大连理工大学

硕士学位论文

Java EE项目开发与领域驱动设计实践

姓名：孙全智

申请学位级别：硕士

专业：计算机技术

指导教师：林鸿飞

20091201

摘　要

软件所要专注的业务领域十分复杂与千变万化造成了软性的复杂性，于是开发人员在不断研究和寻找更好的软件构建方法，从面向过程到面向对象，从EJB企业应用到轻量级开发框架，软件工程的理论和方法在不断进化与发展，研究如何快速有效地开发高质量的应用系统有着十分重要的现实意义，而领域驱动设计（Domain-Driven Design，通常简称DDD）就是新近出现和成长的一个重要理论和方法。

领域驱动设计是面向对象设计的一个重要方面，它的中心内容是研究如何将业务领域概念映射到软件系统中，强调以领域模型组织业务模型，领域模型是应用程序问题域的对象模型，因此领域建模的成败很大程度决定着软件项目的成败。领域驱动设计的领域建模的基本理论并不复杂，如何把理论实际应用到软件项目的开发过程中才是最关键的，但截至目前，市面上尤其是中文材料中有关内容的详细论述并不多见，本文结合具体项目，使用领域驱动设计开发的理论指导、运用 Java EE 技术开发了一个名为E-Learning的在线学习网站，并总结一套高水平的、适合中小型应用的运用轻量级开发框架的开发方法。

本论文主要分四个部分。第一部分概要介绍领域驱动设计的基本理论，重点讨论了领域驱动设计中的领域分层方法和领域模型分类。第二部分结合 E-Leraning 的建模过程详细阐述领域驱动设计中的六种领域模型的概念和抽象方法，介绍了典型的遵循领域驱动设计开发框架所必须支持的特征，这其中包括简单 Java 对象（POJO）、依赖注入（DI）、面向切面编程（AOP）和对象关系映射（ORM）等，以及如何应用 Spring、Hibernate 等 Java EE 开发工具来支持这些特性。第三部分结合了国内自行研发的 DDD 框架 JdonFramework 展示了 E-Learning 系统的部分模块的开发过程，着重介绍其较为独特的贫血式领域模型规范和高效的开发流程设计。最后，本文对 E-Learning 项目的最终产品的系统测试和运行情况作了一些简要介绍。

关键词：领域驱动设计；轻量级开发；Java EE；领域建模

Java EE Project Development and Domain-Driven Design Practices

Abstract

The business area software needs focus on is very complex and ever-changing, which resulted in the complexity of software. So the developers continue to research and find better approach to build software. We can see that from the process-oriented to object-oriented or from EJB to lightweight development, the theory and methods of software engineering are keeping continuous evolution and development. It has great practical significance to study how to develop high-quality applications quickly and efficiently. And the Domain-Driven Design (DDD) is the newly emergent and growing important theory and methods.

Domain-driven design is an important aspect of the content of object-oriented design. Its core content is to study how to map business domain concepts into software systems, and emphasize to organize business model with domain model. Domain model is the object model of an application problem domain, so the success or failure of modeling is a large extent determine to the success or failure of software projects. In domain-driven design, domain modeling is not complicated, but it is the most crucial how to apply theory to practical development process of software project. But so far, the market particular the Chinese material is not discussed in detail about related contents. In this paper, we use the theory of domain-driven design guiding the development of a online learning websites named "E-Learning", and summarize a set of high-level, suitable for small and medium sized application development framework for the use of lightweight development methodology.

This paper mainly consists of four parts. The first part is an overview of the basic theory of DDD, which focuses on the hierarchy method and domain model classification of DDD. Combined with domain modeling phase of "E-Learning" projects, the second part states in detail six domain model concepts and abstract methods and introduces the typical supportive features of DDD development framework, which includes POJO, DI, AOP and ORM, as well as how to use Spring, Hibernate and other Java EE development tools to support these features. The third section describes the outstanding open source frameworks – JdonFramework in domestic research and DDD, which focuses on the specific part of the development for "E-Learning" project. Finally, the paper makes some brief introduction on system testing and operation of final product testing and operation by applying domain-driven design.

Key Words: Domain-Driven Design; Lightweight Development; Java EE; Domain Modeling

大连理工大学学位论文独创性声明

作者郑重声明：所呈交的学位论文，是本人在导师的指导下进行研究工作所取得的成果。尽我所知，除文中已经注明引用内容和致谢的地方外，本论文不包含其他个人或集体已经发表的研究成果，也不包含其他已申请学位或其他用途使用过的成果。与我一同工作的同志对本研究所做的贡献均已在论文中做了明确的说明并表示了谢意。

若有不实之处，本人愿意承担相关法律责任。

学位论文题目：Java EE项目开发与领域驱动设计实践

作者签名：孙全智　　日期：2009年12月20日

大连理工大学学位论文版权使用授权书

本人完全了解学校有关学位论文知识产权的规定，在校攻读学位期间论文工作的知识产权属于大连理工大学，允许论文被查阅和借阅。学校有权保留论文并向国家有关部门或机构送交论文的复印件和电子版，可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印、或扫描等复制手段保存和汇编本学位论文。

学位论文题目：Java EE项目开发与领域驱动设计实践

作者签名：刘金智　　日期：2009年12月20日

导师签名：　　日期：2009年12月20日

引　言

在过去的 20 年中，软件行业发展和使用多种创建软件产品的理论和方法，到了今天，以 Java、C#语言为代表的面向对象(Object Oriented,简称 OO)语言与设计方法已全面取代了以往 VB、Delphi 为代表的面向过程的程序设计方法。根据面向对象理论，我们可以通过对象及对象之间的关系来反映软件所专注的领域，以面向对象语言来描述领域。简言之，有了面向对象语言及其衍生的 OO 设计方法，软件设计可以更专注于领域本身，而不是具体的技术细节。

尽管有了 OO 技术的支持，但我们常见的软件开发与设计流程通常还是这样的：先从客户那里详细了解软件需求，然后从建立数据模型开始，画出数据模型图、ER 图，考虑数据库表的结构，然后围绕数据模型开始编码，最终实现客户需要软件。这期间可能采用瀑布流程法一次性开发，或是采取多次迭代，运用了足够多与先进的编程技术，但软件设计的核心仍然是数据，而对象只是数据的附属品，它的构建与设计往往是从数据反推回来的。由此带来的问题是，与书本上所说的那些面向对象技术能给我们带来的那些优越性相比，软件设计人员和开发人员感受到的更多的是所带来的麻烦。最常见的问题是，任何数据结构上的变动都可能会连带修改整个 OO 多层架构上的所有相关对象的属性和方法，任何并不复杂的业务修改都必须进行十分谨慎的开发和大量的调试，而在两层架构的 VB 程序里，类似的修改就要便捷的多。但我们又绝不可能为此而倒退到 VB 时代去，于是人们开始重新审视软件的构建，看看到底是什么最终决定软件的价值。

Eric Evans 在 2004 年出版的《领域驱动设计—软件核心设计复杂性应对之道》一书通过大量丰富的实例，从事实和理论都给了我们一个十分令人满意的解答。领域驱动设计方法主张软件既然是被设计来解决领域问题，那么领域问题如何被映射到软件系统才是软件设计的关键，必须围绕领域模型构建系统才能更有效、准确地构建用户需要的软件系统，而让抽象的数据与数据库技术只作为软件架构中数据持久层的专有技术存在。

Evans 对领域对象的定义并不复杂，按照其在软件系统中扮演的角色划分为实体、值对象和服务三种类型，而为了管理领域对象的生命周期，又引入了工厂和仓储两个领域概念。领域驱动设计追求使用单一的领域模型同时满足分析原型和软件设计两方面的需求，建模和设计成为单个迭代循环，将领域模型和设计紧密联系。

领域驱动设计理论从其诞生之日起就伴随着最先进的 Java 技术与理念共同成长，这其中包括依赖注入（DI）和面向方面编程（AOP），如果没有这两种技术，就无法解决程序的耦合与对象之间的依赖问题，目前已被认为是领域驱动设计实践的必备要件。

对象关系映射(ORM)技术在领域驱动设计中的地位也越来越重要，它打破了前面所说的那种对象和实体之间的天然阻抗，被认为是领域驱动设计中持久层的最佳实现方案。

目前一些重要的开发框架都已经可以很好的集成与支持领域驱动设计的开发，其中最优秀的产品是 Spring。国内对于领域驱动设计的研究和讨论最早开端于 Jdon 社区（www.jdon.com），并且有了一个十分优秀的开源产品——JdonFramework。JdonFramework 是在领域驱动设计思想指导下出现的优秀的开源框架，并且完全由国人所倡导开发，它在设计上运用了 DI 和 AOP，并且可以轻松支持 Hibernate 为代表的 ORM 框架。

本文将通过一个叫做 E-Learning 的日语学习网站项目为例，用 Java 企业级开发（Java Enterprise Edition，简称 Java EE）的开发工具来实践领域驱动设计的理论和方法。该系统主要有以下几个基本功能：

（1） 学员注册登录功能。

（2） 课程管理功能。该部分功能包括课程发布管理和学习进度查询两大部分，课程发布管理主要面向网站管理者，用于网站学习内容、题库内容的管理。学习进度查询同时面向网站管理员和注册学员开发，用于了解学员的学习程度。

（3） 在线学习功能。注册学员可以在网站上学习指定的课程，系统记录学员的学习进度。

（4） 在线考试功能。注册学员通过在网站上的答题，得到自己的学习程度的反馈。

无论是采用 Spring 或 JdonFramework 都可以作为这一系统开发的核心框架，Spring 可以在伸缩性和功能性上为系统开发提供良好的支持；JdonFramework 则具有更为优秀的单机性能，同时相对于 Spring 是一个更为轻量的开发框架。为了论述领域驱动设计思想在轻量级开发的指导作用，本文在讲解 E-Learning 系统的设计和开发的过程中，将兼顾两个开发框架的特点来做比较和说明。

1 领域驱动设计的理论和方法

领域驱动设计（Domain-Driven Design，通常简称为“DDD”）是 Eric Evans 在 2004 年出版的《领域驱动设计——软件核心复杂性应对之道》（原文：*Domain-Driven Design Tackling Complexity in the heart of Software*）一书中所总结的一种软件设计模式，这一开发模式提供了一个建模和软件设计的系统方法，可以有效帮助和指导复杂业务需求的软件系统的开发与维护。

Java EE 是建立在 Java 平台上的企业级应用的解决方案，它提供了对 OOP 和 Java 技术的良好支持，是用来领域驱动设计实践的良好平台。

1.1 什么是领域驱动设计

“领域”是软件开发的起始点。比如，做一个银行的软件系统必须要充分了解其背后的金融业务，金融业务就是我们要开发的软件的领域。软件必须要根植于它所服务的领域，要能够反映领域里重要的核心概念和元素以及它们之间的关系，这样才能很好的应对变化。

软件设计人员可以从领域专家那里学习领域知识，并与之合作，共同对领域知识进行迭代、加工，将其转化为领域模型，但实际项目中，客户和领域专家往往不懂软件，而软件人员又往往不了解领域知识，为了解决这个矛盾，领域驱动设计引入了通用语言。所谓通用语言，是建立在领域基础上的一套表达工具，可以是图表，可以是文档，甚至可以是简单的几个文字，关键就是要能够缩小客户、领域专家和软件人员之间的交流沟壑。因此领域建模的目的并不是要得到某种特定的设计图形，而是其传达的思想——经过严格组织并进行选择性抽象的知识。

领域建模过程是必须的并且会贯穿软件设计和开发的全过程。领域模型是软件设计的最基础部分，会被用来解决项目范围的复杂的问题。

良好的领域模型会给软件项目提供一个很好的架构基础，随之才能进行程序设计和编码等后续的开发工作。优良的设计会加速开发的过程，开发过程中的反馈也会进一步优化设计，领域驱动设计就是一门结合设计和开发实践的软件开发模式，它可以很好地展示设计和开发如何协同工作以创建一个更好的解决方案。

1.2 绑定模型与实现

领域建模的目的是创建一个优良模型，这个模型不仅可以帮助早期的分析，同时也是设计的基础。之后我们要进行的是将模型实现成代码。

将模型实现成代码的过程通常被称作模型驱动设计（Model-Driven Design），建模与编码是领域驱动设计开发中同等重要的两个阶段。

为了紧密捆绑起实现和模型，通常需要支持建模范型的软件开发工具和语言，面向对象编程（OOP）非常适合对模型的实现，它提供了对象的类和类之间的关联关系、对象实例以及对象实例之间的消息通信，让建立模型对象、对象关系与它们的编程副本之间的直接映射成为可能。

领域驱动设计特别强调模型与设计的结合，追求让代码成为模型的表达式，以达到对代码的变更就成为对模型的变更的目的。

图 1.1 是模型驱动设计模式的导航图，它展现了使用领域驱动设计之后的对象建模和软件设计的最基本元素。

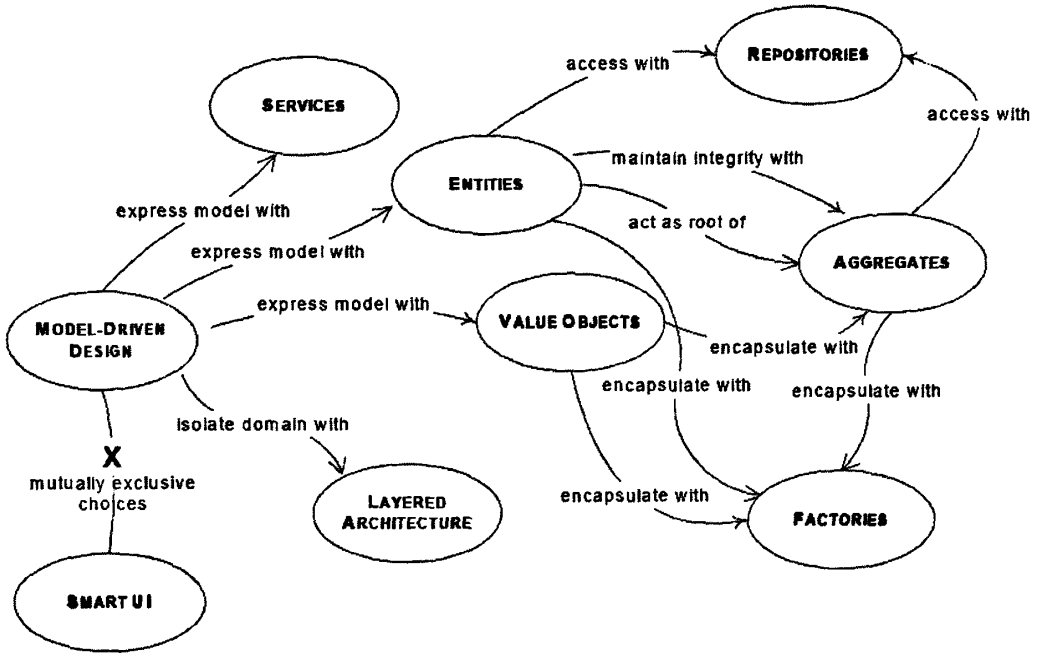

图 1.1 模型驱动设计模式的导航图

Fig. 1.1 Navigation map of Model-driven design

领域驱动设计必须遵循分层架构的设计原则。分层架构是 OOP 的一个重要概念，它指的是将一个复杂的程序切分成层，每层负责特定的功能，层与层之间保持松散关联，每一层只依赖于它的上一层，并向下一层提供服务，程序通过层与层的信息交互最终实现用户要求的功能。通过分层架构可以很好的管理代码，提供程序的可维护性。

在领域驱动设计中，Evans 规定一个通用领域驱动设计的架构性解决方案应包含四个概念层（如表 1.1 所示）。

表 1.1 领域驱动设计的分层架构

Tab. 1.1 Layered architecture of Domain-driven design

层	说明
用户界面 / 展现层	负责向用户展现信息以及解释用户命令。
应用层	很薄的一层，用来协调应用的活动，它不包含业务逻辑，不保留业务对象的状态，但它保有应用任务的进度状态。
领域层	本层包含关于领域的信息，这是业务软件的核心所在。在这里保留业务对象的状态，对业务对象和它们状态的持久化被委托给了基础设施层。
基础设施层	本层作为其他层的支撑库存在。它提供了层间的通信，实现对业务对象的持久化，包含对用户界面层的支撑库等作用。

领域驱动设计的分层架构的主要特点在于将领域层从基础设施层与用户界面层中分离出来。前期抽象出来的领域模型属于领域层，用来展现领域对象的显示自己、保存自己、管理应用任务等职责，这会让一个模型富含知识，更清晰地捕获基础的业务知识，让它们正常工作，而不应该涉及基础设施类的活动。用户界面既不跟业务逻辑紧密捆绑也不包含通常属于基础设施层的任务。另外，应用层是必要的，它的用途在于成为业务逻辑之上的管理者，用来监督和协调应用的整个活动。

为了实现各层的最大解耦，使用 IOC/DI 容器是目前最好的技术解决方案。

1.3 领域驱动设计的模型描述

来自不同问题域的领域模型差异极大，模型可以按照他们在领域模型中的角色进行分类，模型的角色隐含着特定类型的责任以及它和领域模型中的其它类的关系。领域驱动设计中的模型的角色划分通常包含如下六种：实体(Entity)、值对象(Value Object)、服务(Service)、聚合(Aggregate)、工厂(Factory)、仓储(Repository)。其中实体、值对象、服务是基本的领域模型的基本元素，聚合、工厂、仓储是为了管理领域对象的生命周期、维护对象的完整性而引入的领域模式。

