一种实用的对象持久化框架

古思山 赵黎阳 李师贤
(中山大学信息科学与技术学院 广州 510275)

摘 要 面向方面编程分离了核心关注点和横切关注点,提供了模块化横切关注点的机制,很好地解决了面向对象技术处理横切关注点时存在的代码散布和代码纠缠问题。和日记记录、安全验证等业务一样,持久化也被认为是经典的横切关注点业务,适合用AOP来实现。分析了现有的基于AOP的持久化实现,发现大部分实现过于追求AOP要求的obliviousness特性,而在功能或性能上难以满足现实应用的需求。探讨了持久化的特点及持久方面化的机制,提出了一套实用的基于AOP的持久化框架。该框架保持了面向对象持久化技术的功能及性能,同时又具有方面化持久业务所带来的更高的可重用性、可维护性及可移植性。

关键词 面向方面,持久化,AOP,AspectJ
中图法分类号 TP311 **文献标识码** A

Practical Framework of Object Persistence

GU Si-shan ZHAO Li-yang LI Shi-xian
(School of Information Science and Technology, Sun Yat-sen University, Guangzhou 510275, China)

Abstract By separating cross-cutting concerns from core concerns and supplying additional mechanisms to modularize cross-cutting concerns, aspect oriented programming gracefully solves the problem of code scatting and code tangling coming across when object oriented technique deals with cross-cutting concerns. Just like logging and secure authentication, persistence is considered as a kind of classic cross-cutting concerns and suited to be dealt with using AOP. After analyzing state-of-the-art persistence frameworks or implementations, we found most of them cared too much about the obliviousness property which is inherent in AOP and hard to meet the need of actual applications either in functionality or performance. A practical framework of object persistence was proposed after probing into characters of persistence and mechanisms of aspectizing persistence. The framework not only preserved functionality and performance achieved by object oriented solution, but also wined higher reusability, maintainability and portability by aspectizing persistence.

Keywords Aspect-oriented, Persistence, AOP, AspectJ

1 引言

面向方面编程(Aspect-Oriented Programming, AOP)[1]是一种新型程序设计范式,其构建在传统的面向对象程序设计方法之上,并基于关注点分离思想,提供了模块化横切关注点的机制,很好地解决了面向对象程序设计方法处理横切关注点时存在的代码散布(code scattering)和代码纠缠(code tangling)问题,使程序具有更好的模块性和结构性。

在程序设计阶段,AOP将系统功能分解为核心关注点和横切关注点。核心关注点实现系统的业务逻辑,是系统功能的主体;横切关注点则实现系统中相对独立的功能,这些功能往往分散于整个系统中,如日志记录、安全验证、事务管理、错误处理、持久化等。在实现阶段,核心关注点沿用传统的面向对象技术实现,横切关注点则封装成独立的模块(方面,Aspect)。Aspect不仅描述了横切关注点的功能,还描述了横切关注点和核心关注点的联系。最后通过编织(weaving)机制将这两大部分整合成完整的系统。

AOP构筑在面向对象技术之上,其很多实现也就直接构建在现有的面向对象语言上,如构建在Java之上的AspectJ[2],AspectWerkZ[3],JBoss AOP[4],及构建在C++之上的AspectC++[5],FeatureC++[6],以及构建在.Net上的Aspect.Net[7],Aspect#[8]。其中AspectJ是目前最成熟、最稳定也是最流行的AOP工具。

持久化是指在程序执行过程中从非易失性存储设备中存取数据。如果程序没有持久化能力,则数据只能保存在内存,作暂时存储。在程序执行完毕或关机后,这些数据也将随之消失。非易失性存储设备从硬件上来看包括硬盘、磁盘、光盘等,从软件上来看可以是一般的文件,也可以是专门用于存储数据的数据库系统。其中前者又被称为是轻量级持久化,后者为重量级持久化。

对象模型和关系模型分别是主流的软件开发方法和数据库系统的数据模型。对象/关系映射(Object-Relational Map-

到稿日期:2009-09-02 返修日期:2009-12-30 本文受广东省自然科学基金自由申请项目资助。
古思山 男,博士生,主要研究方向为形式语义学、AOP、MDA等,E-mail:sishangu@gmail.com;赵黎阳 女,硕士生,主要研究方向为AOP、持久化等;李师贤 男,教授,博士生导师,主要研究方向为形式语义学、软件开发方法学。

ping,ORM)[9]提供了对象模型和关系模型相互转换的支持。通过 ORM,编程人员可以透明地操作对象的存取,不必编写关系数据库存取数据相关的代码(如 JDBC 相关代码)。JDO[10],EJB[11,12],Hibernate[13]都是比较成熟的 ORM 工具。

