整个项目为分层设计：

在 Acme.BookStore 项目的分层架构中，每一层都有其特定的功能和职责，以下是对每一层功能的详细描述：

一，表示层（Presentation Layer）

功能：负责与用户进行交互，提供用户界面或API接口。

职责：

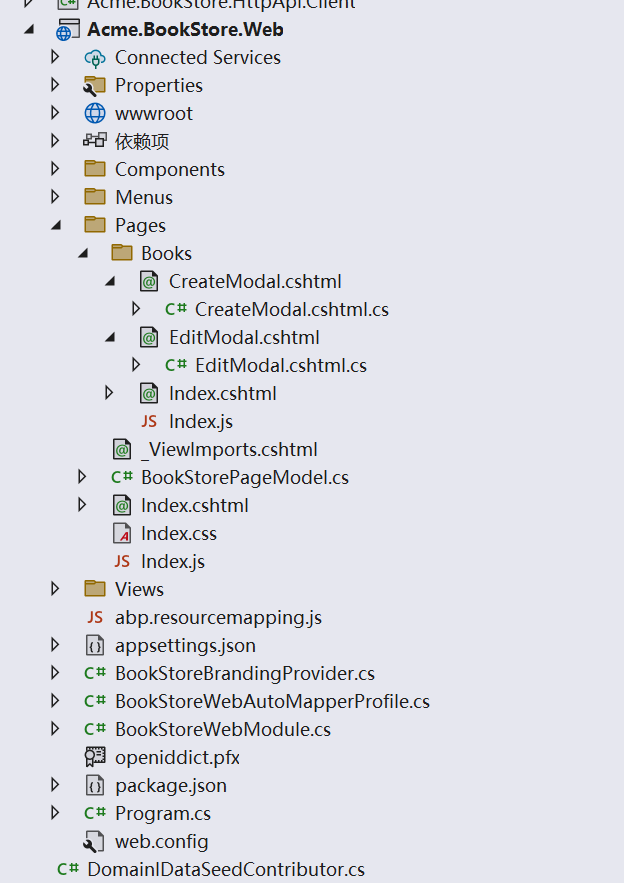
处理用户输入和输出。

展示数据和接收用户操作。

调用应用层的服务来处理业务逻辑。

使用DTO（Data Transfer Objects）与应用层进行数据交换，避免直接暴露领域模型。

示例项目名：Acme.BookStore.Web



二，应用层（Application Layer）

功能：负责协调业务逻辑，处理用户请求，管理应用流程。

职责：

提供应用服务（Application Services），处理用户请求。

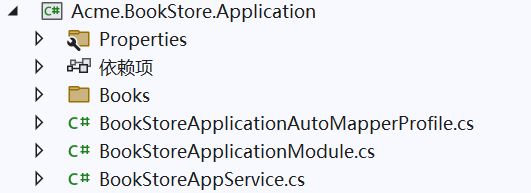
协调领域层和基础设施层的交互。

使用命令和查询（Commands and Queries）实现CQRS模式。

处理授权和认证。

发布和处理领域事件。

示例项目名：Acme.BookStore.Application



1. 领域层（Domain Layer）

功能：包含核心业务逻辑和领域模型，是整个应用的核心。

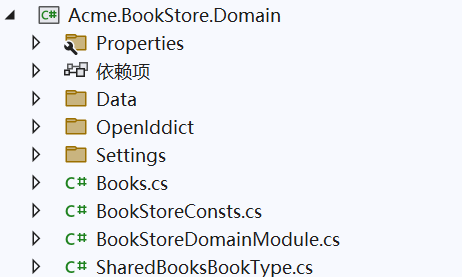
职责：

定义实体（Entities）、值对象（Value Objects）、领域服务（Domain Services）等。

处理业务规则和领域事件。

提供仓储接口（Repository Interfaces），但不实现具体的数据访问逻辑。

示例项目名：Acme.BookStore.Domain



1. 基础设施层（Infrastructure Layer）

功能：提供通用的技术功能，如数据访问、文件存储、消息队列等。

职责：

实现领域层的接口，如仓储（Repositories）。

处理外部服务和系统的集成。

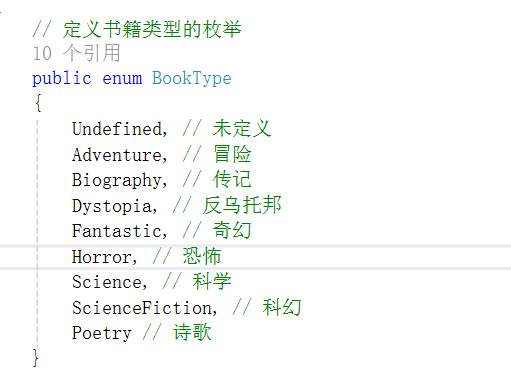
提供通用的工具类和辅助功能。

示例项目名：Acme.BookStore.Infrastructure

首先创建Book实体类定义数据



2，定义类中用到枚举类型所以需要创建枚举类



将Book实体添加到DbContext中和Book实体映射到数据库表

1. 定义一个类该类需要继承框架自带的ApplicationService基类
2. 创建应用程序
3. dto类用于数据的修改



1，应用层（Application Layer）

应用服务（Application Services）：应用服务接收来自表示层的请求，并调用领域层的服务或直接操作领域对象来执行数据操作。

命令和查询（Commands and Queries）：应用层使用命令模式来处理写操作（如创建、更新、删除），使用查询模式来处理读操作。

public class BookAppService : IBookAppService

{

private readonly IBookRepository \_bookRepository;

public BookAppService(IBookRepository bookRepository)