在图 1.1 已展示了上述模型在领域驱动设计中是如何关联、交互的，下面再简要介绍一下它们的确切含义。

实体：是具有唯一业务标识的对象。该标识和对象属性分离，即使两个对象的属性值完全相同，两个实体也不相同，不能交换使用。实体通常对应于现实世界的概念，是领域模型的中心。

值对象：没有唯一标识的对象。主要用来描述领域的特殊方面的属性。值对象通常不可变，即一旦创建就不能更新，如果两个实例的属性具有相同值，他们就可以交换使用。

服务：用来实现无法指派给单个类的责任，并封装领域模型。服务不应实现大量业务逻辑，而应该通过仓储获取对象，然后委派给服务类。

聚合：是针对数据变化可以考虑成一个单元的一组相关的对象。主要用来定义对象所有权和边界，避免出现盘根错节的对象关系网。每个聚合都有一个根和一个边界。

工厂：用来封装对象创建所必需的知识，对于创建聚合特别有帮助。

仓储：用来管理既存的实体，主要用来查找和删除实体。仓储通常封装了持久层的框架，由接口和及其实现类组成。

在领域驱动设计中，如何将繁杂的领域知识准确合理地抽象为上述领域模型，是软件设计阶段的最重要工作。

1.4 持续重构

通过前文的六种领域模型的运用，模型现在已经跟它所源自的领域紧密关联，下面要做的是围绕模型展开代码设计。

代码设计首先必须围绕模型展开，脱离了模型的设计会导致软件不能反映它所服务的领域，甚至可能得不到期望的行为。

其次，在设计和开发过程中必须注意进行重构，重构分为代码重构和设计重构。

代码重构是不改变应用行为而重新设计代码以让它更好的过程。重构通常是非常谨慎的，按照小幅且可控的步骤进行，以避免带来程序 Bug。应用自动化测试可以为我们确保未破坏任何事情提供很大的帮助。

设计重构是由于我们在代码设计过程中有时会对领域有新的理解，有些事物会变得更清晰，或者元素间的新关系被发现，此时需要先停下手头工作，重新理顺与设计模型，进行面向深层理解的重构，直至让代码真正捕获模型的主旨。

一个忽略表面内容且捕捉到本质内涵的深层模型会让软件更加和领域专家的思路合拍，也更能满足用户的需要。要想获得深层模型，必须使用迭代的重构过程，并且需要领域专家和开发人员一起密切关注对领域的学习。

1.5 保持模型的一致性

当多个团队开发一个项目时，代码开发是并行的，每个团队都会被指派模型的一个特定部分，那些部分不是独立的，多少都有些关联性。它们都是从一个大的模型出发，然后实现其中的一部分。甚至可以说开发过程就是每个团队先创建自己的模块，然后提供给其他的团队使用。由此造成的问题是每个团队在使用别人开发的模块时都可能发现还缺少一些功能，于是就自行增加了这些功能并放到代码库里面，但这种变更很有可能就无意识中破坏了系统的整体模型，最终让开始良好的模型和流程到最后变得一塌糊涂。

怎么样才能解决这个难题呢？领域驱动设计认为，由于在开发流程中团队需要高度的独立性，统一的企业模型是不容易实现的理想状态，试图为整个企业项目维持一个大的统一模型以获得模型完整性的做法是无法真正有效实现的。因此应该做的是有意识地将大模型分解成数个较小的部分。只要遵守相绑定的契约，整合得好的小模型会越来越有独立性。让每个模型都有一个清晰的边界，最终模型之间的关系也就被精确地定义出来了。

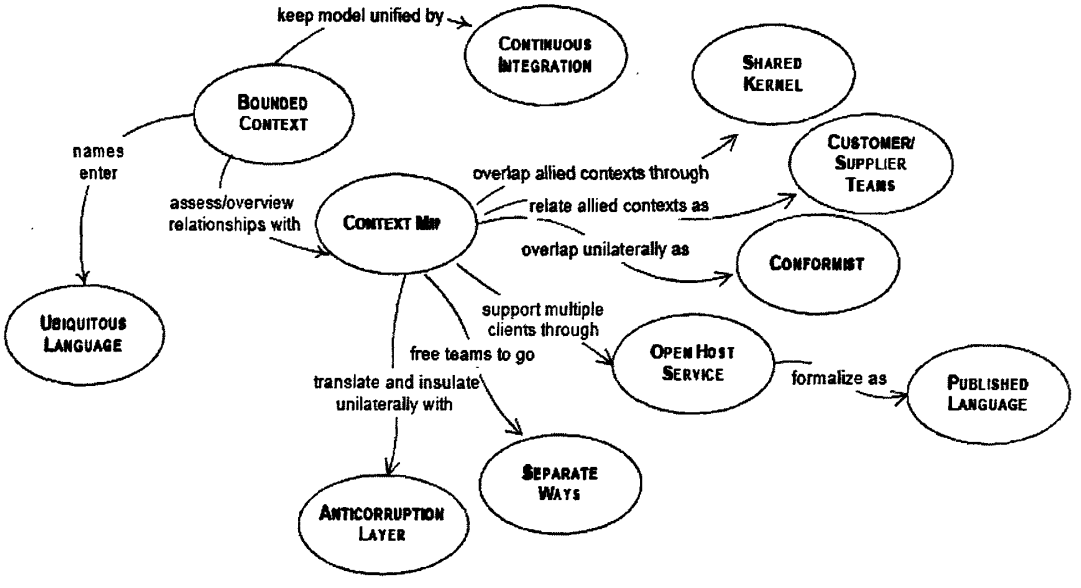

图 1.2 模型完整性模式的导航图

Fig. 1.2 Navigation map of model integrity patterns

对于这一主题，领域驱动设计提供了一整套可用的方案来保持模型的完整性。应用这些方案可以有效帮助DDD的项目管理。鉴于篇幅所限，本文将结合E-Learning项目重点讨论关于领域驱动设计中有关领域建模和代码实现的方法，关于这一主题的更多细节将不在本文进行深入探讨。

1.6 小结

“领域驱动设计”作为一个名词诞生的时间并不久，但组成它的理论和技术其实早就在被广泛运用，Eric Evens做到的也许只是根据自己多年的实践经验来把它提炼成一种“方法学”上的理论，但在软件业界所引起反响却是巨大而深远的。美中不足的是尽管Eric Evens的原著里虽然有很多项目实例，但主要都是从系统架构师的层次上论述本章所介绍的这些理论和方法，而从软件开发的其他角色来看，帮助或许并不大，这就迫切需要我们在学习的过程中做到理论联系实际，自己来消化这些知识。

从目前的国内外对DDD的理论和方法的研究来看，在如何运用领域驱动设计进行开发的问题上有如下的一些共识：

（1）领域驱动设计的主要思想是在设计中将业务领域中的概念与软件元素对应起来，面向对象的编程方法（Object Oriented Programming，OOP）是DDD实现中的核心要素。OOP中的对象就代表现实中某个实体。因此在充分利用诸如继承、封装和多态等OOP概念的基础上，领域对象应被设计成简单类和接口。

（2）在一个典型的业务单元中，领域对象需要与其他对象协同工作，无论此对象是服务、仓储还是工厂。与此同时，诸如领域状态变化跟踪、审核、缓存、事务管理（包括事务重试）之类的横切关注点也是领域对象必须密切关注的。然而，这些东西大都是可重用的并且是非领域相关的，它们往往会散布或者重复出现在包括领域层的所有代码中。这些嵌入领域代码中的非领域逻辑往往会导致领域层的繁杂与混乱。为了很好管理代码间依赖，同时解耦领域对象与独立的横切关注点，只使用OOP技术不能为领域驱动设计提供一流的设计和开发解决方案。因此催生了依赖注入(DI)和面向方面编程(AOP)概念。它们作为OOP的补充，能很好地减少代码间紧密耦合、增加产品模块化特性和管理横切关注点，目前已成为应用领域驱动设计的重要技术标准。

（3）DDD并不依托于特定的开发平台，目前主流的Java EE和Microsoft .NET框架都有足够的开发工具支持DDD的开发。但其中Java EE平台代表着一种新的设计和编程体系，它由整个Java社区推动发展，和开源力量结合在一起为用户服务。事实上DI和AOP概念的出现和推广也都是伴随着Java EE平台的一些优秀框架（比如Spring）一起成长起来的，因此本文介绍的E-Learning系统采用了Java EE作为项目的支撑平台。

（4）实践 DDD 离不开较为成熟的开发框架（Framework），这样可以应用前人经验，避免“重复制造车轮”，最终有效减少自己项目的开发周期，目前比较经典的 DDD 开发框架组合有 Spring+Struts+Hibernate（SSH）等。

在以上共识的基础上，接着就可以结合 E-Learning 项目的开发，从软件开发的系统设计师、程序员、测试工程师的角度来做进一步的总结，从而了解运用领域驱动设计理论和方法进行开发的先进功效，并指导未来的其它项目的开发。

著名架构师 Ramnivas Laddad 推荐按照如下的步骤实现领域对象模型。他强调要更侧重于领域模型中的领域对象，而不是服务：

（1）一开始从领域实体和领域逻辑开始起步。

（2）只添加那些逻辑不属于任何领域实体或值对象的服务。

（3）利用通用语言、契约式设计、自动化测试、重构等技术，使实现尽可能地与领域模型紧密结合。

2 领域驱动设计实践

本文实践领域驱动设计的项目范例是G公司的Japanese E-Learning系统。G公司是最早进入中国开展BPO业务的外资企业之一，基于公司的业务特性，为了应对公司急剧扩大且复杂化的业务，提高员工的日语水平，公司早在2002年即在公司内部网站开通了Japanese E-Learning在线学习/培训系统，将公司培训部的所有教材与分散在各部门内的听力教材进行了数字化开发，在多年的运营过程中，该系统被实践证明是提高员工日语能力的有效途径之一。

对于大多数在当时开发的系统来说都是采用ASP或其它的两层B/S架构技术进行开发，虽然当前系统运行十分稳定，但是当公司决定将这套优秀的日语培训系统进行产品化改造将之推向市场时，原有的两层架构、面向数据的平台成为了制约项目成败的瓶颈，因此公司决定按照现有系统的模式，采用Java EE平台的软件技术重新开发该系统。

2.1 E-Learning系统的需求分析

与传统的OO开发模式一样，采用DDD进行软件开发的首步骤也是用例分析。通过对旧系统的分析与培训部门的交流，我们很容易地得到该系统的用例关系，如图2.1。

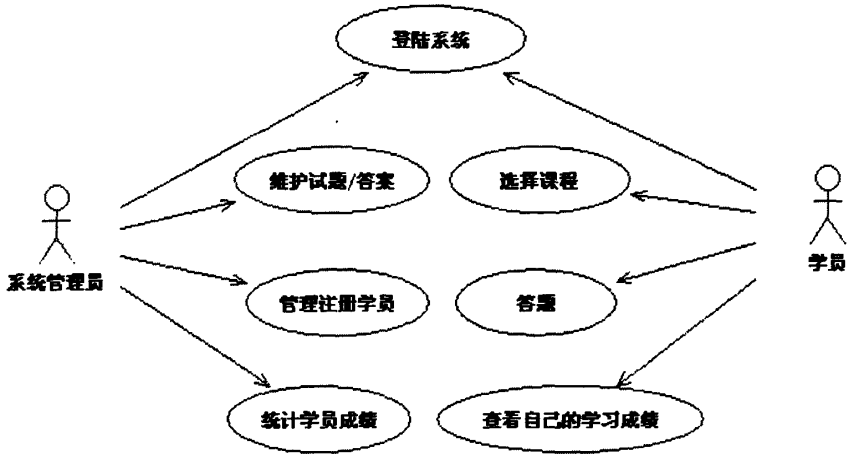

图2.1 E-Learning系统用例图

Fig. 2.1 Use case diagram of E-Learning system

由图2.1可知系统参与者只有系统管理员和学员两种角色。用例部分的“管理注册学员”、“登陆系统”通常可作为系统用例不计入项目的业务需求中；“统计学员成绩”、

“查看自己的学习成绩”可以简化为数据查询处理，因此主要的业务对象包含在剩余的三个个业务用例“维护试题/答案”、“选择课程”、“答题”当中，根据对现有系统的分析和与G公司培训部门的交流，我们首先得到了一张业务对象关系图，如图2.2。

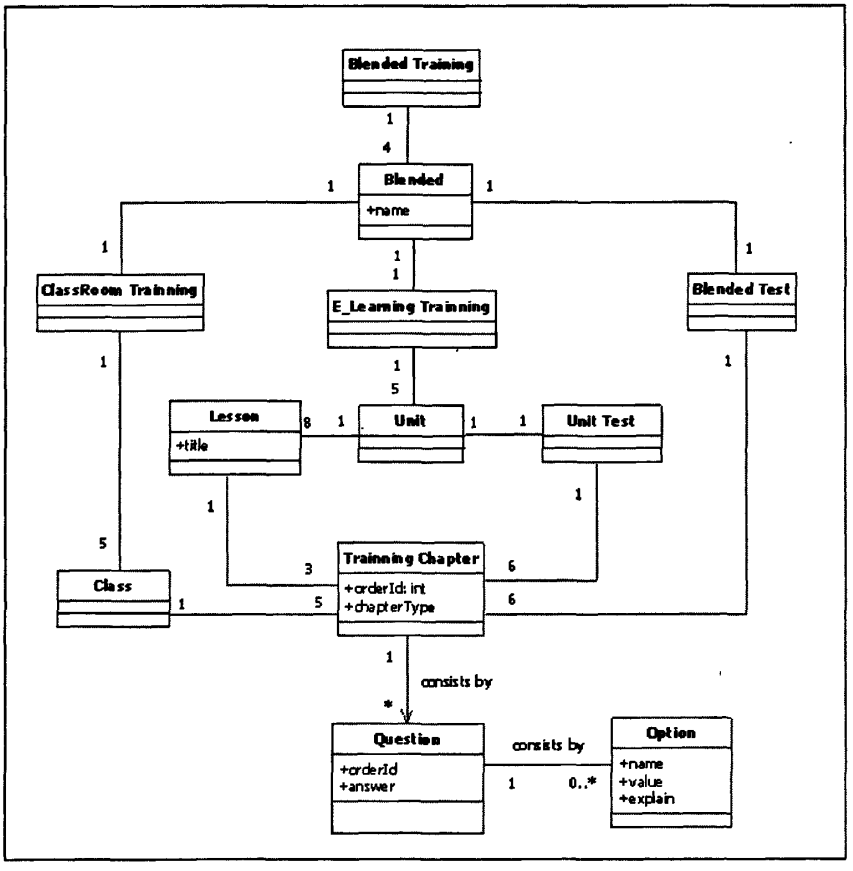

图2.2 E-Learning系统的E-R关系图

Fig. 2.2 Entity-Relation diagram of E-Learning system

对这张E-R关系图可解读如下（按照图2.2从上至下、从左至右的顺序）：

（1） E-Learning系统又被称作Blended Training系统，它由4个等级的混合训练课程（Blended）组成，其中课程的难度按等级递增。

（2） 每个Blended由课堂训练（ClassRoom Training）、在线训练（E-Learning Training）和等级测试（Blended Test）三部分内容组成。用户必须完成每个Blended的前两部分内容的学习才能进行Blended Test，当每个Blended Test通过考试后，则该Blended的学习全部完成。

（3） 每个ClassRoom Training由5个课程（Class）组成。

（4） 每个E-Learning Training由5个单元（Unit）组成，每个单元包含8组在线学习课程（Lesson）和一组单元测试（Unit Test）组成，用户必须学完一个Unit的所有Lesson的学习才能进行Unit Test，如果Unit Test通过，则该Unit的学习全部完成。

（5） 每个Class、Lesson都是由若干个训练章节（Training Chapter）组成的，Training Chapter必定属于听力、词汇、语法、句型、读写、听写中的一个固定类别，Training Chapter按照一定的策略与Class、Lesson搭配。

（6） 每个Unit Test或Blended Test也由若干Training Chapter组成，Training Chapter必定属于听力、词汇、语法、句型、读写、听写中的一个固定类别，Training Chapter按照一定的策略与Class、Lesson搭配，该策略主要是在所属的Unit或Blended内随机选题组成试卷，即与Class、Lesson、Unit Test、Blended Test共享题库。

（7） 每个Training Chapter是由若干个习题（Question）组成，习题分为填空和选择两种，每个题目都有管理员预设的答案（Answer）。不管是在线学习还是在线考试，用户选择或填写一个Training Chapter的所有习题的答案，然后提交给系统，系统自动对用户提交的内容进行评价给出分数。

（8） 当Question的题型是选择题时，每个选择题由若干个选项（Option）组成，每个选项都可以有管理员预设的选项说明（Explain，用于做Answer的说明）。

图2.2表达了最基本、朴实、未经雕琢的业务对象以及它们之间的关系，但它不能从深层次表达业务系统的内在逻辑，因此不可以直接用来进行项目开发，我们必须将其抽象为符合领域驱动设计规范的领域模型。

