



(12) 发明专利

(10) 授权公告号 CN 101980213 B

(45) 授权公告日 2012.09.19

(21) 申请号 201010562533.0

CN 101067814 A, 2007.11.07, 全文.

(22) 申请日 2010.11.23

欧黎源等."基于 JPA 的数据持久化模型设计与实现".《计算机工程》.2009,第 35 卷(第 20 期),76,77,80.

(73) 专利权人 中国科学院软件研究所
地址 100190 北京市海淀区中关村南四街 4 号

苗晓辉."基于 J2EE 的数据持久化的研究与实现".《计算机工程》.2007,第 33 卷(第 5 期),272-274.

(72) 发明人 王帅 屈夏 杨燕 钟华

审查员 吴少鸿

(74) 专利代理机构 北京君尚知识产权代理事务所(普通合伙) 11200

代理人 冯艺东

(51) Int. Cl.

G06F 17/30(2006.01)

G06F 9/44(2006.01)

(56) 对比文件

US 6912541 B1, 2005.06.28, 全文.

US 2005/0073959 A1, 2005.04.07, 全文.

CN 101645074 A, 2010.02.10, 全文.

CN 101110030 A, 2008.01.23, 全文.

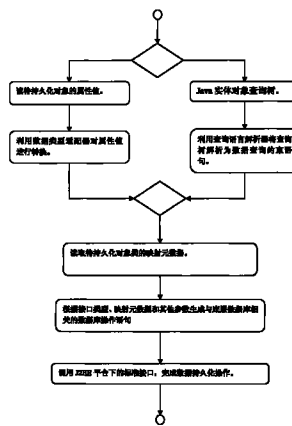
权利要求书 2 页 说明书 9 页 附图 4 页

(54) 发明名称

一种基于 J2EE 的数据持久化方法及系统

(57) 摘要

本发明公开了一种基于 J2EE 的数据持久化方法和系统,其方法包括:1) 用与底层数据库相关的注释语言对待持久化的 Java 类进行映射元数据标记;2) 构建待持久化 Java 类对象,对对象的属性值进行赋值;3) 构建 Java 实体对象查询树;4) 以创建的待持久化 Java 类对象和 Java 实体对象查询树为参数,调用持久化接口进行持久化操作。系统包括数据类型适配器、查询语言解析器和操作语言解析器。本发明的方法和系统解决了 Java 实体对象与底层数据库的“阻抗不匹配”,屏蔽了异构数据库对信息系统所提供接口的差异,不依赖于任何脚本语言,整个过程完全是 Java 接口调用,降低了持久化层的开发难度。



CN 101980213 B

1. 一种基于 J2EE 的数据持久化方法,包括下列步骤:

1) 用与底层数据库相关的注释语言对待持久化的 Java 类进行映射元数据标记;

2) 构建待持久化 Java 类对象,对对象的属性值进行赋值;

3) 构建 Java 实体对象查询树;

4) 以创建的待持久化 Java 类对象和 Java 实体对象查询树为参数,调用持久化接口进行持久化操作;

所述注释语言将 Java 实体对象中的结构和字段与底层数据库的相关结构和字段进行一一映射,消除所述结构和字段间的数据结构不匹配;

所述持久化操作为:

A、从待持久化 Java 类提供的数据库读取接口,读取 Java 实体对象数据域的值,将所述值转换为底层数据库数据要求的类型和格式;

B、将 Java 实体对象查询树解析为与底层数据库相对应的数据库查询约束语句;

C、根据所调用的持久化接口类型,和第 A、B 步生成的结果按照底层数据库操作语言要求生成数据库操作语句;

D、以 C 步骤生成的数据库操作语句为参数,调用底层数据库的标准接口,进行数据库操作;

所述持久化接口包括 isObjectExist 接口、saveObject 接口、loadObject 接口、getObjects 接口、updateObject 接口和 deleteObject 接口,其中,

isObjectExist 接口判断待持久化 Java 类对象是否已经存在;

saveObject 接口保存待持久化 Java 类对象到数据库中;

loadObject 接口从数据库中读取一个满足约束条件的对象;

getObjects 接口从数据库中读取所有满足约束条件的对象;

updateObject 接口将数据库中满足约束条件的对象进行更新;

deleteObject 接口从数据库中删除符合约束条件的对象。

2. 根据权利要求 1 所述的基于 J2EE 的数据持久化方法,其特征在于所述的映射元数据包括类元数据和属性元数据,类元数据指定 Java 实体对象类中的主键属性并描述 Java 实体对象类与底层数据库结构之间的映射关系;属性元数据描述 Java 实体对象属性与底层数据库数据项之间的关系。

3. 根据权利要求 1 所述的基于 J2EE 的数据持久化方法,其特征在于将所述值转换为底层数据库数据要求的类型格式的方法为:

第一步将所述值按照底层数据库的类型与格式进行重新组装;

第二步调用数据库的赋值函数完成组装后数据的持久。

4. 根据权利要求 1 所述的基于 J2EE 的数据持久化方法,其特征在于所述查询树包括逻辑节点和表达式节点。

5. 根据权利要求 4 所述的基于 J2EE 的数据持久化方法,其特征在于所述逻辑节点包含一个或多个逻辑或表达式节点作为子节点;表达式节点表达属性名、属性值及属性名和属性值之间的关系。

6. 根据权利要求 1 所述的基于 J2EE 的数据持久化方法,其特征在于所述的数据库操作语句以持久化接口类型区分操作类型,以数据库查询约束语句表示查询约束条件,采用对象

类的映射元数据指定操作位置。

7. 一种基于 J2EE 的数据持久化系统,包括:

第一单元,用于用与底层数据库相关的注释语言对待持久化的 Java 类进行映射元数据标记;所述注释语言将 Java 实体对象中的结构和字段与底层数据库的相关结构和字段进行一一映射,消除所述结构和字段间的数据结构不匹配;

