分类号_____ 密级_____

UDC[注1]_____

学 位 论 文

基于开源框架的通用代码生成引擎设计与实现

（题名和副题名）

黄 钦

（作者姓名）

指导教师姓名 佘 堃 教 授

电子科技大学 成 都

（职务、职称、学位、单位名称及地址）

申请专业学位级别 硕士 专业名称 计算机应用技术

论文提交日期 2007.4 论文答辩日期 2007.5

学位授予单位和日期 电子科技大学

答辩委员会主席 [illegible]

评阅人 [illegible]

2007 年 4 月 22 日

注 1：注明《国际十进分类法 UDC》的类号。

摘 要

随着各行业信息化建设的迅速展开,都希望能够快速的构建适合自身业务需要的信息化系统。J2EE 作为一个新型成熟的分布式计算技术，已经广泛应用在很多领域，其可伸缩性、可扩展性的框架体系为应用系统带来了灵活的选择和实现；尤其是它的各层架构清晰和跨平台的特性，使得其在信息化建设领域广受欢迎和推崇。信息化系统的开发不但要解决技术上的难题，同时还需要面对纷繁复杂的业务需求，这种现状极大的影响了应用系统的成功率。如何帮助开发人员从烦琐的编码的技术细节中解脱出来，减少开发的工作量，把主要精力用于关注业务问题，提高开发效率和质量，正是本文关注点。

本文的目标是实现基于关系数据库的J2EE应用设计成果到应用项目实现代码的快速转化。通过参与多个基于 J2EE 技术平台的应用软件项目的开发工作，总结了这类系统从架构设计到实现细节上的共性，结合主流开源框架软件的特点，设计实现了快速开发应用系统的代码生成引擎。代码生成引擎固化了应用系统的架构设计及部分编码细节，使开发人员从繁琐的技术细节中解脱出来，并且开源软件是通过充分测试的。本课题包括 XML 建模体系构建和代码生成引擎设计实现两部分，项目需求和部分设计细节表现为 XML 需求模型文件，通过引擎解析 XML 文件生成业务层代码（CRUD ACTION、VALIDATION、CONVERSION、INTERNATIONALIZE 等），模型层代码(JAVABEAN、DAO、OR-MAPPING 等)及表示层代码。代码生成引擎在 ant 环境下执行，所生成代码格式规整有注释，支持多表关联等。最后，本引擎是独立的辅助开发工具，对其的任何扩展改进简单方便。

本课题的构思设计及最终实现均由作者独立完成。本文首先介绍了本课题涉及的知识体系，紧接着详细阐述了 XML 需求建模体系和代码生成引擎的设计、实现和应用。

关键词：代码生成引擎，J2EE，开源框架，XML

ABSTRACT

Following the informatization footsteps, most company all expect building some systems adapt to their business requirement. As a new form of mature distributed computing technology, J2EE is widely used in most fields. It is flexible to implement any application system with it's adjustable and patulous framework. Especially, for its plain structure of each layer and platform unattached feature,J2EE is a preferred choice for informatization construction. To developing information system, developers will be in the face of not only the technology details but also the complicated business requirement, which situation blocking the headway of success. How to help the developers break away from cockamamie technology details and pay more attention on business logic is our main role in this dissertation.

The purpose of this paper is to realize the fast transform of J2EE application based on relational database from design to practical code. This effort releases the developers from boring coding and technical details, so that they could pay more attention to the analysis and research of business and strengthen the quality of the software products. The author has attended some J2EE based application projects, has particular sight on the commonness of application projects. Combined with the open source framework, the author designed a new developing method, which consists of two parts: the XML requirement system and the code generation engine. Through parsing the XML requirement model files, the code generation engine build all the files the new project need, most java classes of the business layer (such as CRUD ACTION, VALIDATION, CONVERSION, INTERNATIONALIZE files), the model layer (such as DAO, JAVABEAN, OR-MAPPING files) and even the view layer. This engine is executed with ant, and the code built by the engine is neat and well commented. The engine is a self-governed tool, and it is convenient to improve it.

Code generation engine is developed independently. This dissertation describes the related knowledge system at first. Then, expatiates the XML requirement model system and the design and implementation of the code generation engine in particular, even the application of the engine.

Keywords: Code Generation Engine，J2EE，open source framework，XML

图目录

表目录

独 创 性 声 明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得电子科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

签名：黄欣　　日期：2007年6月23日

关于论文使用授权的说明

本学位论文作者完全了解电子科技大学有关保留、使用学位论文的规定，有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人授权电子科技大学可以将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、汇编学位论文。

（保密的学位论文在解密后应遵守此规定）

签名：黄欣　　导师签名：

日期：2007年6月23日

第一章 绪 论

1.1 研究背景

随着我国社会政治、经济、文化的全面发展，各行业的信息化建设的步伐日益加快。把计算机融入到日常工作中去，利用现代化的管理工具对本行业的工作进行必要的管理，这既是各行业加强自身管理的要求，也是提高工作效率以及信息的准确程度的要求。同时计算机技术的日新月异，信息系统更新的速度也越来越快。

目前国内很多正在规划建设信息系统，也有些正准备对现有的信息系统进行更新。目前国内有很多项目是遵循 J2EE(JAVA 2 Enterprise Edition)规范开发的，而且基于 J2EE 平台的应用现在呈上升趋势。并且大都是基于关系数据库系统的 WEB 应用系统。但国内当前情况是应用软件开发周期和实施周期长，实施成功率低。

基于 MVC 模式的公用框架的产生对项目研发带来了革命性的突破[1]。利用 MVC 模式的框架，可以针对项目进行研发的分层，实现项目的快速开发。但从目前现状来看，又面临着新的需求：

- 项目集中出现

因为大环境以及实际情况的影响，造成了新上项目往往集中在一个时间段内，但是人力资源是单位时间内有限的关键资源，这就造成了资源上的冲突，这种冲突就影响了公司承接项目的并发能力。

- 时间要求严格

新项目往往对时间要求很严格，在时间非常紧张的情况下，重新做设计并且编码是对研发人员的严峻的考验，造成了研发压力很大，项目质量有可能下降。

- 项目维护困难

研发人员水平、风格参差不齐，书写的代码千差万别，造成了程序风格不统一，修改、维护起来很不方便。造成了项目维护的效率下降。

根据上面的需求，迫切需要一种新的研发模式替代以前的研发模式。新的模式要解决上面的需求，对项目的研发起到推动作用。调高开发效率，缩短项目周期，同时提高软件的整体质量。针对 J2EE 平台上开发的应用软件，需要一个基于关系数据库的应用软件的 J2EE 平台的代码自动生成引擎。实现基于关系数据库的业务系统的设计成果到实现代码的快速转化，并且可以实现由非 J2EE 的基于关系数据库的应用到先进的 J2EE 的平台的快速转化。缩短设计成果到代码实现的转化的时间，节约人力资源的成本。解决最大限度的提高代码重用性、易维护性以及减轻开发人员负担的问题。帮助开发人员从烦琐的编码的技术细节中解脱出来，把更多的时间和精力投入到业务问题的分析和研究上，提高软件的质量。

1.2 国内外研究现状

随着编程技术的进一步发展，以及编程思想的百家争鸣，广大的开发人员和软件企业不再满足于仅仅能够进行相关的项目开发实现，更多的将注意力转向了项目开发的效率上，人性化体验上。代码生成技术作为一种能大大提高开发速度的发展方向，在国内外开始慢慢的发展起来，并且也取得了很大的成就。基于.net 和基于 java 甚至 php 技术的代码生成工具都逐渐出现甚至发展壮大，在基于 java 语言的代码生成工具主要集中在某些方面的应用，比如简单小型网站开发框架的代码自动生成，基于对象持久化技术的数据库访问代码自动生成，较好的例子有著名的 J2EE 开发插件 MyEclipse 就提供了从数据库到 DAO 及 DAOFactory，javabean，O/R Mapping 文件的生成工具。当今软件开发领域，速度和效率成了广大工程师追求的目标，很多工程师都想到了快速的代码生成来提高开发的效率，这是一个很有潜力的发展方向，很多组织和个人也在这个方面付出努力，并且涌现了很多的工具和思路，比如 GenerateJavaCode、Real-Time Workshop、Stateflow Coder、MyGeneration 等[2]，因为不同的组织和个人在代码生成方面的研究和侧重点各不相同，很少有应用于项目完整开发过程的代码生成工具出现。总的来说，虽然有众多的代码生成工具的涌现，但是能站在一个系统的高度来解决实际应用中某些问题的代码生成工具还是很少。

1.3 相关研究工作

近期剑桥 Forreste：研究中心基于对数百家 IT 组织的调查结果发布了两份应用发展报告，题目名为："模型驱动比 MDA 更有用"和"基于模式开发的吸引"。报告结果对模型驱动架构 MDA 并不正面[2]，而看好基于模式的开发(pattern-based development)和模型驱动的开发(model-driven development)"模型驱动比 MDA 更有用"报告中指出：众多 IT 组织都对 OMG 的 MDA 很感兴趣，但在实践中他们往往选择更实际可行的方法，例如模型驱动的开发(model-driven development)。MDA 是 OMG 指定的平台无关的模型应用方式，但是，调查报告显示，目前采用这一方式的人还太少。根据 OMG 的定义，MDA 中会创建平台无关的模型文档化业务功能和应用的行为，并和具体实现的技术平台代码相分离。相反，模型驱动的开发 MDD 中降低了 MDA 那种严格的要求，并为模型驱动的转换提供更多的实用价值；报告中指出，用户需要的是较低层次但更加实用的开发支持。

报告指出，尽管 MDA 的工具厂商们吹得天花乱坠，MDA 产品除了转换或简单的代码生成之外，其它的开发支持了了无几，而且因为 MDA 太通用所以其生成的代码也几乎不可用。而模式则封装了"最佳实践"的支持，基于模式的开发使用模式来解决一些公共的设计问题、技术实施并解决垂直领域的一些特定问题。

目前国外一些优秀的 IDE(集成开发环境)可视化开发工具，是针对具体代码的编写，不具备设计成果到代码的快速转化的功能。也有像 Rational 的工具可以实现分析、设计和代码自动实现：PowerDesigner 可以使用 UML 设计成果，同步在数据库中生成表结构。但往往价格昂贵。中小企业采用这种方案实现业务信息系统的成本太高。

开源社区有些代码生成工具，不过大都是针对某一层次的特定框架，生成的代码无法作为完整的应用，本地化不强，还需要大量的二期开发，以适应国内的使用现状。

国内也有些企业针对特定 J2EE 框架，开发了自主产权的代码生成器，但是限定在其独有的框架上，帮定在一起使用，较为封闭，且投资比较大，无法为其他企业，尤其是中小企业所使用所使用。

在 J2EE 开发领域，大都是基于关系数据库的 WEB 应用。且 Struts,Webwork Spring 和 Hibernate 等开源框架现在得到广泛的使用，大量的实践证明了这些框架的稳定性、可靠性和强大的功能，同时这些框架由于众多优秀程序员的贡献正日趋完善。Ant, Velocity 等开源工具也被广泛使用，有很好的用户群体和口碑，XML 语言也成为业界的一种标准。现在需要的是如何在此基础上，快速高质量的实现业务信息系统，以满意现在信息化建设的需求[3]。

1.4 论文结构组织

本文共分以下七个章节，逐步展现了本系统相关技术、系统分析、设计实现和应用的全过程：

第一章介绍了本项目的应用背景、国内外现状和本文的组织结构；

第二章介绍了本文涉及的主要技术之一 MVC 设计模式及基于此模式的优秀框架 Webwork；

第三章介绍了对象持久化理论及基于持久化理论的持久层设计及开源框架 Hibernate；

第四章讲述了本项目涉及的基于 XML 技术为项目进行需求建模的思想是本文设计理念的一个核心思想；

第五章首先介绍了软件项目需求的重要性和软件可重用性在提高生产效率方面的作用，接着重点阐述了系统适用的体系架构及需求建模体系的设计，最后介绍了基于 XML 建模体系的代码生成引擎的设计；最后紧承设计介绍了代码生成引擎在项目开发中的实际应用；

最后一章是对全文的总结，并且提出一些今后进行进一步完善的方面。

第二章 MVC设计模式与Webwork

MVC[1]是Model-View-Controller的简称，即模型-视图-控制器。MVC是Xerox PARC在20世纪80年代为编程语言Smalltalk- 80发明的一种软件设计模式，至今已被广泛使用，最近几年被推荐为Sun公司J2EE平台的设计模式，受到越来越多的Web开发者的欢迎。

本章对MVC模式的结构、特点、适用范围作了简单的分析，对典型的MVC开源框架Webwork从原理到核心技术组成进行了必要的分析。

2.1 MVC设计模式

2.1.1 MVC设计模式概述

在面向对象系统的设计中，可以认为，类及其生成的对象是构成面向对象系统的最基本元素；采用设计方法组合这些元素，得到构成面向对象系统的构件，同时，这些构件的设计方法在经过不断的改进和完善后逐渐成型，成为构成面向对象系统的基本设计参考，也称为设计模式。采用设计模式来抽象和总结系统构件的设计方法，同时将它用于新的系统构件的设计中。使用类和对象，使元素在层次上实现了重用性，而使用设计模式，则在系统构件的层次上实现了重用性。设计模式提供了在特定应用场景下解决问题的类、对象及相互关系的设计方法。这些方法并不针对于具体的系统，只是提供了一种设计系统的思考方法，一个设计模式可能用于不同的系统，一个系统也可能会用到多个设计模式。

通常一个设计模式由四个基本要素组成：

1. 模式名称(name)：设计模式的名称[4]；

2. 问题(problem)：描述了该设计模式的使用条件；

3. 解决方案(solution)：详细描述该设计模式的结构、实现、各组件之间相互关系及各自的职责和协作方式；

4. 结果(consequences):描述该设计模式的应用效果及使用该设计模式应权衡的问题。

在 GOF 所编写的《设计模式:可复用面向对象软件的基础》一书中一共提及 23 个标准的设计模式，在此基础上可以组合使用这些设计模式，也可以创建新的设计模式，按照不同的应用原则，可以将标准的设计模式分为三类:

1. 创建型设计模式(Creational Patterns):用于创建对象，如 Factory Method, Builder 等;

2. 结构型设计模式(Structural Patterns):用于将类和对象组合得到相应的结构，如 Adapter, Proxy 等;

3. 行为型设计模式(Behavioral Patterns):描述类和对象之间如何交互，一个任务如何交由不同的对象进行处理，如 Command, Visitor 等[6]。

2.1.2 MVC 模式的结构

MVC 设计模式由三部分组成。模型是应用对象，没有用户界面；视图表示它再屏幕上的显示，代表流向用户的数据；控制器定义用户界面对用户输入的响应方式，负责把用户的动作转成针对 Model 的操作；Model 通过更新 View 的数据来反映数据的变化[5][7]。三者之间的关系由下图示。

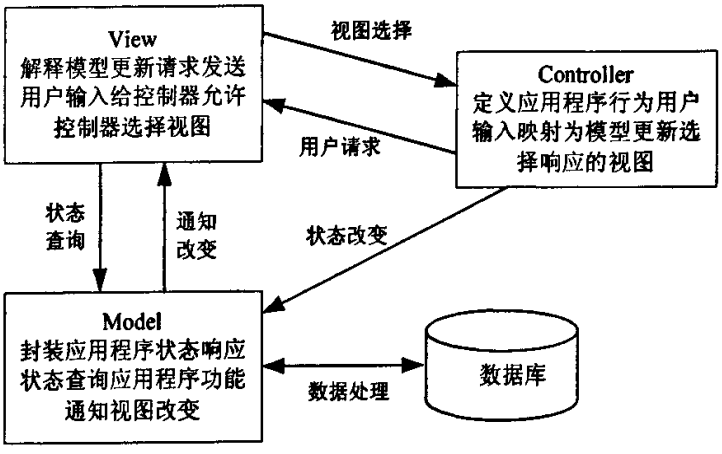

图 2- 1 MVC 结构图

（1） 视图（View）

代表用户交互界面，对于 Web 应用来说，可以概括为 HTML 界面，但有可能 XHTML，XML 和 Applet。随着应用的复杂性和规模性，界面的处理也变得具有挑战性。一个应用可能有很多不同的视图，MVC 设计模式对于视图的处理限于处理仅限于视图上数据的采集和处理，以及用户的请求，而不包括在视图上的业务流程的处理。业务流程的处理交予模型（Model）处理。比如一个订单的视图只接受来自模型的数据并显示给用户，以及将用户界面的输入数据和请求传递给控制和模型。

（2） 模型（Model）

就是业务流程/状态的处理以及业务规则的制定。模型接受视图请求的数据，并返回最终的处理结果。业务模型的设计可以说是 MVC 最主要的核心。MVC 并没有提供模型的设计方法，而只告诉用户应用该组织管理这些模型，以便于模型的重构和提高重用性。

业务模型还有一个很重要的模型那就是数据模型。数据模型主要是指实体对象的数据保存。比如将一张订单保存到数据库，从数据库获取订单。可以将这个模型单独列出，所有关系数据库的操作只限制在该模型中。

（3） 控制器（Controller）

可以理解为从用户接收请求，将模型与视图匹配在一起，共同完成用户的请求。划分控制层的作用也很明显，它清楚地告诉你，它就是一个分发器，选择什么样的模型，选择什么样的视图，可以完成什么样的请求。控制层并不做任何的数据处理。例如，用户点击一个链接，控制层接受请求后，并不处理业务信息，它只把用户的信息传递给模型，告诉模型做什么，选择符合的视图返回给用户。因此，一个模型可能对应多个视图，一个视图可能对应多个模型。

2.1.3 MVC 模式优点

在最初的 JSP 网页中，像数据库查询这样的数据层代码和像 HTML 这样的表示层代码混合在一起，而 MVC 模式从根本上将它们强制分开，这样给开发带来的

好处是毋庸置疑的。

首先，多个视图能共享一个模型。同一个 Web 应用程序会提供多种用户界面，比如用户希望既能通过浏览器来收发电子邮件，还希望通过手机来访问电子邮箱，这就需要 Web 网站同时提供 Internet 和 WAP 界面。在 MVC 中，模型响应用户请求并返回响应数据，视图负责格式化数据并把它们呈现给用户，业务逻辑和表示层分离，同一个模型可以被不同的视图重用，所以大大提高了代码的可重用性。

其次，模型是自包含的，与控制器和视图保持相对独立，所以可以方便地改变应用程序的数据层和业务规则。如果把数据库从 MySQL 移植到 Oracle，或者把 RDBMS 改变成 LDAP 数据源，只需改变模型即可。一旦正确实现了模型，不管数据来自数据库还是 LDAP 服务器，视图都会正确显示它们。由于 MVC 的三个模块相互独立，改变其中一个不会影响其它，所以依据这种设计思想能构造良好的松耦合的构件。

此外，控制器提高了应用程序的灵活性和可配置性。控制器可以用来连接不同的模型和视图去完成用户的需求，也可以为构造应用程序提供强有力的手段。给定一些可重用的模型和视图，控制器可以根据用户的需求选择适当的模型进行处理，然后选择适当的视图将处理结果显示给用户。

2.2 开源框架 Webwork

2.2.1 Webwork 简介

Webwork[8]是由 OpenSymphony 组织开发的[9]，致力于组件化和代码重用的拉出式 MVC 模式 J2EE Web 框架。Webwork 目前最新版本式 2.2，现在的 Webwork2.x 前身是 Rickard Oberg 开发的 Webwork，但现在 Webwork 已经被拆分成了 Xwork1 和 Webwork2 两个项目。

Xwork 简洁、灵活功能强大，它是一个标准的 Command 模式实现，并且完全从 web 层脱离出来。Xwork 提供了很多核心功能：前端拦截机（Interceptor），运行时表单属性验证，类型转换，强大的表达式语言（OGNL, the Object Graph Notation Language），Ioc(Inversion of Control 反转控制)容器等。

Webwork 建立在 Xwork 之上，处理 HTTP 的响应和请求。Webwork 使用 ServletDispatcher 将 HTTP 请求变成 Action（业务层 Action 类），Session（会话）Application（应用程序）范围的映射，Request 请求参数映射。Webwork 支持多视图表示，视图部分可以使用 Jsp，Velocity，FreeMarker，XML 等。

2.2.2 Webwork 的核心技术

Webwork 实际上是一个前端控制器（Front Controller）设计模式的实现。它通过一个分发器(Dispatcher)对象把客户端发送的 URL 请求映射到一个命令对象中，该命令对象在后台完成与系统内部模型的交互，并根据处理后的返回结果把控制权映射到某一个具体的视图，最后在视图层表达式语言 OGNL 对更新后的结果进行显示。其原理如下图 2-2 示：

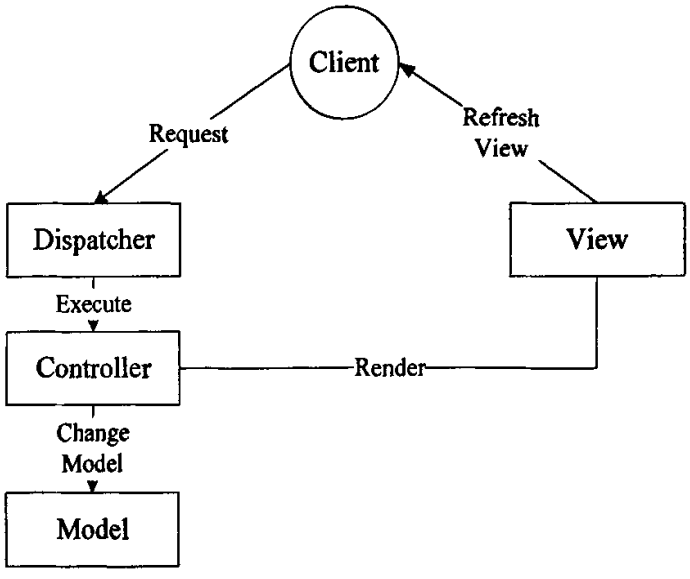

图 2- 2 Webwork 原理图

2.2.2.1 ServletDispatcher 的原理

ServletDispatcher 是 Webwork 框架的控制器。所有对 Action 调用的请求都将通过这个 ServletDispatcher 调度。它是在 web.xml 里配置 ServletDispatcher 时指定的，让所有对 Webwork Action（默认的是. action 的后缀）的请求都对应到该调度的 JavaServlet 中。ServletDispatcher 接受客户端的 HTTP 请求，将 JavaServlet 的很

多相关对象进行包装，再传给 XWork 框架，由 XWork 框架去解析 xwork. xml 配置文件，根据配置文件的信息，创建对应的 Action，组装并调用相应的拦截器，执行 Action，返回执行结果。Webwork 使用 XWork 的核心，主要是由这个 ServletDispatcher 去实现的。

ServletDispatcher 的主要功能调用如下：

1）init()方法在服务器启动时调用

a、初始化 Velocity 引擎，检查配置文件，对一些文件上传信息进行设置。

2）service()方法，每次客户端的请求都将调用此方法。

a、通过 request 请求取得 action 的命名空间（namespace，与 xwork.xml 配置文件里 package 标签的 name 对应）。

例如：/foo/tool/MyAction.action，取得的命名空间为/foo/tool 在 xwork.xml 配置文件里应该有这一段：

<package name="foo.tool"...

b、根据 servlet 请求的 Path，解析出要调用该请求的 Action 的名字（actionName)，例如：

(.../foo/tool/MyAction.action—>MyAction)

在 xwork.xml 配置文件里应该有：

<package name=" foo.bar"...