2.2 领域建模

2.2.1 E-Learning系统的领域分解

在开始领域建模前，先回顾一下前面1.2节介绍的领域驱动设计的分层架构原则，我们需要将E-Learning系统的功能按照DDD分层原则解析一下。

用户界面/ 展现层：接受用户输入（屏幕录入 ），向用户显示信息 （显示结果）。在DDD的系统框架中，我们可引入数据传输对象（DTO）来穿越DDD的分层架构来进行数据传输。

应用层：调配领域对象来解决问题，主要放置工厂和统一接口（POJO Facade）。系统的这个层本身不反映任何业务情况，它主要用作调配领域层的对象之间的协作，包括其与基础设施层的通讯等。

领域层：真正执行业务逻辑的层，需要通过领域建模的技巧来将需求分析阶段看到的业务，比如管理所有试题，察看学员学习情况，答题等，提炼为实体、值对象、服务等领域模型。

基础设施层：主要执行访问数据库（查询数据库，提交数据库更改，网络通信等），应用消息发送等操作。在软件开发采用的框架中，基础设施层还要负责支持四层之间的交互。

应用 DDD 进行软件开发，领域层和应用层的设计是重点。需要强调的是，领域建模是一个迭代的过程，不要寄希望于一次性获得稳定、完全反映业务的领域模型。目前绝大多数开发者提倡采取敏捷（Agile Methodology，简称 AM）的方式进行这部分工作，即建模、编码、单元测试的反复迭代，在客户的参与下、在迭代的过程中去逐渐提炼业务模型——并且有经过测试的代码作为每个小的迭代阶段的成果。

敏捷的相反面是瀑布式开发和 RUP，前者机械的割裂软件开发为分析、设计、编码和测试等阶段的开发模式早已过时；后者则过于重量级，过于强调软件质量控制，所以往往造成项目的文档十分繁杂，喧宾夺主（相比代码），较适用于大型软件团队开发大型项目。因此对于对于 DDD 来说，采用轻量级框架+敏捷开发模式是较好的选择。

2.2.2 实体、值对象与聚合

实体是 DDD 中最常遇见、也是最重要的领域对象。E-Learning 系统中的参与者“学员”是最显而易见的实体，因为在业务处理的过程中，我们必须对“学员”进行区分，这样才能知道每个人的考试成绩和学习情况，同时为了方便和避免重名，我们会使用“学号”来区分学生，“学号”就是区分不同学生的唯一标识。当使用了数据库后，我们会使用更为可靠的数据库主键来做唯一业务标识。总之，实体一定具有唯一业务标识。

值对象的情况与实体相比最大的不同是，我们需要知道它所表示的对象是什么，但不需要知道它到底是哪一个。在 E-Learning 系统中，“选择题的的选项”可以被认为是值对象，一个重要判断标准是系统判断学员做的的答案对不对，只要知道用户选的 ABCD 选项是不是题目指定的那个，而不需要知道 ABCD 对应的内容具体是什么。

实体和值对象的判断依赖于特定的上下文（Context），以“选择题的选项”为例，假设系统中有一个管理模块负责所有题目的选项的录入，那么在这个特定的环境下，每个“选择题的选项”既要知道它的具体内容，又要知道它对应哪个题目，就应被视为实体

值得注意的是我们往往会不经意地将系统中本应被定义为值对象的领域对象定义为实体，这种思维定势通常来自以往的“数据库驱动设计”的经验，在两层开发流行的

时代，数据库设计是项目开发的决定性环节，每个数据表中都会被添加一个主键以区分不同的记录，这个经验带入到 DDD 后，感觉上所有的对象都是实体。但实际在 DDD 中，数据库已经退化到只是数据持久的一种选择而已，依靠良好的领域对象设计和 ORM 工具，领域对象的设计可以自动转换为数据库脚本，通常字典表的主键 ORM 工具可以自己管理，完全不需要开发者进行干预。

并没有一个绝对的标准去判断哪些领域对象应该被视为实体，哪些应该被定义为值对象，对实体和值对象的判断要靠软件人员的经验，从领域对象的关系中进行深层挖掘，最终找到最符合用户需求的判断。

根据上述原则，经过抽象和提炼的过程，我们得到了 E-Learning 系统经过第一次抽象后的“题库” 模块的领域模型图（如图 2.3），在当前的设计中，我们把 “Training Chapter”作为实体，图中的其它的领域模型全部都是值对象。

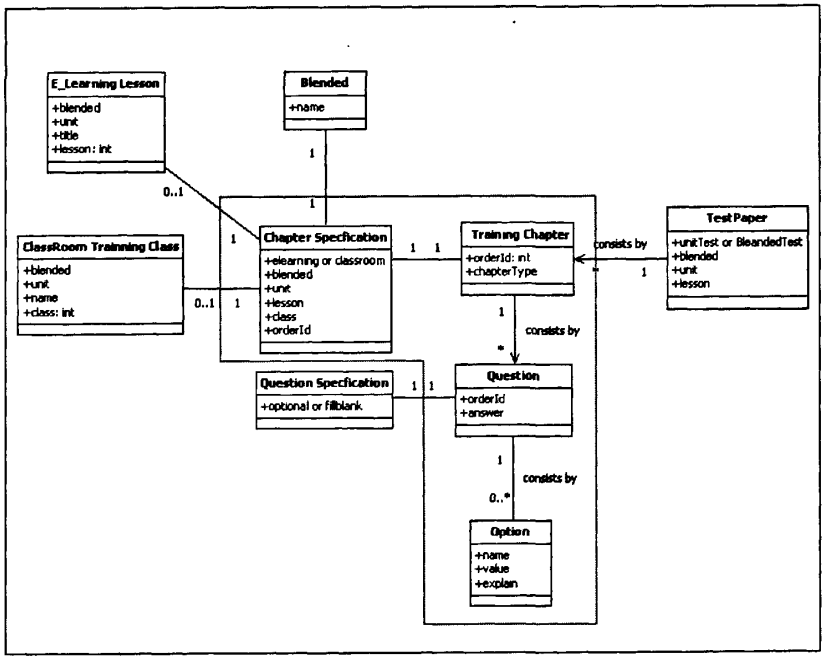

图 2.3　E-Learning 系统的实体和值对象

Fig. 2.3　Entity and value object of E-learning system

在图 2.3 中被线框包围的部分代表的是一个非常重要的领域概念——聚合，线框是聚合的边界，线框内的 Training Chapter 是聚合的根，线框内的 Chapter Specification 对象并不显式地存在于前面 2.2 节的 ER 图中，而它来自于是设计人员以及客户对其中 ER 关系的提炼，提炼出的对象归纳了 Training Chapter 的属性的关联使之更明了。

聚合主要用来表达领域对象的整体和部分的关系，边界定义了聚合中应该包含什么，根是包含在聚合中的单个特定实体，在实际应用中，只有聚合的根可以被外部进行引用，是外界访问聚合内部的唯一的接口；而在聚合边界内，对象之间可以相互引用。换言之，对图 2.3 聚合边界内的任何对象的属性进行更改，必须先获得一个 Training Chapter 对象，然后再通过关联导航找到要修改的属性进行修改，最后必须将 Training Chapter 对象整个进行提交完成修改；如果要对某个 Training Chapter 进行删除，那么与其相关联的所有聚合内部对象也必须要同时清理干净。只有这样才能满足聚合最重要的特性——不变量（invariant）。

在这个设计中有一个重要的领域对象 Test Paper 被作为了值对象而不是实体处理。这是因为在客户当前需求中，试卷只作即时的随机选题用，学员的每次答题后可以马上知道通过或不通过的结果，但试卷本身不需要做数据持久，客户并不要求对每一张试卷的情况进行跟踪，因此当前在这里采取值对象为试卷建模是一个合理的设计。

2.2.3 应用工厂创建领域对象

DDD 认为简单对象（主要是实体）的创建可以是它自身的主要行为而通过类的构造函数来进行创建，但复杂的组装行为不应该成为被创建对象的职责，因此引入了一个新的领域概念——工厂，由工厂来帮助封装复杂的对象创建过程，这对聚合的创建特别有用。

在 E-Learning 项目里， Training Chapter 作为聚合的根被建立时，所有聚合中包含的对象也要随之建立，以维持聚合概念所要求的不变量。

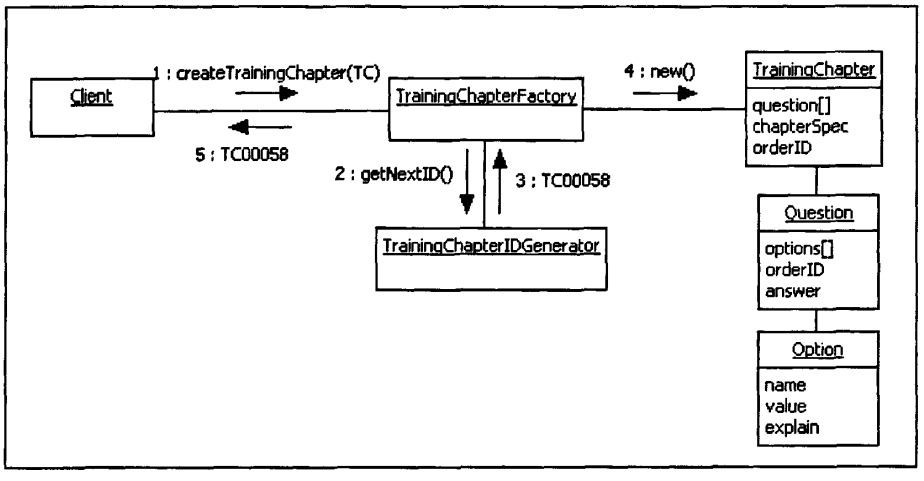

图 2.4 工厂原理示意图

Fig. 2.4 Factory diagram

图 2.4 演示了由工厂创建聚合的原理，在图中，TrainingChapterFactory 是创建 Training Chapter 对象的工厂，其中的 createTrainingChapter()方法中封装获取当前数据 ID 的逻辑。然后通过特定机制完成 TrainingChapter 聚合内所有关联对象的创建和数据持久。图中所示意使用的机制是 TrainingChapter 对象的构造函数，该构造函数去创建聚合中的 Question 和 Option 的机制是交由数据持久层框架自动完成的。

createTrainingChapter()方法的参数“TC”可以用数据传输对象（DTO）来实现，DTO 又被称作边界类，它的主要任务是穿梭软件各个分层进行传输数据的简单数据对象，它不包含任何业务逻辑处理，因此通常只有属性而没有方法，而且其属性应定义为基本数据类型、简单的时间类或其它 DTO。在 DDD 中，通常用户界面层的数据源和程序返回给用户界面层的对象应该是 DTO 而不是领域对象，这就要求 DDD 框架的用户界面层应能够自动化的组装 DTO，同时工厂依赖的业务模块能够对于传来的 DTO 进行解耦和重新封装，而数据持久层需要有机制可以将 DTO 自动映射为领域对象以便进行持久化。

2.2.4 应用仓储管理领域对象

工厂负责“创建”领域对象，并把对象持久化到数据库中，当客户程序需要使用其中一个对象时，最常见的获得方式是直接访问数据库，从中检索出对象并使用它。但请注意这种面向数据的处理方式是背离了领域驱动设计的思想的，因为这种方式不经过领域层就直接操作数据库会使得领域层成为了摆设；而且允许越过领域层进行数据库操作还极有可能破坏了聚合的封装性，带来未知的结果，从而使辛苦构建的领域模型处于不稳定状态。因此 DDD 引入了仓储这一领域模型，应用仓储来“重建”（reconstitution）已经存在的对象是领域驱动设计的一个十分重要的原则。

仓储的主要作用是封装所有获取对象引用所需的逻辑，领域对象不再必须通过基础设施（主要指数据库）才能得到领域中对其他对象的所需的引用，而只需从仓储中获取它们，这样便使得模型保持清晰以及凸显其在项目中的重要作用。对于面向过程经验根深蒂固的开发者来说必须转变思想，避免对数据库等具体实现机制的“念念不忘”，而把关注点放在领域，这样才能及早领悟领域驱动设计的精髓。

仓储的工作方式是：当一个对象被创建出来时，它会被直接保存到仓储中，然后以后再需要使用时就可从仓储直接检索到被缓存的对象中的数据。如果对象长时间不使用，它可能会被 DDD 框架主动从仓储中移除，这时如果客户程序从仓储中再次请求这一对象，仓储负责从存储介质（比如数据库）中去重新获取它，这一操作对客户程序是透明的。总之，仓储是作为一个全局的可访问对象的存储点而存在，它不仅承担数据库操作的责任，同时还承担数据缓存的作用。

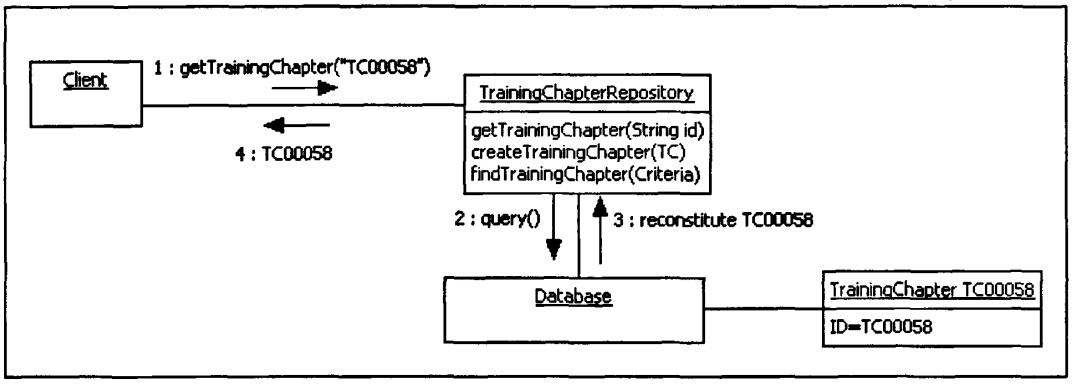

图 2.5 仓储原理示意图

Fig. 2.5 Repository diagram

图 2.5 展示了通过仓储重建对象的工作原理。getTrainingChapter()方法是以接口的方式公开给客户程序，客户程序并不需要知道其中的实现细节。并且程序只需要对真正需要直接访问的聚合根提供仓储，其它的对象的存储和访问等操作可封装给聚合根来实现。

当要求返回大批量数据时，可以使用规约（specification）来定义一个复杂的查询条件，例如图 2.5 中的 findTrainingChapter(Criteria)方法。

在图 2.5 中我们还可以观察到在仓储中有一个创建方法 createTrainingChapter()，这是否与工厂的职责发生冲突了呢？

事实上，工厂和仓储之间是有一定的关系的，它们都是领域驱动设计中的重要模式，帮助我们管理领域对象的生命周期，但工厂关注的是对象的创建，而仓储关心的是已经存在的对象。从技术的角度上，仓储也可以被看作一种工厂，不过它不是从无到有创建新的对象，而是对已有对象的重建。当一个新对象被添加到仓储时，它应该是先出工厂创建过的，然后它应该被传递给仓储以便将来保存它（参见图 2.6）。

因此系统中虽然是由工厂承担创建对象的职责，但具体到将对象持久到数据库的操作，是委托给仓储（实际是仓储的实现类，仓储本身只以接口的形式公布给客户程序）来进行真正的数据操作，而在仓储进行数据记录创建之前，仓储要把被持久的对象进行缓存，这样外界便可以直接通过仓储的查询接口直接获得到被要求的对象，只在被缓存的数据失效等必要的时刻，才到数据库里去重新激活它。

因此在实际编码时，仓储类必要时也完全可以代理工厂类的创建对象方法，我们只要注意区分工厂和仓储类的职责，保证把具体的操作交给符合 DDD 规范的实现类来处理就可以了。

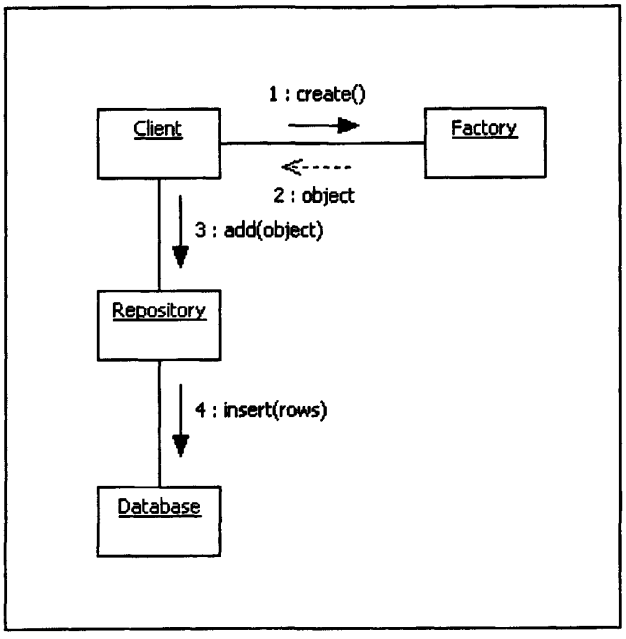

图 2.6 利用仓储与工厂保存新对象

Fig. 2.6 Use repository and factory to create new object

2.2.5 领域服务的设计原则

通常我们可以把对于实体或值对象增删改查功能看作是服务，但这种特定的领域概念的固有行为不是 DDD 所指的作为领域模型的“服务”。DDD 所指的服务是代表了领域中的某些重要的行为，但却不能简单的把它们合并到某个实体或者值对象中，并且这些行为通常会跨越若干个对象协调完成一个重要的领域操作，把这种行为抽象为独立的接口加入领域模型中才是 DDD 中所指的“服务”。例如 E-Learning 系统中，“答题”行为要协调“学员”、“考卷”、“训练章节”等多个领域对象，包含一定的业务规则，是系统不可分割的重要功能之一，因此显然应定义为服务。

