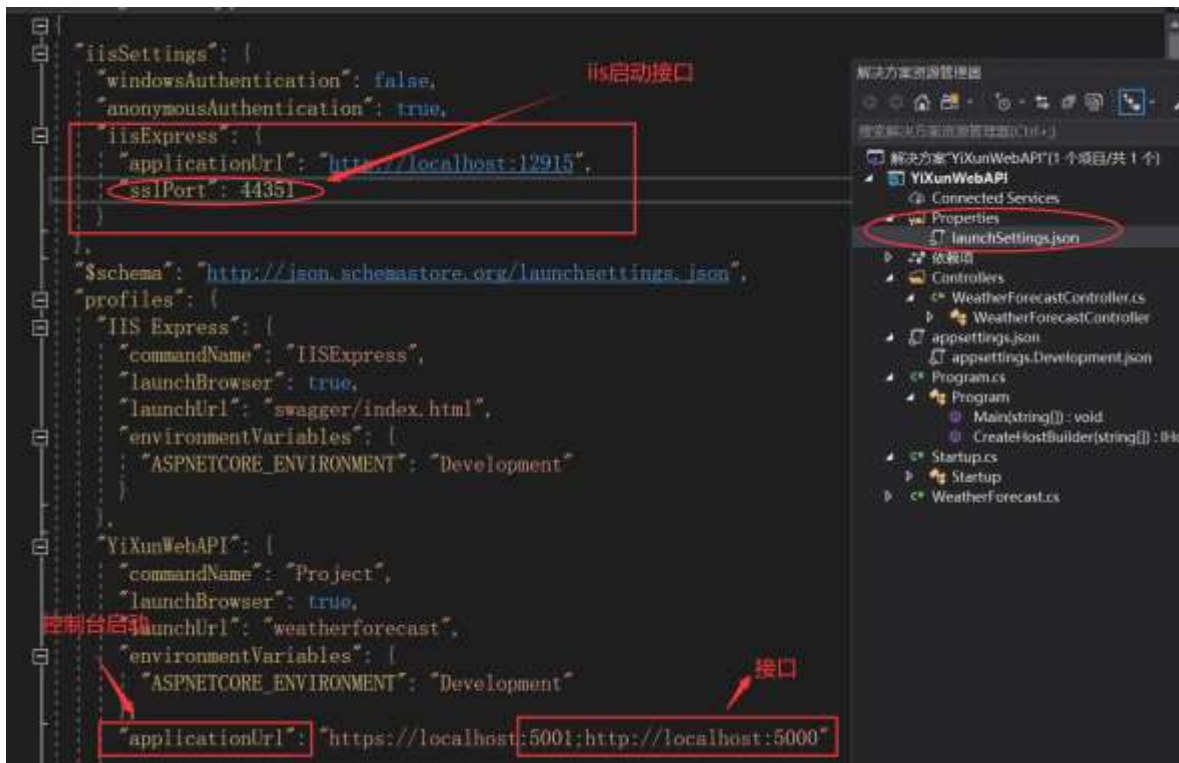
8、对Startup.cs文件中代码的一些讲解



9、总结：

- 整个项目程序的入口在**Program.cs**中的**Main**方法（相当于**C++**的**main**函数）
- **Startup.cs**文件主要负责初始的一些配置。**Services.*****的代码用于引入工具（例如**Services.AddControllers()**后就可以使用**controllers**这个模块）；**app.*****的代码是使用之前引入的工具（例如在上一页的代码中启用了**swagger**）。
- **Controllers**文件放置连接外部的**API**，其中一些方法，例如之前提到的**GET**，就是于外部网页传递数据的接口。
- **Appsettings.json**是配置文件。**.json**结尾的文件都是配置文件。