和日志记录、安全验证一样,持久化也被认为是经典的横切关注点业务,适合用 AOP 来实现。文献[14-18,20,24,25]分别提出了一套基于 AOP 的持久化框架。本文分析了这些框架后,发现它们大部分过于追求 AOP 要求的 obliviousness[19]特性,而在功能、性能、适用性及可使用性上考虑较少,难以满足现实应用的需求。本文综合考虑了这些框架的机制,提出了一套实用的基于 AOP 的对象持久化框架。本框架在具有较强的功能、性能的同时,还具有较好的可使用性、可重用性、可维护性和可移植性。

2 相关研究

较早进行持久性方面化研究的是 Awais Rashid 团队[14-18]。他们的目标之一是探讨持久性能否方面化;如果能方面化,那么方面化后的持久框架能否具有高可重用性,并满足 AOP 的 obliviousness 特性。该特性具体体现为业务逻辑编程人员完全不知道有持久化方面的存在,业务逻辑代码也完全没有持久化相关的任何信息。满足 obliviousness 特性的实现最大限度地降低了系统的耦合程度,具有很高的可复用性、可维护性及可移植性。他们基于 AspectJ 构建了一套持久化框架,该框架引入持久化根类(PersistentRoot),以区别持久化对象与非持久化对象。持久类必须继承类 PersistentRoot,这项工作可以通过 AspectJ 中方面定义的 Inter-type Declaration 功能来实现,编程人员不必为每个持久化类添加继承类 PersistentRoot 相应的代码。PersistentRoot 只封装了一个属性——isDeleted,及相应的方法——delete 和 isDeleted,用来标识对象的删除。持久化框架通过持久对象状态的变化来触发数据库的插入和更新操作。具体地,通过拦截持久对象的 New 操作和 Setter 方法来触发数据库的插入和更新操作。而删除操作则是通过拦截对象从持久化根类继承的 delete 方法来触发。持久化根类定义了 delete 方法,其方法体是空的,并没有具体的操作代码。持久类在继承持久化根类时也不必重写(overriding)该方法。因为 delete 方法只是用作锚点,标识删除切入点。具体的删除操作由框架中的方面负责。

为了支持持久化中的查询操作,框架引入了接口 PersistentData。PersistentData 提供了查询相关的几个方法。与数据库查询操作相关的类需要实现接口 PersistentData,通过调用该接口定义的方法来标识查询切入点。同样,这些类实现该接口时也不必具体实现该接口定义的方法,实现为空方法体即可。

和查询操作的实现相似,该框架引用了其他接口来实现支持数据库事务和元信息操作。该框架在功能上对持久化的支持比较完备,在结构设计上合理利用相关设计模式,使得整个框架具有较高的可重用性,而且部分满足了 AOP 要求的 obliviousness 特性。业务逻辑编程人员只需对持久化框架有一定程度的了解,就可以透明地存取对象。

由于 Awais Rashid 团队的主要目标并不是构建一套能用于现实应用的持久化框架,该框架并没有考虑太多实用性相关的因素。在面向对象程序设计中,往往会频繁调用对象的 New 方法来创建对象及 Setter 方法来改变对象的状态。单纯地通过拦截持久对象的 New 操作和 Setter 方法来触发相应的数据库操作,将会频繁访问数据库,导致系统效率低下。另外,编程人员在调用持久对象的 New 操作创建对象时,可能只是暂时使用该对象,并不希望持久化该对象。同样,在调用持久对象的 Setter 方法时,也可能只是做暂时更新,并不希望更新至数据库。这就使得该框架在适用性上大打折扣。同时在实现上,该框架强迫持久类继承类 PersistentRoot,这也给只支持单继承的语言(如 Java)带来一些麻烦。

