.NET Conf China 2025

改变世界 改变自己

2025 年 11 月 30 日 | 中国 上海

.NET Conf China 2025
改变世界 改变自己

.NET 企业级开发新范式

CleanDDD + AI Agent

肖伟宇 2025.11.30

我是谁

- 微软最有价值专家（Developer Technologies方向）
- FireUG 技术社区组织者之一（B站频道5万粉）
- CleanDDD 创始人
- Java 转 .NET 推广人

AI时代前

软件工程的挑战是什么

屎山，屎山，还是屎山

通常是怎么解决屎山的

- 能用就行 （虽然是屎，顶饿就行）
- 改不动就推翻重来 （打不过我还躲不过？）

AI时代带来了什么变化

AI带来的变化

- 代码生成速度急速增快（堆屎山的速度大大提高）
- AI生成的代码越来越懒得看（看不懂就让AI自己Review）

结果就是

代码更快速地腐化

迭代速度更快速地衰减

系统地掌控力更快速地流失

我们如何应对？

企业级软件开发范式

为什么是.NET

满血ORM EF Core

- 支持变更跟踪，优雅实现满血仓储模式
- LINQ 几乎屏蔽繁琐的SQL

事件驱动框架

- 领域事件实现框架中介者模式 MediatR 等
- 事件最终一致性 Outbox 模型 CAP组件

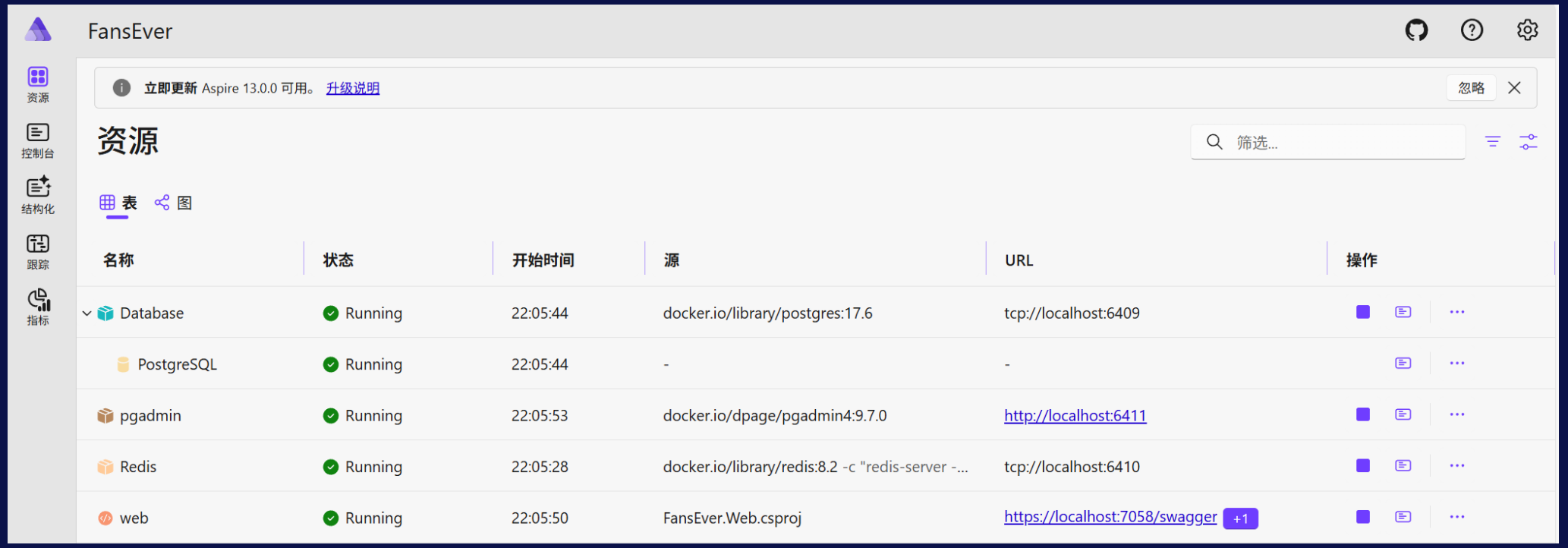
高性能

原生值类型，数值类数据处理性能开销低

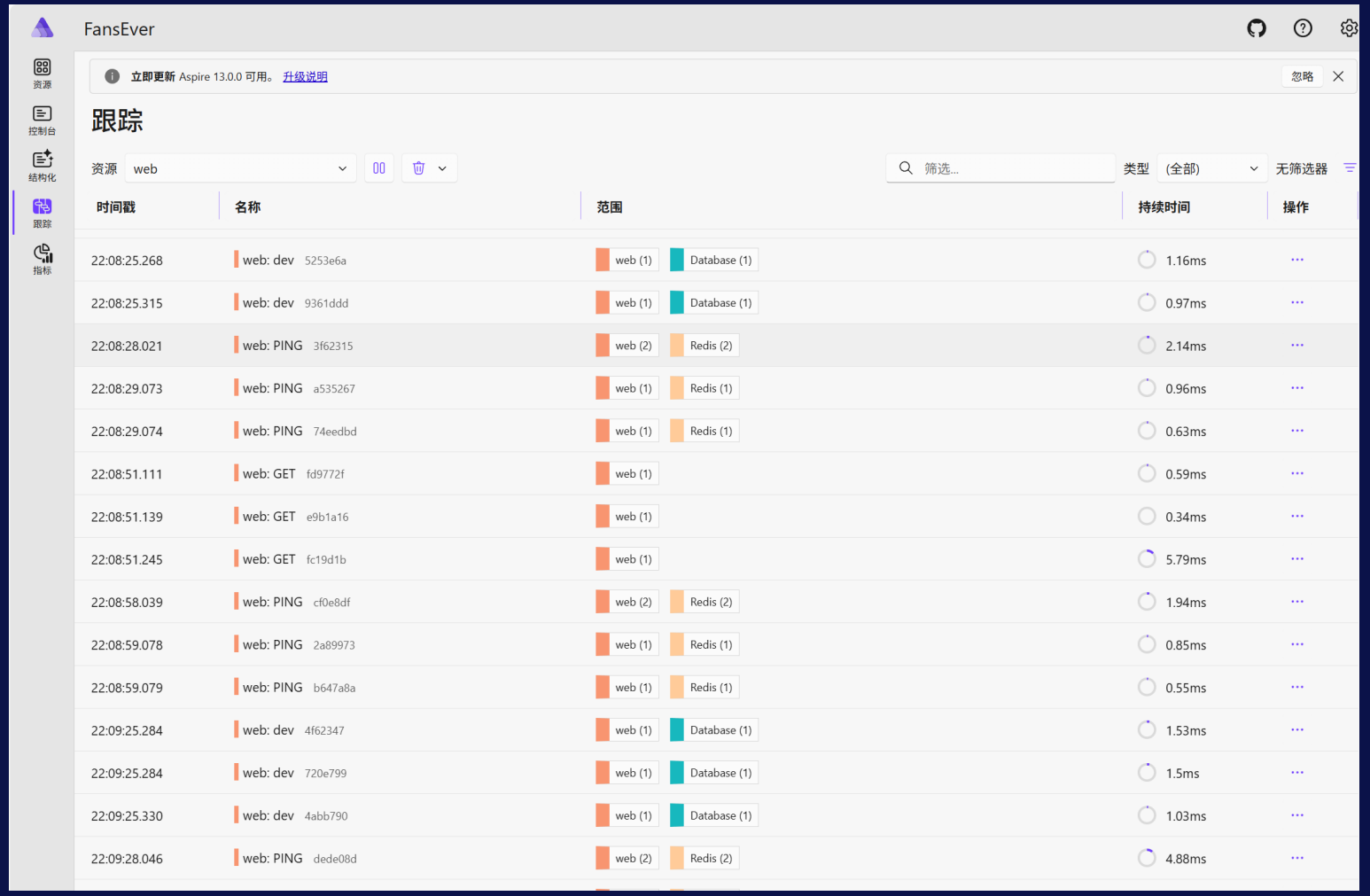
像写同步代码一样写异步代码

无侵入分库分表方案 ShardingCore等

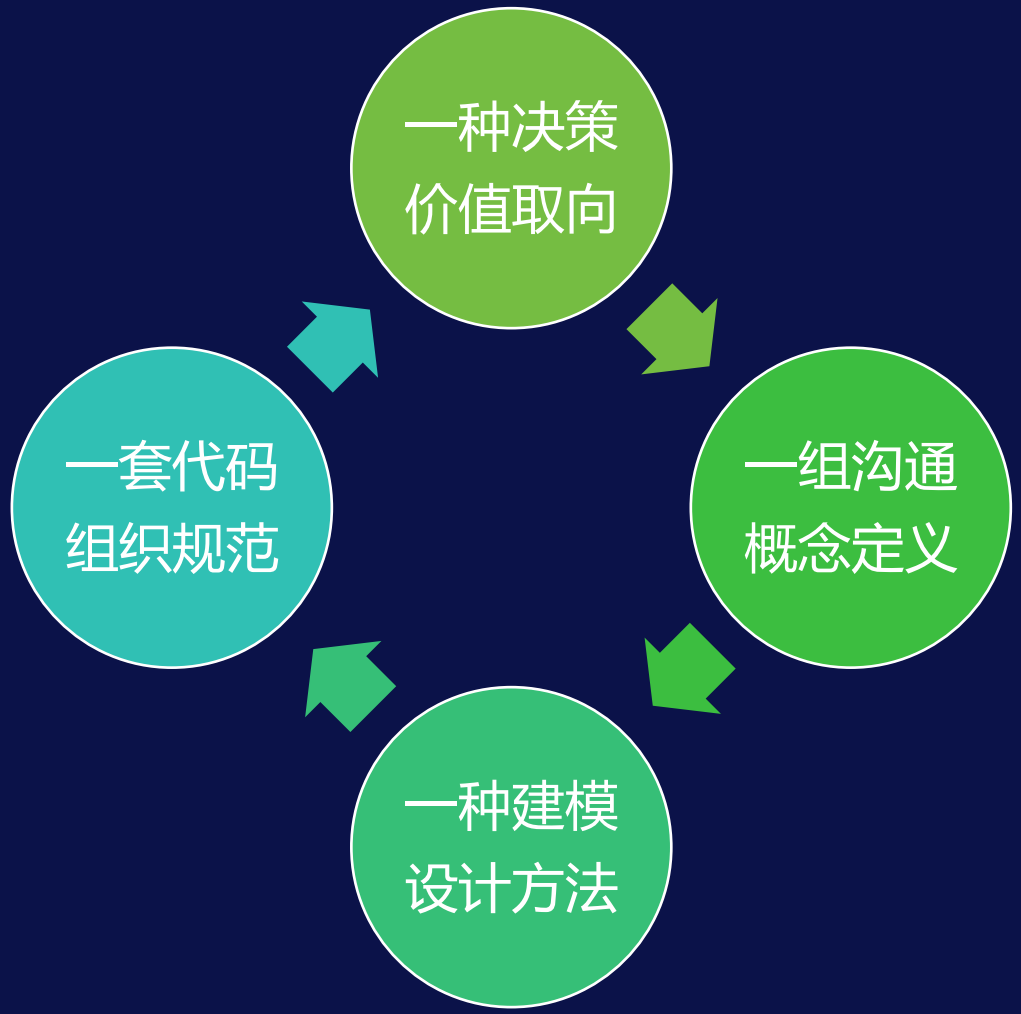
可视化开发调试体验


FansEver
资源
控制台
结构化
跟踪
指标
立即更新 Aspire 13.0.0 可用。 升级说明
忽略
资源
筛选...
表
图
名称
状态
开始时间
源
URL
操作
Database
Running
22:05:44
docker.io/library/postgres:17.6
tcp://localhost:6409
PostgreSQL
Running
22:05:44
-
-
pgadmin
Running
22:05:53
docker.io/dpage/pgadmin4:9.7.0
http://localhost:6411
Redis
Running
