## 命令查询职责分离

命令与查询是否需要分离，这一设计决策会对系统架构、限界上下文乃至领域模型直接产生影响，在领域驱动设计中，这是一个重要的战略考量。针对领域模型对象的操作往往包含命令和查询操作，但在大多数领域场景中，它们的关注点无疑是不尽相同的。命令和查询操作的差异包括：

* 查询操作没有副作用，具有幂等性；命令操作会修改状态，其中新增操作若不加约束则不具有幂等性
* 查询操作发起同步请求，需要实时返回查询结果，因而往往是阻塞式的 Request/Response 操作；命令操作可以发起异步请求，甚至可以不用返回结果，即采用非阻塞式的 Fire-and-Forget 操作
* 查询结果往往需要面向 UI 表示层，命令操作只是引起状态的变更，无需呈现操作结果
* 查询操作的频率要远远高于命令操作，而领域复杂性却又要低于命令操作

既然命令操作与查询操作存在如此多的差异，采用一致的设计方案就无法更好地应对不同的客户端请求。按照领域驱动设计的原则，针对同一个领域逻辑，应该建立一个统一的领域模型。然而，一个领域模型却可能无法同时满足具有复杂 UI 呈现与丰富领域逻辑的需求，无法同时满足具有同步实时与异步低延迟的需求；这时，就需要寻求改变，将一个领域模型按照操作类型的不同分为两个不同的模型，这正是提出命令查询职责分离模式（CQRS）的原因所在。

### CQS 模式

在代码实现层面，一个设计良好的方法需要将命令与查询分离，这就是命令查询分离（Command Query Separation，CQS）模式。提出该模式的 Bertrand Meyer 认为：

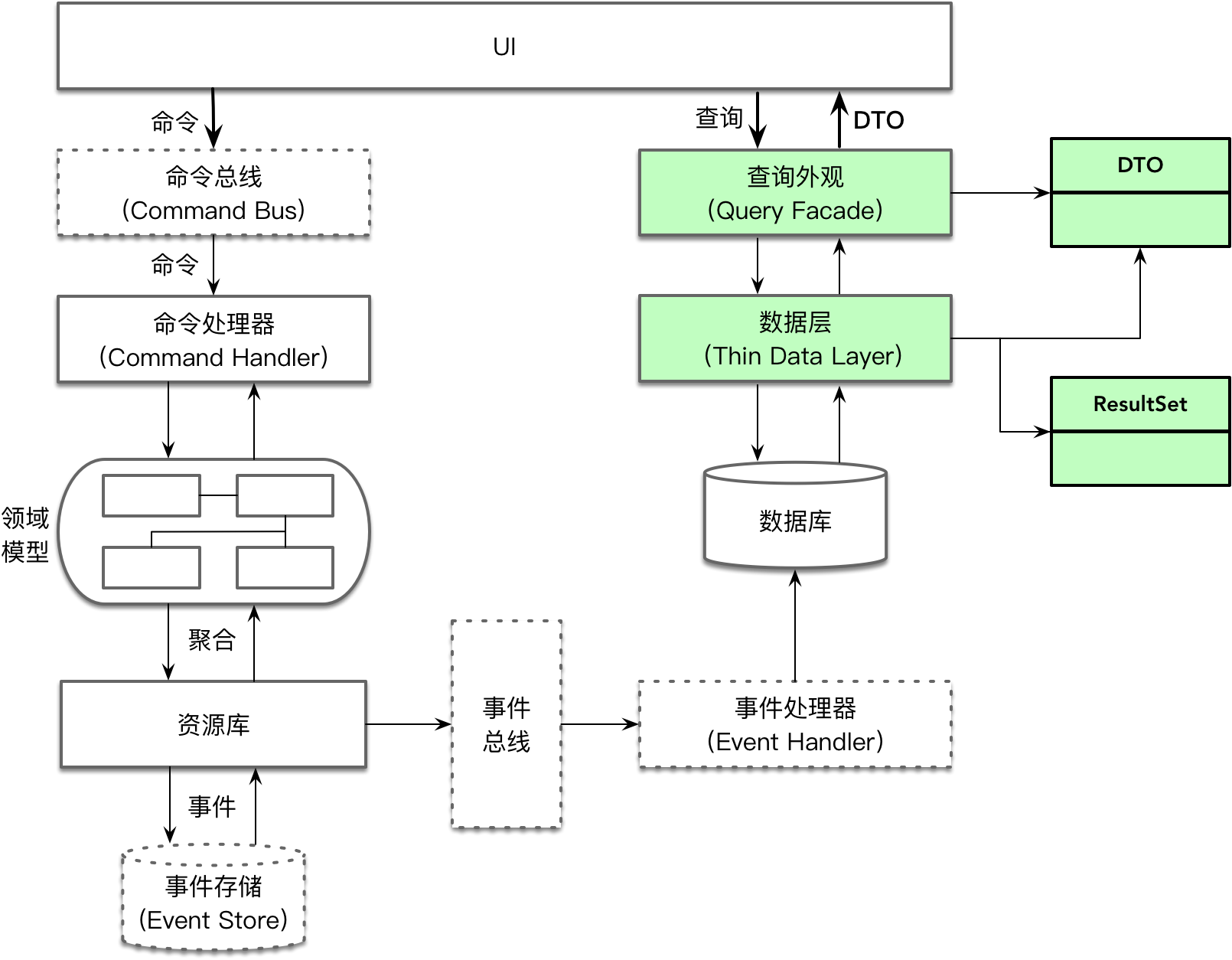
一个方法要么是执行某种动作的命令，要么是返回数据的查询，而不能两者皆是。换句话说，问题不应该对答案进行修改。更正式的解释是，一个方法只有在具有引用透明（referentially transparent）时才能返回数据，此时该方法不会产生副作用。

