AggregateFamework Reference

作者
changmingxie

介绍

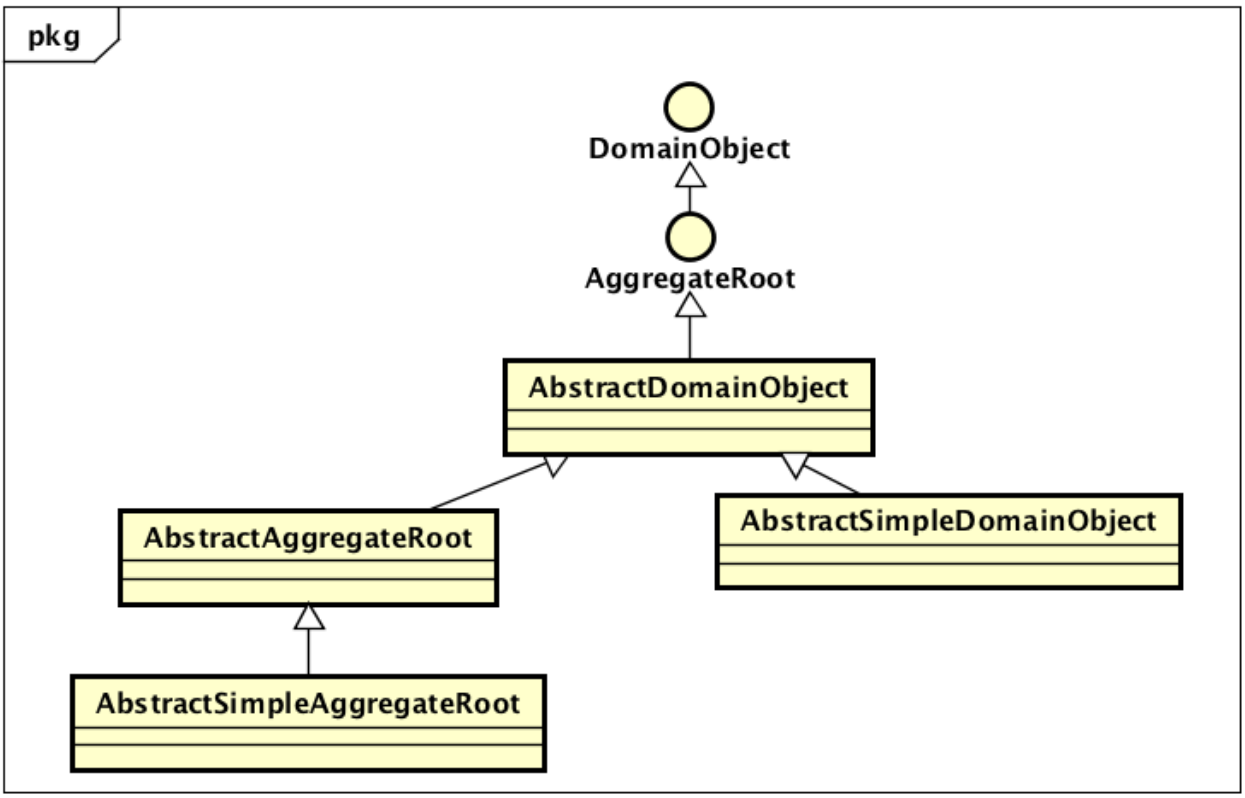
什么是 AggregateFramework

Aggregate Framework 是为方便开发人员运用 DDD 和 CQRS 思想来构建复杂的、可扩展的 Java 企业应用系统而提供的 Java 技术框架。该框架提供了 Aggregate、Repository、Domain Event 等构建块的实现；使用 DomainEvent，借助于内建的 Disruptor 组件，AggregateFramework 可使开发人员方便的实现高性能 SEDA 架构。此外，该框架支持与 Spring 集成，提供使用 annotation 的方式让开发人员方便地为 Domain Event 定义一个或多个事件处理, 同时可指定事件处理是同步还是异步触发，是否需要重试，以及重试策略; 使用 Spring 事务管理器管理事务时，支持 Unit Of Work 数据访问模式以及内建一级缓存以提高访问性能。