22:05:28
docker.io/library/redis:8.2 -c "redis-server -...
tcp://localhost:6410
web
Running
22:05:50
FansEver.Web.csproj
https://localhost:7058/swagger
+1

可视化开发调试体验


FansEver
资源
控制台
结构化
跟踪
指标
立即更新 Aspire 13.0.0 可用。 升级说明
忽略
跟踪
资源 web
筛选...
类型 (全部)
无筛选器
时间戳 名称 范围 持续时间 操作
22:08:25.268 web: dev 5253e6a web (1) Database (1) 1.16ms
22:08:25.315 web: dev 9361ddd web (1) Database (1) 0.97ms
22:08:28.021 web: PING 3f62315 web (2) Redis (2) 2.14ms
22:08:29.073 web: PING a535267 web (1) Redis (1) 0.96ms
22:08:29.074 web: PING 74eedbd web (1) Redis (1) 0.63ms
22:08:51.111 web: GET fd9772f web (1) 0.59ms
22:08:51.139 web: GET e9b1a16 web (1) 0.34ms
22:08:51.245 web: GET fc19d1b web (1) 5.79ms
22:08:58.039 web: PING cf0e8df web (2) Redis (2) 1.94ms
22:08:59.078 web: PING 2a89973 web (1) Redis (1) 0.85ms
22:08:59.079 web: PING b647a8a web (1) Redis (1) 0.55ms
22:09:25.284 web: dev 4f62347 web (1) Database (1) 1.53ms
22:09:25.284 web: dev 720e799 web (1) Database (1) 1.5ms
22:09:25.330 web: dev 4abb790 web (1) Database (1) 1.03ms
22:09:28.046 web: PING dede08d web (2) Redis (2) 4.88ms

