ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.19, No.5, May 2008, pp.1160−1172
DOI: 10.3724/SP.J.1001.2008.01160

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

网构软件数据语用的一种动态支撑方法*

滕 腾[1,2], 黄 罡[1,2+], 陈兴润[1,2], 梅 宏[1,2]

[1](北京大学 信息科学技术学院 软件研究所,北京 100871)

[2](北京大学 高可信软件技术教育部重点实验室,北京 100871)

An Approach to Dynamically Operating Data Pragmatics for Internetware

TENG Teng[1,2], HUANG Gang[1,2+], CHEN Xing-Run[1,2], MEI Hong[1,2]

[1](Software Institute, School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China)

[2](Key Laboratory of High Confidence Software Technologies of Ministry of Education, Peking University, Beijing 100871, China)

+ Corresponding author: E-mail: huanggang@sei.pku.edu.cn, http://www.sei.pku.edu.cn

Teng T, Huang G, Chen XR, Mei H. An approach to dynamically operating data pragmatics for Internetware. *Journal of Software*, 2008,19(5):1160−1172. http://www.jos.org.cn/1000-9825/19/1160.htm

Abstract: The uncertain data requirements of Internetware lead to the DP (data pragmatics) uncertainty. And then it is required that the DP operating mechanism has the ability to adapt to the uncertain DP, i.e. the corresponding relationships between application objects and database tables can be dynamically created, updated and removed on demand. As current persistence technologies lack of dynamic adaptation of DP, this paper proposes an approach to dynamically operating DP for Internetware, POD (persistence on demand), based on ORM (object/relational mapping). Together with the mainstream ORM framework, a prototype of persistence framework following POD is implemented.

Key words: data pragmatics; persistence; Internetware; middleware

摘 要: 网构软件不确定的数据需求导致了其数据语用(data pragmatics,简称 DP)的不确定性,进而要求数据语用的支撑机制具备动态适应能力,即应用的对象属性与数据表之间的对应关系可以按需动态地创建、更新、删除.针对当前持久化技术缺乏数据语用的动态适应性问题,提出了一种基于对象/关系映射的数据语用动态支撑方法POD(persistence on demand),并且在主流ORM(object/relational mapping)框架的基础上实现了一个遵循POD的持久化框架的原型系统.

关键词: 数据语用;持久化;网构软件;中间件

中图法分类号: TP311 **文献标识码**: A

数据是企业核心资产之一,"数据语义"描述的是数据本身内在、固定、不受外界因素(语境)影响的含义.在企业计算中,应用需要和数据结合在一起,共同完成业务逻辑功能.应用为数据提供了使用的场景,数据为应用的业务逻辑提供了必要的信息来源.数据的提供方称为数据源.应用程序中变量和数据源之间的结合方式、对

* Supported by the National Natural Science Foundation of China under Grant Nos.60403030, 90412011, 90612011 (国家自然科学基金); the National Basic Research Program of China under Grant No.2005CB321805 (国家重点基础研究发展计划(973))

Received 2007-06-10; Accepted 2007-10-15

应关系以及变量取值和数据项之间的相互转换过程构成了应用的"数据语用(data pragmatics,简称 DP)",即数据在应用特定的语境中的含义.数据语用可以用一个等式来描述:*DP=App*⟨+⟩*tab*.其中,*DP* 是指数据语用,*App* 是指应用,*DS* 是指数据源,符号"⟨+⟩"中的"+"是指应用中变量和数据源之间的结合方式及对应关系,"⟨⟩"是指变量取值和数据项之间的相互转换过程.由于面向对象的软件开发已成为当今主流,所以,本文中"应用"、"程序"在不加特别说明时都是指面向对象的应用.应用的数据语用通常由持久化框架(一种主流的中间件)实现,即对象状态根据业务逻辑而改变,持久化框架根据用户给定的数据语用策略(指定持久对象和目标数据源以及对象属性和数据项的对应关系),将对象状态的变动转换成一系列持久化操作并在目标数据源中执行,以保持应用状态和数据源的同步.

