朝夕通用管理平台敏捷开发--.NET7后台

大家好， 我是Richard老师，欢迎来到朝夕课堂。 朝夕教育一直致力于给小伙伴儿们输出更多的编程技术干货。从这个视频开始Richard老师给大家准备了一个敏捷开发通用后台开发。 从这个视频开始老师会从零开始给大家搭建框架， 做业务分析和开发.

前后端分离完成的；

1、环境准备

> VS2022、 IIS、.NET7
>
> VSCode、NodeJs、ElementPlus
>
> SqlServer数据库

2、后端服务搭建

2.1 创建项目---解决方案（Zhaoxi.AgiletyFramework.PortalProject）

2.2 CoreWebapi（Zhaoxi.AgiletyFramework.WebApi）

3、配置Log4net

3.1 nuget引入：

```
log4net
Microsoft.Extensions.Logging.Log4Net.AspNetCore
System.Data.SqlClient    //数据库操作组件
```

3.2 准备配置文件--在CfgFile/log4net.Config内容

涵盖了文本日志和数据库写日志、其中写数据库日志的链接字符串--配置为使用的数据库地址

```
<?xml version="1.0" encoding="utf-8"?>
<log4net>
    <!-- Define some output appenders -->
    <appender name="rollingAppender" type="log4net.Appender.RollingFileAppender">
        <file value="log4\log.txt" />
        <!--追加日志内容-->
        <appendToFile value="true" />

        <!--防止多线程时不能写Log,官方说线程非安全-->
        <lockingModel type="log4net.Appender.FileAppender+MinimalLock" />

        <!--可以为:Once|Size|Date|Composite-->
        <!--Composite为Size和Date的组合-->
        <rollingStyle value="Composite" />

        <!--当备份文件时,为文件名加的后缀-->
        <datePattern value="yyyyMMdd.TXT" />

        <!--日志最大个数,都是最新的-->
        <!--rollingStyle节点为Size时,只能有value个日志-->
        <!--rollingStyle节点为Composite时,每天有value个日志-->
        <maxSizeRollBackups value="20" />

        <!--可用的单位:KB|MB|GB-->
        <maximumFileSize value="3MB" />

        <!--置为true,当前最新日志文件名永远为file节中的名字-->
        <staticLogFileName value="true" />

        <!--输出级别在INFO和ERROR之间的日志-->
        <filter type="log4net.Filter.LevelRangeFilter">
            <param name="LevelMin" value="ALL" />
            <param name="LevelMax" value="FATAL" />
        </filter>
        <layout type="log4net.Layout.PatternLayout">
            <conversionPattern value="%date [%thread] %-5level %logger - %message%newline"/>
        </layout>

        <!--<layout type="Zhaoxi.Manage.MentApi.Utility.Log4netExt.CustomLogLayout">
            <conversionPattern value="%date [%thread] %-5level %logger - %message%newline"/>
        </layout>-->


    </appender>

    <!--SqlServer形式-->
    <!--log4net日志配置：http://logging.apache.org/log4net/release/config-examples.html -->
    <appender name="AdoNetAppender_SqlServer" type="log4net.Appender.AdoNetAppender">
        <!--日志缓存写入条数 设置为0时只要有一条就立刻写到数据库-->
        <bufferSize value="0" />
        <connectionType value="System.Data.SqlClient.SqlConnection,System.Data.SqlClient,
Version=4.6.1.3, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
```

```
        <connectionString value="Data Source=PC-202206030027;Initial Catalog=Zhaoxi.SmartFactory;User ID=sa;Password=sa123;MultipleActiveResultSets=true" />
        <commandText value="INSERT INTO SystemLog ([Date],[Thread],[Level],[Logger],[Message],[Exception]) VALUES (@log_date, @thread, @log_level, @logger, @message, @exception)" />
        <parameter>
            <parameterName value="@log_date" />
            <dbType value="DateTime" />
            <layout type="log4net.Layout.RawTimeStampLayout" />
        </parameter>
        <parameter>
            <parameterName value="@thread" />
            <dbType value="String" />
            <size value="255" />
            <layout type="log4net.Layout.PatternLayout">
                <conversionPattern value="%thread" />
            </layout>
        </parameter>
        <parameter>
            <parameterName value="@log_level" />
            <dbType value="String" />
            <size value="50" />
            <layout type="log4net.Layout.PatternLayout">
                <conversionPattern value="%level" />
            </layout>
        </parameter>
        <parameter>
            <parameterName value="@logger" />
            <dbType value="String" />
            <size value="255" />
            <layout type="log4net.Layout.PatternLayout">
                <conversionPattern value="%logger" />
            </layout>
        </parameter>
        <parameter>
            <parameterName value="@message" />
            <dbType value="String" />
            <size value="4000" />
            <layout type="log4net.Layout.PatternLayout">
                <conversionPattern value="%message" />
            </layout>
        </parameter>
        <parameter>
            <parameterName value="@exception" />
            <dbType value="String" />
            <size value="2000" />
            <layout type="log4net.Layout.ExceptionLayout" />
        </parameter>
    </appender>

    <root>

        <!--控制级别，由低到高: ALL|DEBUG|INFO|WARN|ERROR|FATAL|OFF-->
        <!--OFF:0-->
        <!--FATAL:FATAL-->
        <!--ERROR: ERROR,FATAL-->
        <!--WARN: WARN,ERROR,FATAL-->
        <!--INFO: INFO,WARN,ERROR,FATAL-->
        <!--DEBUG: INFO,WARN,ERROR,FATAL-->
```

```
        <!--ALL: DEBUG,INFO,WARN,ERROR,FATAL-->
        <priority value="ALL"/>

        <level value="INFO"/>
        <appender-ref ref="rollingAppender" />
        <appender-ref ref="AdoNetAppender_SqlServer" />
    </root>
</log4net>
```

> **注意：**
>
> 上面配置的节点中：<appender name="rollingAppender" 为写文本日志的
>
> 上面配置的节点中：<appender name="AdoNetAppender_SqlServer" 为写数据库日志的
>
> appender-ref ref= 节点决定了使用什么方式写入日志；
>
> 如果配置了： 表示写文本日志生效
>
> 如果配置了： 表示写数据库日志生效

写入数据库日志的表脚本

```
/****** Object:  Table [dbo].[SystemLog]     Script Date: 2023/7/12 10:01:30 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[SystemLog](
    [Id] [int] IDENTITY(1,1) NOT NULL,
    [Date] [datetime] NOT NULL,
    [Thread] [varchar](255) NOT NULL,
    [Level] [varchar](50) NOT NULL,
    [Logger] [varchar](255) NOT NULL,
    [Message] [varchar](4000) NOT NULL,
    [Exception] [varchar](2000) NULL,
 CONSTRAINT [PK_SystemLog] PRIMARY KEY CLUSTERED
(
    [Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

3.3 在Main方法中加入

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

builder.Logging.AddLog4Net("CfgFile/log4net.Config");     //加入的内容

var app = builder.Build();
```

3.4 在控制器中注入使用

```
 private readonly ILogger<TestController> _logger;

          /// <summary>
          /// 构造函数
          /// </summary>
          /// <param name="logger"></param>
public TestController(ILogger<TestController> logger)
{
     _logger = logger;
    logger.LogInformation($"{this.GetType()} 被构造....");
}
```

4、Swagger配置

创建项目初期，勾选“启用OpenApi支持”默认生成

```
app.UseSwagger();
app.UseSwaggerUI();

-------------------------------

services.AddEndpointsApiExplorer();
services.AddSwaggerGen() ;
```

4.1、支持注释

4.1.1、选择Corewebapi----属性---应用程序---勾选引用程序集，生成包含项目公共API的引用程序集

```
 services.AddSwaggerGen(option =>
    {
         // xml文档绝对路径
         var file = Path.Combine(AppContext.BaseDirectory,"Zhaoxi.AgiletyFramework.WebApi.xml");

                     // true : 显示控制器层注释
                     option.IncludeXmlComments(file, true);
                     // 对action的名称进行排序，如果有多个，就可以看见效果了。
                     option.OrderActionsBy(o => o.RelativePath);
       });
```

4.2、支持版本控制

4.2.1、添加版本枚举类

```
/// <summary>
    /// Api版本枚举
    /// </summary>
    public enum ApiVersions
    {
        V1,
        V2,
        V3,
        V4
    }
```

4.2.2、枚举生效

配置AddSwaggerGen

```
  services.AddSwaggerGen(option =>
                {
                    foreach (var version in typeof(ApiVersions).GetEnumNames())
                    {
                        option.SwaggerDoc(version, new OpenApiInfo()
                        {
                            Title = $"朝夕敏捷后台管理项目实战Api文档",
                            Version = version,
                            Description = $"通用版本的CoreApi版本v1"
                        });
                    }
  });
```

4.2.3、配置UseSwaggerUI

```
  app.UseSwaggerUI(option =>
                {
                    foreach (string version in typeof(ApiVersions).GetEnumNames())
                    {
                        option.SwaggerEndpoint($"/swagger/{version}/swagger.json", $"朝夕敏捷后台
管理项目实战Api文档【{version}】版本");
                    }
                });
```

4.2.4、哪个控制器中的Api要归属于哪个版本，就在控制器上标记特性

ApiExplorerSettings(IgnoreApi = false, GroupName = nameof(ApiVersions.V1))]

```
[ApiExplorerSettings(IgnoreApi = false, GroupName = nameof(ApiVersions.V1))]
    public class TestController : ControllerBase
    {
        private readonly ILogger<TestController> _logger;
        private readonly IUserService _IUserService;
        /// <summary>
        /// 构造函数
        /// </summary>
        /// <param name="logger"></param>
        public TestController(ILogger<TestController> logger, IUserService iUserService)
        {
            _logger = logger;
            logger.LogInformation($"{this.GetType()} 被构造....");
            _IUserService = iUserService;
        }
 }
```

4.3、支持Beare授权Token传递

4.3.1、配置AddSwaggerGen 支持Token

```
services.AddSwaggerGen(option =>
                {
                      //添加安全定义--配置支持token授权机制
                      option.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme
                      {
                            Description = "请输入token,格式为 Bearer xxxxxxxx（注意中间必须有空格）",
                            Name = "Authorization",
                            In = ParameterLocation.Header,
                            Type = SecuritySchemeType.ApiKey,
                            BearerFormat = "JWT",
                            Scheme = "Bearer"
                      });
                      //添加安全要求
                      option.AddSecurityRequirement(new OpenApiSecurityRequirement
                      {
                         {
                                  new OpenApiSecurityScheme
                                  {
                                        Reference =new OpenApiReference()
                                        {
                                              Type = ReferenceType.SecurityScheme,
                                              Id ="Bearer"
                                        }
                                  },
                                  new string[]{ }
                            }
                      });
                });
```

5、支持跨域请求

5.1、配置跨域扩展方法

```
public static void AddCorsExt(this IServiceCollection services)
          {
                //中间件解决跨域问题
                services.AddCors(options =>
                {
                      // allcore: 策略名称
                      options.AddPolicy("allcore", corsBuilder =>
                      {
                            corsBuilder.AllowAnyHeader()
                                       .AllowAnyOrigin()
                                       .AllowAnyMethod();
                      });
                });
          }
```

5.2、跨域生效

```
    app.UseCors("allcore");
```

> 注意：allcore为上面配置的支持所有的api跨域策略

6、配置支持EFCore

6.1 nuget引入

```
Microsoft.EntityFrameworkCore                          //EFCore核心应用程序
Microsoft.EntityFrameworkCore.SqlServer               //SqlServer数据库驱动程序
```

6.2 增加数据库访问层

Zhaoxi.AgiletyFramework.DbModels

增加实体~

```
    public class UserEntity
     {
           public int Id { get; set; }
           public string? Name { get; set; }
           public int Age { get; set; }
     }
```

6.3、准备DbContext

---EFCore操作数据库的核心

```
public class AgiletyDbContext : DbContext
     {
           public AgiletyDbContext()
           {
           }

           public AgiletyDbContext(DbContextOptions<AgiletyDbContext> options)
                 : base(options)
           {
           }

           public virtual DbSet<UserEntity> BookEntities { get; set; }

           /// <summary>
           /// 配置DbContext需要的参数---例如  数据库连接字符串
           /// </summary>
           /// <param name="optionsBuilder"></param>
           protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
           {
                 if (!optionsBuilder.IsConfigured)
```

```
                {
                      optionsBuilder.UseSqlServer("Data Source=PC-202206030027;Initial
Catalog=Zhaoxi.AgiletyFramework.DB;User ID=sa;Password=sa123;TrustServerCertificate=true");
                }
          }

          /// <summary>
          /// 配置数据库表映射关系
          /// </summary>
          /// <param name="modelBuilder"></param>
          protected override void OnModelCreating(ModelBuilder modelBuilder)
          {
                modelBuilder.Entity<UserEntity>(entity =>
                {
                      entity.ToTable("UserEntity");
                });
          }
    }
```

6.4、生成数据库

6.4.1、根据数据库连接字符串生成

```
Data Source=PC-202206030027;Initial Catalog=Zhaoxi.AgiletyFramework.DB;User
ID=sa;Password=sa123;TrustServerCertificate=true
```

