基于ORM的轻量级数据持久化技术研究及应用

李 杰
(重庆大学计算机学院 重庆 400044)

摘 要 针对如何在对象和关系数据库之间建立一种高效的映射关系的问题,在深入研究数据持久层的核心功能的基础上,分析和比较了当前流行的解决数据持久层问题的典型技术方案及其各自的优缺点,借鉴数据持久层设计的一般思想和Hibernate的功能与实现思路,具体实现了一个较为通用的数据持久层框架。

关键词 ORM,轻量级,数据持久化,Hibernate
中图法分类号 TP311 **文献标识码** A

Research and Application of Lightweight Data Persistence Technology Based on ORM

LI Jie
(College of Computer Science, Chongqing University, Chongqing 400044, China)

Abstract In order to setup an efficient mapping relation between object and database, the paper proposed a method for data persistence, in which a data persistence layer between logical layer and database layer is created based on the key function of data persistence layer. Compared the typical methods for data persistence and analyzed their merits and demerits, got inspirited from the function of Hibernate, then realized a universal data persistence layer frame.

Keywords ORM, Lightweight, Data persistence, Hibernate

1 引言

在如今的企业级应用开发环境中,面向对象的开发方式已成为主流。一方面,面向对象技术的理想存储机制——面向对象数据库还不成熟;另一方面,关系数据库是目前使用最广泛的数据库,因此,在开发过程中通常会采用面向对象的应用和关系数据库相结合的方式。

然而,关系数据库中存放的是关系数据,它是非面向对象的。显然,如何用关系数据库实现对象持久化是面向对象软件开发面临的一大问题。事实上,对象模型与关系数据库模型之间存在着"阻抗不匹配"[1],为了解决这个"阻抗不匹配"问题,提高开发效率,对象/关系映射(Object/Relational Mapping,ORM)技术应运而生。ORM问题实际上就是如何实现数据存储和访问的问题,该问题通常也称作数据持久性问题。一个更好的解决方案是在应用程序的业务逻辑层和数据库层之间构建一个持久层,由持久层来提供对象/关系映射服务,封装数据访问细节。采用持久层的方法可以从根本上消除应用程序和关系数据库的耦合,使得数据访问对于应用程序的源代码是透明的。如果数据库的库表结构发生改动,则需要对持久层的配置文件做适当修改,但不会对应用程序造成影响。ORM是应用程序的对象到关系数据库中的表的自动的和透明的持久化。它的作用是在关系型数据库和业务实体对象之间作一个映射。这样,在具体地操作业务对象时,就不需要再去和复杂的SQL语句打交道,只需简单地操作对象的属性和方法[2]。

2 基于轻量级数据持久化技术的数据库设计

2.1 Hibernate概述

Hibernate是一个面向Java环境的对象/关系数据库映射工具。Hibernate不仅仅管理Java类到数据库表的映射,还提供数据查询和获取数据的方法,可以大幅度减少开发时人工使用SQL和JDBC处理数据的时间,而只对对象进行操作[3]。Hibernate是轻量级的封装,避免过多复杂的功能,减轻了程序员的负担。同时,它也是一个开源的代码,提供开放的API,用户可以自行扩充其功能。Hibernate在大多数主流J2EE应用服务器的受管理环境中都可以良好运作,也可以作为独立应用程序运行。

Hibernate的精髓是持久层实现模式[4]。它完全是针对对象的持久化,即把一个普通的Java对象映射到关系数据库中。面向对象设计中的继承与多态机制在Hibernate里也得到了支持,在数据查询中,它支持动态Query,并提供对十六种数据库语言的支持,它沿用传统数据库的事务模型,使程序员不必为新的事务模型大伤脑筋。另外,Hibernate是本地调用,比Entity Bean有更高的性能,而且它改进的速度之快也是其他ORM产品无法企及的。