随着 Internet 的普及以及网构软件概念的提出[1];Internet 开放、动态、多变的特征对软件系统产生了巨大的影响[2].特别地,网构软件的数据需求在运行前可能无法明确,在运行中也可能持续变化,从而导致其数据语用呈现出典型的不确定性.期望开发阶段一劳永逸地解决不确定的数据语用既不合理,也不现实.换言之,问题的解决需要延伸到运行时刻,要求持久化框架必须具备动态适应的能力,即对象属性与数据表之间的绑定和映射关系的创建、更新、删除都可以动态地实现与调整.具体而言,持久化框架面对的动态性包括:(1) 对象持久态与非持久态的转换;(2) 数据源的切换;(3) 对象的持久属性与非持久属性的转换;(4) 对象主键属性和非主键属性的转换等等.由于目前的持久化框架主要针对相对封闭、静态、稳定的传统软件,仅提供数据语用的静态支撑机制,应用数据语用一旦确定并生效执行,那么在运行时刻将不能被更改,除非终止应用的运行,修改配置文件甚至源码,再重启应用让新数据语用策略生效.这种方式缺乏足够的灵活性,代价高昂.更为重要的是,网构软件往往要求提供 7 天×24 小时的不间断服务,这种离线调整方式显然不适用.

针对网构软件数据语用的不确定性以及现有持久化技术动态适应能力的不足,本文提出了一种数据语用的动态支撑方法:POD(persistence on demand),其核心思想在于,数据语用和应用完整分离后,为应用隐式注入数据语用实现及调整的机制,运行时通过中间件数据语用控制器的协调和监控,动态调整运行态对象的数据语用.本文用数学符号和抽象数据类型在抽象层次上对 POD 方法进行了形式化描述,与具体实现技术和机制相分离,提高了方法的通用性和实现的灵活性.

本文第 1 节介绍研究背景.第 2 节给出 POD 方法基本功能的形式化描述.第 3 节讨论 POD 原型系统实现的关键性技术.第 4 节给出 POD 原型系统的实例研究及相关性能测试数据.第 5 节讨论本文工作的不足与未来工作的重点.第 6 节讨论相关工作.第 7 节总结全文.

1 研究背景

本文提出的数据语用的概念借用了语言学中的"语用"一词.语法、语义和语用这 3 个术语是现代语言学研究中的重要概念.语法是语言构成和表达的规则,语义是指语言中的意义.传统的语义学家把意义看作是语言文字本身固有的属性,这种属性是内在的、固定的、不受外界因素如时间、地点等的影响,因而语义具有不变的特性.而语用是指利用语境表达出的意义,它可能与语言的语义不尽相同.语用和语义的不同之处在于,语义学研究的是不受语境影响的句子、词语等本身的意义,而语用学研究的是在语境中才能确定的意义,即语言使用上的意义[3].

根据语言学中语义和语用的概念,我们认为,数据语法是指数据的格式和编码.数据语义描述数据本身内在、固定、不受外界因素(语境)影响的含义.数据语用则描述了数据在应用特定的使用语境下表现出来的意义,可以把数据语用看作是应用对固有数据语义的一个具体解释.例如,就字符串"tom@163.com"而言,首先,它符合字符串语法规则:0 个或多个字符组成的有限序列;其次,它的数据语义指的是一个电子邮件地址.但其数据语用的情况有所不同:在某应用中,它可以被用作系统注册用户的唯一登录名,而在另一个应用中,它可能仅仅被用作注册客户的联系方式,这就是该字符串的两个不同的数据语用,前者的修改往往意味着注册了一个新用户,而后者的修改仅仅意味着已注册客户的一个属性发生了变化.该例表明,在不同的应用语境中,数据的语境特定的意义可以不一致.回顾计算机软件数据管理发展史,可以看到数据语用不同的展现形式:

- 在人工数据管理阶段,应用程序独占式地使用数据,数据只存在于程序运行期间,不保存也不共享,数据与程序之间没有独立性,数据本身就是程序的一部分.因此,数据的物理存在、数据语义以及数据语用都硬编码在程序中.
- 在数据管理的文件系统阶段,数据本身从应用中分离出来,保存在文件之中.数据文件的格式都是应用特定的.应用 A 如果需要共享应用 B 建立的数据文件,就必须了解应用 B 特定的数据文件格式和使用方式.因此,数据的物理存在已经从应用中分离了,但是数据语义仍然和应用紧密地绑定,而其他共享该数据的应用的数据语用仍然存在于各自的源码中.
- 在数据管理的数据库系统阶段,数据的物理存在以及数据语义(主要通过 Schema 体现)都集中在数据库中保存、定义和维护.数据不再从属于某个特定的应用,应用共享数据不再需要去了解最初建立该数据的应用.但是,应用的数据语用仍然存在于各应用自身源码中,动态适应能力较差.
- 在数据管理的中间件阶段,此时,应用的数据语用开始从源码中分离,通过配置文件进行描述,由中间件来具体实现,但应用代码中仍显式地存在与具体持久化机制相关的代码,动态适应能力不足.

不难发现,数据管理的发展趋势是数据物理存在、数据语义以及数据语用和应用之间逐步分离,以控制数据变化对应用的不良影响.Internet 使得数据变化的广度、深度、频率以及影响均达到前所未有的程度,自然需要将数据语用从网构软件应用代码中完全剥离,以实现网构软件与数据相关的自主性和适应性.

2 POD 方法原理

网构软件的不确定性数据语用的解决,需要数据语用的动态支撑方法.本文基于目前最重要的编程范型——面向对象模型以及最流行的数据库类型——关系数据模型,给出了 POD 方法中数据语用的形式化定义及其动态支撑技术原理的描述.POD 方法消除了传统软件系统中显式存在并由开发者控制的持久化操作,从而达到将数据语用从软件代码中完全剥离,以进行动态调整的目的.但是,应用对象状态仍然需要在合适的时机和目标数据源进行同步.因此,POD 方法需要给出隐式持久化操作的触发时机、条件及功能,只有如此,才可以在其基础上描述数据语用的动态调整过程.于是,本文运用集合论、一阶谓词逻辑、抽象数据类型等数学符号对 POD 方法中支撑数据语用的隐式持久化操作的功能、触发时机和条件进行描述,然后,在此基础上简明扼要地描述了 POD 方法中数据语用动态调整机制的原理.由于形式化描述中采用了数学符号和抽象数据类型,使得 POD 方法原理的描述都在抽象层次上进行,与具体实现技术和机制相分离,提高了方法的通用性、实现的灵活性以及理解的一致性.

2.1 数据语用定义

由于数据语用的定义涉及到数据源、数据表、应用、对象类等相关概念,因此,在定义数据语用时,必须首先定义这些相关的重要概念.

定义 1. 应用 $A\{OBJ_1,OBJ_2,OBJ_3,\ldots\}$,$OBJ_i$ 意即应用 A 中包括的业务对象类,每个类都可以根据用户需求设定为持久态或者是非持久态.

定义 2. 定义对象类的属性构成:$OBJ\{attr_1,attr_2,attr_3,\ldots\}$,$attr_i$ 代表对象类包括的属性,每个属性都可以根据用户需求设定为持久态或者是非持久态.定义 $OBJ_{set}\{obj_1,obj_2,obj_3,\ldots\}$ 为 OBJ 类的实例集合.$obj\in OBJ_{set}$ 意即 obj 是类 OBJ 的一个实例,而 $obj[attr_i]$ 代表对象实例 obj 在属性 $attr_i$ 上的取值.定义对象类主键属性集合:$OBJ_{PK}\{attr_1,attr_2,attr_3,\ldots\}$,其中,$attr_i$ 为主键属性.类实例的持久态唯一标识符由其主键属性取值的组合构成.

定义 3. 谓词 $isPER(x)$:x 为持久态,其中,$x\in A\{OBJ_1,OBJ_2,OBJ_3,\ldots\}$ 或 $x\in OBJ\{attr_1,attr_2,attr_3,\ldots\}$.

除了应用以外,数据是数据语用的另一个关键参与者,因此,本文接下来定义数据源和数据表,它们提供了映射应用对象类的可用数据项.

定义 4. 数据源 $DS\{tab_1,tab_2,tab_3,\ldots\}$,$tab_i$ 意即数据源 DS 中包含的可用数据表.