什么是CleanDDD



CleanDDD的价值取向

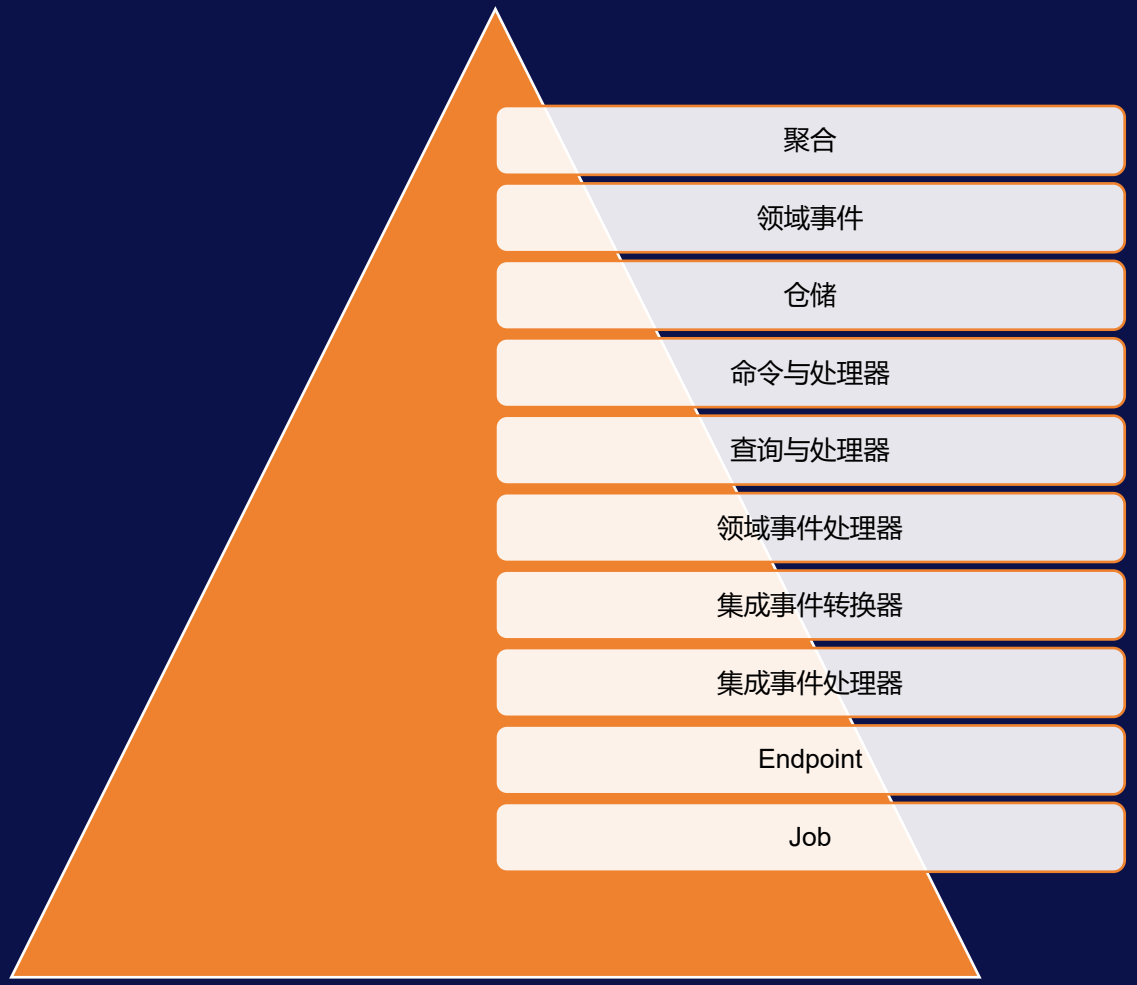
保持

边界明确是最重要的事

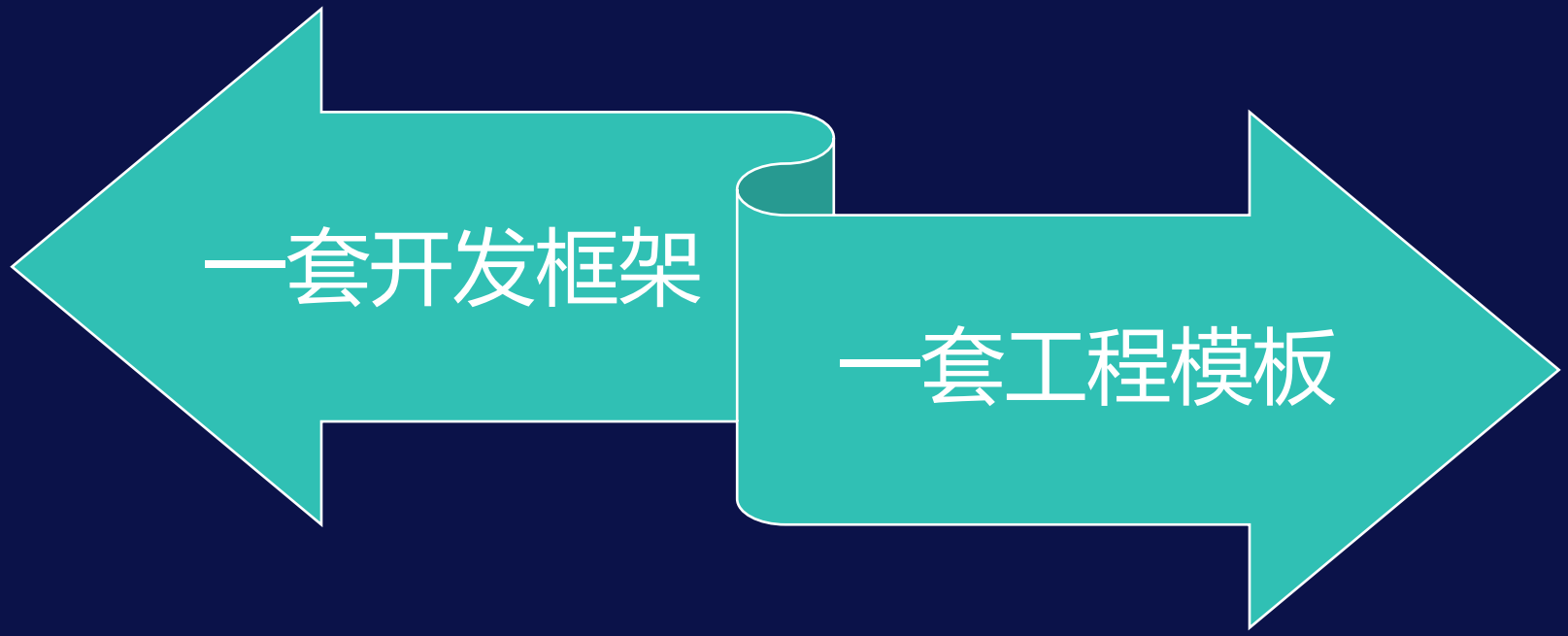
CleanDDD的核心原则

- 聚合之间不相互依赖
- 不跨聚合Join查询
- 用事件驱动处理聚合之间相互影响

CleanDDD的核心概念



.NET + CleanDDD 的开发体验



工程模板

如何使用

安装模板

```
dotnet new install NetCorePal.Template
```

安装Preview版本

```
dotnet new install NetCorePal.Template::<package-version>"

# 例如
dotnet new install NetCorePal.Template::3.0.0
```