在代码层面分离命令与查询，目的是隔离副作用。一个没有副作用的方法就是指根据输入参数给出运算结果之外没有其他的影响，例如整数的加法方法（函数），它接收两个整数值并返回一个整数值。对于给定的两个整数值，它的返回值永远是相同的整数值。这样的方法满足“引用透明”，它要求方法不论进行了任何操作都可以用它的返回值来代替。假设一个代码块调用的都是这样满足引用透明规则的方法，执行这段代码块的过程就是用一个一个等价值进行替代的过程。这一个过程可以称之为是“等式推理（Equational Reasoning）”。

函数范式非常强调函数的无副作用，要求定义为引用透明的纯函数。对象范式对方法定义虽没有这样严格的要求，但遵循 CQS 模式仍有一定的必要性。如果将其放在架构层面来考虑，命令操作与查询操作的分离不仅仅是隔离副作用，还承担了分离领域模型、响应不同调用者需求的职责。例如，当 UI 表示层需要获得极为丰富的查询模型时，通过严谨设计获得的聚合是否能够直接满足这一需求呢？如果希望执行高性能的查询请求，频繁映射关系表与对象的查询接口是否带来了太多不必要的间接转换成本呢？如果查询采用同步操作，命令采用异步操作，采用同一套领域模型是否能够很好地满足不同的执行请求？因此，可以说 CQRS 模式脱胎于 CQS 模式，是其模式在架构层面上的设计思想延续。

### CQRS 模式的架构

CQRS 模式做出的革命性改变是将模型一分为二，分为查询模型和命令模型。同时，根据命令操作的特性以及质量属性的要求，酌情考虑引入命令总线、事件总线以及事件存储。遵循 CQRS 模式的架构如下图所示：



