基于 Hibernate 的数据持久层关键技术的研究*

唐拥政[1],衡冬梅[2]
(1.盐城工学院 现代教育技术中心,江苏 盐城 224003;2.河海大学 计算机及信息工程学院,江苏 南京 210098)

摘　要:Hibernate 是连接 Java 对象模型和关系数据模型的桥梁,本课题的目标是开发一个基于 Hibernate 的信息查询系统,进行项目总体设计的时候,在整体结构中加入一个数据库访问层,为系统中的其它模块的数据库操作统一的函数接口。经过综合比较各种持久化方案,文中选择使用 DAO 封装 Hibernate 的方式实现数据访问层,同时选择 ThreadLocal 模式来管理 Session。

关键词:持久层;Hibernate;DAO;ThreadLocal
中图分类号:TP391　　**文献标识码**:A　　**文章编号**:1671-5322(2006)02-0018-04

Hibernate 是连接 Java 对象模型和关系数据模型的桥梁,它使用数据库和配置文件数据来为应用程序提供持久化服务和对象。Hibernate 的目的是让普通的 Java 对象变成持久化类,以完全实现面向对象的理念,对象的各种属性通过 getter 和 setter 方法访问,对外隐藏实现的细节。Hibernate 不仅可以管理 Java 类到数据表的映射,还提供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,尽量简化开发人员所做的和数据持久化相关的编程任务[1]。

本课题的目标是开发一个基于 Hibernate 的信息查询系统,在这个项目中有大量的对象需要存储到关系数据库中,有一些数据在系统运行过程中还需要被恢复成对象,而且这些对象中包含了大量的数据,如果用传统的方法,为这些对象的存储一条一条的写 SQL 语句,将是一件工作量巨大的工作[2]。所以在进行项目总体设计的时候,特别在整体结构中加入一个数据库访问层,为系统中的其它模块的数据库操作提供统一的函数接口。经过综合比较各种持久化框架,最终选择用 DAO 封装 Hibernate 的方式实现这一数据访问层。本文着重对系统实现过程中的一些关键技术做一些探讨和研究。

1　DAO 模式的实现

持久层的初始设计思想是把需要映射的数据抽象为"值对象"的属性,即"值对象"封装了所有需要存入数据库中的数据项。持久层一方面把"值对象"映射进数据库中的关系表,另一方面把数据库中的关系表映射成为"值对象",并为每一个"值对象"建立一个 O/R 转换的类。

当需要增加一个主机时,业务层的调用如下:

```
AccountDB accountDB = new AccountDB ( )
;
AccountDB. addAccount (account) ;
```

XXXDB 类负责值对象与数据库之间的 O/R 转换,一方面将数据对象转换成数据库中的一条记录,实现数据对象的持久化;另一方面将数据库中的一条记录转换成数据对象提供给业务层,实现数据对象的生成。经过分析,初始时对 XXXDB 类所起作用的命名并不确切,它的职责不仅仅是数据库的操作,而应该是进行数据访问,因此它应该被称为数据访问对象(Data Access Object)。于是,将 XXXDB 类进行重命名(rename),重命名为 XXXDAO,如图 1 所示。

* 收稿日期:2005-11-15
作者简介:唐拥政(1973-),男,江苏建湖县人,盐城工学院讲师,硕士,主要研究方向为网络数据库。

图 1　对象名重构示意图

Fig. 1　Map of Object Name Reconstruction

业务层在调用持久层时,采用的是直接调用 AccountDAO 的构造器。考虑到持久层应该对外提供清晰、良好的接口,因此直接调用构造器的做法需要改进。在这里,考虑采用重构技术中的"静态工厂方法代替构造函数"(replace construct with factory method)。静态工厂方法代码如下:

```
...
/**
静态工厂方法
*/
public static AccountDAO getAccountDAO() {
return new AccountDAO() ;
}
...
```

此时,如果增加一个主机,业务层的调用如下:

```
...
AccountDAO accountDAO = AccountDAO.
getAccountDAO() ;
AccountDAO. addAccount (account) ;
...
```

同理,其它的 XXXDAO 类也都进行同样的重构,用静态工厂方法代替它们的构造函数。

运用重构技术"提取类"(extract class) 创建一个新类,因为该类的功能是生产各种数据访问对象,因此把它命名为 DAOFactory,表明它是数据访问对象的生产工厂。然后,将所有的数据访问对象的静态工厂方法放入其中。在这个新类的设计上引入设计模式中的工厂模式,并且在这里把工厂模式作为这次重构的目标,如图 2 所示。

此时,如果增加一个主机,业务层的调用如下:

```
...
AccountDAO accountDAO = DAOFactory. ge-
tA ccountDAO() ;
accountDAO. addA ccount (account) ;
...
```