工程模板

使用示例

```
# 使用默认配置（.NET 9.0 + MySQL + RabbitMQ）
dotnet new netcorepal-web -n My.Project.Name

# 使用 .NET 8.0 框架
dotnet new netcorepal-web -n My.Project.Name --Framework net8.0
# 或使用短参数
dotnet new netcorepal-web -n My.Project.Name -F net8.0

# 使用 SQL Server 数据库
dotnet new netcorepal-web -n My.Project.Name --Database SqlServer
# 或使用短参数
dotnet new netcorepal-web -n My.Project.Name -D SqlServer

# 使用 PostgreSQL 数据库
dotnet new netcorepal-web -n My.Project.Name --Database PostgreSQL

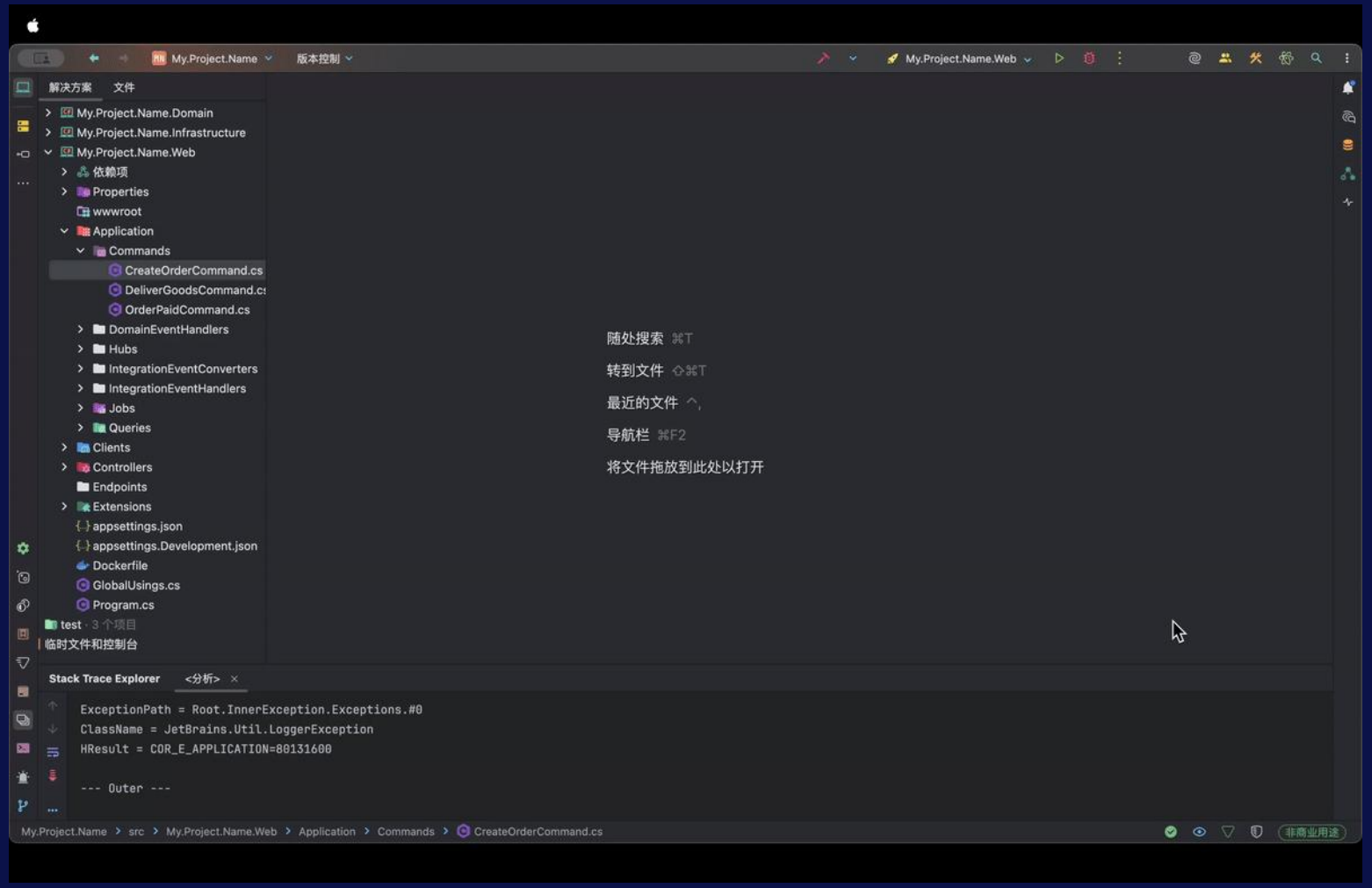
# 使用 SQLite 数据库（轻量级文件数据库，适合开发和测试）
dotnet new netcorepal-web -n My.Project.Name --Database Sqlite
# 或使用短参数
dotnet new netcorepal-web -n My.Project.Name -D Sqlite
```

