<<领域驱动设计与模式实战>>

图书基本信息

书名：<<领域驱动设计与模式实战>>

13位ISBN编号：9787115212771

10位ISBN编号：7115212775

出版时间：2009-10

出版时间：人民邮电出版社

作者：[瑞典] Jimmy Nilsson 著

页数：354

译者：赵俐 马燕新 等

前言

构建企业应用程序并非易事。
尽管我们拥有大量工具和框架来简化这项任务，但仍然需要弄清楚如何更好地使用这些工具。
大量方法可供我们选用，但关键是要知道在特定情况下应使用哪种方法，因为一种方法很难适用于所有情况。
在过去几年中，有一个社区逐渐成长起来，人们不断寻找用于设计企业应用的方法，并以模式的形式将它们记录下来（我整理了一份概述，在http：//martinfowler.com／articles／enterprisePattems.html站点上）。
参与这项工作的人们（例如我）试图找到公共方法，并描述如何更好地使用它们，以及它们何时适用。
最后的结果过于宽泛，会导致为读者提供了过多的选择。
当我开始创作Patterns Df Enterprise Application Architecture（《企业应用架构模式》，Addison.’Wesley公司2002出版）时，我曾在微软技术方面寻找过这种类型的设计建议。
几经努力之后几乎一无所获，只找到了一本讨论此领域的书，就是Jimmy的前一本书。
我喜欢他平易近人的写作风格，也喜欢他深入挖掘易被他人忽略的概念的热情。
因此，Jimmy决定从我和企业模式社区中的其他人借鉴一些思想，并向读者展示在编写.NET应用程序时如何应用这些思想，我觉得再合适不过了。

内容概要

《领域驱动设计与模式实战》全面详细地解释了领域驱动设计、测试驱动开发、依赖注入、持久化、重构、模式等很多基本概念，并以C#和.NET实例为依托，展示了这些概念的实际应用和重要价值。
更重要的是，《领域驱动设计与模式实战》还将这些概念整合到一起，为开发人员从头至尾地揭示了完整的开发路线。
阅读《领域驱动设计与模式实战》后，读者将能真正掌握这些重要概念，并有效地将它们结合起来，应用到实际开发过程中。
《领域驱动设计与模式实战》适合软件架构师和开发人员阅读。

作者简介

作者：(瑞典)尼尔森(Jimmy Nilsson) 译者：赵俐 马燕新 等Jimmy Nilsson，资深软件架构师，有超过20年从业经验，2008年在瑞典主要IT媒体评选的全国软件架构师和开发人员排行榜上名列第2。
目前担任factor10咨询公司CEO，客户包括爱立信、微软、沃尔沃等。
本书是他的代表作，已被翻译为日、俄等多种文字，他的另一部著作.NET Enterprise Design with Visual Basic .NET and SQL Server 2000也获得Amazon 4星半评价。
他的博客是http://JimmyNilsson.com/blog/。