服务类应采用显式声明，无状态，以保证客户程序可以任意调用。原则上当某个操作凸现为一个领域中的重要概念时，就需要为它建立一个服务了。

服务不仅有领域层服务，还有应用层服务和基础设施层服务，进行软件设计时要把他们能够区分出来，表 2.1 是本项目中答题服务的分层设计。

决定一个服务所应归属的层通常是比较困难的事情。一般如果所执行的操作概念上属于领域层，那么服务就应该放到领域层；如果操作和领域对象相关，而且确实也跟领域有关，能够满足领域的需要，那么它也应该属于领域层。

表 2.1 将服务分层

Tab. 2.1 Split service layer

层	说明
应用层	答题应用服务 - 读取输入，组装或析构 DTO。 - 发送消息给领域服务，要求处理。 - 监听领域层的确认消息。 - 决定用基础结构层的服务发送通告。
领域层	答题领域服务 - 协调 TrainingChapter、Question 等领域对象的相互作用，完整正确的结果判断。 - 判定用户每题的答案对错。 - 判定用户考卷是否通过。
基础设施层	发送通告服务 - 有应用选择通告方法，发送电子邮件、信件或者其他通信途径。

当完成一定阶段的设计之后，再将视线返回到领域服务中，判断一下当前面所有的领域对象都被抽象出来之后，是否每个领域服务的子功能都可以到其中的某个实体或者值对象的方法或者属性中找到对应的解决方案，如果每个答案皆“是”，这预示着领域建模已经接近成功，此时要做的就是持续重构，从设计和性能上继续完善我们的系统。

图 2.7 是 E-Learning 系统的选题、答题、判断学员考试成绩相关的服务的设计图。其中 ELearningPOJOFacade 是应用层接口类，TestService 是领域层服务类，还有基础设施服务类 ElearningUtility。TestService 依赖 TestPolicy 通过 initTestPaper() 方法创建 TesePaper，TestPolicy 模型中封装了 UnitTest 和 BlendedTest 选题相关的策略；学员的答题通过 TestPaperDTO 传入 TestService，然后其中的 answerTestPaper() 方法比较传入的 DTO 中每题的答案和 TestPaper 中的 TrainingChapter 中相关的 Question 的预设的答案并给出学员考试结果，再回写到 testPaperDTO 中。isBlendedTestPassed()方法和 isUnitTestPassed()方法判断用户的考试结果是否达到标准，客户程序根据该返回值更新学员学习档案，如果是通过的话，基础设施层服务还会给学员发送通知 Email。

图 2.8 是展示了 E-Learning 关于答题服务的应用层、领域层、基础设施服务层的交互细节的 UML 顺序图。

至此，通过本节的介绍我们不仅了解到进行模型驱动设计时的各种领域模型实践要素，并且在介绍的过程中已将 E-Learning 项目的核心用例涉及到的模型基本建模完毕。

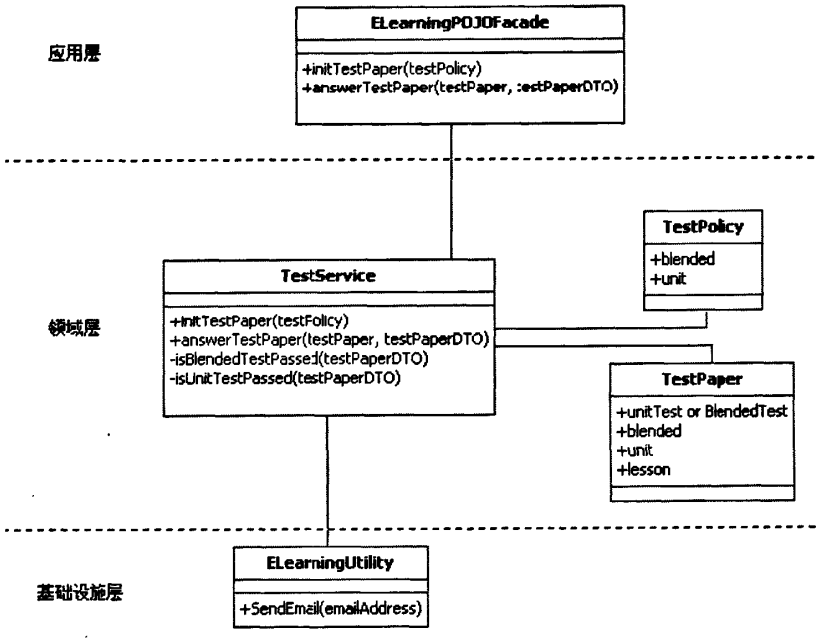

图 2.7　E-learning 系统的答题服务

Fig. 2.7　The service of answer question in E-learning system

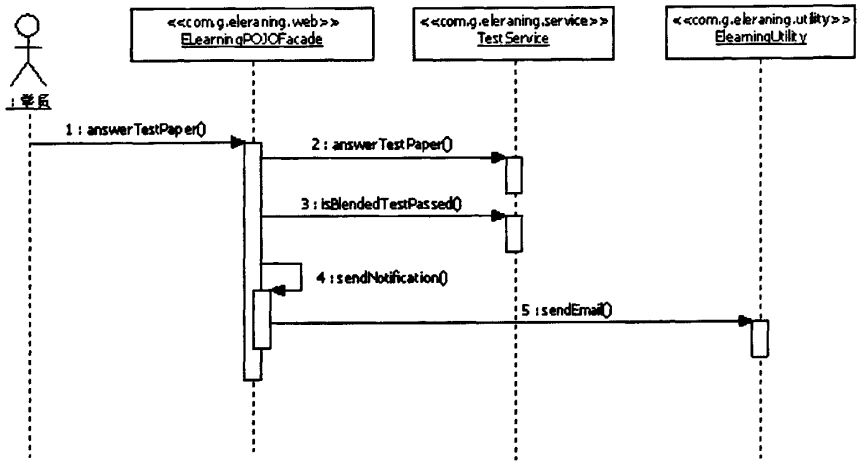

图 2.8　E-learning 系统的答题服务的顺序图

Fig. 2.8　Sequence diagram of answer question service in E-learning system

2.3 领域建模的编码实现

2.3.1 充血式领域模型

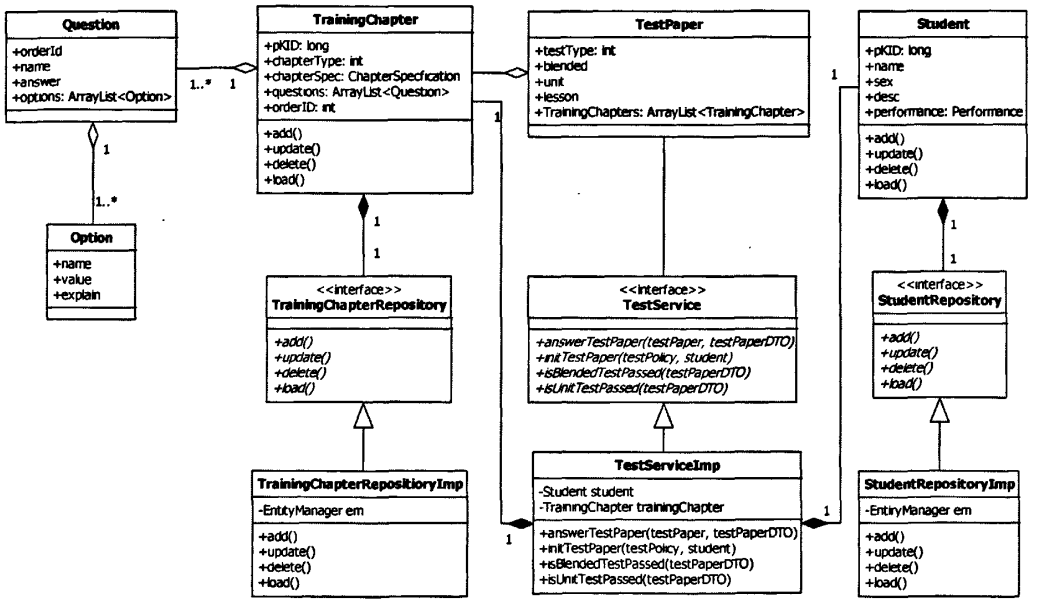

图 2.9 E-learning 系统的领域模型图

Fig. 2.9 Domain-Model of E-learning system

图 2.9 是根据 2.2 节的分析得到的一个可行的、同时也是 Eric Evans 在其著作里推荐的“充血”模型的 E-Learning 系统的核心部分的领域模型图。所谓充血模型是指实体类在设计上直接包含增删改查等实体固有行为的方法而不是只有属性和弱行为 setter 和 getter 方法，模型可通过自身的行为管理其生命周期。

这张图的设计已经接近尾声，很大程度上我们已经可以将它视为实现时的类图，根据这张图的设计，采用适当的 DDD 开发框架，例如 Spring+Struts+Hibernate，就可以进入到编码实施的阶段了。

采用如图 2.9 所示的领域模型进行系统开发时的要点在于：

（1） TrainingChapter 和 Student 是实体，Question、Option、TestPaper 是值对象。编码时，实体应设计为简单的 POJO 对象（Plain Old Java Object），不依赖于任何开发框架。

（2） TrainingChapter 同时是聚合根，承担聚合内部的 Question、Option 对象的创建和管理职责。

（3） 领域层只有一个核心的服务：TestService，它被声明为接口，有一个具体的实现类TestServiceImp。只需向客户程序公开TestService服务和实体、值对象即可完成系统的绝大多数核心功能。

（4） TrainingChapterRepository和StudentRepository是两个仓储类的接口，它们被公开给实体类作为实体类的属性。它们的实现类TrainingChapterRepositoryImp和StudentRepositoryImp承担真正的仓储职责。实体类的增删改查方法实际上是调用仓储的相应方法实现的。例如：

```
public class TrainingChapterImp implements TrainingChapterService {
    ……
    public void add(TrainingChapter trainingChapter) {
        try {
            ……
            trainingChapterRepository.add(trainingChapterDTO);
        } catch (RepositoryException re) {
            re.printStackTrace();
        }
    }
    ……
}
```

（5） TrainingChapterRepositoryImp和StudentRepositoryImp中持有的EntityManager属性的em对象代表的是系统采用的持久层框架的模版类，所有真正的数据库操作是由仓储调用EntityManager对象的方法来实现的，EntityManager可以是用户用JDBC控件自己实现的一个简单的持久模版，也可以用成熟的O/R Mapping持久层框架来实现。Hibernate等ORM框架是现阶段最为推荐使用的DDD持久层框架。

（6） TrainingChapterRepositoryImp和StudentRepositoryImp同时承担领域层工厂的职责。图中并没有显式的工厂类（例如TrainingChapterFactory），这是因为一方面这里的系统逻辑比较简单明晰，另一方面底层如采用ORM框架可以有效管理聚合的不变量，减少编程的复杂性，因此这里的设计将工厂和仓储类进行了合并。

（7） TestServiceImp和TrainingChapter、Student应采用依赖注入（DI）的方式进行调用以减少对象之间的依赖性。

（8） TrainingChapterRepositoryImp和StudentRepositoryImp的代码应采用适当的框架实现缓存，缓存的实现方式应采用框架提供的声明式事务与面向切面编程（AOP），以保证主程序的健壮和简单明晰。

（9）TrainingChapterRepositoryImp的代码应采用适当的事务控制机制来确保聚合的不变量。

下面将就对上面的要点中所涉及到的 DDD 重要技术进行详细阐述，从中我们可以了解和掌握进行领域驱动设计的最佳实践环节。

2.3.2 依赖注入（DI）

依赖注入（Dependency Injection，简称 DI）是一种软件设计模式，也称为微内核容器或者反转模式（Inversion of Control、简称 IoC）。表现在代码中将软件类之间依赖关系先剥离，然后在适当时候再注射进入。DI 模式实现了调用者和被调用者之间关系在某处完全分离，是目前解耦程度最高的设计模式。一个高质量的 DDD 实现是离不开 DI 的，DI 可以被认为是采用 DDD 开发项目的骨架。

下面是 E-Leraning 案例中应用 Spring 实现依赖注入的代码的一个简化的例子：

调用者 TrainingChapter 类

```
package com.g.elearning.model;
public class TrainingChapter{
    TrainingChapterRepository tcr;
    //构造方法
    public TrainingChapter (TrainingChapterRepository tcr){
          this.tcr = tcr
    }
    //业务方法
    public void add(TrainingChapterDTO tcDTO){
          tcr.add(tcDTO);
    }
}
```

被调用者 TrainingChapterRepositoryImp 类：

```
package com.g.elearning repository.;
public class TrainingChapterRepositoryImp implements TrainingChapterRepository {
    //业务方法
    public void add(TrainingChapterDTO tcDTO){
          em.add(TrainingChapterDTO tcDTO);
    }
}
```

TrainingChapter 类只有一个构造方法，因此最简单的实例化代码应该如下：

```
TrainingChapterRepository obj = new TrainingChapterRepositoryImp ();
TrainingChapter tc = new TrainingChapter (obj);
```

我们可以观察到要创建TrainingChapter 的实例却必须要照顾到TrainingChapter 类中涉及到其它类（如 TrainingChapterRepositoryImp 类）的创建，因此 Java 默认的实例化方式会带来一系列琐碎的代码编写工作，效率较低。

使用 DI 模式后，TrainingChapter 类生成实例代码变为如下：

```
TrainingChapter tc = (TrainingChapter) factory.getBean("traingCahapter");
tc.add(tcDTO);
```

两行代码中的第一行是在创建实例，第二行是调用该实例的add方法将一个DTO创建为一个新的TrainingChapter对象。采取DI模式令实例化过程就变得好像到一个“实例池”中随意捞取一个TrainingChapter对象，而完全不需要知道TrainingChapter的内部逻辑，无需照顾TrainingChapter中其它类的实例化，达到了松耦合的效果。

实现上述调用只需要对Spring框架的配置文件进行简单的配置：

```
<bean id="trainingChapter" class="com.g.elearning.model.TrainingChapter" lazy-init="false" >
    <property name="tcr" ref="trainingChapterRepository">
</bean>
<bean id="trainingChapterRepository" class="com.g.elearning.model.TrainingChapterRepository" >
    ……
</bean>
```

实际在按图2.9所示的领域模型实现的系统中，TrainingChapterRepositoryImp 类的初始化并不是一个空白的构造方法，它与TrainingChapter的构造函数一样，还要顾及EntityManager的初始化，因此上面的例程中如果不采用DI模式进行初始化，而采用传统的级联式、琐碎的实例化过程， 那么TrainingChapter 类的生成实例代码还要复杂很多。而应用了DI，不管这种传值有多少层调用都是使用相同的实例生成代码，而程序员只需在配置文件中增加或修改相应的配置，DDD框架会自动实现这其中的一系列传值调用。

综上，使用DI带来了革命性优点：

（1） 类创建成实例的过程简化。颠覆对象使用之前必须创建的基本定律，正象无需关心对象销毁一样，用户可以无需关心对象创建。

（2） 松耦合；更换各种子类方便。上例中，假设TrainingChapterRepository有另外一个实现子类TrainingChapterRepositoryAnotherImp 类，只要将上述配置中：

```
<property name="tcr" ref="trainingChapterRepository">
```

更换为：

```
<property name="tcr" ref="trainingChapterAnotherRepository">
```

同时增加trainingChapterAnotherRepository的配置项：

```
<bean id="trainingChapterAnotherRepository"
        class="com.g.elearning.model.TrainingChapterAnotherRepository" >
```

程序发布过程变成不需要发布新的主程序文件，而只需要变更配置文件的配置项。

在按图2.9所示的领域模型实现的系统中，通常所有的服务类、实体类、仓储类都需要采用DI模式进行组件注册。

2.3.3 面向切面编程（AOP）

AOP 是 Aspect Oriented Programming 的缩写，意思是面向切面编程，它是进行领取驱动设计实践的另一个不可获缺的技术环节。AOP 实际是 GoF 设计模式的延续，设计模式孜孜不倦追求的是调用者和被调用者之间的解耦，AOP 可以说也是这种目标的一种实现。应用 AOP 可以帮助 Java EE 容器分离应用系统的一些通用功能，例如事务机制、安全机制以及缓存或线程池等性能优化机制，而这些功能机制是每个应用系统几乎都需要的，因此可以从具体应用系统中分离出来，形成一个通用的框架平台，而且，这些功能机制的设计开发有一定难度，同时运行的稳定性和快速性都非常重要，必须经过长时间调试和运行经验积累而成。

AOP 强调的是分散关注（separation of concerns）的思路。将通用需求功能从不相关类之中分离出来；同时，能够使得很多类共享一个行为，一旦行为发生变化，不必修改很多类，只要修改这个行为就可以。图 2.10 展示了这种机制的基本原理。