<Action name="MyAction"...

c、创建 Action 上下文(extraContext)。extraContext 是一个 Map 对象，它将所有应用请求和 servlet 属性保存到一个 Hashmap 中。

d、根据前面获得的 namespace、actionName、extraContext，创建一个 ActionProxy

ActionProxy 在它的构造函数会根据 namespace、actionName 读取 xwork.xml 配置文件里这个 Action 的所有配置信息。然后它会创建 ActionInvocation 的构造函数中又会创建 Action 对象，将它放入 OgnlValueStack 中，再根据 extraContext，创建与 Action 对应的上下文（ActionContext）。最后它会取得与这个 Action 相关的所有拦截器，放到 java.util.Iterator 对象中。

2.2.2.2 Action

Webwork 最重要的特征之一就是 Action 接口。它提供在页面与业务逻辑之间的映射，从而起到控制业务流程的作用。下面是 Action 接口的代码：

```
Package com.opensymphony.xwork;

Import java.io.Serializable;

Public interface Action extends Serializable{

Public static final String SUCCESS = "success";

Public static final String NONE = "none";

Public static final String ERROR = "error";

Public static final String INPUT = "input";

Public static final String LOGIN = "login";

Public String execute() throws Exception;

}
```

execute()是 Action 里最主要的方法。它执行后返回 String 类型，返回值一般用向上面定义的静态常量。这和 xwork.xml 配置文件中的 result 标签的 name 是对应的（关于 xwork.xml 文件的配置，将在下面介绍）。它决定了在调用 Action 的方法后，返回什么样的结果。上面定义的静态常量的含义如下：

SUCCESS：表示 Action 正确的执行完成，返回相应的视图；

NOTE：表示 Action 正确地执行完成，但并不返回任何视图；

ERROR：表示 Action 执行失败，返回到错误处理视图；

INPUT：表示 Action 的执行，需要从前端界面获取参数，INPUT 就是代表这个参数输入的界面，一般在应用中，会对这些参数进行验证，如果验证没有通过，将自动返回到该视图；

LOGIN：表示 Action 因为用户没有登陆的原因没有正确的执行，将返回该登陆视图，要求用户进行登陆验证。

由于一个用户请求就会对应一个 Action，因此所要做的就是写一些自己的 Action 类，这些类一定要实现 Action 接口，并填写 execute 方法。例如：

```
Public class HelloWorldAction implements Action{
Public String execute(){
System.out.prinltn("Hello World!");
//在这里可以调用其他的业务逻辑
Return SUCCESS;
}
}
```

可以看到，在 Action 的 execute 方法中，不需要任何参数，这样设计的一个很明显的好处就是，不将整个 Web 应用部署到 Web 容器中，即可对所写的 Action 类进行测试，这给开发人员带来了极大的方便。而且由于它不依赖于 servlet 请求，更加强了它的可移植、可重用性。

那么，Action 又是怎样来获得用户的输入数据的呢？下面是一段 xwork.xml 中的配置文件：

```
<action name="Hello" class="com.test.HelloAction">
<result name="success" type="dispatcher">
<param name="location">/hello-result.jsp</param>
```

```
    </result>
    <interceptor-ref name="params"/>
  </action>
```

假设已经有了 hello.jsp, 和 hello-result.jsp 这两个页面，hello.jsp 代码如下：

```
<html>
  <head>Hello</head>
  <body>
    <form name="" action="hello.action" method="post">
    <input type="text" name="name">
    <input type="submit" name="submit" value="Submit">
    </form>
  </body>
</html>
```

该页面就只有一个输入框和一个提交按钮。点击后，请求被 servletDispatcher 接收，它根据配置文件中的<action name="Hello" class="com.test.HelloAction">，将会调 HelloAction 来处理业务。result 标签指明了程序正确返回后将会转向的页面，这里的 name 是和 execute() 方法中返回的 success 对应的，<param name="location">/hello-result.jsp</param>指定执行成功后，将会转到 hello-result.jsp 页面。但是，如果用户要在 hello.jsp 中提交数据给 Action 处理，并得到处理后的结果，那么，Webwork 又是怎么获得用户输入，并返回结果的呢？这就不得不提到表达式语言 EL 和 OGNL 了。

在这之前，要先准备一个 User 类 javabean, 它只有一个成员变量 username，然后在 HelloAction 类中添加一个 User 类型成员变量 user 和一个 getUser()方法，该方法返回对成员变量 user 的引用。

2.2.2.3 表达式语言 EL 和 OGNL

OGNL[10]是 Object-Graph Navigation Language 的缩写，它是一种功能强大的表达式语言（Expression Language，简称为 EL），通过它的简单一致的表达式语言，可以存取对象的任意属性，调用对象的方法，遍历整个对象的结构图，实现字段类型转化等功能。它使用相同的表达式去存取对象的属性。例如，在上面的 hello.jsp 中，将输入框代码替换成："<input type="text" name="user.username">"在 hello-result.jsp 中，加入这样一段代码：" <ww:property value="user.username"/>"。可以看到，不管是输入还是显示，他们都用了同样的表达式：user.username。但是 input 框中的"user.username"，解析成 java 语句为 getUser().setUsername()；property 标签里的"user.username"解析为 java 语句：getUser().getUsername()。

在 xwork 框架中，ONGL 是被用来支持值堆栈 OgnlValueStack 的，它的功能就在于根据表达式来存取对象的属性。用户输入的数据会根据表达式保存到堆栈中的对象中，操作完成后，可以从堆栈中再次通过表达式来获得对象的属性。

在上面提到的 xwork.xml 配置文件里，有这样特殊的一句：<interceptor-ref name="params"/>，interceptor-ref 标签设置这个 Action 用到的拦截器（Interceptor），那么拦截器是指什么呢？

2.2.2.4 拦截器及其原理

Interceptor(拦截器)将 Action 共用的行为独立出来，在 Action 执行前后运行。这也是所说的 AOP（Aspect Oriented Programming, 面向方面变成），它是分散关注的编程方法，它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。

Interceptor 将很多功能从用户的 Action 中独立出来，大量减少了 Action 的代码，独立出来的行为具有很好的重用性。XWork、Webwork 的许多功能都是由 Interceptor 实现，可以在配置文件中组装 Action 用到的 Interceptor，它会按照你指定的顺序，在 Action 执行前后运行。这就将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。这就是通常所说的面向方面编程了。

在 xwork.xml 中的<interceptor-ref name="params"/>正是用到了一个 params 拦截器，它会在 HelloAction 的 execute()方法调用之前执行。作用是将 request 请求的参数值通过表达式语言设置到相应的 Action 模型里，例如上文的 hello.jsp 中的 <input type="text" name="user.username">, 它输入的值会由 HelloAction 类的 getUser()和 User 类的 setUsername("....")设置到这个 User 模型里。

假设在页面输入 "tang"，提交后，拦截器会首先从请求中取得参数的名字和名字对应的值，分别为："user.username"和"tang"，根据这个名字，从 OgnlValueStack 中取得堆栈最上面的 getUser().setUsername("tang")操作，即取得 HelloAction 对象的 User 模型，并设置 username 属性的值为 "tang"。在这里，input 输入框里的 name="user.username"必须遵守 OGNL 的命名规则。也正式拦截器的使用，才使得 Action 类和 Web 实现了完全的解耦，让 Action 能如此简单而优雅。

Webwork 框架给提供了很多实用的 Interceptor，它们都在 Webwork-default.xml 配置文件中定义。其定义格式为 <interceptors><interceptor name="timer" class="com.opensymphony.xwork.interceptor.TimerInterceptor"/>这个拦截器的功能就是记录 Action 执行的时间，并作为日志信息输出。当然还可以根据需要加入自己的拦截器。

2.2.3 Webwork 的优点

Webwork 的 Action 类仅需要实现接口 com.opensymphony.xwork.Action, 也可以实现其它的接口来实现更多的功能，譬如验证（validate）和国际化（localware）等。

在 Webwork 中，每个请求对应一个 Action。Servlet 容器对每个请求产生多个对象，这样就不会有线程安全问题，而且对性能也不会有什么影响。

Webwork 的 Action 不用依赖 Web 层和其它的容器。它可以通过 ActionContext 直接访问 Request 和 Response。但这个是可选的，只有在必须的请求下使用。正是由于它不依赖 Web 层和其它窗口，所以只需要对 action 赋予一定的属性，就可以执行单元测试。同时也可以使用一个 mock 的实例测试，而不是通过启动 web 容器来进行测试。

Webwork 的表达式语言使用了功能强大的 OGNL。它使用 OGNL 建立一个 OgnlValueStack 来搜索数据。Webwork2 前端也可以使用 JSTL，但它同时支持 velocity，freemarker，jspparer 和 xml。

Webwork 使用 OGNL 进行类型转化，提供了所有基本类型的转化功能。类型转化可以直接对一个 Class 进行（Class 级别）转化，也可以对 Class 的字段进行类型转化。它使用拦截器可以很容易的将类型转化的错误信息返回给用户，而且错误信息可以对应到一个相应的字段。

Webwork 允许您处理 Action 可以通过拦截器，就是在每一个 Action 处理前或后进行其它操作。它的拦截器可以在配置文件中动态添加，这样 Action 和拦截器之间完全解耦，更好的实现了组件化。

Webwork 使用 Xwork 的验证框架进行验证处理，它可以通过配置拦截器来激活。它可以为每个需要验证的 Class 指定一个 xml 验证文件，也可以为一个 Class 在不同的情况指定不同的 xml 验证文件。Webwork 验证可以给每个 Action 类指定对应的验证文件，也可以给 Action 的字段指定验证文件。通过拦截器来组装 Action 和验证文件，使他们之间完全解耦。

2.3 本章小结

本章介绍了代码生成引擎针对的应用系统中的 MVC 模式，从 MVC 的三层结构，每层的具体意义特点做了必要的介绍；同时阐述了优秀的开源框架 Webwork。从 Webwork 的分发原理到 Action 再到表达式语言和国际化方面的特色，还有 Webwork 特有的拦截器及原理走了精要的说明。为进一步讲述代码生成引擎应用的环境做了良好的铺垫。

第三章　持久层设计与 Hibernate 框架

现今的应用系统设计中，MVC 作为主流系统架构模式之一，贯穿了整个设计流程。MVC 中的 M，也就是所谓的 Model，则可以说是与业务逻辑和数据逻辑关联最为紧密的部分。而持久层作为 Model 层面中的主要组成，其设计的优劣势必对系统的整体表现产生至关重要的影响。

3.1 持久层概述

持久(Persistence)，即把数据保存到可掉电式存储设备中供以后使用。大多数情况下，特别是企业级应用，数据持久化也就意味着将内存中的数据保存到磁盘上加以“固化”，而持久化的实现过程则大多通过关系型数据库来实现[11]。

而所谓持久层，就是在系统逻辑层面上，专注于实现数据持久化的一个相当独立的领域。之所以独立出一个“持久层”的概念，而不是“持久模块”、“持久单元”，也就意味着，系统架构中，应该有一个相当独立的逻辑层面，专注于数据持久化逻辑的实现。与系统其他部分而言，这个层面应该拥有一个较为清晰和严格的逻辑边界，如图 3-1 所示。

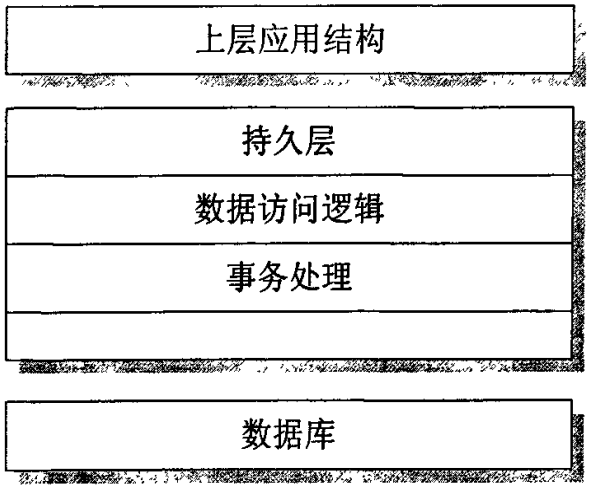

图 3- 1 持久层逻辑边界

上图的最大特色就在于各部分在边界和职责上的清晰划分。实际上，即使在设计思潮奔涌的今天，在开发的系统中，其中很多也没有完整的所谓“持久层”的概念存在[12]。那么，在系统中单独进行数据持久层的设计能为我们带来哪些好处呢？

首先，如果表示层发生了变化，我们的数据持久化代码不会受到影响；

其次，如果业务逻辑层发生了变化，我们的数据持久化代码也不会受到影响；

最后，反过来，如果底层数据持久化机制发生了改变，如更换数据库类型，系统中的非数据持久化的代码(包括表示层和业务逻辑层)将同样不会受到影响。

数据持久层的魅力如此之大，在我们的系统中单独进行持久层设计方面的考量将变得非常必要。

3.1 数据持久层设计

数据持久层的设计目标是为整个项目提供一个高层的、统一、安全、并发的数据持久机制。完成对各种数据进行持久化的编程工作，并为系统业务逻辑层提供服务。数据持久层能够使程序员避免手工编写访问数据持久层的方法，使其专注于业务逻辑的开发，并且能够在不同项目中重用本框架，大大简化了数据的增、删、改、查功能的开发过程，同时又不丧失多层结构的天然优势，继承延续 J2EE 特有的可伸缩性和可扩展性。

3.2.1 关键实现技术 DAO 模式

DAO (Data Access Object)模式实际上是两个模式的组合，即 DataAccessor 模式和 Active Domain Object 模式，其中 Data Accessor 模式通过将数据访问的实现机制加以封装，实现了数据访问和业务逻辑代码的分离，而 Active Domain Object 模式实现了业务数据的对象化封装[13]。

DAO 模式通过对业务逻辑层提供数据抽象层接口，实现了以下目标：

(1) 数据存储逻辑的分离

通过对数据访问逻辑进行抽象，为上层结构提供抽象化的数据访问接口。业务层无需关心具体的 select, insert，即 date, delete 操作，这样一方面避免了业务代码中混杂 JDBC 调用语句，使得业务逻辑实现更加清晰，另一方面，也使得开发人员的专业划分成为可能。某些精通数据库操作技术的开发人员可以根据接口提供数据库访问的最优化实现，而精通业务的开发人员则可以抛开数据层的繁琐细节，专注于业务逻辑代码。

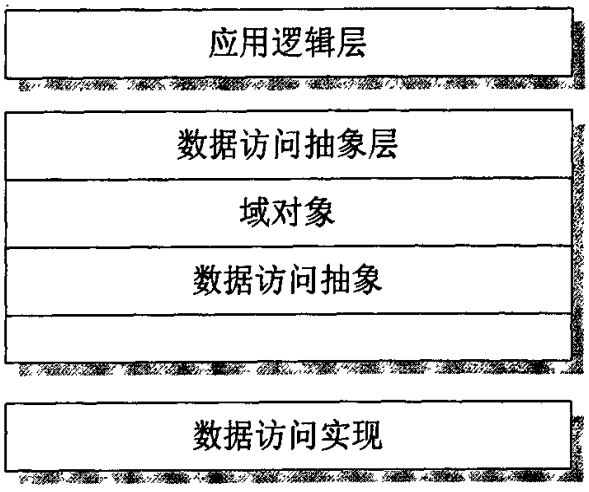

图 3- 2 DAO 模式的实现层次

(2) 数据访问底层实现的分离

DAO 模式通过将数据访问划分为抽象层和实现层，从而分离了数据使用和数据访问的底层实现细节。这意味着业务层与数据访问的底层细节无关，也就是说，我们可以在保持上层结构不变的情况下，通过切换底层实现来修改数据访问的具体机制。

(3) 资源管理和调度的分离

在数据库操作中，资源管理和调度是一个非常重要的课题，大多数系统的性能瓶颈往往并非集中于业务逻辑处理本身，而是系统涉及的各种资源调度过程。DAO 模式将数据访问逻辑从业务逻辑中脱离开来，使得在数据访问层实现统一的资源调度成为可能，通过数据库连接池以及各种缓存机制的配合使用，往往可以在保持上层系统不变的情况下，大幅度提升系统性能。

(4) 数据抽象

在直接基于 JDBC 调用的代码中，程序员面对的数据往往是原始的 RecordSet 数据集，诚然这样的数据集可以提供足够的信息，但对于业务逻辑开发过程而言，如此琐碎和缺乏寓意的字段型数据实在令人厌倦。DAO 模式通过对底层数据的封装，为业务层提供一个面向对象的接口，使得业务开发人员可以面向业务中的实体进行编码，通过引入 DAO 模式，业务逻辑更加清晰，且富于形象性和描述性，这将为日后的维护带来极大的便利。

3.2.2 提升性能的缓存和连接池技术

(1) 缓存(Cache)

对于数据库来说，厂商的做法往往是在内存中开辟相应的区域来存储可能被多次存取的数据和可能被多次执行的语句，以使这些数据在下次被访问时不必再次提交对 DBMS 的请求和那些语句在下次执行时不必再次编译。同样，数据持久层采用缓存技术来保存已经从数据库中检索出来的部分常用数据。客户端访问持久层时，持久层将首先访问缓存，如果能够命中则直接从缓存中提取数据，否则再向数据库发送提取数据的指令。这种设计能够大幅度的提高数据访问速度。

(2) 连接池(Connection Pool)

池是一个很普遍的概念，和缓冲存储机制有相近的地方，都是缩减了访问的环节，但它更注重于资源的共享。对于访问数据库来说，建立连接的代价比较昂贵，因此，数据持久层建立了“连接池”以提高访问的性能。数据持久层把连接当作对象，整个系统启动后，连接池首先建立若干连接，访问本来需要与数据库的连接的地方，都改为和池相连，池临时分配连接供访问使用，结果返回后，访问将连接交还。这种设计消除了 JDBC 与数据源建立连接的延时，同时在应用级提供了对数据源的并发访问。