工程模板

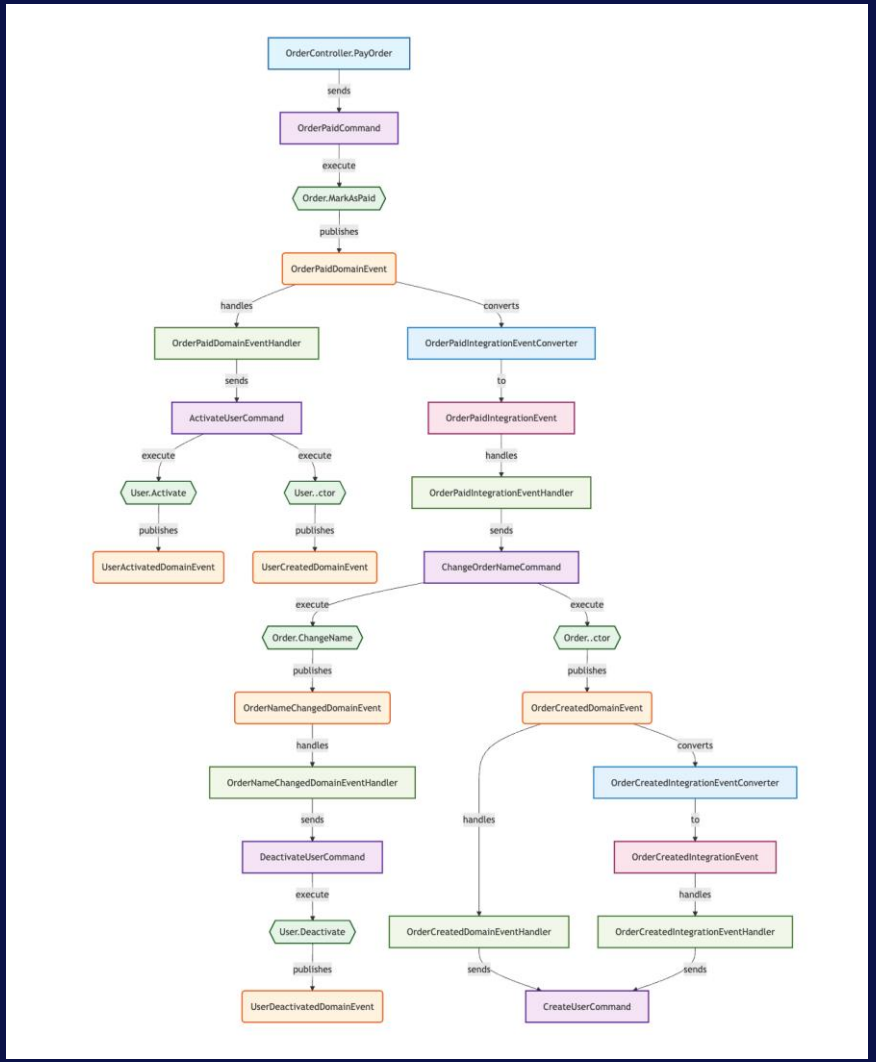
可用参数

参数	短参数	说明	可选值	默认值
`--Framework`	`-F`	目标 .NET 框架版本	`net8.0`, `net9.0`, `net10.0`	`net9.0`
`--Database`	`-D`	数据库提供程序	`MySql`, `SqlServer`, `PostgreSQL`, `Sqlite`	`MySql`
`--MessageQueue`	`-M`	消息队列提供程序	`RabbitMQ`, `Kafka`, `AzureServiceBus`, `AmazonSQS`, `NATS`, `RedisStreams`, `Pulsar`	`RabbitMQ`
`--UseAspire`	`-U`	启用 Aspire Dashboard 支持	`true`, `false`	`false`

代码快捷键


My.Project.Name
版本控制
My.Project.Name.Web
解决方案
文件
My.Project.Name.Domain
My.Project.Name.Infrastructure
My.Project.Name.Web
依赖项
Properties
wwwroot
Application
Commands
CreateOrderCommand.cs
DeliverGoodsCommand.cs
OrderPaidCommand.cs
DomainEventHandlers
Hubs
IntegrationEventConverters
IntegrationEventHandlers
Jobs
Queries
Clients
Controllers
Endpoints
Extensions
appsettings.json
appsettings.Development.json
Dockerfile
GlobalUsings.cs
Program.cs
test · 3 个项目
临时文件和控制台
随处搜索 ⌘T
转到文件 ⇧⌘T
最近的文件 ^,
导航栏 ⌘F2
将文件拖放到此处以打开
Stack Trace Explorer
<分析>
ExceptionPath = Root.InnerException.Exceptions.#0
ClassName = JetBrains.Util.LoggerException
HResult = COR_E_APPLICATION=80131600
--- Outer ---
My.Project.Name › src › My.Project.Name.Web › Application › Commands › CreateOrderCommand.cs
非商业用途

模型可视化


OrderController.PayOrder
sends
OrderPaidCommand
execute
Order.MarkAsPaid
publishes
OrderPaidDomainEvent
handles
converts
OrderPaidDomainEventHandler
OrderPaidIntegrationEventConverter
sends
to
ActivateUserCommand
OrderPaidIntegrationEvent
execute
execute
handles
User.Activate
User..ctor
OrderPaidIntegrationEventHandler
publishes
publishes
sends
UserActivatedDomainEvent
UserCreatedDomainEvent
ChangeOrderNameCommand
execute
execute
Order.ChangeName
Order..ctor
publishes
publishes
OrderNameChangedDomainEvent
OrderCreatedDomainEvent
handles
converts
OrderNameChangedDomainEventHandler
OrderCreatedIntegrationEventConverter
handles
sends
to
DeactivateUserCommand
OrderCreatedIntegrationEvent
execute
handles
User.Deactivate
OrderCreatedDomainEventHandler
OrderCreatedIntegrationEventHandler
publishes
sends
sends
UserDeactivatedDomainEvent
CreateUserCommand

聚合

```
namespace FansEver.Domain.AggregatesModel.OrderAggregate;

1 个引用
public partial record OrderId : IGuidStronglyTypedId;
```

```
2 个引用
10  public partial class Order : Entity<OrderId>, IAggregateRoot
11  {
        0 个引用
12      protected Order()…
15
        0 个引用
16      public Order(string name, int count)
17      {
18          this.Name = name;
19          this.Count = count;
20          this.AddDomainEvent(new OrderCreatedDomainEvent(this));
21      }
22
        2 个引用
23      public bool Paid { get; private set; } = false;
24
        1 个引用
25      public string Name { get; private set; } = string.Empty;
26
        1 个引用
27      public int Count { get; private set; }
28
        0 个引用
29      public RowVersion RowVersion { get; private set; } = new RowVersion();
30
        0 个引用
31      public UpdateTime UpdateTime { get; private set; } = new UpdateTime(DateTimeOffset.UtcNow);
32
        0 个引用
33      public void OrderPaid()
34      {
35          if (Paid)
36          {
37              throw new KnownException("Order has been paid");
38          }
39          else
40          {
41              this.Paid = true;
42              this.AddDomainEvent(new OrderPaidDomainEvent(this));
43          }
44      }
45  }
```

领域事件

```
1   using FansEver.Domain.AggregatesModel.OrderAggregate;
2
3   namespace FansEver.Domain.DomainEvents;
4
    0 个引用
5   public record OrderCreatedDomainEvent(Order Order) : IDomainEvent;
6
    0 个引用
7   public record OrderPaidDomainEvent(Order Order) : IDomainEvent;
8
```

仓储

```
namespace FansEver.Infrastructure.Repositories;

1 个引用
public interface IOrderRepository : IRepository<Order, OrderId>
{
}

0 个引用
public class OrderRepository(ApplicationDbContext context) :
    RepositoryBase<Order, OrderId, ApplicationDbContext>(context), IOrderRepository
{
}
```