Aggregate

Aggregate(聚合)定义了一组具有内聚关系的相关对象的集合，是一致性修改数据的单元。聚合里包含有聚合根，实体、值对象。AggregateFramework 为聚合定义了接口和抽象父类，开发人员实现领域模型时需继承这些抽象父类。

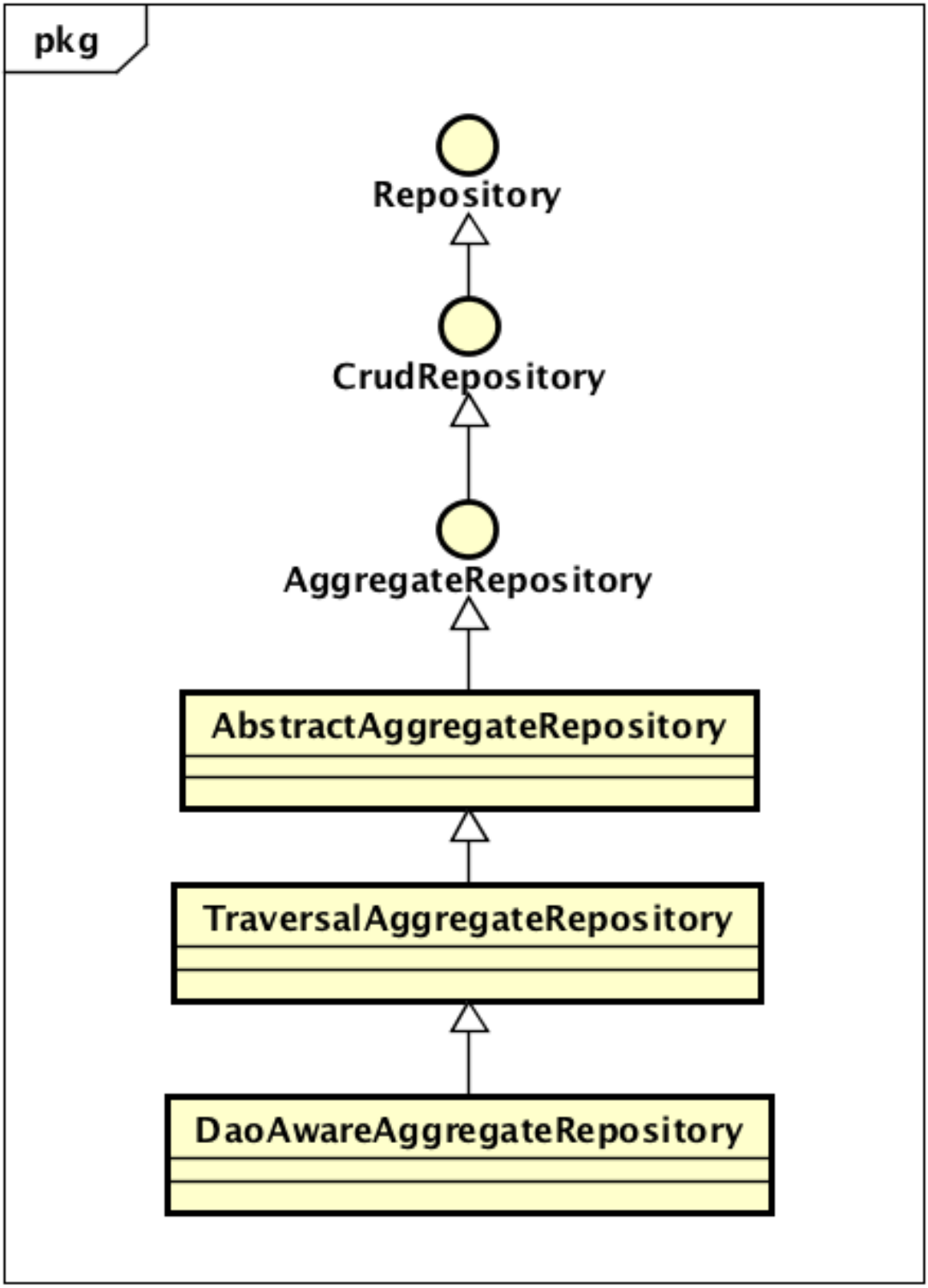
在 Aggregate Framework 中，定义了聚合对象的接口及抽象父类,类层次结构如下。聚合最核心的接口是 DomainObject，聚合对象都实现该接口。另一个接口是 AggregetRoot，该接口继承自 DomainObject，聚合根实现该接口。AbstractDomainObject 抽象类实现了 DomainObject 接口, 并提供了对部分方法的重载。AbstractAggregateRoot 继承 AbstractDomainObject，在 AbstractDomainObject 上新增了对 Event 的支持。为方便定义聚合对象，AbstractSimpleAggregateRoot 和 AbstractSimpleDomainObject 分别对 AbstractAggregateRoot 和 AbstractDomainObject 进 行了简单实现，提供对接口的默认实现，让开发人员不需关系内部事件注册及发布机制。