DS 中可用数据表 tab 的方案(schema)由一组字段(column)组成.

定义 5. 定义可用数据表的方案为 $tab\{col_1,col_2,col_3,\ldots\}$,$col_i$ 意即数据表 tab 包含的字段;定义数据表 tab 的

元组集合为 $tab_{set}\{t_1,t_2,t_3,\ldots\}$.$t\in tab_{set}$ 意指 t 是表 tab 中的一个元组,而 $t[col_i]$表示元组 t 在字段 col_i 上的投影值.定义对目标数据表的插入操作 $INSERT(tab_{set},t)$,意为将元组 t 插入到表 tab 中去.定义赋值操作 $ASSIGN(x,y)$,意为将 y 的值赋给 x.定义返回操作 $RETURN(x)$,意为返回 x 的取值.其中,x 的取值范围为对象实例的属性值以及元组在某个字段上的投影值.如果 x 为"NULL",表明执行一个空返回操作,即什么也不做.

定义 6. 符号"←→"描述对象类的一个持久属性和目标数据表的一个字段间的对应关系.

根据对象/关系映射规则的基本要求,应用 A 中所有持久态的对象类都要映射到 DS 中的数据表上.其中,每一个对象类映射到数据库中一个目标数据表,且该类的每个持久属性都要与该数据表中某个字段相对应.对象类和数据表之间的映射策略由一组持久态属性和字段之间的对应关系构成,即映射规则:

$$(\forall OBJ)((OBJ\in A\wedge isPER(OBJ))\rightarrow(\exists tab)(tab\in DS\wedge(\forall attr)((attr\in OBJ\wedge isPER(attr))\rightarrow(\exists col)(col\in tab\wedge attr\leftrightarrow col)))).$$

在定义了数据语用所涉及的相关重要概念之后,本文接下来定义应用对象类的数据语用.

定义 7. 定义应用对象类的数据语用 p 为一个三元组:$p(OBJ,\langle+\rangle,tab)$.其中,$OBJ$ 为对象类,tab 为该类绑定的目标数据表,符号"+"为 OBJ 到 tab 的一个绑定和映射的对应结合关系,符号"⟨⟩"为 OBJ 实例变量取值和 tab 数据项之间的转换过程.$p(OBJ,\langle+\rangle,tab)$描述了类 OBJ 和表 tab 之间一个遵循绑定和映射规则"+"的具体结合过程.三元组 p 中的目标数据表 tab 可以设为空值"NULL",即,OBJ 被设为非持久态;否则,OBJ 被设为持久态.

定义 8. 应用 A 的数据语用由其所有对象类的数据语用形成的一个非重复集组成,记作:$P_A\{p_1,p_2,\ldots,p_n\}$,其中,$n\in\mathbf{N}$,$\mathbf{N}$ 为自然数集.可以看出,应用的数据语用是其所有对象类数据语用的汇总.

在以上数据语用相关定义的基础上,下文描述 POD 方法为应用提供的数据语用的隐式持久化基本操作.

2.2 支撑数据语用的隐式持久化操作

对于类 OBJ 的实例 obj 而言,其状态发生改变当且仅当其属性的值被写入,其状态被获取当且仅当其属性的值被读取.虽然对象属性值的写入和读取本身并不涉及到持久化,但是对于持久性应用系统而言,其对象状态需要在合适时机和数据源进行同步以保持相互一致.因此,支撑数据语用的隐式持久化操作,其隐式触发的时机必须选取在对象实例的属性值被写入和读取时.为了达到隐式支撑数据语用的目的,需要对这两类属性访问动作进行必要的修改,为其加入相应的持久化操作及其触发条件,在此基础上,才能有效地实现数据语用的动态支撑.因此,在描述数据语用动态调整机制之前,本节首先描述读取和写入这两类对象属性的访问动作.

定义 9. 谓词 $write(obj.x,y)$:对象状态访问动作"将 obj 的属性 x 的取值设为 y"被执行了,$obj\in OBJ_{set},x\in OBJ,y\in Domain$,$Domain$ 为所有可赋值组成的集合.

定义 10. 谓词 $read(obj.x)$:对象状态访问动作"读取 obj 的属性 x 的取值"被执行了,$obj\in OBJ_{set},x\in OBJ$.