第二单元,用于构建待持久化 Java 类对象,对对象的属性值进行赋值;

第三单元,用于构建 Java 实体对象查询树;

第四单元,用于用以创建的待持久化 Java 类对象和 Java 实体对象查询树为参数,调用持久化接口进行持久化操作;

所述持久化接口包括 isObjectExist 接口、saveObject 接口、loadObject 接口、getObjects 接口、updateObject 接口和 deleteObject 接口,其中,

isObjectExist 接口判断待持久化 Java 类对象是否已经存在;

saveObject 接口保存待持久化 Java 类对象到数据库中;

loadObject 接口从数据库中读取一个满足约束条件的对象;

getObjects 接口从数据库中读取所有满足约束条件的对象;

updateObject 接口将数据库中满足约束条件的对象进行更新;

deleteObject 接口从数据库中删除符合约束条件的对象;

所述持久化操作为:

A、从待持久化 Java 类提供的数据库读取接口,读取 Java 实体对象数据域的值,将所述值转换为底层数据库数据要求的类型和格式;

B、将 Java 实体对象查询树解析为与底层数据库相对应的数据库查询约束语句;

C、根据所调用的持久化接口类型,和第 A、B 步生成的结果按照底层数据库操作语言要求生成数据库操作语句;

D、以 C 步骤生成的数据库操作语句为参数,调用底层数据库的标准接口,进行数据操作。

一种基于 J2EE 的数据持久化方法及系统

技术领域

[0001] 本发明涉及一种在 J2EE 平台下异构数据库环境中通用的数据持久化方法及系统,目标是提高小型信息系统持久层的开发效率、降低在异构数据库之间的迁移成本。

背景技术

[0002] 作为信息持久化的主要手段,数据库已经成为绝大多数信息系统中不可或缺的一部分。为了满足各种各样的数据库应用需求,大量的异构数据库应运而生。目前在信息系统中使用较为广泛的异构数据库有:

[0003] ●关系型数据库:建立在关系数据模型上,将数据分解成为二维数据表进行存储。关系数据库具有数据结构化强、数据独立性大、数据冗余较多等特点,其对于数据读、写的表现比较均衡。

[0004] ●对象数据库:将面向对象的方法和数据库技术结合起来,使数据库的分析、设计最大程度地与人们对客观世界的认识相一致,其读写的表现比较均衡,但整体性能要比关系数据库稍逊一筹。

[0005] ●XML数据库:建立在XML格式文档的基础上,能够对半结构化和具有一定层次结构的数据进行有效的存取管理,特别适用于管理和查询网络数据,其读写的表现比较均衡,整体性能要比对象数据库更差,但是由于XML格式信息更适于在网络中传输并且其数据结构便于人类理解,XML数据库仍然在数据库的使用中占有一席之地。

[0006] ●目录数据库:一种基于目录服务的数据库,对于读取操作做了特别的优化,因而更适用于读数据比写数据次数大的多的信息系统。

[0007] 此外,随着Internet的广泛普及,Web信息系统的数量突飞猛增。Web服务器作为连接客户端和数据库服务器的一个中间层,既要对用户的访问请求进行分析和处理,又要和数据库进行数据交互。为了实现Web服务器的模块化和层次化,系统设计开发时都会把Web服务器划分为负责分析和处理用户访问请求的业务逻辑层以及负责与数据库服务器进行交互的数据持久层。于是,如何使信息系统的数据库持久层方便、快捷、高效的与各种异构数据库之间进行交互成为了一个重要的研究课题。

[0008] 为了方便用户在J2EE平台上实现对数据库的访问,J2EE推出了一系列访问数据库的标准接口,如Java Database Connectivity (Oracle Corporation. JDBC[EB/OL]. <http://java.sun.com/products/jdbc/overview.html>)、Java Naming and Directory Interface (Oracle Corporation. JNDI[EB/OL]. <http://java.sun.com/products/jndi/overview.html>)等。使用这些接口有很多的好处:首先,利用标准接口开发的信息系统不但可以跨平台运行,更可以在同构数据库之间不受数据库供应商的限制;其次,标准接口通常都使用该种数据库的主流语言,并保留了其查询语言的全部能力,允许用户调用存储过程、执行特殊自定义查询甚至使用与数据库相关的特性操作;最后,直接使用标准接口会获得很好的操作性能。然而直接使用标准接口也会有很多不便:一、开发人员必须确保数据库资源得到了妥善的处理。其中最显著的问题是能否有效的管理数据库连接。在打开并使用

完数据库连接之后（或者遇到异常时）必须确保将其正确的关闭，否则就有可能耗尽数据库连接从而导致系统崩溃。二、直接使用标准接口编程意味着开发人员需要在代码中构造与数据库相关的查询语句，一旦数据模型发生了变化，那么必须手工修改四处散布的查询语句以及代码。三、大多数标准接口所提供的 API 方法都可能抛出已检测的 DBException，开发人员不仅要保证捕获这些异常，而且还要保证一旦异常抛出会执行相应的清除工作。此外，DBException 的异常处理逻辑往往非常相似，导致持久层中会出现大量重复代码，降低了系统的可维护性。四、直接使用标准接口往往会导致应用程序与数据源之间的高度耦合，而高耦合、高重复是使得程序难以测试的两个主要原因（J. B. Rainberger, Scott Stirling. JUnit Recipes: Practical Methods for Programmer Testing[M]. Manning Publications Co., 2005.）。

[0009] 鉴于直接使用标准数据库接口的诸多弊端，一些持久化中间件试图在标准接口之上提供一层轻量级的封装来解决这些问题（汪萌，曲俊华．基于 Hibernate 技术的持久层解决方案及实现 [J]．计算机系统应用，2010，(03)．）。这些中间件为用户隐藏了使用标准接口的大部分细节，带来了透明的数据持久化体验。相对于直接使用标准接口，使用持久化中间件可以帮助开发人员节省大量的时间。但持久化中间件也存在一个重要的缺陷，即：只提供了对于某一个标准接口的封装，使用这些中间件进行数据持久化仍然会引起信息系统与底层的数据库较高的耦合度，如果信息系统由于某种原因要将现有数据库更换为另一种异构数据库（如从关系数据库变成目录数据库），则需要付出较为沉重的代价。