2.2 对Hibernate持久化策略的改进与扩展

Hibernate虽然功能强大,但针对一个实际项目的具体要求而言,有时又显得有些繁琐和复杂。通过对持久层的相关理论与技术及对Hibernate核心功能的研究,并结合项目实例,本文提出了一种改进和扩展Hibernate持久化策略的方

到稿日期:2009-09-02 返修日期:2010-01-09 本文受重庆市教学改革项目(项目编号:0635207)资助。
李 杰(1976—),男,硕士,讲师,主要研究方向为电子商务、远程教育,E-mail:jsjx@cqu.edu.cn。

案。

(1)Hibernate的灵活性主要体现在通过映射文件在对象属性和关系字段之间建立联系[5]。这种方法有易于修改的优点,但有时也相当麻烦。例如,如果一张表中有很多字段,每一个字段都至少要在配置文件中出现一次,这显然是相当麻烦的。那么,问题是能不能不通过配置文件就可以让每一个需要持久化的类和表自动地完成映射呢?

答案是肯定的。因为Hibernate进行对象与关系映射的核心思想是通过Java的类反射机制来建立这种映射,所以,借助于Java的类反射机制,并对类和表在设计上做一些规范约定,那么就完全可以不建立配置文件而实现对象与关系的映射。

(2)Hibernate仍在发展和完善中。目前它还没有对视图、触发器、存储过程等数据库的高级功能提供支持。为了扩充Hibernate的功能,可以通过自定义组件的方式来实现Hibernate所不具备的功能[6]。

(3)Hibernate拥有自己的查询语言,功能也足够强大,但是掌握这种新的语言规范毕竟需要时间和精力,而且广泛应用该语言还需要得到使用者的认同。所以,一个简单可行的方法是通过自定义符合面向对象编程理念的查询帮助组件,对标准的SQL语句进行封装,以实现对数据库的查询操作。理想的自定义组件,将会对Hibernate查询功能起到补充的作用。

(4)采用Hibernate和自定义持久层组件相结合的方法,在比较复杂和需要经常变化的业务逻辑中使用Hibernate,而在大量的一般业务逻辑中则使用自定义持久层组件,共同完成对数据库的操作,以提高开发效率和减少开发工作量[7]。

3 应用系统主要实现技术及相关代码

本课题来源于重庆市教改项目"高等学校毕业设计(论文)教学环节现代化管理的研究与实践"(项目编号:0635207)。课题的目的是要设计一个本科毕业设计(论文)管理系统,要实现毕业设计教学管理的工作流程,完成提交论文选题、课题的双向选择、管理论文材料、论文评审分组、论文评分、打印工作所需材料等一整套业务逻辑。并对系统的所有功能模块进行测试,及在系统试运行期间调整、修改程序。

3.1 Web层的开发与实现

本系统使用了Struts框架来分离显示逻辑和业务逻辑。Struts框架实现了MVC模式,在Struts框架中,视图由一组JSP文件构成,控制器由ActionServlet和Action来实现。视图和控制器共同完成了Web层的开发。

(1)前端视图

前端视图主要采用JSP技术创建。在前端显示的JSP页面的编写中,应尽量不含逻辑处理的Java脚本代码,而且不应该含有业务逻辑。

Struts提供了自定义的标记库可以使用,通过这些自定义标记可以非常好地与系统的Model部分交互,而且使用这些自定义标记创建的JSP表单,可以实现和Model部分中的ActionForm的映射,完成对用户数据的封装,同时这些自定义标记还提供了像模板定制等多种显示功能。