3.2.3 持久层的实现类型

(1) 混杂模式

混杂模式是持久化功能的原始实现模式。即在业务类中混杂 JDBC 访问代码，

从而提供所需的持久化功能，如图 4-2 所示。

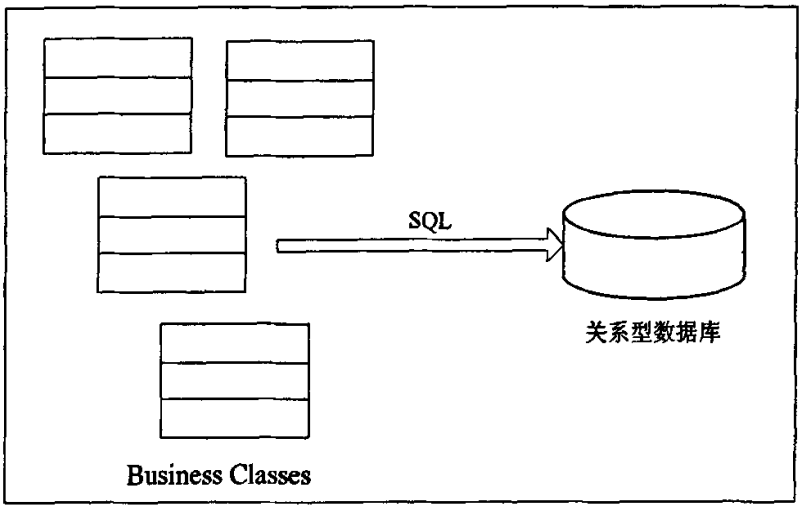

图 3- 3 持久层实现的混杂模式

这种模式的优点在于开发的迅速便捷。对于原形系统或者小型应用而言显得别具意义。但我们也应看到，基于这种模式开发的系统的，其维护性和扩展性较差，对象属性、数据库结构的变动都将直接导致业务逻辑代码的修改。实际上，在这种模式中，我们并不能清晰地分辨出所谓的“持久层”的逻辑层次。

(2) 基于 Data Class 的持久层实现模式

在这种模式中，数据类(Data Class)作为业务类与持久层沟通的桥梁，起着承上启下的作用，前面讨论的 DAO 既是这种模式的一个典型实现。DataClass 实际上包含了 DAO 模式中 Domain Class 和 Data Accessor Class o DomainClass 作为对现实世界的抽象，起着信息携带者的作用。而 Data Accessor Class 则通过 JDBC 代码将 Domain Class 与数据库表相关联，这种模式如图 4-4 所示。

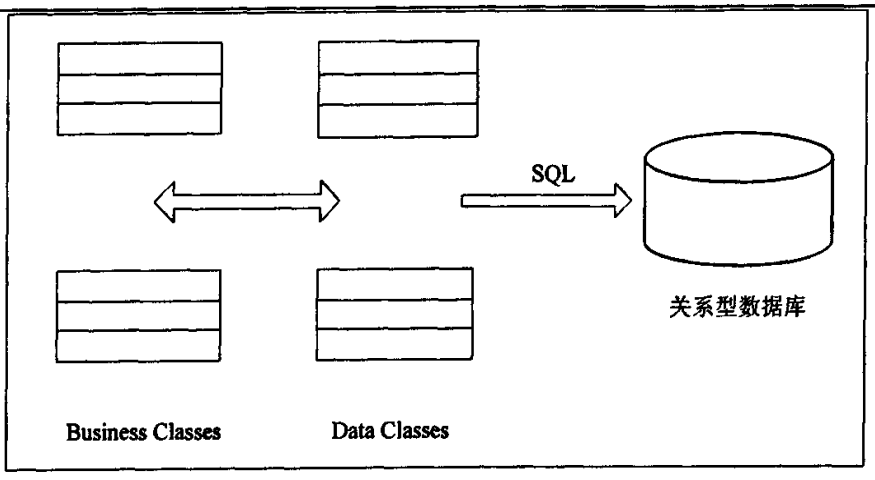

图 3- 4 基于 Data Class 的持久层实现模式

在这个模式中，我们实现了业务逻辑和底层数据结构之间的分离。DataClass 作为一个相对独立的逻辑层次，较为清晰的体现了所谓持久层的概念。底层关系数据结构的变化，可以较好的屏蔽在数据类这个层次，从而避免了对上层业务逻辑造成的影响。

而这种模式的缺陷也显而易见，随着涉及层次的增多，代码量的增加相当可观。相对第一种混杂模式而言，系统结构上得到较大提升的同时，项目设计和开发成本也相应增高。

所幸，Sun J2EE 规范制定小组和开源社区已经为我们提供了诸如 EntityBean. JDO, Hibernate, iBatis. Apache OJB 等杰出的持久层框架。而基于这些成熟可靠的第三方框架，于是我们有了下面的第三种解决方案。

(3) 基于现有持久层框架的实现模式

实际上，这种模式是第二种模式的延伸。Data Class 所包含的 Data Accessor 和 Domain Class 数量并没有减少，只是我们把最为繁琐的工作一基于 JDBC 的 OR 映射工作，交由第三方组件完成。Data Accessor 中的繁琐编码工作因此得到了空间简化，而与此同时，伴随持久层框架而来的辅助工具也大大减轻了 Domain Class 的编码负担。本文的持久层即是采用这种模式实现，如图 4-5 所示。

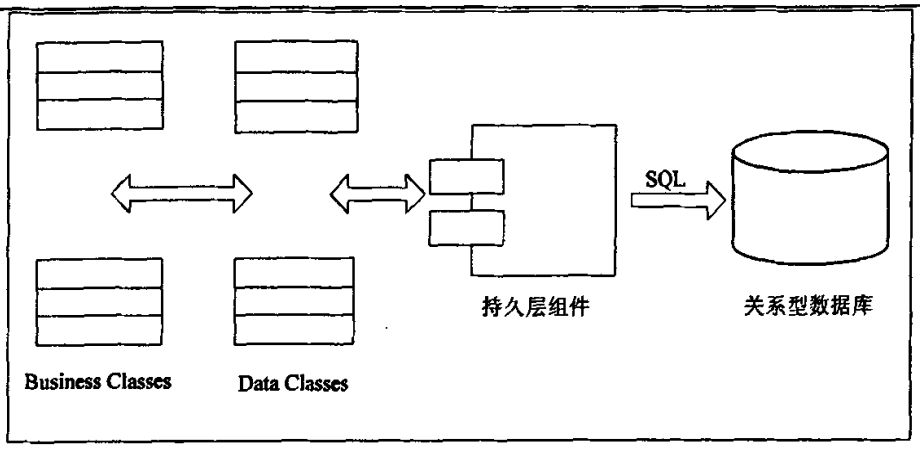

图 3- 5 基于现有持久层框架的实现模式

表达式求值是程序设计语言编译中的一个很古老又经典问题。通常书写的表达式是由操作数和运算符以及改变运算次序的圆括号连接而成的式子。常见的运算符包括单目运算符和双目运算符两类，单目运算符只要求一个操作数，并被放在该操作数的前面，双目运算符要求有两个操作数，其位置因表示方法不同而有所差异。

和其他具有独立功能的操作不同，表达式本身在用例的动态区中不构成独立的操作，表达式通常是作为带参操作的参数。例如：Assign 赋值操作，其赋值的目的操作字段和源操作字段我们就可以用表达式来处理；条件语句 if 、else if 、while 等语句，其条件可能不是一个简单的布尔类型值，大多数情况下可能中间经过了各种混合运算，此时，也是将其条件视为表达式，经过表达式处理后按照表达式列表布局格式存储在操作列表中。

表达式可以是一个常量或者一个消息字段等最基本组成单元，也可以是基本组成单元进行算术、逻辑或者关系运算组成。表达式大量用于各个脚本关键字操作中，它是一个通常模块。

3.2 持久层框架概述

在 Java 发展的初期阶段，直接调用 JDBC 几乎是数据库访问的唯一手段。随着近年来设计思想和 java 技术本身的演化，出现了许多 JDBC 封装技术[22]，这些

些技术为我们的数据库访问层实现提供了更多的选择，目前主流的 JDBC 封装框架包括:Hibernate, Apache OJB, iBatis, JDO 以及 J2EE 框架中的 CMP 等。这些框架以优良的设计大大提高了数据库访问层的开发效率，并且通过对数据访问中各种资源和数据的缓存调度，实现了更佳的性能。

在项目开发或架构设计中引入成熟的持久层框架实现，能给我们带来以下明显的好处[13][14]。

(1) 减少乏味的代码

持久层编码大多是些令人索然无味的内容:获取数据库连接、执行 SQL 语句、关闭数据库连接等，这些冗长、毫无创造性可言的工作充斥着整个编码过程。持久层框架封装了数据库持久层的大多数技术细节，如事务处理、数据库连接管理、SQL 生成。得益于这些成熟优秀的底层实现，我们可以从 JDBC 编码的炼狱中解脱，从而将精力和创造力投入到真正有价值的工作中。

(2) 更加面向对象的设计

目前的持久层框架，大多都建立在面向对象的设计思想之上。ORM 几乎是目前主流持久层框架的基本特性。ORM 为系统设计提供了更加自然的实现方式，我们可以通过 ORM 将系统中的 Domain Object 表，从而编码中只需关心 Object 的相关属性，而无需再纠缠于 JDBC ResulSet 中毫无意义的字段型数据。

(3) 更好的性能

持久层框架大多提供了优秀的性能优化机制，如内置的数据库连接池支持、Prepared Statement 缓存、数据缓存等。这些优化机制的综合使用大大提升了系统性能。更重要的是，由于设计上更加全面的考虑，这些机制对于上层架构完全透明，我们无需关心其中复杂的实现细节即可享用其所带来的性能提升。

(4) 更好的移植性

基于 Java 的跨平台特性，我们的系统可以在不同的操作系统之间切换，但由于数据库之间的差异，系统在数据库平台之间的迁移却遇到了阻力。上面提及的这

些成熟的持久层框架，由于设计上的良好隔离，从而提供了对不同数据库的良好支持，我们只需简单的修改其配置参数，即可实现底层数据库的切换。自动映射到各个数据库。

3.3 Hibernate 体系结构

Hibernate 充分体现了前面所述的设计理念，并针对实际应用开发进行了大量补充。它提供了强大、高性能的对象关系型数据库的持久化服务[13]。利用 Hibernate，开发人员可以按照 Java 的基础语义(包括关联、继承、多态、组合以及 Java 的集合架构)进行持久层开发。Hibernate 提供的 HQL 是面向对象的查询语言，它在对象型数据和关系型数据之间构建了一条快速、高效、便捷的沟通渠道。

目前比较成熟的 Hibernate2.1 支持几乎所有的流行的数据库，它可以和多种 Web 服务器良好集成，图 3-6 展示了 Hibernate 的完整体系结构。

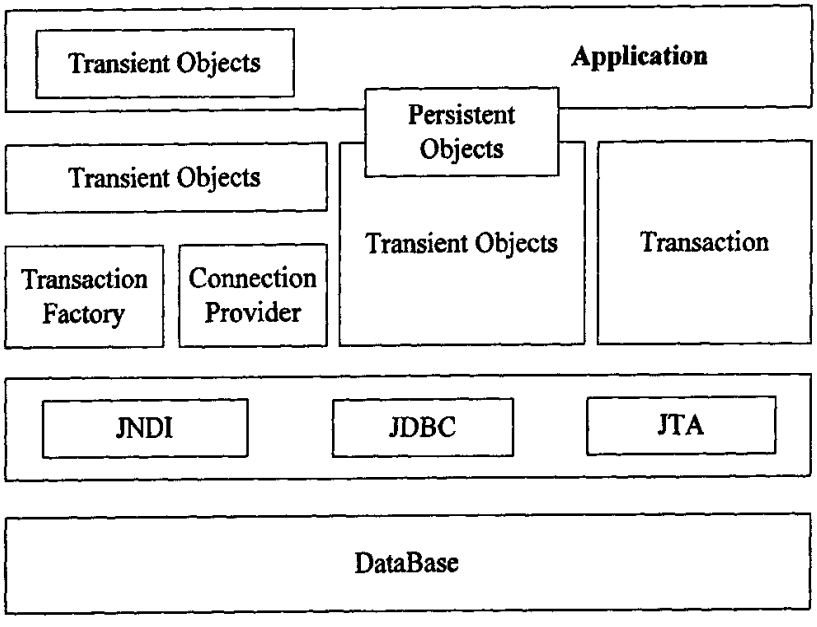

图 3- 6 Hibernate 体系结构

下面是对图 4-6 的基本释义：

会话工厂 SessionFactory：用于创建会话，是 Session 的工厂，也是

ConnectionProvide：的客户。它是对编译过的映射文件的一个线程安全的，不可变的缓存快照。

持久层对象 Persistent Objects：生命期短促的单线程对象，包含了持久化状态和商业功能。它们可能是普通的 JavaBeans，唯一不同的是它们现在从属于且仅从属于一个 Session

临时对象 Trasient Objects：临时对象是指目前还没有从属于一个 Session 的持久化类的实例。它们可能刚刚被程序实例化，还没有来得及被持久化，或者是被一个已经关闭了的 Session 所实例化的。

事务 Transaction：单线程生命期短促的对象。应用程序用它来表示一批工作的原始操作，是底层的 JDBC，JTA 和 CORBA 事务的抽象。它是由 Session 创建的，一个 Session 可能跨越多个 Transaction 事务。

ConnectionProvider：JDBC 连接的工厂和池。从底层的 Datasource 或者 DriverManage 抽象出来，对应用程序不可见。

TransactionFactory：事务实例的工厂。对应用程序不可见。

3.4 Hibernate 主要类

(1) Configuration

Configuration 类负责管理 Hibernate 的配置信息。Hibernate 运行时需要获取一些底层实现的基本信息，其中几个关键的属性包括：数据库 URL，数据库用户名和密码，数据库 JDBC 驱动类，数据库适配器（用于对特定数据库提供支持）。这些属性可以在 Hibernate 配置文件 hibernate.cfg.xml 中加以设定，当我们调用："Configuration config = new Configuration().config();"时，Hibernate 会自动在当前的 CLASSPATH 中搜寻 hiberante.cfg.xml 文件并将其加载到内存中，作为后续操作的基础配置。

Configuration 类一般只有在获取 SessionFactory 时需要涉及，当 SessionFactory 实例创建之后，由于配置信息已经由 Hiberante 绑定在返回的 SessionFactory 之中，

因此一般情况下无需在对其进行操作。

(2) SessionFacroty

SessionFactory 负责 ffIJ 建 Session 实例。我们可以通过 Configuration 实例构造 SessionFactory：

```
Configuration config = new Configration().config();
SesstionFactory sessionFactory = config.buildSessionFactory();
```

SessionFactroy 中保存了对应当前数据库配置的所有映射关系，同时也负责当前的二级缓存和 Statement Pool。由此可见，SessionFactory 的创建过程必然非常复杂、代价高昂，我们应该在系统设计中充分考虑到 SessionFactory 的重用策略。由于 SessionFactory 采取了线程安全的设计，可有多个线程并发调用，大多数情况下，一个应用中针对一个数据库共享一个 SessionFactory 实例即可。

(3) Session

Session 是 Hibernate 持久化操作的基础，其作为贯穿 Hibernate 的持久化管理器核心，提供了众多持久化方法，如 save, update, delete. fmd 等，通过这些方法即可透明的完成对象的增删改查(CRUD)。同时，Hibernate Session 的设计是非线程安全的，即一个 Session 实例同时只可有一个线程使用。

Session 实例由 SessionFactory 构建：

```
Configuration config = new Configration().config();
SesstiotrFaetory sessionFactory = config.buildSessionFactory();
Session session = sessionFacory.openSession();
```

之后，我们就可通过调用 Session 所提供的 save，get，delete，find 等方法完成持久层操作。

(4) Save

```
//新增名为“Emma”的用户记录
TUser user = new TUser();
```

```
user.setName("Emma");
session.save(user);
```

(5) Get/Load

```
//假设 tUser 表中存在 id=1 的记录
TUser user = (TUser)session.get('TIJser.class，new Integer(1));
```

(6) Delete

```
//假设 T User 表中存在 id=1 的记录
TUser user = (TUser)session.get(TLJser.class，new Integer(1));
session.delete(user);
```

(7) Find

```
//返回数据库中所有名为”Erica”的用户记录
String hql = "from TUser t where t.name=`Erica"';
List userList = session.find(hql);
```

3.5 Hibernate O/R 映射

O/R 映射关系无疑是 ORM 框架中最为重要的组成部分，也是日常开发中我们必须时刻关注的内容。对象和关系数据库之间的映射是用一个 XML 文档来定义的。这个映射文件被设计为易读的，并且可以手工修改的文档。映射语言是以 Java 为中心的，这意味着映射是按照持久化类的定义来创建的，而非数据库表的定义。

我们现在开始讨论映射文档的内容。我们只描述 Hibernate 在运行时用到的主要文档元素和属性，更详细的内容请参考相关文档[15][16][17]。

(1) Doctype

所有的 XML 映射都需要定义 Doctype 元素。Hibernate 总是会在它的 classpath 中首先搜索 DTD 文件。

(2) hibernate-mapping

这个元素包含三个可选的属性。Schema 属性，指明了这个映射所引用的表所在的 schema 名称。假若指定了这个属性，表名会加上指定的 schema 的名字扩展为全限定名。如果没有指定，表名就不会使用全限定名；Default-cascade 指定了未明确注明 cascade 属性的 Java 属性和集合类 Java 会采用什么样的默认级联风格;Auto-import 属性默认让我们在查询语言中可以使用非全限定名的类名。

```
<hibernate-mapping
    schema = "schemaName"
    default-cascade = "none | save-update"
    auto-import = "true | false"
    package = "package.name"/>
```

(3) dass

我们可以使用 class 元素来定义一个持久化类：

```
<class name = "C1assName" table = "tableName".....>
```

(4) id

被映射的类必须声明对应数据库表主键字段。<id>元素定义了该属性到数据库表主键字段的映射。

```
<id
  name = "propertyName"
  type = "typename"
  column = "column name"
  unsaved-value = "any | none | null | id_value"
  access = "field | property | ClassName">
  <generator class = "generatorClass"/>
</id>
```

(5) property

<property>元素为类声明了一个持久化的、JavaBean 风格的属性。

```
<property
    name = "propertyName"
    column = "column name"
    type = "typename"
    update = "true | false"
    insert = "true | false"
    formula = "arbitrary SQL expression"
    access = "field | property | ClassName"/>
```

(6) 多对一(many-to-one)

当数据库的两个表关联时，两个持久化类就通过这类的元素来表示关系。这种关系模型是多对一关联，实际上是一个对象引用。

```
<many-to-one name= "propertyName"
    column = "column name"
    class = "ClassName"
    cascade = "all | none | save-update | delete"
    outer join = "true | false | auto"
    update = "true | false"
    insert = "true | false"
    property-ref = "propertyNameFromAssociatedClass"
    access = "field | property | ClassName"/>
```

(7) 一对一

持久化对象之间一对一的关联关系是通过 one-to-one 元素定义的。

3.6 HQL 查询语言

Hibernate 拥有一种优秀的查询语言，简称 HQL[13]，它看上去很像 SQL。但是，HQL 是一种面向对象的语言，它具备继承、多态和关联等特性。由于 HQL 和 SQL 语言相似，很多 SQL 语言的语法，在 HQL 里一样能用，所以我们只要有 SQL 语言的基础，会很容易理解 HQL 语句的含义。

```
String hql=="from TUser";
Query query = session.createQuery(hql);
List userLast=query.list();
```

需要注意的是，Hibernate 中，查询的目标实体存在着继承关系，上面的语句将返回 TUser 以及 TUser 的子类所有对应记录，对应的 SQL 为“select*from T User” a Hibernate 查询非常强大复杂，实际上，它的强有力的查询语言是 hibernate 的主要特色之一。

3.7 本章小结

本章介绍了代码生成引擎设计的基础之一持久层问题。在 JDBC 作为数据库访问的主流途径后，很多的开源组织和个人都开始探索简化其使用的方法。从而有了持久层之说，也有对象持久化理论的产生，更有了很多持久化框架的涌现，通过增加一个中间层来屏蔽使用 JDBC 的繁杂工作虽然使代码量有所增长，但是，中间层带给开发人员统一易用的开发接口，使数据访问变的更加规整更加方便，大大的加快了开发的速度。本章的重点从持久化理论和持久层的设计问题开始，讨论了持久层设计的几种方式，并且用较多的篇幅介绍了目前最为流行的持久层框架 Hibernate，包括其体系架构，主要接口类，对象关系映射问题以至 Hibernate 查询语言。