[0010] 更重量级的 J2EE 持久层框架弥补了持久层中间件在异构数据库集成方面的不足，它们符合目前 J2EE 平台的持久化规范，并且可以集成各种异构数据库的标准访问接口。然而这些功能强大的持久层框架往往非常复杂而且体积庞大。例如：DataNucleus (DaanucleusCommunity. Daanucleus[EB/OL]. <http://www.datanucleus.org/>.) 2.1.0Release 的核心 JAR 包的大小就接近 2MB。不仅如此，要想使用 DataNucleus 的全部功能还需要许多额外的 JAR 包，如 asm、ehcache、poi 等，共计 19.5MB。如此重量级的持久化框架对于一些小型 Web 信息系统来说实在过于庞大，程序员要熟练掌握类似的持久化框架并发挥它的优势也绝非易事。

[0011] 综上，J2EE 平台下的已有数据持久化方法的缺点如下：

[0012] 第一，标准接口功能强大却难以使用；

[0013] 第二，简单的持久化中间件虽然在一定程度上简化了标准接口的使用，但仍然因为与数据库类型高度耦合而受到了限制；

[0014] 第三，功能强大的持久化框架可以让我们彻底摆脱对于底层数据库的考虑、专心于业务逻辑的处理，却结构复杂且难于掌握。

[0015] 针对需要快速开发的小型信息系统，需要一种简单易用、能够低成本的在各种异构数据库之间进行迁移的 J2EE 数据持久化方法。该方法接口明确、易于开发、所有操作完全符合面向对象的思想、能够集成可通过 J2EE 标准接口访问的多种异构数据库、只有很少或者没有第三方依赖、能够通过对于 Java 实体对象进行 CRUD (Wikipedia. CRUD[EB/OL]. http://en.wikipedia.org/wiki/Create,_read,_update_and_delete) 操作。使用该方法进行开发的 Web 信息系统，能够达到简单便捷的构建系统持久化层、通过简单配置即可实现异构数据库之间的持久化迁移。

发明内容

[0016] 本发明的目的在于克服现有持久化技术中存在的一系列问题,提供 J2EE 环境下的通用数据持久化方法及系统。

[0017] 本发明的技术方案为:一种基于 J2EE 的数据持久化方法,包括以下步骤:

[0018] 1) 用与底层数据库相关的注释语言对待持久化的 Java 类进行映射元数据标记;

[0019] 2) 构建待持久化 Java 类对象,对对象的属性值进行赋值;

[0020] 3) 构建 Java 实体对象查询树。

[0021] 4) 以创建的待持久化 Java 类对象和 Java 实体对象查询树为参数,调用持久化接口进行持久化操作。

[0022] 所述的映射元数据包括类元数据和属性元数据,类元数据指定 Java 实体对象类中的主键属性并描述 Java 实体对象类与底层数据库结构之间的映射关系;属性元数据描述 Java 实体对象属性与底层数据库数据项之间的关系。

[0023] 所述持久化接口包括 isObjectExist 接口、saveObject 接口、loadObject 接口、getObjects 接口、updateObject 接口、updateObject 接口和 deleteObject 接口,其中,

[0024] isObjectExist 接口判断待持久化 Java 类对象是否已经存在;

[0025] saveObject 接口保存待持久化 Java 类对象到底层数据库中;

[0026] loadObject 接口从数据库中读取一个满足约束条件的对象;

[0027] getObjects 接口从数据库中读取所有满足约束条件的对象;

[0028] updateObject 接口将数据库中满足约束条件的对象进行更新;

[0029] deleteObject 接口从数据库中删除符合约束条件的对象。

[0030] 所述持久化操作的方法为:

[0031] A、从待持久化 Java 类提供的数据库读取接口,读取 Java 实体对象数据域的值,将所述值转换为底层数据库数据要求的类型和格式;

[0032] B、将 Java 实体对象查询树解析为与底层数据库相对应的数据库查询约束语句;

[0033] C、按照底层数据库操作语言要求生成数据库操作语句;

[0034] D、以 C 步骤生成的数据库操作语句为参数,调用底层数据库的标准接口,进行数据库操作。

[0035] 将所述值转换类型和格式的方法为:

[0036] 第一步将所述值按照底层数据库的类型与格式进行重新组装;

[0037] 第二步调用数据汇的赋值函数完成组装后数据的持久。

[0038] 所述查询树包括逻辑节点和表达式节点。

[0039] 所述逻辑节点包含一个或多个逻辑或表达式节点作为子节点;表达式节点表达属性名、属性值及属性名和属性值之间的关系。

[0040] 所述的数据库操作语句以持久化接口类型区分操作类型,以数据库查询约束语句表示查询约束条件,采用对象类的映射元数据指定操作位置。

[0041] 本发明的另一个目的是提供基于 J2EE 的数据持久化系统,包括数据类型适配器、查询语言解析器和操作语言解析器

[0042] 数据类型适配器,将 Java 实体对象数据域的值转换成为符合底层数据库要求的

格式和类型。

[0043] 查询语言解析器,将 Java 实体对象查询树解析为相应的数据查询约束语句;

[0044] 操作语言解析器,根据调用的持久化接口类型、标记待持久化 Java 类对象的映射元数据和数据查询约束语句生成数据库操作语句。

[0045] 与现有数据持久化方法相比,本发明的方法解决了如下问题:

[0046] 第一,通过向用户提供一组纯 Java 的异构数据库通用访问接口,屏蔽了异构数据库对信息系统所提供接口的差异,使信息系统与底层数据库类型完全解耦,把接口使用者从大量的数据库配置细节和操作编码当中解放出来。