为了更好地满足持久方面化的 obliviousness 特性,文献[20]引入了持久容器(persisting containers)和路径表达式切入点(path expression pointcuts)[21]。持久容器其实是类 PersistedList 的一个实例,由方面负责管理。持久容器为容器内的每个对象提供持久化服务:当对象添加进持久容器时,系统将为该对象提供持久服务;当对象从持久容器移出时,系统将不再为该对象提供持久服务。系统通过持久容器区分持久化对象和非持久化对象,避免了对持久化对象的类型的限制,同时可以动态地将持久对象改为非持久对象。反之亦然。路径表达式切入点是一种基于路径表达式[22]的切入点模型。在方面定义时,该模型可以获取对象引用图的非局域性对象信息,以满足存储的 persistence by reachability[23]特性。

文献[20]提出的框架虽然在持久方面化的 obliviousness 特性有一定的特色,但是该框架还处于发展初期,很多因素还没有考虑。比如对象添加进及移出持久容器的时机和方法、具体持久化操作的实现以及路径表达式切入点模型的成熟性等。可以说,该框架是作者满足持久方面化的 obliviousness 特征的初步构想,离应用还有一段较大的差距。

北京大学的陈兴润、梅宏等提出了一种对象/关系映射隐式持久化框架 POD(Persistence On Demand)[24],通过对象的状态变化来触发持久化操作。当需要持久化的对象执行 New 操作或 Setter 方法时,触发数据库的插入或更新操作。POD 框架能够隐式持久化对象,是指在应用系统中不需要出现任何与持久化相关的代码,由 POD 框架将封装后的持久化代码注入到原系统中,用户通过配置持久化策略,即可实现对象持久化,自动保持持久化实例与数据库数据之间的同步。而且,POD 框架通过持久化策略的动态重配置,实现运行时刻持久化关系的创建、更新和删除,实现动态持久化。隐式持久化不仅能够在开发系统时完全将业务逻辑和持久化操作分离出来,而且能够在升级原有系统时,将没有持久化功能的系统在完全不修改原有系统的前提下扩展成含有持久化功能的系统。

POD 框架真正实现了 AOP 要求的 obliviousness 特性,在这个意义上可以说是持久性方面化的完美实现。POD 框架最小化了业务逻辑与持久化操作的耦合程度,具有非常高的可重用性、可维护性和可移植性。遗憾的是,POD 框架支持的持久化操作只包括对象的插入与更新操作,并不支持查询、删除以及数据库事务等操作。持久化操作中的增、删、改、查等操作密不可分,不能完全支持这些操作将极大地限制其在现实系统中的应用。POD 框架与 Awais Rashid 框架的机制相近,也同样存在频繁访问数据库、系统效率低下等问题。

开源项目 JPA[25] 基于 AOP 实现了 POJO(Plain Old Java Objects)类到预定义的数据库表的自动映射。使用该框架，编程人员不必编写 JDBC 代码，只需用很少的代码就可以完成对象的持久化。JPA 为每个持久化 POJO 类增加几个持久化相关的空方法(createObject，updateObject 等)，用于标识持久化操作的切入点和持久化操作的类型。与传统的持久化实现相同的是，持久化操作都需要显示调用特定方法来实现；区别在于这些方法都是空方法，不包含任何持久化处理的代码。具体的持久化操作由相应的持久化方面实现。持久化方面拦截 POJO 新添加的几个方法，完成相应的持久化操作。

JPA 支持增、删、改、查等多种持久化操作，能较轻易地扩展至支持数据库事务、数据库元信息等操作(目前 JPA 还不支持)。而且其方法简单、直接，可使用性极高。缺点在于为每个持久化 POJO 类添加几个方法较为繁琐，代码重复较多，且持久化需要显式声明，失去隐式持久化的诸多优点。

Rashid 的框架和 POD 框架将持久对象的 New 方法和 Setter 方法作为持久化新增操作和修改操作的切入点，虽然其在满足持久方面化的 obliviousness 特性上有所提高，但也牺牲了程序的性能及程序的适用性。反而，JPA 项目为每个持久化类增加几个空方法，作为持久化操作的切入点，虽然降低了对持久方面化的 obliviousness 特性的满足程度，但是获得了良好的适用性和可使用性，也保持了程序相对于传统持久化实现的功能和性能。本文通过改进 JPA 项目的持久化框架，提出了一种实用的基于 AOP 的对象持久化框架。