运行流程



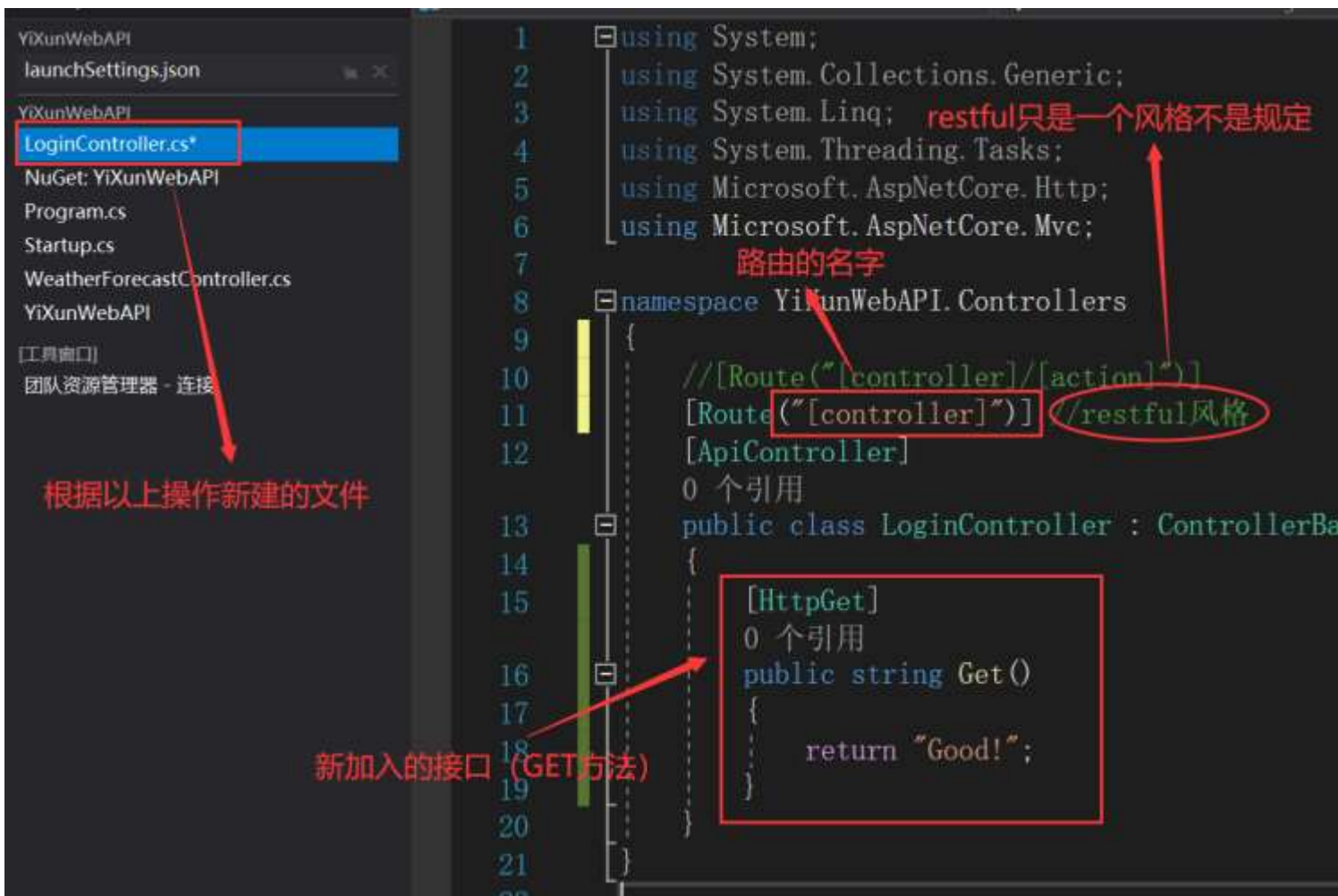
Chapter2: 创建API&&连接数据库

1、创建接口

Controller文件夹右键 → 添加 → 控制器 → **API(API 控制器 空)**

注意：命名规则，末尾的**Controll**不能删掉！！

一个简单的接口 (**GET**)：其他方法还有**POST/PUT/DELETE**



The screenshot shows the Visual Studio IDE. On the left, the 'Solution Explorer' displays the project structure for 'YiXunWebAPI'. A red box highlights 'LoginController.cs*' under the 'Controllers' folder. A red arrow points from this box to the main code editor. The code editor shows the following code:

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Threading.Tasks;
5 using Microsoft.AspNetCore.Http;
6 using Microsoft.AspNetCore.Mvc;
7
8 namespace YiXunWebAPI.Controllers
9 {
10     // [Route("[controller]/[action]")]
11     [Route("[controller]")]
12     [ApiController]
13     public class LoginController : ControllerBase
14     {
15         [HttpGet]
16         public string Get()
17         {
18             return "Good!";
19         }
20     }
21 }
```

Annotations in the code editor:

- A red arrow points to the `[Route("[controller]")]` attribute with the text "路由的名字" (Route name).
- A red arrow points to the `// [Route("[controller]/[action]"]]` comment with the text "restful只是一个风格不是规定" (restful is just a style, not a rule).
- A red arrow points to the `[HttpGet]` attribute with the text "新加入的接口 (GET方法)" (Newly added interface (GET method)).

Below the code editor, a red arrow points to the text "根据以上操作新建的文件" (New file created according to the above operations).



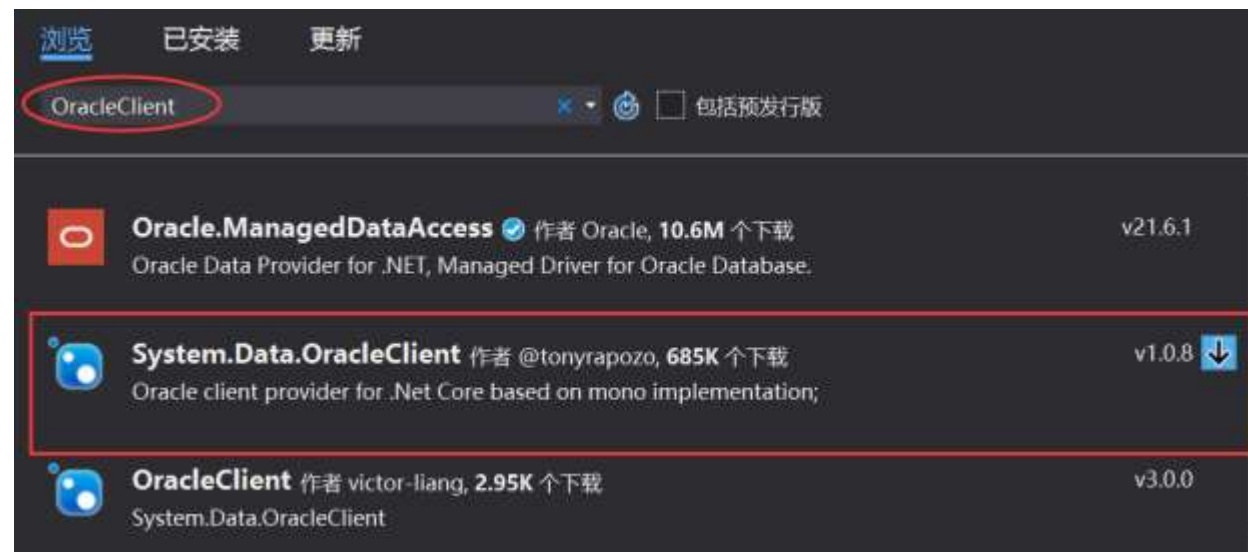
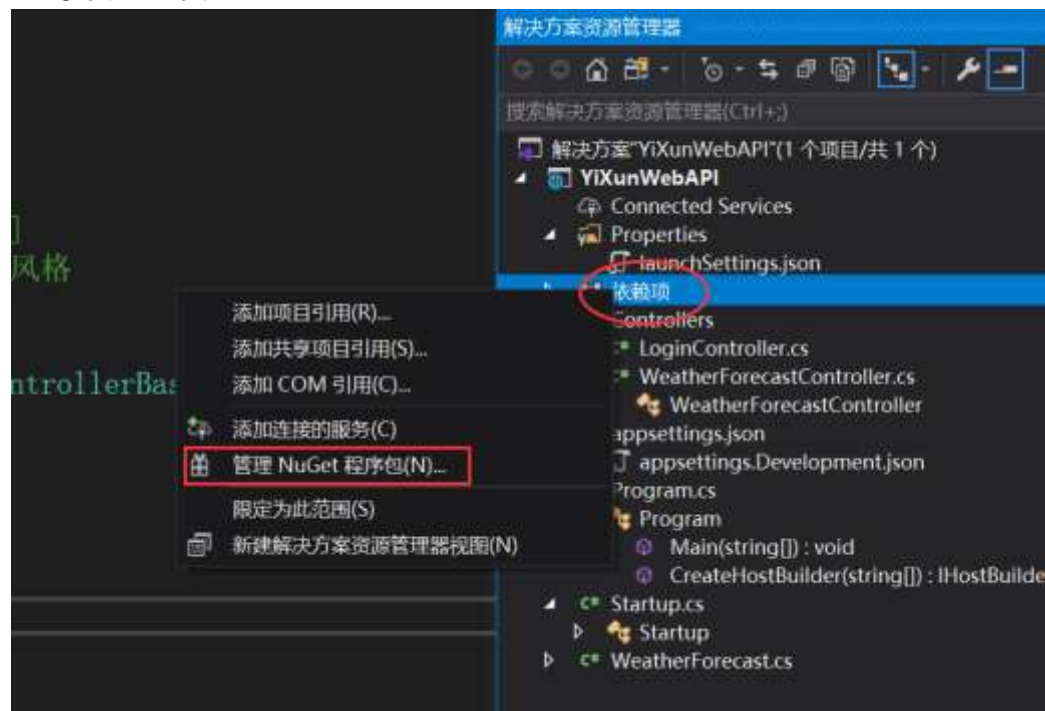
2、ADO和ORM框架及Nuget包管理器的使用（连接数据库前置知识）

一些tips:

- 路由：一条路；
- 路由规则：一条路标；
- 前端找到**API**（**API**：从数据库找到数据并返回）。
- **C#**中可以用**ADO/ORM**框架从数据库拿数据（**O**： **object**； **R**： **relation**）
- 从数据库取数据流程：开冰箱门（打开数据库） → 把大象取出来（取数据） → 关冰箱门（关闭数据库）

打开数据库需要的类：**SqlConnection**（需要**Nuget**包管理器引入，具体方法如下：☺）

资源管理器 → 依赖项 → 管理**NuGet**程序包 → 下载**OracleClient**的包（**B**站教程里下的是**sqlClient**，但我们用的是**Oracle**）



3、打开数据的准备（注意：这一页的内容是**SQL Server** 的代码，事实上只需要把开头的**Sql**换成**Oracle**就**OK**了，详见下一页**PPT**）

经过左边图片上的操作后得到右边图片的结果。再加入一个**conn.Open()**函数打开数据库。

