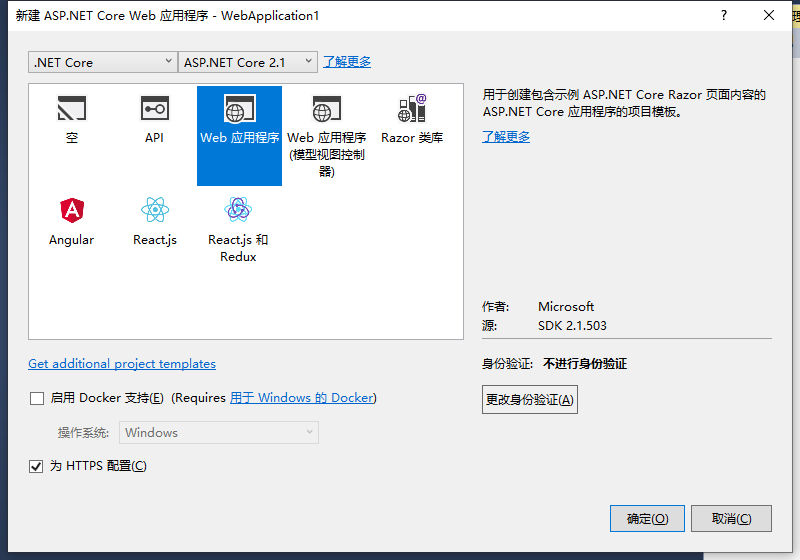
通用架构实现方案

## 创建web 应用程序



使用dotnet 的restore、build、run命令分别用来加载依赖，编译和运行

## 命令

Dotnet –version 查看版本信息

Dotnet build 生成.net 项目

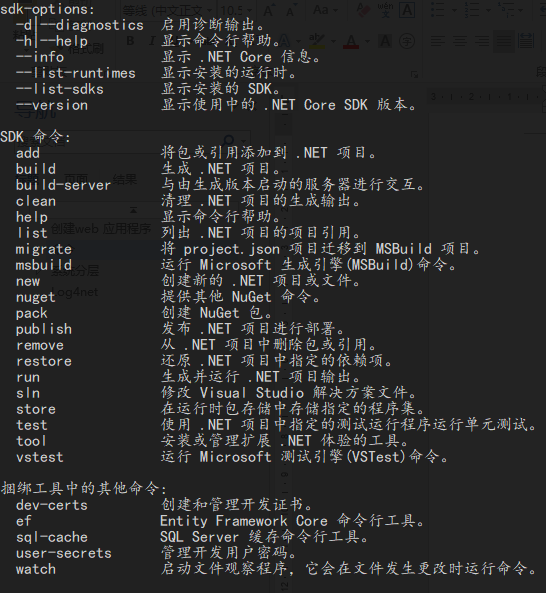
Dotnet list 列出.net 项目的项目引用

Dotnet run 生成并运行.net项目输出

Dotnet store 在运行时包存储指定的程序集

Dotnet restore 1.还原.net 项目中指定的依赖包

Dotnet watch run 2



## 引入dotnet watch

由于新建的项目中，默认是不包含dotnet watch工具的，我们需要把该工具引入，打开.csproj文件，加入一下标签。

<ItemGroup>

<DotNetCliToolReference Include="Microsoft.DotNet.Watcher.Tools" Version="2.0.0" />