6.4.2、生成数据库

6.4.2.1、新建控制台：Zhaoxi.AgiletyFramework.InitDatabase

6.4.2.2、引入Zhaoxi.AgiletyFramework.DbModels

```
static void Main(string[] args)
{
    try
    {
      using (AgiletyDbContext context=new AgiletyDbContext())
      {
         context.Database.EnsureDeleted();//根据数据库连接字符串的配置删除数据库，如果不存在就不操作
         context.Database.EnsureCreated(); //根据数据库连接字符串的配置创建数据库，如果存在就不创建
        }
     }
     catch (Exception)
     {
               throw;
     }
}
```

数据库生成 Zhaoxi.AgiletyFramework.DB~

6.5、EFCore构建读写分离

详细见：

7、构建业务逻辑层

7.1、为支持IOC容器，这创建的业务逻辑层有抽象层有具体实现层

7.2、抽象层： Zhaoxi.AgiletyFramework.IBusinessServices

7.3、具体实现层：Zhaoxi.AgiletyFramework.BusinessServices

7.4、创建基础接口（通用接口） IBaseService， 增加分页实体：PagingData

```
public interface IBaseService
    {

        #region Query
        /// <summary>
        /// 根据id查询实体
        /// </summary>
        /// <param name="id"></param>
        /// <returns></returns>
        T Find<T>(int id) where T : class;

        /// <summary>
        /// 提供对单表的查询
        /// </summary>
        /// <returns>IQueryable类型集合</returns>
        [Obsolete("尽量避免使用，using 带表达式目录树的 代替")]
        IQueryable<T> Set<T>() where T : class;

        /// <summary>
        /// 查询
        /// </summary>
        /// <typeparam name="T"></typeparam>
        /// <param name="funcWhere"></param>
        /// <returns></returns>
        IQueryable<T> Query<T>(Expression<Func<T, bool>> funcWhere) where T : class;

        /// <summary>
        /// 分页查询
        /// </summary>
        /// <typeparam name="T"></typeparam>
        /// <typeparam name="S"></typeparam>
        /// <param name="funcWhere"></param>
        /// <param name="pageSize"></param>
        /// <param name="pageIndex"></param>
        /// <param name="funcOrderby"></param>
        /// <param name="isAsc"></param>
        /// <returns></returns>
        PagingData<T> QueryPage<T, S>(Expression<Func<T, bool>> funcWhere, int pageSize, int
pageIndex, Expression<Func<T, S>> funcOrderby, bool isAsc = true) where T : class;
        #endregion

        #region Add
        /// <summary>
        /// 新增数据，即时Commit
        /// </summary>
        /// <param name="t"></param>
        /// <returns>返回带主键的实体</returns>
        T Insert<T>(T t) where T : class;
```

```
        /// <summary>
        /// 新增数据，即时Commit
        /// 多条sql 一个连接，事务插入
        /// </summary>
        /// <param name="tList"></param>
        IEnumerable<T> Insert<T>(IEnumerable<T> tList) where T : class;
        #endregion

        #region Update
        /// <summary>
        /// 更新数据，即时Commit
        /// </summary>
        /// <param name="t"></param>
        void Update<T>(T t) where T : class;

        /// <summary>
        /// 更新数据，即时Commit
        /// </summary>
        /// <param name="tList"></param>
        void Update<T>(IEnumerable<T> tList) where T : class;
        #endregion

        #region Delete
        /// <summary>
        /// 根据主键删除数据，即时Commit
        /// </summary>
        /// <param name="t"></param>
        void Delete<T>(int Id) where T : class;

        /// <su+mary>
        /// 删除数据，即时Commit
        /// </summary>
        /// <param name="t"></param>
        void Delete<T>(T t) where T : class;

        /// <summary>
        /// 删除数据，即时Commit
        /// </summary>
        /// <param name="tList"></param>
        void Delete<T>(IEnumerable<T> tList) where T : class;
        #endregion

        #region Other
        /// <summary>
        /// 立即保存全部修改
        /// 把增/删的savechange给放到这里，是为了保证事务的
        /// </summary>
        void Commit();

        /// <summary>
        /// 执行sql 返回集合
        /// </summary>
        /// <param name="sql"></param>
        /// <param name="parameters"></param>
        /// <returns></returns>
        IQueryable<T> ExcuteQuery<T>(string sql, SqlParameter[] parameters) where T : class;
```

```
        /// <summary>
        /// 执行sql，无返回
        /// </summary>
        /// <param name="sql"></param>
        /// <param name="parameters"></param>
        void Excute<T>(string sql, SqlParameter[] parameters) where T : class;

        #endregion
    }
}
```

7.6、实现基础接口IBaseService

```
public abstract class BaseService : IBaseService
{
        protected DbContext Context { get; set; }

        public BaseService(DbContext context)
        {
            Context = context;
        }

        #region Query
        public T Find<T>(int id) where T : class
        {
            return this.Context.Set<T>().Find(id);
        }

        /// <summary>
        ///  不应该暴露给上端使用者，尽量少用
        /// </summary>
        /// <typeparam name="T"></typeparam>
        /// <returns></returns>
        //[Obsolete("尽量避免使用，using 带表达式目录树的代替")]
        public IQueryable<T> Set<T>() where T : class
        {
            return Context.Set<T>();
        }

        /// <summary>
        /// 这才是合理的做法，上端给条件，这里查询
        /// </summary>
        /// <typeparam name="T"></typeparam>
        /// <param name="funcWhere"></param>
        /// <returns></returns>
        public IQueryable<T> Query<T>(Expression<Func<T, bool>> funcWhere) where T : class
        {
            return Context.Set<T>().Where(funcWhere);
        }

        /// <summary>
        /// 分页查询
        /// </summary>
        /// <typeparam name="T"></typeparam>
        /// <typeparam name="S"></typeparam>
        /// <param name="funcWhere"></param>
```

```
        /// <param name="pageSize"></param>
        /// <param name="pageIndex"></param>
        /// <param name="funcOrderby"></param>
        /// <param name="isAsc"></param>
        /// <returns></returns>
        public PagingData<T> QueryPage<T, S>(Expression<Func<T, bool>> funcWhere, int pageSize, int
pageIndex, Expression<Func<T, S>> funcOrderby, bool isAsc = true) where T : class
        {

            var list = Set<T>();
            if (funcWhere != null)
            {
                list = list.Where(funcWhere);
            }
            if (isAsc)
            {
                list = list.OrderBy(funcOrderby);
            }
            else
            {
                list = list.OrderByDescending(funcOrderby);
            }
            PagingData<T> result = new PagingData<T>()
            {
                DataList = list.Skip((pageIndex - 1) * pageSize).Take(pageSize).ToList(),
                PageIndex = pageIndex,
                PageSize = pageSize,
                RecordCount = list.Count()
            };
            return result;
        }
        #endregion

        #region Insert
        /// <summary>
        /// 即使保存  不需要再Commit
        /// </summary>
        /// <typeparam name="T"></typeparam>
        /// <param name="t"></param>
        /// <returns></returns>
        public T Insert<T>(T t) where T : class
        {
            Context.Set<T>().Add(t);
            Commit();//写在这里  就不需要单独commit  不写就需要
            return t;
        }

        public IEnumerable<T> Insert<T>(IEnumerable<T> tList) where T : class
        {
            Context.Set<T>().AddRange(tList);
            Commit();//一个链接  多个sql
            return tList;
        }
        #endregion

        #region Update
        /// <summary>
        /// 是没有实现查询，直接更新的,需要Attach和State
```

```
        ///
        /// 如果是已经在context，只能再封装一个(在具体的service)
        /// </summary>
        /// <typeparam name="T"></typeparam>
        /// <param name="t"></param>
        public void Update<T>(T t) where T : class
        {

              if (t == null) throw new Exception("t is null");

              Context.Set<T>().Attach(t);//将数据附加到上下文，支持实体修改和新实体，重置为UnChanged
              Context.Entry(t).State = EntityState.Modified;
              Commit();//保存 然后重置为UnChanged
        }

        public void Update<T>(IEnumerable<T> tList) where T : class
        {

              foreach (var t in tList)
              {
                    Context.Set<T>().Attach(t);
                    Context.Entry(t).State = EntityState.Modified;
              }
              Commit();
        }

        #endregion

        #region Delete
        /// <summary>
        /// 先附加 再删除
        /// </summary>
        /// <typeparam name="T"></typeparam>
        /// <param name="t"></param>
        public void Delete<T>(T t) where T : class
        {

              if (t == null) throw new Exception("t is null");
              Context.Set<T>().Attach(t);
              Context.Set<T>().Remove(t);
              Commit();
        }

        /// <summary>
        /// 还可以增加非即时commit版本的，
        /// 做成protected
        /// </summary>
        /// <typeparam name="T"></typeparam>
        /// <param name="Id"></param>
        public void Delete<T>(int Id) where T : class
        {

              T t = Find<T>(Id);//也可以附加
              if (t == null) throw new Exception("t is null");
              Context.Set<T>().Remove(t);
              Commit();
        }
```

```
        public void Delete<T>(IEnumerable<T> tList) where T : class
        {

            foreach (var t in tList)
            {
                Context.Set<T>().Attach(t);
            }
            Context.Set<T>().RemoveRange(tList);
            Commit();
        }
        #endregion

        #region Other
        public void Commit()
        {
            Context.SaveChanges(); //EFCore中对于增删改 ,必须要执行这句话才能生效
        }

        public IQueryable<T> ExcuteQuery<T>(string sql, SqlParameter[] parameters) where T : class
        {
            return this.Context.Set<T>().FromSqlRaw(sql, parameters);
        }

        /// <summary>
        /// 执行Sql语句，返回实体对象
        /// </summary>
        /// <typeparam name="T"></typeparam>
        /// <param name="sql"></param>
        /// <param name="parameters"></param>
        public void Excute<T>(string sql, SqlParameter[] parameters) where T : class
        {
            IDbContextTransaction trans = null;
            try
            {
                trans = Context.Database.BeginTransaction();
                this.Context.Database.ExecuteSqlRaw(sql, parameters);
                trans.Commit();
            }
            catch (Exception)
            {
                if (trans != null)
                    trans.Rollback();
                throw;
            }
        }

        /// <summary>
        /// 释放回收
        /// </summary>
        public virtual void Dispose()
        {
            if (Context != null)
            {
                Context.Dispose();
            }
        }
        #endregion
    }
```

7.7、根据需求创建不同业务的Service

案例：UserService 和 IUserService

```
public class UserService : BaseService, IUserService
{
    public UserService(DbContext context) : base(context)
    {
    }
}
```

```
public interface UserService : BaseService
{
}
```

8、IOC容器支持测试

8.1、注册抽象和具体之间的关系

```
#注册EFCore--配置数据库连接字符串
 builder.Services.AddDbContext<DbContext,AgiletyDbContext>(option =>
                {
                      option.UseSqlServer("Data Source=PC-202206030027;Initial
Catalog=Zhaoxi.AgiletyFramework.DB;User ID=sa;Password=sa123;TrustServerCertificate=true");
                });

#注册服务层
builder.Services.AddTransient<IUserService, UserService>();
```

8.2、控制期构造函数注入

```
public class TestController : ControllerBase
    {
          private readonly ILogger<TestController> _logger;
          private readonly IUserService _IUserService;
          /// <summary>
          /// 构造函数
          /// </summary>
          /// <param name="logger"></param>
          public TestController(ILogger<TestController> logger, IUserService iUserService)
          {
                _logger = logger;
                logger.LogInformation($"{this.GetType()} 被构造....");
                _IUserService = iUserService;
          }

          /// <summary>
          /// 连接业务逻辑层实现数据增加和查询
          /// </summary>
          /// <returns></returns>
          [HttpGet]
          [Route("GetUser")]
          public UserEntity GetUser()
```

```
            {
                  UserEntity user = new UserEntity()
                  {
                        Name = "测试数据",
                        Age = 30
                  };
                  _IUserService.Insert<UserEntity>(user);
                  return _IUserService.Query<UserEntity>(c => true).OrderByDescending(c =>
c.Id).First();
            }
}
```

9、Autofac支持

9.1、nuget引入：

```
Autofac
Autofac.Extensions.DependencyInjection     #ASP.NET Core扩展程序
```

9.2、配置生效整合到Core Webapi

```
//通过工厂替换，把Autofac整合进来
builder.Host.UseServiceProviderFactory(new AutofacServiceProviderFactory());
builder.Host.ConfigureContainer<ContainerBuilder>(containerBuilder =>
{
    containerBuilder.RegisterType<UserService>().As<IUserService>()
     .EnableInterfaceInterceptors(new ProxyGenerationOptions() //扩展aop
       {
           //Selector = new CustomInterceptorSelector()
       });

       //在这里 注册用户Servcie
       containerBuilder.RegisterType<IUserService>().As<UserService>();

       //注册每个控制器和抽象之间的关系
       var controllerBaseType = typeof(ControllerBase);
        containerBuilder.RegisterAssemblyTypes(typeof(Program).Assembly)
        .Where(t => controllerBaseType.IsAssignableFrom(t) && t != controllerBaseType)
       .PropertiesAutowired(new CusotmPropertySelector()); //支持属性注入

});
```

