

第27期-设计微服务CQRS与DDD模式

2020年5月9日 9:38

命令和查询责任分离 (CQRS) 模式

CQRS 是一种分离数据读取与写入模型的体系结构模式，基本思想是可以将系统操作划分为命令和查询两个界限明显的类别。

优势：独立缩放，优化的数据架构，安全性，将问题分离，查询更简单。

挑战：复杂性，重复消息，最终一致性。

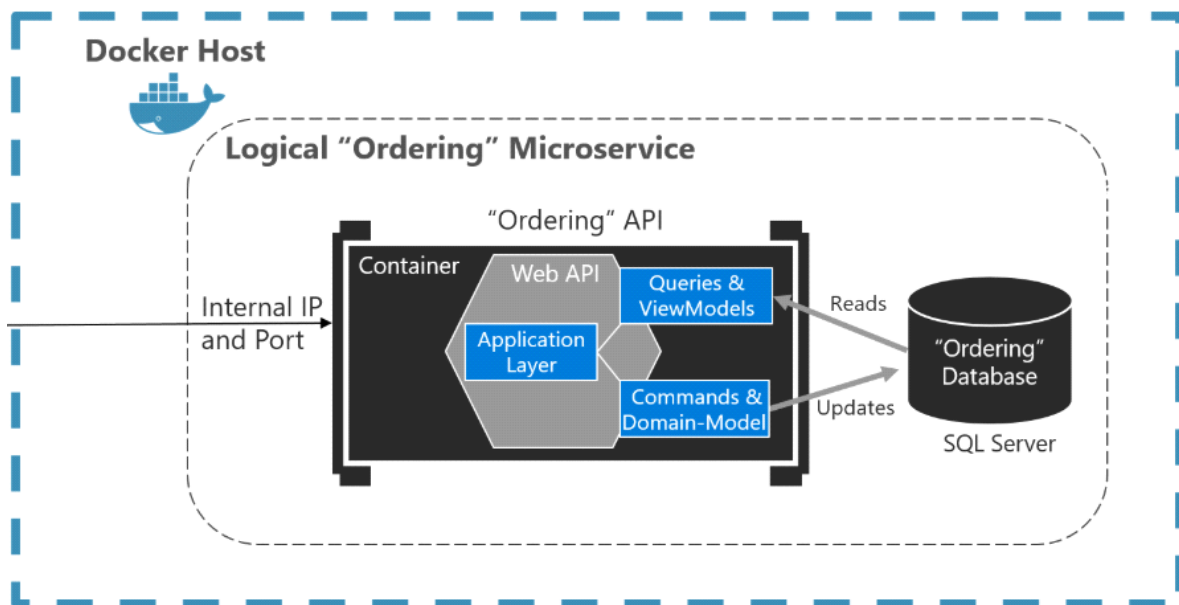


在微服务中应用简化后的 CQRS 和 DDD 模式

eShopOnContainers 引用应用程序处的订单微服务设计基于 CQRS 原则。但是，它使用最简单的方法：只需将查询与命令分离，且执行这两种操作时使用相同的数据库。

这些模式的本质，以及这里的要点是，查询是幂等的：无论查询一个系统多少次，该系统的状态都不会改变。换言之，查询没有副作用。

因此，可以使用不同的“读取”数据模型而不是事务逻辑的“写入”域模型，尽管排序微服务使用的是相同的数据库。因此，这是简化的 CQRS 方法。



事件溯源和 CQRS 模式

CQRS 模式通常与事件溯源模式一起使用。基于 CQRS 的系统使用分离的读取和写入数据模型，每个模型针对相关任务定制，并且通常位于物理分离存储中。

当与事件源模式一起使用时，事件的存储是写入模型，是官方的信息来源。基于 CQRS 系统的读取模型提供数据的具体化视图，通常是高度非规范化视图。针对应用程序的接口和显示要求定制这些视图，这有助于最大限度地提高显示和查询性能。