[0047] 第二,借助关于持久化类的一些映射信息和数据类型适配器,程序员不必考虑 Java 实体对象与底层数据库的“阻抗不匹配”(田珂,谢世波,方马. J2EE 数据持久层的解决方案 [J]. 计算机工程,2003, (22).),降低了信息系统持久化层的开发难度。

[0048] 第三,使用 Java Annotation(Oracle Corporation. JDBC[EB/OL]. <http://download.oracle.com/javase/1.5.0/docs/guide/language/annotations.html>) 实现对 Java 类对象的标记,实现了配置文件与实体数据的完整统一,避免了在配置文件与实体数据分离模式下对其一进行修改所造成的不同步性。

[0049] 第四,进行数据持久化时不依赖于任何脚本语言,整个过程完全是 Java 接口调用,与数据库相关的操作语言由框架根据 Java 类及其注释统一生成,减轻了开发者的负担,避免了由于开发者对数据库语言使用不规范造成的数据损毁。

附图说明

[0050] 图 1 是数据持久化方法的流程框图;

[0051] 图 2 是转换数据类型和格式的方法流程示意图;

[0052] 图 3 是 Java 实体对象查询树示意图。

[0053] 图 4 是本发明系统中各模块协作示意图。

[0054] 图 5 是实施例中的基于异构数据库的轻量级持久层框架的结构示意图。。

具体实施方式

[0055] 以下结合具体实施例和附图对本发明方法进行详细说明。

[0056] 本发明方法旨在为小型 Web 信息系统开发者提供一个快速搭建数据持久层的途径,为了满足绝大多数小型信息系统的数据库持久化需求而又不至于过于臃肿,本方法将数据库持久操作抽象为六种持久化操作接口。各接口的参数及返回值如表 1 所示。使用这些接口,用户可以直接面向被映射元数据标记过的 Java 实体对象实现数据的持久化。

[0057] 表 1LAPFA 数据库统一访问接口

[0058]

类型	返回值	接口名	参数	备注
I	Boolean	isObjectExist	Object obj	判断待持久化 Java 类对象是否已经存在
C	void	saveObject	Object obj	保存待持久化 Java 类对象到数据库中
R	Object	loadObject	Object obj	从数据库中读取一个满足约束条件的对象
R	List	getObjects	Class clazz, Filter filter	从数据库中读取所有满足约束条件的对象
U	void	updateObject	Object obj	将数据库中满足约束条件的对象进行更新
D	void	deleteObject	Object obj	从数据库中删除符合约束条件的对象

[0059] 本发明的方法为：

[0060] 1) 用与底层数据库相关的注释语言对待持久化的 Java 类进行映射元数据标记；

[0061] 2) 通过调用待持久化类的默认构造方法创建待持久化 Java 类的对象，并通过调用对象属性赋值函数对对象的属性值进行赋值；

[0062] 3) 根据持久化操作的具体需要，利用自定义的查询结构构建 Java 实体对象查询树。

[0063] 4) 以创建的待持久化对象和 Java 实体对象查询树为参数，根据需要调用上述表 1 中的持久化接口进行持久化操作，操作步骤如下：

[0064] A、从待持久化 Java 类提供的数据库读取接口，读取 Java 实体对象数据域的值，将所述值转换为底层数据库数据要求的类型和格式；

[0065] B、将 Java 实体对象查询树解析为与底层数据库相对应的数据库查询约束语句；

[0066] C、根据所调用的持久化接口类型，和第 A、B 步生成的结果按照底层数据库操作语言要求生成数据库操作语句；

[0067] D、以 C 步骤生成的数据库操作语句为参数，调用底层数据库的标准接口，进行数据库操作。

[0068] 将 Java 实体对象在非面向对象的异构数据库中进行持久化的最大障碍就是实体对象与各种异构数据库之间的“阻抗不匹配”。上述二者之间的“阻抗不匹配”体现在“数据结构”和“数据类型与格式”两个方面。数据结构的不匹配是指 Java 实体对象所采用的数据结构与数据库所采用的数据结构的不一致性。这一点在关系数据库中尤为明显：Java 实体对象采用嵌套的层次结构表示，而关系数据库采用单一的二维表。本发明方法采用与底层数据库相关的注释语言对待持久化的 Java 类进行映射元数据标记，即将 Java 实体对象中的结构和字段与底层数据库的相关结构和字段进行一一映射，消除它们之间的数据结构不匹配。

[0069] 映射元数据分为两类：类元数据和属性元数据。类元数据用于指定 Java 实体对象类中的主键属性并描述 Java 实体对象类与数据库结构之间的映射关系。如：在关系数据库中对象与表之间的映射，在目录服务数据库中对象与条目之间的映射等。属性元数据用于描述 Java 实体对象属性与数据库数据项之间的关系。如：在关系数据库中属性名与表中列

名,对象属性类型和 SQL 数据类型之间的映射。

[0070] 数据类型与格式的不匹配是指 Java 实体对象中属性的类型和数据格式与数据库中对数据项的类型和格式的不一致。比如,在 Java 实体类中的一个属性 `java.util.Date birthday = new java.util.Date(1985,10,1)`;在保存到 LDAP 数据库前需要将其转换成 String 类型的“19851001000000z”。为了解决数据类型与格式不匹配,本发明方法引入了数据类型适配器。在使用本发明方法进行数据存取时,数据类型适配器会从数据源读取数据,将其按照底层数据库的类型与格式进行重新组装,最后调用数据汇的赋值函数完成数据的持久,当从底层数据库中提取到相应数据时,也可以根据需要的格式重新恢复数据的格式和类型,在此所述的重新组装可以是数据类型和格式的双向转换,参见图 2 所示。