3 持久化框架

3.1 持久化切入点

AOP 分离了核心关注点和横切关注点，分别进行封装，然后通过编织将这两大部分整合。编织时需要核心关注点和横切关注点的关联信息，这些信息由方面中的切入点来描述。AOP 的一些经典应用，如日志记录、安全验证、错误处理等，切入点比较容易确定。日志记录和安全验证通常发生在某些类的某些方法执行之前或之后，错误处理则常发生在某些异常抛出后。大多数 AOP 实现(如 AspectJ)的切入点模型都支持这类切入点的描述。持久化操作则复杂许多。首先，持久化操作和业务逻辑结合紧密，何时存储对象、何时删除对象及查找哪些对象，往往由业务逻辑编程人员决定；其次，持久化操作种类较多，包括增、删、改、查等几种操作，在描述切入点时还需要描述做哪种持久化操作。这些都使得现有的切入点模型难以合理地描述持久化的切入点。JPA 通过为持久类添加几个空方法，作为持久化的切入点，为持久化的切入点描述提供了一种新的思路。如图 1 所示，持久类 Department 添加了 getObjects()，createObject()，updateObject()，deleteObject() 等方法，需要做持久化操作时，直接调用这些方法即可。

```
import java.ucil.*;
public class Department{
  public String departmentName;
  public double departmentGid;
  public void getObject(double departmentGid){}
  public void getObjects(){}
  public void createObject(){}
  public void updateObject(){}
  public void deleteObject(){}
}
```

图 1 JPA 中的持久类示意

这种方法虽然简单、实用，但是也存在一些缺陷。首先，破坏了对象的封装性，这些添加的方法在责任归属上并不属于该对象；其次，为每一个持久类添加这些方法将导致代码大量重复。可以利用面向对象的思想进行抽象，将这些共同的方法泛化为父类或接口，如图 2、图 3 所示。

```
public class PersistenceRoot{
  public void getObject(Object id){};
  public void getObjects(){};
  public void createObject(){};
  public void updateObject(){};
  public void deleteObject(){};
}
```

图 2 泛化为父类

```
public interface PersistenceInterface{
  public void getObject(Object id);
  public void getObjects();
  public void createObject();
  public void updateObject();
  public void deleteObject();
}
```

图 3 泛化为接口

泛化为父类时，持久类需要继承该父类，之后即可直接调用这些方法了。泛化为接口时，持久类需要实现该接口。此时，并不需要具体实现这些方法，只需提供空方法体的实现即可。大部分集成开发环境都支持这类代码的自动生成，这在很大程度上减轻了编程人员的负担。

和 JPA 相比，泛化为接口时，虽然减轻了编程人员的负担，但是并没有降低代码的重复度；泛化为父类时，虽然降低了代码的重复度，但浪费了一个继承资源，这给单继承语言会带来一些麻烦。为了避免上述问题，可以效仿 JDK 工具类的设计思想，设计一个工具类，封装这些方法，给需要持久化的对象调用。

3.2 持久化框架实现

图 4 为持久化框架类图。其中，方面用构造型≪Aspect≫标识，斜体的方法声明表示该方法为抽象方法，带下划线的方法声明表示该方法为静态方法。为简明起见，省略了方法声明中参数的参数名，只保留了参数的类型。方面 PersistenceAspect 封装了持久化操作的切入点和持久化操作的具体实现。其切入点定义如图 5 所示。为提高框架的可重用性，该方面的设计采用了 Template 模式[26]。该模式将算法封装为一个方法，在该方法中定义算法的基本骨架，将算法细分为多个步骤，将其中的某些步骤实现为抽象方法，延迟到子类实现。子类通过重写这些方法来修改这些步骤的具体实现，而又保持了算法的基本骨架。方面 HibernateAspect 和 JDBCAspect 继承了方面 PersistenceAspect，分别实现了基于 Hibernate 和 JDBC 的持久化操作。编程人员可以自定义方面继承 PersistenceAspect，封装所需要的持久化操作。