<<领域驱动设计与模式实战>>

书籍目录

第一部分 背景知识第1章 应重视的价值，也是对过去几年的沉重反思1.1 总体价值1.2 应重视的架构风格1.2.1 焦点之一：模型1.2.2 焦点之二：用例1.2.3 如果重视模型，就可以使用领域模型模式1.2.4 慎重处理数据库1.2.5 领域模型与关系数据库之间的阻抗失配1.2.6 谨慎处理分布式1.2.7 消息传递很重要1.3 对过程的各个组成部分的评价1.3.1 预先架构设计1.3.2 领域驱动设计1.3.3 测试驱动开发1.3.4 重构1.3.5 选择一种还是选择组合1.4 持续集成1.4.1 解决方案（或至少是正确方向上的一大步）1.4.2 从我的组织汲取的教训1.4.3 更多信息1.5 不要忘记运行机制1.5.1 有关何时需要运行机制的一个例子1.5.2 运行机制的一些例子1.5.3 它不仅仅是我们的过错1.6 小结第2章 模式起步2.1 模式概述2.1.1 为什么要学习模式2.1.2 在模式方面要注意哪些事情2.2 设计模式2.3 架构模式2.3.1 示例：层2.3.2 另一个示例：领域模型模式2.4 针对具体应用程序类型的设计模式2.5 领域模式2.6 小结第3章 TDD与重构3.1 TDD3.1.1 TDD流程3.1.2 演示3.1.3 设计效果3.1.4 问题3.1.5 下一个阶段3.2 模拟和桩3.2.1 典型单元测试3.2.2 声明独立性3.2.3 处理困难因素3.2.4 用测试桩替换协作对象3.2.5 用模拟对象替换协作对象3.2.6 设计含义3.2.7 结论3.2.8 更多信息3.3 重构3.4 小结第二部分 应用DDD第4章 新的默认架构4.1 新的默认架构的基础知识4.1.1 从以数据库为中心过渡到以领域模型为中心4.1.2 进一步关注DDD4.1.3 根据DDD进行分层4.2 轮廓4.2.1 领域模型示例的问题/特性4.2.2 逐个处理特性4.2.3 到目前为止的领域模型4.3 初次尝试将UI与领域模型挂接4.3.1 基本目标4.3.2 简单UI的当前焦点4.3.3 为客户列出订单4.3.4 添加订单4.3.5 刚才我们看到了什么4.4 另一个维度4.4.1 领域模型的位置4.4.2 孤立或共享的实例4.4.3 有状态或无状态领域模型实例化4.4.4 领域模型的完整实例化或子集实例化4.5 小结第5章 领域驱动设计进阶5.1 通过简单的TDD实验来精化领域模型5.1.1 从Order和OrderFactory的创建开始5.1.2 一些领域逻辑5.1.3 第二个任务：OrderRepository+OrderNumber5.1.4 重建持久化的实体：如何从外部设置值5.1.5 获取订单列表5.1.6 该到讨论实体的时候了5.1.7 再次回到流程上来5.1.8 总览图5.1.9 建立OrderRepository的伪实现5.1.10 简单讨论一下保存5.1.11 每个订单的总量5.1.12 历史客户信息5.1.13 实例的生命周期5.1.14 订单类型5.1.15 订单的介绍人5.2 连贯接口5.3 小结第6章 准备基础架构6.1 将POCO作为工作方式6.1.1 实体和值对象的PI6.1.2 是否使用PI6.1.3 运行时与编译时PI6.1.4 PI实体/值对象的代价6.1.5 将PI用于存储库6.1.6 单组存储库的代价6.2 对保存场景的处理6.3 建立伪版本机制6.3.1 伪版本机制的更多特性6.3.2 伪版本的实现6.3.3 影响单元测试6.4 数据库测试6.4.1 在每次测试之前重置数据库6.4.2 在测试运行期间保持数据库的状态6.4.3 测试之前重置测试所使用的数据6.4.4 不要忘记不断演变的模式6.4.5 分离单元测试和数据库调用测试6.5 查询6.5.1 单组查询对象6.5.2 单组查询对象的代价6.5.3 将查询定位到哪里6.5.4 再次将聚合作为工具6.5.5 将规格用于查询6.5.6 其他查询选择6.6 小结第7章 应用规则7.1 规则的分类7.2 规则的原则及用法7.2.1 双向规则检查：可选的（可能的）主动检查，必需的（和自动的）被动检查7.2.2 所有状态（即使是错误状态）都应该是可保存的7.2.3 规则应该高效使用7.2.4 规则应该是可配置的，以便添加自定义规则7.2.5 规则应与状态放在一起7.2.6 规则应该具有很高的可测试性7.2.7 系统应阻止我们进入错的状态7.3 开始创建API7.3.1 上下文，上下文，还是上下文7.3.2 数据库约束7.3.3 将规则绑定到与领域有关的转换，还是绑定到与基础架构有关的转换7.3.4 精化原则：所有状态，即使是错误状态，都应该是可保存的7.4 与持久化有关的基本的规则API的需求7.4.1 回到已发现的API问题上7.4.2 问题是什么7.4.3 我们允许了不正确的转换7.4.4 如果忘记检查怎么办7.5 关注与领域有关的规则7.5.1 需要合作的规则7.5.2 使用基于集合的处理方法7.5.3 基于服务的验证7.5.4 在不应该转换时尝试转换7.5.5 业务ID7.5.6 避免问题7.5.7 再次将聚合作为工具7.6 扩展API7.6.1 查询用于设置UI的规则7.6.2 使注入规则成为可能7.7 对实现进行精化7.7.1 一个初步实现7.7.2 创建规则类，离开最不成熟的阶段7.7.3 设置规则列表7.7.4 使用规则列表7.7.5 处理子列表7.7.6 一个API改进7.7.7 自定义7.7.8 为使用者提供元数据7.7.9 是否适合用模式来解决此问题7.7.10 复杂规则又是什么情况7.8 绑定到持久化抽象7.8.1 使验证接口成为可插入的7.8.2 在保存方面实现被动验证的替代解决方案7.8.3 重用映射元数据7.9 使用泛型和匿名方法7.10 其他人都做了什么7.11 小结第三部分 应用PoEAA第8章 用于持久化的基础架构8.1 持久化基础架构的需求8.2 将数据存储到哪里8.2.1 RAM8.2.2 文件系统8.2.3 对象数据库8.2.4 关系数据库8.2.5 使用一个还是多个资源管理器8.2.6 其他因素8.2.7 选择和前进8.3 方法8.3.1 自定义手工编码8.3.2 自定义代码的代码生成8.3.3 元数据映射（对象关系（O/R）映射工具）8.3.4 再次选择8.4 分类8.4.1 领域模型风格8.4.2 映射工具风格8.4.3 起点8.4.4 API焦