第四章　基于 XML Schema 的需求建模

随着软件系统复杂程度的提高，对好的建模语言的需求也越来越迫切，面向对象建模语言就是应这样的需求而生。其实早在 20 世纪 70 年代就陆续出现了面向对象的建模方法，在 80 年代末到 90 年代中期，各种建模方法如雨后春笋般从不到 10 种增加到 50 多种。但方法种类的膨胀，使用户很难根据自身应用的特点选择合适的建模方法，极大地妨碍了用户的使用和交流。

在如此众多的方法流派的竞争中，UML（Unified Modeling Language，统一建模语言）举起了统一的大旗[23][24]。它融合了多种优秀的面向对象建模方法，以及多种得到认可的软件工程方法，消除了因方法林立且相互独立带来的种种不便。它通过统一的表示法，使不同知识背景的领域专家、系统分析和开发人员以及用户可以方便地交流。

它的出现为面向对象建模语言的历史翻开了新的一页，并受到工业界、学术界以及用户的广泛支持，成为面向对象技术领域占主导地位的建模语言。OMG（对象管理组织）采纳它为标准建模语言，进一步将它推向事实上的工业标准的地位，目前它正向 ISO（国际标准化组织）提出标准化申请。

在项目的需求阶段我们需要获取详尽的需求信息，借助相关的建模工具，比如 Rational Rose，Microsoft 的 Visio 等，将抽象的需求信息表示成一系列相互关联着的对象，然后我们的项目会随着工作的深入进入设计阶段，我们需要把具体的对象模型转化成一系列对应的数据模型，把这些模型映射到数据库中一系列数据表中，同时创建表与表之间的关联，这个环节是一个项目成功与否的关键，因为不同的项目复杂程度悬殊，对于比较复杂的项目我们就常常借助一些数据建模工具来做这个工作，比如较流行的 PowerDesigner，它能把需求模型表示成一个个表的缩影图，包含所有的属性以及与其他模型的关系，同时通过 PowerDesigner 我们还可以很方便的创建表到数据库中去。这是个很优秀的建模方式，不过借助 XML 我们同样可以开发一套适合多数项目开发需要的需求建模方式，这就涉及 XML Schema 技术和 UML 思想，本章主要介绍这些内容。

4.1 XML Schema 简介

XML Schema 是 W3C 的推荐标准，于 2001 年 5 月正式发布，经过数年的大规模讨论和开发，终于最终奠定下来，使得 XML 建模有了一个国际标准。XML Schema 一确定下来，立刻成为全球公认得首选 XML 环境下的建模工具，已经基本取代了 DTD 在 XML 刚刚成为 W3C 推荐标准时的地位。由于 XML 是 SGML 的一个子集，因此它也继承了 SGML 世界中用于建模的 DTD，当时使用 DTD 的好处是可以利用大量的在 SGML 世界中现有的 DTD 工具，使得开发应用代价维持在一个相对较低的水平。然而，DTD 有着不少缺陷：1）DTD 是基于正则表达式的，描述能力有限；2) DTD 没有数据类型的支持，在大多数应用环境下能力不足；3) DTD 的约束定义能力不足，无法对 XML 实例文档作出更细致的语义限制；4) DTD 的结构不够结构化，重用的代价相对较高；5）DTD 并非使用 XML 作为描述手段，而 DTD 的构建和访问并没有标准的编程接口，无法使用标准的编程方式进行 DTD 维护。而 XML Schema 正是针对这些 DTD 的缺点而设计的，XML Schema 是完全使用 XML 作为描述手段，具有很强的描述能力、扩展能力和处理维护能力。因此 XML Schema 必将取代 DTD 而成为定义和验证 XML 文档的标准。但与此同时，一些人也认为 XML Schema 过于复杂，他们发现很难直接根据业务模型定义出正确的 XML Schema，而且在用 XML Schema 表示出业务模型后，很难与用户和业务伙伴进行交流。针对上面这个问题，有必要开发一种基于 XML Schema 技术的项目需求建模方法。我们可以把 XML Schema 看作是业务模型的具体实现，而从与用户交流的角度来说，其它一些模型表示方法可能会比 XML Schema 更有效，而这其中 UML 是一种比较好的选择。UML 是一种功能强大的图形化建模语言，它可以用来为面向对象系统建模、描述系统架构和业务过程。用 UML 表示的产品易于理解，便于不同知识背景的客户和系统分析、设计、开发人员的交流，有利于产品的推广，也易于自我扩展。UML 的表示能力和直观的可视化形式要强于 XML 当前的表示能力[26]。

4.2 XML Schema 三层模型

由于难以从业务模型直接定义出XML Schema，因此参考软件系统的设计开发过程，引入了三层设计模式，分别对应于业务模型的需求、分析、设计的过程[27][28]。因为建模过程是一个由无形到有形，由粗到细的循序渐进的过程，所以把模型分

为三个层次，即概念层模型、逻辑层模型和实现层模型。概念层模型是用标准的UML类图来表示的；逻辑层模型也是UML类图，不过在概念层模型的基础上引入了一些构造型；实现层模型就是最终生成的XML Schema[29]。

为了更清晰地给出三层定义的不同，本文采用一个实例说明三层设计模式，描述图书馆中的一次借书记录（Transaction）：一个借书者（教师或学生）一次可借阅若干本书。

4.2.1 XML Schema 的概念层模型

在定义概念层模型阶段，通过精化和组织需求捕获阶段所描述的需求（用UML用例图描述）来对其进行分析，把整个系统划分为易于管理的各个部分，充分考虑复用可能性[28]。

由于教师和学生都可以作为借书者，且他们的部分属性是相同的，因此可以把这部分相同的属性放在一个类person中，把教师和学生作为这个类的子类。把Transaction分为两个部分来定义：一部分是核心的transaction类型；另一部分是person类型。在UML中，这两部分定义可以作为两个（package）包。图4-1显示了的transaction概念层模型。

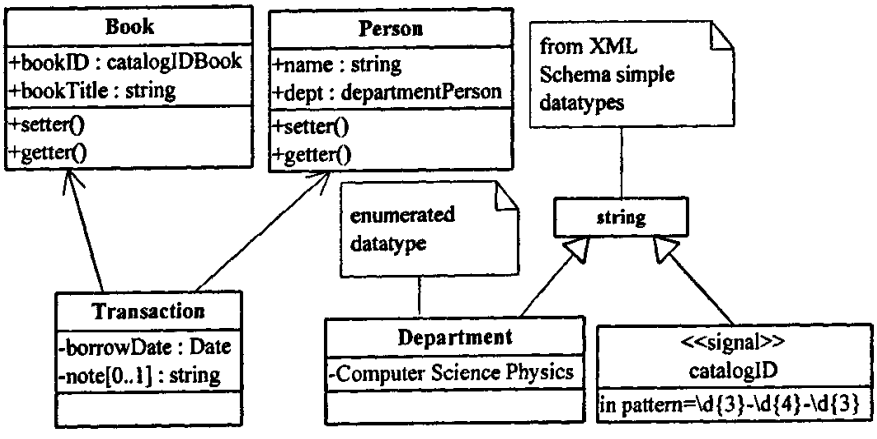

图 4- 1 transaction 的概念层模型

概念层模型是用标准的UML类图来表示的，并通过加入一些注释来对数据类型、约束条件等进行说明。为了便于映射到XML Schema，约定类图中属性的数

据类型使用的均是XML Schema内建的简单数据类型。因为需要的只是数据模型，而不是对象模型，所以只需定义静态类型，而不用考虑类中的操作。

4.2.2 XML Schema的逻辑层模型

通过已经建立的概念层模型，可以与领域专家、用户进行交流，以验证其正确性。本节给出了把该模型转换为逻辑层模型的方法。由于在从UML到XML Schema的映射过程中，可能会面临很多种不同的选择，不同的人按不同的方法生成的XML Schema可能会不一样。例如：UML中属性和关联应该映射为XML Schema中的属性还是元素；UML中类和数据类型的泛化关系怎样映射到XML Schema的定义。

因此，为了解决在从UML到XML Schema的映射过程中的这些问题，使用了UML的扩展机制，分别为构造型(stereotype)、标记值(tagged value)和约束(constraint)三种。这里，主要使用了构造型和标记值。例如，可以用构造型<<simpleType>>来表示这个类应该被映射为用户定义的简单数据类型。可以使用{modelGroup}来定义一个类中的属性及关联在被映射为XML Schema的元素后应该使用哪种模型组，all、choice还是sequence。定义的几种构造型如表一。

表4- 1 构造型

构造型名称	作用于	意义
<<comlexType>>	类	映射为Schema的复合类型
<<simpleType>>	类	映射为Schema的简单类型
<<element>>	属性或关联	映射为Schema的元素
<<attribute>>	属性或关联	映射为Schema的属性
<<facet>>	属性	使用Schema的刻画
<<XSDsimpleType>>	类	Schema内建的简单类型

在一个UML模型中，可以为一些所使用的扩展机制设置默认值。例如，可以把{modelGroup}的默认值设为sequence，这样，就不必为每一个类分别设置这个值。在下面的例子中，把UML类属性缺省的映射为XML Schema的元素，需要映射为XML Schema的属性的时候，使用构造型<<attribute>>加以说明。图4-2 显示了transaction的逻辑层模型。

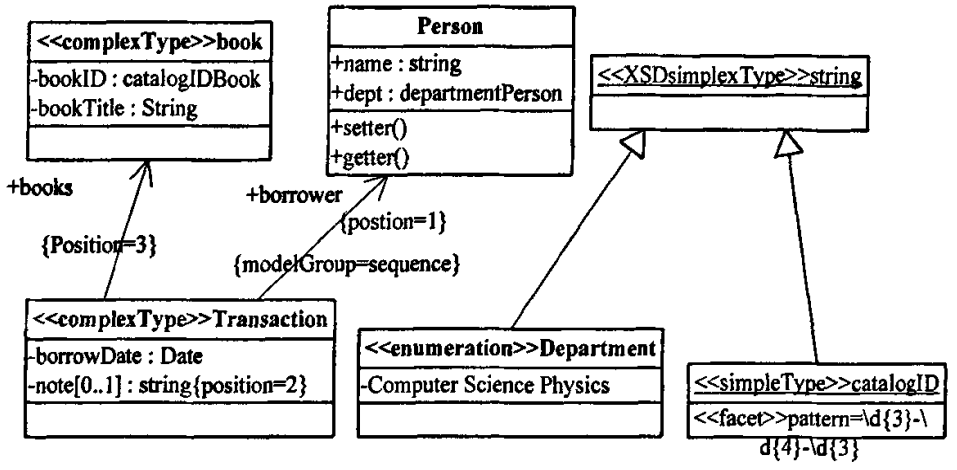

图 4- 2 transaction 的逻辑层模型

4.2.3 XML Schema 的实现层模型

在得到了逻辑层模型后，就可以根据它来生成最终的实现层模型，也就是XML Schema。下面定义了一种从逻辑层模型到XML Schema的映射方法。

4.2.3.1 类和属性的映射方法

在UML中一个类定义了一种复杂的数据结构，它应该缺省地映射为中的。这一点在图4-2中已经用构造型<<complexType>>表示出来了。transaction类和它的属性被映射为如下的XML Schema定义：

```
<xsd:complexType name="transactionType">
<xsd:sequence>
<xsd:element name="note" minOccurs="0" maxOccurs="1"/>
</xsd:sequence>
<xsd:attribute name="borrowDate" type="xsd:date"/>
</xsd:complexType>
```

UML中的属性的多重性在XML Schema中用minOccurs和maxOccurs表示。我们为XML中每一个complexType缺省的生成一个顶层元素。transaction的顶层元素如下：

<xsd:element name=transaction type=transactionType/>

4.2.3.2 关联的映射方法

图4-2的transaction类不仅包括属性，它还包括两个关联，borrower和books。每一个关联都有一个角色名以及多重性说明。这些关联被映射为complexType中的元素，与从UML类的属性映射而来的元素放在一起。

<xsd:complexType name="transaction">

<xsd:sequence>

<xsd:element name="borrower" type="person"/>

...

</xsd:sequence>

<xsd:attribute name="borrowDate" type="xsd:date"/>

</xsd:complexType>

为了表示各个元素在这个序列中的顺序，在中使UML用了标记值。

4.2.3.3 用户定义数据类型的映射方法

在图中，我们定义了catalogID数据类型。在缺省情况下，用户定义数据类型将被映射为XML Schema中的复合类型。但在这个例子中在这个类型的定义中使用了构造型<<XSDsimpleType>>，所以在映射时将把它们映射为XML Schema中的简单类型：

<xsd:simpleType name="catalogID" base="xsd:string">

<xsd:pattern value="\d{3}-\d{4}-\d{3}"/>

</xsd:simplType>

为了对catalogID类型加以限制，可以使用XML Schema中的(facet)刻面。为了反映这一点，在图4-2的catalogID中加入了属性pattern，它将被映射为XML Schema简单类型的刻面Pattern

4.2.3.4 泛化的映射方法

面向对象分析和设计中的一个重要概念就是泛化，子类可以从它的父类那里继承属性和关联。在person类型的定义中， teacher是person的子类。根据缺省规则，把person和teacher映射为XML Schema的复合类型：

```
<xsd:element name="person" type="person" abstract="true"/>
<xsd:complexType name="person" abstract="true">
<xsd:element name="name" type="xsd:string"/>
<xsd:element name="dept" type="department"/>
</xsd:complexType>
<xsd:complexType name="teacher">
<complexContent>
<extension base="person">
<xsd:element name="teacherNo" type="workNo">
<xsd:element name="teacherTitle" type="title">
</extension>
</complexContent>
</xsd:complexType>
```

在person的顶层元素和复合类型定义中包含了一个属性abstract="true"，这说明它是从一个抽象类映射而来的。复合类型teacher是从person扩展而来，这相当于UML中的继承关系。由于person的属性abstract="true"，因此在元borrower素中不能使用类型自身，而应该使用从它扩展而来的person或student。

4.2.3.5 枚举类型的映射方法

person中的元素引用了简单数据类型定义department。在中是一个枚举类型，我们给department department UML它加了一个构造型来表明在映射为 <<enumeration>> XML时将生成一个刻面： Schema enumeration

```
<xsd:simpleType name="department" base="xsd:string">
<xsd:enumeration value="Computer Science"/>
```

<xsd:enumeration value= "Physics"/>

</xsd:simpleType>

4.3 XML Schema 的模块化及复用

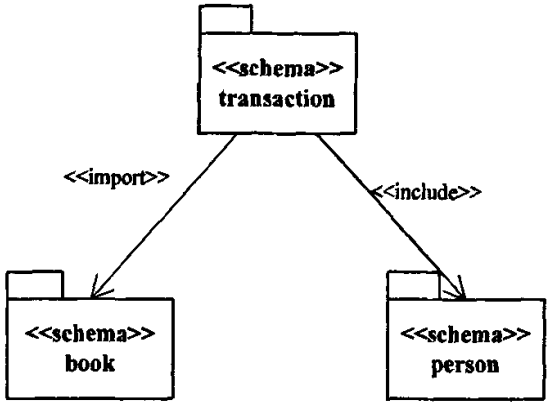

图 4- 3 Schema UML 图模块依赖关系的包

在为 XML Schema 建模的过程中使用 UML 的一个好处是它有利于在 XML 应用中实现 Schema 的模块化及复用。在前面的例子中， transaction 和 person 分别在两张 UML 类图中进行了描述，可以把它们看作两个模块，并且在必要时可以重用这些模块。例如，如果另一个 XML Schema 中也需要使 person 类型，就没有必要重新定义它，只需要简单地引用已有的 person 模块就可以了。甚至可以自定义 person 的新的子类。UML 中用包来表示各个模块以及它们之间的依赖关系。如果把 book 的类型声明也作为一个单独的模块，并使用与 transaction 和 person 不同的目标名称空间(target- Namespace)，那么可以用图三表示这个模块之间的关系。

在映射为 XML Schema 的时候，每一个包都生成一个单独的 schema 文件。不同包之间的依赖关系在 XML Schema 中用<include>或<import>来表示。到底是使用<include>还是使用<import>依赖于相关包的目标名称空间是否相同，当它们的目标名称空间相同时使用<include>，否则使用<import>。

4.4 本章小结

本章首先介绍了 XML Schema 技术，接下来以实例的形式介绍了基于 XML Schema 的三层模型，和 XML Schema 的模块化和复用问题。基于 XML Schema 技术的 XML 需求建模是本文后续章节的一个重点内容，也是设计代码生成引擎不可缺少的前提准备因素，在本章介绍 XML Schema 技术主要是为了在后文介绍基于 XML Schema 的元素设计，需求建模思想做一个理论基础的铺垫。

第五章　需求建模体系与代码生成引擎设计

随着时代的发展，业界对软件开发的速度和质量要求越来越迫切，很多优秀的思想和实践都聚焦在如何提高软件开发的速度和质量上，作为软件工程，首先软件需求是一个相当关键的环节，其次一个系统的架构和设计也是关系软件质量性能的决定因素。当今软件开发领域，速度和效率成了广大工程师追求的目标，很多工程师都想到了快速的代码生成来提高开发的效率，这是一个很有潜力的发展方向，很多组织和个人也在这个方面付出努力，并且涌现了很多的工具和思路，因为不同的组织和个人在代码生成方面的研究和侧重点各不相同，很少有关于应用于项目完整开发过程的代码生成工具出现，在我们实际的开发工作中，一直在探索更加可靠高效的开发方式。我们发现很多应用系统，不论复杂程度的大小都具备一些相似的特性，比如基本都有类似的层次划分，视图层，业务逻辑层，数据模型层等，只是系统的构建方式存在差异。因此我们寻找了一条应用基于 MVC 模式的框架组合快速开发项目的途径。这种方式借助 XML 来做需求建模，通过专门的代码生成引擎，简单讲就是一个 XML 解析器，生成一个应用系统的主要代码，从而简化应用开发的代码量和减少不必要错误的出现。

5.1 软件项目的需求工程

软件需求是指用户对目标软件系统在功能、行为、性能、设计约束等方面的期望。通过对应问题及其环境的理解与分析，为问题涉及的信息、功能及系统行为建立模型，将用户需求精确化、完全化，最终形成需求规格说明，这一系列的活动即构成软件开发生命周期的需求分析阶段[20]。

需求分析是介于系统分析和软件设计阶段之间的桥梁。一方面，需求分析以系统规格说明和项目规划作为分析活动的基本出发点，并从软件角度对它们进行检查与调整；另一方面，需求规格说明又是软件设计、实现、测试直至维护的主要基础。良好的分析活动有助于避免或尽早剔除早期错误，从而提高软件生产率，降低开发成本，改进软件质量[32][33]。

人们逐渐认识到需求分析活动不再仅限于软件开发的最初阶段，它贯穿于系统开发的整个生命周期。软件需求建模是软件工程中一个非常重要的环节,同时也是一个非常困难的环节。在现在的软件开发过程中，我们可以借助很多的工具来进行软件的需求分析，比如 Rose，Visio，RationalXDE，Together 等，他们给软件开发者提供了一整套的可视化建模工具，包括系统建模、模型集成、软件系统测试、软件文档的生成、从模型生成代码的前向工程、从代码生成模型的逆向工程、软件开发的项目管理、团队开发管理等。但是这些工具都是在 UML 的基础上衍生出来的一系列辅助工具。UML 定义良好、易于表达、功能强大，不仅支持面向对象的分析与设计，而且支持从需求分析开始的软件开发全过程[41]。通过这些工具软件设计人员可以很清晰的勾勒出一个完整的系统，方便进一步的设计和实现。在软件的设计阶段，设计人员还得把需求模型（对象模型）创建出对应的数据模型，常常设计人员可以借助 PowerDesigner 来完成这个工作，而且模型的每一个属性，模型间的每一个关系都可以反映出来，然后再把数据模型转化成一张张数据表，通过键关联来反映这些对象（表）间的关系。有这些工具很大程度的提高了软件开发的质量和速度，但是我们在开发项目的时候遇到了新的问题，比如，我们做完需求分析后一般都形成了描述文档，然后再转化成数据库中的表，系统的设计，这样步骤繁多而且关联紧密，可能很麻烦；还有，当我们用 PowerDesigner 设计好数据模型并且构建到数据库后，项目的开发过程需求发生一点点变化，我们就得做一系列的修改，从文档，模型到表，数据，不堪其烦。需求变化是不可避免的问题，首先要有良好的分析设计开发的方法才能减少需求变化带来的额外工作。