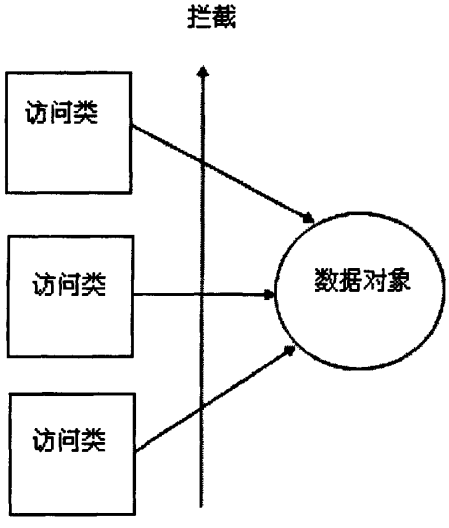

图 2.10 AOP 的原理示意图

Fig. 2.10 AOP diagram

DDD 应用系统的仓储类的缓存机制是 AOP 实现的非常好的实践环节。下面是 E-Learning 系统的数据持久框架应用和不应用 AOP 机制的代码比较的一个例子（其中使用了一些命名复杂的方法来表示繁琐的实现细节）。

未使用 AOP 的程序，在进行增删改查操作必须人工管理缓存对象从缓存池中“进”、“出”的时机：

```
public class EntityManagerWithoutAOP(){
        ……
        public void insert(UserModel usermodel) throws Exception{
            addCacheOfItem(usermodel);
            doInsertUserModelIntoDB(usermodel);
        }
        public void update(UserModel usermodel) throws Exception{
            doUpdateModelInDB (usermodel);
            clearCacheOfItem(usermodel);
        }
        public UserModel load(String id) throws Exception{
            if(loadCacheOfItem (id)==false){
                UserModel usermodel= doLoadModelFromDB(id);
                addCacheOfItem(usermodel);
                return usermodel;
             }
        }
        public void delete(UserModel usermodel) throws Exception{
            doDeleteModel(usermodel);
            clearCacheOfItem(usermodel);
        }
        ……
}
```

使用了AOP的程序，缓存机制对开发者完全透明，缓存的同步机制被AOP组件进行拦截来实现，主程序没有了琐碎的控制代码而变得十分干净明了：

```
public class EntityManagerWithAOP(){
        ……
        public void insert(UserModel usermodel) throws Exception{
            doInserUserModelIntoDB(usermodel);
        }
        public void update(UserModel usermodel) throws Exception{
            doUpdateModelInDB (usermodel);
        }
        public UserModel load(String id) throws Exception{
            return doLoadModelFromDB(id);
        }
        public void delete(UserModel usermodel) throws Exception{
            doDeleteModel(usermodel);
        }
        ……
}
```

与DI类似，AOP的实现机制依赖于DDD框架，其实现的内部逻辑非常复杂，应由DDD开发框架来提供支持，而不是由客户程序来实现的，通常它的调用是通过配置文件

进行声明即可。因为篇幅的限制，这里就不对其具体如何配置进行展开，Spring 是目前最流行和成熟的支持 DI/AOP 的开发框架，感兴趣的读者可以参考 Spring 开发手册相关章节的介绍。

Spring 中还有另外一个很好的 AOP 应用设计，那就是声明式事务，与上面的例子相似，如果应用了声明式事务，那么我们的程序中事务管理代码就可以全部过滤掉。

2.3.4 数据持久与 ORM

当我们的设计中应用了聚合后，通过聚合的不变量特性可以保证 TrainingChapter 相关的所有对象（Question、Option 等）同时创建、修改、删除。目前我们还不得不使用关系数据库作为最终的数据持久方案， 因此必须使用数据库的事务机制来作为确保这种聚合的不变性的技术手段。

DDD 的数据持久层总的可以应用三种解决方案：一种是默认的 JDBC 通用控制类。将 JDBC 的数据库操作指令封装到一个具体的工具类中，该类不包含任何业务，仓储的实现类只是持有它来实现真正的数据库操作。这种情况下，虽然系统也可以很好地实现功能，但 TrainingChapterRepositoryImp 类甚至 TestService 类中不可避免地一定会存在一些事务控制的代码或声明，造成仓储类中业务和数据持久手段的耦合，显然不是好的设计方案。下面是一个采用 JDBC 通用控制类的仓储实现类进行事务控制的程序片断，其中关于聚合的所有相关对象的实现细节都要由程序员来一一实现。

```
public class TrainingChapterRepositoryImp implements TrainingChapterRepository{
    ……
    Private EntityManager em;
    public synchronized void add(TrainingChapterDTO tcDTO)throws SQLException{
        try{
            em.beginTransation();
            em.add(tcDto.getTrainingChapter())
            List<Question> questions = tcDto.getTrainingChapter().getQuestions();
            while(questions.itreator().hasNext()){
                Question question = (Question)ir.next();
                if(question.questionType=Constants.FILLBLANK){
                    em.add(question);
                }else{
                    //如果题型是选择，则还要继续从question中遍历Option以插入数据库
                    ……
                }
            }
            em.commitTransaction();
        }catch(Exception e){
            em.rollBackTransaction();
```

```
                e.printStackTrace();
            }
        }
        ……
}
```

第二种解决方案是上一节提到的 Spring 声明式事务，这种方法能够省掉代码中有关事务的 em.beginTransation()、em.commitTransaction()、em.rollBackTransaction()等代码，但还是要管理聚合中的所有实体、值对象的创建和删除等操作。

最彻底的解决方案是应用一些成熟的 ORM 框架作为数据持久方案。ORM 是用来解决面向对象设计中对象和数据库的关系映射关系的最佳解决方案，实际上，DDD 演化到今天，ORM 已经成为事实上的 DDD 数据持久框架的标准配置，很难想象不使用 ORM 的 DDD 框架会有多么高的生产力。下面是应用 ORM 后的仓储类代码片断，程序中不仅没有了显式的事务代码，连关于模型的实现细节都全部可被省略，程序代码简化成了简单的接口调用，把一切实现细节都交由 ORM 工具来进行实现，节省了大量的代码量。例程如下：

```
public class TrainingChapterRepositoryImp implements TrainingChapterRepository{
    ……
    Private EntityManager em;
    public synchronized void add(TrainingChapterDTO tcDTO)throws SQLException{
            try{
                //ORM工具封装对聚合的细节的持久化操作
                em.add(tcDto);
            }
    ……
    }
}
```

另一个需要注意的问题是，程序还要保证多用户并发对同一聚合对象发出创建、修改、删除请求的情况下能够准确的达成用户的指令，同时保证聚合不变量的规则不被破坏。这里可以采用数据库的加锁机制进行控制，例如多数数据库引擎都会支持的乐观锁、悲观锁等实施机制。还可以在 Web 容器或 JVM 环节上进行控制，例如只需要在仓储类的一些关键方法上使用 Java 的 synchronized 关键字就可以了保证客户程序的进程同步（可参见上面的代码）。

E-Learning 系统选用的数据库产品是 MySQL。MySQL 是一个开源的小型关系数据库管理系统，被广泛地应用在 Internet 上的中小型网站中。相对于 Oracle、MS SQL 等大型数据库产品，体积小、速度快、总体拥有成本低是 MySQL 的主要优点，尤其是开源这一特点，可以令许多中小型网站降低成本。目前所有主流的 ORM 工具都支持 MySQL 数据库。

2.3.5 如何应用 Hibernate

ORM 工具对于软件的项目失败的影响是越来越大了，可以被选择使用的 ORM 工具有很多，比较常见的有 Hibernate、iBatis、JDO 等，其中 Hibernate 是截至目前设计最出色和应用最广泛的 ORM 工具，同时作为开源工具，它还是完全免费的。上节中已通过代码的对比展示了使用其实现工具部分的编码的强大功能，下面将结合 E-Learning 项目简单介绍一下 Hibernate 达成这一效果的一些实现细节。

E-Learning 系统中在 TrainingChapter 聚合中包含 TrainningChapter、Question、Option 三种主要的业务对象，他们在关系上从大到小、依次包含的。因此在类设计上，必须体现这种聚合关系。我们观察图 2.9，可以发现在类设计上，这种聚合关系是对象直接持有下一级对象的实例，而不是聚合下一级对象的 ID，应用 Hibernate，对象间的关联务必直接持有对象而不是持有持有数据库主键，这是设计上的一个关键点。下面是实体类 TrainingChapter 的代码片断：

```
public class TrainingChapter {
    ......
    private String id;
    private Integer chapterType;
    private Collection<Question> questions = new ArrayList<Question>(0);
    ......
    public String getId() {
        return this.id;
    }

    public void setId(String id) {
        this.id = id;
    }
    public int getChapterType() {
        return this.chapterType;
    }

    public void setChapterType (Integer chapterType) {
        this.chapterType = chapterType;
    }
    public Collection<Question> getQuestions() {
        return this.questions;
    }

    public void setQuestions(Collection<Question> questions) {
        this.questions = questions;
    }
```

```
    ……
}
```

有了TrainingChapter类，还要根据其内容编写一个XML格式的配置文件来描述实体属性和数据库的表和字段的关联，之所以被叫做OR Mapping工具，主要就体现在这个配置文件上。下面的代码是实体类TrainingChapter的配置文件片断。

TrainingChapter.hbm.xml:

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class name="com.g.elearning.TrainingChapter" table="tblTrainingChapter">
      <id name="id" type="java.lang.String">
        <generator class="uuid.hex"></generator>
      </id>
      <property name="chapterType" type="java.lang.Integer"></property>
      ……
      <bag name="questions" cascade="all">
         <key column="chapterId"></key>
         <one-to-many class=" com.g.elearning.Question"/>
      </bag>
  </class>
</hibernate-mapping>
```

与之类似，Question和Option也要进行ORM配置。这个配置文件表明了com.g.elearning.TrainingChapter类对应的数据库表为tblTrainingChapter，同时TrainingChapter的每个属性在数据库中的相应字段名和字段属性都要被进行了声明。最后也是最关键的一部分，它声明了其内部的集合属性questions会与另一个类com.g.elearning.Question进行关联，在与之关联时，TrainingChapter类的主键“id”会与Question类的配置文件声明的“chapterId”属性对应的数据库字段（通过TrainingChapter的配置文件还不知道具体字段名是什么，这在Question类的配置文件会有声明）形成外键关联。

当所有的实体和数据库都被正确的建立，并通过Mapping配置文件建立了他们的关联之后，Hibernate框架就可以掌握对实体对象进行持久化的一切必要的实现细节。因此我们只要把用户界面层的领域对象按照对象之间的关系进行组织封装，并通过DTO传递给领域层的相关服务，领域模型就可以被自动地持久化到数据库中，事务控制和聚合的不变量全部由Hibernate内部逻辑来实现而不是由程序员来控制，大大提高了系统的可靠性，同时也节约了大量的编码工作量。

Hibernate 另一个伟大之处在于我们甚至并不需要按照 ORM 配置到数据库中亲自去创建表，在产品部署到 Java EE 容器时，它会自动根据 ORM 配置去创建所有的数据表，由此可见，DDD 极力主张的降低数据库设计在系统设计中的作用而以领域对象为主的设计思路并不是空谈。

Hibernate 支持市场上所有主流的数据库，它是完全独立的产品框架。考虑到应减少 DDD 框架内对象与具体产品的依赖性，并且让系统可以在必要时自由地更换数据库产品，我们在进行 DDD 设计时应少依赖某个数据库产品特性，尽量使用符合 SQL-92 标准的数据库设置来进行 ORM 的配置。另外一个要注意的问题是 E-Learning 项目所采用的 MySQL 数据库的事务引擎是以插件的形式进行配置的，但并不是所有的引擎都支持事务，很容易会使用默认的 MyISAM 引擎造成事务控制无效，开发者应引起重视。

DDD 框架除了应用 Hibernate 的自动化持久之外还可以使用 Hibernate API 实现增删改查，但绝大多数情况下这会导致所有的 AOP 和 ORM 给 DDD 带来的核心价值的丧失，除非必要，不要采用。

2.3.6 缓存

缓存是为了降低应用程序对物理数据源的访问频次而出现的一种提高系统性能的机制。它的原理是，缓存内的数据是对物理数据源中的数据的复制，应用程序在运行时先从缓存读写数据，在特定的时刻或特定的事件会同步缓存和物理数据源的数据。缓存是 GOF 设计模式中代理模式的一种实现，并不是 DDD 的独创，但 DDD 中的仓储模型在实现上与其概念高度吻合，因此 DDD 可以说是在理论上对缓存的设计和实现进行了完善和升华。

缓存的最佳介质是内存，因为内存的读写速度最快，当缓存中存放的数据量非常大时，也可以用硬盘作为缓存介质。缓存的实现不光要考虑存储的介质，还要考虑到管理缓存的并发访问和缓存数据的生命周期等。

从程序架构上讲，缓存介于应用程序和物理数据源之间，有多种架构方式，应用层、领域层和基础设施层（数据持久层）都可以实现缓存功能，采用不同的架构方式则缓存的技术方案也有不同，从原理上讲，缓存在程序分层上越靠前则对系统性能提高越快，开发成本也越高，因此在数据持久层实现的数据库级缓存是较为普遍采用的一种的数据缓存方式。

Hibernate 支持两极缓存，第一级是 Session 级的缓存，通常对应一个数据库事务或者一个应用事务，只针对当前用户的当前连接有效；Hibernate 的二级缓存是全局性、应用级的缓存，由 SessionFactory 负责管理，可插拔，其生命周期对应应用程序的整

个生命周期，因此与DDD中仓储所指的缓存概念是最相符的。但对这种二级缓存的控制必须兼顾到并发，设计和实现都较为复杂一些。

Hibenate中的ehcache是实现缓存的一个很好的组件，后文要介绍的JdonFramework框架甚至用它脱离Hibernate而独立地在应用层实现了一个完整的缓存组件，这种应用级的缓存在系统架构中的应用会带来更高的性能和便利。

2.3.7 用户界面层

UI层的设计也是DDD的一个重要的方面。在进行用户界面层设计时，应特别注意不要把任何业务逻辑混合到用户界面层，造成用户界面层直接实现功能而打破原有的领域模型设计。对于领域驱动设计来说，任何不经过领域模型实现的功能都是不可以接受的，都可能给系统带来不可挽回的负面影响。

在前面图1.1所示的模型驱动设计模式的导航图中明确指出智能UI（Smart UI）与领域驱动设计是相互抵触的设计。从实现的角度来讲，这种说法的一个主要根据在于仓储。我们知道在DDD中仓储是对象和数据库之间的桥梁，而智能UI提倡的是用户界面直接对应业务，换句话说用户在界面上的业务操作是直接操作在数据库的数据上，这就会造成突破仓储直接操作数据库的数据，但是这样做的问题是，DDD框架中第一次请求数据库数据一定会先在仓储中建立一个缓存，然后其它领域对象访问同一对象时是通过该缓存得到数据，而不是到数据库去得到数据，绕过仓储的数据变更都不会被DDD应用系统接受到，甚至可能造成其它严重后果。

因此，我们在设计上应该只让用户界面层承担DTO封装功能，不要包含任何用于控制的Java代码，让该层成为用户的输入与业务层的一个简单桥梁就可以了。

DTO的实现方式可以简单把图2.9中所有实体类中的业务方法(add、update、delete和load)去掉就可以了。DTO还可以由其它简单DTO组成，例如TestPaperDTO是TrainingChapterDTO和studentDTO组成的，参见下面的例程：

```
public class TrainingChapterDTO  {
   private Collection<TrainingChapter> trainingChapters;
   private StudentDTO studentDTO;

   public Collection<TrainingChapter> getTrainingChapters() {
      return this.trainingChapters;
   }
   public void setTrainingChapters(Collection<TrainingChapter>
trainingChapters) {
      this. trainingChapters = trainingChapters;
   }
   public StudentDTO getStudentDTO() {
```