单例模式能够确保在整个系统中,一个类只

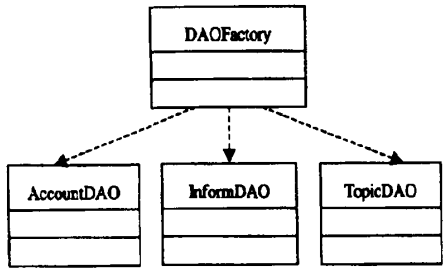

图 2　工厂模式类关系图

Fig. 2　Class Relation Map of Factory Pattern

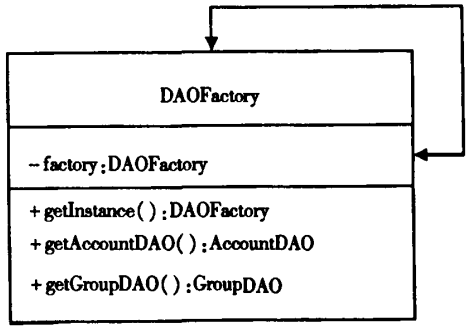

图 3　DAOFactory 类图

Fig. 3　Class Map of DAOFactory

有一个实例,而且它能够自行实例化,并向整个系统提供这个实例。于是,给 DAOFacto ry 类增加一个新方法 getInstance ( )。如图 3 所示。

至此,一个基于单数据源的对象持久化架构基本完成,持久层对外(业务层) 已经形成了一套清晰完整的 API。

2　Session 的管理

Session 是 Hibernate 运作的灵魂,作为贯穿 Hibernate 应用的关键,Session 中包含数据库操作相关的状态信息。如对 JDBC Connection 的维护,数据实体的状态维护等。对 Session 进行有效管理的意义,类似于 JDBCA 程序设计中对于 JDBC Connection 的调度管理。有效的 Session 管理机制,是 Hibernate 应用设计的关键。大多数情况下,Session 管理的目标聚焦于通过合理的设计,避免 Session 的频繁创建和销毁,从而避免大量的内存开销和频繁的 JVM 垃圾回收,保证系统高效平稳运行。

Session 由 SessionFactory 负责创建,Session-Factory 是线程安全的,多个并发线程可以同时访

问一个 SessionFactory 并从中获得 Session 实例。而 Session 并非线程安全,也就是说,如果多个线程同时使用一个 Session 实例进行数据存取,则会导致 Session 数据存取逻辑混乱。

经过综合比较各种 Session 管理方案,我们最终选择 ThreadLocal 模式来管理 Session。

2.1 ThreadLocal 的概念

ThreadLocal 是 Java 中一种较为特殊的线程绑定机制。通过 ThreadLocal 存取的数据,总是与当前线程相关。其实 ThreadLocal 并非一个线程的本地实现版本,它并不是一个 Thread,而是 thread local variable(线程局部变量)。也许把它命名为 ThreadLocalVar 更加合适。ThreadLocal 的功用其实非常简单,就是为每一个使用该变量的线程都提供一个变量值的副本,是每一个线程都可以独立地改变自己的副本,而不会和其它线程的副本冲突。从线程的角度看,就好像每一个线程都完全拥有该变量。

ThreadLocal 和其它所有的同步机制都是为了解决多线程中的对同一变量的访问冲突,普通的同步机制通过对象加锁来实现多个线程对同一变量的安全访问,这时该变量是多个线程共享的,使用这种同步机制需要很细致地分析在什么时候对变量进行读写,什么时候需要锁定某个对象,什么时候释放该对象的锁等等很多。所有这些都是因为多个线程共享了资源造成的。ThreadLocal 从另一个角度来解决多线程的并发访问,ThreadLocal 会为每一个线程维护一个和该线程绑定的变量的副本,从而隔离多个线程的数据,每一个线程都拥有自己的变量副本,从而也就没有必要对该变量进行同步了。ThreadLocal 提供了线程安全的共享对象,在编写多线程代码时,可以把不安全的整个变量封装进 ThreadLocal,或者把该对象的特定于线程的状态封装进 ThreadLocal。