JSP中的JSTL标签库也提供了丰富的功能,特别是其中的core标签,可以更好地控制显示逻辑。

本系统的JSP基本上都是用HTML标签、JSTL和Struts标签组合而成,几乎没有嵌入代码在里面,这样不仅使得文档结构清晰,而且彻底分离了页面显示和业务逻辑。在页面中看不到任何模型信息,非常有利于保护业务安全。由于JSP页面清晰简单,没有业务逻辑,因此维护页面的工作也将变得更加方便。下面给出一个学生打印所有论文材料的页面,页面中控制动态显示的代码片断如下:

```
〈html〉
……
〈bean:write name="student" property="name"/〉的毕业设计论文相关材料〈/td〉
〈/tr〉
……
〈bean:define id="materials" name="student" property="thesisMaterials"/〉
〈tr〉
〈logic:empty name="materials" property="ktReport"〉
……
    〈/logic:empty〉
〈logic:notEmpty name="materials" property="ktReport"〉
        〈td class="data" align="center" height="30"〉
〈html:link action="/thesiswork/viewThesisMaterialPage?materialType=ktReport" paramName="materials" paramId="materialId" paramProperty="ktReport.id"〉开题报告〈/html:link〉〈/td〉
        〈td class="data" align="center" height="30"〉
……
〈/html〉
```

由代码清单的前几行可以看到,本页面使用了Struts HTML标签库、Struts Bean标签库和Struts Logic标签库。〈bean:define〉标签用于定义变量,〈bean:write〉标签用于输出变量的属性,此处输出学生姓名,就用了这个标签。〈html:link〉标签用于生成HTML〈a〉元素,但是〈html:link〉标签在创建超级链接时,可以允许多种方式在URL中包含请求参数,如显示打印开题报告的语句。Struts Logic标签库,顾名思义,是用来控制逻辑的。由于学生有可能没有提交毕业设计相关材料,因此必须判断学生毕业设计的相关材料是否存在,如果存在,就用〈html:link〉创建链接,转发请求,如果不存在,则直接显示该材料的名称。这里没有用Java代码片断来控制这些逻辑,而是使用了Struts Logic标签库中的〈logic:empty〉与〈logic:notEmpty〉标签。

整个页面结构清晰,不含任何Java代码片断及任何业务逻辑。使用标签来代替Java代码片断,完全隔离业务逻辑,不仅使得页面编写方便,而且页面也易于维护。

(2)控制器

在本系统中,控制器由ActionServlet类和Action类来实现。ActionServlet类是Struts框架中的核心组件。ActionServlet继承了javax.servlet.http.HttpServlet类,它在MVC模型中扮演中央控制器的角色。ActionServlet主要负责接收HTTP请求信息,根据配置文件struts-config.xml的配置信息,把请求转发给适当的Action对象。如学生登录后,在论文工作菜单下面单击论文材料打印,就是向服务器提交请求到ActionServlet。然后ActionServlet根据配置文件的配置信息找到上一个页面对应的action。PageAction是继承Struts框架中Action组件而来的。excuteAction()方法执行更新

JSP 页面内建对象；checkAccessRights()方法确认当前用户是否有访问权限。当 ActionServlet 控制器收到学生要求论文材料打印的请求之后，ActionServlet 根据 struts-config. xml 中的配置信息，转发到上述 action。这个 action 首先检查是否有权限访问，如果没有权限，则返回错误信息页面给 ActionServlet，然后 ActionServlet 根据配置信息，请求一个错误显示页面给学生。如果有访问权限，则返回空，并执行 excuteAction()方法。此方法负责调用模型的方法，更新 JSP 页面内建对象，然后根据 struts-config. xml 中的配置信息跳转到相应页面。

在本系统的设计中，还设计了一个 CommandAction 类，同样继承自 Struts 框架中的 Action 组件。在 CommandAction 类中，excuteAction()方法负责调用模型的方法，更新模型的状态。然后跳转到 pageaction 去更新 JSP 页面内建对象。

所有的控制都是通过 ActionServlet 和 RequestProcessor 组件集中处理的，由于所有的请求都通过这两个组件的过滤，因此可以降低视图组件之间以及视图组件和模型组件之间的相互依赖关系，提高每个组件的相对独立性。由控制器组件来决定把合适的视图组件返回给用户，这可以减少视图组件之间直接的、错综复杂的链接关系，使应用更加灵活，也便于维护。