96fda924-06a2-4ed7-9728-14008634e3d7.png

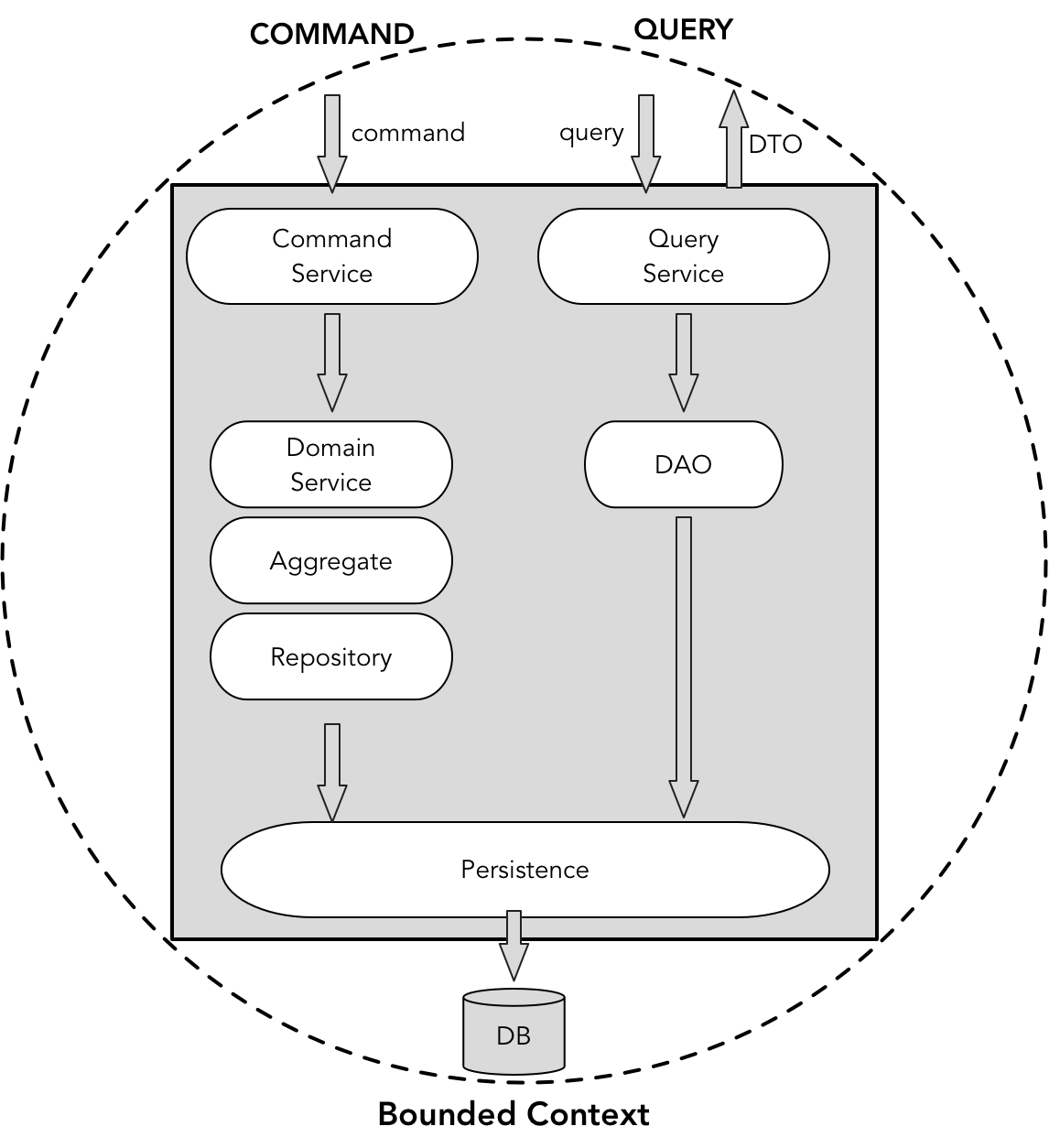
如上图所示，左侧命令处理器操作的领域模型就是命令模型。如果没有采用事件溯源与事件存储，该领域模型与普通领域模型并无任何区别，仍然包括实体、值对象、领域服务、资源库和工厂，实体与值对象放在聚合边界内，若有必要还可以引入领域事件。相反，上图右侧查询操作面对的查询模型，其实是直接响应调用者请求的 DTO 对象，即响应消息对象。响应消息对象并不属于领域模型，因为查询端要求查询操作干净利落、直截了当，尽量减少不必要的对象转换，故而没有定义领域层，而是通过一个薄薄的数据层直接访问数据库。为了应对查询的数据需求并提高查询性能，还可以在数据库中专门为查询操作建立对应的视图。查询返回的结果无需经过领域模型，直接转换为调用者需要的响应请求对象。