5.2 面向对象的开发方法

对传统的软件开发方法如功能分解、数据流方法和信息模型方法来讲，其最根本的缺陷在于从需求分析过渡到设计时表示方法的转变，有的时候这样的转变过于突然以至于很难进行逆向追踪，同时对于系统的变化无力捕捉，不能适应系统的不断变化，这是开发的软件系统很难适应用户的需求，大量的系统归于失败的重要原因。在对传统开发方法缺陷进行反思的基础上，人们提出了面向对象的软件开发方法(包括面向对象的分析、面向对象的设计和面向对象的实现)，这种方法的最大优点是它提供了从需求分析、设计到实现的一致的表示方法，同时把握了系统中最稳定的因素，也就是问题空间的对象。面向对象的分析(OOA)由五个步骤，即标识对象、标识结构、定义主题、定义属性(及实例连接)、定义服务

及消息连接)组成的。OOA 模型由主题层、对象层、结构层、属性层、服务层所构成。面向对象的设计(OOD)模型在与这五层对应的发现类及对象、识别结构、识别主题、定义属性、定义服务的基础上。由设计问题域、设计人机交互、设计任务管理、设计数据管理等部分构成。面向对象的开发方法作为软件开发方法的一次根本变革，符合人类认识客观世界的规律，反映了人们概括客观事物的三个基本方法(即区别对象及其属性、分类结构和组装结构)，必将具有广阔的发展前景[37][42]。这种开发方法从分析、设计到实现的过程也就是类及其对象的不断扩充不断细化的过程，在分析阶段识别的类及其对象可以在设计(也可以分为概要设计和详细设计)、实现阶段重用，在设计阶段增加的类及其对象可以在实现阶段重用，在分析、设计和实现阶段识别的类及其对象可以存入类库中，并提供相应类及其对象的说明书，使得它们能应用在未来的软件开发过程中。

在常规的过程式程序设计环境中，软件部件主要是过程和函数，这样的部件由于匹配新需求的机会比较少，因而重用相对来说是比较困难的。如果没有合适的工具的支持，修改和一般化这些部件是比较困难的，甚至会由于模块间的高度藕合(比如为了提高效率而对大量全局数据的访问)，重用基本是不可能的。在面向对象的程序设计环境中，普通的模块是容易重用的类。因为类通过继承、重载(包括成员函数的重载和操作符的重载)等可以相当容易地适应新的需求，这样类比过程部件有更多的重用机会。

在常规的过程式程序设计中，软件部件(过程和函数)在测试之后，存入可重用部件库中，用于将来的重用。这当然提供了一些重用。但在高度特殊的应用领域，这样的部件的重用性是很有限的。在大多数情况下，部件需要修改以适应将来的应用环境。由于普遍缺乏模块详细设计说明书，缺乏与其它模块的接口细节信息和数据耦合的详细说明，修改模块不仅是困难的，而且可能引入新的错误，带来无法预测的后果。因此，在常规语言设计环境下，为了提高模块的重用性，要求在软件开发中遵循软件工程的开发规范，每一步骤都要留下经复审合格的软件文挡。否则，修改模块的成本可能超过重新开发的成本，而且其质量不容易保证，维护的成本也没有得到节约。

相反，在面向对象的程序设计环境中，软件的适应性不是通过修改已有的部件实现的，而是在继承的基础上，通过扩展(加入适应新需求的静态属性即数据成员和成员函数即方法)和特化(specialize，可以理解为通过虚函数的覆盖即多态机

制实现的)已有的部件(类)完成的。这样，即使软件部件并不确切匹配新的需求，仍然可以实现重用，这时的重用是通过类的继承和封装机制实现的。

在面向对象的程序设计中，封装(encapsulation)和继承(inheritance)发挥了重要的作用。封装确定了数据结构和一组操纵数据结构的操作(方法)。数据结构只能通过一个良好定义的、经过精心参数化设计的标准界面才能存取。而界面是以尽可能少地透露内部实现细节的方式设计的。封装是软件重用的基础，它把类(或数据抽象)用为黑箱，防止用户查看其内部细节，从而最大限度地降低了没有继承关系的类之间的数据依赖，增强了类之间的独立性。

继承是用来表示类之间的相似性的机制，它简化了与以前定义的类相似的类的定义。在理想的情况下，一旦定义了可重用部件，就可以不加修改地重用。实际上，软件部件很少确切匹配新的需求。这样，考虑到新系统的特殊需要，使已有的部件适应这种需要，就是一件很必要的工作。有了继承机制，这种适应可以很容易地通过扩展、特化完成。如果没有继承机制，这样的适应只能通过对可重用模块的修改来实现。相比之下，继承机制具有很大的优越性。

面向对象程序设计环境中的类是易于重用的。从设计者的观点看，即使在设计类时没有考虑到重用的问题，由于封装和继承机制的存在，类比过程部件更容易重用。这是因为对部分匹配的类来说，可以通过继承重用共同的特征，可以通过多态机制覆盖不完全符合要求的方法，可以增加适应特殊需求的数据项和方法，这样，就可以精确地匹配新的需求。原有的已被证明的正确的模块，在修改之后，还需重新制定测试计划、设计测试用例进行单元测试。这时的成本(特别是在缺乏模块的详细设计说明书的情况下)可能比重新编制所需功能的模块更高。即使在存在一般化部件的情况下，如果不能仅仅通过参数传递实现所需的功能，还需编制胶合代码(glue code)，使一般化部件适应新的需求。因此只要是类和过程部件不能完全符合新的需求，就需要权衡修改部件和重新编制代码的成本和收益，根据实际情况做出正确的抉择。一般来说，通过继承和多态机制修改类比修改过程部件更容易一些，效果也更好一些。

5.3 XML 需求建模体系设计

在实际的项目开发过程中，效率是我们都很关心的一个问题，在我们借助 J2EE

技术开发应用系统的时候，需要经历需求、分析、设计、编码、测试等不同的步骤，在这些环节中，需求，设计和编码是需要耗用相当的时间和人力的步骤，在这些环节的工作中，我们也常常借助 Rose，PowerDesigner 等工具来简化工作。但是尽管这样，效率还是不尽如人意，比如，做需求分析的时候，一般都整理成文档，或者再转化成设计模型，便于我们理清整个系统的轮廓，然后再根据这些文档或模型来设计表，设计整个项目的架构，在开发的过程中往往也会发生一些意料不到的事情，影响整个项目的进程。针对此情况我们探索一种新的开发方式，这种方式同样也包括上面提到的软件工程的几大步骤，不过我们把这些步骤之间的联系性变得更加紧密了，前面的环节直接可以为后面的环节提供强有力的铺垫和帮助。此需求建模体系针对当前流行的开源框架 Webwork 和 Hibernate 而设计，很多的 XML 标签设计都是切合框架的特点而设计[35]。

5.3.1 概述

项目的需求信息总是通过与客户交流，一点点获取，一步步完善，但是对于很多中小项目而言，经过一个初步的需求调研获得了基本的需求信息后，由于进度的需要，不得不很快进入设计阶段，根据这些需求信息开始创建对应的数据模型，字段，类型，约束……，就这样一个系统的数据库甚至也创建好了，而项目的系统架构，由于开源框架的应用，没有过多的特性，就这样开始了一个项目的编码工作，但是随着开发工作的进一步，开发人员会发现需求工作不到位，导致一些问题，公共类，扩展接口实用性不强，业务之间的耦合性太强，以致于牵一发而动全身，导致频繁的修改。还有的项目需求做充分了，设计也可以了，但是客户的需求始终没有定型，随时都要求设计人员做调整，往往开发到一定程度的项目因为需求的变化而变得寸步维艰，比如，我们开发的时候借助 PowerDesigner 创建了系统的数据模型，同时也创建了系统的数据库，同时也录入了初始数据和测试数据，进展的很顺利，但是某天客户突然说某个模块我们需要修改，于是设计师，开始考虑变更，数据模型、业务处理类、表示方式、数据表、各种数据统统需要随之而变，于是很多的人力都投入到需求的变化带来的修改中，项目的进展也因此被延误。

基于这种情况，我们总结各种项目的特点，发现在很多的应用项目中，不管项目复杂程度差异，可以是一般的中小型应用系统，也可以是非常复杂的甚至大型分布式系统，都具备几个层次的划分，那就是表示层，业务逻辑层，数据模型

层。所以针对这种情况我们借助 XML 技术根据第四章提出的需求建模理论设计了一套需求建模体系，这个体系全部由 XML 文件来构建，包括有一系列的标签元素的定义，运用标签构建需求模型，用于控制生成文件的部分等，需求分析完成后，使用面向对象分析与设计的思想，把系统的需求模型归纳成对象模型，我们用 XML 文件来记录这些对象模型，任何业务逻辑都可以引申为对象与对象之间的关系。结合代码生成引擎，项目前期重心就在 XML 需求建模上了。

基于 XML 的需求建模体系包括建模需要用到的所有元素（elements）的定义，数据类型的定义，编译选项的定义，需求模型文件的定义等，需求体系工程结构图如下：

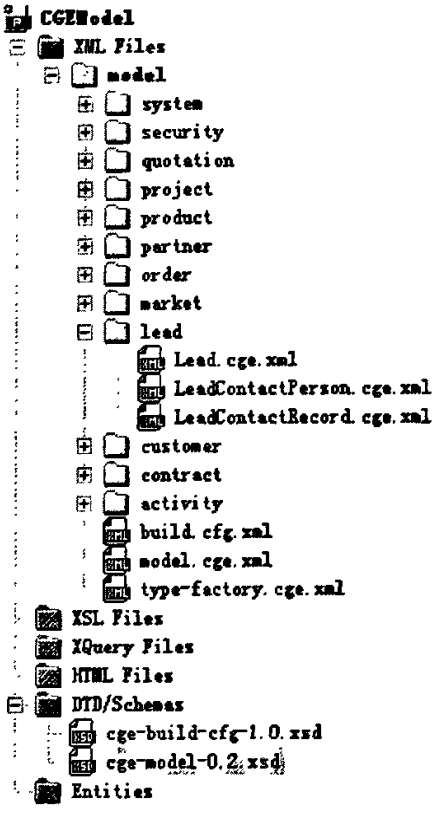

图 5- 1 XML 建模体系结构图

5.3.2 标签元素（Element）设计

整个 XML 需求建模体系首先最核心的是 Schema 文件，在此体系中我们涉及

的 Schema 文件包括两部分：第一部分就是贯穿整个体系的众多标签元素的定义，其次是一个在编译时告诉生成引擎选择性的生成代码。这一节介绍建模体系的主要标签元素的设计。

➢ meta 元素

meta 元素是整个 XML 需求建模体系中最常用的元素，作为一个通用元素，它能应用于绝大部分场合，可以嵌套使用，meta 元素的子元素包括自身（meta）和 meta-attribute 两个，有字符串（string）类型的 name 和 namespace，name 是 meta 的名称，namespace 指明 meta 元素的作用域。关于 meta 元素的定义请看下图：

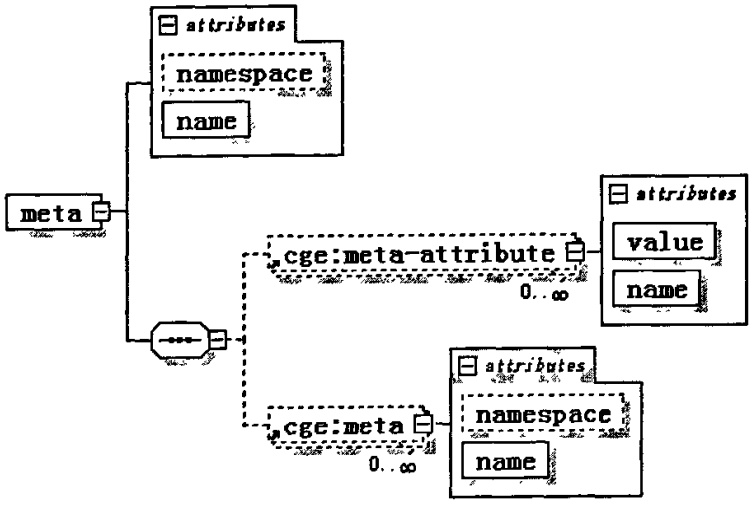

图 5- 2 meta 元素

➢ simple-type 元素

simple-type 元素是为了方便用户定义在需求建模过程中遇到的特殊类型，它不同于 XML Schema 中提到的 simpleType,它完全是用户自行设计定义的标签元素。在我们的需求建模过程中会遇到很多的对象属性，无法借用一般的基本类型，如 string、int、long、float 等就能准确的定义，因此我们自定义了一个专门的数据类型来完成这项工作。simple-type 元素的元素结构和属性如下图示：

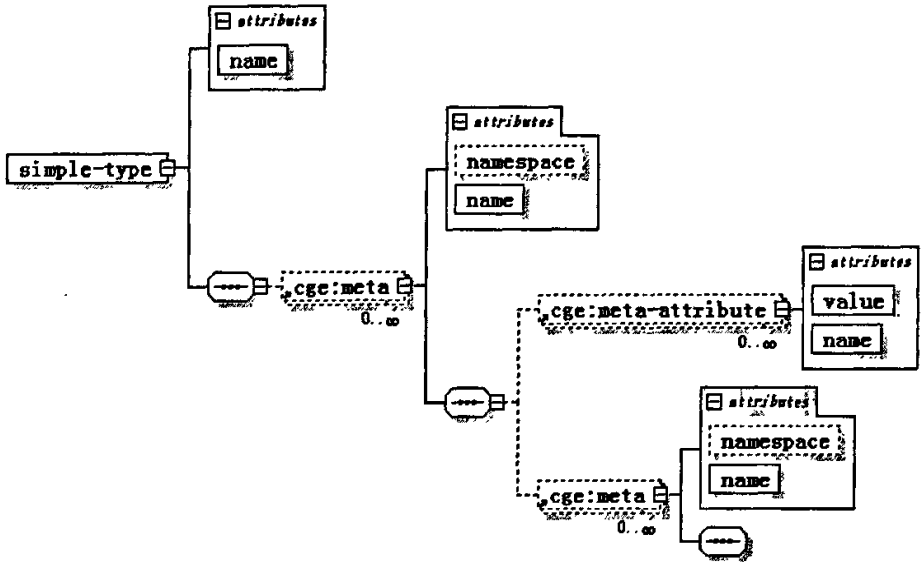

图 5- 3 simple-type 元素

➢ unary-collection 元素和 binary-collection 元素

这两种元素都是用来描述一个数据类型集合的在使用上基本上相似，其标签元素定义如下：

```
<element name="unary-collection">
    <complexType>
        <sequence>
            <element ref="cge:meta" minOccurs="0"
            maxOccurs="unbounded"/>
            <element ref="cge:collection-element"/>
        </sequence>
        <attribute name="name" type="string" default="set"/>
    </complexType>
</element>
<element name="binary-collection">
    <complexType>
        <sequence>
            <element ref="cge:meta" minOccurs="0"
```

```
                maxOccurs="unbounded"/>
                <element ref="cge:collection-key"/>
                <element ref="cge:collection-value"/>
            </sequence>
            <attribute name="name" type="string" default="map"/>
        </complexType>
    </element>
```

从定义代码中我们可以看到unary-collection和binary-collection都用来描述集合类型，不同的是前者应用于数据建模中的一对多，多对多等场合，此时的集合类型是 set 类型。而后者应用于存在 key-value 映射的场合，集合类型是 map 类型。

➢ type-factory 元素

type-factory 元素是设计用于将我们自定义的类型映射到 java 类型和 Hibernate 数据类型，其标签元素定义如下图。

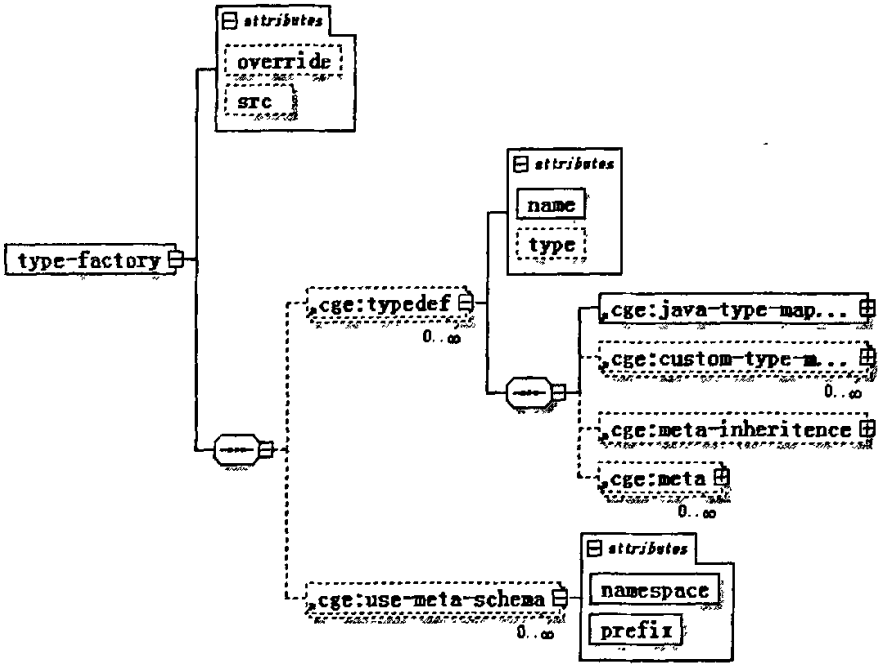

图 5- 4 type-factory 元素

type-factory 元素包含了两个子元素 typedef 元素和 use-meta-schema 元素，前者主要是完成元素类型定义的，同时此元素支持 meta 元素的继承功能；后者提示是否使用名空间。type-factory 元素包括允许类型覆盖的 override 属性和相关域空间的链接 src 属性。

➢ interface-method 元素

interface-method 元素用于定义接口内部方法，这里需求设计人员（或者系统的参与人员）经过对需求的细化，设计系统的一些公共接口和基类，这些接口或者类可以借助 XML 建模文件表达出来，比如使用 interface-method 元素可以定义公共接口中涉及的一些接口方法。interface-method 元素组成结构参看下图 5-5：

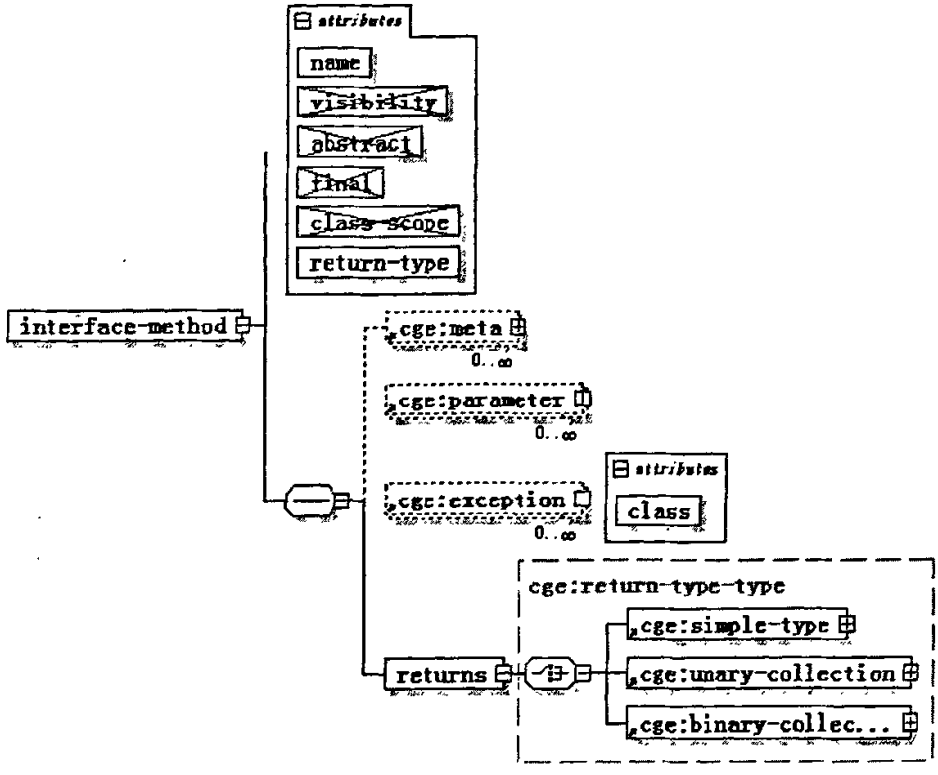

图 5- 5 interface-method 元素

可见，interface-method 元素允许设计人员定义方法的名称返回类型，还有相关的参数和异常情况。

➢ methods 元素

methods 元素是为了在类中定义一系列方法而设计的。它也是用于在需求建模阶段完成一部分设计工作而定义的标签元素，借助 class 元素，interface 元素，interface-methods 元素，extends 元素，implements 元素，methods 元素，method 元素等可以实现基本上所有的类和接口设计。而且借助这些元素几乎所有的对象模型都可以完整的定义好，在代码生成的时候就能自动的获得整个项目的各种类。下图是 methods 元素的定义：