```
        return this.studentDTO;
    }
    public void setStudentDTO(StudentDTO studentDTO) {
        this. studentDTO = studentDTO;
    }
}
```

Struts 是 Apache 组织的一个开源项目，Struts 作为一个比较好的 MVC 框架可以对表现层的设计和实现提供良好的底层支持，它采用的主要技术是 Servlet，JSP 和自定义标签库。MVC 模式通常被归为一种设计模式（Design Pattern），它的引入也正是为了解决传统 Web 开发中表现层和逻辑层耦合的弊病。图 2.11 是 Struts 的组件模型图。

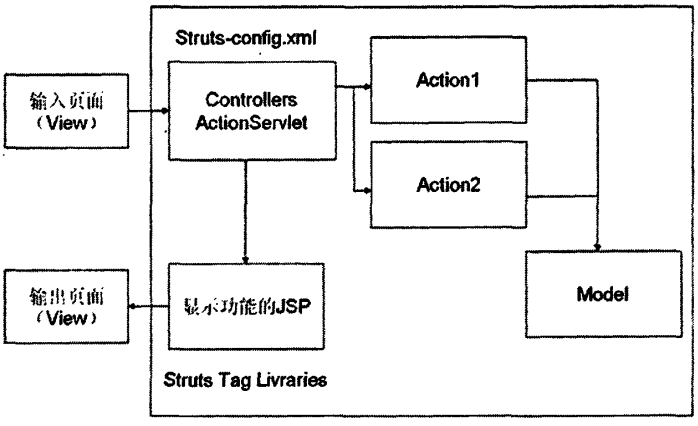

图 2.11 Struts 的组件模型图

Fig. 2.11 Struts process map

MVC 框架有三个主要组件组成：

（1） 模型（Model）：现实世界对象的抽象，可以理解为 DDD 中的实体、值对象和领域服务。

（2） 视图（View）：负责显示功能的组件，可以理解为前文所说的 DTO。

（3） 控制器（Controller）：控制器是视图与模型的接口，是 MVC 模式的实现核心。可以理解为 DDD 中的应用层服务和部分领域层服务的集合。

2.4 小结

本章的内容主要是从系统分析人员和编码人员的角度对 E-Learning 系统构建过程中的领域建模和编码实现阶段的工作做了总的介绍。在这个过程中，不仅展现了 E-Learning 项目的系统架构和关键模块的构建过程，还对领域驱动设计的开发框架所必

须支持的POJO、DI、AOP和ORM等Java EE技术特性和应用方法进行了总结，在这样的基础上，运用Spring+Struts+Hibernate（SSH）对前文所述的E-Learning项目需求进行开发将不会在再有什么大的风险。同时，在这个基础上，SSH也可以作为轻量级框架的开发平台应用于类似的软件项目的开发。

通过领域驱动思想建模并进行开发程序会有以下几个显著的特点：

（1） 代码松耦合。程序间的调用主要基于接口，并且采用多层配置取代多层编码，主要的实现类都是可配置、可替换的，主程序的代码优雅简洁，相对传统开发模式，程序的可扩展性和可维护性有本质提高。

（2） 高数据访问性能。这是仓储以及实现它的缓存和连接池技术带给我们的直接优点。

（3） 开发框架无关性。DDD要求模型应使用POJO类实现，虽然仓储、服务的实现类必然要与Hibernate等底层框架耦合，但当需要换用其它开发框架时，用另一个类来实现功能，然后配置给DDD框架就可以了。

（4） 便于自动化测试。由于大部分的组件都是独立和可配置的，并且与开发框架无关，这样我们就可以方便的进行自动化测试。

需要注意的是，并不是采用了SSH进行开发就代表运用DDD进行软件开发了，这个道理在于SSH本身并不是DDD开发框架，只有当我们基于DDD的思想进行软件建模，然后选用SSH进行程序开发时，那么SSH才是帮助我们进行DDD实践的良好开发框架。

此外，为了更好地理解DDD思想，开发者和设计者应熟悉和掌握GOF设计模式的要点。

3 JdonFramework 的领域驱动设计应用

本章将主要介绍一个国产的 DDD 开发框架——JdonFramework，以及比较使用该框架同样开发 E-Learning 系统会与上一章介绍的 SSH 平台开发有何异同。

3.1 JdonFramework 简介

JdonFramework 是由国人独立开发的一个 DDD 开发框架，同时也是一个免费开源的软件开发平台，它最早在 2004 年由 Jdon.com 社区的创始人彭晨阳进行倡导、设计并推出了 1.0 版本，刚推出时主要目的是针对软件开发的快速性和灵活性问题，随着 5 年的发展，JdonFramework 在不断增加入新的设计概念和功能，已逐渐发展成为一个整合 DDD 设计思想和开发模式于一体的、提供快速开发 Java EE 途径的工具框架，截至本文成稿前的最新版本是 2009 年 2 月 6 日推出的 5.6 版。

应用 JdonFramework 常见的开发框架是 Struts+JdonFramework+Hibernate（SJH），由此可见，JdonFramework 是与 Spring 进行竞争的产品，它有自己的 DI 和 AOP 组件，其中采用 PicoContainer 实现的 DI 组件已经十分成熟，比 Spring 的配置方式更简便，而 AOP 部分相对设计的拦截粒度较粗，还不支持声明式事务，主要为了实现该系统的缓存功能，缓存功能的设计和实现是整个 JdonFramework 框架中最有价值的部分。

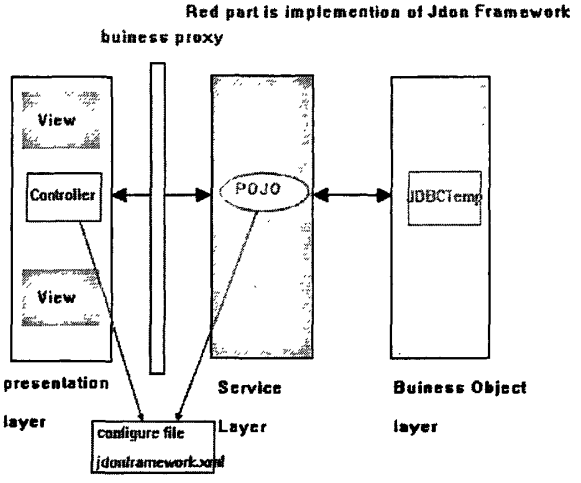

图 3.1 JdonFramework 在 Java EE 架构中的作用

Fig. 3.1 The use of JdonFramework in Java EE architecture

图 3.1 展示了 JdonFramework 在 Java EE 架构中的作用和位置。

3.2 JdonFramework 开发模式

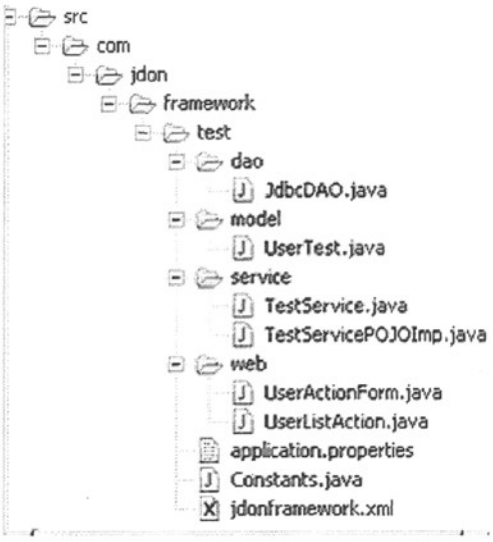

图 3.2 JdonFramework 开发目录结构

Fig. 3.2 JdonFramework development file directory

图 3.2 是应用 JdonFramework 开发的一个简单案例的源代码目录结构，该案例主要由 Model、Service、DAO、UI 四个层次的文件划分和一些配置文件、工具类组成。其中，model 文件夹下的 UserTest.java 代表的是模型，所有的 DDD 实体和值对象应在此建立模型。service 是服务文件夹，所有的 DDD 实体和聚合根应在此建立服务类，建立时接口和实现类应分离。Dao 文件夹代表的是数据访问对象(Data Access Object，简称 DAO)，在 JdonFramwork 中，DAO 对象从概念上可与 DDD 的仓储进行对应，DAO 夹里的程序文件同样也应遵循接口和实现类分离的原则。Web 文件夹放置与 UI 层相关的一些 DTO 和控制对象。各个层次的对象通过系统配置文件（jdonframework.xm）进行配置，采用 DI 方式进行调用，代码保持松耦合。

JdonFramework 中整合了对于模型的增删改查基础上的高效、快速开发经验，它把数据的增删改查流程被抽象为：ViewAction → 表单 Jsp 页面 → SaveAction → 结果 Jsp 页面，如图 3.3。

在如图 3.3 所示的流程图中，对于同一个实体模型，用户通过浏览器网址（或 Html 链接）调用用于新增或编辑的页面，当用户按“确定”按钮保存输入的数据后，框架会激活 Struts 框架的 ModelSaveAction，ModelSaveAction，它们通过 jdonframework.xml

的配置调用相应的 Service 类，然后程序会根据原始网址或 Html 链接调用的参数 action 不一样而调用不同的 Service 接口中相应的方法实现增删改查的功能。

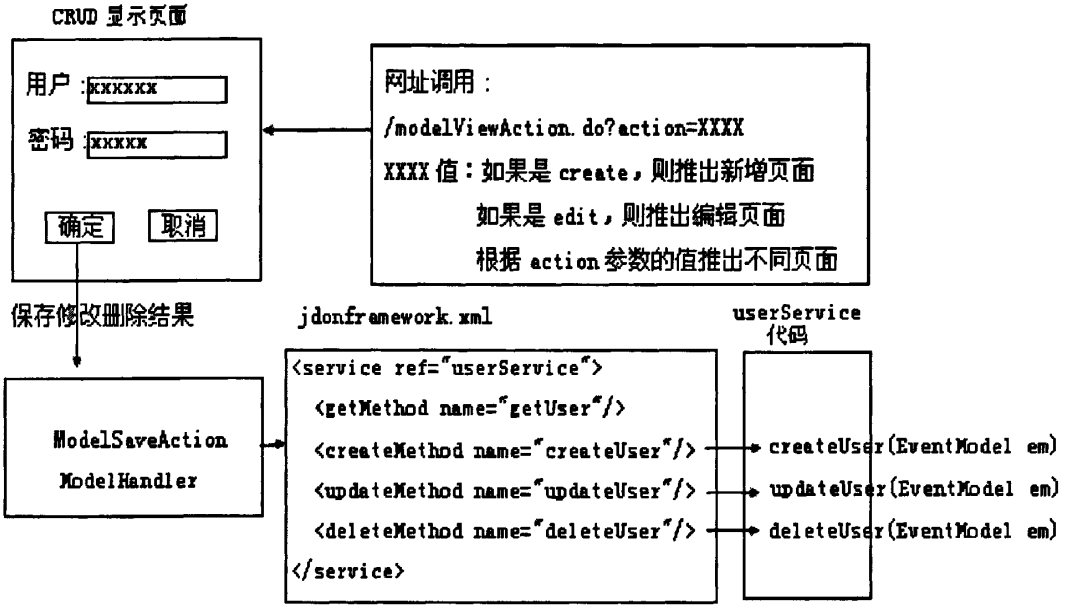

图 3.3 JdonFramework 的开发模式流程图

Fig. 3.3 JdonFramework development model flowchart

对于程序员来说，由于 JdonFramwork 高效地集成了对于 Hibernate、Struts 等主流开发工具的支持，自己的程序只要按照框架要求的方式创建实体、简单声明接口、配置类关联，就能完成大约 80%的基础系统功能，剩下的 20%工作量主要在于大批量数据的查询功能上。

虽然 JdonFramework 推荐使用 SJH 方式进行开发，并且 Struts 和 Hibernate 对于运用 JdonFramework 开发的促进作用非常巨大，但 Struts 和 Hibernate 仍然是该开发模式中的可选项，并非必备项，这就可以保证应用该框架进行项目开发的灵活性。

3.3 JdonFramework 领域模型特点

3.3.1 贫血式领域模型

与 DDD 一书中的建议不同，JdonFramework 中采用的是“贫血”式的领域模型。所谓“贫血”式的领域模型是指实体对象中只有 getter、setter 弱行为方法，而将实体

对象本身的 create、update、delete 方法放入到服务类中。图 3.4 是采用 JdonFramework 开发模式的 E-Learning 系统的领域模型图。

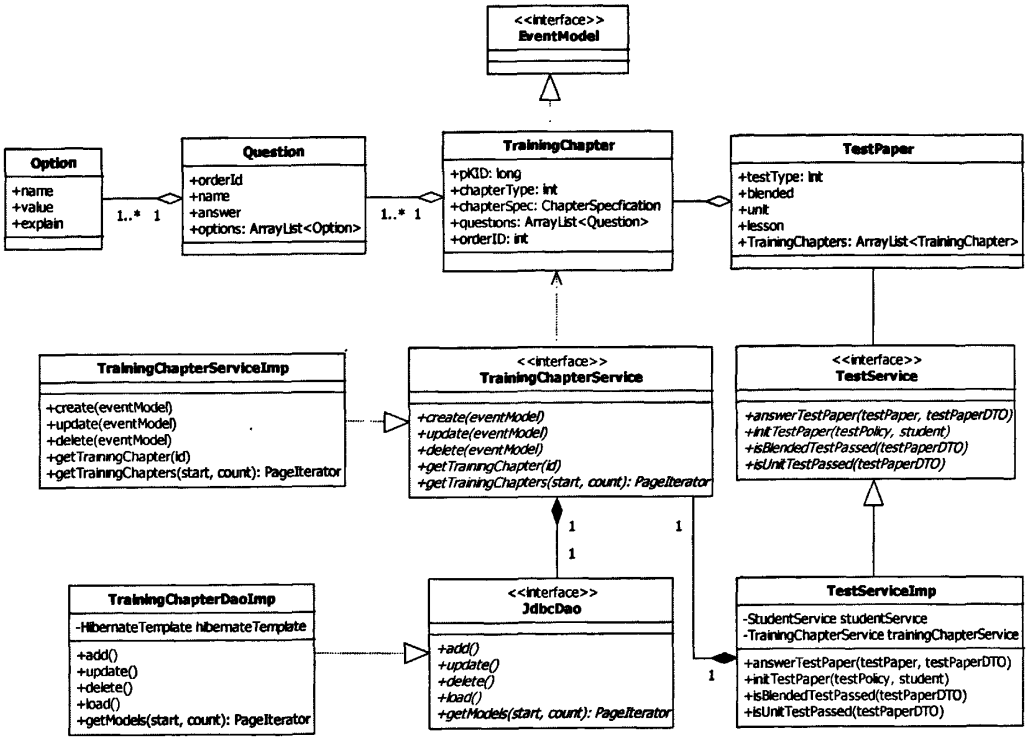

图 3.4　JdonFramewok 开发模式的领域模型图

Fig. 3.4　Domain-Model of E-learning system with JdonFramework

与 2.3.1 小节所介绍的“充血”领域模型相比，JdonFramework 的“贫血”领域模型开发模式的显著不同点如下：

（1） 实体类 TrainingChapter 中只有弱行为方法，并且必须实现 EventModel 接口（或者继承一个实现该接口的基类，该类已集成在框架中）。

（2） 所有实体必须有一个实体服务类（如图中的 TrainingChapterService 及其实现类），但该服务类区别于领域模型中的“服务”概念，可理解为只是与实体类配套的控制类，提供对实体对象的增删改查的方法。

（3） 实体服务类持有一个 JdbcDao 对象，该对象代表 JDBC 模版类，封装 JDBC 与数据库的交互细节，采用 DI 方式注入到服务类的实现类中。JdbaDao 类和

TrainingChapterDaoImp 类与图 2.9 中的 TrainingChapterRepository 和 TrainingChapterRepositoryImp 无论从概念和功能上都是完全一致的。

（4） 系统的领域服务类仍然是 TestService 和其实现类 TestServiceImp，但是 TestServiceImp 中持有的不再是实体类 TrainingChapter 本身，而是实体服务类 TrainingChapterService，该类采用 DI 模式注入到 TestServiceImp 中。

3.3.2 JdonFramework 模型规范的优缺点

以下是对于 JdonFramework 的例程进行研究后，再结合前文的 SSH 开发模式下的 E-Learning 产品进行对比所总结的 JdonFramwork 模型规范的优缺点。

（1） 以贫血模型方式建模最大的好处在于解决了“充血” 模型中服务类直接持有实体可能带来的问题。

在 DDD 一书中对于“服务”的约定是应采用显式声明，无状态，保证客户端可无障碍的调用。JdonFramework 类设计中，无论是 TestService 还是 TrainingChapterService 类都只有行为没有状态，因此 JdonFramework 的设计模式遵循了这一原则。而图 2.9 的充血模型中，TestService 直接持有 TrainingChapter，实体类是有状态的，这样 Service 就可以是有状态的，因此图 2.9 中的设计是违反了这一约定的。但实际该设计的本意是 TestService 服务中持有的实体类只是直接作为实体操作的外观类使用，它在编码过程中将被人为规定不会被赋值，因此正常情况下系统不会出现因服务持有状态所带来的问题。但尽管如此，这种设计方式在直观感觉上会给人带来困扰，而且实际编码过程中也有被使用者无意间打破该规约的风险。

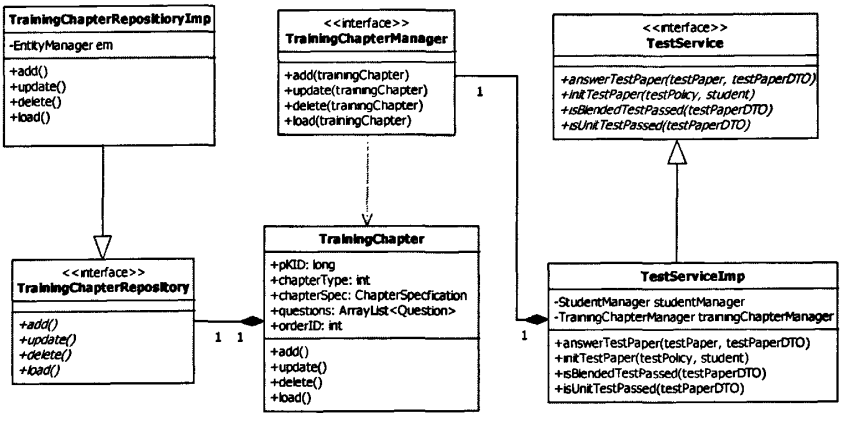

图 3.5 对充血模型的改进

Fig. 3.5 Improve for rich domain object

图 3.5 的设计可对上述问题进行一些改进，那就是让 TestService 持有一个 TrainingChapter 的代理类 TrainingChapterManager 就可以规避上述问题。但本质上来说，该设计只要让 TrainingChapterManager 直接持有 TrainingChapterRepository，那么 TrainingChapter 的增删改查就会被架空，整个设计与 JdonFramework 的贫血模型的架构就变得没什么不同。而且这种情况下，增加的代理类功能上只负责函数传值调用，功能单一，属于典型的多余抽象和间接层次，会导致系统复杂性，因此两相比较，在这里采用的贫血模型的架构更有助于提升系统的设计。

（2） 实体直接持有增删改查方法还带来另一个问题，同时也是充血模型和贫血模型两种设计方式之间最大的理论性争议，那就是一个对象能够创建自身是否合理？请看下面的代码比较：

充血模型方式，由模型创建自身：