领域模型之所以需要为命令操作保留，是由命令操作本身具有的业务复杂性决定的。注意，虽然 CQRS 模式脱胎于 CQS 模式，但并不意味着命令操作对应的方法都具有副作用。如薪资管理系统中 HourlyEmployee 类的 payroll() 方法，会根据结算周期与工作时间卡执行薪资计算，只要结算周期与工作时间卡的值是确定的，方法返回的结果也是确定的，满足了引用透明的规则。 **换言之，如果没有采用事件范式，聚合中的实体与值对象、领域服务的设计并不受 CQRS 模式的影响。CQRS 模式之所以划分命令操作与查询操作，实则是针对资源库进行的改良。**

资源库作为管理聚合生命周期的对象，承担了增删改查的职责。由于 CQRS 模式要求分离命令操作和查询操作，就相当于砍掉了资源库执行查询操作的职责。在去掉查询操作后，命令操作执行的聚合又来自何处呢？难道还需要去求助专门的查询接口吗？其实不然，虽然命令模型的资源库不再提供查询方法，然而根据聚合根实体的 ID 执行查询的方法仍然需要保留，否则就无从管理聚合的生命周期了。因此，命令模型的一个典型资源库接口应如下所示：

package …….commandmodel;  
  
public interface {CommandModel}Repository {  
 Optional<AggregateRoot> fromId(Identity aggregateId);  
 void add(AggregateRoot aggregate);  
 void update(AggregateRoot aggregate);  
 void remove(AggregateRoot aggregate);  
}

在命令端，除了需要将其余查询方法从资源库接口中分离出去外，与领域驱动战术设计的要求完全保持一致，也遵循整洁架构的思想，形成基础设施层、领域层和应用层的分层架构。查询端则不同，它可以打破领域驱动分层架构的约束，直接通过远程的查询服务调用对应的数据访问对象（DAO）即可。DAO 对象返回的结果直接转换为对应的响应消息对象，甚至可以是 UI 前端需要的视图模型对象。整个架构如下图所示：



58279831.png

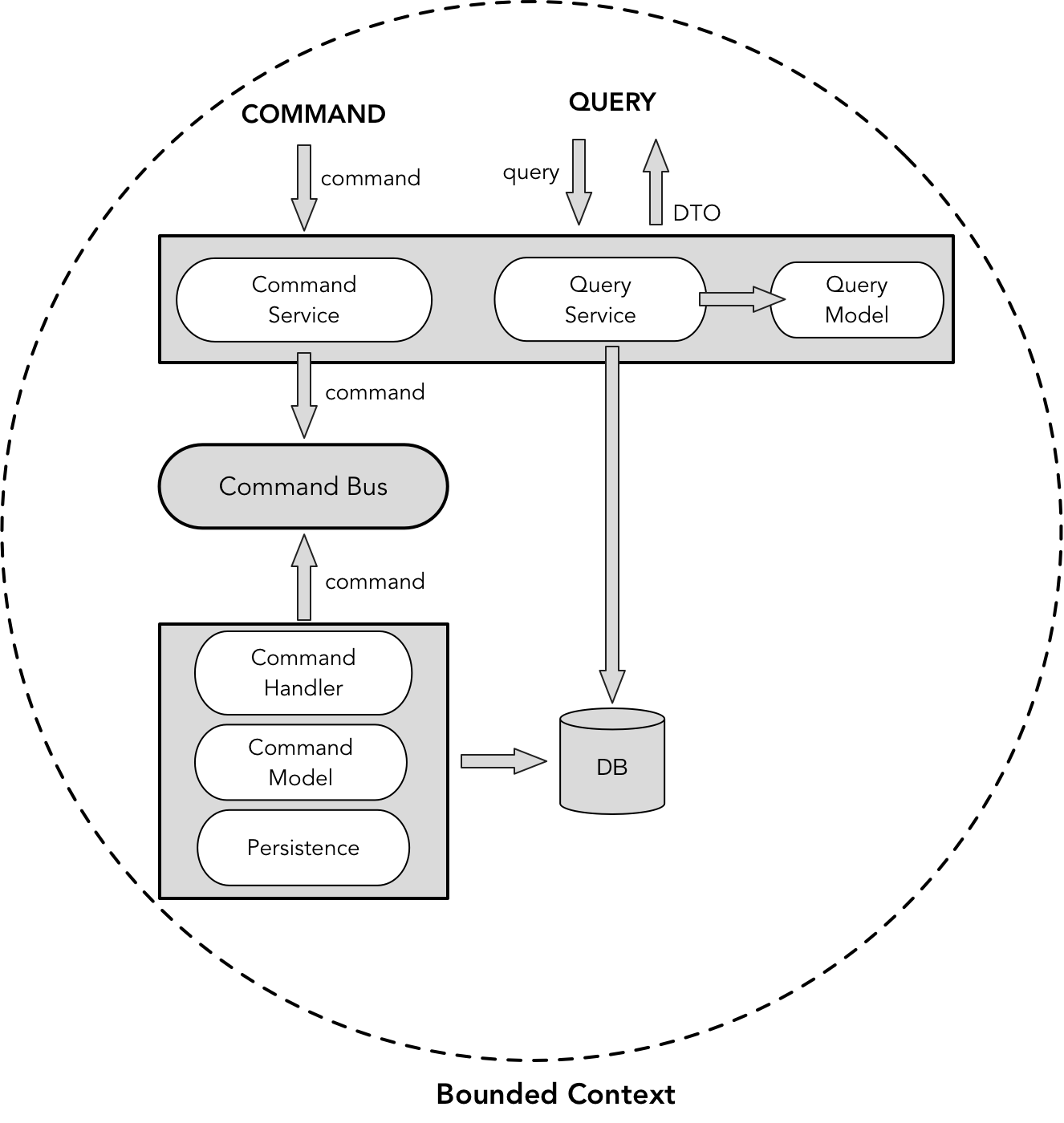
如图所示，命令端与服务端都在一个限界上下文内，但它们采用了不同的分层架构。关键之处在于查询端无需领域模型，从而减少了不必要的抽象与间接，满足快速查询的业务需求。