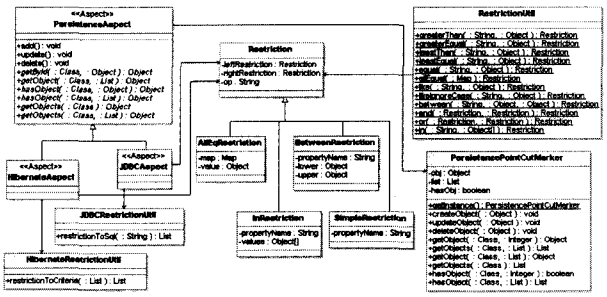

图 4　持久化框架类图

```
pointcut addRecord():execution(
 * PersistencePointCutMarker.createObject(Object));
after():addRecord(){…}
pointcut updateRecord():execution(
 * PersistencePointCurMarker.updateObject(Object));
after():updateRecord(){…}
pointcut deleteRecord():execution(
 * PersistencePointCutMarker.deleteObject(Object));
after():deleteRecord(){…}
pointcut retrivalById():call(
 * PersistencePointCutMarker.getObject(Class,Integer));
before():retrivalById(){…}
pointcut retrivalRecord():call(
 * PersistencePointCutMarker.getObject(Class,List));
before():retrivalRecord(){…}
pointcut hasRecordById():call(
 * PersistencePointCutMarker.hasObject(Class,Integer));
before():hasRecordById(){…}
pointcut hasRecord():call(
 * PersistencePointCutMarker.hasObject(Class,List));
before():hasRecord(){…}
pointcut retrivalAllRecords():call(
 * PersistencePointCutMarker.getObject(Class));
before():retrivalAllRecords(){…}
pointcut retrivalAllRecordsWithArgs():call(
 * PersistencePointCutMarker.getObject(Class,List));
before():retrivalAllRecordsWithArgs(){…}
```

图 5　持久化切入点

类 PersistencePointCutMarker 是持久化操作的工具类，用来标识持久化操作的切入点。其封装了业务逻辑编程人员需要使用的持久化操作。业务逻辑编程人员需要做持久化操作时，需要持有该类的对象的引用。为方便使用，该类还提供了获取该类实例的方法。类 PersistencePointCutMarker 封装了 3 个属性，分别是 Object 类型的 obj，List 类型的 list 及布尔型的 hasObj，用作查询时的返回值。图 6 展示了类 PersistencePointCutMarker 3 个方法的定义，返回值就是这 3 个属性。方面 PersistenceAspect 拦截到 PersistencePointCutMarker 的某个对象调用查询方法时（如 hasObject()），在执行查询方法的方法体（如 return hasObj）之前，需先进行查询操作。在查询完成后，利用 Java 的反射机制，将查询结果注入该对象的相应属性（如 hasObj），再执行对象调用的查询方法的方法体（如 return hasObj）。这样，确保了返回给编程人员的是该次查询的结果。为了确保方面定义的查询操作在 PersistencePointCutMarker 的查询方法之前执行，将查询操作的切入点定义为在方法调用（call）之前（before），具体代码如图 5 所示。

```
public Object getObject(Class cs,List args){
    return obj;
}
public boolean hasObject(Class cs,List args){
    return hasObj;
}
public List getObjects(Class cs,List args){
    return list;
}
```

图 6　PersistencePointCutMarker 的其中 3 个方法

在查询数据时，编程人员需要使用某种描述语言显式描述查询。参考 Apache OJB[27] 的设计思想，我们设计了抽象类 Restriction 及其子类，将查询条件封装为一个对象或对象集合。为支持 Restriction 的逻辑合成，类 Restriction 设计成类似于树结构的结点，由左结点，数据域（此处为操作符 op）和右结点组成。叶子结点包括 SimpleRestriction，InRestrictionAllEqRestrictiont，BetweenRestriction 4 种类型。叶子结点的左结点和右结点都为空（NULL）。其中类 SimpleRestriction 用于封装 SQL 单个类似“field > value”的表达式，包括>，>=，<，<=，=，like 运算符；类 AllEqRestriction 用于封装多个“field=value”表达式；类 InRestriction 用于封装“field in {…}”表达式；类 BetweenRestriction 用于封装“between”表达式。Restriction 支持多个 Restriction 的逻辑复合，分别用 or 和 and 表示逻辑或操作和逻辑与操作。为方便业务逻辑编程人员表达查询条件，类 RestrictionUtil 提供了构造 Restriction 对象的接口，使得业务逻辑编程人员无需了解 Restriction 的类结构及创建哪种类型的 Restriction。在方面 PersistenceAspect 的查询方法中，参数 List 对象容纳了一个或多个 Restriction 对象。通过对这些对象的分析，获取具体的查询条件，再转换为相应的查询描述。这分别由类 HibernateRestrictionUtil 和 JDBCRestrictionUtil 实现。