```
TrainingChapterDTO tcDTO = request.getParameter("tcDTO");
TrainingChapter tc = (TrainingChapter) WebAppUtil.getService("tc");
tc.add(tcDTO);
```

贫血模型方式，由模型的外观类创建模型：

```
TrainingChapter tc = request.getParameter("tc");
TrainingChapterService tcService =
          (TrainingChapterService) WebAppUtil.getService("tcService");
tcService.create(tc);
```

通常认为贫血模型的由外观类进行控制的创建方式更为符合客观事物的规律，是目前更为推荐的建模方式。

（3） 由于 JdonFramework 的实体类为贫血模型，一定程度上可以直接作为 DTO 使用，这会为节省单存简单复制实体类来创建 DTO 这种简单工作的编码量。

（4） 框架本身对于增删改查功能的整合十分严谨，切实加快项目开发速度。

（5）JdonFramework 的贫血模型设计要求任何实体类都必须有一个配套的服务类，这从概念和功能上都容易与领域模型自身的服务类混淆，如果控制不好，很容易在设计上把属于领域服务类的职责错误地分配给实体服务类，从而给设计带来服务层“臃肿”的问题。

（6） JdonFramework 的实体类必须实现 EventModel 接口，这违背了 DDD 推荐的领域对象应采用 POJO 设计的原则，尽管这种接口设计以及实体服务类中方法的命名规范都是为了对于框架的增删改查机制的提供支持，但会造成模型与开发框架的耦合，通常会对项目的产品移植造成不便。

基于上述对比与分析，究竟两种建模规范孰优孰劣？我认为任何情况下都不宜做这种武断的判断。DDD 是思想，是“方法学”的范畴而不是硬性规范，前文介绍的两种不

同的模型设计在总的思想上都遵循了领域模型划分的大方向，因此都是可行的系统实施方案。我们进行项目实践时，应结合自己的项目去采取最佳的建模方式，实际上选择贫血模型和充血模型其实并不重要，因为技术在不断发展，这些框架的开发者们也在不断融会贯通，理论上的壁垒是可以被打破的，只要系统设计和实现环节遵循其领域模型和逻辑层次的划分，实体、仓储、服务类抽象正确、各司其职，然后通过 UML 设计工具辅助、使用类似 Spring、JdonFramework 这样强大的开发工具支持，都可以开发出架构合理、功能强大的应用软件。

3.4 JdonFramework 的开发步骤

下面以 E-Learning 系统的核心模型 TrainingChapter 为例详细讲解一下如何运用 JdonFramework 进行增删改查（CRUD）和批量分页查询的开发流程。实体的 CRUD 和批量分页查询编码是每个应用系统中都要对应的基础工作。通过 JdonFramework 框架可快速完成这些基本工作，从而可以让项目组把精力集中在体现系统价值的特殊业务功能上。

3.4.1 JdonFramework CRUD 流程的代码编写

根据 3.2 节介绍的 JdonFramework 设计规范和 Hibernate 配置文件的需要，TrainingChaper 的增删改查方法的全部实现需要完成以下几个文件的代码编写与配置：

model/TrainingChapter.java
model/Question.java
model/Option.java
model/TrainingChapter.hbm.xml
model/Question.hbm.xml
model/Option.hbm.xml
service/TraingChapterService.java
service/TraingChapterServiceImp.java
dao/TrainingChapterDaoImp
web/TrainingChapterForm.java
front/chapter.jsp
front/chapterList.jsp

下面结合 JdonFramwork 的开发步骤对它们的开发做一下简要的介绍。使用 JdonFramwork 进行开发，编码通常分三个步骤：

（1） 建立实体类。

```
public class TrainingChapter extends Model {……};
```

```
public class Question{……};
public class Option{……};
```

作为聚合根的实体类 TrainingChapter 必须继承 JdonFramework 框架的 com.jdon.controller.model.Model 类或实现 com.jdon.controller.model.ModelIF 接口(聚合内部对象则未必)。所有模型类必须有一个能够标识其对象唯一性的主键，我们可以理解为就是数据表的主键。

TrainingChapter.hbm.xml、Question.hbm.xml、Option.hbm.xml 是 Hibernate 的映射文件，我们在前面对于它的编写已经有了详细地介绍（参见 2.3.5），这里必须要强调的是，由于目前绝大多数的软件项目仍旧离不开数据库的支持，因此建模阶段还是要对模型、数据库两者进行通盘考量，但既然基于 DDD 思想进行开发，一定要以模型作为总的出发点，让数据库退居其次。

（2） 建立服务类和仓储类。

首先建立服务类的接口，例程如下：

```
public interface TrainingChapterService {
    public void createTrainingChapter (EventModel em);
    public void updateTrainingChapter (EventModel em);
    public void deleteTrainingChapter (EventModel em);
    public TrainingChapter getTrainingChapter (String chapterId);
    ……
}
```

TrainingChapterService 的实现子类调用仓储类相应方法来实现具体功能。例如：

```
public class TrainingChapterServiceImp implements TrainingChapterService {
    ……
    public void updateTrainingChapter(EventModel em){
        TrainingChapter chapter = (TrainingChapter)em.getModel();
        try {
            TrainingChapter chapter = getTrainingChapter(chapter.getId());
            if (chapter != null)
                jdbcDao.update(chapter);
        } catch (Exception e) {
            logger.error(e);
        }
    }
    ……
}
```

基于 JdonFramework 的编码规范，仓储类的实现 TrainingChapterDaoImp 只需继承 JdonFramework 的 DaoCRUDTemplate 类就可自动对应上面的服务类的 CRUD 方法，因此仓储类 TrainingChapterServiceImp 中并不需要自己编写任何 CRUD 代码。

（3）建立 Model 的表现层边界模型。例程如下：

```
public class TrainingChapterForm extends ModelForm {
    private String id;
    private Integer orderId;
    private String sound;
    private String image;
    private String desc;
    private Collection<Question> questions;
    private ChapterProperty chapterProperty;

    public String getID() { return ID; }
    public void setID(String ID) { this.id = id; }
    ……
}
```

JdonFramework 的表现层模型 TrainingChapterForm 的代码基本上是从领域层模型 TrainingChapter 类中拷贝过来的，主要是为了保持 TrainingChapterForm 和 TrainingChapter 的字段一致。由于 DTO 继承了框架中的 ModelForm，就可以通过 JdonFramwork 框架的 Struts 组件实现自动封装/拆封 DTO，用户通常也就不需要自己编写任何的封装和拆封 DTO 的代码了。

完成表现层模型后，再将他们绑定到美工组设计的网页 chapter.jsp/chapterList 上。

需要注意的是，JdonFramework 中的表现层模型并不是前面介绍充血模型时的 DTO 的概念，而是纯 Struts MVC 模型概念中的表现层视图模型（ActionForm），它不应被应用到领域层的内部传值（会导致应用层和表现层的耦合），因为 JdonFramework 中的模型类本身就可以起到充血模型的 DTO 的功用。

TrainingChapter 对象的有关 CRUD 实现的代码工作到此结束。E-Learning 系统中的所有实体模型都可按照上述步骤进行实现。

3.4.2 JdonFramework CRUD 流程的配置

编码阶段的成功必须结合 DI 的配置才能真正生效。JdonFramework 中有两个主要的配置文件，这两个配置文件分别是用于将前面三步编写的类建立关系的 jdonframework.xml 和用于配置界面流程的 struts-config.xml。有了这两个配置文件，框架的依赖注入功能才能生效。

（1） Jdon 框架配置文件

jdonframework.xml 是 JdonFramework 框架的核心配置文件，主要用于将开发过程中编写的三个类：模型 TrainingChapter、服务 TrainingChapterService 和界面模型 TrainingChapterForm 建立起联系。jdonframework.xml 配置内容如下：

```
<models>
    <!-- 配置模型的类是 TrainingChapter，其主键是 chapterId -->
    <model key="chapterId" class =" com.g.elearning.model.TrainingChapter">
        <!-- 下行是配置界面模型 TrainingChapterForm -->
        <actionForm name="chapterForm"/>
        <handler>
            <!-- 以下配置 TrainingChapterService -->
            <service ref="chapterService">
                <getMethod name="getTrainingChapter" />
                <createMethod name="createTrainingChapter" />
                <updateMethod name="updateTrainingChapter" />
                <deleteMethod name="deleteTrainingChapter" />
            </service>
        </handler>
    </model>
</models>
<services>
    <!-- 以下配置 TrainingChapterService -->
  <pojoService name="chapterService" class=" com.g.elearning..service. TrainingChapterServiceImp"/>
  <component name="hibernateDao" class="com.jdon.persistence.hibernate.HibernateCRUDTemplate"/>
</services>
```

下面简要介绍一下三个部分的配置的含义：

① 模型 TrainingChapter 的配置

TrainingChapter 模型的配置段中表明了该模型主键是 chapterId，这个 chapterId 必须是 TrainingChapter 类的属性和唯一标识。

② 界面模型 TrainingChapterForm 的配置

这里并没有写界面模型完整类名 com. g. elearning. web. TrainingChapterForm，这是因为它已在稍后要介绍的 struts-config. xml 中进行了 ActionForm 定义，配置段如下：

```
<struts-config>
    ……
    <form-beans>
      <form-bean name=" chapterForm " type=" com.g.elearning.web.TrainingChapterForm" />
        ……
```

```
</form-beans>
……
</struts-config>
```

③ 模型服务 TrainingChapterService 配置

在 jdonframework.xml 中，我们首先声明 TrainingChapterService 的完整实现类是 com.g.elearning.service.TrainingChapterServiceImp，并且命名为 chapterService，然后将前面编写的 chapterService 的 CRUD 方法名注册给 Jdon 框架。getMethod，createMethod，updateMethod，deleteMethod 正是服务类接口 com.g.elearning.service.TrainingChapterService 的四个 CRUD 方法。

请注意在上面的 pojoService 中配置中不用像 Spring 那样去设定 chapterService 和 HibernateDao 两个配置项的依赖关系，JdonFramework 的 PicoContainer 组件实现了自动匹配，这种配置方式会为程序调试节省大量的人力。

（2） 界面流程配置

界面流程是指配置 CRUD 界面流程，Jdon 框架 CRUD 流程主要分两个部分：一个是供用户新增修改删除的前台界面，我们可称它为 CRUD 内容页，在命名上一般会采用 ObjectAction 的命名方式；另一个是对于用户提交的的数据进行真正数据操作的 CRUD 控制类，我们可称它为 ObjectSaveAction。所谓 Action 通常是对 Struts 框架 MVC 模式中的控制器的一种命名规范，通常被配置的 Action 对象在前台浏览器中加上".do"的文件后缀和相应参数就可以进行激活 Struts 的 forward 流程。

这部分配置主要是配置 struts-config.xml，它是 Struts 框架的核心配置文件：

① 配置 CRUD 内容页

```
<action path="/chapterAction" scope="request" validate="false" name="chapterForm"
type="com.jdon.strutsutil.ModelViewAction" >
    <forward name="create" path="/chapter.jsp" />
    <forward name="edit" path="/ chapter.jsp" />
</action>
```

这段代码中配置了一个名为 chapterAction 的控制器，该控制器的实现类 com.jdon.strutsutil.ModelViewAction 是 JdonFramework 的 框架类。有了这个配置，只要客户端浏览器调用 http://localhost/chapterAction.do，系统将激活上面配置中 forward 的 name="create"流程， 用户会得到一个空白表单的页面 chapter.jsp ，在其中可以对单条的 TrainingChapter 数据进行任意编辑和提交数据；另一种情境下， 如果客户端访问的是一个 chapterListAction.do 的数据列表，那么当浏览器点击其中一行数据进行编辑时就会进入一个类似 http://localhost/chapterAction.do?action=edit&chapterId=18 的连接，此

时就激活上面的配置中 forward name="edit"流程，系统会将一个填满 TrainingChapter 对象的所有数据的表单页面返回给前台，用户可以在其中修改内容，然后进行数据提交。

② 配置 CRUD 控制类

```
<action path="/chapterSaveAction" scope="request" validate="true" input="/chapter.jsp"
name="chapterForm" type="com.jdon.strutsutil.ModelSaveAction">
    <forward name="success" path="/result.jsp" />
    <forward name="failure" path="/result.jsp" />
</action>
```

chapterSaveAction 控制器的实现类 com.jdon.strutsutil.ModelSaveAction 也是 JdonFramework 的框架类。它一般用于表单文件的提交过程中，例如上面提到的在 chapter.jsp 中，会有一个 Form 控件，其 action 值就对应这里配置的 path 值：/chapterSaveAction.do：

```
<html:form action="/chapterSaveAction.do" method="POST" >
    <html:hidden property="action"/>
    ChapterId：<html:text property="chapterId"/>
    <br>Name：<html:text property="name"/>
    ……
    <br><html:submit property="submit" value="Submit"/>
</html:form>
```

用户按”Submit”按钮提交数据时，就是将画面中的数据封装为 DTO，然后传给 chapterSaveAction，再由它转给领域层进行处理的一个过程。

至此，领域对象 TrainingChapter 的 CRUD 功能开发完毕。

3.4.3 JdonFramework 批量查询的实现

批量查询是指取得大批量实体的列表方法，JdonFramework 中的批量查询方法也是由代码+配置的方式进行实现的，与 CRUD 功能的开发类似。整个模块的开发流程十分简单，用少量代码就可以完成这个重要的功能，而且还支持自动分页。其具体实现方式可以参考 JdonFramework 的开发手册。

3.5 小结

JdonFramework 是一款以 DDD 思想为指导，严格坚持分层架构下的快速开发，兼顾快速性和灵活性综合平衡的开发框架，它为系统开发带来了简洁的解决之道，不只是纯粹简化，而是基于高质量的简化。对于小项目，可以开发出高质量可扩展的好的系统；

对于大项目，可以有效加快开发速度。以下是应用 JdonFramework 进行项目开发的几个显著优点：

（1） 灵活性（Extendable）

JdonFramework 宣称所有组件包括 JdonFramework 自身组件都是可替换的,特别是采取 PicoContainer 作为 DI 容器，类依赖关系是自动配对，无需配置指定的，这就区别于 Spring 等手工配置指定的 DI 组件（太复杂的 XML 配置会增加开发的复杂性），大幅提高开发效率。

（2） 易用性（High-Availability）

JdonFramework 着重的是快速性和易用性；在坚持多层架构的前提下，提供了数据增删改查（CRUD）和批量查询这两个最常见的基础功能的模板化开发，以约定配置替换了琐碎的编码工作，可以实现中小型项目的快速开发。

（3） 良好的性能（Performance）

JdonFramework 通过缓存优化了应用系统性能，提高了大数据量的查询性能。运用 AOP 技术，对模型对象通过缓存拦截器提升性能；对 POJO 服务使用对象池拦截器等；

（4）可伸缩性（Scalability）

JdonFramework 还提供小系统向大系统平滑过渡的能力。可以并行支持两种服务架构(Web 架构和 EJB 架构)在一个系统内运行;通过配置可以很方便地在纯 Web 系统和 EJB 系统之间切换；支持远程胖客户端访问，用 C/S 架构实现系统。

对于 E-Learning 这种规模的项目来说，使用 JdonFramework 进行开发也是一个很好的选择，以 JdonFramework 来进行开发总的来说开发流程更为简单高效，唯一的缺点也是 JdonFramework 框架的最大痼疾，那就是软件开发方式上的强迫性约束过多，当项目范围逐渐增大，软件的业务逻辑越来越复杂时，很可能就会有无法突破的技术瓶颈，给项目带来风险。

4 E-Learning 系统的系统测试

好的测试是一个项目成功的必要因素。在软件工程中，建模是软件项目成败的根本点，编码使我们获得功能和性能符合要求的软件产品，而测试是保证软件产品最终功能和性能达到预期的关键步骤。随着软件工程领域的发展，软件测试的研究和实践也得到了非常迅速的发展，出现了很多测试理论、测试方法和测试工具。

软件项目的测试大体可分为功能测试和性能测试两大类。功能测试是测试工作中的重点，它是指根据功能测试用例，逐项测试系统功能，检查产品是否达到用户要求的功能；性能测试主要针对的是服务器端在负载压力足够大的情况下是否能保证性能长期稳定。领域驱动设计可以很好地支持功能测试中的单元测试部分；而基于仓储和缓存的设计，应用领域驱动设计的项目通常在压力测试和负载测试上会得到令人满意的分数。

4.1 领域驱动设计与单元测试

支持与单元测试框架整合是采用 DDD 开发模式的一个重要特征。领域驱动设计通常对于测试驱动开发(Test Driven Development，简称 TDD)来说是很理想的，因为领域对象通常都采用 POJO 的实现方法，不过多依赖于容器或者其它基础设施代码，其状态和行为都包含在独立的领域类中，对于他们的单元测试是很容易的，可以不用太关注数据访问持久化的实现细节，还可以基于相同的测试用例十分容易的更换实现这些功能的底层组件，从而比较判断最佳的实施方案。

测试驱动开发提倡在写任何功能代码之前，先写它的测试代码，它不仅是一种测试技术，更是一种设计方法，会给项目开发带来很多好处：

（1） 测试先行会确保所写的代码就是功能所需要的代码，这是由于断言在先，可以确保实现的功能符合要求。

（2）测试先行可以更好地对类进行设计，因为测试先行让我们在编码前更多地考虑类和方法怎么样使用。在开发过程中，开发者可以不断修改功能代码使新增的测试用例，从而提升程序的设计。

测试先行可以让我们更自信地进行重构，因为应用 JUnit 可以很快地通过已设定好的测试用例对重构过的代码进行测试，确保重构的代码在功能上与原来的版本一致。

JUnit 是一个用来进行 Java 单元测试的开源框架，它的使用简单，应用广泛。

使用 JUnit 进行测试实际只需要两步：

（1） 继承 junit.framework.TestCase 类编写自己的测试类；

（2） 在测试类中编写自己的测试方法，格式如 testXXXXX( )。

JUnit 使用断言方法来判定单元测试是否通过。请看下面的例子：