2.2 使用 ThreadLocal 实现对 Session 的管理

具体实现代码如下:

```
import net.sf.hibernate.*;
import net.sf.hibernate.cfg.*;
public class HibernateUtil {
    private static Log log = LogFactory.getLog(HibernateUtil.class);
    private static final SessionFactory sessionFactory;
    private static Connection conn;
    static {
      try {
        sessionFactory = new Configuration().configure().buildSessionFactory();
      } catch (Throwable ex) {
        log.error("Initial SessionFactory creation failed.", ex);
          throw new ExceptionInInitializerError(ex);
      }
    }
    public static final ThreadLocal session = new ThreadLocal();
    public static Session getSession() throws HibernateException {
      Session s = (Session) session.get();
      //假如没有一个可用的线程,开启一个新Session
    if (s == null) {
    conn = DBConnectionManager.getConnection();
    s = sessionFactory.openSession(conn);
      session.set(s);
    }
    return s;
    }
    public static void closeSession() throws HibernateException {
      Session s = (Session) session.get();
      session.set(null);
      if (s != null)
        s.close();
      if(conn!=null)
    DBConnectionManager.returnConnection(conn);
      }
    }
```

其中,DBConnectionManager 是一个 DAO 类,提供连接池的实现,session 是一个 ThreadLocal 类型的对象。在 getSession()方法中,我们使用 session 的 get 方法取出 Session 实例,假如没有一个可用的线程,就通过 conn 连接来创建一个 Session,然后通过 session 的 set 方法将获取的 Session 实例保存。这是 ThreadLocal 的独特之处,它会为

每个线程维护一个私有的变量空间。其实现原理是在 JVM 中维护一个 Map,这个 Map 的 Key 就是当前的线程对象,而 value 则是线程通过 ThreadLocal. set 方法保存的对象实例。当线程调用 ThreadLocal. get 方法时,ThreadLocal 会根据当前线程对象的引用,取出 Map 中对应的对象返回。这样,ThreadLocal 通过以各个线程对象的引用作为区分,从而将不同线程的变量隔离开来。在其他的 Java 程序中,只要通过这个类来获取 Session 实例,我们就可以实现线程范围内的 Session 共享,从而避免了在线程中频繁的创建和销毁 Session 实例。不过要注意在线程结束时关闭 Session。

同时值得一提的是,Hibernate 在处理 Session 时已经内置了延迟加载机制,只有在真正发生数据库操作的时候,才会从数据库连接池获取数据库连接,我们不必过于担心 Session 的共享会导致整个线程生命周期内数据库连接被持续占用。

此外,在我们的 Web 程序中,我们借助了 Servlet2.3 规范中的 Filter 机制,实现线程生命周期内的 Session 管理。Filter 的生命周期贯穿了其所覆盖的 Servlet(JSP 也可以看作是一种特殊的 Servlet)及其底层对象。Filter 在 Servlet 被调用之前执行,在 Servlet 调用结束之后结束。因此,在 Filter 中管理 Session 对于 Web 程序而言就显得水到渠成。

通过在 doFilter 中获取和关闭 Session,在周期内运行的所有对象对此 Session 实例进行重用,保证了一个 HttpRequest 处理过程中只占用一个 Session,提高了整体性能表现。

在实际设计中,大多数情况下 Session 重用做到线程级别一般已经足够,企图通过 HttpSession 实现用户级的 Session 重用反而可能导致其他的问题。凡事不能过火,Session 重用也一样。

3　结束语

要开发一个完备的信息查询系统的数据库访问层是一件相当复杂的工作,因为在具体的应用中,对象之间的关系是非常复杂多变的,我们不可能针对每种对象的情况都适用,只能尽可能的完善设计模式,使它更加通用。

参考文献:

[1] 胡建华. 基于 Hibernate 开发与数据库无关的系统[J]. 计算机与现代化,2005,(9):50-53.
[2] 董洪杉,窦延平. 利用 Hibernater J2EE 数据持久层的解决方案[J]. 计算机工程,2004,12(30):17-18.
[3] 董刚. 使用 Hibernate 进行开发[J]. 湖南冶金职业技术学院学报,2004,12(4):313-315.
[4] 高昂,卫文学. 基于 Hibernate 与 Struts 框架的数据持久化应用研究[J]. 计算机应用,2005,12(25):2817-2818.
[5] 宋汉增,沈琳. 利用 Hibernate 对象持久化服务简化 Java 数据库访问[J]. 计算机应用,2005,12(23):135-137.

Research on the Key Technologies of Persistent Layer Based on Hibernate

TANG Yong-zheng[1], HENG Dong-mei[2]

(1. Department of The Modern Educational Technology Center, Yancheng Institute of Technology, Jiangsu Yancheng 224003, China;
2. College of Computer and Information Engineering of HoHai University, Jiangsu Nanjing 210098, China)

Abstract: Hibernate is the bridge of the java object model and the relational data model. The goal of this object is to develop an information inquiry system based on hibernate. In the time of overall design, data access layer was joined in the whole structure to provide a uniform function interface for the database operations of other module. In this paper, hibernate was encapsulated in DAO to implement the data access layer, and ThreadLocal model was chosen to manage session.

Keywords: hibernate; persistent Layer; DAO; ThreadLocal