[0071] 数据查询是数据持久化层工作的一个必不可少的功能点,在这个功能点上,方便灵活的表示查询约束条件显得尤为重要。本发明方法采用 Java 实体对象查询树解析的数据查询约束语句表示查询约束条件,一方面保证了约束条件表示覆盖的广泛性,另一方面也使得查询约束条件可以完全使用 Java 实体类对象构建的树形结构表示,使其彻底与底层数据库解耦。

[0072] 查询树包括两类节点:逻辑节点和表达式节点,两者采用组合模式共同构造查询树。逻辑节点有一个或多个逻辑或表达式节点作为子节点,逻辑节点本身还要保存其子节点之间的逻辑关系,这些逻辑关系是指数据库查询中表达式之间的“与”、“或”、“非”等逻辑关系。表达式节点没有子节点,但要保存三个重要信息:属性名、属性值以及属性名和属性值之间的关系,是指“相似”、“大于”等数据库查询约束条件中变量名和变量值之间的关系。参见图 3 中的查询树,则将该查询树解析为 SQL 中查询约束语句为“(age > 20 AND sn = wang) OR (age < 21 and NOT sex = male)”。

[0073] 数据库操作语言是应用程序向数据库发送请求的唯一载体。本发明方法设计了一种在 Java 环境下能满足系统接口需求、简单易用且可以灵活表示多种异构数据库操作的数据库操作语言。该数据库操作语言使用接口函数区分操作类型、采用上述定义的 Java 对象组成的查询树表示查询约束条件,借助对象类的映射元数据指定操作位置,实现了持久化操作的无脚本化。在进行数据持久化操作时,与底层数据库相关的操作语言解析器完全可以根据使用者调用的持久化接口类型、待持久化对象所标记的映射元数据和 Java 实体对象查询树来构造出底层数据库的查询语句。

[0074] 图 4 表明了本发明的基于 J2EE 的数据持久化系统中的各模块之间是如何协同工作的。

[0075] 本发明的持久化系统,包括映射元数据读取器、数据类型适配器、查询语言解析器、持久化接口调用器和操作语言解析器

[0076] 映射元数据读取器对待持久化的 Java 类进行映射元数据标记并读取映射元数据;

[0077] 数据类型适配器,将 Java 实体对象数据域的值转换为符合底层数据库要求的格式和类型

[0078] 查询语言解析器,将 Java 实体对象查询树解析为与底层数据库相对应的数据查询约束语句;

[0079] 持久化接口调用器,根据需要调用表 1 中的持久化接口。

[0080] 操作语言解析器,根据调用的持久化接口类型、待持久化对象所标记的映射元数据和查询语言解析器生成的数据查询约束语句生成数据库操作语句。

[0081] 为了验证本发明方法的有效性,本文利用该方法设计了一个基于异构数据库的轻量级持久层框架 LAPFA (Lightweight dAta Persistence Framework for jAva),其整体构架如图 5 所示。该框架系统包括映射元数据读取器、数据类型适配器、查询语言解析器、持久化接口调用器和操作语言解析器。

[0082] 下文以 Web 信息系统中的用户信息作为数据模型,采用标准 LDAP 目录数据库作为底层数据库结合 LAPFA 的工作流程,进一步阐述上述构件如何相互配合完成数据持久化。

[0083] LAPFA 中 LDAP 目录数据库的映射元数据由两个注释类 ClassTag 和 FieldTag 定义,其中 ClassTag 注释类的说明如下:

[0084] 表 2 注释 ClassTag 属性表

[0085]

属性名	类型	说明
branch	String	类在 LDAP 中生成 entry 的位置。
idFieldName	String	指定类中某一属性为数据的 RDN。
objectClasses	String[]	类在 LDAP 中对应 entry 的 objectClasses。

[0086] 其中, [] 表示返回类型是一个数组。

[0087] FieldTag 注释类的说明如下:

[0088] 表 3 注释 FieldTag 属性表

[0089]

属性名	类型	说明
attrID	String	属性在数据库中的字段名。
attrType	String	属性在数据库中的类型。
singleValue	boolean	该属性是否为单值属性。
cascade	boolean	是否级联保存(用于非原生类型)。
classType	String	指定其 Java 类名(用于非原生类型)。

[0090] Web 信息系统中的用户信息 User 包括以下属性:

[0091] 表 4 用户类 User 属性表

[0092]

属性名	类型	说明
username	String	用户名

password	byte[]	用户密码
loginCounter	int	登陆次数

[0093] 其中, [] 表示返回类型是一个数组。

[0094] 1) 首先映射元数据读取器用与底层数据库相关的注释语言对待持久化的 Java 类进行映射元数据标记, 注释后的代码表如下:

[0095] 表 5 用户类 User 被注释后的代码表

```
[0096] package org.sam
ple;
import .....;
@ClassTag(branch="ou=user", idFieldName="userName",
objectClasses =
{"inetOrgPerson","onceportalUser"})
public class User {
@FieldTag(attrID = "cn", attrType="DirectoryString")
String userName;
@FieldTag(attrID = "userPassword", attrType="Password")
byte[] password;
@FieldTag(attrID = "loginCounter", attrType="Integer")
int loginCounter;
//methods that belongs to org.sample.User
}
```

[0097] 2) 创建了一个待持久化类的对象 xw, 并将其属性赋值:

[0098] 3) 根据查询需求创建了 Java 实体类查询对象树 and。

[0099] 4) 以创建的待持久化对象 xw 和 Java 实体对象查询树 and 为参数, 持久化接口调用器调用持久化接口进行持久化操作,

[0100] 首先利用对象 xw 的数据读取接口读取 Java 实体对象的属性值, 利用数据适配器将所述值转换为底层数据库数据要求的类型和格式。

[0101] 然后调用 LAPFA 查询语言解析器将 Java 对象查询树 and 解析成为符合 LDAP 协议的数据查询约束语句: &(loginCounter < 3) (username ~ = xiao);