(3)struts-config. xml 配置文件

在 Struts 中，描述用户请求路径和 Action 映射关系的配置信息都被存储在 struts-config. xml 中。struts-config. xml 描述了 Controller 所使用的把请求对应到具体处理的法则，同时它还描述了客户提供的数据与 ActionForm 组件的对应映射关系。在该配置文件中，每一个 action 的映射信息都通过一个〈action〉元素来配置。如上面那个打印论文材料的 action 配置信息如下：

```
〈action
path="/thesisgrade/student/printThesisMaterialsPage"
type="computermis. struts. action. thesisgrade. student. printThesisMaterialsPage "
    name="form"
    scope="request"
    input="error"
    roles="student"
    unknown="false"
    validate="true"
  〉
    〈forward
      name="nextpage"
path = "/WEB-INF/jsp/thesisgrade/printThesisMaterialsPage.jsp"
      redirect="false"
    /〉
〈/action〉
```

〈action〉元素的 path 属性指定请求访问 Action 的路径，type 属性指定 Action 的完整类名，name 属性指定需要传递给 Action 的 ActionForm Bean，scope 是指定 ActionForm Bean 的存放范围，validate 指定是否执行表单验证，在此为 true，表明需要执行表单验证。roles 指定当前用户的角色，如果角色不符，则通不过验证。

集中组件配置信息，使得 Struts 有着良好的页面导航功能，通过 struts-config. xml 配置文件，开发者可以把握整个系统各部分之间的联系，这对于后期维护有着莫大的帮助。

然而，在开发的过程中，由于编写的 action 日益增多，struts-config. xml 变得越来越庞大，维护起来越来越复杂，有时甚至会忘了切换到 struts。

config. xml 去增加新的 action 配置信息，于是采用 XDoclet 编写了在 struts-config. xml 中生成〈action〉元素配置信息的脚本，使得维护此配置文件变得方便起来。

3.2 业务逻辑层的实现

虽然 Struts 将 MVC 分离显示逻辑和业务逻辑的能力发挥得淋漓尽致，但是它主要是针对表示层设计的，对后端的逻辑层的支持不是很强。可以想象，如果不能很好地组织对象的话，当需求改变时，即使前端的 JSP 维护方便，由于业逻辑层的混乱，系统还是将变得很难维护。在本系统中，我们采用了 Spring 实现的 IoC 机制来控制对象间的关系和进行事务管理，利用 Spring 的依赖注入机制可以使整个系统保持良好的针对接口编程而不是针对类编程的习惯。

(1) 业务逻辑层对象的组织方法

在本系统中，模型部分被分为 3 种对象：业务逻辑对象、域对象和数据访问对象。

业务逻辑对象是完成系统业务逻辑的对象。在本系统中这些对象通通放在名为 service 的包内。把为完成某一业务逻辑而设计的方法集中定义在名为 IXxxService 的接口中，然后为每个 IXxxService 接口写一个接口实现类，名为 XxxServiceImpl。采用这种方式使得对象间的依赖关系变为对象对接口的依赖，这种方式不仅屏蔽了具体对象的实现细节，而且分离了对象间的依赖关系，从而便于 Spring 框架采用依赖注入方式对对象间的关系进行管理。比如 IGroupService 中定义了有关 Group 的操作，诸如增加分组、自动分组、修改分组等方法。然后在 GroupService 中实现。接口和实现类之间的关系由 Spring 控制。

域对象是处于问题域中的实体在系统中的对象表示。如果说业务逻辑对象是更改系统状态动作的话，那么域对象就是表明了系统的内部状态。这些对象统一被放在名为 domain 的包内。如论文评审工作涉及到 teacher，student，group，groupDuty 等对象。

数据访问对象基于 Hibernate 框架构建了对数据存取的访问操作。这些对象被通通放在名为 Dao 的包中。这些对象在组织方式上和业务逻辑对象类似，都是命名为 IXxxDao 和 XxxDaoImpl 的形式。它们之间的松耦合关系也是由 Spring 来控制的。但是数据访问对象在实现上要与 Hibernate 结合起来。