{

\_bookRepository = bookRepository;

}

public async Task<BookDto> GetBookByIdAsync(int id)

{

var book = await \_bookRepository.GetByIdAsync(id);

return book.MapToDto();

}

public async Task CreateBookAsync(CreateBookDto dto)

{

var book = new Book

{

Title = dto.Title,

Author = dto.Author,

PublishDate = dto.PublishDate

};

await \_bookRepository.SaveAsync(book);

}

}

2，领域层（Domain Layer）

实体（Entities）：实体是领域模型的核心，包含业务数据和业务逻辑。例如，Book 实体可能包含书的基本信息和相关操作。

值对象（Value Objects）：值对象是不可变的，用于表示没有标识符的属性集合。例如，Address 可以是一个值对象。

领域服务（Domain Services）：领域服务封装了跨越多个实体的业务逻辑。

仓储接口（Repository Interfaces）：领域层定义仓储接口，但不实现具体的数据访问逻辑。

public class Book

{

public int Id { get; set; }

public string Title { get; set; }

public string Author { get; set; }

public DateTime PublishDate { get; set; }

}

public interface IBookRepository

{

Task<Book> GetByIdAsync(int id);

Task SaveAsync(Book book);

}

3，基础设施层（Infrastructure Layer）

仓储实现（Repository Implementations）：基础设施层实现领域层定义的仓储接口，提供具体的数据访问逻辑。例如，BookRepository 实现 IBookRepository 接口，使用ORM（如Entity Framework）或直接的SQL查询来访问数据库。

数据上下文（Data Context）：基础设施层可能包含数据上下文，用于管理数据库连接和事务。

public class BookRepository : IBookRepository

{

private readonly BookStoreDbContext \_dbContext;

public BookRepository(BookStoreDbContext dbContext)

{

\_dbContext = dbContext;

}

public async Task<Book> GetByIdAsync(int id)

{

return await \_dbContext.Books.FindAsync(id);

}

public async Task SaveAsync(Book book)