点8.4.5 查询风格8.4.6 高级数据库支持8.4.7 其他功能8.5 另一个分类：基础架构模式8.5.1 元数据映射：元数据的类型8.5.2 标识字段8.5.3 外键映射8.5.4 嵌入值8.5.5 继承解决方案8.5.6 标识映射8.5.7 操作单元8.5.8 延迟加载/立即加载8.5.9 并发控制8.6 小结第9章 应用NHibernate9.1 为什么使用NHibernate9.2 NHibernate简介9.2.1 准备9.2.2 一些映射元数据9.2.3 一个小的API示例9.2.4 事务9.3 持久化基础架构的需求9.3.1 高级持久化透明9.3.2 持久化实体的生命周期所需的特定特性9.3.3 谨慎处理关系数据库9.4 分类9.4.1 领域模型风格9.4.2 映射工具风格9.4.3 起点9.4.4 API焦点9.4.5 查询语言风格9.4.6 高级数据库支持9.4.7 其他功能9.5 另一种分类：基础架构模式9.5.1 元数据映射：元数据类型9.5.2 标识字段9.5.3 外键映射9.5.4 嵌入值9.5.5 继承解决方案9.5.6 标识映射9.5.7 操作单元9.5.8 延迟加载/立即加载9.5.9 并发性控制9.5.10 额外功能：验证挂钩9.6 NHibernate和DDD9.6.1 程序集概览9.6.2 ISession和存储库9.6.3 ISession、存储库和事务9.6.4 得到了什么结果9.7 小结第四部分 下一步骤第10章 博采其他设计技术10.1 上下文为王10.1.1 层和分区10.1.2 分区的原因10.1.3 限界上下文10.1.4 限界上下文与分区有何关联10.1.5 向上扩展DDD项目10.1.6 为什么对领域模型——SO分区10.2 SOA简介10.2.1 什么是SOA10.2.2 为什么需要SOA10.2.3 SOA有什么不同10.2.4 什么是服务10.2.5 服务中包括什么10.2.6 深入分析4条原则10.2.7 再来看一下什么是服务10.2.8 OO在SOA中的定位10.2.9 客户-服务器和SOA10.2.10 单向异步消息传递10.2.11 SOA如何提高可伸缩性10.2.12 SOA服务的设计10.2.13 服务之间如何交互10.2.14 SOA和不可用的服务10.2.15 复杂的消息传递处理10.2.16 服务的可伸缩性10.2.17 小结10.3 控制反转和依赖注入10.3.1 任何对象都不是孤岛10.3.2 工厂、注册类和服务定位器10.3.3 构造方法依赖注入10.3.4 setter依赖注入10.3.5 控制反转10.3.6 使用了Spring.NET框架的依赖注入10.3.7 利用PicoContainer.NET进行自动装配10.3.8 嵌套容器10.3.9 服务定位器与依赖注入的比较10.3.10 小结10.4 面向方面编程10.4.1 热门话题有哪些10.4.2 AOP术语定义10.4.3 .NET中的AOP10.4.4 小结10.5 小结第11章 关注UI11.1 提前结语11.2 模型-视图-控制器模式11.2.1 示例：Joe的Shoe Shop程序11.2.2 通过适配器简化视图界面11.2.3 将控制器从视图解耦11.2.4 将视图和控制器结合起来11.2.5 是否值得使用MVC11.3 测试驱动的Web窗体11.3.1 背景11.3.2 一个示例11.3.3 领域模型11.3.4 GUI的TDD11.3.5 Web窗体实现11.3.6 小结11.3.7 用NMock创建模拟11.4 映射和包装11.4.1 映射和包装11.4.2 用表示模型来包装领域模型11.4.3 将表示模型映射到领域模型11.4.4 管理关系11.4.5 状态问题11.4.6 最后的想法11.5 小结11.6 结束语第五部分 附录附录A 其他领域模型风格附录B 已讨论的模式的目录