通过如上组织方式，模型部分对象间的全部关系都被纳入到 Spring 的管理之中。各对象之间遵守了“针对接口编程”的原则，本系统每个对象对外的功能都被抽象为接口，于是每个对象就都成为实现了的抽象接口的具体实例。采用这种方式使得对象间的依赖关系变为对象对接口的依赖，这种方式不仅屏蔽了具体对象的实现细节，而且分离了对象间的依赖关系，从而达到对象间的松耦合关系，便于 Spring 框架采用依赖注入方式对对象间的关系进行管理。

(2) Spring 控制对象和事务管理

上面提到遵循“针对接口编程”的原则，亦即将对象对外的功能抽象为接口，让对象间的依赖关系变成对象对接口的依赖，从而便于 Spring 框架采用依赖注入方式对对象间的关系进行管理，它是通过其配置文件 applicationContext. xml 实现的。

下面为本系统中的 GroupDao 在 applicationContext. xml 中的配置代码。

```
〈bean id="groupDao" class="computermis. dao. hibernate. GroupDaoImpl"〉
    〈property name="baseDao"〉
        〈ref local="baseDao"/〉
    〈/property〉
    〈/bean〉
    〈bean id="groupServiceTarget" class="computermis. service. impl. GroupServiceImpl" 〉
        〈property name="groupDao"〉
            〈ref local="groupDao"/〉
        〈/property〉
    〈/bean〉
```

上述代码就为几个对象实现了依赖注入。

另外 Spring 可以声明式地管理事务，比如本系统中的 Group 数据访问对象，接口是 IGroupDao，它的实现类是 GroupDaoImpl。

只要底层数据库支持，就可以在一个事务中添加到所有用户。这样就可以保持事务的一致性，而不致于利用程序为每个用户逐一添加到事务。

由于模型部分全部处于 Spring 的管理之下，为了让 Struts 构建的控制器能够调用 Spring 管理的业务逻辑完成业务，设计中采用了服务定位器模式(Service Locator)来进行访问。

3.3 数据持久层的实现

(1) Hibernate 进行 OR 映射

Hibernate 使用 XML(*. hbm. xml) 文件将 Java 类映射到表，将 JavaBean 属性映射到数据库表。有一组 XDoclet 标签支持 Hibernate 开发，使得创建所需要的 *. hbm. xml 文件更容易了。下面的代码将本系统的 Group 类映射到数据库表。

```
/**
 * @hibernate. class table="DBGroup" proxy="computermis. domain. Group"
 ......
 */
public class Group {
    private String id = null;
    private String grade = null;
  private Set students = new HashSet();
 ......
    /**
     * @hibernate. property length="50" not-null="true"
     * @return
     */
 ......
    public String getId() {
        return id;
    }
    /**
     * @hibernate. set inverse="true" table="Student" lazy="true" cascade="all"
     * @hibernate. collection-key column="Group_id"
     * @hibernate. collection-one-to-many class="computermis. domain. Student"
     * @return
     */
     ......
}
```

在 XDoclet 标签中还设置了对象间的关系，来表明答辩组和学生之间是一对多的关系。

在本系统的开发中，通过 ORM 将系统中的 Domain Object 自动映射到各个数据库表，从而使得编码工作只需关心 Object 的相关属性，而无需再纠缠于 JDBC ResultSet 中毫无意义的字段型数据。

(2)基于 Hibernate 的数据访问对象

下面给出 GroupDao 的实现过程。首先是定义了一个 IBaseDao 继承自 Hibernate DaoSupport、作为所有数据访问对象的一个超类。在下面的代码清单中给出了 IGroupDao 实现类的部分代码。

IGroupDao 实现类的部分代码：

```
public class GroupDaoImpl implements IGroupDao {
    private IBaseDao baseDao;
    public void setBaseDao(IBaseDao baseDao) {
        this. baseDao = baseDao;
    }
    public Group getGroupById(String id) {
        return(Group) baseDao. load(Group. class,id);
    }
    public void addGroup(Group group) {
        baseDao. add(group);
    }
    public void updateGroup(Group group) {
        baseDao. update(group);
    }
    ......
}
```