命令与处理器

```
3 个引用
public record CreateOrderCommand(string Name, int Price, int Count) : ICommand<OrderId>;

1 个引用
public class CreateOrderCommandValidator : AbstractValidator<CreateOrderCommand>
{
    0 个引用
    public CreateOrderCommandValidator()
    {
        RuleFor(x => x.Name).NotEmpty().MaximumLength(10).WithErrorCode("name error code");
        RuleFor(x => x.Price).InclusiveBetween(18, 60).WithErrorCode("price error code");
    }
}

1 个引用
public class CreateOrderCommandHandler(IOrderRepository orderRepository, ILogger<CreateOrderCommandHandler> logger)
                    : ICommandHandler<CreateOrderCommand, OrderId>
{

    0 个引用
    public async Task<OrderId> Handle(CreateOrderCommand request, CancellationToken cancellationToken)
    {
        var order = new Order(request.Name, request.Count);
        order = await orderRepository.AddAsync(order, cancellationToken);
        logger.LogInformation("order created, id:{OrderId}", order.Id);
        return order.Id;
    }
}
```

命令与处理器

```
    4 个引用
6   public record PayOrderCommand(OrderId OrderId) : ICommand;
7
    0 个引用
8   public class PayOrderCommandLock : ICommandLock<PayOrderCommand>
9   {
        0 个引用
10      public Task<CommandLockSettings> GetLockKeysAsync(PayOrderCommand command,
11          CancellationToken cancellationToken = new CancellationToken())
12      {
13          return Task.FromResult(command.OrderId.ToCommandLockSettings());
14      }
15  }
16
    0 个引用
17  public class PayOrderCommandHandler(IOrderRepository orderRepository) : ICommandHandler<PayOrderCommand>
18  {
        0 个引用
19      public async Task Handle(PayOrderCommand request, CancellationToken cancellationToken)
20      {
21          var order = await orderRepository.GetAsync(request.OrderId, cancellationToken) ??
22                      throw new KnownException($"未找到订单, OrderId = {request.OrderId}");
23          order.OrderPaid();
24      }
25  }
```

查询与处理器

```
    2 个引用
7   public record QueryOrder(OrderId Id) : IQuery<QueryOrderResult>;
8
    4 个引用
9   public record QueryOrderResult(OrderId Id, string Name, int Count);
10
    0 个引用
11  public class OrderQueryHandler(ApplicationDbContext applicationDbContext)
12      : IQueryHandler<QueryOrder, QueryOrderResult>
13  {
        0 个引用
14      public async Task<QueryOrderResult> Handle(QueryOrder request, CancellationToken cancellationToken)
15      {
16          var result = await applicationDbContext.Orders.Where(p => p.Id == request.Id)
17                  .Select(p => new QueryOrderResult(p.Id, p.Name, p.Count))
18                  .FirstOrDefaultAsync(cancellationToken);
19          return result ?? throw new KnownException("Order not found");
20      }
21  }
22
```

领域事件处理器

```
0 个引用
public class OrderCreatedDomainEventHandlerForDeliverGoods(IMediator mediator)
        : IDomainEventHandler<OrderCreatedDomainEvent>
{
    0 个引用
    public Task Handle(OrderCreatedDomainEvent notification, CancellationToken cancellationToken)
    {
        return mediator.Send(new DeliverGoodsCommand(notification.Order.Id), cancellationToken);
    }
}
```

集成事件

```
3   namespace FansEver.Web.Application.IntegrationEvents;
4
    0 个引用
5   public record OrderPaidIntegrationEvent(OrderId OrderId);
6
```

集成事件转换器

```
    0 个引用
6   public class OrderPaidIntegrationEventConverter
7       : IIntegrationEventConverter<OrderPaidDomainEvent, OrderPaidIntegrationEvent>
8   {
        0 个引用
9       public OrderPaidIntegrationEvent Convert(OrderPaidDomainEvent domainEvent)
10      {
11          return new OrderPaidIntegrationEvent(domainEvent.Order.Id);
12      }
13  }
```

集成事件处理器

```
0 个引用
public class OrderPaidIntegrationEventHandlerForDeliverGoods(IMediator mediator)
    : IIntegrationEventHandler<OrderPaidIntegrationEvent>
{
    0 个引用
    public async Task HandleAsync(OrderPaidIntegrationEvent eventData, CancellationToken cancellationToken = default)
    {
        var cmd = new DeliverGoodsCommand(eventData.OrderId);
        _ = await mediator.Send(cmd, cancellationToken);
    }
}
```

Endpoints

```
    2 个引用
10  public record CreateOrderRequest(string Name, int Price, int Count);
11
12  [Tags("Orders")]
13  [HttpPost("/api/order")]
14  [AllowAnonymous]
    0 个引用
15  public class CreateOrderEndpoint(IMediator mediator) : Endpoint<CreateOrderRequest, ResponseData<OrderId>>
16  {
        0 个引用
17      public override async Task HandleAsync(CreateOrderRequest req, CancellationToken ct)
18      {
19          var cmd = new CreateOrderCommand(req.Name, req.Price, req.Count);
20          var id = await mediator.Send(cmd, ct);
21          await Send.OkAsync(id.AsResponseData(), cancellation: ct);
22      }
23  }
```