{

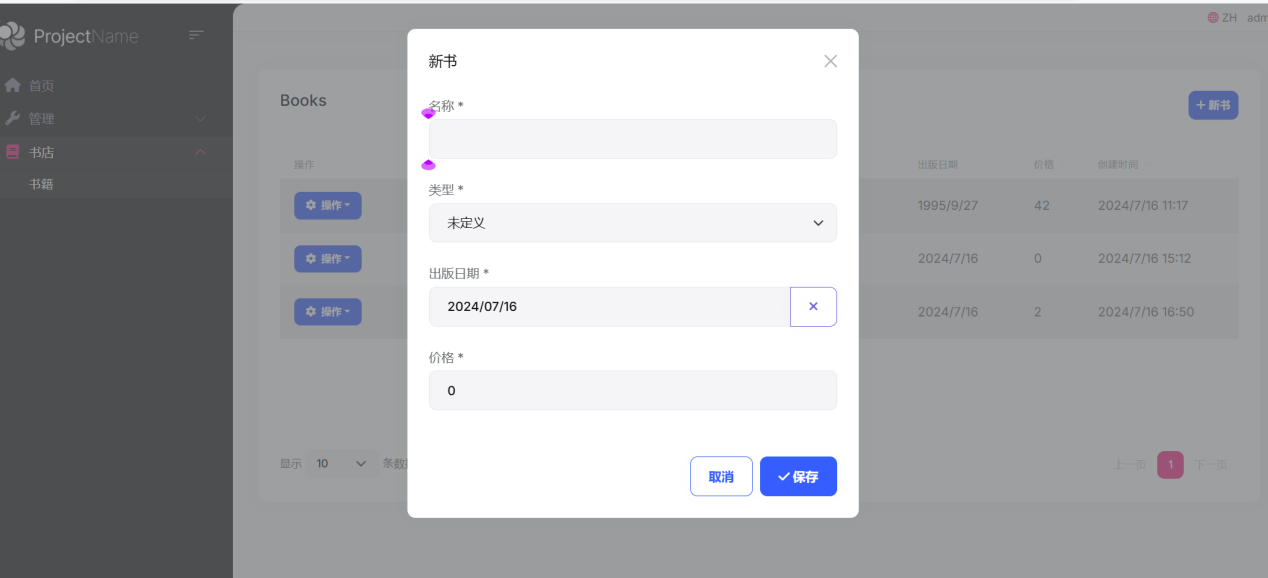
\_dbContext.Books.Add(book);

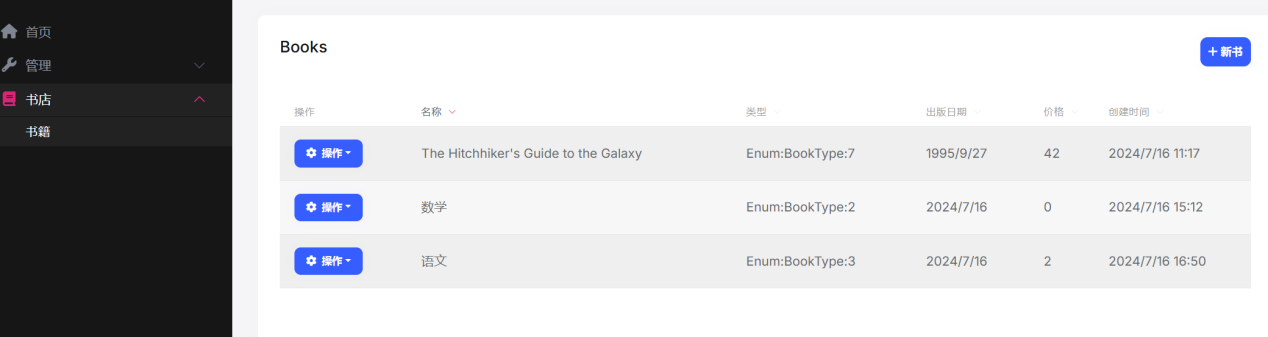
await \_dbContext.SaveChangesAsync();

}

}

添加数据如下





删除数据如下





该框架自动生成对应WEBAPI接口

单元测试模块主要针对应用层和领域层的代码进行测试，确保每个单元（如方法、类）的功能正确。

using Xunit;

using Moq;

using Acme.BookStore.Application.Services;

using Acme.BookStore.Domain.Entities;

using Acme.BookStore.Domain.Repositories;

using Acme.BookStore.Application.DTOs;

using System.Threading.Tasks;

public class BookAppServiceTests

{

[Fact]

public async Task GetBookByIdAsync\_ShouldReturnBookDto()

{

// Arrange

var bookId = 1;

var book = new Book { Id = bookId, Title = "Test Book", Author = "Test Author", PublishDate = DateTime.Now };

var bookRepositoryMock = new Mock<IBookRepository>();

bookRepositoryMock.Setup(repo => repo.GetByIdAsync(bookId)).ReturnsAsync(book);

var bookAppService = new BookAppService(bookRepositoryMock.Object);

// Act

var result = await bookAppService.GetBookByIdAsync(bookId);

// Assert

Assert.NotNull(result);

Assert.Equal(book.Id, result.Id);

Assert.Equal(book.Title, result.Title);