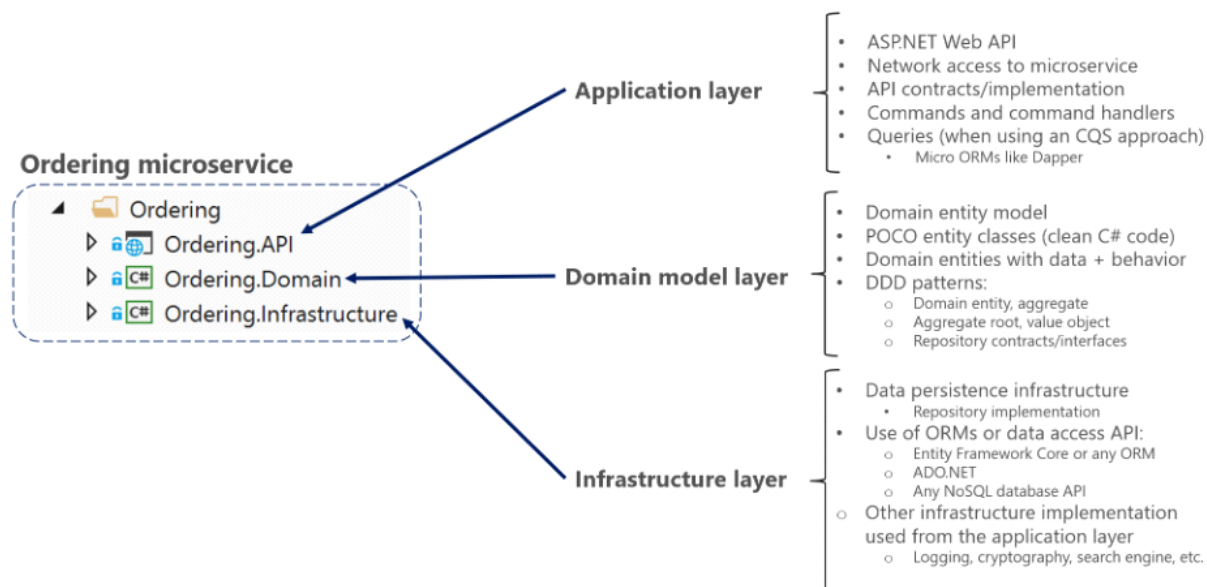
设计面向 DDD 领域驱动设计的微服务

理解充血模型与贫血模型，DDD 推荐使用充血模型，保持相对较小的微服务上下文边界，层是一种逻辑项目，与服务部署无关。实体与 ViewModel 的关系与区分。

领域模型层：负责表示业务概念、有关业务状况的信息和业务规则。反映业务状况的状态是通过这个层进行控制和利用的，但有关状态存储的具体技术细节则由基础结构负责实施。这一层是业务软件的核心。领域模型层不能直接依赖于任何基础结构框架。

应用程序层：定义软件要完成的作业并指导富有表现力的域对象解决问题。这一层负责执行对业务具有意义的任务或与其他系统的应用层进行交互时需执行的任务。这一层很“薄”。它不包含业务规则或知识，仅针对下一层中域对象之间的协作，协调任务和委派工作。它不具有反映业务状况的状态，但它可以具有状态，用于反映用户或程序的任务的进度。

基础结构层：是关于如何将最初存放在域实体中的数据（内存中）持久保存在数据库或另一个持久性存储区中。



同一标识可以跨多个绑定上下文或微服务建模，DDD 中的域实体必须实现与实体数据（在内存中访问的对象）相关的域逻辑或行为，**值对象模式（无标识不可变）**。值对象在关系数据库和 ORM 中很难管理，可用 Ef Core 中的拥有实体类型间接实现值对象模式。

聚合根，也称为根实体或主实体。此外，它还可以有多个子实体和值对象，所有实体和对象一起共同实现所需的行为和事务。聚合根的目的是确保聚合的一致性。

在域模型层中设计验证

验证条件和引发异常，在模型中根据数据注释使用验证属性，双重验证，客户端验证。

<https://github.com/FluentValidation/FluentValidation>

领域事件的设计与实现

领域事件是一个领域模型中极其重要的部分，用来表示领域中发生的事件。忽略不相关的领域活动，同时明确领域专家要跟踪或希望被通知的事情，或其他模型对象中的状态更改相关联。

