

基于标注和反射机制的通用 JavaBean 持久化工具类

檀 明, 许 强, 李 丛

(合肥学院 网络与智能信息处理重点实验室, 安徽 合肥 230601)

摘 要: 作为一种可复用的组件, 使用 JavaBean 有利于实现模块化和分层的开发. 使用标注和反射技术设计并实现一种针对 JavaBean 的持久化通用工具类, 对 JDBC 操作进行了更进一步的封装, 避免了开发人员在 JavaBean 实现中使用大量的 SQL 语句通过 JDBC 的方式对数据库进行操作, 可以有效地提高开发效率.

关键词: 持久化; 反射机制; 标注机制

中图分类号: TP393

文献标识码: A

文章编号: 1000-7180(2010)04-0029-04

A General Persistence Tool Class for JavaBean with Java Reflection and Annotation

TAN Ming, XU Qiang, LI Cong

(Key Laboratory of Network and Intelligent Information Processing, Hefei University, Hefei 230601, China)

Abstract: As a reusable component, application of JavaBean facilitates modularization and hierarchical classification. Based on reflection and annotation, a general persistence tool class for JavaBean was designed and realized, using it, the developers can avoid writing many SQL Sentences to operate the database through JDBC in the realization of JavaBeans. And it can improve the development efficiency evidently.

Key words: persistence; reflection; annotation

1 引言

基于 Java 的数据库应用开发一般采用将 JDBC 封装到不同的 JavaBean 中, 实现数据库表到业务对象映射, 通过调用这些 JavaBean 以完成对数据库访问. 因此, 针对实际应用需要编写大量的 JavaBean 以封装数据库表的字段和对数据库表的操作, 这种设计方式存在着明显的不足. 其一, 增加了工作的重复性, 间接增加了开发工作量; 其二, 代码的修改和维护工作量大.

JDK 在 java.lang.reflect 包中提供了反射 (reflection) 的能力, 通过调用 object.getClass() 方法获得 object 对应的 Class 对象, 再利用返回的 Class 对象调用相应方法实现对 object 的属性和方法的动态访问. 在 JDK1.5 中增加的标注 (annotation) 能为程

序元素 (如类、方法、属性、参数、本地变量、包) 关联任何信息. 关联信息能够通过 Java 反射机制被访问.

文中利用 Java 的标注和反射机制, 设计并实现了一种针对 JavaBean 的通用持久化工具 (Persistence Tool), 提出针对 JavaBean 的一种简便的持久化方案.

2 通用持久化工具类的设计

针对同时要封装数据库表字段和操作方法而造成 JavaBean 设计复杂的问题, 将 JavaBean 中业务数据的新增、修改、删除、加载以及查询方法从实现中剥离, 改由通用持久化工具类 (Persister) 完成, 从而降低了 JavaBean 的实现复杂度. Persister 类对传入的 JavaBean 的实例对象 beanInstance, 利用反射对

其字段名和字段值进行动态访问,并生成相应的 SQL 语句,通过执行 SQL 语句完成对数据库表的 CRUD(增删改查)操作.主要属性和方法定义如下^[1-2]:

```
public class Persister{
    private Connection connection;
    public Persister(Connection connection);
    public void insert(Object beanInstance );//提取 beanInstance 中数据并插入数据库表
    public void insertMany(List beanList); // 批量插入 bean 列表中的对象到数据表中
    public void delete(Object beanInstance);// 删除在数据库表中 beanInstance 所对应的行
    public void deleteMany(List beanList);// 批量删除数据库表中 bean 列表中所有 bean 对象所对应的行
    public void update(Object beanInstance);// 提取 beanInstance 中数据并更新数据库表中对应记录
    public void updateMany(List beanList);// 批量更新数据库表中 bean 列表中所有 bean 对象对应的行
    public Boolean load(Object beanInstance);// 按 beanInstance 指定的主键值从数据库表中提取数据,并填充 beanInstance 的属性
    List<?> query(Class<?> clazz, ConditionHolder conditionHolder);//按照条件查询,返回符合查询条件的 JavaBean 的对象列表,ConditionHolder 封装了查询条件
    String getTableName(Class<?> beanClass);// 返回 bean 对应数据库表名
    String[] getIdName(Class<?> beanClass);// 返回 bean 对应数据库表的主键字段名
}
```

3 JavaBean 的设计

JavaBean^[3]主要完成对数据库字段的封装和对数据合法性的验证,对数据的持久化操作由 Persister 负责实现. Persister 类要针对具体的 JavaBean 动态产生 SQL 语句,这就要求能获得 JavaBean 所封装的数据库表名及表的主键字段名.为此,定义 ID annotation 和 Table annotation 分别实现将主键字段名、数据库表名与 JavaBean 信息关联,相对于使用配置文件,利用标注描述 JavaBean 映射信息的方式更加简洁,维护成本更低.