9.3、控制器生效

```
public class TestController : ControllerBase
     {
           private readonly ILogger<TestController> _logger;
           private readonly IUserService _IUserService;
           private readonly IUserService _IUserServiceNew;
           /// <summary>
           /// 构造函数
           /// </summary>
```

```
        /// <param name="logger"></param>
        public TestController(ILogger<TestController> logger, IUserService
iUserService,Autofac.IComponentContext componentContext)
        {
            _logger = logger;
            logger.LogInformation($"{this.GetType()} 被构造....");
            _IUserService = iUserService;
            _IUserServiceNew = componentContext.Resolve<IUserService>();
        }

        /// <summary>
        /// 连接业务逻辑层实现数据增加和查询
        /// </summary>
        /// <returns></returns>
        [HttpGet]
        [Route("GetUser")]
        public UserEntity GetUser()
        {
            UserEntity user = new UserEntity()
            {
                Name = "测试数据",
                Age = 30
            };
            _IUserService.Insert<UserEntity>(user);
            return _IUserServiceNew.Query<UserEntity>(c => true).OrderByDescending(c =>
c.Id).First();
        }
```

9.4、Autofac扩展AOP

面向切面编程---历史代码固定不变的情况下，可以在xx方法前执行点业务逻辑，也可以在xx方法后，执行点业务逻辑；

两种方式：

1、通过接口实现AOP扩展，扩展后，实现该接口的所有方法都会支持AOP扩展功能

2、通过类实现AOP扩展、扩展后，定义为虚方法的实现方法支持AOP扩展

二者比较：通过类实现更加灵活。可以选择性支持AOP

9.4.1、nuget引入：

```
Castle.Core
```

9.4.1、扩展AOP写日志~

```
public class CustomLogInterceptor : IInterceptor
{
        //支持依赖注入
          private readonly ILogger<CustomLogInterceptor> _Logger;
          public CustomLogInterceptor(ILogger<CustomLogInterceptor> logger)
          {
              _Logger= logger;
          }
```

```
        public void Intercept(IInvocation invocation)
        {
            _Logger.LogInformation($"=================={invocation.Method.Name} 执行前
~~========================");
            _Logger.LogInformation($"=================={invocation.Method.Name} 执行前
~~========================");
            _Logger.LogInformation($"=================={invocation.Method.Name} 执行前
~~========================");

            invocation.Proceed();// 开始去执行目标方法

            _Logger.LogInformation($"=================={{{invocation.Method.Name} 执行后
~~========================");
            _Logger.LogInformation($"=================={{{invocation.Method.Name} 执行后
~~========================");
            _Logger.LogInformation($"=================={{{invocation.Method.Name} 执行后
~~========================");
        }
}
```

9.4.2、CustomLogInterceptor通过接口生效

在IOC容器注册服务时生效

```
  public static void AutofacRegister(this ConfigureHostBuilder host)
        {
            host.UseServiceProviderFactory(new AutofacServiceProviderFactory());//通过工厂替换，把
Autofac整合进来
            host.ConfigureContainer<ContainerBuilder>(containerBuilder =>
            {
                containerBuilder.RegisterType<UserService>().As<IUserService>()
                .EnableInterfaceInterceptors(); //通过接口支持AOP扩展

                  //注册AOP扩展  注意这里一定要注册
                containerBuilder.RegisterType<CustomLogInterceptor>();
            });

        }
```

9.4.3、标记接口上生效

```
[Intercept(typeof(CustomLogInterceptor))] //当前接口下的所有的方法都生效
public interface IUserService : IBaseService
{
}
```

测试~~~

9.4.4、CustomLogInterceptor通过类生效

```
 public static void AutofacRegister(this ConfigureHostBuilder host)
        {
            host.UseServiceProviderFactory(new AutofacServiceProviderFactory());//通过工厂替换，把
Autofac整合进来
            host.ConfigureContainer<ContainerBuilder>(containerBuilder =>
            {
                containerBuilder.RegisterType<UserService>().As<IUserService>()
                //.EnableInterfaceInterceptors(); //通过接口支持AOP扩展
                .EnableClassInterceptors();          //通过类支持AOP扩展
```

```
                    //注册AOP扩展  注意这里一定要注册
                    containerBuilder.RegisterType<CustomLogInterceptor>();

                    //注册每个控制器和抽象之间的关系
                    var controllerBaseType = typeof(ControllerBase);
                    containerBuilder.RegisterAssemblyTypes(typeof(Program).Assembly)
                          .Where(t => controllerBaseType.IsAssignableFrom(t) && t !=
controllerBaseType)
                          .PropertiesAutowired(); //支持属性注入
                });
          }
```

9.4.5、类中虚方法AOP支持

```
[Intercept(typeof(CustomLogInterceptor))] //当前类下面的方法，只有标记为virtual方法方可支持AOP
public class UserService : BaseService, IUserService
{
          public UserService(DbContext context) : base(context)
          {
          }

          /// <summary>
          /// 标记为标记为virtual方法方可支持AOP
          /// </summary>
          public virtual void Show1()
          {

          }

          //没有标记为virtual方法，不可支持AOP
          public void Show2()
          {

          }
 }
```

10、Api返回统一格式

定义通用的返回结果类

```
/// <summary>
     /// Api  JSON通用返回格式
     /// </summary>
     public class ApiResult
     {
          /// <summary>
          /// 是否正常返回
          /// </summary>
          public bool Success { get; set; }

          /// <summary>
          /// 处理消息
          /// </summary>
          public string? Message { get; set; }
     }
```

```
public class ApiDataResult<T> : ApiResult
{
	/// <summary>
	/// 结果集
	/// </summary>
	public T? Data { get; set; }

	/// <summary>
	/// 冗余结果
	/// </summary>
	public object? OValue { get; set; }
}
```

注意：

在Api返回的时候，统一通过ApiResult 或者是ApiDataResult包装一层

```
[HttpGet]
		[Route("GetUserApiResult")]
		public IActionResult GetUser()
		{
			UserEntity user = new UserEntity()
			{
				Name = "测试数据",
				Age = 30
			};
			var data = _IUserServiceNew.Query<UserEntity>(c => true).OrderByDescending(c =>
c.Id).First();
			return new JsonResult(new ApiDataResult<UserEntity>() { Data=
data,Success=true,Message="获取user数据" });
		}
```

11、Automapper支持

在结果返回的时候，一般不使用和数据库表对应的实体返回，会定义一个Dto实体用作上端的使用。这里就需要一个实体之间的转换

创建Dto类库---Zhaoxi.AgiletyFramework.ModelDto

Automapper首选

10.1、nuget引入：

```
AutoMapper
AutoMapper.Extensions.Microsoft.DependencyInjection
```

10.2 配置映射规则

```
public class AutoMapConfig : Profile
{
```

```
    //构造函数
    public AutoMapConfig()
    {
                //映射规则
          CreateMap<UserEntity, UserDto>()                          //从UserEntity 映射转换到 UserDto
             .ForMember(c => c.Id, s => s.MapFrom(x => x.Id))     //id映射到id
             .ForMember(c => c.Name, s => s.MapFrom(x => x.Name)) //Name映射到Name
             .ReverseMap();                                         //可以相互转换

          //规则就在这里定义

    }
}
```

10.3、规则生效

进入api控制台中配置~

```
 ////引入Autommaper
 builder.Services.AddAutoMapper(typeof(AutoMapConfig));
```

10.4、依赖注入使用

```
 private readonly IMapper _IMapper;    //AutoMapper映射使用
          public TestController(ILogger<TestController> logger, IUserService iUserService,
Autofac.IComponentContext componentContext, IMapper iMapper)
          {
                _logger = logger;
                logger.LogInformation($"{this.GetType()} 被构造....");
                _IUserService = iUserService;
                _IUserServiceNew = componentContext.Resolve<IUserService>();
                _IMapper = iMapper;
          }
          [HttpGet]
          [Route("GetUserDto")]
          public UserDto GetUserDto()
          {
                UserEntity user = new UserEntity()
                {
                      Name = "测试数据",
                      Age = 30
                };
                _IUserService.Insert<UserEntity>(user);
                var data = _IUserServiceNew.Query<UserEntity>(c => true).OrderByDescending(c =>
c.Id).First();
                var result = _IMapper.Map<UserEntity, UserDto>(data);
                return result;
          }
```

12、运行起来

12.1、运行计划

> **默认数据库地址：** PC-202206030027

> **数据库连接字符串：** Data Source=PC-202206030027;Initial Catalog=Zhaoxi.AgiletyFramework.DB;Persist Security Info=True;User ID=sa;Password=sa123;TrustServerCertificate=true
>
> **本地运行：** 认证授权服务器地址： **http://localhost:5726**
>
> **本地运行：** Webapi服务器地址： **http://localhost:7200**
>
> **本地运行：** 前端服务器地址： **http://localhost:5177**

12.2 配置修改后端项目数据库连接字符串

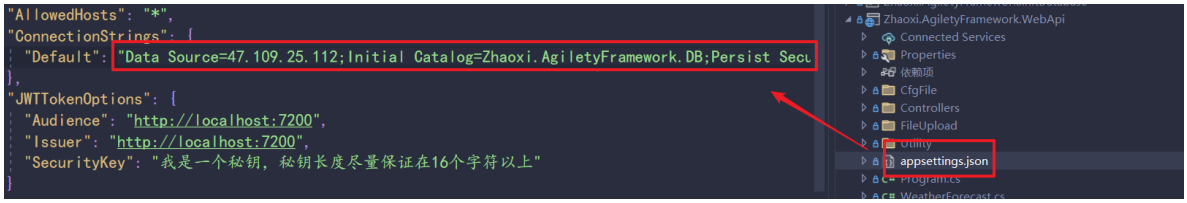
按照以上运行计划配置，共计有四处需要修改字符串的:

> **注意：这里是使用的互联网云服务器数据库，如果要使用本地数据请更改字符串为：**
>
> Data Source=本地计算机名称或者本地IP;Initial Catalog=Zhaoxi.AgiletyFramework.DB;Persist Security Info=True;User ID=sa;Password=本地数据库sa账号对应密码;TrustServerCertificate=true

12.2.1、Zhaoxi.AgiletyFramework.WebApi 项目配置文件修改字符串为，如下图

这里配置的时云服务器数据库连接字符串，如果是本地，请按照上面规则修改后，写入本地的数据库连接字符串

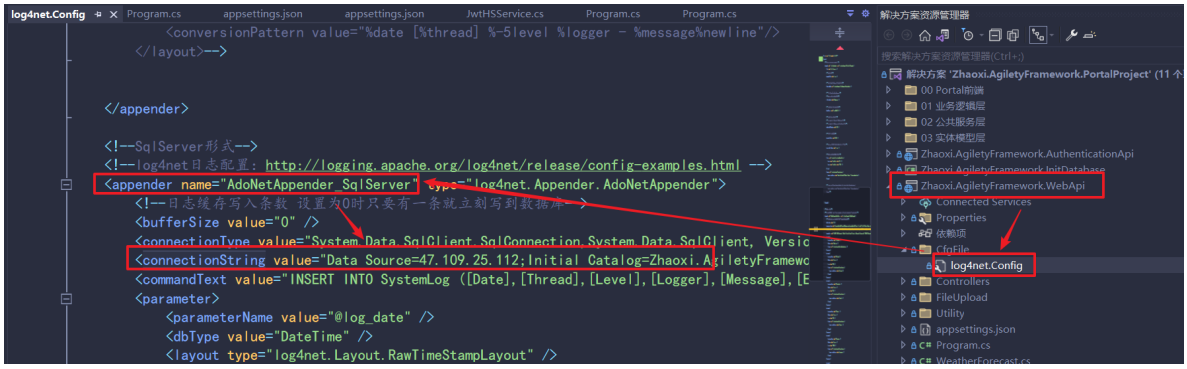
Data Source=PC-202206030027;Initial Catalog=Zhaoxi.AgiletyFramework.DB;Persist Security Info=True;User ID=sa;Password=ZHAOxi@2019;TrustServerCertificate=true



12.2.2、Zhaoxi.AgiletyFramework.WebApi项目中，log4net.Config配置文件中，connectionString节点数据

这里配置的时云服务器数据库连接字符串，如果是本地，请按照上面规则修改后，写入本地的数据库连接字符串

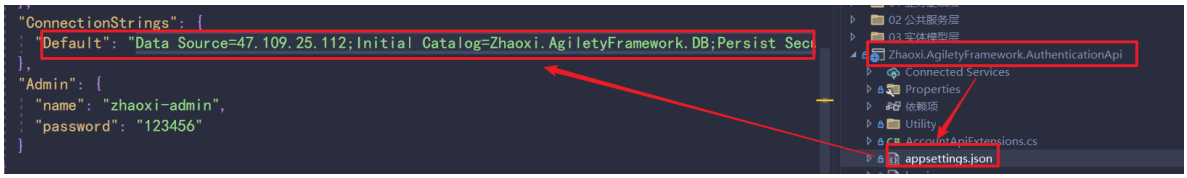
Data Source=PC-202206030027;Initial Catalog=Zhaoxi.AgiletyFramework.DB;Persist Security Info=True;User ID=sa;Password=ZHAOxi@2019;TrustServerCertificate=true