Repository

Repository 用于协调领域和数据映射层，利用类似于集合的接口来访问领域对象，每个聚合根有一个对应的 Repository, 用于对这个聚合对象的数据访问。Aggregate Framework 定义 Repository 相关的接口及抽象父类，开发人员在实现聚合根的 Repository 时需要继承这些抽象父类。

在 Aggregate Framework 中，定义了 Repository 对象的接口及抽象父类,类层次结构如下。在 Repository 构建块中，最核心的接口莫过于 Repository。它是个标记接口，CrudRepository 继承该接口，并为聚合对象提供 了专门的 CRUD 方法。AggreateRepository 继承 CrudRepository, 限制操作的实体是聚合根。AbstractAggregateRepository 提供了对 AggregateReposiotry 的部分实现，将事件的 发布进行了简单处理。TraversalAggregateRepository 继承了 AbstractAggregateRepository，同时实 现了对聚合里对象的成员变量进行遍历并调用对应的 DAO 方法进行 CRUD 操作。DaoAwareAggregateRepository 继承自 TraversalAggregateRepository，实现了基于 Spring 获取 DAO 依赖实现聚合对象的遍历 CRUD 操作。



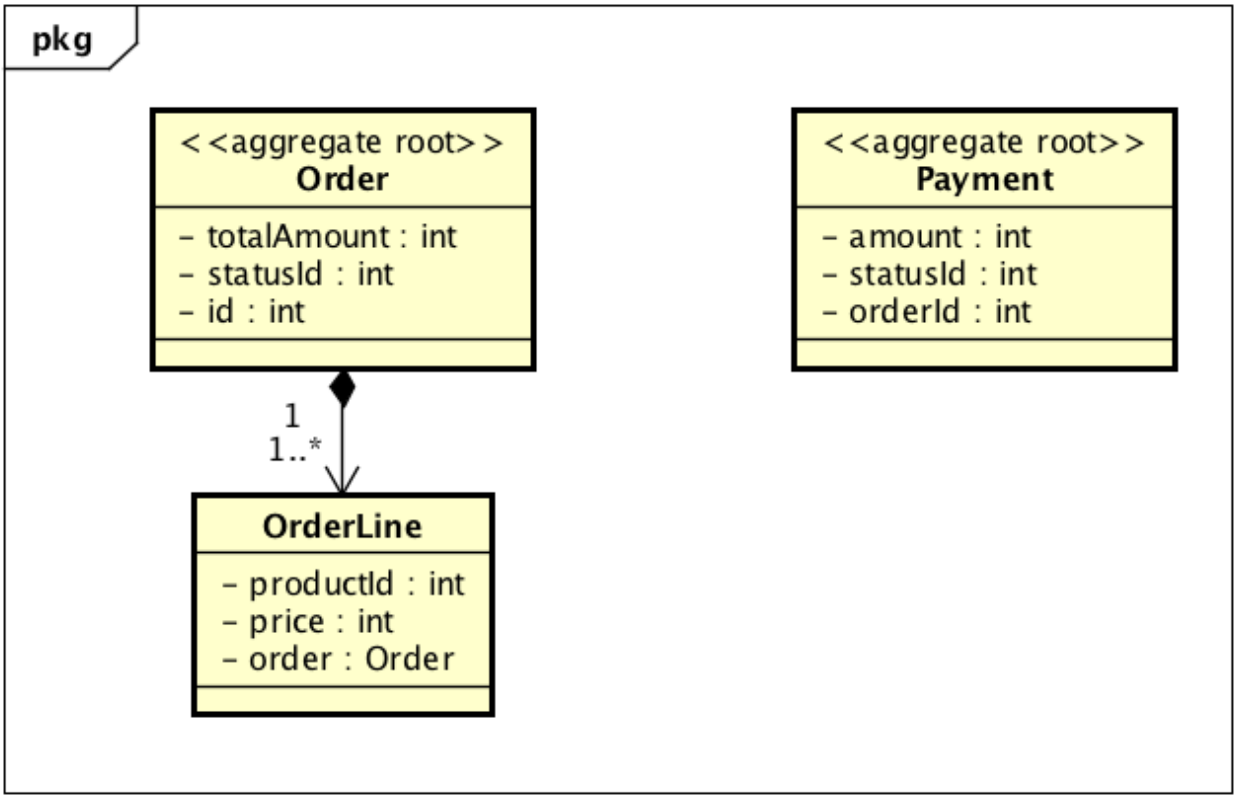
Domain Event

Domain Event 是领域中已经发生的事件，当领域对象的状态发生了改变，便会产生 Domain Event。

在 AggregateFramework 中聚合根的抽象父类 AbstractAggregateRoot 中定义了注册 Domain Event 的方法 apply(Object eventPayload), 当领域对象的改变其状态后，可使用 apply 方法来注册一个 Domain Event，以表示产生了一个 Domain Event。当调用聚合根的 Repository 方法 save 来保存聚合对象时，框架会将注册的 Domain Event 发布出去。

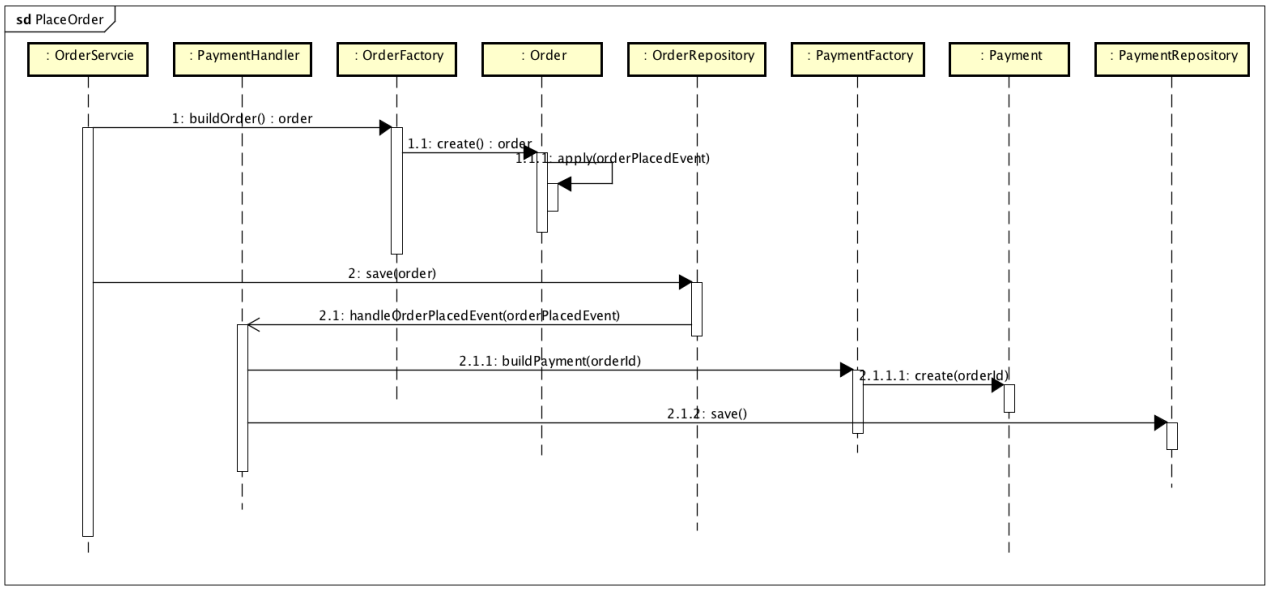
使用指南

本指南使用一个示例来阐述如何使用 AggregateFramework 开发。示例的领域模型如下图，Order 和 Payment 是聚合根，OrderLine 是 Order 聚合里的实体。