Assert.Equal(book.Author, result.Author);

Assert.Equal(book.PublishDate, result.PublishDate);

}

[Fact]

public async Task CreateBookAsync\_ShouldSaveBook()

{

// Arrange

var createBookDto = new CreateBookDto { Title = "New Book", Author = "New Author", PublishDate = DateTime.Now };

var bookRepositoryMock = new Mock<IBookRepository>();

bookRepositoryMock.Setup(repo => repo.SaveAsync(It.IsAny<Book>())).Returns(Task.CompletedTask);

var bookAppService = new BookAppService(bookRepositoryMock.Object);

// Act

await bookAppService.CreateBookAsync(createBookDto);

// Assert

bookRepositoryMock.Verify(repo => repo.SaveAsync(It.Is<Book>(b => b.Title == createBookDto.Title && b.Author == createBookDto.Author && b.PublishDate == createBookDto.PublishDate)), Times.Once);

}

}

集成测试：

集成测试主要针对基础设施层的代码进行测试，确保各个组件之间的集成和数据访问逻辑正确。

基础设施层集成测试

代码如下

using Xunit;

using Microsoft.EntityFrameworkCore;

using Acme.BookStore.Infrastructure.Data;

using Acme.BookStore.Infrastructure.Repositories;

using Acme.BookStore.Domain.Entities;

using System.Threading.Tasks;

public class BookRepositoryTests

{

private readonly BookStoreDbContext \_dbContext;

public BookRepositoryTests()

{

var options = new DbContextOptionsBuilder<BookStoreDbContext>()

.UseInMemoryDatabase(databaseName: "BookStoreTestDatabase")

.Options;

\_dbContext = new BookStoreDbContext(options);

}

[Fact]

public async Task GetByIdAsync\_ShouldReturnBook()

{

// Arrange

var bookId = 1;

var book = new Book { Id = bookId, Title = "Test Book", Author = "Test Author", PublishDate = DateTime.Now };

\_dbContext.Books.Add(book);

await \_dbContext.SaveChangesAsync();

var bookRepository = new BookRepository(\_dbContext);

// Act

var result = await bookRepository.GetByIdAsync(bookId);

// Assert

Assert.NotNull(result);

Assert.Equal(book.Id, result.Id);

Assert.Equal(book.Title, result.Title);

Assert.Equal(book.Author, result.Author);

Assert.Equal(book.PublishDate, result.PublishDate);

}

[Fact]

public async Task SaveAsync\_ShouldSaveBook()

{

// Arrange

var book = new Book { Title = "New Book", Author = "New Author", PublishDate = DateTime.Now };

var bookRepository = new BookRepository(\_dbContext);

// Act

await bookRepository.SaveAsync(book);

// Assert

var savedBook = await \_dbContext.Books.FindAsync(book.Id);

Assert.NotNull(savedBook);

Assert.Equal(book.Title, savedBook.Title);

Assert.Equal(book.Author, savedBook.Author);

Assert.Equal(book.PublishDate, savedBook.PublishDate);

}

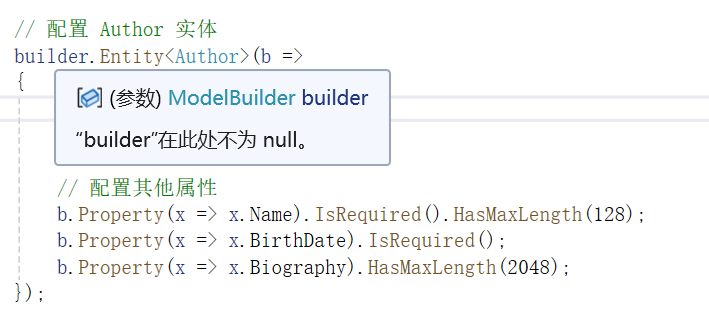
}

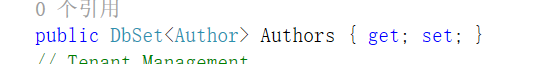
通过这些单元测试和集成测试，我们可以确保 Book 实体及其相关服务的功能正确，并且各个层次之间的集成和数据访问逻辑正确。

创建作者实体类



映射表到数据库





实现自动审计日志和软删除（软删除及他表中显示删除但数据库中有个字段记录后台可查到以删除数据）