12.2.3、Zhaoxi.AgiletyFramework.AuthenticationApi项目配置文件修改字符串为，如下图

这里配置的时云服务器数据库连接字符串，如果是本地，请按照上面规则修改后，写入本地的数据库连接字符串

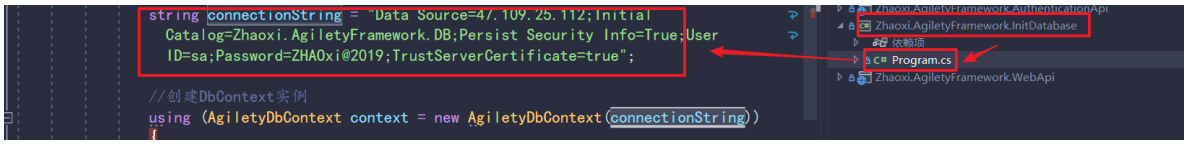
Data Source=PC-202206030027;Initial Catalog=Zhaoxi.AgiletyFramework.DB;Persist Security Info=True;User ID=sa;Password=ZHAOxi@2019;TrustServerCertificate=true



12.2.4、Zhaoxi.AgiletyFramework.InitDatabase控制台项目中，connectionString的值，如下图

这里配置的时云服务器数据库连接字符串，如果是本地，请按照上面规则修改后，写入本地的数据库连接字符串

Data Source=PC-202206030027;Initial Catalog=Zhaoxi.AgiletyFramework.DB;Persist Security Info=True;User ID=sa;Password=ZHAOxi@2019;TrustServerCertificate=true



12.3、开始本地启动后端项目

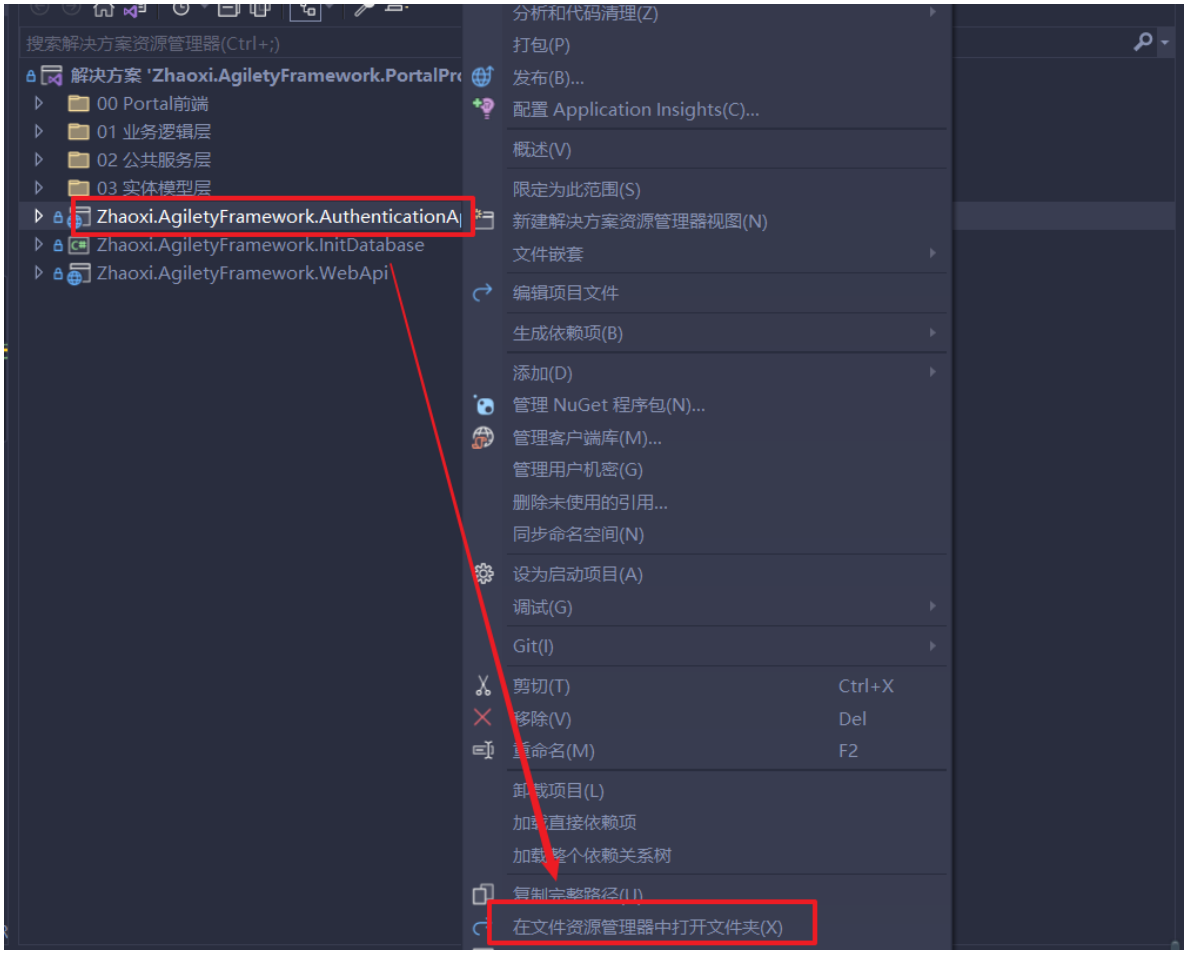
12.3.1、编译整个解决方案

12.3.2、执行项目Zhaoxi.AgiletyFramework.InitDatabase，用作初始化数据库结构

设置Zhaoxi.AgiletyFramework.InitDatabase为启动项目， 运行VS；生成数据库结构

12.3.3、启动认证服务器

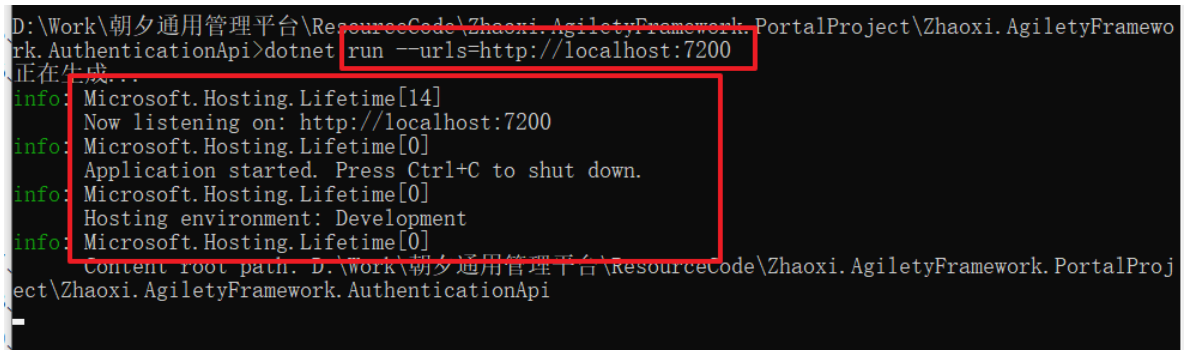
a、鼠标右键点击Zhaoxi.AgiletyFramework.AuthenticationApi 选择“文件资源管理器中打开文件夹”进入到项目所在文件件内。如图；



b. 进入项目文件夹后，输入cmd 打开命令窗口

c.输入命令

```
dotnet run --urls=http://localhost:7200
```



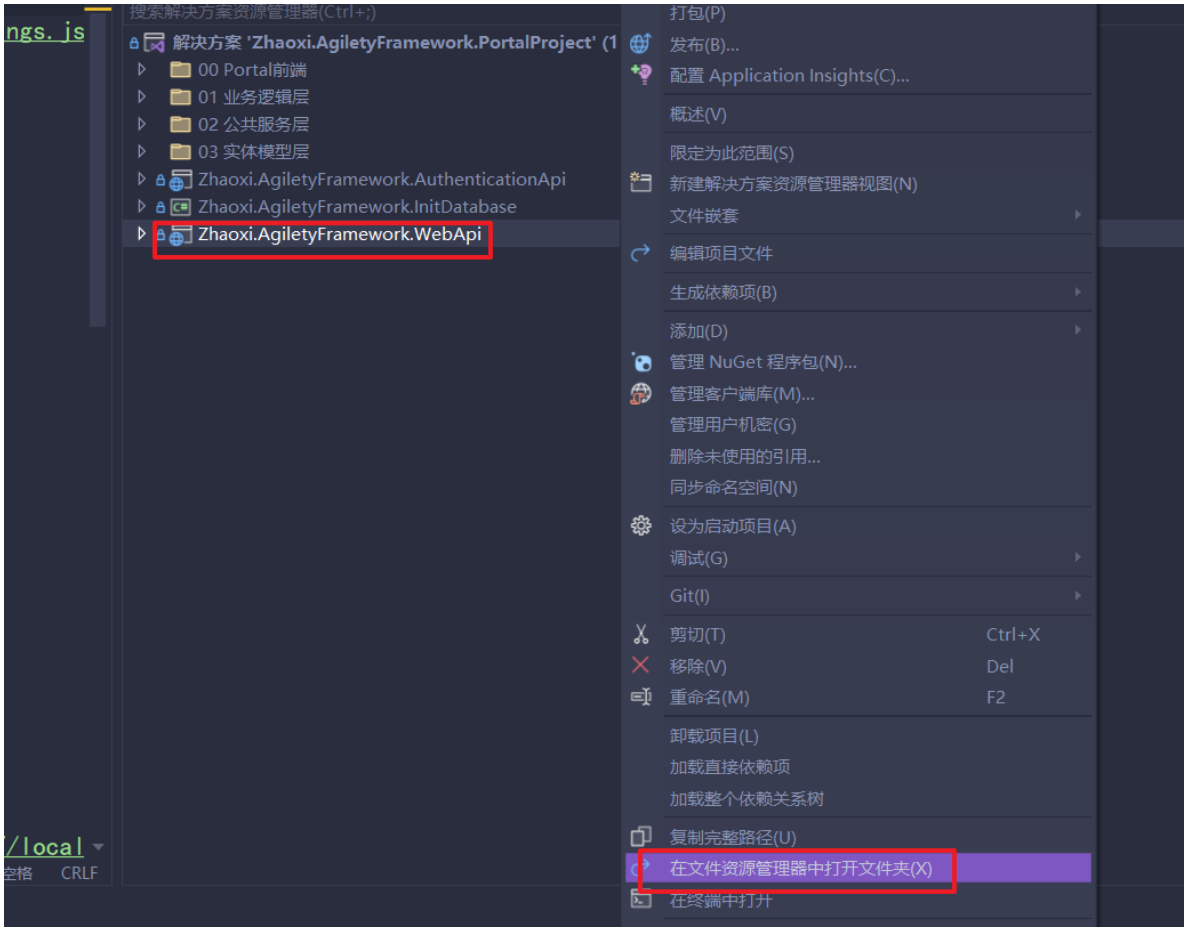
d、测试访问

```
访问： http:localhost:7200/swagger
```

