基于Hibernate技术的持久层解决方案及实现①

汪　萌　曲俊华 (华北电力大学 计算机科学与技术系 北京 102206)

摘　要： 为了提高 B/S 结构中数据并发访问控制能力，使应用系统的结构层次更加清晰，在比较了目前流行的几种数据持久化解决方案的基础上，深入研究了 Hibernate 数据持久化技术，并将此技术应用于具体的 web 应用系统，实现了基于 Hibernate 技术的持久层解决方案，改善原系统的性能，达到预期目的。

关键词： 数据持久化；持久化中间件；Hibernate；DAO 模式；工厂模式

Research and Implementation of Persistence Solution Based on Hibernate Technology

WANG Meng, QU Jun-Hua

(Department of Computer Science and Technology, North China Electric Power University, Beijing 102206, China)

Abstract: To improve the data concurrency control ability in B/S architecture and make the system hierarchical architecture clearer, this paper presents a thorough study of hibernate data persistence technology after comparing and analyzing several popular solutions to data persistence. It applies this technology to a Web system to resolve the persistence based on Hibernate technology. With the technology, the system performance has been improved and the expected results are achieved.

Keywords: data persistence; persistence middleware; hibernate; DAO; factory pattern

为了提高数据的集成度和软件的易操作性、开放性和可扩充性，选择 J2EE 架构是比较好的解决方案，但是在传统的三层软件架构中存在应用程序和数据库结构的直接耦合。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持久层，这种数据持久化方案能够提高其数据存储效率和并发用户承受能力，能解决 B/S 结构中数据并发访问控制能力较低的问题，使应用系统的结构层次更加清晰，并实现层与层之间的解耦，提高系统的扩展性及可维护性。

本文将基于 Hibernate 技术的持久层解决方案应用于师生交互应用系统中，分析数据持久层的本质，深入剖析了 Hibernate 原理，并通过 DAO 模式来完成持久层的设计与实现，力求改善原系统的性能，达到预期目的。

1　数据持久化的概念和相关技术

数据持久化就是将内存中的数据存储在关系型的数据库，磁盘文件，XML 数据文件等等中[1]。持久化技术封装了数据访问细节，为大部分业务逻辑提供面向对象的 API。通过持久化技术可以能够完成大部分数据库操作并减少访问数据库数据的次数，增加应用程序执行速度；持久化不依赖于底层数据库和上层业务逻辑实现，更换数据库时只需修改配置文件而不用修改代码，封装了所有的数据访问细节，使得业务逻辑层只专注于业务逻辑。

目前实现数据持久化的技术主要有以下几种：

JDBC 访问数据库：JDBC 是一种用于执行 SQL 语句的 Java API。JDBC 是 Java 访问数据库的基石，其他几种技术本质上只是更好的封装了 JDBC，提供了更为上层的更为强大的接口而已。

主动域对象：域对象(Domain Object)是真实世界的软件抽象[2]。可分为实体域对象、过程域对象、事件域对象。实体域对象封装自身的数据访问细节，过程域对象负责业务逻辑，程序结构更加清晰。关系

① 收稿时间:2009-06-15

数据模型发生改变，只需修改主动域对象，不需要修改过程域对象的业务方法。

ORM 模型：ORM 是通过使用描述对象和数据库之间映射的元数据，将 java 程序中的对象自动持久化到关系数据库中，充当了业务逻辑层和数据库层的桥梁,其中域模型是面向对象的，关系数据库模型是面向关系的。ORM 中间件运行时参照对象关系映射文件的信息，把域对象持久化到数据库中。

CMP 模式：CMP 模式是针对 J2EE 架构提出的，在 J2EE 架构中， EJB 组件分为会话 EJB 和实体 EJB，实体 EJB 又分为 BMP(EJB 本身管理持久化)和 CMP(容器管理持久化)[2]。

JDO 模式:JDO 是近几年新兴的数据持久性技术，是 SUN 公司制定的描述对象持久化语义的标准 API。它支持把对象持久化到任意一种存储系统中，包括面向对象的数据库、基于 XML 的数据库，以及其他专有存储系统。