在 GroupDao 的实现中，使用了代理模式，通过 baseDao 进行操作，整个操作显得非常简单。

结束语 数据持久化问题是应用系统开发中的一个非常重要的问题。由于面向对象编程和关系型数据库的广泛使用，使得解决对象与关系映射的问题成为现阶段实现数据持久性问题的一个主要途径。通过对“高等学校毕业设计(论文)教学环节现代化管理的研究与实践”系统进行实现，并在系统实现中运用本文所设计的数据持久层架构及架构所提供的服务，利用 Hibernate 完成对数据持久层的设计，进一步验证了该数据持久层框架的可行性与实用性。

参 考 文 献

[1] Ambler S W. Mapping Objects to Relational Databases: O/R Mapping In Detail [OL]. http://www. agiledata. org/essays/mapp-ingObjects. html, 2000

(下转第 208 页)

VPRS模型得到的SVM识别结果。表3记录了VPRSCA模型中各数据集在不同噪音阈值上的SVM识别结果。图1、图2反映了基于VPRSCA模型的属性约简与基于VPRS模型的属性约简测试准确率的变化趋势。从实验结果可以看出，当噪音域值β较小时，Wine数据集使用VPRS模型约简效果较好，Pima与Taylor数据集使用VPRS模型与VPRSCA模型几乎一致；但当噪音域值β较大时，基于VPRSCA模型的约简效果更好；另一方面，Glass数据集在整个噪音域值β上，使用VPRSCA模型约简效果好，这说明本文建立的VPRSCA模型具有比VPRS模型更强的噪音处理能力。

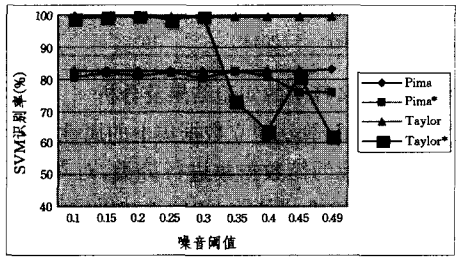

图1 Pima，Taylor数据集在VPRSCA及VPRS模型上约简的识别结果

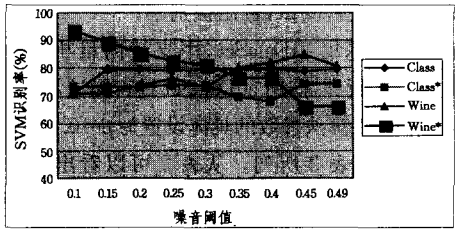

图2 Glass，Wine数据集在VPRSCA及VPRS模型上约简的识别结果

表3 VPRSCA模型约简的测试结果

Dataset \ Threshold β	0.1	0.15	0.2	0.25	0.30	0.35	0.4	0.45	0.49
Pima	82.6	82.6	82.6	82.6	82.6	82.6	82.6	82.6	82.8
Wine	71.1	71.0	73.6	76.2	74.3	80.3	82.2	84.6	80.9
Glass	70.9	79.7	79.7	79.1	79.6	79.6	79.6	79.3	79.6
Taylor	99.8	99.7	99.7	99.7	99.7	99.7	99.7	99.7	99.7

实验3 VPRSCA模型约简与经典粗糙集模型RS约简的对比及分析

由于RS属性约简已有多种方法，实验3的目的是将CVAR方法与多种属性约简方法进行对比。实验步骤如下：

Step1 对表3中各训练集TrDataSet，采用信息熵的离散化方法[9]对连续值属性做离散化处理。

Step2 用多种约简方法进行约简，生成约简数据集rsDataSet，用SVM对rsDataSet数据集进行训练，用allDataSet进行测试，输出识别率。

实验采用重庆邮电大学计算机与科学技术研究所开发的RIDAS测试平台来生成[10]。本文中RS模型下各种约简方法均来自文献[9]。

实验3的结果如表4所列。表4反映了各数据集在RS模型下各约简集上的SVM识别率。结合实验2中表3所列的结果，可以看出，只有Wine数据集在基于信息熵的约简、基于分明矩阵约简上的测试准确率比CVAR方法稍高，其它各数据集的测试准确率均低于CVAR约简方法上的测试准确率。这也说明了本文建立的VPRSCA模型的有效性及对噪音的处理能力。