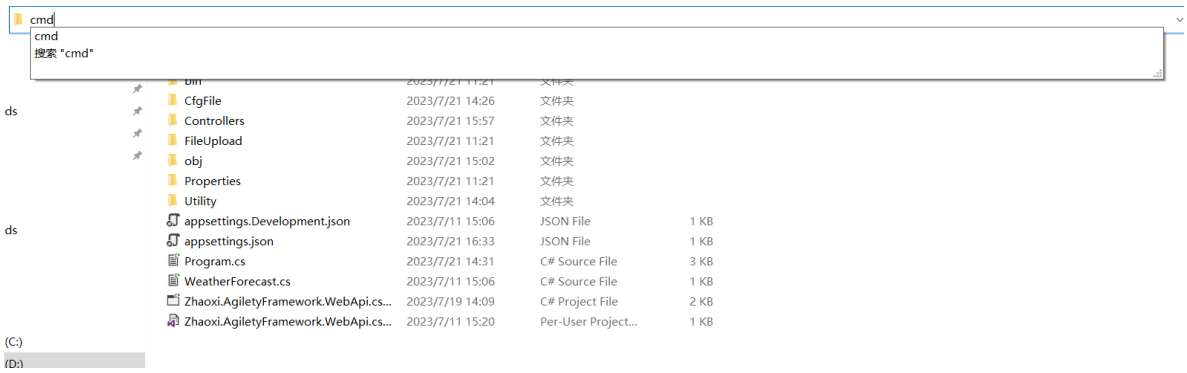
认证服务器启动成功

12.3.4、启动Api业务服务器

a、鼠标右键点击Zhaoxi.AgiletyFramework.WebApi选择“文件资源管理器中打开文件夹”进入到项目所在文件件内。如图；

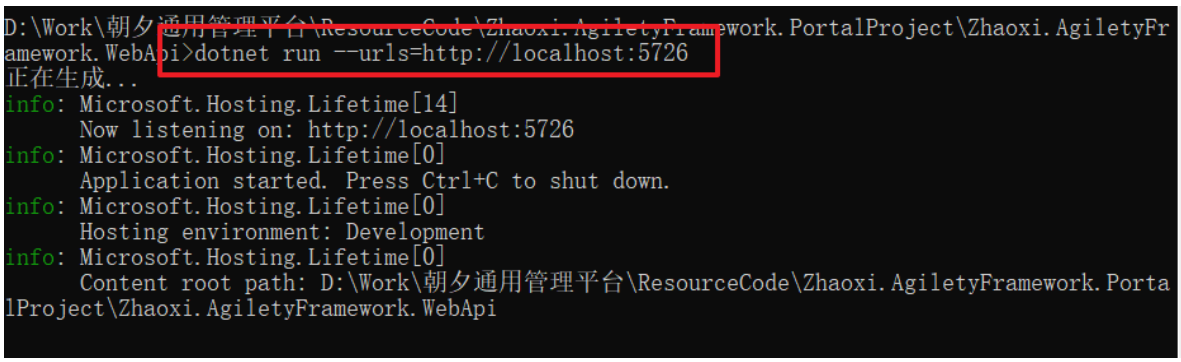


b. 进入项目文件夹后，输入cmd 打开命令窗口



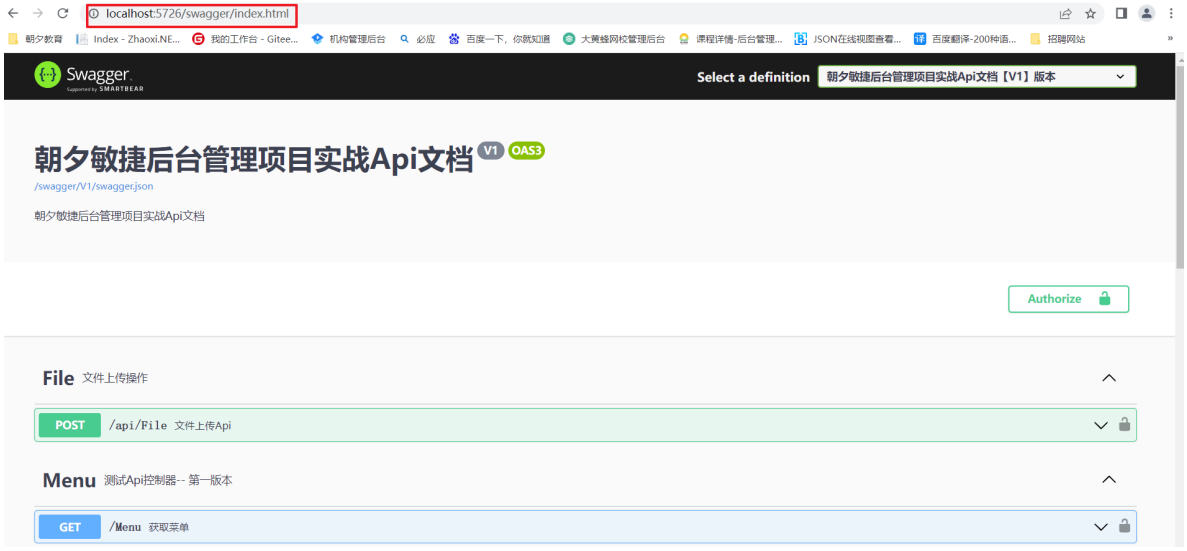
c.输入命令

```
dotnet run --urls=http://localhost:5726
```



d、测试访问

```
访问： http:localhost:5726/swagger
```



业务服务器启动成功

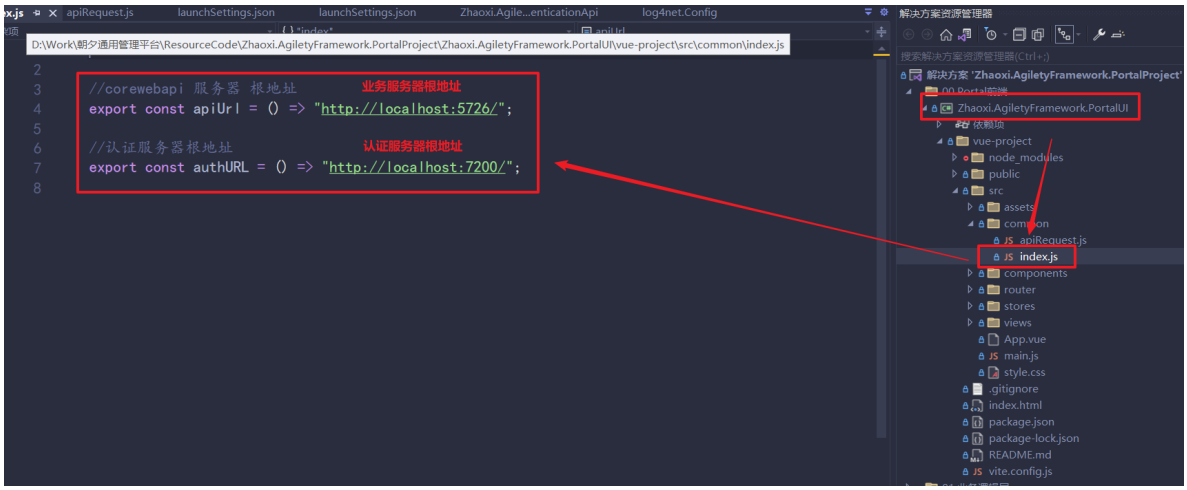
12.4、开始启动前端项目

12.4.1、配置前端项目请求的api地址

> 注意：这里有两个，一个是认证服务器Api地址，一个是业务服务器api地址

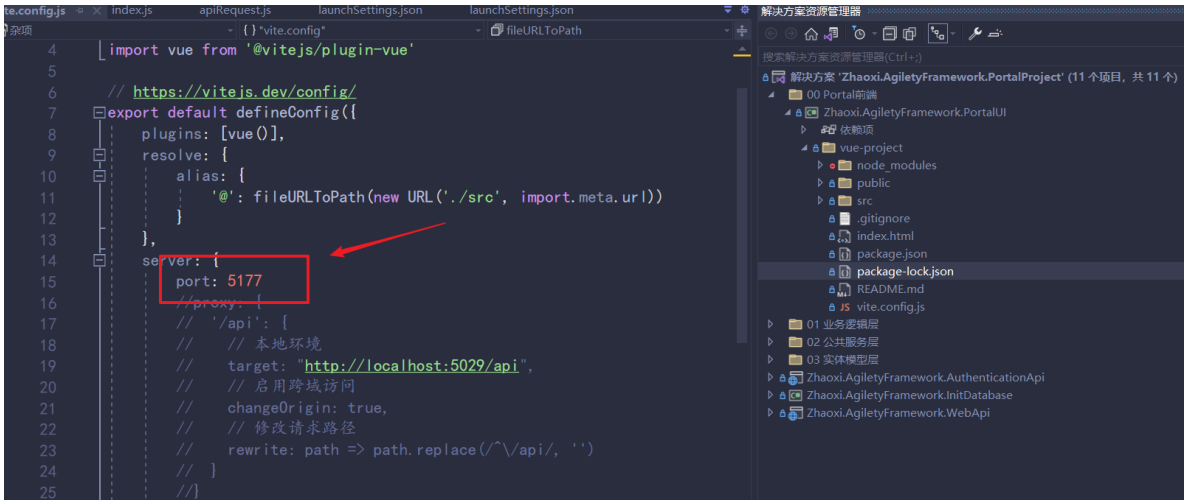
展开：Zhaoxi.AgiletyFramework.PortalUI ----> 展开：vue-project-----> 展开： src-----> 展开：common

点击index.js 文件。修改apiUrl 和 authURL的值； （根据已经启动的认证服务器和业务服务器地址来配置）



配置前端项目启动的时候，监听的进程端口号：

进入到：Zhaoxi.AgiletyFramework.PortalUI ----> 展开：vue-project- 打开vite.config.js配置文件。修改下图中红色框中的配置，这里配置的时5177. 根据自己的情况可以自行配置没有被占用的端口。



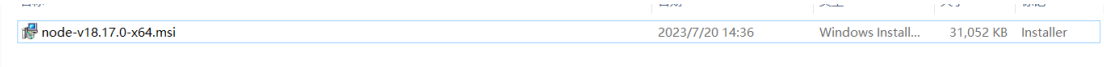
12.4.2、前端运行环境准备

12.4.2.1、安装NodeJs

NodeJs安装---浏览器访问： **https://nodejs.org/en**

> **安装包包含在课件中**

下载后得到安装包



12.4.2.2、双击安装Nodejs组件，点击下一步即可~

12.4.2.3、测试是否安装成功，进入cmd命令窗口执行

```
node -v    # 输出版本号
```

12.4.3、设置npm仓库地址

这个设置就类似于在VS中设置 Nuget安装工具下载路径；因为前端Vue3需要很多依赖，这里的配置就是指定到哪个服务器地址去下载安装包。

12.4.3.1、进入命令窗口

```
#查看地址：
npm config get registry
```


```
#设置为淘宝镜像
npm config set registry http://registry.npm.taobao.org/
```

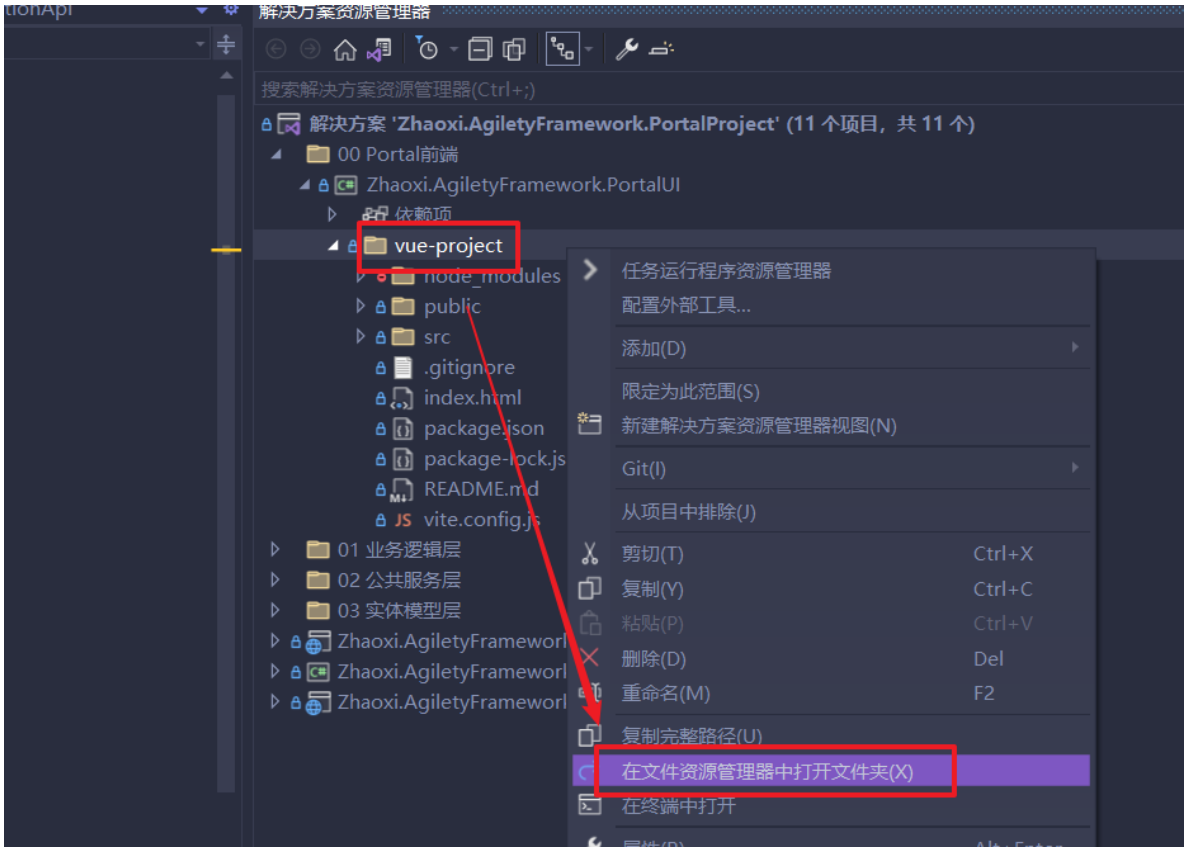
下面几个一般不用操作，是使用默认下载依赖包地址的，网速比较慢，不建议使用

```
#设置当前地址（设置为默认地址）：
npm config set registry https://registry.npmjs.org/
```

```
#恢复默认镜像地址：
npm config delete registry
```