2 Hibernate技术

Hibernate 是一种基于 Java 的持久化中间件，对 JDBC 做了轻量级的封装，提供 ORM 映射服务，数据查询和缓存功能[3]。Hibernate 的核心接口如图 1 所示，其中 Configuration 接口负责配置启动 Hibernate 并创建 SessionFactory 对象；SessionFactory 接口负责初始化 Hibernate， 创建 Session 对象，一个 SessionFactory 实例对应一个数据源,它是线程安全的；Session 接口负责保存、更新、删除、加载、查询对象，它不是线程安全的，Session 实例是轻量级的，创建和销毁都不需要太多的资源；Transaction 接口负责管理事务；Query 和 Criteria 接口负责执行数据库查询。

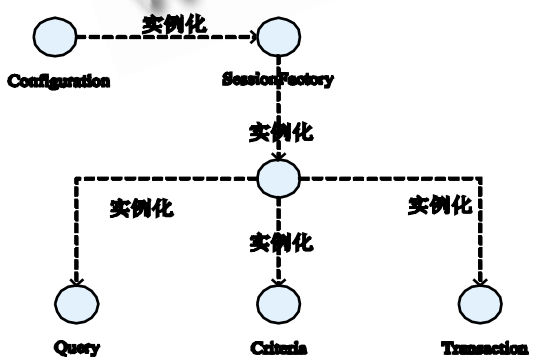

图 1 Hibernate 核心接口

Hibernate 中 Java 对象的状态，下文以代码形式介绍 Hibernate 的生命周期和对象状态的转化。

```
tx.session.beginTransaction();
Teacher c1 = new Teacher("a", new HashSet);
```

通过 new 语句创建了 Teacher 对象，该对象的生命周期开始，此时它还没有被持久化，不处于 Session 的缓存中，没有与任何一个 Session 实例相关联，它处于临时状态。

```
session.save(c1);
```

Teacher 对象由临时状态变为持久化状态，这时位于一个 session 实例的缓存里，与数据库的一条记录关联。

```
Long id = c1.getId();
c1 = null;
Teacher c2=(Teacher)session.load(Teacher.class,id);
tx.commit();
```

Teacher 对象处于持久化状态，通过 session 的 load()或者 get()方法返回的对象处于持久化状态。

```
session.close();
```

session 的 close()方法完成了由持久化状态向游离状态的转变，session 的缓存被清空， Teacher 对象不再与 session 关联。如果 Teacher 对象不再被任何变量引用，就会结束生命周期，它占用的内存可以被 JVM 的垃圾回收器回收。

以上就是一个 Teacher 对象的生命周期及其状态转换过程。

3 Hibernate持久层解决方案的设计与实现

3.1 基于 Hibernate 的持久层解决方案总体设计

基于 Hibernate 的持久层解决方案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，从根本上消除应用程序和关系数据库的耦合。此外数据持久层使用 DAO 工厂和业务逻辑工厂来分别管理 DAO 组件和业务逻辑组件，降低系统的耦合性。该解决方案应用于 B/S 架构的师生交互系统，最大限度的降低了系统内部各模块、子系统间的耦合性，保证持久层的业务逻辑层相对稳定，基本不需要因持久层的调整改变而进行逻辑层的变动。

师生交互系统提供一个师生交流的平台，包括作业管理，在线答疑，在线考试，留言板等多个模块。该解决方案按 DAO 层，业务逻辑层，控制器层的方式分层：DAO 组件被封装在 DAO 层内，提供了 DAO 工厂管理的 DAO 组件；业务逻辑组件被封装在业务逻辑层，由业务逻辑组件进行管理。师生交互系统的整体架构图如图 2 所示。下文对在线考试成绩管理模块在各层的设计与实现予以介绍。

