## 如何使用本书

Eric Evans在他那本 《领域驱动设计》 中向我们展示了一整套模式语言。模式语言是相互关联的众多软件模式的一个集合，任何一种模式都会引用并依赖于其他一种或多种模式。这意味着什么呢？

这意味着当你在阅读本书时，某个章节中出现的有些DDD模式并不会在该章节中讲到，甚至在该章节之前都没有被谈及到。不要担心，继续往下读，被引用的模式将在本书的其他章节中做详细讲解。

在本书中，我将使用下表中的行文惯例 ：

表G.1 本文的行文惯例

|  |  |
| --- | --- |
| 出现文本 | 含义 |
| 模式名字(#) | 1：该模式是第一次出现在本书中，或者 2：该模式已经在本章中出现了，并且非常重要。 |
| 限界上下文(2) | 表示所引用的限界上下文在第2章中有详细的讲解。 |
| 限界上下文 | 表明限界上下文已经在本章中出现过了，这里我并不会每次引用一个模式时都将其标为粗体并后加章号 |
| [REFERENCE] | 表明对参考文献的引用。 |
| [Evans]或[Evans,Ref] | 表明对于被引用的模式 ，你可以参考 Evans 的著作以获得更多信息。［Evans］表示他那本经典的《领域驱动设计》 ，［Evans, Ref］并不表示引用Evans那本书本身，而是另外的引用源，该引用源也引用了Evans书中模式，并且在此 基础上有更新和扩展。 |
| [Gamma et al.］和 [Fowler，P of EAA］ | [Gamma et al.］表示Gamma等人所著的那本经经典的《设计模式》。  [Fowler，P of EAA］ 表示Martin Flowler的《企业应用架构模式》。  在本书中，我会经常引用到这两本书，虽然我还引用了其他的，但这两本是主要的。 |

在阅读的过程中，如果遇到对某个模式的引用，比如限界上下文，此时你通常可以在另外某个章节找到对该模式的讲解 。

如果你已经读过［Evans］ ，并且对其中的模式有一定的了解，那么本书可以帮助你进一步澄清DDD的概念，然后引导你对既有的模型进行改进。此时你可能并不需要一个总览式的介绍。但是，如果你还是DDD的新手，那么下面的内容将为你讲到不同的DDD模式是如何协同工作的，并且如何更好地使用本书，接着往下读吧。

### DDD总览

早些时候，我讲到了**DDD的通用语言 （Ubiquitous Language，1）。**通用语言作用于某个**限界上下文 （ Bounded Context , 2 ），**它对于领域建模是非常重要的，你应该好好地熟悉一下。请记住，不管你是在战术上还是战略上设计软件模型，你都应该保证：在一个特定的限界上下文中只使用一套通用语言，并且保证它的清晰性和简洁性 。

#### 战略建模

限界上下文是一种概念上的边界，领域模型便工作于其中。同时，限界上下文为通用语言提供了一套环境，项目成员便通过通用语言来表达软件模型 ，如图G.1所示。

通用语言（1）建模于内部

限界上下文(2)

图G.1 限界上下文和通用语言

在战略设计的过程中，你将发现**上下文映射图（Context Map, 3）** 是非常有用 的，如图G.2所示。你的团队将使用上下文映射图来理解项目的范围。

以上我们简要地了解了DDD的战略设计，这是我们必须好好理解的概念 。

U

D

线下文之间的集成关系：  
开放主机服务、发布语言、防腐层、  
客户方-供应方、合作关系、遵奉者、  
共享内核

限界上下文（2）

？

？

图G.2 上下文映射图展示限界上下文之间的关系

##### 架构

有时，一个新的限界上下文或上下文映射图可能需要一种新的**架构(Archi-tecture,4）**。你应该牢记：通过战略和战术设计而成的领域模型应该是架构中立的。当然，在模型周围和模型之间则是存在架构的。一种能够支撑限界上下文的架构是六边形 （ Hexagonal） 架构，它可以辅助其他架构风格，比如**面向服务 （ Ser­ vice-Oriented ）** 架构、**REST**和**事件驱动 （ Event-Driven）** 等。六边形架构如图 G.3 所示，从表面看，这种架构有点复杂，但是事实上却恰恰相反。

有时我们过于强调架构而忽略了DDD建模的重要性。架构固然是好的，但是架构并非一成不变。此时我们须要正确地处理优先级,将重点放在领域模型上,因为领域模型将产生更多的业务价值，并且更具有持久性。

y

适配器

适配器

适配器

适配器

适配器

适配器

适配器

适配器

架构（4）比如六边形架构风格

战术领域上下文位于限界上下文的中心

图G.3 六边形架构风格 ，领域模型位于软件的中心

应用程序

#### 战术建模

我们在限界上下文中进行DDD的战术建模。战术设计的一个重要模式是聚合 (Aggregate, 10），如图G.4所示。

聚合可以由单个**实体（Entity, 5）** 组成，也可以由一组实体和**值对象(Value Object, 6)**组成，此时我们必须在聚合的整个生命周期中保证事务上的一致性。 有效地对聚合进行建模是重要的 ，同时聚合又是DDD 中最不容易理解的概念之一。

你可能会问，既然聚合如此重要，那为什么要将其放在本书的后面呢？首先，本书中战术模式的出现顺序和［Evans］一样，此外，由于聚合以其他战术模式为基础，所以我们会先讲到实体、值对象等基本模式，再讲解聚合。

聚合实例通过**资源库(Repository, 12)**进行持久化，另外，对聚合的查找和获取也通过资源库完成 ，如图G.4所示。

《值对象》  
值类型1

《聚合根》  
根实体1

《值对象》  
值类型2

3

聚合类型1

《聚合根》  
根实体2

《值对象》  
值类型3

《值对象》  
值类型3

《值对象》  
值类型4

《资源库》  
资源库1

《资源库》  
资源库2

内部状态所反映的业务规则必须保持完全一致。

使用资源库(12)来持久化某个聚合类型

图G.4 两个聚合类型，它们拥有各自的事务一致性边界

拥有事务一致性边界的集合(10)

聚合类型2

在领域模型中，有些业务操作并不能自然地放在实体或值对象上，此时我们可以使用无状态的**领域服务(DomainService，7）**，如图G.5所示:

《领域服务》  
领域服务1

《聚合根》  
根实体1

《聚合根》  
根实体2

使用领域服务(7) 来执行跨聚合的操作

图G.5 领域服务执行特定于领域的操作，其中可能涉及到多个领域对象

**领域事件(DomainEvent，8）**表示领域模型中发生的重要事件。有多种方式可以对领域事件进行建模。在对聚合进行命令操作时，聚合本身将发布领域事件，如图G.6所示。

订阅方

订阅方

事件发布器

事件

聚合

订阅方