AI 时代是怎样的？

天然 AI Agent 友好

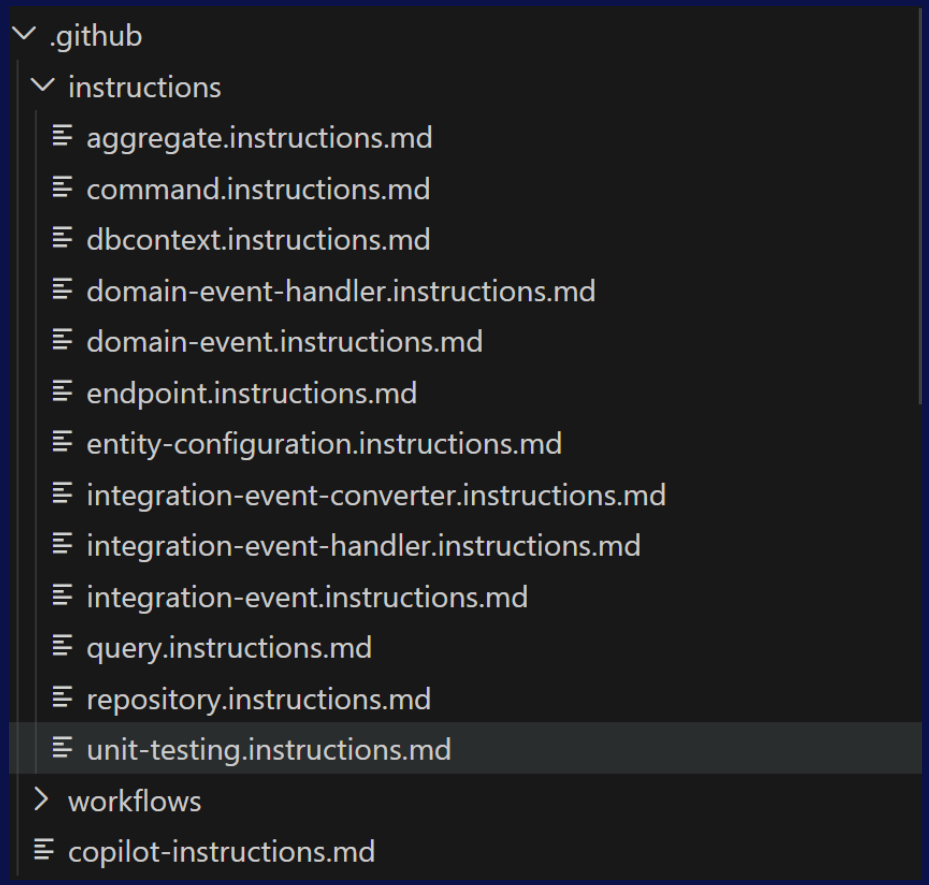
- 充分明确的代码规范
- 充分明确的设计原则
- 充分内聚的上下文信息
- 充分明确的上下文边界

AI时代，企业级软件开发新范式

人能够轻松掌控并实践

AI也能够掌握，甚至执行得更好

Github Copilot Agent 支持



Github Copilot Agent 支持

```
---
applyTo: "src/FansEver.Domain/AggregatesModel/**/*.cs"
---

# 聚合与强类型ID开发指南

## 开发原则

### 必须

- **聚合根定义**:
  - 聚合内必须有一个且只有一个聚合根。
  - 必须继承 `Entity<TId>` 并实现 `IAggregateRoot` 接口。
  - 必须有 `protected` 无参构造器供 EF Core 使用。
  - 所有属性使用 `private set`，并显式设置默认值。
  - 状态改变时发布领域事件，使用 `this.AddDomainEvent()`。
  - `Deleted` 属性表示软删除状态。
  - `RowVersion` 属性用于乐观并发控制。
- **强类型ID定义**:
  - 必须使用 `IInt64StronglyTypedId` 或 `IGuidStronglyTypedId`，优先使用 `IGuidStronglyTypedId`。
  - 必须使用 `partial record` 声明，让框架生成具体实现。
  - 必须是 `public` 类型。
  - 必须与聚合/实体在同一个文件中定义。
  - 命名格式必须为 `{EntityName}Id`。
- **子实体定义**:
  - 必须是 `public` 类。
  - 必须有一个无参构造器。
  - 必须有一个强类型ID，推荐使用 `IGuidStronglyTypedId`。
  - 必须继承自 `Entity<TId>`，并实现 `IEntity` 接口。
  - 聚合内允许多个子实体。
```

Github Copilot Agent 支持

````
## 文件命名规则

- 类文件应放置在 `src/FansEver.Domain/AggregatesModel/{AggregateName}Aggregate/` 目录下。
- 例如 `src/FansEver.Domain/AggregatesModel/UserAggregate/User.cs`。
- 每个聚合在独立文件夹中。
- 聚合根类名与文件名一致。
- 强类型ID与聚合根定义在同一文件中。

## 代码示例

文件: `src/FansEver.Domain/AggregatesModel/UserAggregate/User.cs`

```csharp
using FansEver.Domain.DomainEvents; // 必需：引用领域事件

namespace FansEver.Domain.AggregatesModel.UserAggregate;

// 强类型ID定义 - 与聚合根在同一文件中
public partial record UserId : IGuidStronglyTypedId;

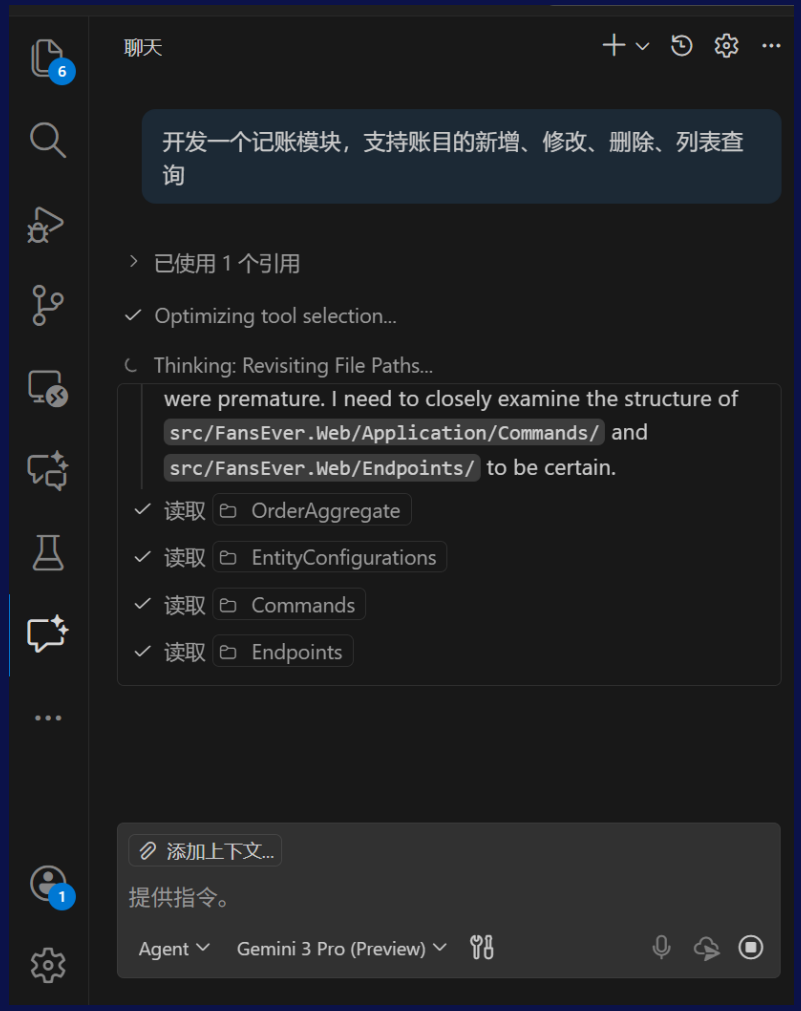
// 聚合根定义
public class User : Entity<UserId>, IAggregateRoot
{
    protected User() { }

    public User(string name, string email)
    {
        // 不手动设置ID，由EF Core值生成器自动生成
        Name = name;
        Email = email;
        this.AddDomainEvent(new UserCreatedDomainEvent(this));
    }