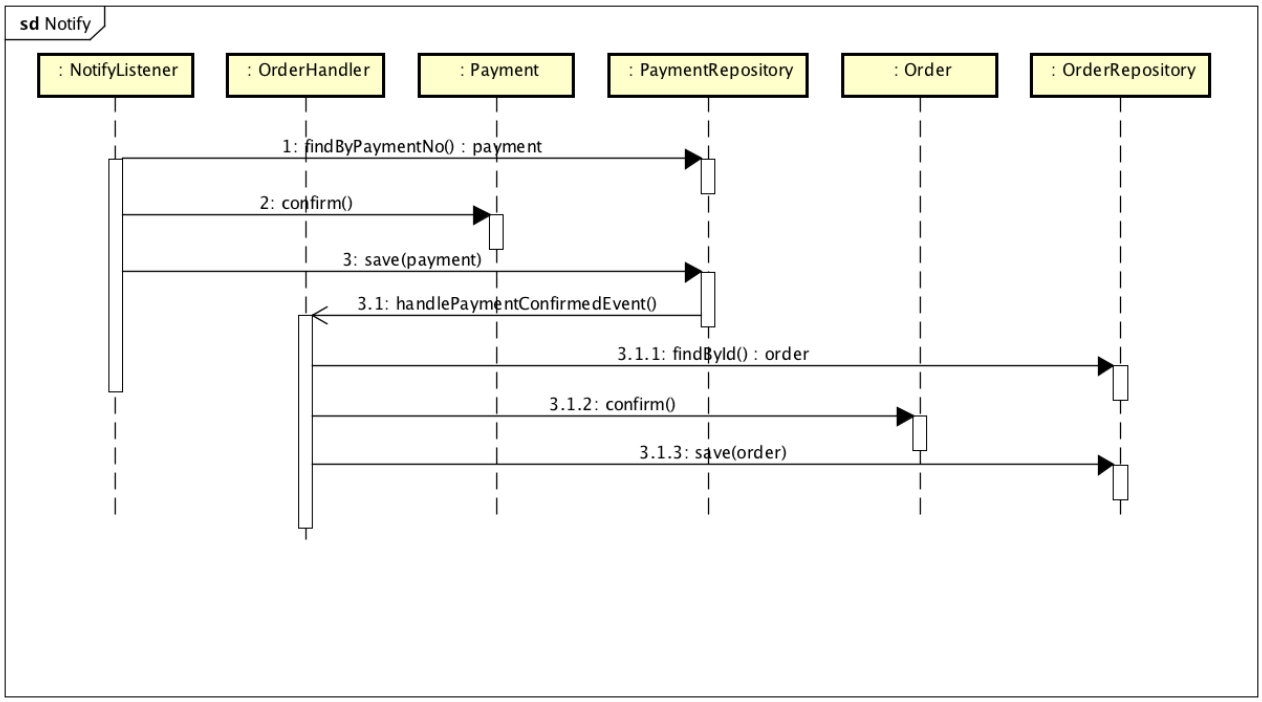


在示例里，将阐述两个业务流程，一个是下订单流程，另一个是第三方支付通知支付成功流程。

下订单过程为：创建 Order 聚合对象，创建后将发布 OrderPlaced 事件，PaymentHandler 接受 OrderPlaced 事件，并为创建的 Order 聚合对象产生一个 Payment。下面是流程的序列图。



第三方支付通知支付成功过程为：根据支付订单号获取 Payment, 更新 Payment 为 Confirmed 状态，更新状态后将发布 PaymentConfirmed 事件，OrderHandler 接受 PaymentConfirmed 事件，并将对应的 Order 更新为 confirm 状态。下面是流程的序列图。