3.1 标注(annotation)的定义

```
@Target(ElementType. TYPE)
@Retention(RetentionPolicy. RUNTIME)
public @interface Id{String[] value();}
```

在以上 Id 标注的定义中,@Target 说明 Annotation 所修饰的目标类型,ElementType. TYPE 用于

对类、接口、枚举、Annotation 类型的修饰. @Retention 设置为 RetentionPolicy. RUNTIME 说明该 Annotation 被编译在 class 文件中,能够在运行时通过反射技术动态访问. Id 标注仅仅包含一个方法 value,其类型为字符串数组,对应 JavaBean 的主键字段名(支持多主键). Table 标注的定义与 Id 标注类似,其 value 方法为字符串类型,bean 与表名一一对应.

3.2 JavaBean 的定义

以表 1 的 t-book 为例,其对应的 JavaBean 定义如下:

```
@Id({"bookId"}) // 标注数据库表的主键字段名
@Table("t_book") // 标注数据库表名
public class BookBean implements Serializable{
    private String bookId;
    private String title;
    private Integer amount;
    public Boolean validateInsert(); // 对新增数据操作的合法性验证
    public Boolean validateDelete(); // 对删除数据操作的合法性验证
    public Boolean validateUpdate(); // 对修改数据操作的合法性验证
    ...// 省略属性字段的 Getters and setters 及其他方法
}
```

表 1 t-book 的数据表结构

字段名	字段类型	主键
book-Id	VarChar(20)	Y
title	VarChar(50)	
amount	Numeric(6,0)	
price	Numeric(6,2)	
...	...	

4 Persister 类主要方法的实现

4.1 Annotation 访问方法的实现

JavaBean 的映射信息通过 Annotation 关联后,会存放在编译的 Class 类中,可以通过反射来获取.针对 4.2 节中所给出的 BookBean, Persister 类^[4]的 getIdName(Object beanInstance) 和 getTableName(Object beanInstance) 方法利用反射技术访问 Id 标注和 Table 标注,得到关联的主键字段名 bookId 和数据库表名 t-book. 以 getIdName 方法为例,首先通过调用 getClass() 方法获得实例对象 beanInstance 对应的 Class 对象 clazz,使用无限制通配符的泛型声明 Class<?> 可确保方法适应于任何给定的对

象.再通过调用 `clazz.getAnnotation(Id.class)` 得到与 `BookBean` 关联的 `ID` annotation,利用 `idAnnotation.value()` 获得 `BookBean` 类的主键字段名: `bookId`.主要实现代码如下:

```
...
Class <? > clazz = beanInstance . getClass (); //
beanInstance 为 BookBean 类的实例
Id idAnnotation = clazz.getAnnotation(Id.class);
String[] idNames = idAnnotation.value(); // 获得
12BookBean 类的主键字段名数组:{"bookId"}
...
```

4.2 insert 方法的实现

调用 `getClass()` 方法获得实例对象 `beanInstance` 对应的 `Class` 对象 `clazz`,再通过反射技术动态分析 `beanInstance` 的字段名和字段值并生成 `insert` 语句.默认情况下 `JavaBean` 的字段属性与其封装的数据库表字段同名,且提供 `get/set` 方法.为了避免生成 `Insert` 语句时针对不同类型的字段要采取不同的赋值处理方式而带来的实现复杂性,利用 `Field` 类的 `get` 方法获得 `beanInstance` 的字段值并存入字段数组 `values` 中.由于字段属性为 `private`(对外是不可访问),调用 `Field` 类的 `get` 方法前应先使用 `setAccessible(true)` 方法设置字段属性为可访问.以字段值数组 `values` 为传入参数,利用 `PreparedStatement` 接口对象的 `setObject` 方法(自动进行类型转换)为各参数赋值,最后通过 `preparedStatement` 接口对象的 `executeUpdate()` 方法完成数据插入.主要语句如下:

```
...
StringBuffer sbSql = new StringBuffer(); // 存储 insert
sql 语句
List values = new ArrayList(); // 保存对象的字段值的
列表
sbSql.append("insert into");
sbSql.append(getTableName(beanInstance.getClass()));
sbSql.append(" ( ");
Field[] fields = beanInstance.getClass().getDeclared-
Fields();
//通过反射获得 bean 的属性数组
for (int index = 0; index < fields.length; index++) {
Field field = fields[index];
if (Modifier.isStatic(field.getModifiers()) || Modifier.is-
Transient(field.getModifiers())) {
continue; //如果字段属性具有 static 或者 transient 修饰
符,则不映射到数据库表
}
```

```
sbSql.append(field.getName());
field.setAccessible(true); //设置属性可访问
values.add(field.get(beanInstance)); //字段值暂存到值
列表 values 中
if (index < fields.length - 1) { sbSql.append(","); }
}
sbSql.append(") values(");
for (int index = 0; index < values.size(); index++) {
sbSql.append("?");
if (index < values.size() - 1) { sbSql.append(","); }
}
sbSql.append(""); //占位符加入完毕
PreparedStatement statement = connection. pre-
pareStatement(sbSql.toString(),
Result Set. TYPE_SCROLL_INSENSITIVE,
Result Set. CONCUR_UPDATABLE
for (int index = 0; index < values.size(); index++) {
statement.setObject(index + 1, values.get(index));
//通过字段值数组 values 为所有占位符参数赋值
}
statement.executeUpdate(); //执行更新操作
```