域事件和集成事件，前者进程内通信，支持同步和异步，可用 MediatR 库实现，后者使用分布式队列只能是异步。

参阅《eShopOnWeb整洁架构视频教程》

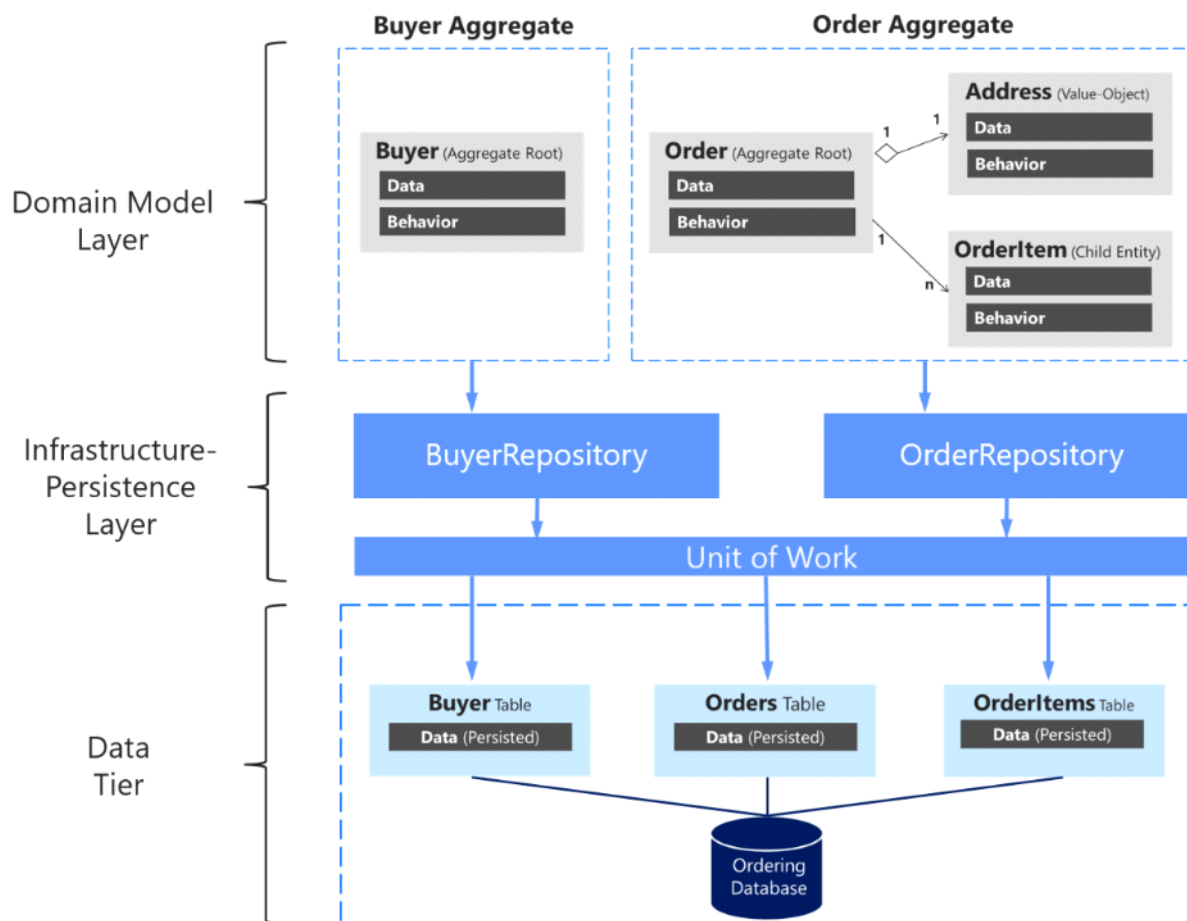
第09期-使用MediatR实现中介者发布订阅

中介者模式可避免交互对象之间的紧耦合关系，发布与订阅形式，使用 MediatR 中介者实现发布与订阅，单播与多播消息，有返回值的单播消息，无返回值的单播消息，单播消息处理器，多播通知消息，

多播通知处理器，多路广播中的分发策略设置，中介者的同步与异步调用，关于 MediatR 在 eShopOnWeb 项目中的实践应用。

设计基础结构持久性层

数据持久性组件提供对微服务边界（即微服务的数据库）内托管的数据的访问。它们包含组件（例如存储库和工作单元类）的实际实现，例如自定义实体框架 (EF) DbContext 对象。EF DbContext 实现两种模式：存储库模式和工作单元模式。