### 引入命令总线

如果命令请求需要执行较长时间，或者服务端需要承受高并发的压力，又无需实时获取执行命令的结果，就可以引入命令总线，将同步的命令请求改为异步方式，如此即可有效利用分布式资源，降低整个系统的延迟。

在大型的软件系统中，通常使用消息队列中间件作为命令总线。消息队列引入的异步通信机制，使得发送方和接收方都不用等待对方返回成功消息即可执行后续的代码，从而提高了数据处理的能力。尤其当访问量和数据流量较大的情况下，可结合消息队列与后台任务，通过避开高峰期对任务进行批量处理，就可以有效降低数据库处理数据的负荷，同时也减轻了命令请求服务端的压力。

为保证命令端与查询端的一致性，可以采用共同的远程服务层，以 REST 服务或 RPC 服务接口暴露给客户端的调用者。当远程服务接收到调用者的命令请求后，不做任何处理，立即将命令消息转发给消息队列。命令处理器作为命令消息的订阅者，在收到命令消息后调用领域模型对象执行对应的领域逻辑。如此一来，限界上下文的架构就会发生变化，接收命令请求的远程服务和命令处理器在逻辑上属于同一个限界上下文，但在物理上却部署在不同的服务器节点：



58395933.png

不同的命令请求会执行不同的业务逻辑， **应用服务作为业务用例的统一外观，承担命令处理器的职责** ，提供与该业务用例对应的命令处理方法。如订单应用服务需要响应下订单和取消订单命令：

// 此时的应用服务作为命令处理器  
public class OrderAppService {  
 public void placeOrder(PlaceOrderRequest placeOrderRequest) {}  
 public void cancleOrder(CancleOrderRequest cancelOrderRequest) {}  
}

应用服务的方法内部会调用命令请求对象或者装配器的转换方法，将命令请求对象转换为领域模型对象，然后将其委派给领域服务的对应方法。领域服务与聚合以及资源库之间的协作，和普通的领域驱动设计实现没有任何区别。显然，命令总线的引入增加了架构的复杂度，即使针对一个限界上下文，也引入了复杂的分布式通信机制，它带来的好处是提高了整个限界上下文面向调用者的响应能力。

#### 引入事件溯源模式

多数命令操作都具有副作用。如果将聚合状态的变更视为一种事件，就可以将命令操作转换为一种纯函数：Command -> Event。这实际上就引入了事件溯源模式。这一模式不仅改变了领域模型的建模方式，同时也改变了资源库的实现。通常，事件溯源模式需要与事件存储结合起来，因为资源库需要通过事件存储获得过去发生的事件，实现聚合的重建与更新操作。