接下来分别讨论这两类属性访问动作的实现过程.为了判定对象的运行状态,我们给出以下谓词定义:

定义 11. 谓词 $isPKAttr(x,OBJ)$:如属性 x 是类 OBJ 的主键属性,即 $x\in OBJ_{PK}$,则值为真;否则为非真,$x\in OBJ$.

定义 12. 谓词 $isModified(x)$:如属性 x 取值已被改动过,且尚未被持久化,则值为真;否则为非真,$x\in OBJ$.

定义 13. 谓词 $hasBeenFirstSet(x)$:如属性 x 在对象被初始化后首次被赋值,则值为真;否则为非真,$x\in OBJ$.

定义 14. 谓词 $hasBeenSet(x)$:如属性 x 在对象被初始化后被赋过值,则值为真;否则为非真,$x\in OBJ$.

2.2.1 修改对象实例的属性取值 $write(obj.x,y)$的实现过程

定义 9 描述的是类 OBJ 的实例 obj 的属性 x 被赋值为 y 这样一个动作,下面讨论其过程的实现策略:

根据定义 6 中给出的映射规则,存在目标数据表 tab,其字段 col 与类 OBJ 的实例 obj 的属性 x 相对应,即 $obj.x\leftrightarrow tab.col$.于是,此时动作 $write(obj.x,y)$的实质就是将 obj 的属性 x 的取值 y 持久化到 tab 的字段 col 上.将该操作过程记作谓词 $persist(obj.x,y,tab.col)$,即,目标数据访问操作"在 obj 的属性 x 的取值被设为 y 后,向表 tab 的字段 col 同步数据"被执行,其实现逻辑描述如下:

1) x 为非主键属性:由于 x 为非主键属性,因此,tab 中不需要添加新记录.此时,如果 obj 的所有主键属性的取值都已经设定,那么说明实例 obj 的状态已被写入 tab,于是在 tab 中找到相应元组 t,将其在字段 col 上的投影值更新为 y 即可.

2) x 属于主键属性:由于 x 是主键属性,意味着此时可能需要向 *tab* 中添加新元组.由于 *OBJ* 的主键可能是由多个属性组成的复合主键,因此下面再分情况讨论:

- x 为单一主键:如果 x 是类 *OBJ* 的唯一主键属性,那么 x 取值的任何改变,都将导致对象持久态唯一标识符发生改变,这意味着新实例的生成.因此,如果此时该值所代表的元组不存在于 *tab* 中,那么需要在 *tab* 中插入新元组,以持久化 *obj* 的当前状态.
- x 为复合主键之一:如果属性 x 是类 *OBJ* 的复合主键属性之一,那么 x 取值发生的任何改变都会导致对象持久态唯一标识符发生改变:
  - A. 如果此时 *obj* 尚有其他某个主键属性从未被赋值,则说明 *obj* 的复合主键值尚未完全设定,因此不触发持久化操作;
  - B. 如果此时 *obj* 的所有除 x 之外的主键属性都已被赋值,假定这是属性 x 第 1 次被赋值,那么将触发一个插入操作,向 *tab* 中插入一条代表 *obj* 此刻状态的新元组;如果这不是属性 x 第 1 次被赋值,说明实例 *obj* 已经不是原来的对象实例了,已经变成了一个新的持久性的对象实例,此时,可能需要插入新元组.但是,由于用户的目标可能不仅仅是修改 x 的值,而有可能需要等到继续修改完其他主键属性值之后才认为是新实例的生成,因此,POD 采取的策略是根据应用实际情况让用户事先指定类 *OBJ* 的主键属性集合的一个非空子集 S,即 $S\neq\varnothing\wedge S\subseteq OBJ_{PK}$,只有当 S 中所有主键属性被同一批次地修改之后,才认为 *obj* 已转变成了一个新实例;如果此刻 *obj* 的复合主键值在 *tab* 中不存在,那么需要在 *tab* 中插入新元组,以持久化新实例 *obj* 的状态.