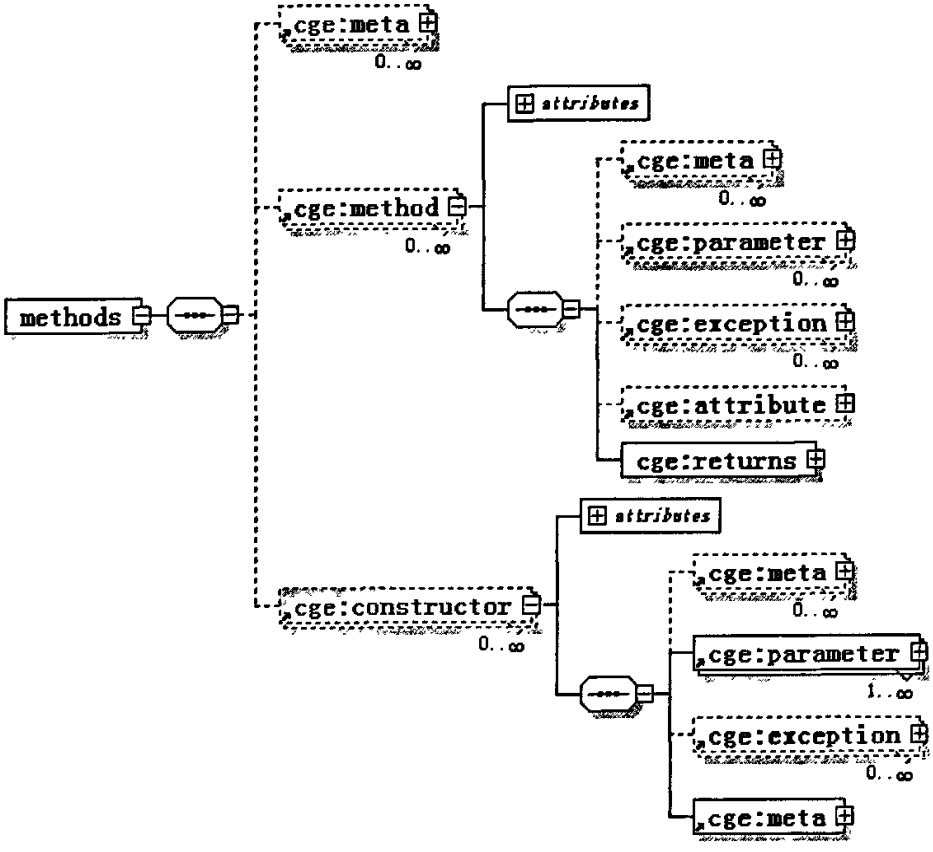

图 5- 6 methods 元素

methods 元素包括子元素：method 元素，meta 元素和 constructor 元素。method 元素主要用于每个具体方法的定义，constructor 元素用于定义某个类中的构造器。

➢ interface 元素和 class 元素

interface 元素和 class 元素都是设计用于在需求建模阶段定义系统的接口和类的元素，面向对象分析和设计方法把需求转化成一个个的对象，而对象对应的就

是一个个的类，对象的公共信息可以通过 interface、class 元素定义。关于 interface 元素定义如下图 5-7。

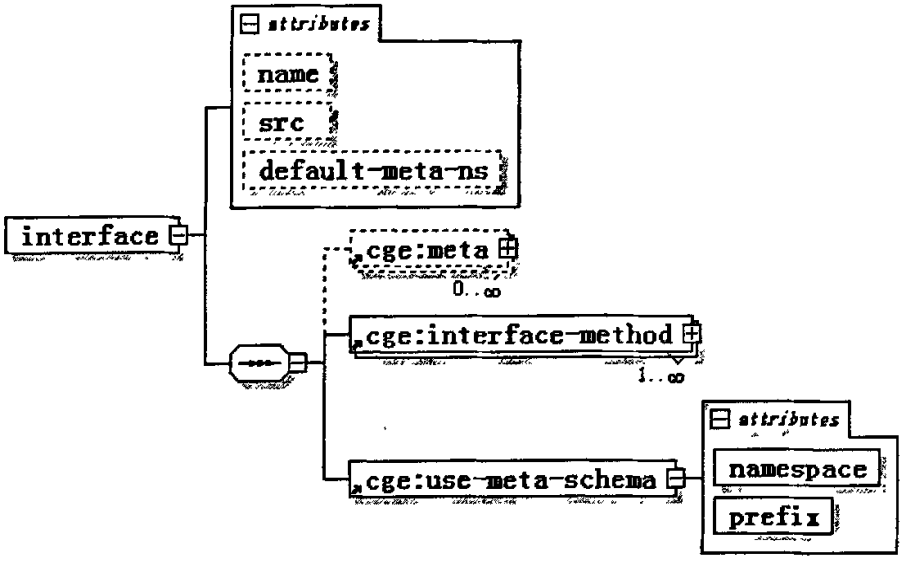

图 5- 7 interface 元素

➢ model 元素及其他元素

model 元素对应的是一个项目。在以这个元素为根元素的 xml 文件里，我们习惯把它命名为 model-cge.xml，包含了一个项目的数据库连接信息，数据访问层基本代码设计，所有数据类型，再就是整个项目的所有需求模型。此外还有很多的标签元素是构成我们整个需求体系的不可缺少部分，比如 package 元素，设计用于定义不同的包的情况；实现继承和扩展的标签元素 extends 元素和 implements 元素，通过他们各个类各个接口之间就有了沟通的桥梁；还有 inner-class 元素，exception 元素，visibility-enum 元素，java-type-mapping 元素等等。

5.3.3 需求建模

上面已经设计了一套标签元素，下面的工作就是运用这些标签元素进行需求模型的构建。需求分析是一项很复杂的工作，没有经验的软件人员一般情况下很难做到快、准、省的程度，往往都是多次反复，不过即使运用这个新的开发模式，也同样要求需求分析人员能够准确的把握来自项目的各中需求，特别是能直接转

化为对象的信息。在开发基于 B/S 架构的应用系统（我们这里提到的应用系统都是基于 J2EE 平台）时，除了获取详尽的需求信息外，通常都还有一系列需要解决的共同问题，比如数据信息的验证问题，系统的配置信息，不同数据类型转换问题，针对不同的语言的国际化问题，公共类接口的设计，常规的增、查、改、删操作（CRUD），对象模型到数据库表的映射问题，javaBeans 等等。因为建模本身是从需求信息的一个升华，当时的考虑如果在我们的 XML 工程中附带过多的信息会使得建模变得复杂臃肿，有些问题，比如国际化，映射关系，转换器等我们设置在生成引擎当中。以下我们结合上述某些问题介绍如何进行需求建模。

➢ 关于 meta-schema

在建模体系中，我们会涉及到系统级，架构级、框架级，应用级等不同级别的标签应用场合，所以，我们引入 meta-schema 来区分不同的级别，如下面代码定义所示：

```
<!-- meta schema declarations:定义需要使用的MetaSchema-->
<cge:use-meta-schema namespace="http://www.haichusoft.com/meta/hsml/core"
prefix="c"/>
<cge:use-meta-schema namespace="http://www.haichusoft.com/meta/hbm-meta.xml"
prefix="hbm"/>
<cge:use-meta-schema
namespace="http://www.haichusoft.com/meta/webwork-meta.xml" prefix="ww"/>
<cge:use-meta-schema    namespace="http://www.haichusoft.com/meta/crm-meta.xml"
prefix="crm"/>
```

prefix c 代表系统核心架构设计方面的应用，prefix hbm 代表的是数据模型层，具体来讲是基于 Hibernate 的数据模型及关系映射应用，prefix ww 代表的是业务逻辑层，具体来讲是基于 Webwork2 的业务逻辑及表示层的相关方面的应用，而 prefix crm 指本系统是一个客户关系管理方面的应用，这是切合项目本身应用方面的标签，不同的应用这个标签可以不同，比如如果是 erp 方面的应用，可以相应的把 crm 改为 erp。而系统级的 prefix 则是 cge 这是我们约定的一个名空间。

➢ 数据模型层的设定

每个软件项目总有一个数据库作为它的数据支撑，而软件系统必然少不了对

数据的操作，本代码生成引擎使用基于开源框架 Hibernate 的数据访问中间层来实现这个功能。对于相关的参数设定我们又是如何通过 XML 建模工程文件设定呢？如下代码所示，详细的指出了 Hibernate 数据库连接及相关策略的情况。

```
<!-- Hibernate CGE options : Hibernate CGE选项-->
<!--hibernate session factory : Hibernate Session Factory属性-->
<cge:meta name="hbm:session-factory">
    <cge:meta-attribute name="hibernate.connection.driver_class"
    value="com.mysql.jdbc.Driver"/>
    <cge:meta-attribute name="hibernate.dialect"
    value="org.hibernate.dialect.MySQLDialect"/>
    <cge:meta-attribute name="hibernate.connection.url"
    value="jdbc:mysql://192.168.1.102/temp?useUnicode=true&amp;characterEnc
    oding=utf8"/>
     <cge:meta-attribute name="hibernate.connection.username" value="crm"/>
     <cge:meta-attribute name="hibernate.connection.password" value="crm"/>
     <cge:meta-attribute name="hibernate.show_sql" value="true"/>
</cge:meta>
```

上面代码设定了 Hibernate 连接 MySQL 数据库的会话工厂（session-factory）的相关参数，比如连接驱动类为"com.mysql.jdbc.Driver"，使用的数据库方言是"org.hibernate.dialect.MySQLDialect"，此外还有登陆数据库的地址，帐号密码信息及设定了调试时显示 sql 语句。

➢ 公共 DAO 接口和实现类设定

为了使应用开发更加方便高效，我们常常都需要设计一些公共的接口和类提供给开发人员使用，在 XML 需求建模的时候我们同样也设计了用于数据访问的最基本 DAO 相关类和接口，在生成代码时，引擎会根据这些设定生成对应的接口和实现类。下面代码展示了具体的设定情况：

```
<cge:meta name="c:param">
    <cge:meta-attribute name="GenericDAOInterfaceClass"
    value="com.haichu.crm.persistence.GenericDAO"/>
</cge:meta>
```

```
<cge:meta name="c:param">
    <cge:meta-attribute name="GenericDAOHibernateImplClass"
    value="com.haichu.crm.persistence.GenericHibernateDAO"/>
</cge:meta>
```

在这里定义了一个基本的 DAO 接口和一个 DAO 实现类，进一步的扩充，我们在代码生成引擎里作了特殊设计，通过它即可产生几乎所有的 DAO 类和接口满足整个项目使用的需要。

➢ 相关操作设定

每个对象模型对应着每种不同的对象，对象作为信息的载体必须允许被查找被排序，被不同角色权限的人操作，这时候就涉及到很多问题，比如排序，访问权限，地域敏感性，是否允许搜索等，这些信息的设定如下代码所示。名为 user-owned-object 的标签指明此对象由哪个操作用户创建并管理，当遇到此对象时对应有操作权限的用户从哪里去匹配；名为 region-sensitive 的标签指明这个对象对于不同的地域是否有区别，比如很多跨国公司的产品在不同的国家和地区是区别对待的，根据 region 来区分产品的差异；enable-search 标签是允许查找此对象；而 default-order 指明此对象的初始排序方式。

```
<cge:meta name="crm:user-owned-object">
    <cge:meta-attribute name="check-owner-at" value="user"/>
</cge:meta>
<cge:meta name="crm:region-sensitive">
    <cge:meta-attribute name="check-region-at" value="region"/>
</cge:meta>
<!--enable search : 允许搜索-->
<cge:meta name="crm:enable-search"/>
<!--default order by : 默认排序-->
<cge:meta name="crm:default-orders">
    <cge:meta name="crm:order">
        <cge:meta-attribute name="order-by" value="id"/>
        <cge:meta-attribute name="type" value="desc"/>
    </cge:meta>
```

```
        </cge:meta>
```

➢ 对象属性设定

对象是由很多不同的属性及对属性的操作组成，在对象的创建、修改或者查找时，都需要根据不同的属性来进行，但是具体每一个属性又如何来设定呢，不同的属性有不同的取值类型及约束，又怎么映射到数据库呢？下面代码详细的演示了这些问题。

```
<!--name:公司名称-->
<cge:attribute name="companyName">
    <cge:meta name="crm:enable-search"/>
    <cge:meta name="ww:validation">
        <cge:meta name="ww:requiredstring"/>
        <cge:meta name="ww:stringlength">
            <cge:meta-attribute name="minLength" value="2"/>
            <cge:meta-attribute name="maxLength" value="50"/>
        </cge:meta>
    </cge:meta>
    <cge:simple-type name="string">
        <cge:meta name="hbm:property">
            <cge:meta-attribute name="not-null" value="true"/>
            <cge:meta name="hbm:column">
                <cge:meta-attribute name="name" value="company_name"/>
                <cge:meta-attribute name="length" value="50"/>
            </cge:meta>
        </cge:meta>
    </cge:simple-type>
</cge:attribute>
```

上面代码是以公司名称为例来介绍如何设定属性的。enable-search 标签允许用户根据公司名称来查找此对象，validation 标签设定了客户端在创建或者修改此属性时输入内容的验证信息，比如 requiredstring 标签指明此属性不能为空，stringlength 标签指明此属性的长度范围是2-50个字符；接下来通过一个 simple-type 元素设定了一个字符串类型的属性，接下来有 property 标签指明这是映射到数据库

的字段，此字段不能为空，对应的字段名为 company_name，最大长度为 50 字符。代码生成引擎执行后会生成一系列的验证文件（xml 文件）包括客户端验证和服务器端验证，这些信息就是根据这里的信息而生成。

➢ 逻辑主键的设定

主键是用来区分表中每一条记录的参考信息，但是在实际的开发过程中，单单一个字段无法真正区分不同的记录，因此需要使用逻辑主键，相当于联合主键来作为不同记录的区分标志。关于逻辑主键的设定，参看如下代码，下面代码以用户名称和用户所在部门来作为区分合作伙伴的逻辑主键。

```
<!--natural id<名称,部门>-->
<cge:meta name="hbm:natural-id">
    <cge:meta name="hbm:property-ref">
        <cge:meta-attribute name="name" value="name"/>
    </cge:meta>
    <cge:meta name="hbm:property-ref">
        <cge:meta-attribute name="name" value="department"/>
    </cge:meta>
</cge:meta>
```

➢ 泛化关系设定

既然是面向对象的分析与设计，通过需求分析，项目的信息升华到具体的每个对象及对象间关系，而对象之间同样也是有泛化的关系，此建模体系允许设计人员在 XML 对象建模过程中将存在泛化关系的对象表述出来，通过代码生成引擎，即可生成存在泛化关系的类，并且同样可以生成对应的 sql 语句在数据库中创建存在泛化关系的类对应的表，比如具有继承关系的两个类，同时在 Hibernate 映射文件及 javaBean 生成过程中自动的反映出这种继承关系。关于泛化关系的设定示例参阅附录 B（Partner.cge.xml 及 ContactTarget.cge.xml）。

➢ 对应关系设定

对象与对象之间存在着复杂的关系，通过这些关系才使不同的对象能构成一个有机统一的系统。对象间关系最基本的就是对应关系，有一对一关系，一对多关系，多对一关系，多对多关系，而且有的对应关系必须允许互逆，而有的又不

能。如何在 XML 对象建模的时候表达这种不同的对应关系也是我们建模体系设计的一个重点。下面以订单和订单对应的产品为例说明一对多的关系，一个订单对应一个或者多个产品，而且从订单能查到包括哪些产品，反过来，从某一个产品也必须能查出他属于哪个订单，这就是反转关系。他们之间的关系在数据库中表现为通过一中间表来记录。由下面代码可以知道产品对应是一个集合，这个集合的数据来自中间表 typed_product_order_items，他们与对应 order_id 的订单关联，同时生成代码 TypedProductOrderItem.java 这是一个对应于中间表的 javaBean。对订单增删产品的操作必须注意 inverse 属性，也就是说订单添加了一项新产品，同时产品对应的订单项也必须同步得到更新。

```
<!--productOrderItems : 包含的产品订单-->
<cge:attribute name="typedProductOrderItems">
    <cge:unary-collection name="set">
        <cge:meta name="hbm:set">
          <cge:meta-attribute name="table"
      value="typed_product_order_items"/>
            <cge:meta-attribute name="inverse" value="true"/>
            <cge:meta-attribute name="cascade" value="all"/>
        </cge:meta>
        <cge:meta name="hbm:key">
            <cge:meta-attribute name="column" value="order_id"/>
        </cge:meta>
        <cge:collection-element>
            <cge:simple-type name="TypedProductOrderItem">
                <cge:meta name="hbm:one-to-many"/>
            </cge:simple-type>
        </cge:collection-element>
    </cge:unary-collection>
</cge:attribute>
```

下面以项目和项目合伙人为例来介绍对象间多对多关系。一个项目可以有一个或者多个合伙人，同时一个合伙人可以投资一个或者多个项目，这就是项目和合伙人之间的多对多的关系。具体 XML 建模设定如下。在数据库中保持这种对应关系同样得借助中间表来实现，下面定义通过代码生成引擎解析会产生一个名为

project_partners 的中间表，表中记录的几位项目主键与合伙人主键，在这里都是使用 id 作为主键，同样也会产生一个对应中间表的 javaBean。

```
<!--partners:合作伙伴-->
<cge:attribute name="partners">
    <cge:unary-collection name="set">
        <cge:meta name="hbm:set">
            <cge:meta-attribute name="table" value="project_partners"/>
        </cge:meta>
        <cge:meta name="hbm:key">
            <cge:meta-attribute name="column" value="project_id"/>
        </cge:meta>
        <cge:collection-element>
            <cge:simple-type name="Partner">
                <cge:meta name="hbm:many-to-many">
                    <cge:meta-attribute name="column" value="partner_id"/>
                </cge:meta>
            </cge:simple-type>
        </cge:collection-element>
    </cge:unary-collection>
</cge:attribute>
```

关于使用 XML 建模体系进行需求建模的基本要点都进行了说明，如希望了解更详细情况请参阅附录 A。

5.4 代码生成引擎

前面一节介绍了作为代码生成引擎体系中基于 XML 的需求建模体系，但是只有它还只是把项目中的需求信息，对项目的一些设计思想反映到 XML 文档里，这还远远达不到项目开发应用的目的。下面的工作就是如何把 XML 文档形式的需求模型，设计理念转化成为项目实际可以应用的配置文件、类接口文件以及数据库文件。这里我们需要一个 XML 文件解析器，这个解析器根据 XML 需求模型生成对应的项目文件。本节将从 XML 解析器的分析、用例、处理流程到架构设计与实现等方面进行详细的阐述。

5.4.1 系统分析

目前国内有很多应用软件是遵循J2EE(JAVA 2 Enterprise Edition)规范开发的，而且基于 J2EE 平台的应用现在呈上升趋势。并且大都是基于关系数据库系统的B/S 结构的 WEB 应用系统。在开发 J2EE 应用时，我们一般把它纵向分为以下几个层，最下层为 OS，JVM，network 层，它们负责系统的底层操作和网络数据的传输，一般我们开发人员不用关心运行在什么具体操作系统上，什么样的网络环境下。它的上层为 J2EE 服务层，一般由 J2EE 服务器(如 WebSphere、 WebLogic 等)提供各种基础服务，如事务的管理(JTS)，命名目录服务(JNDI，负载均衡(Load Balancing)，容错(failover)，安全(security)等。其次是通用业务层，它一般完成与具体业务无关的基本操作，包括如基础的框架((Framework) commons 组件，如通用的数据库处理组件，系统错误处理组件，字符处理和数值处理组件，日志(log)处理，数据转化和编码维护等核心层。最上层才是我们的具体业务逻辑模块，它完成具体的业务逻辑[36]。如下图 5-1 所示：

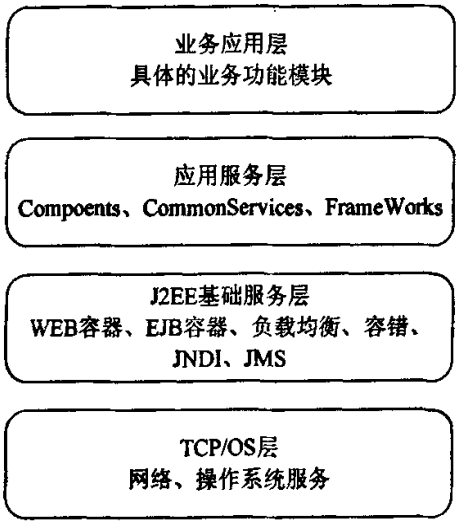

图 5- 8 系统层次图

应用软件的目的是解决某一领域的业务问题，然而在开发过程中，除了业务需求要关注，技术方面也会有大量的问题。在软件开发维护过程中任何一个信息系统中都存在大量的基础数据对象维护，大部分只是简单的维护功能，很少有业务逻辑，但是由于数量很多，既便是复制、粘贴也需要较多工作量去调试、维护