在图 4 框架类图中，左边部分为持久化方面，业务逻辑编程人员不必了解这一部分，甚至不必知道这一部分的存在；右边部分为持久化接口，业务逻辑编程人员只需掌握这部分接口，就能够透明地操作对象的存取；中间部分为封装查询条件的类结构，业务逻辑编程人员也不必了解这一部分。查询时，框架的这 3 部分交互机制类似于打包（packing）与拆包（unpacking）。业务逻辑编程人员通过右边的接口将查询条件打包，左边的部分则需要进行拆包、解释，以获取查询条件。中间部分即为包的内容。

4　性能测试

框架分别提供了基于 Hibernate 和 JDBC 的方面的参考实现，为分析框架的性能，分别测试了基于 Hibernate、JDBC 的方面的持久化应用及直接基于 Hibernate、JDBC 的持久化应用。设计了持久类 Contactor（见图 7）及对应数据库表 t_contactor（见图 8）。测试的软件环境为 Windows XP sp3 的操作系统、Mysql 5.0 数据库、1.6 版的 JDK 及 3.1 版的 Hibernate。

```
public class Contactor{
    private Integer id;
    private String name;
    private Boolean sex;
    private String tel;
    private String email;
    ...
}
```

图 7　持久对象 Contactor

```
CREATE TABLE `t-contactor` (
`id` int(11) NOT NULL auto_increment,
`name` varchar(20) default NULL,
`sex` bit(1) default NULL,
`tel` varchar(13) default NULL,
`email` varchar(50) default NULL,
PRIMARY KEY(`id`)
) ENGINE=InnoDB DEFAULT CHAR-
SET=latin1;
```

图 8　数据库表 t_contactor

测试方案如图 9 所示。在开始测试前，数据库表已经添加了随机生成的 1000 条记录。测试时首先添加 1000 条记录，然后修改 1000 条记录，最后删除这 1000 条记录，保证数据库里的记录还是初始生成的那 1000 条。每遍测试自动重复 5 次，记录每次测试花费的时间。共测试 50 遍，计算每次测试的平均值。在添加、修改及删除对象前，先查询对象是否已经存在数据库中。为避免随机性带来的影响，采用了一些特别的处理，使得添加、修改及删除对象总能执行。具体的方法是初始化数据库记录时记录的名字(name 属性)不含数字；ID 设置为整数，从 1 开始，自动增长。插入对象前将对象的名称修改为原来的名字(如“mick”)，加一串数字(如“123”)作为前缀，对象的新名字将为“123mick”，肯定不存在于数据库中。更新对象前随机生成的 ID 范围为[1,1000]，对应于数据库初始的 1000 条记录。删除对象时，条件设置为对应 SQL “name like ‘123%’”表达式，匹配之前插入的 1000 条数据。

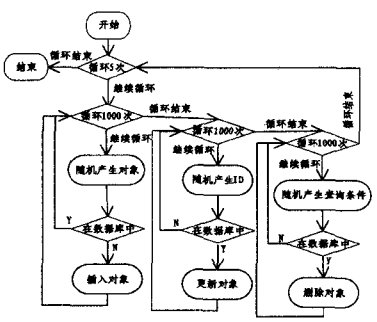

图 9　测试方案

测试结果如图 10 所示。标示为 JDBC，JDBCA，HIB 和 HIBA 的拆线分别是直接 JDBC 存储、JDBCAspect 存储、直接 Hibernate 存储和 HibernateAspect 存储每次所花费的时间。总的来说，这 4 种持久化技术花费的时间相近，性能非常接近。具体来说，基于 JDBC(直接 JDBC，JDBCAspect)的存储稍好于基于 Hibernate(直接 Hibernate，HibernateAspect)的存储。而直接 JDBC 或 Hibernate 存储又分别要好于其基于方面的实现。这 4 种持久化技术第一次的存储时间都比其它各次所花的时间多，主要是因为第一次循环时系统要初始化一些资源。这在基于 Hibernate(直接 Hibernate，HibernateAspect)的实现中表现得尤其突出。

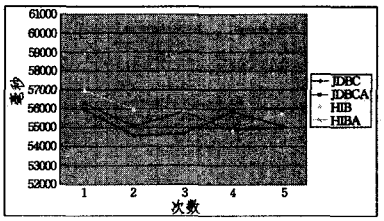

图 10　测试结果

在开发测试样例时，HibernateAspect 存储和 JDBCAspect 存储可以共用测试样例，而直接 Hibernate 存储和直接 JDBC 存储则需要各自的测试样例。这正体现了该框架高可移值性的优点，在不同的存储系统、不同的存储中间件之间迁移时，完全不需要改动业务逻辑代码。同时，还体现了框架的高可复用性和可扩展性。当需要扩展该框架时，比如需要不同的存储系统、不同的存储中间件，则直接重用该框架，无需改动，然后再自定义 PersistenceAspect 的具体实现即可。