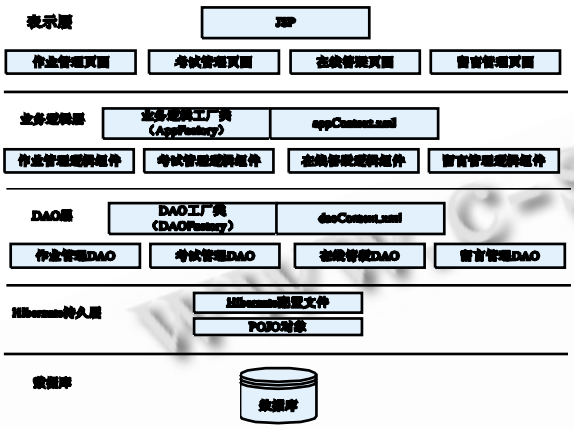

图 2　师生交互系统总体架构

3.2 Hibernate 持久层的设计与实现

Hibernate 是 O/R 映射框架，用持久化实体与数据库进行映射[4]。在线考试成绩管理模块涉及到学生实体，课程实体，成绩实体。实体间存在关联关系。学生实体，和 student 表对应；课程实体，和 course 表对应；　成绩实体，和 score 表对应。其中成绩和课程是多对一关系，学生和成绩是一对多关系。

Hibernate 持久层使用 POJO(Plain Old Java Objects，普通的 Java 对象)和数据库的表进行映射，其实现方式就是普通的 POJO 对象+XML 映射文件，映射文件指定了 POJO 与表的关联关系，以及 POJO 属性和表字段的关系。

下面列出了学生 POJO 的部分代码：

```
public class Student {
    private String sid;
……
    private Set scores = new HashSet(0);
    public Student() {}
    public Student(String sid, String sname,
String classId, String sdepart, String spwd, Set scores)
{
        ……
        }
        public String getSid() { return this.sid; }
        public void setSid(String sid) { this.sid = sid;}
        ……
    }
```

对于学生和成绩一对多的关系，使用集合属性访问关联的持久类，通过 getScores()可获得该学生所有的成绩。

对象关系的映射是由映射文件来说明的。Hibernate-mapping 是映射文件的根元素，这个元素下可以有多个 class 元素，每个 class 元素对应一个持久化类的映射，每一个持久化类要有一个标识属性，通常要指定主键生成策略。另外对象间的管理关系也可以通过 Hibernate 的管理映射来实现。下面是学生持久化类的映射文件和成绩持久化类的映射文件。

学生持久化类的映射文件：

```
<!-- 映射普通属性-->
<property name="name" type="string"/>
……
<!-- 指定集合属性 set，  指定集合属性的表名 -->
<set name="scores" table= “score_table”  >
<!-- 集合属性的外键 -->
<key>
<column name="sid" length="50" />
</key>
<one-to-many class="Score" />
</set>
</class>
</hibernate-mapping>
```

成绩持久化类的映射文件：

```
<hibernate-mapping>
  <class name=" Score" table="score_ table">
     <id name="id">
   <generator class="native" />
  </id>
  <property name="score" type="string"/>
  ……
```

<!-- 用来映射关联的 Student，column 是 Student 的外键列名 -->

```
<many-to-one name=" student" column=" sid" not-null=" true" />
</class>
</hibernate-mapping>
```

3.3 DAO 组件工厂模式的设计与实现

在 J2EE 应用架构的基础上将中间层进一步分为 MVC 中的控制器，业务逻辑组件[5]和 DAO 组件[5]。控制器调用业务逻辑组件的业务逻辑方法处理用户请求，业务逻辑组件依赖 DAO 组件提供的数据库原子操作。DAO 模式的分层方式层次清晰，数据持久层封装于 DAO 层下，不会扩散到业务逻辑，更不会在表示层进行持久层访问。

数据持久层使用数据访问对象(DAO)来抽象和封装所有对数据源的访问。DAO 管理着与数据源的连接以便检索和存储数据，内部封装了对 Hibernate 数据操纵、事务处理、会话管理等，外界依赖于 DAO 的业务组件为其客户端使用 DAO 提供了更简单的接口。采用该模式允许 DAO 调整不同的存储模式，而不会影响其客户端或业务组件，即使将来不再采用 Hibernate 作为关系映射框架，上层客户端也不会受到任何影响。

本系统采用 DAO 组件工厂模式，业务逻辑组件不与 DAO 组件的实现类耦合，只与 DAO 组件的接口耦合。DAO 组件通过 DAO 工厂管理，DAO 工厂负责生成 DAO 组件实例，并维护 DAO 实例。DAO 工厂是一个工厂实例。DAO 模式下包括 DAO 接口，DAO 实现类，DAO 工厂。