程序代码;而一个复杂的软件系统中有几百上千个业务对象。如何帮助开发人员从烦琐的编码的技术细节中解脱出来，减少开发的工作量，把主要精力用于关注业务问题，并提高开发效率和质量。

在J2EE平台上，Java作为面向对象的编程语言，虽然有了面向对象数据库，但是还相当不成熟，而且一些企业老的应用都跑在关系数据库上。种种的因素都可能导致你同时选择了面向对象技术和关系型数据库技术。这种情形存在一天，我们就需要考虑对象到表的映射问题。面向对象设计基于如耦合、聚合、封装等理论，而关系模型基于数学原理。不同的理论基础导致了不同的优缺点。对象模型侧重于使用包含数据和行为的对象来构建应用程序；关系模型则主要针对于数据的存储。面向对象和关系模型可以说是完全不同的编程模式。当对象需要保存在关系型数据库中的时候，我们需要的是对象到表的映射。不但是数据映射的问题，还有面向对象编程中的许多概念需要映射到关系型的表结构中，包括如下几种：

- 聚合(Aggregation)
- 继承(inheritance)和多态(polymorphism)
- 类间的关联(association)
- 比SQL数据类型更加灵活的数据类型

关于对象关系映射，有很多优秀的技术、方法和实现工具，本文后面的章节也会介绍一种实现方法。

开发基于关系数据库的企业应用系统，一旦得到数据结构模型。很多针对数据库的访问的代码是有固定规律的，比如说单表的增删改，主从表结构的增删改。如果在前后台的实现上，开发组织有自己的框架，组件等等，就可以生成大部分的java代码，jsp代码，sql脚本和配置文件等。可以加快开发速度，提高代码质量。由业务对象的模型到数据库表结构，有数据结构到面向对象的语言编写的代码，一正一反的过程，完成了应用系统的一个普通CRUD功能的开发。

5.4.2 系统用例

本代码生成引擎是基于 MVC 开源框架 Webwork 及持久层框架 Hibernate 之上开发的，目的就是使开发人员能够更快的应用开源框架进行项目开发。使用该工具开发一个业务系统的基本过程如下：

- 首先分析设计人员通过 UML 建模工具分析、设计得到业务对象的模型，接着将业务对象模型转化成为 XML 需求模型；
- 然后通过代码生成引擎解析 XML 需求模型文件，得到在 J2EE 企业应用开源框架之上的系统框架、配置文件及业务对象等运行系统核心代码。
- 最后创建新的应用项目工程，提取代码生成引擎生成的核心代码，将这些代码部署到新的工程中。

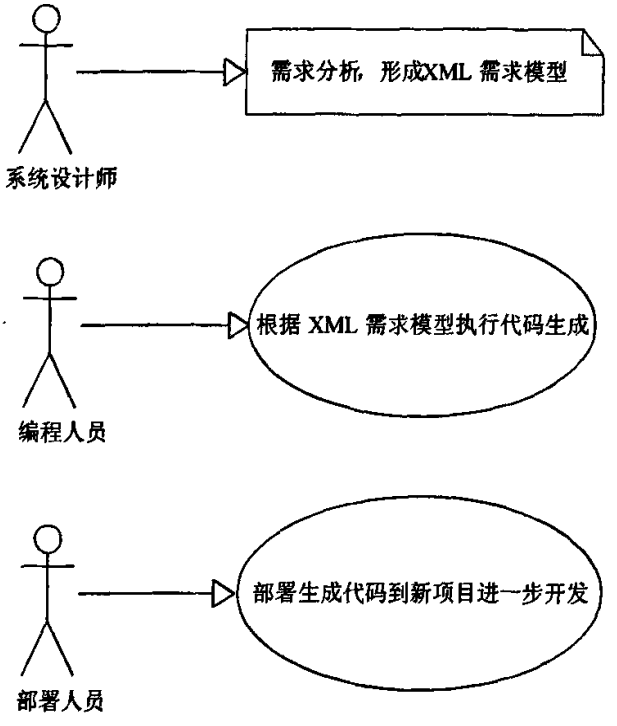

图 5- 9 系统用例图

从软件工程的角度来看，元数据主要是减轻了编码的工作量，模式化开发能

更好的避免人为失误，提高架构设计质量，加快开发速度。采用 XML 需求模型和代码生成引擎的开发模式在整个项目开发过程中的作用如下：

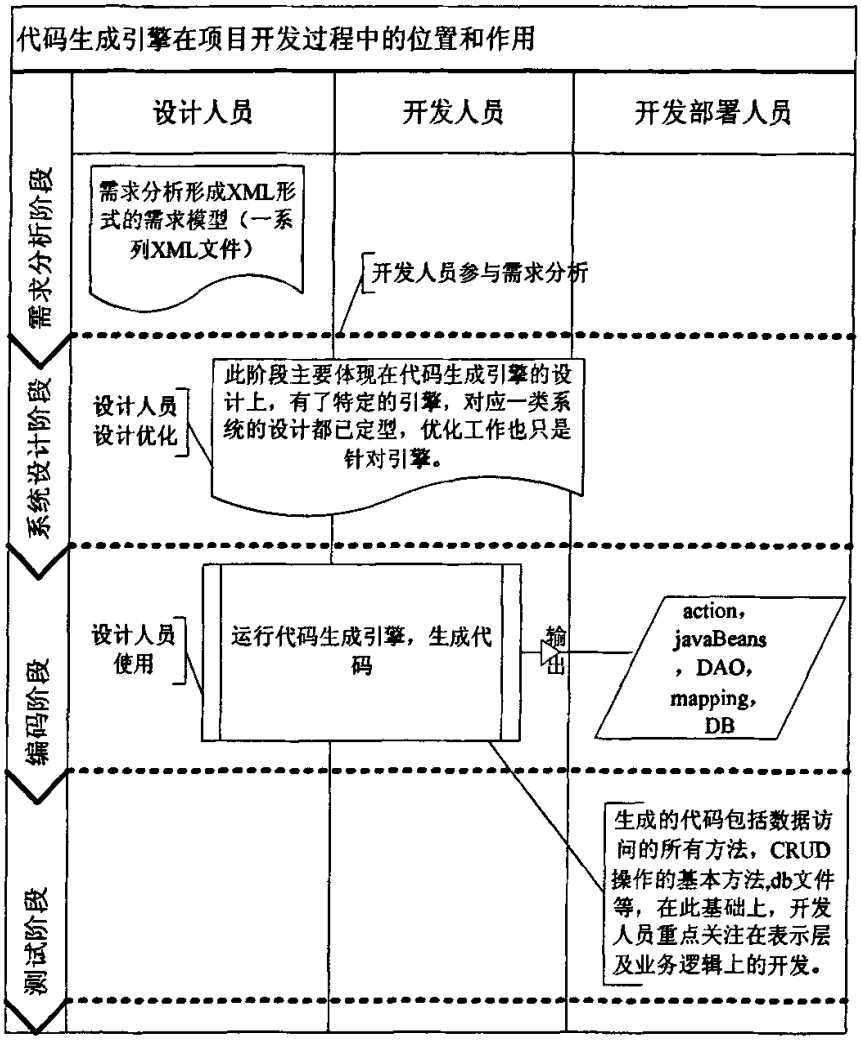

图 5- 10 代码生成引擎作用图

由上图分析，本文的代码生成器，定位在需求阶段和设计编码阶段，目的是能够完成基于需求模型的 J2EE 应用的设计成果到实现代码的快速转化，并且能够实现由非 J2EE 的基于 XML 需求建模的应用到先进的 J2EE 的平台的快速升级。

5.4.3 系统流程

使用 UML 序列图分析了，利用由业务对象的模型转化而来的数据结构描述文件，生成业务对象的运行代码的流程。由遗留的原有关系数据库生成业务对象的运行代码与此有相似的过程。

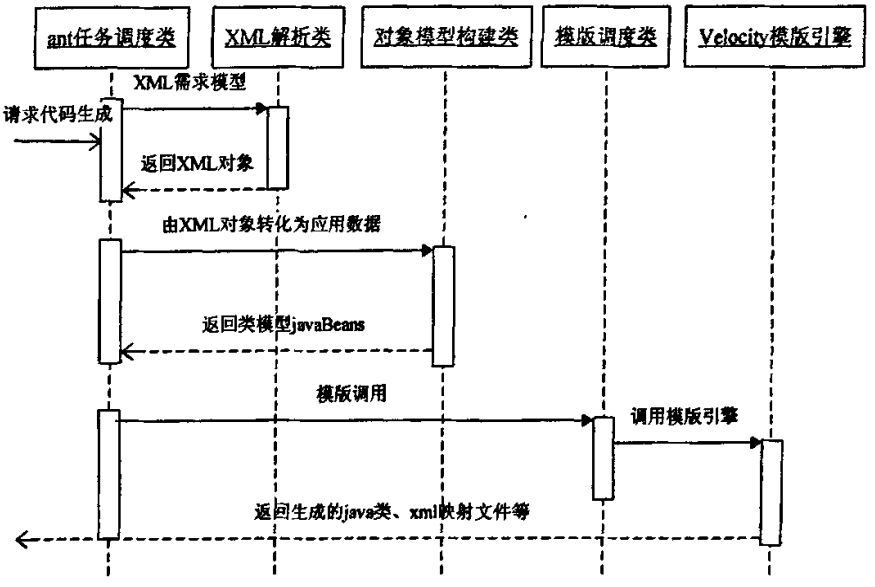

图 5- 11 系统工作序列图

5.4.4 系统架构设计

按照“架构为基础，模式为指导”的原则，设计本系统。采用 MVC 架构模式，各层次的实现如下：

- View 层：各种 Velocity 模板(vm 格式的文件)是 view 层的实现，与框架的接口，在这一层模板的代码中体现；
- Control 层：Ant 的任务的调用者类 GeneratorTask 和模板的控制类是任务控制层实现；
- Model 层：业务逻辑实现者是 XmlparseToApp 和 MetaToApp，读取解析 xml

和元数据读取的类属于持久层的实现，数据模型是作为数据载体存在。

整个系统设计由四大部分组成：核心部分主要包括体系中各元素对应类的处理，异常处理，模版调度处理及整个生成引擎的运行入口；业务层基于 Webwork 框架的代码生成，主要包括 CRUD action 代码的生成，validation 代码的生成，config 代码的生成，conversion 代码的生成，部分 view 层代码生成及代码生成策略的优化；数据模型层是基于 Hibernate 数据操作的一系列代码生成处理，包括 DAO 接口，DAO 实现类，POJO，映射关系等的生成处理；第四部分主要是系统使用的代码生成模版，包括业务层和数据层代码的生成模版。包结构类图如图 5-示。

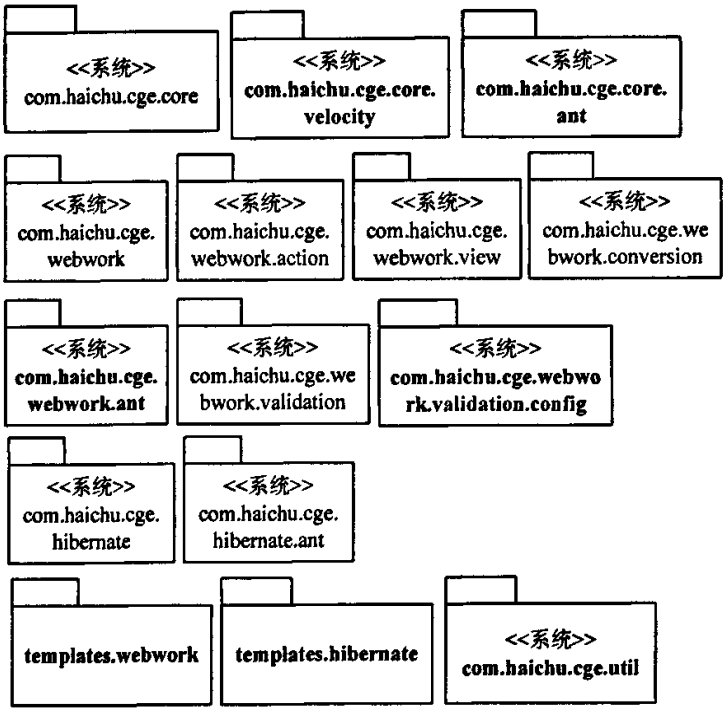

图 5- 12 系统包结构类图

正因为基于此架构模式设计，所以在其他两层不动的情况下，替换掉 View 层的模板文件，则可以实现基于其他框架的代码生成。这也体现了此架构模式的优势。

5.4.5 代码模板方案

上一节中提到系统使用的代码生成模版，用于提取一个完整的业务用例的代码，包括：Java，xml 配置文件等等。

Java 文件包含表示层，控制层，服务层，和持久层，各层次的代码，全部的增、删、改、查方法。

Xml 文件包含 Webwork 和 Hibernate 的专有配置文件，对象关系映射文件。

以上面提到的，从三个框架整合的应用抽取的代码为基础来设计代码模板，包括代码目录结构，样式，包引用和日志等，大量模式应用都能在此处得到发掘和展现。Velocity 是一个基于 java 的模板引擎(template engine)。它允许任何人仅仅简单的使用模板语言(template language)来引用由 Java 代码定义的对象。在这里，设计了 javabean 模板，jsp 模板，js 模板，xml 配置文件模板，sql 模板等，基本上所有的 CURD 操作都能在此实现。

写模板的过程实际是一个逆向过程，就是你要知道你生成的代码是什么样的，然后借助一下数据库对象模型，可以保证类似的代码只写一遍，这就是模板。写模板的过程跟写实际代码的过程没什么两样，除了利用经常用到的变量 table。

如果熟悉 velocity 语言的话，结合本代码生成器的代码模板样例 templates/hibernate/dao-interface.vm 来介绍一下模板。(vm 模板注释是以“##”开头的行)

```
package $daoPackage;
import $genericDAO;
##import Entity
import $class.qualifiedName;
##import naturalId
#set($naturalIds = $daoHelper.getNaturalId($class))
//set naturalIds by set command
#foreach ($attr in $naturalIds)
```

```
#if(!$attr.type.javaClassName.startsWith("java.lang")
&& !$attr.type.javaClassName.equals($class.qualifiedName))
import $attr.type.javaClassName;
#end
#end
##import collection element type
#*
#foreach ($symbol in
$daoHelper.getClassCollectionTypeAttributeImports($class))
#if (!$symbol.equals($class.qualifiedName))
import $symbol;
#end
#end
*#
#set($id = $daoHelper.getId($class))
/**
 * DAO interface for entity $class.qualifiedName
 * @author: Generated by CGE (http://www.haichusoft.com/cge) version 0.02
 * @createDate: $buildDate
 */
public interface ${class.unqualifiedName}DAO extends
GenericDAO<$class.unqualifiedName, $id.type.javaTypeDeclaration>{
#if ($naturalIds.size() != 0)
    //retrive $class.unqualifiedName by natural
    //id (#foreach ($attr in $naturalIds)$attr.name
    //#if($velocityCount != $naturalIds.size()), #end#end)
    public $class.unqualifiedName getByNaturalId(
    #foreach ($attr in $naturalIds)$attr.type.javaTypeDeclaration
```

```
    $attr.name#if($velocityCount != $naturalIds.size()), #end#end);

  #end

  #*

  #foreach ($attr in $class.attributes)

  #if (!$attr.type.simpleType)

    //add and remove methods for collection type
${class.unqualifiedName}.$attr.name

    public void $daoHelper.generateAddCollectionMethodSignature($attr);

    public void $daoHelper.generateRemoveCollectionMethodSignature($attr);

  #end

  #end*#

    //TODO: add other method for business logic related to
$class.unqualifiedName

  }
```

可见，里面的这个 table 核心变量是 xml 数据模型里的 table 变量，而这个变量是从 xml 模型文件接收的，现在一切都很清晰。要扩展模板的话，先看看现有模板的组织，然后明白你要生成的代码，写模板就是一件很容易的事了，现有的模板很容易看懂，并且还有充分的注释。

在企业级开发中代码数量非常庞大，目录结构也很复杂，因此在模板中添加了控制器，分目录，分层次控制代码的生成。对生成后的代码包的目录结构也进行了规划，先按业务模块划分，然后按各层次，划分包路径。层次和数据模型(AppData)中的各层次一一对应。

5.4.6 系统实现

代码生成引擎的具体实现是在前面分析设计的基础上的代码实现，各个功能都有特定的类来完成，引擎主要类及实现的功能如下：

CodeGenerationTask.java：这是使用 ant 执行代码生成引擎的启动类，所有的服务都通过这个类来启动；

DefaultVelocityBasedCodeGenerationStrategy.java：基于默认的模版代码生成策

略类，在没有指明特别模版的情况下系统根据此默认的模版设置进行代码生成；

VelocityBuilder.java：模版文件创建类；

DAOGenerator.java：DAO 文件生成类；

DefaultDAOFactoryHibernateImplGenerationStrategy.java：Hibernate DAO 工厂实现类的默认生成策略类；

DefaultDAOFactoryInterfaceGenerationStrategy.java：默认的 DAO 工厂接口类的生成策略类；

DefaultDAOHibernateImplGenerationStrategy.java 默认的 Hibernate DAO 实现类生成策略；

DefaultHbmGenerationStrategy.java：默认 Hibernate 业务对象关系映射文件生成策略类；

DefaultPOJOGenerationStrategy.java：默认的持久化 java 对象生成策略类；

HbmGenerator.java：Hibernate 业务对象关系映射文件生成类；

POJOGenerator.java：持久化对象生成类；

CRUDActionGenerator.java：CRUD 操作（类）生成类；

DefaultAddEntityRelationshipActionGenerationStrategy.java：新建实体关系操作类的默认生成策略类；

DefaultCRUDActionGenerationStrategy.java：CRUD 操作类的默认生成策略类；

DefaultResourceFileGenerationStrategy.java：系统资源文件默认生成策略类；

XWorkCfgGenerator.java：Xwork 配置文件生成类；

DefaultEntityConverterGenerationStrategy.java 默认实体转换文件生成策略类；

DefaultXWorkConversionCfgGenerationStrategy.java 默认 Xwork 配置信息转化文件生成策略类；

ValidationXmlGenerator.java：系统 XML 验证文件生成类；

上面诸类体现了整个代码生成引擎的基本功能架构，这些类与其他辅助类共同实现了整个代码生成引擎系统的完整功能。

5.5 代码生成引擎在客户关系管理系统的应用

代码生成引擎体系在实际项目开发中的应用大大改善了项目开发的速度和质量，但也必须按照一定的步骤来进行，总结起来就是下面几个环节：

- 需求分析，规划分层架构，设计公共类接口，划分清晰的模块。
- 设计编写 XML 需求建模文件。
- 创建模板文件。
- 执行代码生成引擎生成代码。
- 部署代码，进行二次开发并调试。

客户关系是企业保持和发展客户资源的基本手段，客户关系营销水平体现企业营销渠道的基本素质和营销能力，研究客户关系，对于企业提高客户关系水平，增强市场竞争力，保持健康稳定发展，意义深远。

本项目是一款面向中小企业的客户关系管理系统，经过深入的需求调研，我们使用模块化设计，充分的发掘需求，力求全面的模块设计，面向不同的行业，不同的需求，可以让客户根据自身的需要来选择不同的模块组装自己的客户关系管理系统。而且本项目是使用优秀的开源框架 Webwork2 和 Hibernate3 作为主要技术，同时结合表示层技术 Freemarker 进行开发的。我们的代码生成引擎是也是在 Webwork2 和 Hibernate3 的基础上实现的，同时因为将要面对的项目基本上都是基于 MVC 模式。现在使用范围很广的开源框架，使用 Ant, Log4J，Xdoclet，Velocity，JUnit，CVS 等技术和开源工具，我们可以依靠开源软件的功能降低开发成本。借助代码生成引擎加快避免很多重复的问题，专注业务层，提高开发速度。

代码生成引擎根据适合 J2EE 架构的经过实践考验的设计模式来生成代码。由于模型和模式框定了你的应用程序的框架和实现，可以确保你的产品的各个模块都是同等质量的代码，开发人员水平参差不齐的问题也在一定程度上解决了。