表4 RS模型下不同约简方法[9]的SVM测试结果

Dataset \ Methods of attribute reduction	基于信息熵约简	一般属性约简	特征选择约简	MIBARK属性约简	基于分明矩阵约简
Pima	72.4	81.0	82.3	69.7	80.0
Wine	87.1	76.4	75.3	66.9	87.6
Glass	53.3	64.5	44.4	45.3	63.6
Taylor	79.6	99.5	99.5	99.5	79.6

结束语 本文提出了连续值属性决策表中的可变精度粗糙集模型VPRSCA和连续值属性决策表中属性约简方法CVAR。该方法不需要事先对连续值属性做离散化处理，从而有效减少了连续值属性离散化的信息损失。UCI数据集上的实验结果表明，与经典粗糙集模型及VPRS模型相比，该方法能够较好地适应连续值属性的决策表中的属性约简，并具有较好的性能。基于VPRSCA的高效约简算法是我们下一步研究工作的一项内容。

参考文献

[1] 张雪英，刘凤玉，Krause J. 粗糙集分类算法的近似决策规则和规则匹配方法[J]. 计算机科学，2005，32(6)：129-132

[2] 薄洪光，刘晓冰，马跃，等. 基于粗糙集的钢铁行业工艺知识发现方法[J]. 计算机集成制造系统，2009，15(1)：135-141

[3] 徐菲菲，苗夺谦，魏莱，等. 基于互信息的模糊粗糙集属性约简[J]. 电子与信息学报，2008，30(6)：1372-1375

[4] 肖迪，胡寿松. 实域粗糙集理论及属性约简[J]. 自动化学报，2007，33(3)：253-258

[5] 刘文军，古云东，李洪兴. 一种基于t相似分类的连续值域决策表的决策算法[J]. 模式识别与人工智能，2005，18(6)：652-656

[6] Pal S K. Case Generation Using Rough Sets with Fuzzy Representation[J]. IEEE Transactions on Knowledge and Data Engineering，2004，16(3)：292-230

[7] Ziarko W. Variable Precision Rough Sets Model[J]. Journal of Computer and System Sciences，1993，46(1)：39-59

[8] PrȩBdki，Słowiński R，Stefanowski J，et al. A Data Analysis and Decision Support System for Multiple Attributes Classification Problems Based on Rough Sets with Indiscernibility Relation [DB/OL]. http://idss.cs.put.poznan.pl/site/rose.html，2009-01-06

[9] 王国胤. Rough集理论与知识获取[M]. 西安：西安交通大学出版社，2001

[10] Wang G Y，Zheng Z，Zhang Y. RIDAS-A Rough Set Based Intelligent Data Analysis System[C]//Proceedings of ICMLC 2002. Beijing：IEEE Press，2002：646-649

(上接第193页)

[2] Bauer C，King G. Hibernate in action[M]. Manning Publication Co.，2005

[3] 孙卫琴. 精通Hibernate Java对象持久化技术详解[M]. 北京：电子工业出版社，2005

[4] Hibernate Reference[EB/OL]. http://www.hibernate.org/hibdoes/reference/htm l/. 2004-03-29

[5] 孙卫民，曹正凤. Hibernate对Struts框架的扩展研究[J]. 计算机工程与设计，2008，29(04)：858-861

[6] 赵广利. 基于NHibernate的数据持久化方案[J]. 计算机工程，2009，35(20)：53-55

[7] 顾春华，贾欣歌，顾兢. 一种动态对象/关系映射框架及其实现[J]. 华东理工大学学报：自然科学版，2009，30(6)：882-885