定义 DAO 接口，就是定义各个 DAO 所包含的方法，但并不提供具体实现，体现出规则和实现的分离，下面列出了 ScoreDAO 接口的实现：

```
public interface ScoreDao extends Dao
{
Score get(Session sess, String id);
……
List findByStudent(Session sess, Student student);
List findByCourse(Session sess, Course course);
}
```

DAO 组件实现就是为每一个接口提供实现类，下面就是对 ScoreDao 的接口实现 ScoreDaoImpl：

```
public class ScoreDaoImpl implements ScoreDao
{
Score get(Session sess, String id)
{Return (Score)sess.load(Score.class, id);}
……
List findByCourse(Session sess, String courseid);
{return sess.createQuery( "from Score as s where s.course = :id" ).setString( "id" , courseid). list();}
……
}
```

本系统通过配置文件管理系统的 DAO 组件，提供更好的扩展性，当增加其他 DAO 组件时，只需在配置文件中增加 DAO 配置，系统会根据配置文件，创建 DAO 组件的实例。配置文件结构如下：

```
<?xml version= '1.0' encoding= "GBK"?>
<daoContext>
<dao id= "scoreDao"
class= "org.dao.impl.ScoreDaoImpl" />
……
</daoContext>
```

每个 DAO 组件对于一个 DAO 元素，包括 id 和 class 元素，分别是该 DAO 组件的标识和对应的 DAO 组件的实现类。系统中有一个 DAO 工厂实例负责维护 DAO 实例，DAO 工厂保证缓冲池中每一个 DAO 组件只有一个 DAO 实例。

3.4 业务逻辑组件工厂设计与实现

系统的业务逻辑都被封装在业务逻辑层，由业务逻辑组件提供管理。业务逻辑组件也采用工厂模式访问 DAO 模式，而不与 DAO 实现类直接耦合，业务逻辑组件工厂同样也是使用配置文件来初始化所有业务逻辑组件，并将业务逻辑组件放入缓冲池中，让控制器和业务逻辑组件的耦合降低到接口层次。业务逻辑的接口类，实现类以及工厂类代码类似于 DAO 模式。

业务逻辑组件的接口不是直接依赖 DAO 组件实现类，而是依赖于 DAO 接口；只要 DAO 接口不改变，DAO 实现类的改变不会影响业务逻辑组件。

同业务逻辑层一样，控制器与业务逻辑工厂耦合，面向业务逻辑组件的接口编程，不与业务逻辑的实现类耦合，直接依赖业务逻辑工厂和业务逻辑接口；只要业务逻辑组件接口不发生变化，控制器代码就无需改变。

(下转第 127 页)

(上接第 157 页)

4 结语

本文在介绍数据持久化概念后，通过比较现今多种数据持久化技术，选择其中的 Hibernate 数据持久化技术展开研究，通过将此技术与具体应用案例的结合来体现其优越性所在。通过基于 Hibernate 的数据持久层解决方案在师生交互系统的应用，用持久化类来存取数据，大大提高数据存取速度，使开发人员无需编写复杂的 SQL 语句，更专注于业务逻辑的实现，简化了数据库增、删、改、查的开发过程，继承和延续了 J2EE 所特有的可伸缩性和可扩展性。同时通过基于 Hibernate 的 DAO 模式在系统开发中的实现，提高了系统的低耦合性，保证了良好的可扩展性。

参考文献

1 孙卫琴.精通 Hibernate: Java 对象持久化技术详解.北京:电子工业出版社, 2005.77－79.

2 刘艳霞.浅谈 J2EE 项目中的数据持久层设计.沿海企业与科技,2003,28(2):55－56.

3 唐拥政,衡冬梅.基于 Hibernate 的数据持久层关键技术的研究.盐城工学院学报, 2001,16(1):24－27.

4 刘金,徐苏,冯豫华.基于 Hibernate 的 J2EE 数据持久层的设计与实现.计算机与现代化, 2000,21(1):54－56.

5 李刚.整合 Struts+Hibernate+Spring 应用开发详解.北京:清华大学出版社, 2007.325－326.