```
namespace YiXunWebAPI.Controllers
{
    //这个文件是我们之前创建的controller文件 (见chapter2 1部分)
    //[Route("~/controller/[action]")]
    [Route("~/controller")] //restful风格
    [ApiController]
    0 个引用
    public class LoginController : ControllerBase
    {
        [HttpGet]
        0 个引用
        public string Get()
        {
            SqlConnection
            return "d!";
        }
    }
}
```

选择

点击

using Microsoft.Data.SqlClient;

Microsoft.Data.SqlClient.SqlConnection
生成属性"LoginController.SqlConnection"
生成字段"LoginController.SqlConnection"
生成只读字段"LoginController.SqlConnection"
生成本地"SqlConnection"
生成参数 "SqlConnection"

CS0103 当前上下文中不存在名称"SqlCo

using Microsoft.AspNetCore.Mvc;
using Microsoft.Data.SqlClient;

```
inWebAPI YiXunWebAPI.Controllers.LoginController
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Threading.Tasks; 这是经过上一步操作后自动加入的
5 using Microsoft.AspNetCore.Http;
6 using Microsoft.AspNetCore.Mvc;
7 using Microsoft.Data.SqlClient;
8
9 namespace YiXunWebAPI.Controllers
10 {
11     //[Route("~/controller/[action]")]
12     [Route("~/controller")] //restful风格
13     [ApiController]
14     0 个引用
15     public class LoginController : ControllerBase
16     {
17         [HttpGet]
18         0 个引用
19         public string Get()
20         {
21             SqlConnection conn = new SqlConnection();
22             return "Good!";
23         }
24     }
25 }
```

SqlConnection变成了一个类并实例化出了一个对象

4、开始从数据库获取数据——SQL语句登场!!!

- 具体的语法（实现相关功能的类）可参考：[C# ADO.NET数据库操作](#)
- 注：这个教程虽然以**Sql Server**为例进行说明，但是**Oracle**类的种类和操作几乎一模一样，只需要把类名从**Sql*****换成**Oracle*****即可

下面通过一个具体实例简单地说明☺（还是在我们新建的**controller**里编写代码）

GET类的示例代码见下页。

```
18 0 个引用
19 public class LoginController : ControllerBase
20 {
21     [HttpGet]
22     0 个引用
23     public string Get()
24     {
25         string connString = 1、打开数据库
26         (@"DATA SOURCE= (DESCRIPTION = (ADDRESS = (PROTOCOL = TCP)(HOST = 8.130.101
27         OracleConnection conn = new OracleConnection(connString);
28         conn.Open(); //打开数据库
29
30         //取数据的类: OracleCommand
31         //“SQL语句” 2、发送取数据的指令 (发送SQL语句)
32         OracleCommand cmd = new OracleCommand("select count('USER_ID') from system.YIX
33
34         //取数据的类: OracleDataAdapter
35         OracleDataAdapter sda = new OracleDataAdapter(cmd);
36
37         //相当于装载数据的容器: DataSet
38         DataSet ds = new DataSet();
39         //将数据装入容器 3、拿到数据, 并把数据装入容器
40         sda.Fill(ds);
41
42         DataTable res = ds.Tables[0];
43         DataRow dr = res.Rows[0];
44         var value = dr["COUNT('USER_ID')"].ToString();
45
46         return value; 4、返回数据
47     }
48 }
```

GET

/Login

Parameters

No parameters

Execute

目前的运行效果, 盲猜是因为我们的数据库里没有数据

Responses

Curl

curl -X 'GET' \
'https://localhost:44351/Login' \
-H 'accept: text/plain'

Request URL

https://localhost:44351/Login

Server response

Code

Details