为每个聚合定义一个存储库，聚合具有事务性，工作单元将状态更改持久化，强制每个存储库使用一个聚合根，存储库模式使应用程序逻辑测试更轻松，理解存储库模式与传统数据访问类（DAL 类）模式之间的区别，存储库不应该是强制性的。

使用自定义存储库与直接使用 EF DbContext 的区别，工作单元可以被多个存储库共享，存储库与工作单元生命周期都应该为基于请求的 Scoped 周期，EF Core 中的 Hi/Lo 算法，映射字段（而非属性），阴影属性，实现查询规范模式（eShsopOnWeb）。

可将将 NoSQL 数据库用作持久性基础结构，适用于 .NET 的 MongoDB API 是基于 NuGet 包的，你需要添加到类似于下图所示的 Locations.API 项目中。

使用 SOLID 原则和依赖关系注入

单一责任原则，开闭原则，里氏替换原则，接口隔离原则，依赖倒置原则。

ASP.NET Core 提供简单的内置 IoC 容器，但也可以使用其他 IoC 容器，如 Autofac 或 Ninject。

<https://www.jianshu.com/p/1c6498da3862>

<https://www.jianshu.com/p/5f1dc9f7b57d>

使用 Web API 实现微服务应用层

如果使用 ASP.NET Core 构建微服务，应用程序层通常是 Web API 库。如果要从自定义应用程序层代码中分离来自 ASP.NET Core 的内容（其基础结构以及你的控制器），还可将应用程序层置于单独的类库，但这是可选操作。

在 .NET Core 中使用 DI 时，可能需要扫描程序集，并自动按约定注册其类型。当前 ASP.NET Core 中未提供此功能。但是，可以使用 Scrutor 库。如果需要在 IoC 容器中注册许多类型，使用该方法很方便。

<https://github.com/khellang/Scrutor>

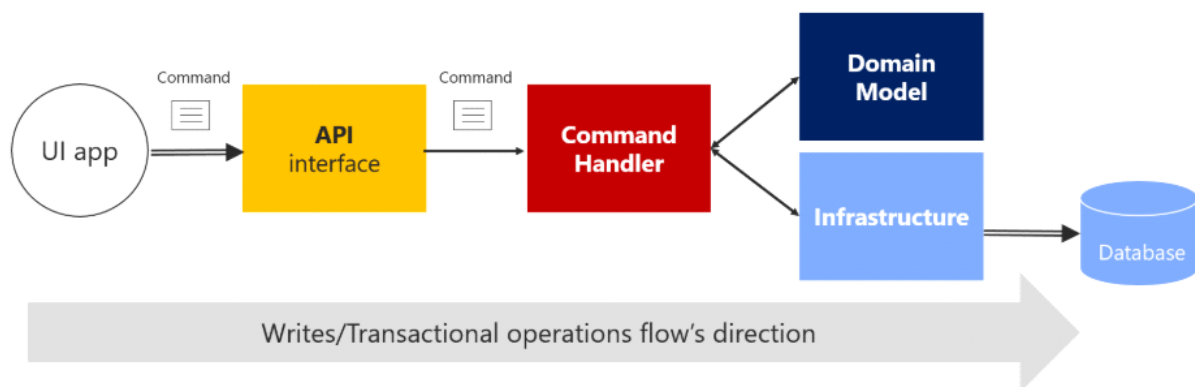
还可使用其他 IoC 容器，并将其插入 ASP.NET Core 管道，就像 eShopOnContainers（使用 Autofac）中的订购微服务一样。使用 Autofac 时通常通过模块注册类型，这可根据类型位置，在多个文件之间拆分注册类型，就像可在多个类库中分布应用程序类型一样。Autofac 还具有用于按名称约定扫描程序集和注册类型的功能。

<https://autofaccn.readthedocs.io/zh/latest>

实现命令和命令处理程序模式

命令模式在本质上与本指南之前介绍的 CQRS 模式相关。CQRS 具有两个功能。第一个功能是查询，通过 Dapper 微 ORM 使用简化的查询，我们已经在前文中介绍过了。第二个功能是命令（这是事务的起点），以及服务外的输入通道。