结束语　AOP 分离了核心关注点和横切关注点，解决了面向对象技术在处理横切关注点时的代码散布和代码纠缠问题。和日志记录、安全验证一样，持久化也被认为是经典的横切关注点业务，适合用 AOP 来实现。本文分析了现有的基于 AOP 的持久化框架，发现这些框架大部分过于追求 AOP 要求的 obliviousness 特性，而在功能、性能、适用性及可使用性上考虑较少，导致了这些框架难以在现实系统中应用。本文综合考虑了这些框架的机制，分析了持久化的特点和持久方面化的机制，提出了一套实用的基于 AOP 的对象持久化框架。本框架的主要特色包括：

①采用工具类的设计思想，避免了对持久类类型的限制；

②采用 Template 设计模式设计持久化方面，提高了框架的可复用性、可维护性及可移植性；

③采用面向对象的查询机制，查询的表达显得简洁、明了。

框架分别提供了基于 Hibernate 和 JDBC 的方面的参考实现。我们测试了基于 Hibernate 和 JDBC 的方面的框架的性能，并与直接采用 Hibernate 和 JDBC 存储进行了对比，结果显示框架的性能基本接近其采用的存储技术的性能。

本文提出的持久化框架具有类似“可插拔”的功能：在不改动业务逻辑代码的前提下，当“拔下”持久化方面时，系统就转化为不具有持久化功能的系统；当“插入”持久化方面后，系统又转化为具有持久化功能的系统；当需要更换存储系统或存储中间件时，“拔下”在用的持久化方面，“插入”所需要的持久化方面。与传统的面向对象实现的持久化框架相比，基于面向方面实现的框架具有更高的可重用性、可维护性和可移植性。与现有的基于面向方面实现的框架相比，我们的框架对功能、性能、适用性、可使用性的考虑更加充分，适合现实系统应用。

本文主要阐述了框架的设计思想及初步的框架实现。但框架在功能上还需要进一步完善，比如持久化操作的批处理、事务及元数据操作、多线程支持等。另外，框架提供的面向对象的查询表达方式的表达能力还需要进一步增强，理想的情

况是能达到 SQL 的表达水平。

参 考 文 献

[1] Gregor K, Lamping J, Mendhekar A, et al. Aspect-oriented Programming[C]// Proceedings of the European Conference on Object-Oriented Programming. vol 1241: 220-242

[2] http://eclipse.org/aspectj/

[3] http://aspectwerkz.codehaus.org/

[4] http://jboss.org/jbossaop/

[5] http://www.aspectc.org/

[6] http://wwwiti.cs.uni-magdeburg.de/iti_db/forschung/fop/featurec/

[7] http://www.castleproject.org/aspectsharp/

[8] http://www.facultyresourcecenter.com/curriculum/pfv.aspx?ID=6801

[9] Douglas B, Torsten S. Solving the Java Object Storage Problem [J]. Computer, 1998, 31(11): 33-40

[10] Java Data Objects Specification[S]. Version: 2.0, 2005

[11] SUN Microsystems. Enterprise JavaBeans[S]. Version 2.0, Final Release, August 2001

[12] SUN Microsystems. Enterprise JavaBeans[S]. Version 3.0, Final Release, 2006

[13] Hibernate Reference Documentation[EB/OL]. Version: 3.2.3.ga, http://www.hibernate.org

[14] Rashid A. On to Aspect Persistence[C]// GCSE Syrup. LNCS 2177. Springer-Verlag, 2000: 26-36

[15] Rashid A. Weaving Aspects in a Persistent Environment[C]// ACM SIGPLAN Notices. Feb. 2002

[16] Rashid A, Loughran N. Relational Database Support for Aspect-Oriented Programming[C]// Proceedings of NetObjectDays. LNCS 2591. Springer-Verlag, 2002: 233-247

[17] Rashid A, Sawyer P. Dynamic Relationships in Object Oriented Databases: A Uniform Approach[C]// DEXA. LNCS 1677. Springer-Verlag, 1999: 26-35