章节摘录

插图：第一部分　背景知识第1章　应重视的价值，也是对过去几年的沉重反思本章的目的是布置场景。
本章将回顾过去几年中我如何思考不同概念，以及我的思想是如何随时间改变的。
我们将围绕很多话题讨论并涵盖大量基础知识，但总体思想是讨论在架构和开发过程方面应该重视的价值。
在这个过程中，我们将介绍并讨论很多概念，本书后面将对这些概念进行深入讨论。
那么，让我们开始吧！
1.1 总体价值过去，我很擅长提前计划。
我经常前瞻性地为项目添加功能、结构和机制。
所添加的工件本身是非常好的，但我总是忘记它们从未发挥任何好的作用。
当然，我也为项目增加了不少负担。
成本是相当高的，包括开发时间和增加的复杂性。
在过去几年中，我们被鼓励使用另一种方法：“做可能管用的最简单的事。”在很大程度上，这个思想来源于极限编程（Extreme Programmin9，XP）运动[Beck xP]。
另一种非常类似的说法是“你将不需要它”（YouAren’tGoingtoNeedIt，YAGNI），这是帮助人们保持循规蹈矩的一种好方法。
我猜想“保持简单，傻瓜”（Keep It Simple Stupid，KISS）也应属于此类。
这两种方法（预先添加所有能想到的与做最简单的事）是两个极端，但我认为它们都忽略了某些事情，即没有考虑到折中。
正如所有关于“这个最好，还是那个最好”的问题一样，答案是“它取决于什么条件”。
这是一个有关折中的问题。
我倾向于选择这二者中间某个位置的方法，并根据情况选择向哪个方向偏移。
“lagom”这个词是一个瑞典语单词，它表示“刚刚好”或“不多也不少”这样的意思。
Lagom或“保持平衡”连同上下文敏感性就是我认为应该重视的总体价值，还有就是持续学习。
让我们更仔细地观察一些更具体的价值领域（架构和过程组成），我们从架构开始，以便将我们引入正确的语境。

媒体关注与评论

“本书向读者展示了如何将测试驱动设计、对象-关系映射和领域驱动设计等方法应用于.NET项目……书中介绍的技术在很多开发人员看来是未来软件开发的关键……随着技术越来越强大，复杂度越来越高，理解如何更好地使用技术也变得越来越重要。
本书在推进这种理解方面迈出了可贵的一步。
”　　——Martin Fowler，ThoughtWorks公司首席科学家，《重构》与《企业应用架构模式》作者“学习领域驱动设计的最好方法是坐在一位友好、耐心且经验丰富的从业者身边，一步一步地共同研究问题。
阅读本书正是这种体验。
”　　——Eric Evans，领域驱动设计创始人

编辑推荐

《领域驱动设计与模式实战》：.NET开发人员必读之作，带领读者踏上领域驱动设计世界的实用、博学之旅。
《企业应用架构模式》与《领域驱动设计》两大名著精髓的实战演练。
不仅足够详细地解释了基本思想，而且将一系列思想综合到一起，帮助开发人员从头到位了解整个路线。
教你穿越业务层、数据层和UI层之间重重障碍，打通任督二脉。
Martin Fowler和Eric Evans两位大师联袂推荐。
模式、领域驱动设计和测试驱动开发赋予架构师和开发人员前所未有的能力。
使他们能够创建功能强大、健壮且可维护的系统。
但是，如何在实际项目中充分发挥这些利器的潜力呢？
《领域驱动设计与模式实战》中，作者将Martin Fowler《企业应用架构模式》和Eric Evans《领域驱动设计》两部经典名著中的思想精髓以及重构、测试驱动开发等技术融会贯通，并通过大量C#实例加以阐释，跨越了领域模型、数据库与UI层之间的障碍。
真实展示了创建高质量的企业级应用架构的全部过程。
《领域驱动设计与模式实战》就像是精彩纷呈的旅行见闻，每一处的所思所想都闪耀着智慧的光芒。
生动诠释了作者对面向对象开发中各种设计选择的深刻理解。

<<领域驱动设计与模式实战>>

版权说明