## **4.1建模**

### **4.1.1聚合**

4.1.1.1基本聚合结构

***@Aggregate***

@Profile("command")

public class GiftCard {

private static final Logger logger = LoggerFactory.getLogger(MethodHandles.lookup().lookupClass());

***@AggregateIdentifier***

private String giftCardId;

private int remainingValue;

***@CommandHandler***

public GiftCard(IssueCmd cmd) {

logger.debug("handling {}", cmd);

if (cmd.getAmount() <= 0) {

throw new IllegalArgumentException("amount <= 0");

}

***apply(new IssuedEvt(cmd.getId(), cmd.getAmount()));***

}

@CommandHandler

public void handle(RedeemCmd cmd) {

logger.debug("handling {}", cmd);

if (cmd.getAmount() <= 0) {

throw new IllegalArgumentException("amount <= 0");

}

if (cmd.getAmount() > remainingValue) {

throw new IllegalStateException("amount > remaining value");

}

apply(new RedeemedEvt(giftCardId, cmd.getAmount()));

}

***@EventSourcingHandler***

public void on(***IssuedEvt*** evt) {

logger.debug("applying {}", evt);

giftCardId = evt.getId();

remainingValue = evt.getAmount();

logger.debug("new remaining value: {}", remainingValue);

}

***public GiftCard()*** {

// Required by Axon

logger.debug("Empty constructor invoked");

}

}

在给定的代码片段中有两个值得注意的概念，它们用编号的Java注释标记，使用这些注释应注意以下要点：

①@AggregateIdentifier是指向GiftCard聚合的外部引用点。这个字段是一个硬要求，因为没有它，Axon将不知道给定命令的目标聚合。请注意，此注释可以放置在字段和方法上。

②@CommandHandler带注释的构造函数，或者以不同的方式将“command handling constructor”放在一起。此注释告诉框架给定的构造函数能够处理IssueCardCommand。@CommandHandler注释函数是放置决策/业务逻辑的地方。              ③静态AggregateLifecycle#apply（Object…）是应该发布事件消息时使用的。              调用此函数时，所提供的对象将在其应用的聚合范围内作为EventMessages发布。              ④使用@EventSourcingHandler告诉框架，当聚合是“源于其事件”时，应该调用带注释的函数。由于所有事件源处理程序组合在一起将形成聚合，因此所有状态更改都在这里发生。请注意，聚合标识符必须在聚合发布的第一个事件的@EventSourcingHandler中设置。这通常是创建事件。最后，@EventSourcingHandler注释的函数是使用特定规则解析的。这些规则对于@EventHandler注释的方法是相同的，并在带注释的事件处理程序中进行了详细说明。

⑤一个无参构造方法是Axon所必需的。Axon框架使用此构造函数在使用过去的事件初始化它之前创建一个空的聚合实例。未提供此构造函数将导致加载聚合时出现异常。

4.1.1.2聚合生命周期操作

在一个聚合的生命周期中，需要执行一些操作。为此，Axon中的AggregateLifecycle类提供了两个静态函数：

①apply（Object）和apply（Object，MetaData）：AggregateLifecycle#app将在EventBus上发布一条事件消息，以便知道它是由执行操作的聚合发出的。可以只提供事件对象，也可以同时提供事件和某些特定的元数据。

②createNew（Class，Callable）：作为处理命令的结果实例化一个新的聚合。              阅读本文了解更多细节。

③isLive（）：检查以验证聚合是否处于“存活”状态。如果一个聚合体完成了历史事件的回放以重现其状态，那么它被认为是“活的”。如果聚合因此处于事件源的过程中，则AggregateLifecycle.isLive()调用将返回false。              使用这个isLive（）方法，您可以执行只有在处理新生成的事件时才应该执行的活动。

④markDeleted（）：将调用函数的聚合实例标记为“deleted”。

如果域指定可以删除/删除/关闭给定的聚合，则非常有用，在此之后，它将不再被允许处理任何命令。应该从@EventSourcingHandler带注释的函数调用此函数，以确保标记为已删除是该聚合状态的一部分。