第 3-19 课《事件溯源模式》已经深入讲解了事件溯源模式，这里就不再赘述。不过，CQRS 对事件溯源是有约束的。由于 CQRS 强调命令与查询分离，命令模型中的资源库不再支持查询操作，又因为事件溯源模式本身也无法很好地支持聚合查询功能，因此命令端的资源库不仅要负责追加事件，还需要将聚合持久化到业务数据库，以便于满足查询端的查询请求。为了避免引入不必要的分布式事务，事件存储与业务数据应放在同一个数据库中。

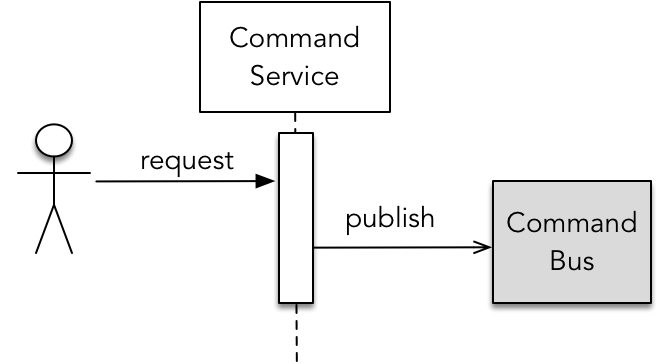
#### 引入事件总线

命令端与查询端还可以进一步引入事件总线来实现两端的完全独立。但在做出这一技术决策之前，需要审慎地判断它的必要性。毫无疑问，事件总线的引入进一步增加了架构的复杂度。

首先，一旦引入事件总线，就需要调整命令端的建模方式，即采用“事件建模范式” 。这种建模范式的建模核心是事件以及事件引起的状态迁移，需要改变建模者观察现实世界的方式。这种迥异于对象范式的建模思想，并非每个团队都能熟练地把握。其次，事件总线的作用是传递事件消息，然后由事件处理器订阅该事件消息，根据事件内容完成最终的命令请求，操作业务数据库的数据。这意味着命令端的领域模型必须采用事件溯源模式，且在存储事件的同时还需要发布事件。事件存储与业务数据位于消息队列的两端，属于不同的数据库，甚至可能选择不同类型的数据库。最后，以消息队列中间件担任事件总线，不可避免增加了分布式系统部署与管理的难度，通信也变得更加复杂。

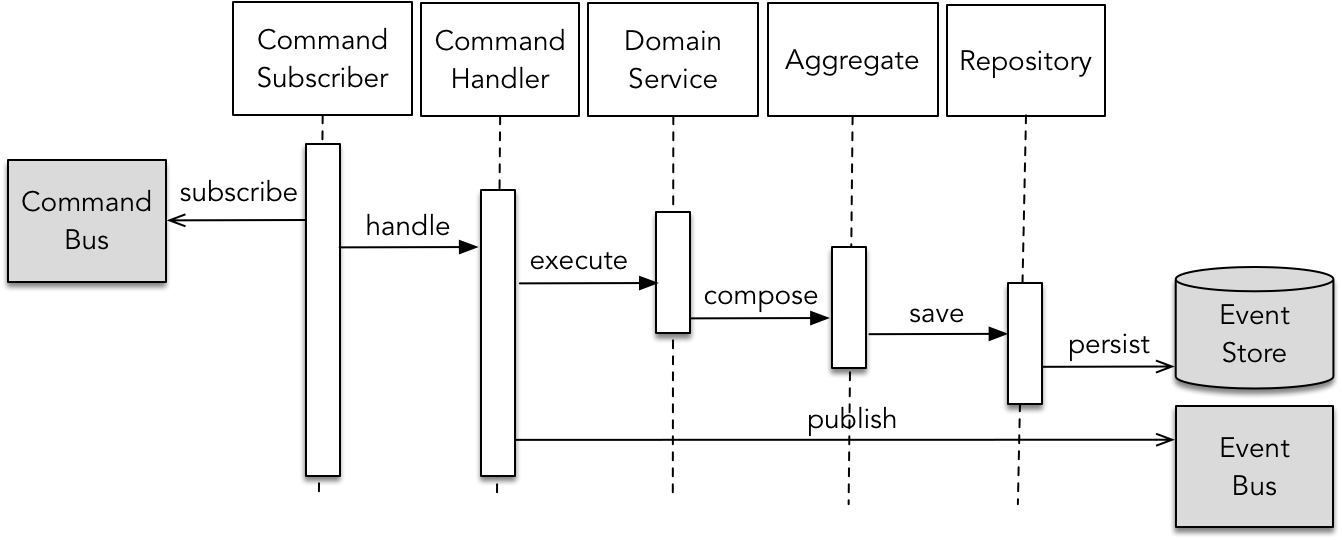
价值呢？在具有非常高的并发访问量时，引入的事件总线无疑可以改进每个服务器节点的响应能力，由于消息队列自身也能支持分布式部署，若能规划好事件发布与订阅的分区和主题设计，就能有效地分配和利用资源，满足不同业务场景的可扩展性需求。一些 CQRS 框架提供了对消息队列的支持，例如 AxonFramework 就允许使用者建立一个基于 AMQP 的事件总线，还可以使用消息代理（Message Broker）对消息进行分配。