[0102] 接下来调用 LAPFA 数据库操作语言解析器, 根据调用接口的类型和注释信息, 生成数据库操作语句;

[0103] 最后以上述数据库操作语句为参数, 调用 LDAP 目录数据库在 J2EE 平台下的标准接口 JNDI, 完成数据查询操作。

[0104] 下面是 TestDriver 类演示了开发者如何使用 LAPFA 完成数据的持久化过程:

```
[0105] package org.sample;
```

```
[0106] import.....;
```

```
[0107] public class TestDriver{
```

```
[0108]     public static void main(String[] args) {
```

```
[0109]         DatabaseService ds = new JndiDatabaseImpl();
```

```
[0110]         User xw = new User();
```

```
[0111]         //set xw.userName with " XiaoWang"
```

```
[0112]         //and all other fields with proper values.
```

```
[0113]         //Test if an User Object with key value
[0114]         // " XiaoWang" exists in the database ;
[0115]         boolean isUserExist = ds.isObjectExist(user) ;
[0116]         //Save all the value of xw into database ;
[0117]         ds.saveObject(user) ;
[0118]         //Load all infomation into another User
[0119]         //Object called xwCopy
[0120]         User xwCopy = new User() ;
[0121]         xwCopy.setUsername(" XiaoWang" ) ;
[0122]         ds.loadObject(xwCopy) ;
[0123]         //Read all users whose loginCounter < 3and
[0124]         //username likes xiao.
[0125]         LessFilter less = new LessFilter(" loginCounter" ,3) ;
[0126]         LikeFilter like = new LikeFilter(" username" , " xiao" ) ;
[0127]         AndFilter and = new AndFilter(less, like) ;
[0128]         List users = ds.getObjects(org.sample.User.class, and) ;
[0129]         //update xw' s loginCount
[0130]         int counter = xw.getLoginCounter()+1 ;
[0131]         xw.setLoginCounter(counter) ;
[0132]         ds.updateObject(xw) ;
[0133]         //delete xw' s all info from the database
[0134]         ds.deleteObject(xw) ;
[0135]     }
[0136] }
```

[0137] 用户调用 `List user = ds.getObjects(org.sample.User.class, and)` 并在该接口内部与底层数据库进行持久化操作时 LAPFA 的工作流程：