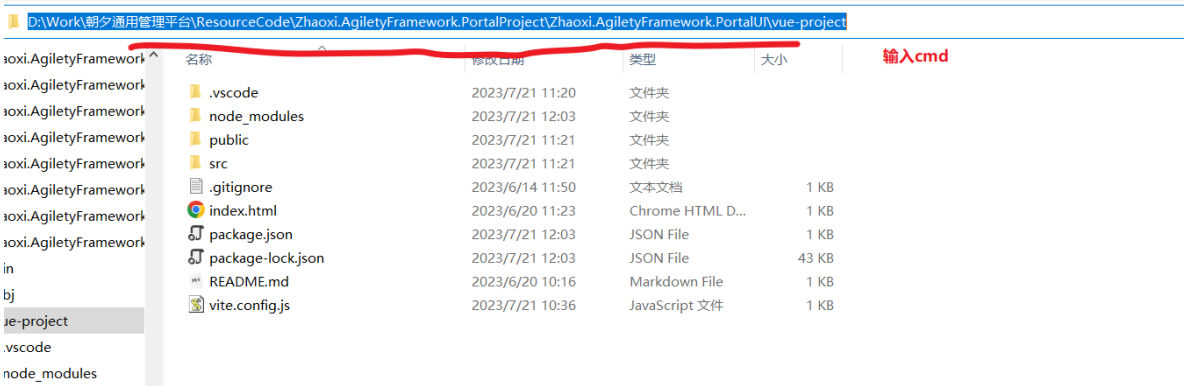
12.4.4、开始启动

12.4.4.1、右键点击vue-project 选择“在文件资源管理器中打开文件夹”，进入前端项目所在文件夹



进入前端项目所在文件夹。

12.4.4.2、选中现有路径地址，输出“cmd” 打开领命窗口



进入命令窗口：

执行命令，安装vue3项目依赖的程序包

```
npm i
```

12.4.4.3、启动项目

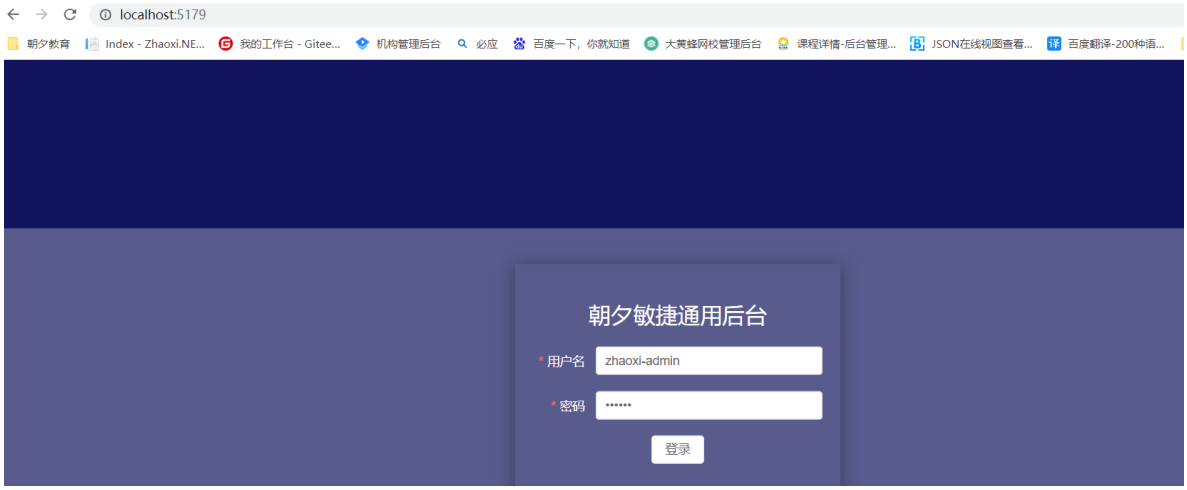
在当前窗口，继续执行命令：

```
npm run dev
```

12.4.4.4、访问测试

访问地址：

```
http://localhost:5179/
```

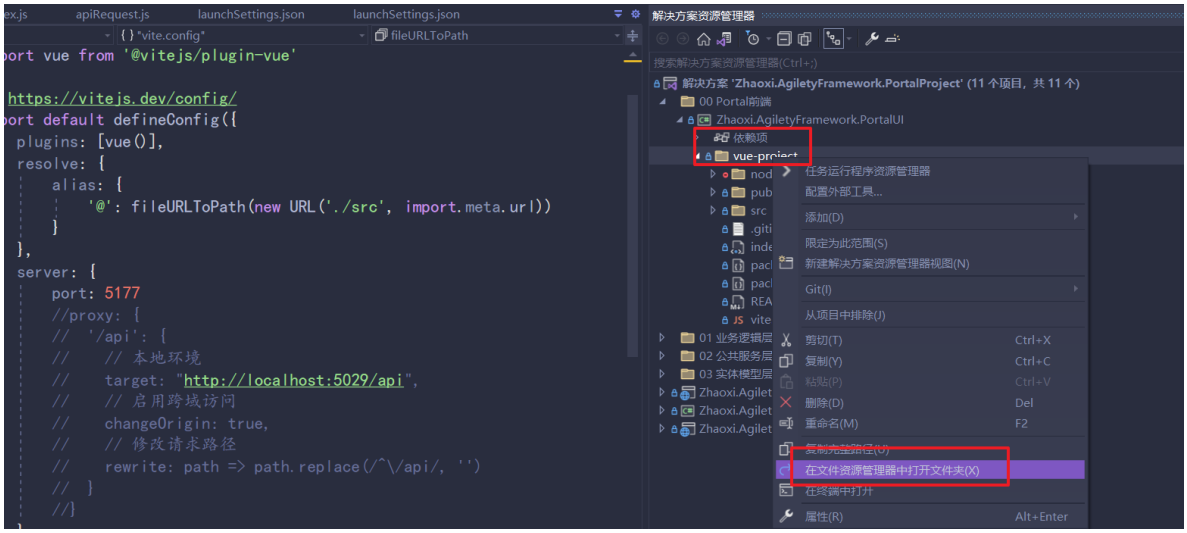


12.5、如果出现登录后没有菜单，可以检查用户角色映射表，角色次啊单映射表里面的用户Id的roleid， 都要改成1就行。

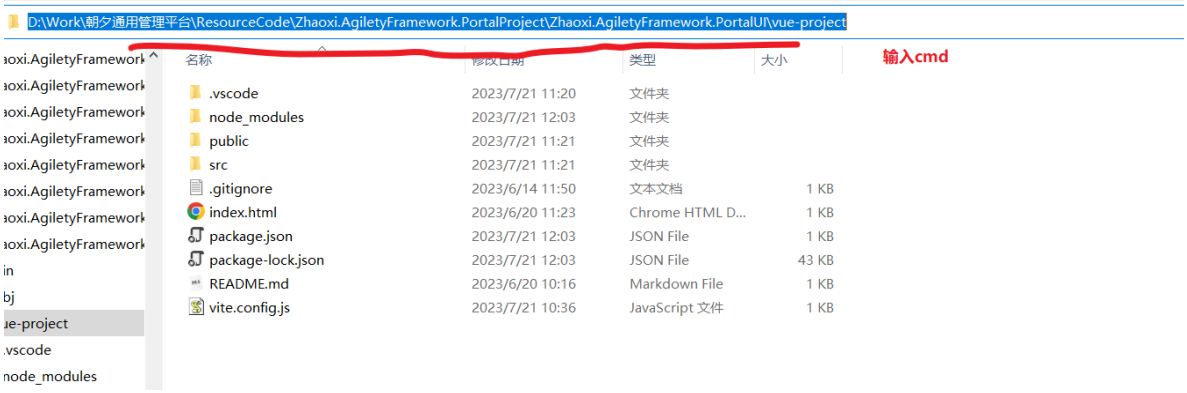
13、前端项目部署服务器

13.1、编译前端项目

进入到前端项目所在文件夹：右键点击vue-project 选择“在文件资源管理器中打开文件夹”，进入前端项目所在文件夹



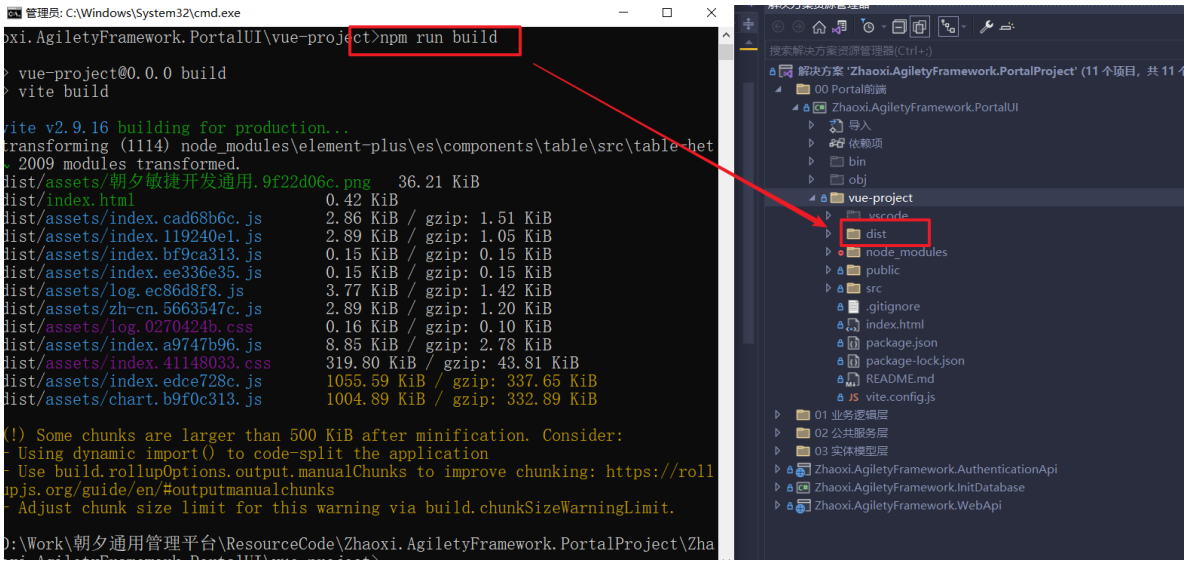
选中现有路径地址，输出“cmd” 打开领命窗口



进入命令窗口：执行命令开始编译文件

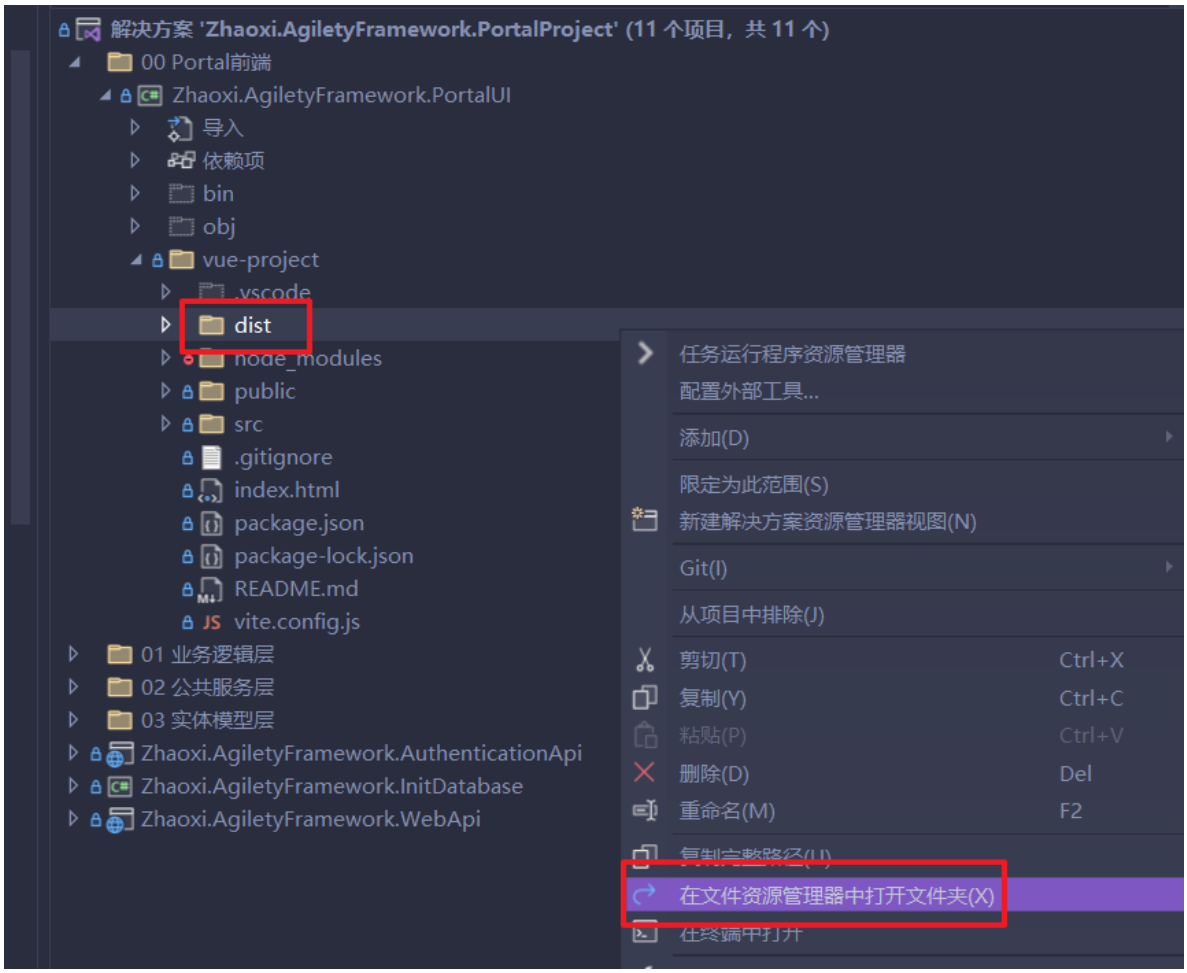