1. 定义 Aggregate 类

聚合根类继承 AbstractAggregateRoot 抽象类，聚合实体类继承 AbstractDomainObject 抽象类，聚合根引用聚合实体，聚合根和聚合实体的关系可以是 1:0..1 或是 1：0..*。为简化实现，AggregateFramework 里提供了对 AbstractAggregateRoot 和 AbstractDomainObject 的缺省实现类，分别为 AbstractSimpleAggregateRoot 类和 AbstractSimpleDomainObject 类。

示例中，Order 和 Payment 是聚合根，继承 AbstractSimpleAggregateRoot 类，Order 和 Payment 都提供 confirm 方法以改变对象的状态。OrderLine 是 Order 聚合对象里的实体,

Order 与 OrderLine 是一对多的关系，Order.orderLines 实现关联多个 OrderLine，同时 OrderLine.order 反向关联一个 Order。

2. 注册 Domain Event

当领域对象的状态改变后，可注册 Domain Event。在 AbstractAggregateRoot 里提供了 apply 方法用于注册 Domain Event。

在示例中，在 Order 的 confirm 方法里，改变了 Order 的状态后，调用 apply 方法注册一个 OrderPlacedEvent 事件, 在 Payment 的 confirm 方法里，改变了 Payment 的状态后，调用 apply 方法注册一个 PaymentConfirmedEvent 事件。

3. 定义 Repository

每个聚合根对应一个 Repository。聚合根的 Repository 类继承 AbstractAggregateRepository 类。通常应用系统使用 DAO 模式进行数据访问，每个实体类对应一个 DAO,AggregateFramework 提供了另外的抽象类 TraversalAggregateRepository 类和 DaoAwareAggregateRepository 类以支持实体类底层使用 DAO 模式进行数据访问。

在示例中，OrderRepository 和 PaymentRepository 继承 DaoAwareAggregateRepository。使用 OrderRepository 和 PaymentRepository 进行数据访问操作时，AggregateFramework 将 CRUD 操作委派给聚合里实体的 DAO 类进行数据访问。定义 DAO 将在下面介绍。

4. 定义 DAO

AggregateFramework 为 DAO 模式提供了支持, 下图为 DAO 类结构图。每个领域实体对应一个 DAO 类，聚合根对应的 DAO 类继承 AggregateRootDao 或是 CollectiveAggregateRootDao，聚合里的实体类对应的 DAO 继承 DomainObjectDao 或 CollectiveDomainObjectDao。 推荐使用 CollectiveAggregateRootDao 和 CollectiveDomainObjectDao,相比另外两个 DAO 增加了 insertAll,updateAll,deleteAll 和 findByIds 集合操作方法以实现对实体对象的批量 CRUD 操作。

示例中，OrderDao 和 PaymentDao 接口类继承了 CollectiveAggregateRootDao 类，OrderLineDao 接口类继承 CollectiveDomainObjectDao 类。示例使用 mybatis 的 MapperFactoryBean 自动生成 DAO 接口类的实例。

5. 定义事件处理方法

领域实体当状态发生了改变后会发出 Domain Event。使用@EventHandler 在 Spring 容器管理的类的方法上来声明领域事件处理方法，方法参数需有一个为 Domain Event 类型的参数以表明领域事件处理方法处理这个 Domain Event。@EventHandler 里有两个属性，一个是 asynchronize, 默认是 false, 表示事件处理方法将和对领域实体的更改在同一个线程中执行，如果设置为 true, 则表示事件处理方法和对领域实体操作在不同线程执行，即异步执行。另一个属性是 postAfterTransaction, 当使用 TransactionManager 管理事务时有效，默认是 false, 表示事件处理方法和对领域实体的更改操作在同一个事务中，如果是 true, 则事件处理方法将在对领域实体更改操作结束后执行。

PaymentHandler 和 OrderHandler 类定义了事件处理方法。

6. 设置重试策略

与@EventHandler 配合使用还有一个注解@Retryable, 类似于 spring-retry，该注解用来设置事件处理方法的重试策略, 比如重试次数，对哪些异常进行重试，对哪些异常不重试，重试间隔。