4.3 delete 方法的实现

利用与 `Insert` 方法类似步骤的产生相应的 `delete` 语句并生成 `PreparedStatement` 对象,由于 `delete` 方法要通过主键值确定删除的记录,通过 `getIdName` 方法确定主键字段名,同样利用 `PreparedStatement` 接口对象的 `setObject` 方法为参数动态赋值,最后通过 `PreparedStatement` 对象的 `executeUpdate` 实现对数据表的删除操作.

```
...
StringBuffer sbSql = new StringBuffer(); //存储 sql
delete 语句
sbSql.append("delete from");
sbSql.append(getTableName(beanInstance.getClass()));
sbSql.append(" where");
// 从类的 Annotation 中提取主键字段并生成对应
where 语句
Id idAnnotation = beanInstance.getClass().getAnnotation(Id.class);
String[] idNames = idAnnotation.value();
for (int index = 0; index < idNames.length; index++) {
Field idField = beanInstance.getClass().
getDeclaredField(idNames[index]); //获得 id 属性
idField.setAccessible(true); //设置属性为可访问
sbSql.append(idNames[index]);
sbSql.append(" = ?"); //在 sql 语句中加入占位符
```

```

values.add(idField.get(beanInstance)); //将字段值暂存
到值列表 values 中
if (index < idNames.length - 1) {sbSql.append(" and
");}
}
PreparedStatement statement = connection. pre-
pareStatement( sbSql.toString(),ResultSet. TYPE-SCROLL-IN-
SENSITIVE,
ResultSet. CONCUR-UPDATABLE);
for (int index = 0; index < values.size(); index++) {
statement.setObject(index + 1, values.get(index));
}
statement.executeUpdate();

```

4.4 update 方法的实现

先利用与 insert 方法同样方式产生 update 语句和对应的 PreparedStatement 对象, 同样使用 PreparedStatement 对象的 setObject 方法为参数赋值, 最后通过 PreparedStatement 接口对象的 executeUpdate() 完成对数据库表的更新.

4.5 load 方法的实现

利用与 insert 方法相同的方式生成 select 语句和对应的 PreparedStatement 对象, 再以 beanInstance 对象所包含的主键属性的值为查询条件, 按照主键值查询出数据表中对应的行记录, 再将 ResultSet 对象中的结果数据赋值填充到 beanInstance 对象的其他属性中去. 使用包含主键的对象作为参数, 而不是像多数开源框架那样使用主键值作为参数, 从而更好地支持具有多主键的 JavaBean 对象.

4.6 query 方法的实现

利用与 insert 方法相同的方式生成 select 语句和对应的 PreparedStatement 对象, 再通过解析 ConditionHolder 对象中所封装的查询条件, 生成对应的 where 子句, 最终执行类似 select * from where order by 的 sql 语句, 获得结果集之后通过读取结果集元数据 (Metadata of ResultSet) 进行逆向的自动生成 JavaBean 过程, 最终得到符合条件的 JavaBean 对象列表. ConditionHolder 对查询条件进行了对象化的封装, 用户不必再拼接 SQL 字符串, 而是使用更加接近关系代数的表达方式来定义条件, 多个条件可以任意连接以表示复杂的条件. 限于篇幅原因, 对此文中不做更详细阐述.