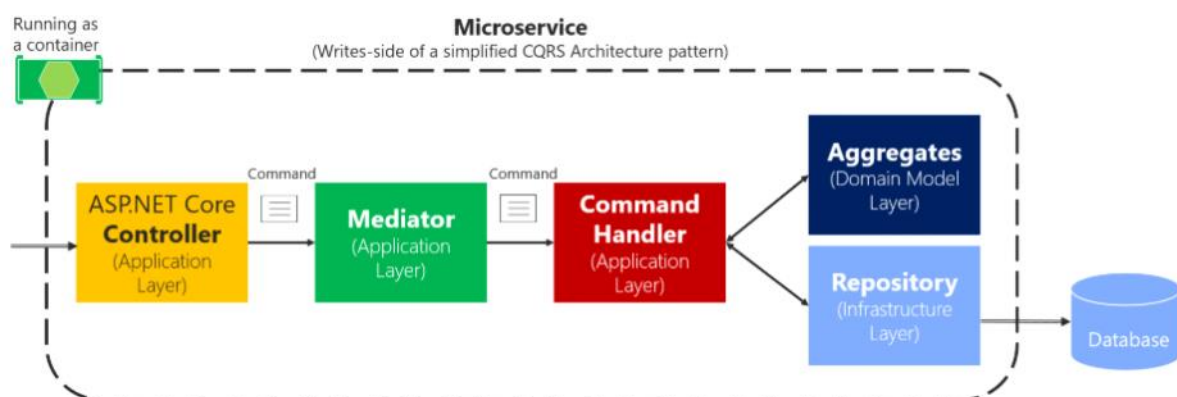
该模式基于接受客户端的命令、根据域模式规则进行处理，最后保持事务状态。



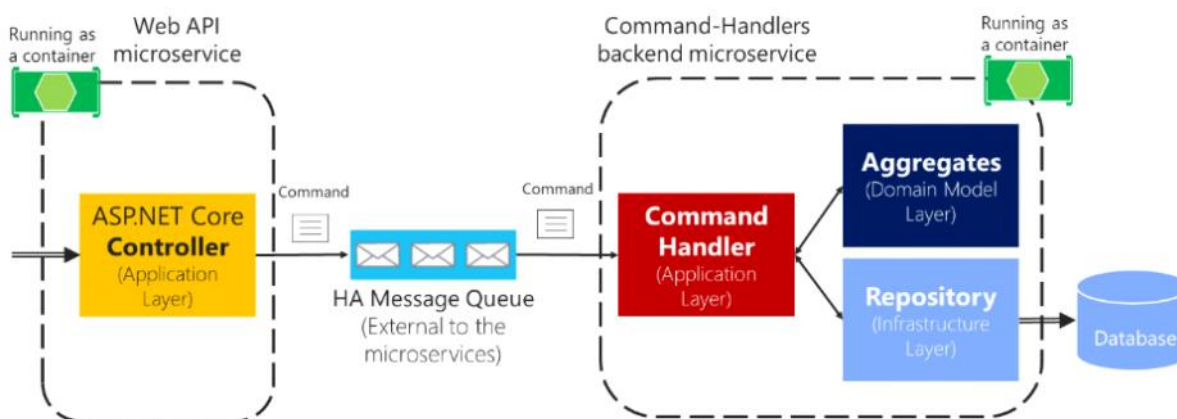
命令类，表示改变状态的操作，仅处理一次，具有命令性，通常采用命令语气使用谓词，如创建、更新、删除和审批等。与事件不同，命令不是过去发生的事实，它只是一个请求，因此可以拒绝它。

建议将命令和更新设置为幂等，可使用 X-Request-ID 模式，命令可以当作 DTO 来使用，使用命令处理程序解耦。

在命令管道中使用转存进程模式（内存中）



在命令的管道中使用消息队列（进程外）



通过转存进程模式 (MediatR) 实现命令进程管道

有关 .NET Core 中的实现，可使用多个开发源代码库来实现转存进程模式。本指南中使用的库是 MediatR 开放源代码库（由 Jimmy Bogard 创建），但也可使用其他方法。MediatR 是一个小型的简单库，可处理命令等内存中消息，同时应用修饰器或行为。

参考资料

[使用 DDD 和 CQRS 模式降低微服务中的业务复杂性](#)

[Azure 体系结构: 命令和查询责任分离 \(CQRS\) 模式](#)