```
npm run build
```

编译完整，会在前端所在项目文件夹下，生成一个dist 文件夹。



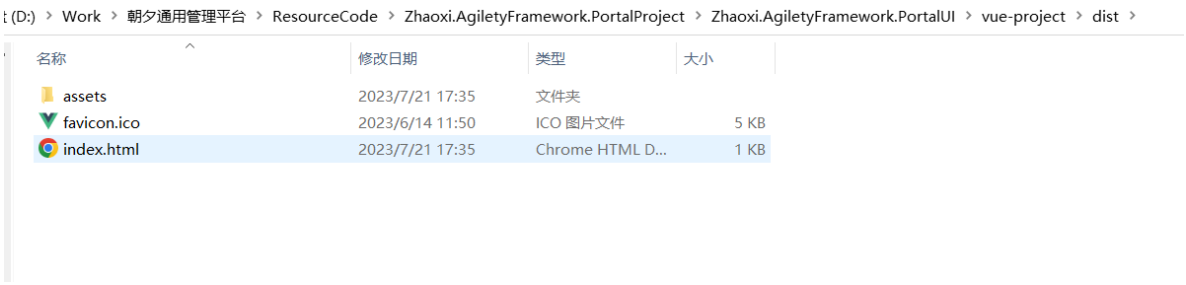
13.2、查看编译项目

13.2.1、右键点击vue-project下的dist 选择“在文件资源管理器中打开文件夹”，进入前端项目所在文件夹



查看会发现其实是生成了一堆静态文件：

13.2.2、查看编译项目



接下来开始部署静态文件到Nginx服务器。 ngixn就是一个静态服务器。编译好的静态文件是可以直接部署在nginx金泰服务器的。

13.3、下载Nginx

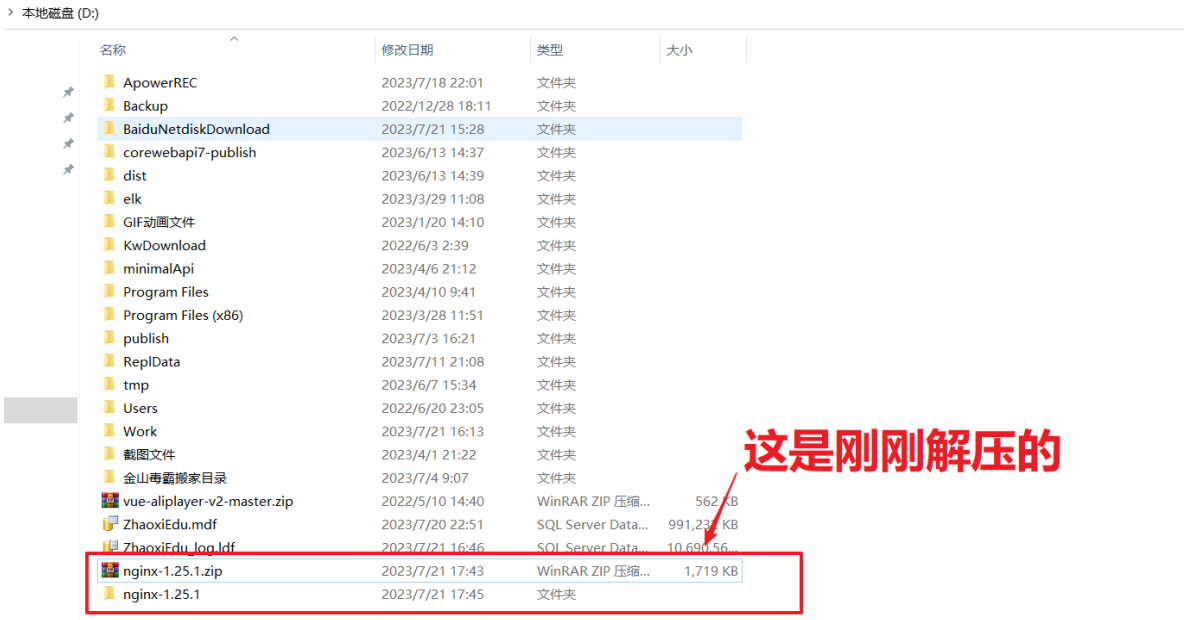
13.3.1、准备nginx

因为本地是Windows环境，所以这里下载的时Windows版本的Nginx，更多的可能是在Linux上操作。

下载地址：

```
http://nginx.org/en/download.html
```

下载后得到的时一个zip包。解压

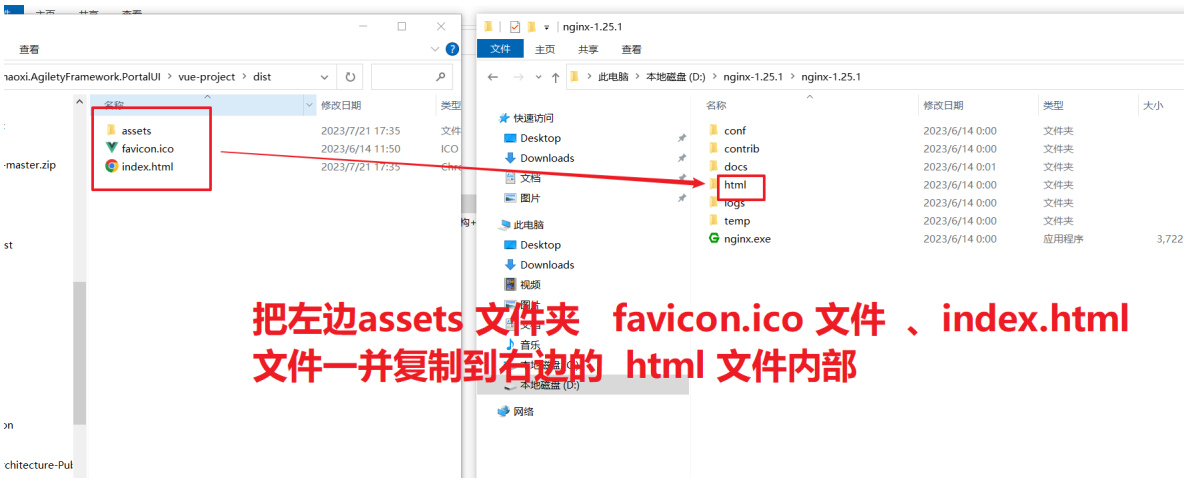


13.3.2、查看nginx内部结构

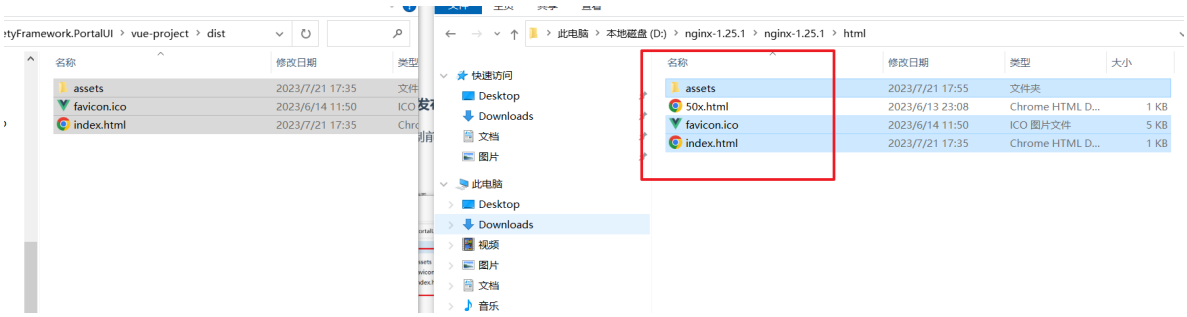
鼠标双击进入到nginx-1.25.1 文件夹内部，这里可能爱解压的时候，会有多层nginx-1.25.1 文件夹。不过这里不影响，继续进入到nginx-1.25.1 文件夹内部。内部如下结构； 内部会有一个html 文件夹。后续这里需要使用的。