代码生成引擎由两大部分构成，一部分是基于 XML 的需求建模体系，另一部分是代码生成引擎，也可以说是一个 XML 文档的智能解析程序。整个代码生成系统是设计在 ant 环境下运行的。所有的需求模型及详细的其他设计文件通通都放在一个 XML 工程中，如前面 5.3 节所介绍。代码生成引擎的相关包，配置信息放置在一个特定的目录下，同时把 XML 工程拷贝到特定目录下，配置好 ant 工具，通过 ant 执行代码生成，就能完整的生成新项目所需要的核心代码了。开发部署人员只需把所生成的文件部署到新的 web 工程里去，这样并不等于项目就开发完成了，接下来开发人员还是需要做很多工作的，不过整个项目的架构都已经生成，核心代码比如数据访问操作，CRUD action，validation，conversion，javaBeans，

configuration，O/R mapping，action mapping，db.sql 等均完整生成，在创建数据库时，只需执行 db.sql 即可生成整个系统的所有数据表而且任何模型中创建好的表间关系均可同步反映到数据库中，不需开发人员手动任何改动；而且除了 CRUD Action 部分代码会因具体的业务处理需要做修改，基本上可以直接使用。有了这些代码，开发人员更多的精力可以投入系统的页面设计，业务优化等方面，这样避免了很多认为的失误，从而保障了快速有效的系统开发。同时，在系统开发过程中需要添加到数据库中的初始化代码和测试代码均可以在系统中通过添加特定代码的方式完成，开发人员所需做的工作就是执行一次这些数据初始化（测试）代码代码，执行完后所有的初始化数据和测试数据将被添加到数据库中，每当系统需求发生变化，开发人员所需做的工作就是，修改 XML 模型，重新执行代码生成引擎并把更新后的代码重新部署到项目中，再修改初始化数据和测试数据并执行一次数据初始化代码。这些工作可以在短短的半小时全部完成。开发人员并不会受到需求改变带来的困扰。

5.6 本章小结

本章是全文的核心，在前面介绍的理论基础上，进一步介绍了软件工程的需求工程以及面向对象需求建模和软件重用框架等。接下来介绍的是基于 XML 的需求建模体系的设计，包括 XML 体系的所有标签元素的实现，以及使用 XML 建模体系进行需求建模的方法。本章最后介绍代码生成引擎的设计，包括代码生成引擎的架构，分析，用例，流程，实现等诸多环节。

第六章 结 论

本论文基于作者几年以来的J2EE开发的实践，参与多个中大型项目的设计和开发。在此基础上进行学习软件开发相关的模式、体系架构等理论知识，研究开源项目的原理并实际使用；总结以上实践经验，为提高开发效率并保证代码的质量，设计并实现了此代码生成生成引擎，并最终完成这篇论文。开源框架和代码生成技术，使开发人员从底层功能中解脱出来，而不必拘泥于一些烦琐和重复的代码编写，为开发节省了大量的时间，可以更好地专注于用户的业务需求。这样，系统的成本降低了，周期缩短了，风险减少了。当然，生成代码的质量好坏，跟前期的需求工作及XML需求模型的创建有着直接的关系，所以，在生成代码之前，设计人员一定要深入理解需求，谨密设计需求模型和相关公共接口类，以避免以后可能造成代码的大面积更改。

在本系统的研究开发过程中，仍然存在一些可以改进的地方：首先，XML建模体系不够完善，和代码生成引擎架构的固化还无法作为大型项目的基础框架使用，同时也影响了模板抽取的质量。再就是数据模型相对简陋，而且只是针对Hibernate框架而设计，不具备普遍通用性。因为此次设计的代码生成引擎是出于一个CRM项目的启发而做出的，很多相关业务方面的设计都是考虑CRM这个领域，存在一定的局限性。同时，没有考虑表示层（页面）代码的整体生成，更多的只是某些表示层数据的生成还不够。

代码生成引擎也有存在一定的缺点，它生成的程序还是需要一个发布、部署、调试和再开发的过程，而且代码生成引擎很难做到逆向工程，如果生成的代码做了修改，而业务对象的结构又有变化还是需要大量的编码。

下一步改进方案：以Web方式使用代码生成引擎，这样保证安全使用，统一管理和升级；增加与页面定制按钮等，权限，初始化数据等相关的数据模型；并完善主从结构的模板、解析程序和数据模型，增强代码生成引擎的通用性，可配置性。

致　谢

在将近三年的研究生学习生活即将结束之际，衷心的感谢这些年来的生活学习过程中给予了我无私教诲，关怀和帮助的老师，朋友和亲人。

首先，感谢电子科技大学为我们提供了良好的学习生活环境。

其次，衷心感谢我的导师余堃教授，三年来在学术研究和生活上给了很大的帮助。尽管他平时学术公务非常繁忙，但是对于我们学术上的要求一点都不放松，真正做到了一丝不苟。

再次，感谢共同度过三年时间的诸多同学，和他们在一起的生活学习是我人生中是一段非常愉快而且最为珍贵的经历。

最后，要特别感谢我的爸爸妈妈弟弟多年来对我的鼓励和支持，感谢我的叔叔婶婶给我的关怀和帮助，让我能全心投入工作和学习当中，完成硕士研究生学业。

参考文献

[1] Erich Gamma Design Patterns:Elements of Reusable Object-Oriented softwareAddison Wesley/Pearson 机械工业出版社 2005-6-1

[2] 陈翔，王学斌，吴泉源. 代码生成技术在 MDA 中的实现 计算机应用研究 2006 年第一期

[3] 作者：Robert C. Martin 译者：邓辉.《敏捷软件开发：原则、模式与实践》原书名：Agile Software Development:Principles,Patterns and Practices Pearson Education 清华大学出版社 2003-9-1 124-128

[4] 黎永良, 崔杜武. MVC 设计模式的改进与应用[J]. 计算机工程, 2005, 31(9):96-97.

[5] 许劲松, 石磊. 基于递归MVC 结构的Web 应用软件分析模式[J]. 计算机工程与设计, 2005, 26(12):3417-3419.

[6] 作者:William Crawford，Jonathan Kaplan 译者:刘绍华、毛天露《J2EE 设计模式》原书名：J2EE Design Patterns

[7] 孙卫琴. 精通 Struts:基于 MVC 的 Java Web 设计与开发. 电子工业出版社[M], 2004. 8

[8] 作者：Patrick Lightbody, Jason Carreira 译者：谭颖华、张云飞、唐勇《Webwork in action 中文版》电子工业出版社 2006 年 11 月

[9] http://www.opensymphony.com/Webwork

[10] http://www.ognl.org

[11] 余俊新，孙涌. J2EE 中对象关系映射的研究与实现 计算机技术与发展第 17 卷第三期

[12] 夏昕.《Hibernate 开发指南》httpa/www.redsaga.com/ 15-239

[13] 孙卫琴编著《精通 Hibernate：java 对象持久化技术详解》 电子工业出版社出版 2005

[14] Martin Fowler《Refactoring:Improving the Design of Existing Code》中国电力出版社

[15] 夏昕，曹晓钢，唐勇. 深入浅出 Hibernate[M]. 北京：电子工业出版社，2005：468-469

[16] 万建成，卢雷.《软件体系结构的原理，组成与应用》2004 231-234

[17] Wolfgang Keller.《Mapping Objects To Tables》

[18] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Design Patterns: Elements of Resuable Object-Oriented Software, Addison Wesley/Pearson, 2000-9-1

[19] Subrahmanyam Allamaraju etc., Professional Java Server Programming J2EE Edition, Wrox Press 2001-9-1

[20] Sommerville I. Software Engineering[M].Fourth Edition, Addison Wesley, 1993.

[21] Jacobson I.Object2Oriented Software Engineering: A Use Case Driven Approach[M] , Addison Wesley ,1992.

[22] 孙鑫鸽，陈刚，孙小玲. 基于 JDBC 数据库连接池技术的研究与设计　计算机与信息技术

[23] 赵枫朝.基于关系数据库的对象持久化研究 北京工业大学硕士论文 2003. 04

[24] 蔡艳敏. 基于 UML 的面向对象的需求分析方法[J]. 电脑知识与技术，2006（29）

[25] 黄贤英. UML 建模过程及在需求分析中的应用[J]. 计算机工程，2001（11）

[26] 屈喜龙. UML 及面向对象的分析与设计的研究[J]. 计算机应用与研究，2005（9）

[27] 马树奇. java XML 程序员参考手册 电子工业出版社 2002-01-01：8-54

[28] Deepak Alur, John Crupi, Dan Malks 《Core J2EE Patterns: Best Practices and Design Strategies》科学出版社

[29] 毛选，魏海萍. XML Schema 数据库编程指南 电子工业出版社 2002-01-01 35-103

[30] Craig Layman 译者：方梁.《UML 与模式应用》 机械工业出版社

[31] (美)Craig Larman 著.《UML 和模式应用:面向对象分析与设计导论》.姚淑珍李虎等译.机械工业出版社.2002 年 1 月第一版.134-236

[32] 作者：赵强《基于开源软件的 J2EE 企业级应用开发》电子工业出版

[33] 作者:Frank Buschmann, Regine meunier, Hans Rohnert, Peter Sommerlad, Michael Stal《面向模式的软件体系结构卷 I：模式系统》 原书名：《Pattern-Oriented Software Architecture, Volume I: A System of Patterns》236-265

[34] Ouyang L Y, Wu K S. Mixture inventory model involving variable lead time with a service level constraint [J].Computers and Operations Research, 1997.

[35] Gupta A , Maranas C D. Managing demand uncertainty in supply chain planning [J].Computers and Chemical Engineering. 2003

[36] Nadir Gulze. 实用 J2EE 应用程序体系结构 清华大学出版社 2003.12

[37] Grady Booch, James Rumbaugh, Ivar Jacobson. 《UML 用户指南》 机械工业出版社 2001. 06

[38] Java Home Page http://java.sun.com

[39] J2EE home Page http://jav.sun.com/j2ee

[40] 张海潘.软件工程导论[M].北京:清华大学出版社, 1998.

[41] Alun Williams [EB/OL]. http://www.umlchina.com/News/Content/143.htm, 2004-05-26.

[42] 邵维忠,杨芙清.面向对象的系统设计[M].北京：清华大学出版社, 2003.

附录 A　XML 需求建模样例（Customer.cge.xml 片段）

这是使用 XML 需求建模体系进行需求建模的示例文件片段，包括基本的对象模型信息的反映方法，键的设置，验证信息，到 Hibernate 数据关系的映射等等。

```
<?xml version="1.0" encoding="UTF-8"?>
<cge:class name="Customer" xmlns:cge="http://www.haichusoft.com/cge-model-0.2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.haichusoft.com/cge-model-0.2
..\cge-model-0.2.xsd">
<!--从ContactTarget派生-->
<cge:meta name="hbm:joined-subclass">
    <cge:meta-attribute name="table" value="customers"/>
    <cge:meta name="hbm:key">
        <cge:meta-attribute name="column" value="id"/>
    </cge:meta>
</cge:meta>
<!--region sensitive : 区域敏感-->
<cge:meta name="crm:region-sensitive">
    <cge:meta-attribute name="check-region-at" value="region"/>
</cge:meta>
<!--enable search : 允许搜索-->
<cge:meta name="crm:enable-search"/>
<!--default order by : 默认排序-->
<cge:meta name="crm:default-orders">
    <cge:meta name="crm:order">
        <cge:meta-attribute name="order-by" value="recentModify"/>
        <cge:meta-attribute name="type" value="desc"/>
    </cge:meta>
</cge:meta>
<cge:extends class="ContactTarget"/>
```

```
<cge:attributes>
    <!--id：数据库主键-->
    <!-- <cge:attribute name="id">
            <cge:simple-type name="long">
                <cge:meta name="hbm:id"/>
            </cge:simple-type>
        </cge:attribute>-->
    <!--name:客户名称(可修改)-->
    <cge:attribute name="name">
        <cge:meta name="crm:enable-search"/>
        <cge:meta name="ww:validation">
            <cge:meta name="ww:requiredstring"/>
            <cge:meta name="ww:stringlength">
                <cge:meta-attribute name="minLength" value="2"/>
                <cge:meta-attribute name="maxLength" value="100"/>
            </cge:meta>
        </cge:meta>
        <cge:simple-type name="string">
            <cge:meta name="hbm:property">
                <cge:meta-attribute name="not-null" value="true"/>
                <cge:meta-attribute name="update" value="true"/>
                <cge:meta name="hbm:column">
                    <cge:meta-attribute name="name" value="name"/>
                    <cge:meta-attribute name="length" value="100"/>
                </cge:meta>
            </cge:meta>
        </cge:simple-type>
    </cge:attribute>
    <!--targetIndustry:应用领域-->
    <cge:attribute name="industry">
        <cge:meta name="crm:enable-search">
            <cge:meta-attribute name="behavior" value="list"/>
```

```
        </cge:meta>
        <cge:meta name="ww:validation">
            <cge:meta name="ww:required"/>
        </cge:meta>
        <cge:simple-type name="Industry">
            <cge:meta name="hbm:many-to-one">
                <cge:meta-attribute name="not-null" value="true"/>
                <cge:meta-attribute name="column" value="industry_id"/>
            </cge:meta>
        </cge:simple-type>
    </cge:attribute>
    <!--province : 省份-->
        <cge:attribute name="province">
            <cge:meta name="crm:enable-search">
                <cge:meta-attribute name="behavior" value="list"/>
            </cge:meta>
            <cge:simple-type name="Province">
                <cge:meta name="hbm:many-to-one">
                    <cge:meta-attribute name="not-null" value="false"/>
                    <cge:meta-attribute name="column" value="province_id"/>

                </cge:meta>
            </cge:simple-type>
        </cge:attribute>
    <!--address : 联系地址-->
    <cge:attribute name="address">
        <cge:meta name="ww:validation">
            <cge:meta name="ww:stringlength">
                <cge:meta-attribute name="minLength" value="0"/>
                <cge:meta-attribute name="maxLength" value="255"/>
            </cge:meta>
        </cge:meta>
```

```
            <cge:simple-type name="string">
                <cge:meta name="hbm:property">
                    <cge:meta-attribute name="not-null" value="false"/>
                    <cge:meta-attribute name="column" value="address"/>
                </cge:meta>
            </cge:simple-type>
    </cge:attribute>
    <!--email : 电子邮件-->
    <cge:attribute name="email">
        <cge:meta name="ww:validation">
            <cge:meta name="ww:email"/>
            <cge:meta name="ww:stringlength">
                <cge:meta-attribute name="minLength" value="0"/>
                <cge:meta-attribute name="maxLength" value="255"/>
            </cge:meta>
        </cge:meta>
        <cge:simple-type name="email">
            <cge:meta name="hbm:property">
                <cge:meta-attribute name="not-null" value="false"/>
                <cge:meta-attribute name="column" value="email"/>
            </cge:meta>
        </cge:simple-type>
    </cge:attribute>
    <!--home page : 主页-->
    <cge:attribute name="homePage">
        <cge:meta name="ww:validation">
        <cge:meta name="ww:url"/>
        <cge:meta name="ww:stringlength">
            <cge:meta-attribute name="minLength" value="0"/>
            <cge:meta-attribute name="maxLength" value="255"/>
        </cge:meta>
        </cge:meta>
```

```
        <cge:simple-type name="url">
            <cge:meta name="hbm:property">
                <cge:meta-attribute name="not-null" value="false"/>
                <cge:meta-attribute name="column" value="home_page_url"/>
            </cge:meta>
        </cge:simple-type>
    </cge:attribute>
    <!--type: 客户创建人-->
    <cge:attribute name="createUser">
        <cge:simple-type name="User">
            <cge:meta name="hbm:many-to-one">
                <cge:meta-attribute name="not-null" value="true"/>
                <cge:meta-attribute name="column" value="createUser_id"/>
        </cge:meta>
        </cge:simple-type>
    </cge:attribute>
    <!--created: 创建时间-->
        <cge:attribute name="created">
            <cge:simple-type name="timestamp">
                <cge:meta name="hbm:property">
                    <cge:meta-attribute name="not-null" value="true"/>
                    <cge:meta-attribute name="column" value="created"/>
                </cge:meta>
            </cge:simple-type>
        </cge:attribute>
    <!--assocations-->
    <!--interested product:感兴趣产品-->
    <cge:attribute name="interestedProducts">
        <cge:unary-collection name="set">
            <cge:meta name="hbm:set">
                <cge:meta-attribute name="table" value="interested_products"/>
            </cge:meta>
```

```
            <cge:meta name="hbm:key">
                <cge:meta-attribute name="column" value="customer_id"/>
            </cge:meta>
            <cge:collection-element>
                <cge:simple-type name="InterestedProduct">
                    <cge:meta name="hbm:one-to-many"/>
                </cge:simple-type>
            </cge:collection-element>
        </cge:unary-collection>
    </cge:attribute>
```

附录 B 数据模型泛化关系示例

此例示例了对象间的继承关系。ContactTarget.cge.xml 定义了一个抽象类模型，Partner.cge.xml 定义了一个继承自 ContactTarget 的对象类。

1、Partner.cge.xml 片段

```
<?xml version="1.0" encoding="UTF-8"?>
<cge:class name="Partner" xmlns:cge="http://www.haichusoft.com/cge-model-0.2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.haichusoft.com/cge-model-0.2
..\cge-model-0.2.xsd">
<cge:meta name="hbm:joined-subclass">
    <cge:meta-attribute name="table" value="partners"/>
    <cge:meta name="hbm:key">
        <cge:meta-attribute name="column" value="id"/>
    </cge:meta>
</cge:meta>
<cge:meta name="crm:region-sensitive">
    <cge:meta-attribute name="check-region-at" value="region"/>
</cge:meta>
<cge:meta name="crm:enable-search"/>
<!--extends ContactableEntity-->
<cge:extends class="ContactTarget"/>
<cge:attributes>
    <!--name:合作伙伴名称-->
    <cge:attribute name="name">
        <cge:meta name="crm:enable-search"/>
        <cge:meta name="ww:validation">
            <cge:meta name="ww:requiredstring"/>
            <cge:meta name="ww:stringlength">
```

```
                <cge:meta-attribute name="minLength" value="2"/>
                <cge:meta-attribute name="maxLength" value="100"/>
            </cge:meta>
        </cge:meta>
        <cge:simple-type name="string">
            <cge:meta name="hbm:property">
                <cge:meta-attribute name="not-null" value="true"/>
                <cge:meta name="hbm:column">
                    <cge:meta-attribute name="name" value="name"/>
                    <cge:meta-attribute name="length" value="100"/>
                </cge:meta>
            </cge:meta>
        </cge:simple-type>
    </cge:attribute>
    ......
```

2、ContactTarget.cge.xml 片段

```
<?xml version="1.0" encoding="UTF-8"?>
<cge:class name="ContactTarget"
abstract="true" :cge="http://www.haichusoft.com/cge-model-0.2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.haichusoft.com/cge-model-0.2
..\cge-model-0.2.xsd">
<cge:meta name="hbm:class">
    <cge:meta-attribute name="table" value="contact_targets"/>
</cge:meta>
<cge:attributes>
    <!--id:数据库主键-->
    <cge:attribute name="id">
        <cge:simple-type name="long">
            <cge:meta name="hbm:id"/>
        </cge:simple-type>
    </cge:attribute>
```

```
<!--assocations-->
<!--contact persons: 客户联系人-->
<cge:attribute name="contactPersons">
    <cge:unary-collection name="set">
        <cge:meta name="hbm:set">
            <cge:meta-attribute name="table" value="contact_persons"/>
            <cge:meta-attribute name="order-by" value="created desc"/>
        </cge:meta>
        <cge:meta name="hbm:key">
            <cge:meta-attribute name="column"
            value="owner_contact_target_id"/>
        </cge:meta>
        <cge:collection-element>
            <cge:simple-type name="ContactPerson">
                <cge:meta name="hbm:one-to-many"/>
            </cge:simple-type>
        </cge:collection-element>
    </cge:unary-collection>
</cge:attribute>
.....
```

学习期间研究成果及发表的学术论文

一、简历

黄钦，男，中共党员，1980 年出生于湖南长沙。

二、科研情况

研究生学习期间，参加的研究项目如下：

项目 1

时间	项目名称	所做工作
2005 年 5 月至 2006 年 1 月	合川市检察院信息系统	需求分析，系统设计开发和测试

项目 2

时间	项目名称	所做工作
2006 年 1 月至 2006 年 11 月	师范教育资源网格门户系统	新技术探索和系统需求设计开发测试

三、发表的论文

[1] 黄钦，余堃，基于 Web 服务的网格资源分配与管理组件技术分析，电子科技大学研究生学报计算机科学与技术增刊第 27 期发表。

基于开源框架的通用代码生成引擎设计与实现

作者： 黄钦
学位授予单位： 电子科技大学