[18] Rashid A, Chitchyan R. Persistence as an Aspect[C]// Proc. of AOSD'03. Boston, USA, 2003: 120-129

[19] Filman R, Friedman D. Aspect-oriented Programming is Quantification and Obliviousness[C]// OOPSLA Workshop on Advanced Separation of Concerns. 2000

[20] Al-Mansari M, Hanenberg S, Unland R. Orthogonal persistence and AOP: a balancing act[C]// Proceedings of the 6th Workshop on Aspects, Components, and Patterns for Infrastructure Software. Vancouver, British Columbia, Canada, March 2007: 12-16

[21] Al-Mansari M, Hanenberg S. Path Expression Pointcuts: Abstracting over Non-Local Object Relationships in Aspect-Oriented Languages[C]// NODe'06. Erfurt, Germany, 2006

[22] Campbell R, Habermann A. The Specification of Process Synchronization by Path Expressions[C]// Sym. on Operating Systems. Springer-Verlag, 1974: 89-102

[23] Elmasri R, Navathe B. Fundamentals of Database Systems(3rd ed)[M]. Addison-Wesley, 2000

[24] 陈兴润，滕腾，黄罡，等. 一种对象/关系映射隐式持久化框架[J]. 电子学报，2007，35(B12)：179-185

[25] Java Persistent Aspect[EB/OL]. http://sourceforge.net/projects/jpa/

[26] Gamma E, Helm R, Johnson R, et al. Design Patterns: Elements of Reusable Object-oriented Software[M]. Reading, MA: Addison-Wesley, 1995

[27] http://db.apache.org/ojb/

(上接第 110 页)

[7] Guo Feng, Zhuang Yi-qi, Hu Bin. Structure of Frequency Hopping Sequences Family Based on Stream Cipher[J]. Chinese Journal of Electron Devices, 2007, 30(5): 1696-1699

[8] Ho Yean Li, Samsudin A, Belaton B. Heuristic cryptanalysis of classical and modern ciphers Networks[C]// 7th International Conference on Communication. Malaysia: IEEE, 2005: 6

[9] Nadeem A, Javed M Y. A Performance Comparison of Data Encryption Algorithms Information and Communication Technologies[J]//First International Conference on Information and Communication Technologies. Karachi, Pakistan: IEEE, 2005: 84-89

[10] Chu W, Colbourn C J. Optimal frequency-hopping sequences via cyclotomy[J]. IEEE Transactions on Information Theory, 2005, 51(3): 1139-1141

[11] 王淑波，梅文华，毕笃彦. 蓝牙自适应跳频序列的性能分析[J]. 电波科学学报，2006，21(4)：612-618

[12] Nafaa A, Ahmed T, Mehaoua A. Unequal and interleaved FEC protocol for robust MPEG-4 multicasting over wireless LANs [C]// IEEE International Conference on Communications. 2004: 1431-1435

[13] Han Sunyoung, Kim Heemin, Son Kiwon, et al. Cross-correlated FEC Scheme for Multimedia Streaming over Wireless LAN[C]// 22nd International Conference on Advanced Information Networking and Applications. 2008: 217-222

[14] Gilbert E N. Capacity of a burst-noise channel[J]. Bell Syst. Tech. J, 1960, 39: 1253-1265

[15] Wilhelmsson L, Milstein L B. On the Effect of Imperfect Interleaving for the Gilbert-Elliott Channel[J]. IEEE Transactions on Communications, 1999, 47(5): 681-688

[16] Federal Information Processing Standards Publication (FIPS) 197. Specification for the Advanced Encryption Standard(AES) [S]. 2001

[17] Salomon D. Data Privacy and Security[M]. Beijing: Tsinghua University Press, 2005: 292-302

[18] Nakajima M, Yamamoto T, Yamasaki M, et al. Low Power Techniques for Mobile Application SoCs Based on Integrated Platform "UniPhier"[C]// IEEE Design Automation Conference. 2007: 649 - 653

[19] 熊志辉，李思昆，陈吉华，等. 支持平台设计方法的系统芯片协同设计环境[J]. 计算机辅助设计与图形学学报，2005，17(7)：1401-1406

[20] 章立生，韩承德. SoC 芯片设计方法及标准化[J]. 计算机研究与发展，2002，39(1)：1-8