13.4、开始发布前面build好的前端静态文件；

13.4.1、复制前面前端build的静态文件 到 nginx 内部的 html 文件夹内。覆盖之前ngixn的内容。

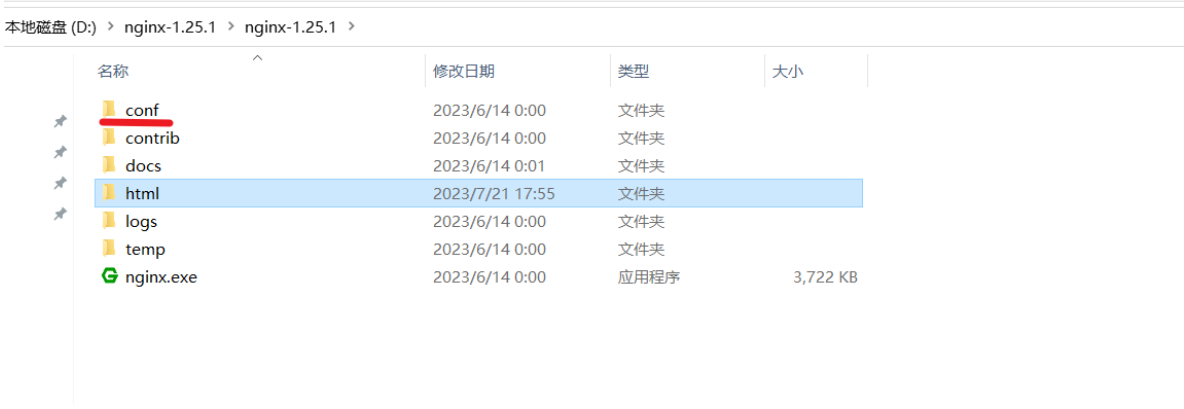


12.4.2、复制完毕后的结果：

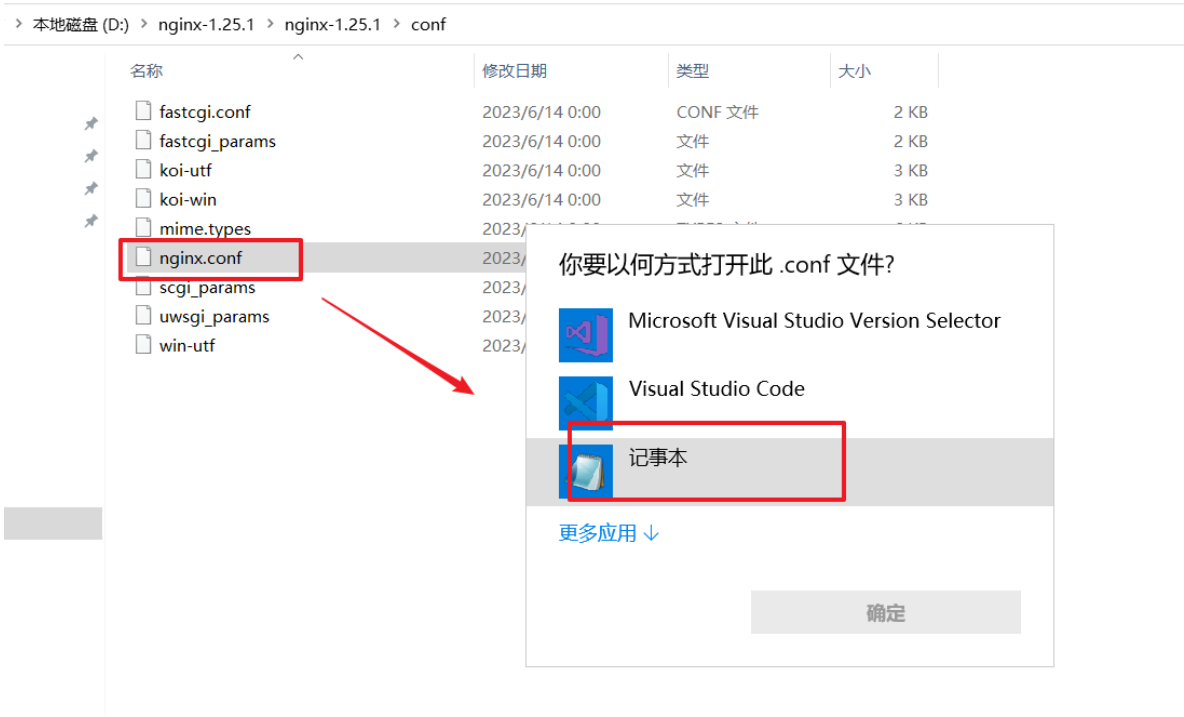


12.4.3、修改nginx 配置文件监听地址。

12.4.3.1、进入 nginx-1.25.1 文件夹内部的 conf 文件夹。

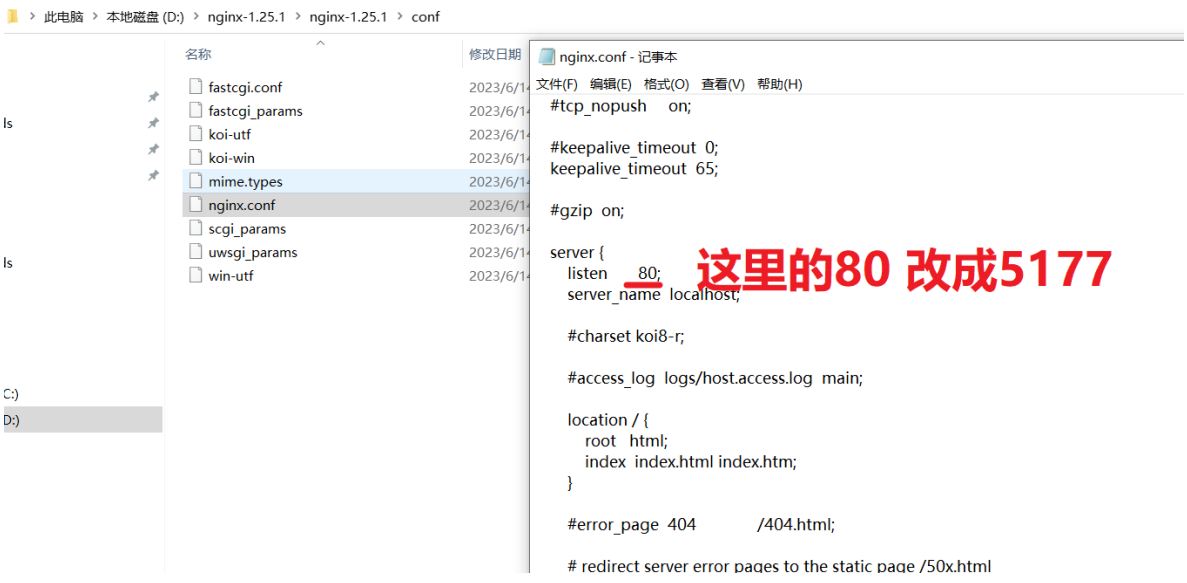


12.4.3.2、使用记事本打开nginx.conf 文件

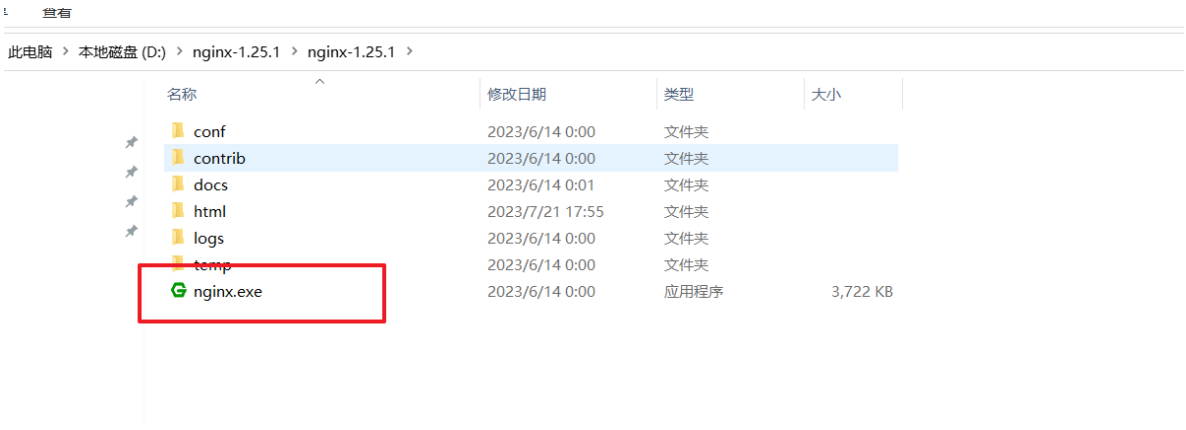


12.4.3.3、修改nginx.conf内部的内容， 把listen 80 改成 listen 5177

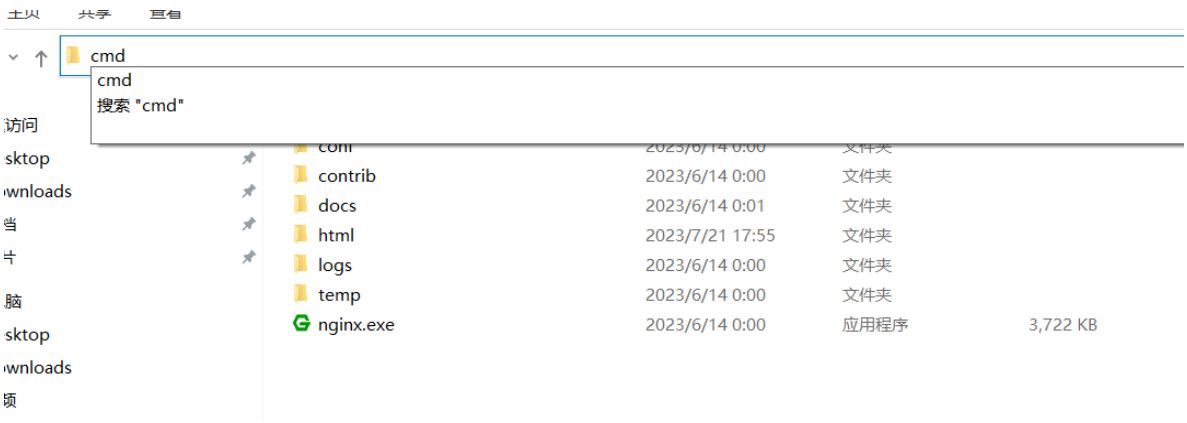
保存。如下图;



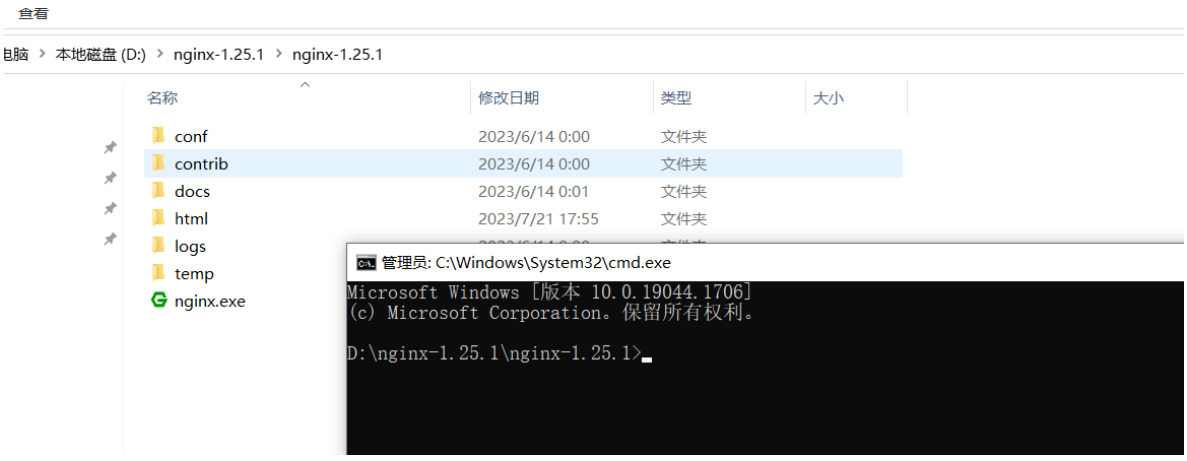
12.4.3.4、进入nginx.exe 所在目录下，如下图



12.4.3.5、选中文件夹路径，输出cmd ， 打开命令窗口

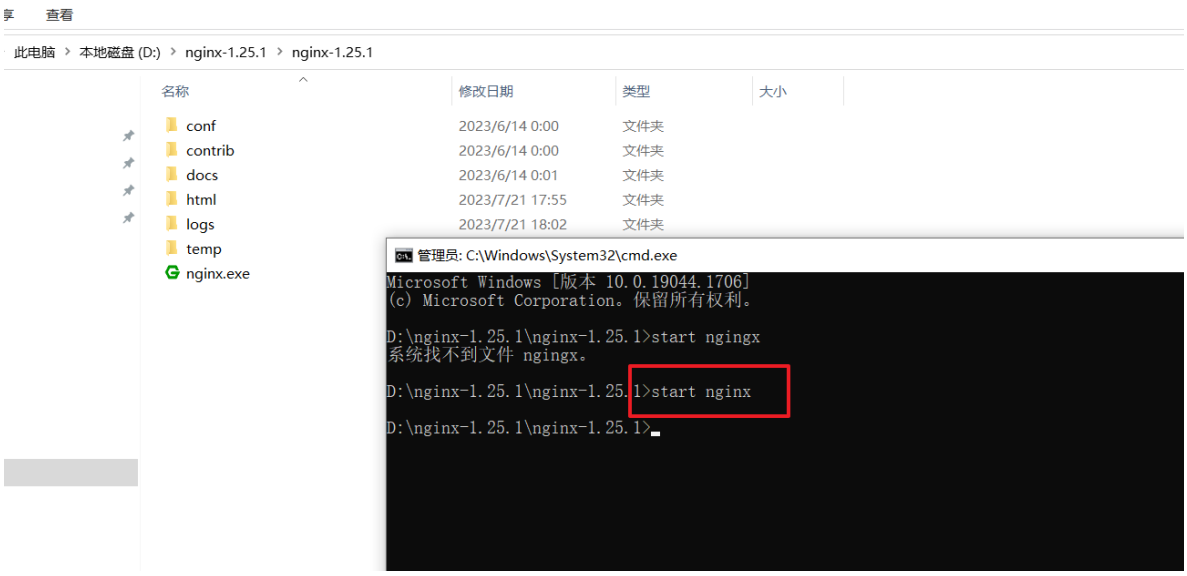


打开命令窗口如下



12.4.3.6、执行命令

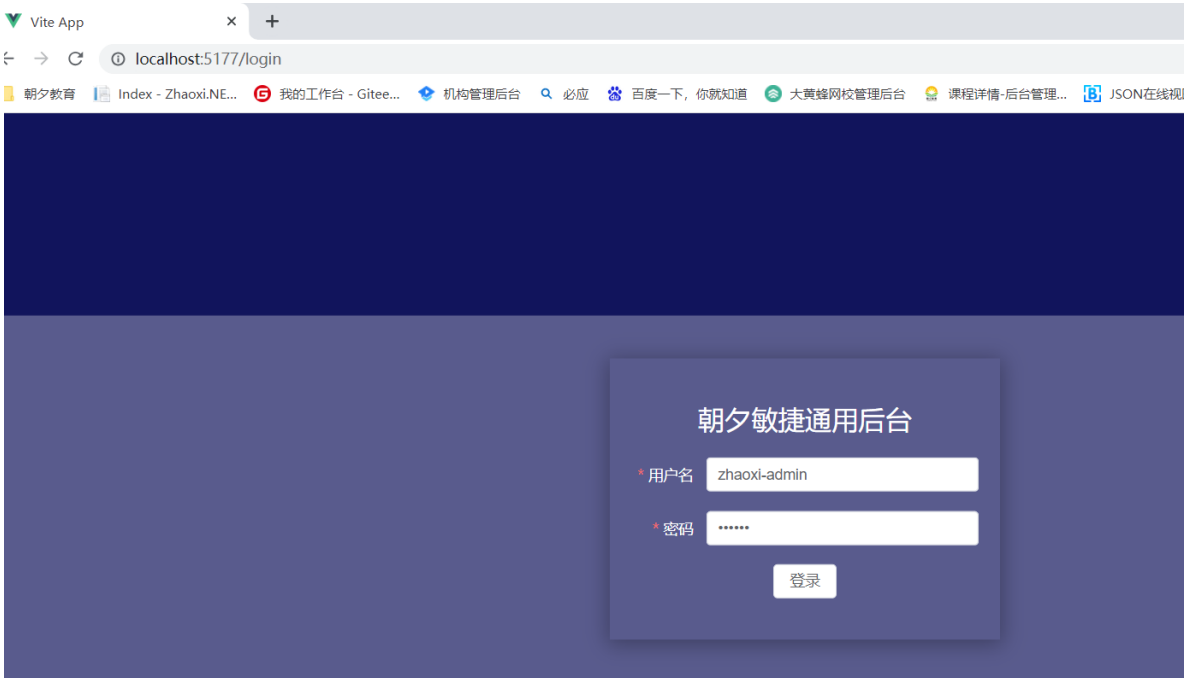
```
start nginx
```



12.4.3.7、开始访问

```
http://localhost:5177/login
```

注意： 这里访问的其实是访问的nginx，nginx响应的时前面build的静态文件。




```
已解决：1  用户初始化在授权中心

2  dotnet run --urls=http://localhost:5726
   dotnet run --urls=http://localhost:7200

已解决：3  把运行 联调的步骤放在后面

已解决：4  log4net里面数据库连接字符串问题----如何多渠道记录---在log4net 环境有说明

已解决：5  授权中心返回，只需要token---前端是可以解析token，直接解析就行、
```

已解决：两个个中间件返回的独立成委托----生成token方法封装一下---------跟鉴权授权扩展关联起来

已解决：6 后缀Ext就改成Extensions---文件夹叫Extend

已解决：7 前端的编译过程命令和nignx挂载
    前端修改默认端口：port: 5177

已解决：8 前端刷新后回到登录页

晚上处理：9 新增用户功能

已解决：10 刷新token时，，refreshtoken的周期不变