    #region Properties

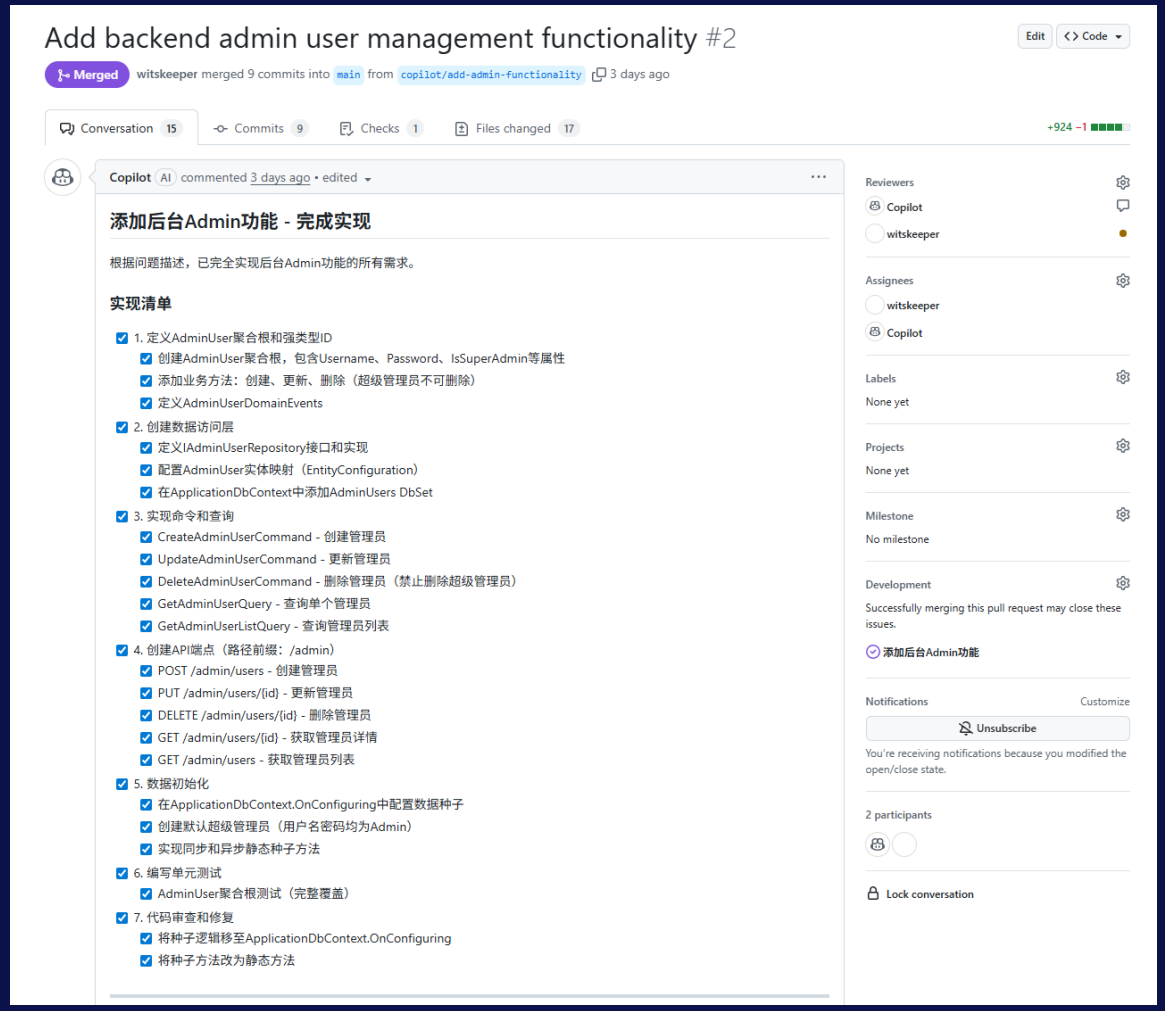
    public string Name { get; private set; } = string.Empty;
````
````

VS Code Agent 模式



Github Action Agent 模式

Github Action Agent 模式



Add backend admin user management functionality #2

Edit · Code

Merged witskeeper merged 9 commits into main from copilot/add-admin-functionality 3 days ago

Conversation 15 · Commits 9 · Checks 1 · Files changed 17 · +924 −1

Copilot AI commented 3 days ago • edited

添加后台Admin功能 - 完成实现

根据问题描述，已完全实现后台Admin功能的所有需求。

实现清单

- [x] 1. 定义AdminUser聚合根和强类型ID
  - [x] 创建AdminUser聚合根，包含Username、Password、IsSuperAdmin等属性
  - [x] 添加业务方法：创建、更新、删除（超级管理员不可删除）
  - [x] 定义AdminUserDomainEvents
- [x] 2. 创建数据访问层
  - [x] 定义IAdminUserRepository接口和实现
  - [x] 配置AdminUser实体映射（EntityConfiguration）
  - [x] 在ApplicationDbContext中添加AdminUsers DbSet
- [x] 3. 实现命令和查询
  - [x] CreateAdminUserCommand - 创建管理员
  - [x] UpdateAdminUserCommand - 更新管理员
  - [x] DeleteAdminUserCommand - 删除管理员（禁止删除超级管理员）
  - [x] GetAdminUserQuery - 查询单个管理员
  - [x] GetAdminUserListQuery - 查询管理员列表
- [x] 4. 创建API端点（路径前缀：/admin）
  - [x] POST /admin/users - 创建管理员
  - [x] PUT /admin/users/{id} - 更新管理员
  - [x] DELETE /admin/users/{id} - 删除管理员
  - [x] GET /admin/users/{id} - 获取管理员详情
  - [x] GET /admin/users - 获取管理员列表
- [x] 5. 数据初始化
  - [x] 在ApplicationDbContext.OnConfiguring中配置数据种子
  - [x] 创建默认超级管理员（用户名密码均为Admin）
  - [x] 实现同步和异步静态种子方法
- [x] 6. 编写单元测试
  - [x] AdminUser聚合根测试（完整覆盖）
- [x] 7. 代码审查和修复
  - [x] 将种子逻辑移至ApplicationDbContext.OnConfiguring
  - [x] 将种子方法改为静态方法

Reviewers: Copilot, witskeeper

Assignees: witskeeper, Copilot

Labels: None yet

Projects: None yet

Milestone: No milestone

Development: Successfully merging this pull request may close these issues. 添加后台Admin功能

Notifications: Customize — Unsubscribe — You're receiving notifications because you modified the open/close state.

2 participants

Lock conversation

.NET Conf China 2025
改变世界 改变自己

.NET + CleanDDD + AI Agent

.NET Conf China 2025
改变世界 改变自己

THANK YOU

B站频道
@FireUG技术社区

B站频道
@老肖想当外语大佬