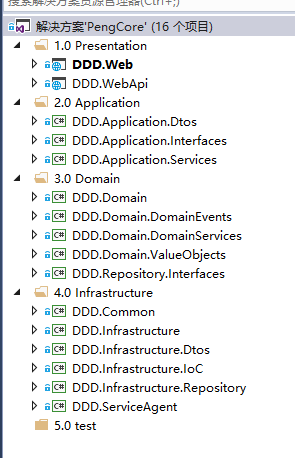
</ItemGroup>

加入后，记得ctrl+s保存代码。然后执行**dotnet restore**命令，即会下载安装dotnet watch工具。

最后 ：运行代码

当dotnet restore命令执行成功后，执行**dotnet watch run**

## 系统分层



### 1.0 Presentation 表现层

DDD.Web 表示层

用于提供用户的数据展现

AutoMapper

AutoMapper 和AutoMapper.Extensions.Microsoft.DependencyInjection

DDD.WebApi 数据接口层

用户提供数据接口

### 2.0 Application 应用层

DDD.Application.Dtos 应用层的数据对象传输

DDD.Application.Interfaces 应用层的数据接口

DDD.Application.Services 应用层的数据实现

### 3.0 Domain 领域层

DDD.Domain 领域实体数据

DDD.Domain.DomainEvents 领域事件

DDD.Domain.DomainServices 领域服务

DDD.Domain.ValueObjects 领域值对象

DDD.Repository.Interfaces 领域接口

定义接口基类和具体的业务接口类

### 4.0 Infrastructure 基础设施

DDD.Common 基础设施公共方法

DDD.Common 数据模型

DDD.Infrastructure.Dtos 基础设施数据对象传输

Appsettings.cs

引入：Microsoft.Extensions.Configuration 和 Microsoft.Extensions.Configuration.Json

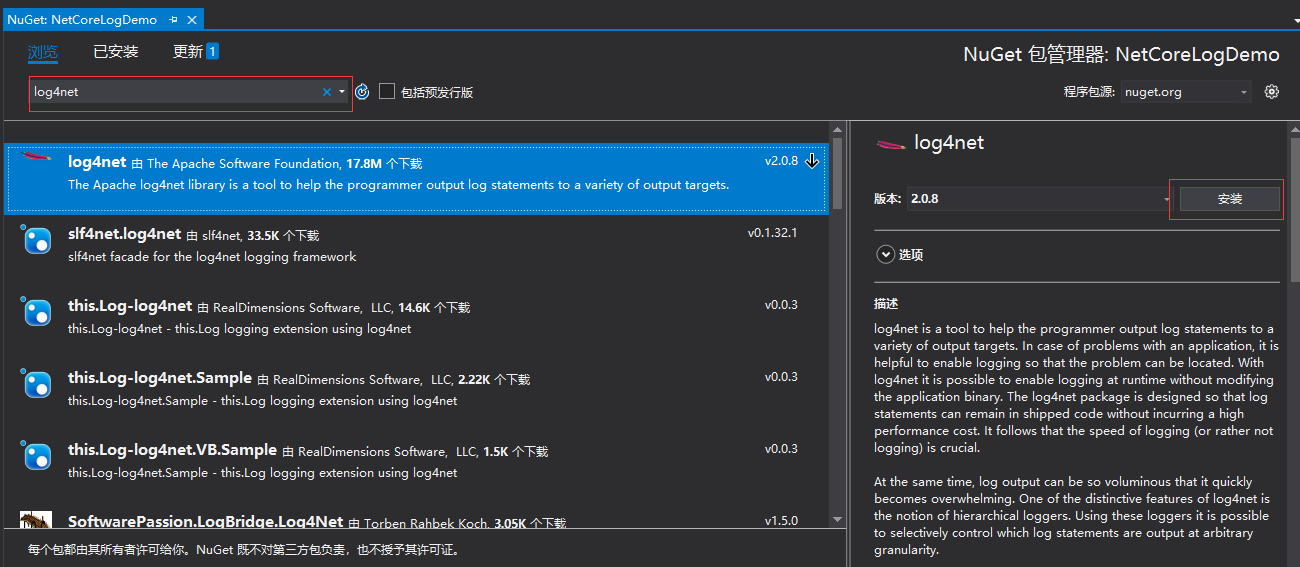
DDD.Infrastructure.IoC 控制反转

DDD.Infrastructure.Repository 基础设施方法实现

DDD.ServiceAgent 基础设施服务代理

## Log4net

1. 安装Log4net



1. 添加 log4net.config 到根目录（文件在项目中）
2. 在Startup.cs类里面配置使用log4net

public static ILoggerRepository repository { get; set; }

public Startup(IConfiguration configuration)

{

Configuration = configuration;

repository = LogManager.CreateRepository("NETCoreRepository");

// 指定配置文件

XmlConfigurator.Configure(repository, new FileInfo("log4net.config"));

}

四、在控制器里面注入log4net

public class HomeController : Controller

{

private ILog log;

public HomeController(IHostingEnvironment hostingEnv)

{

this.log = LogManager.GetLogger(Startup.repository.Name, typeof(HomeController));

}

public IActionResult Index()

{

log.Error("测试日志");

return View();

}

}

## 身份认证

### 认证

最常见的身份验证协议是SAML2p，WS-Federation和OpenID Connect-SAML2p是最流行和部署最广泛的协议。

OpenID Connect是三者中的最新者，但被认为是未来，因为它在现代应用程序中具有最大的潜力。它从一开始就针对移动应用程序场景而构建，并且旨在实现API友好。

### API访问

OAuth2是一种协议，允许应用程序从安全令牌服务请求访问令牌并使用它们与API通信。由于可以集中身份验证和授权，因此这种委派降低了客户端应用程序和API的复杂性。

### OpenID Connect和OAuth 2.0 –更好的结合在一起

OpenID Connect和OAuth 2.0非常相似-实际上，OpenID Connect是OAuth 2.0的扩展。身份验证和API访问这两个基本的安全问题被组合成一个协议-通常只需一次往返于安全令牌服务。

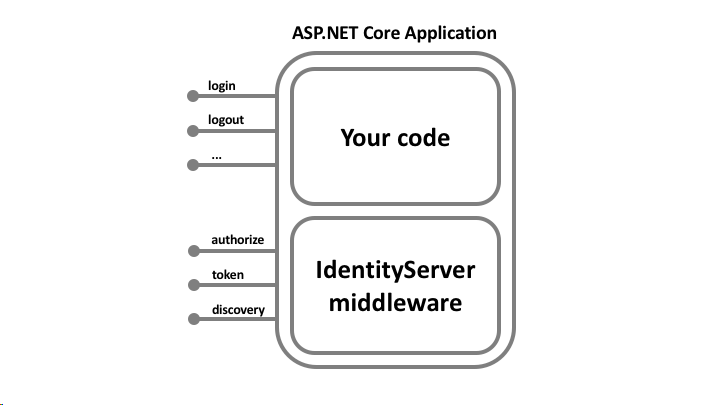
我们认为，在可预见的将来，OpenID Connect和OAuth 2.0的结合是保护现代应用程序安全的最佳方法。IdentityServer4是这两个协议的实现，并且经过高度优化以解决当今移动，本机和Web应用程序的典型安全问题。

### IdentityServer4

IdentityServer是OpenID Connect提供程序-它实现了OpenID Connect和OAuth 2.0协议。

### IdentityServer4可以帮助你做什么

IdentityServer是将规范兼容的OpenID Connect和OAuth 2.0端点添加到任意ASP.NET Core应用程序的中间件。通常，您构建（或重新使用）包含登录和注销页面的应用程序，IdentityServer中间件会向其添加必要的协议头，以便客户端应用程序可以与其对话 使用这些标准协议。



JWT与OAuth的区别 -OAuth2是一种授权框架 ，JWT是一种认证协议 -无论使用哪种方式切记用HTTPS来保证数据的安全性 -OAuth2用在使用第三方账号登录的情况(比如使用weibo, qq, github登录某个app)，而JWT是用在前后端分离, 需要简单的对后台API进行保护时使用。