```
1. public string Get()
2.     {
3.         string connString =
4.             (@"DATA SOURCE= (DESCRIPTION = (ADDRESS = (PROTOCOL = TCP)
5. (HOST = 8.130.101.207)(PORT = 1521))(CONNECT_DATA = (SERVER = DEDICATED)(SERVICE_NAME = yixun)));
6. PASSWORD=Hqiuqiu;USER ID=C##CONNECT ");
7.         OracleConnection conn = new OracleConnection(connString);
8.         conn.Open(); //打开数据库
9.
10.        //取数据的类: OracleCommand
11.        // "SQL语句"
12.        OracleCommand cmd = new OracleCommand("select count('USER_ID') from system.YIXUN_WEB_USER", conn)
13.        ;
14.
15.        //拿数据的类: OracleDataAdapter
16.        OracleDataAdapter sda = new OracleDataAdapter(cmd);
17.
18.        //相当于装载数据的容器: DataSet
19.        DataSet ds = new DataSet();
20.        //将数据装入容器
21.        sda.Fill(ds);
22.        DataTable res = ds.Tables[0];
23.        DataRow dr = res.Rows[0];
24.        var value = dr["COUNT('USER_ID')"].ToString();
25.
26.        return value;
    }
```


5、取出数据后，关闭数据库（关闭冰箱门）

注：事实上教程里还提到了如何编写用户登录的代码（P8）需要时可直接看教程，这里直接跳过。（B站视频

教程链接：[8.用户登录验证_哔哩哔哩_bilibili](#)

关闭数据库有两种方法：

方法一：

```
23 string connString =
24     (@"DATA SOURCE= (DESCRIPTION = (ADDRESS = (PROTOCOL =
25 OracleConnection conn = new OracleConnection(connString)
26 conn.Open(); //打开数据库
27
28 //取数据的类: OracleCommand
29 //SQL语句"
30 OracleCommand cmd = new OracleCommand("select count('USE
31
32 //拿数据的类: OracleDataAdapter
33 OracleDataAdapter sda = new OracleDataAdapter(cmd);
34
35 //相当于装载数据的容器: DataSet
36 DataSet ds = new DataSet();
37 //将数据装入容器
38 sda.Fill(ds);
39 DataTable res = ds.Tables[0];
40 DataRow dr = res.Rows[0];
41
42 //关闭数据库
43 conn.Close();
44 conn.Dispose();
45
46 var value = dr["COUNT('USER_ID')"].ToString();
47
```

新增代码：

取完数据即可关闭数据库

直接在此处加using关键字，conn使用完后自动释放

方法二

```
[HttpGet]
0 个引用
public string Get()
{
    string connString =
        (@"DATA SOURCE= (DESCRIPTION = (ADDRESS = (PROTOCOL = TCP
    using OracleConnection conn = new OracleConnection(connStrin
    conn.Open(); //打开数据库

    //取数据的类: OracleCommand
    //SQL语句"
```

Chapter3:

对数据库的修改

1、插入/更新/删除

插入：**SQL**语句：**insert into**。在之前写入**SQL**语句的地方写入**insert into**语句即可插入数据。参数可通过在方法里声明（和**C**传递参数概念一致）。在**SQL**语句中用“**+<参数名>+**”表示。具体见**B**站视频**P11**。

更新：具体见**B**站视频**P12**。

删除：具体见**B**站视频**P12**。

Chapter4: axios介绍

1、axios简介

Axios是基于**promise**对**ajax**的一种封装。详细介绍，包括代码的编写规范可以查阅：[axios中文文档](#)[|axios中文网](#) | [axios \(axios-js.com\)](#)

个人理解：**axios**是写在前端的代码里的，通过**url**访问后端的项目（目前还没有尝试，此处待完善）

可以使用自定义配置新建一个 axios 实例

`axios.create([config])`

这里就用到了一个url，盲猜是后端项目提供的

```
const instance = axios.create({
  baseURL: 'https://some-domain.com/api/',
  timeout: 1000,
  headers: {'X-Custom-Header': 'foobar'}
});
```