[0138] 第一，创建了一个待持久化类的对象 `xw`，并将其属性赋值。

[0139] 第二，调用 `isObjectExist` 接口判断 `xw` 是否存在于数据库中。

[0140] 第三，调用 `saveObject` 接口将 `xw` 保存到 `ldap` 数据库中。

[0141] 第四，创建另一个待持久化类的对象 `xwCopy`，并将主键赋值。

[0142] 第五，利用 `xwCopy` 读取刚才存入数据库中的 `xw` 信息。

[0143] 第六，根据查询需求创建了 Java 实体类查询对象 `and`。

[0144] 第七，调用 `getObjects` 接口进行数据查询，并将查询结果保存到变量 `users` 中。

[0145] 第八，更改 `xw` 的 `counter` 属性，并调用 `updateObject` 将其在 `ldap` 数据库中的数值更新。

[0146] 第九，调用 `deleteObject` 将 `xw` 从 `LDAP` 数据库中删除。

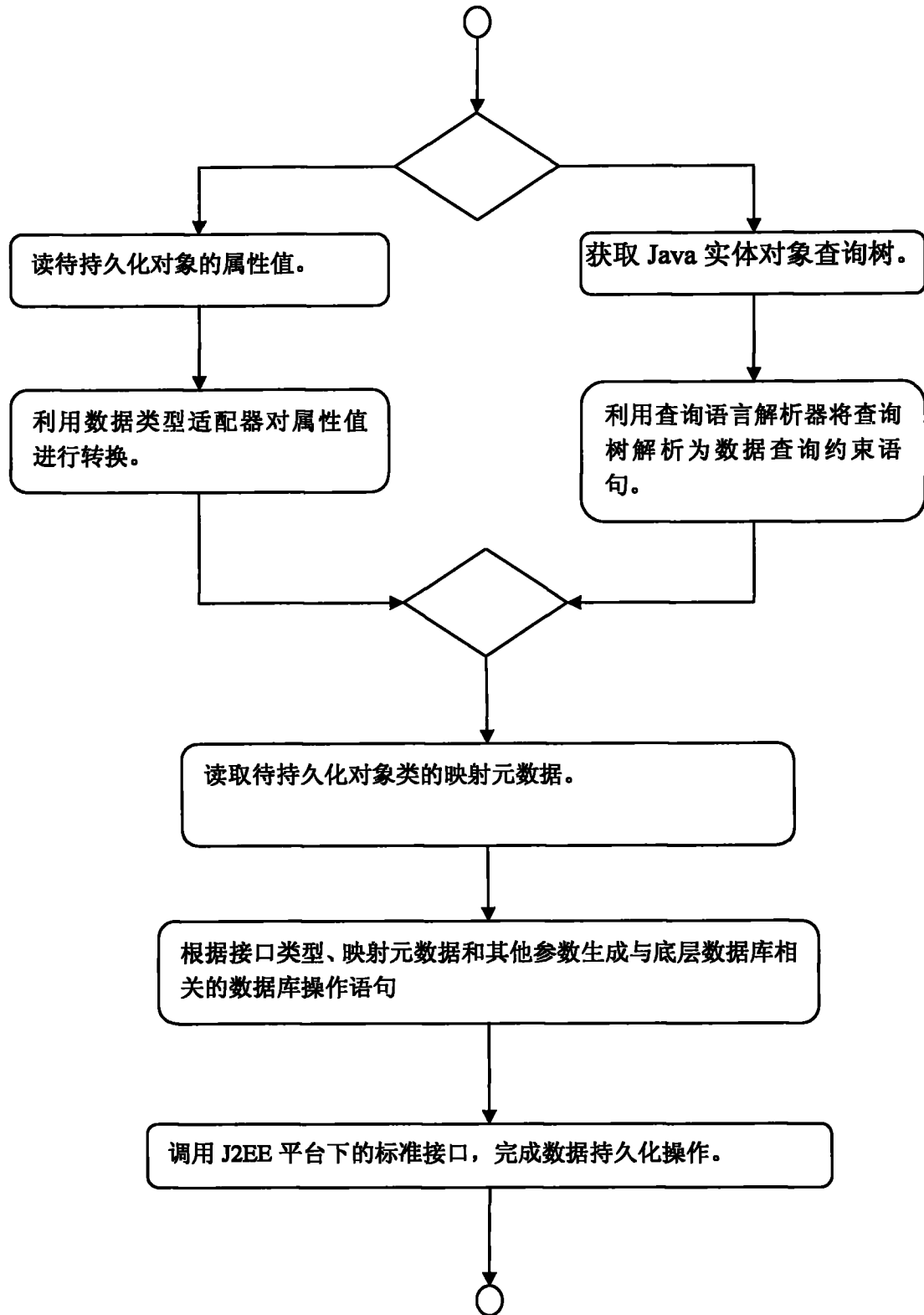


图 1

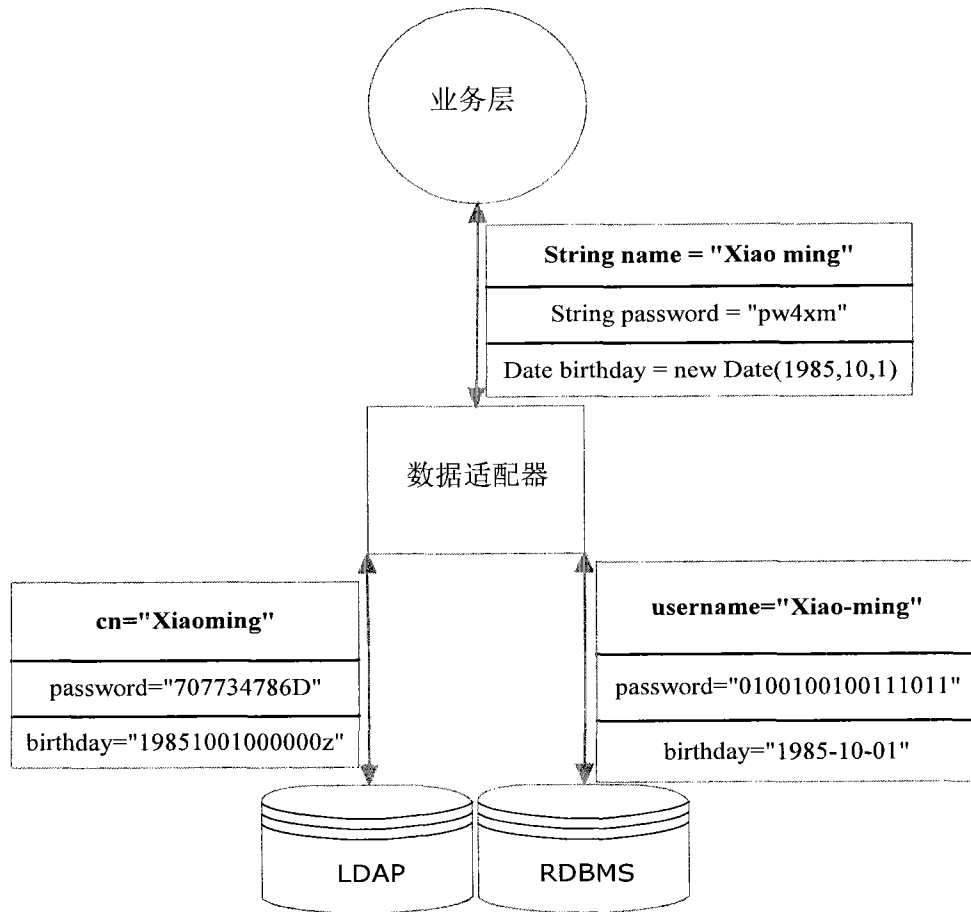


图 2

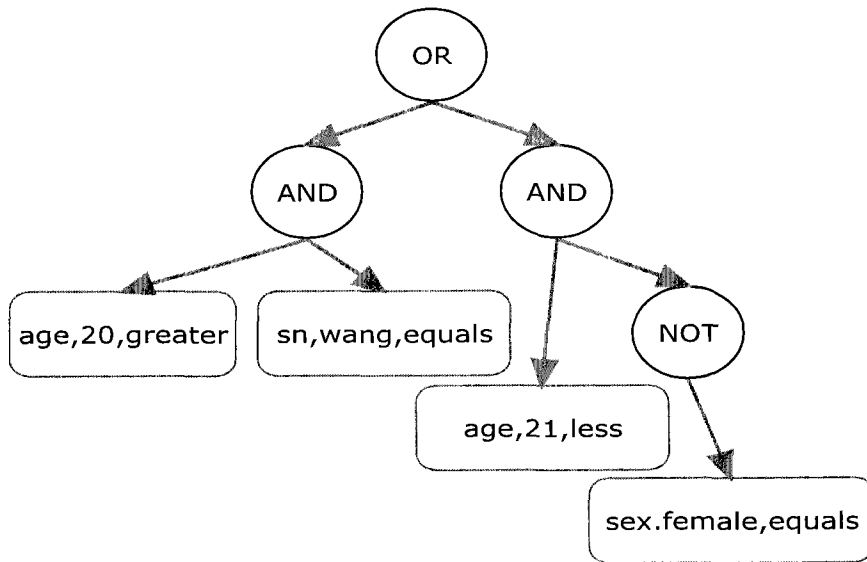


图 3

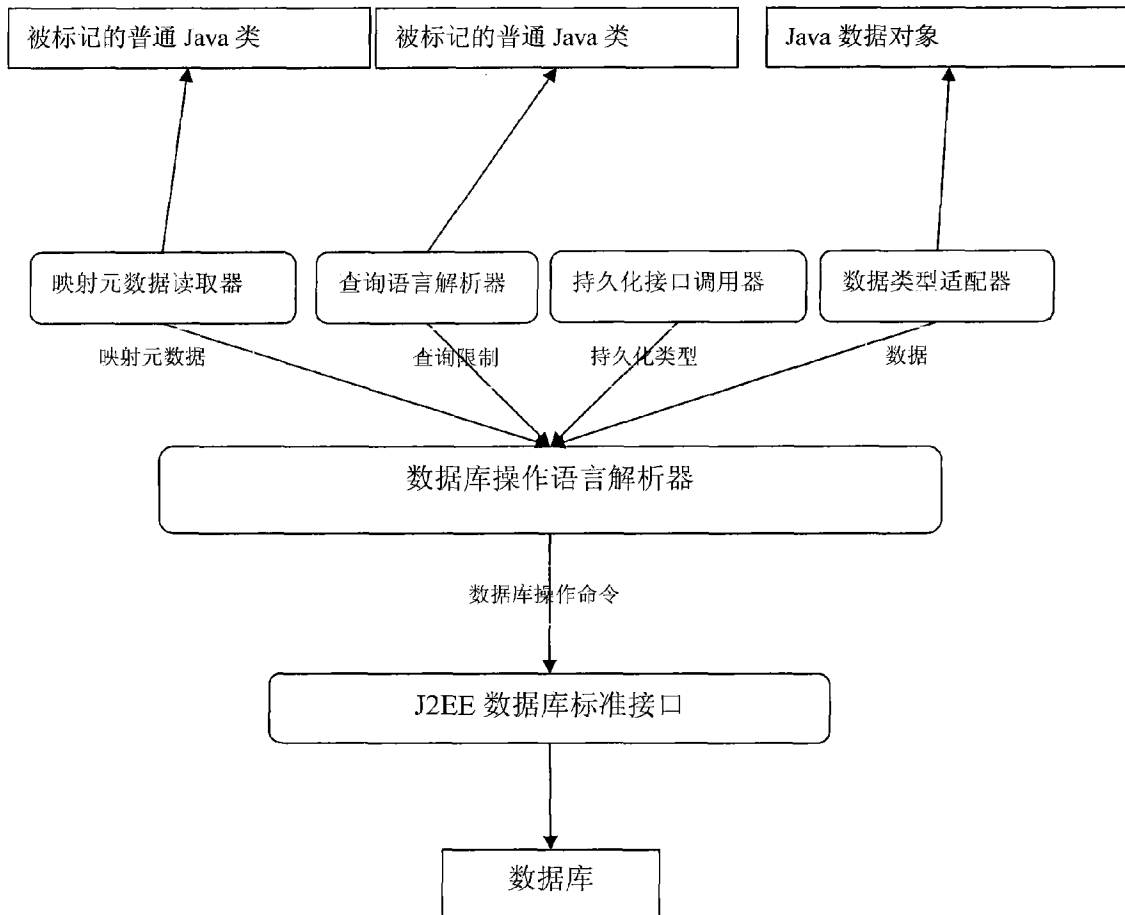


图 4

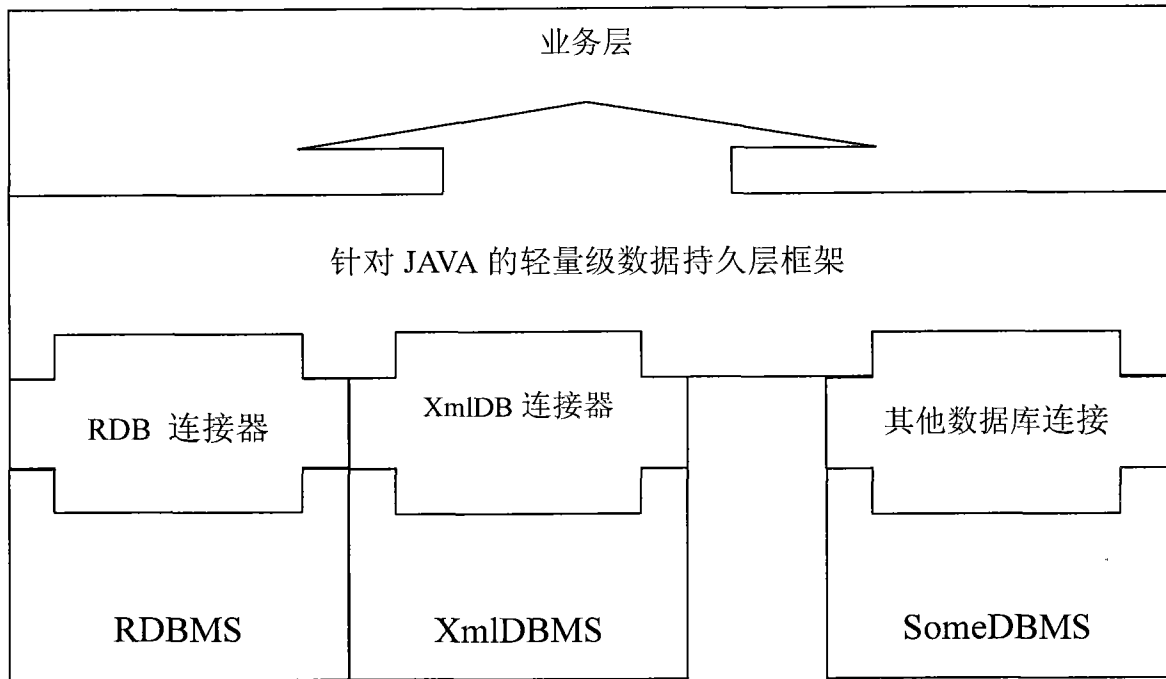


图 5