```
public class TrainingChapterTest extends TestCase {
   TrainingChapterManager trainingChapterManager=new TrainingChapterManager();
   public void testGetByTrainingChapterId()
   {
      List chapterList = trainingChapterManager.getTrainingChapters();
      assertNotNull(chapterList);
      assertTrue(chapterList.size() > 0);

      TrainingChapter chapterInList = (TrainingChapter) chapterList.get(0);
      int chapterId = chapterInList.getId();
      TrainingChapter chapterInDb =
            trainingChapterManager.getTrainingChapter(chapterId);
      assertNotNull(chapterInDb);
      assertEquals(chapterInDb.getChapterTypeId()
                  ,chapterInList. getChapterTypeId ());
   }
}
```

上面的例子演示了一个对于 E-Learning 项目的 TrainingChapter 实体的根据主键获取对象的方法的 JUnit 测试用例。它首先调用了该实体的批量查询方法，然后选取返回对象列表的第一个对象的主键，再用主键查询方法获得一次对象，判定列表返回对象和主键获取对象属性是否一致，以此达到对实体类的 getTrainingChapters() 和 getTrainingChapters(String ID)方法的验证。

例程中的 assertNotNull，assertTrue 和 assertEquals 是 JUnit 的断言方法，表明程序编写者对于当前用例方法的测试结果的预期。JUnit 中还提供了如下其它类型的断言方法：assertFalse，assertNotSame，assertNull，assertSame。如果运行该测试方法后系统抛出 junit.framework.AssertionFailedError 的异常，则表示该方法执行失败，执行结果与断言的预期不符。

系统在开发与调试期间，难免会出现诸多问题，因此需要打印出系统中每个步骤的运行结果，这样可以跟踪调试。Java 中可以使用 System.out.println 来设置输出调试信息，但是这有一个缺点，就是当项目切入正式运行后，不希望这些日志输出到容器的控制台上。因为这样还会降低系统性能，但是如果手工逐个到程序中删除，又将为以后扩展维护带来不便。

Log4j 和 Java 平台标准版开发工具中的内建日志工具（JDK1.4 之后支持）是专门的日志调试工具，使用了它们就可以有效解决上述问题。Jakarta Common 日志工具（commons-logging，JCL）API 是 Jakarta 的另一个开源项目，它可以让 Java EE 项目在不必重新编译的前提下在各种日志工具间切换，以避免项目与特定日志工具的依赖。

JCL支持log4j和JDK日志工具，因此是更为推荐的日志解决方案。使用了JCL日志工具后，当开发和调试E-Leraning项目时，就可以到指定的日志文件中查看信息，跟踪错误信息，从而纠正问题

4.2 领域驱动设计与压力测试

压力测试是性能测试的重要环节。压力测试主要是为了发现在一定条件下软件系统的性能的变化情况，通过改变应用程序的输入以对应用程序施加越来越大的负载（并发，循环操作，多用户）并测量在这些不同的输入时性能的改变，也就是通常说的概念：压力测试考察当前软硬件环境下系统所能承受的最大负荷并帮助找出系统瓶颈所在。

压力测试的目标是测试在一定的负载下系统长时间运行的稳定性，尤其关注大业务量情况下长时间运行系统性能的变化（例如是否反应变慢、是否会内存泄漏导致系统逐渐崩溃、是否能恢复）；压力测试是测试系统的限制和故障恢复能力，它包括两种情况：稳定性压力测试：在选定的压力值下，长时间持续运行。通过这类压力测试，可以考察各项性能指标是否在指定范围内，有无内存泄漏、有无功能性故障等；

JMeter是Apache组织的开放源代码项目，它是功能和性能测试的工具，100%的用java实现，我们可使用它来对E-Learning系统进行压力测试试验（如图4.1）。

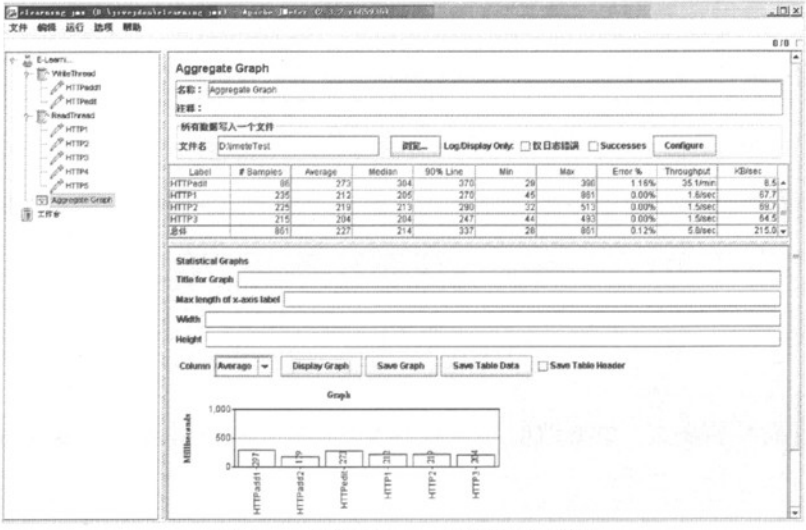

图4.1 使用JMeter进行压力测试

Fig. 4.1 Use JMeter to run pressure test

JMeter可以用于测试静态或者动态资源的性能（文件、Servlets、Perl脚本、java对象、数据库和查询、ftp服务器或者其他的资源）。JMeter用于模拟在服务器、网络

或者其他对象上附加高负载以测试他们提供服务的受压能力，或者分析它们提供的服务在不同负载条件下的总性能情况，并提供的图形化界面分析性能指标或者在高负载情况下测试服务器、脚本、对象行为。

在进行压力测试之前必须完成项目的开发与部署。由于 E-Learaning 项目截止目前只使用了 Java EE 中的 Web 框架，并且程序的生产环境还暂不需要考虑应用分布式部署，因此只要使用专门的 Web 容器服务器进行项目的发布即可，我们选用了 Tomcat 作为本项目的 Web 容器。Tomcat 是免费开源的 Web 服务器，是 Apache Jakarta 项目中的最著名的子项目，它还是 Sun 公司推荐的 servlet 和 jsp 容器。虽然很多人对免费或开放源代码的产品实现商业应用存有疑惑，但是 Tomcat 作为 Java 服务器经过几年来大量使用证明其稳定性和成熟性是有目共睹的。

前面 E-Learning 系统的建模的过程中，我们已经知道该系统的最大负载在于学员与系统的交互（而不是管理员与系统的交互，我们前面大量篇幅介绍 TrainingChapter 对象的 CRUD 方法的实现都是管理员与系统的交互），这些交互由应用层公开的服务传达到领域层后，最终主要都是在做数据的查询，例如，“初始化试卷”方法是系统根据一定的策略，随机抽取题库中的 Training Chapter 去组成 TestPaper，在这个过程中并没有对数据的增加、修改、删除；“答题”方法主要是完成将用户选取或者输入的答案与系统中的预设的答案进行比较，还是在做查询；只有在学员完成各级训练考试的最后步骤，系统判断用户已经通过了该项考试后，需要记录当前学员已通过该训练，此时才调用一次 update 操作更新数据库。根据以上行为的分析，我们判断 E-Learning 系统对生产环境的要求还是十分宽松的，尤其在使用了 Hibernate 和 JdonFramework 框架的缓存模块后，不需要很高的硬件配置就应该足够应付大批量用户的同时使用。

下面是 E-Learning 项目的一个压力测试实验的情况。

测试目的：测试 E-Learning 系统是否支持并发用户和长时间并发下的性能指标。

硬件条件：Web 服务器：P4 2.33G 四核，4G 内存；DB 服务器：P4 2.8G 双核，2G 内存；客户端：P4 2.8G 双核，2G 内存。

服务器软件设置：Web 服务器安装 Tomcat 5.5+JdonFramewok 框架，其中部署 E-Learning 项目作为被测试应用程序。运行 JDK 5.0 自带的 jconsole 组件监控 JVM 的运行情况；在 DB 服务器安装 MySQL 数据库。

客户端软件设置：在客户端机器安装 JMeter。

测试过程：

（1）设定 3 个查询线程与 1 个数据更新线程开始测试。测试期间 Web 服务器的 CPU 占有率在 10%左右波动，DB 服务器 CPU 在 1%~5%波动，稳定运行 1 小时，测试通过。

（2） 逐渐增加线程数进行相同的测试，判断CPU承受能力的临界点（CPU负载达到100%即认定为达到临界点）。最终认定临界测试条件为7个查询线程，4个数据更新线程，在此情况下Web服务器CPU占有率在75%~95%波动，DB服务器CPU占有率在20%~25%波动。稳定运行1小时后，测试通过。

（3） 设定5个查询线程，2个数据更新线程，测试系统最大连续工作时间。在该测试条件下，系统连续运行十小时左右，服务器和客户端一切正常。且在测试时间内，通过jconsole监控Server各项指标正常，无明显内存泄漏现象，测试通过。

基于条件（3）设定的硬件环境，时刻保持5查询+2更新线程相当于模拟一个30-50万PV的中型互动网站，E-Learning项目在单机环境下能够承受这样的模拟情景，绝对可以保证系统日后在生产环境下的正常运行。由此判定，本实验的目标达到。

综上，领域驱动设计对压力测试这项活动的指导意义也许并不大，但对它的产品进行测试的数据却是惊人的，这个数据对传统的数据库驱动开发方式进行开发的项目来说是绝难想象的，这也从另外的角度验证了领域驱动设计思想的强大与先进性。

结　论

本文主要从实践的角度探讨领域驱动设计在软件开发过程中的应用，并以其思想为指导，对一个日语学习网站项目 E-Learning 的建模与开发进行了详尽的讨论。在这个过程中详细地阐述了领域驱动设计在从建模到编码、测试等方面的重要理论和多种 Java EE 实施技术，对领域驱动设计在技术实现角度上目前比较优秀的轻量级框架解决方案 Spring 和 JdonFramework 也进行了分别论述，并用它们展示了运用 DDD 开发 E-Learning 项目的主要细节，从中可以体会领域驱动设计所带给我们的快速开发和高质量并重的特点。

文中的讨论主要对领域驱动设计的领域建模专题和以其在 E-Learning 这样的中小型系统的开发进行展开，但由于篇幅的限制，这个展开还很不够，例如，实际项目中模型通常不会只是简单 CRUD 功能，项目在安全性上会有很多的考量，项目的生产环境可能需要采用比 Tomcat 功能更强大的应用服务器乃至分布式部署。该如何应对这些挑战，都有待今后对领域驱动设计在轻量级开发领域的发展进行更深入的学习和实践。

最后要说的是，领域驱动设计是软件工程本身的一大进步，在理论上它使得原本被割裂的面向对象的分析阶段和设计阶段融合为统一建模；在技术实施上强调轻量级开发，与 DI、AOP、ORM、缓存等 Java EE 领域最先进技术有机结合，使其系统化，确实是值得每一个软件开发人员去学习和掌握的先进理念和技能。

参 考 文 献

[1] EVENS E. Domain-driven design——Tackling complexity in the heart of software[M]. Pearson Education, Inc, 2004.

[2] RICHARDSON C. POJOs in action: Developing enterprise applications with lightweight[M]. Manning Publications Co., 2007.

[3] BREIDENBACH R. Spring in action (second edition) [M]. Manning Publications Co., 2008.

[4] 夏昕 曹晓刚 唐勇等. 深入浅出Hibernate[M]. 北京:电子工业出版社, 2005.

[5] 彭晨阳. JdonFramework使用开发指南[EB/OL]. http://www.jdon.com/jdonframework/manual.htm.

[6] 彭晨阳. Java使用系统开发指南编著[M]. 北京:机械工业出版社, 2004.

[7] ECKEL B. Thinking in Java(second edition)[M]. 北京:机械工业出版社, 2002.

[8] COOPER J W. Java design patterns a tutorial[M]. Addison-Wesley, 2000.

[9] BLOCH J. Effective Java programming language guide[M]. 北京:机械工业出版社, 2003.

[10] LARMAN C. UML和模式应用[M]. 北京:机械工业出版社, 2008.

[11] 彭晨阳. 模型驱动软件开发实战步骤[EB/OL]. http://www.jdon.com/mda/mda.html.

[12] 简朝阳. MySQL性能调优与架构设计[M]. 北京:电子工业出版社, 2009.

[13] PENCHIKALA S, 王娟译. 领域驱动设计和开发实战[EB/OL]. 2009, 1, 15. http://www.infoq.com/cn/articles/ddd-in-practice.

[14] 彭晨阳. 数据库已死[EB/OL]. http://www.jdon.com/artichect/dbdead.htm.

[15] 高洪, 王未央, 杨斌. 基于组件的业务模型的研究与设计[J]. 微计算机信息, 2009, 25(6):57-58.

[16] 顾剑华, 赵文耘, 彭鑫. 基于本体的领域特征建模过程研究[J]. 计算机应用与软件, 2008, 25(2):7-9.

[17] 张凤英, 邹咸林. 基于领域模型的面向对象分析及UML建模[J]. 计算机应用与软件, 2004, 21(6):112-114.

[18] 刘楚达, 楚旺, 刘轶等. 面向问题域的领域建模方法[J]. 计算机工程与应用, 2003, 39(5):17-20.

[19] BERGER I R, STURM A. Utilizing domain models for application design and validation[J]. Information and Software Technology. 2009, 51(8):1275-1289.

[20] LANDRE E, WESENBERG H, OLMHEIM J. Agile enterprise software development using domain-driven design and test first [C]. The 22th ACM SIGPLAN conference on Object oriented programming systems and applications. Quebec, Canada, 2007:983-993.

[21] WESENBERG H, OLMHEIM J, LANDRE E. Using domain-driven design to evaluate commercial off-the-shelf software [C]. The 21th ACM SIGPLAN conference on Object oriented programming systems and applications, Oregon, USA, 2006:824-829.

[22] 姚易. 面向 Spring 框架的 MDA 模型转换方法研究[D]. 长春:东北师范大学, 2009.

[23] 廖芳. 基于模型驱动开发的林业信息系统业务建模研究[D]. 北京:北京林业大学, 2008.

[24] 张钊泉. 基于模型驱动的 BPM 软件开发平台的研究与实现[D]. 成都:电子科技大学, 2008.

[25] 陈成, 李行. 基于 AOP 的 MDA 模型转换[J]. 计算机技术与发展, 2008, 18(7):87-89.

[26] 林青, 许锁坤. 基于 J2EE 的企业级系统持久性框架的设计与实现[J]. 计算机工程与设计, 2007, 28(7):1732-1734.

[27] 王鹏, 刘渊, 冷文浩. 领域驱动设计在 SPP 系统中的应用[J]. 计算机工程与设计, 2008, 29(13):3362-3364.

致　谢

首先，我要感谢我的导师林鸿飞老师，论文的选题到问题的研究以至整个论文的完成的全过程都得到了导师极大的帮助和细心的指导。导师渊博的知识、严谨治学的作风、随和的为人态度以及对问题深入的见解，给了我极大的教诲。

同时，我也非常感谢在我学习、工作过程中各位授课的老师和公司的领导，通过此期间的工作和学习，让我在专业理论知识上和实践技能上都上了一个台阶。

最后，感谢大连理工大学电信学院，感谢学院为我们提供这样一个学习和交流的平台，使我们这些学子在这个平台上不断的完善，不断的提高。

Java EE项目开发与领域驱动设计实践

作者： 孙全智
学位授予单位： 大连理工大学
被引用次数： 1次

本文读者也读过(6条)

1. 郑琴琴 领域模型驱动的Web服务共享平台构建[学位论文]2009
2. 王鹏.刘渊.冷文浩.WANG Peng.LIU Yuan.LENG Wen-hao 领域驱动设计在SPP系统中的应用[期刊论文]-计算机工程与设计2008,29(13)
3. 龚畅 领域驱动设计方法的研究及在电信数据采集平台上的应用[学位论文]2007
4. 陈亮.CHEN Liang 基于领域驱动设计的软件开发方法和实例分析[期刊论文]-铁路计算机应用2010,19(9)
5. 史栋杰.孔华锋.SHI Dong-Jie.KONG Hua-Feng 领域驱动设计中资源库模式的设计与实现[期刊论文]-电脑知识与技术2010,06(33)
6. 张金松 领域驱动设计在航务海事系统中的应用研究[学位论文]2010

引证文献(1条)

1. 李引.袁峰 基于领域驱动设计的应用系统模型[期刊论文]-计算机工程与应用 2013(16)