引入分布式事件总线的 CQRS 模式最为复杂，通常需要结合事件溯源模式。首先，客户端向命令服务发起请求，命令服务在接收到命令之后，将其作为消息发布到命令总线：



74103526.png

命令订阅器会侦听（或订阅）命令总线以接收命令消息，并调用命令处理器处理命令消息。在命令模型中，命令处理器其实就是应用层的应用服务，它会将接收到的命令请求传递给领域服务，领域服务则负责协调聚合与资源库。由于模型采用了事件溯源模式，聚合承担了生成事件的职责，资源库表面看来是聚合的资源库，实际上完成的是领域事件的持久化。一旦领域事件被存储到事件存储中，作为应用服务的命令处理器就会将该领域事件发布到事件总线：



74407230.png

在事件总线的客户端，消息订阅者负责侦听事件总线，一旦接收到事件消息，就会将反序列化后的事件消息对象转发给事件处理器。由于事件处理器与命令处理器分属不同的进程，为了保证它们之间的独立性，传递的事件消息应采用“事件携带状态迁移”风格，事件自身携带了事件处理器需要的聚合数据，交由资源库完成对聚合的持久化：

74552499.png

74552499.png

显然，事件总线发布侧的资源库负责持久化事件，事件订阅侧的资源库则需要访问聚合存储数据库，完成对聚合内实体和值对象的持久化。

### 小结

CQRS 模式的复杂度可繁可简，因而对于领域驱动设计的影响亦可大可小，但最根本的是改变了查询模型的设计。这一设计思想其实与领域驱动设计核心子领域的识别相吻合，即如果领域模型不属于核心子领域，可以选择适合其领域特点的最简便方法。一个限界上下文可能属于领域子领域的范围，然而，由于查询逻辑并不牵涉到太多的领域规则与业务流程，更强调快速方便地获取数据，因此可以打破领域模型的设计约束。

引入命令总线并不意味着必须引入事件，它仅仅改变了命令请求的处理模式。若 CQRS 模式引入了事件总线，它的设计会与事件溯源模式更为匹配，可以更好地发挥事件或领域事件的价值。注意，CQRS 并没有要求总线必须为运行在独立进程中的中间件。在 CQRS 架构模式下，总线的职责就是发布、传递与订阅消息，并根据消息特征与角色的不同分为了命令总线和事件总线，根据消息处理方式的不同分为同步总线和异步总线。只要能够履行这样的职责，并能高效地处理消息，不必一定使用消息队列。例如，为了降低 CQRS 的复杂度，我们也可以使用 Guava 或 AKKA 提供的 EventBus 库，以本地方式实现命令消息和事件消息的传递（AKKA 同时也支持分布式消息）。

完整引入命令总线与事件总线的 CQRS 模式确实存在较高的复杂度，在选择该解决方案时，需要慎之又慎，认真评估复杂度带来的成本与收益之比；同时，团队也需要明白如上所述 CQRS 模式对领域驱动设计带来的影响。

下一章

还没有评论



评论

## 更多资源下载交流请加微信：Morstrong,加入永久会员,网盘更新更快捷！

# 本资源由微信公众号：光明顶一号，提供支持