5 应用实例

JavaBean 中字段属性值的修改即可用代码赋值

方式实现, 也可利用中间层的表单处理类以自动化的方式完成, 其基本原理为对 JavaBean 中的 setXxx() 方法时, 自动寻找表单的对应字段 xxx, 如果找到, 就利用反射调用此方法, 将对应的字段值赋给 JavaBean. 因篇幅所限, 对中间层的表处理类的设计本文从略, 而以代码赋值方式针对文中 3.2 节所给出的 BookBean 为例说明.

```

public class PersisterTest {
static Connection connection;
static Persister persister;
public static void main(String[] args) throws Exception {
...
connection.setAutoCommit(false);
persister = new Persister(connection);
BookBean book = new BookBean();
book.setBookId("978-7-113-07950-5");
...// 获取并设置 book 对象的其他属性
persister.insert(book);
...// 获取并设置 book 对象的其他属性
persister.update(book);
BookBean bookInstance = new BookBean();
bookInstance.setBookId("978-7-113-07950-5");
persister.load(bookInstance); //按照 bookInstance 中包含的主键值, 从数据库读取数据并填充 bookInstance
persister.delete(bookInstance); //删除 bookInstance 所对应的数据表中的行记录
ConditionHolder holder = new ConditionHolder();
Condition c1 = new GreaterThan("title", "h"); //条件 1 为 title 字段大于 "h"
Condition c2 = new LessThan("amount", 10); //条件 2 为 amount 字段小于 10
Condition c3 = new And().add(c1).add(c2); //条件 3 为条件 1 并且条件 2
holder.setCondition(c3); //设置条件 3 为查询条件
holder.setOrder(new Order().addDesc("bookId")); //设置按照 bookId 字段降序排列
List queryResult = persister.query(BookBean.class, holder); //按照条件查询 BookBean 对应的数据库表, 查询结果 List 包含符合条件的 BookBean 类的实例对象
connection.commit();
}
}

```

6 结束语

文中利用 Java 的反射和标注机制, 设计并实现了一种针对 JavaBean 的自动持久化通用工具

(下转第 37 页)

性和可行性.

参考文献:

- [1] Vapnik V N. 统计学习理论[M]. 许建华, 张学工, 译. 北京: 电子工业出版社, 2004.
- [2] 徐勋华. 支撑向量机的多类分类方法[J]. 微电子学与计算机, 2004, 21(10): 149-152.
- [3] Suykens J A K, Vandewalle J. Least squares support vector machine classifiers [J]. Neural Processing Letter, 1999, 9(3): 293-300.
- [4] 王海峰. 最小二乘支持向量机的一种稀疏化算法[J]. 计算机工程与应用, 2005, 43(33): 68-70.
- [5] Wu G, Chang E Y. Class-boundary alignment for imbalanced dataset learning[C]//Workshop on Learning from Imbalanced Datasets (ICML 2003). Washington, DC, 2003: 49-56.
- [6] 刘爽, 贾传炎, 陈鹏. 一种自动选择参数的加权支持向量机算法[J]. 计算机工程与应用, 2006, 42(2): 64-66.
- [7] 孔锐. 一种快速支持向量机增量学习算法[J]. 控制与决策, 2005, 20(10): 1129-1132.
- [8] 姚全珠. 基于最小二乘支持向量机的非平衡分布数据分类[J]. 计算机工程与应用, 2008, 44(5): 167-170.
- [9] Pelckmans K, Suykens J A K, Van T Gestel. et al. LS-SVMLab1. 5Toolbox[EB/OL]. (2003-10-11)[2009-07-21]. <http://www.esat.kuleuven.ac.be/sista/lssvm-lab>.
- [10] UCL Machine Learning Group. Elena database[EB/OL]. [2009-07-15]. <http://www.dice.ucl.ac.be/neural-nets/Research/Projects/ELENA/elena.htm#stuff>.

作者简介:

赵会女, (1982-), 硕士研究生. 研究方向为模式识别与智能系统.
黄景涛男, (1977-), 博士, 副教授. 研究方向为数据挖掘和智能优化算法及应用.
谈书才男, (1984-), 硕士研究生. 研究方向为模式识别与智能系统.

(上接第32页)

(Persistence Tool)类, 给出了针对 JavaBean 一种简便的持久化方案. 在此基础上构建数据库应用系统, 可简化代码编写任务, 加快系统开发进程. Hibernate、Toplink 等开源框架虽然也提供了类似功能, 但都比较复杂^[5]. 使用本工具类不需要外部配置文件, 比较适合于中小型项目的开发. 至于由于引入反射技术而造成程序性能下降的问题, 可通过字节码增强技术与适当的缓存来改善, 限于篇幅不再赘述.

参考文献:

- [1] 唐姗, 赵文耘. 基于反射的动态软件体系结构实现[J]. 微电子学与计算机, 2006, 23(9): 32-37.
- [2] 吴卫平, 王丽芳, 蒋泽军. 基于 ORM 的数据持久层框架

研究[J]. 微电子学与计算机, 2008, 25(7): 188-193.

- [3] 张义. Java5 程序员开发手册[M]. 北京: 北京希望电子出版社, 2007.
- [4] Holzner S. Java 技术内幕[M]. 马朝晖, 译. 北京: 机械工业出版社, 2002.
- [5] 杨飞, 袁建华. Hibernate 框架在权限管理系统中的应用[J]. 微电子学与计算机, 2007, 24(2): 206-208.

作者简介:

檀明男, (1974-), 博士研究生, 讲师. 研究方向为数据库与多媒体技术、工业控制网络.
许强男, (1972-), 硕士, 讲师. 研究方向为软件工程、计算机网络.
李丛男, (1985-), 助教. 研究方向为软件工程.