综上所述,以上给出的 *persist*(*obj.x*,*y*,*tab.col*)的完整触发条件和操作过程的形式化描述如下:

Formula a=$\neg isPKAttr(x,OBJ)\rightarrow((\forall attr)(attr\in OBJ_{PK}\wedge hasBeenSet(attr))\rightarrow((\exists t)(t\in tab_{set}\wedge obj[OBJ_{PK}]=t)\rightarrow ASSIGN(t[col],y))$;

Formula b=$isPKAttr(x,OBJ)\rightarrow(|OBJ_{PK}|=1\rightarrow((\neg\exists t)(t\in tab_{set}\wedge t[col]=y)\rightarrow INSERT(tab_{set},t)\wedge ASSIGN(t[col],y)))$;

Formula c=$isPKAttr(x,OBJ)\rightarrow(|OBJ_{PK}|>1\rightarrow((\neg\exists attr)(isPKAttr(attr,OBJ)\wedge x\neq attr\wedge\neg hasBeenSet(attr))\rightarrow(hasBeenFirstSet(x)\rightarrow((\neg\exists t)(t\in tab_{set}\wedge obj[OBJ_{PK}]=t)\rightarrow INSERT(tab_{set},t)\wedge t=obj[OBJ_{PK}]\wedge ASSIGN(t[col],y)))))$;

Formula d=$isPKAttr(x,OBJ)\rightarrow(|OBJ_{PK}|>1\rightarrow((\neg\exists attr)(isPKAttr(attr,OBJ)\wedge x\neq attr\wedge\neg hasBeenSet(attr))\rightarrow(\neg hasBeenFirstSet(x)\rightarrow(S\neq\varnothing\wedge S\subseteq OBJ_{PK}\wedge x\in S\wedge(\forall attr)(attr\in S\wedge attr\neq x\rightarrow isModified(attr))\rightarrow((\neg\exists t)(t\in tab_{set}\wedge obj[OBJ_{PK}]=t)\rightarrow INSERT(tab_{set},t)\wedge t=obj[OBJ_{PK}]\wedge ASSIGN(t[col],y)))))$.

于是有:*persist*(*obj.x*,*y*,*tab.col*)=a∨b∨c∨d.最后得出:*write*(*obj.x*,*y*)=*ASSIGN*(*obj.x*,*y*)∧*persist*(*obj.x*,*y*,*tab.col*).

2.2.2 读取对象实例属性取值 *read*(*obj.x*)的实现过程

当类 *OBJ* 实例 *obj* 的属性 x 被读取时,调用者的目的是要获知 x 的当前取值.同样,根据定义 6 中的映射规则,有 *obj.x*←→*tab.col*.于是,动作 *read*(*obj.x*)的实质就是将 *obj* 的属性 x 的取值从表 *tab* 的字段 *col* 上取得.于是,将此动作记作谓词 *query*(*obj.x*,*tab.col*),即目标数据访问操作"从表 *tab* 的字段 *col* 向 *obj* 的属性 x 同步数据"被执行.根据属性 x 的不同情况,*query*(*obj.x*,*tab.col*)的实现采取不同策略:(1) 如果 x 是类 *OBJ* 的非主键属性,那么首先根据 *obj* 的主键值去 *tab* 中查找相应元组 t,给 x 赋值为 t[*col*]后返回;若无对应元组 t,则说明该对象状态尚未被持久化,直接返回 x 的取值;(2) 如果 x 是 *obj* 的主键属性,则无论 x 的取值是否已被持久化,都只需直接返回 x 的当前取值即可.于是,以上叙述的 *query*(*obj.x*,*tab.col*)的完整触发条件和操作过程的形式化描述如下:

Formula a_2=$\neg isPKAttr(x,OBJ)\rightarrow((\exists t)(t\in tab_{set}\wedge obj[OBJ_{PK}]=t)\rightarrow ASSIGN(obj.x,t[col]))$;

Formula b_2=$isPKAttr(x,OBJ)\rightarrow RETURN(\mathrm{NULL})$.

于是有:*query*(*obj.x*,*tab.col*)=$a_2\vee b_2$.最后得出:*read*(*obj.x*)=*query*(*obj.x*,*tab.col*)∧*RETURN*(*obj.x*).

2.3 数据语用动态调整机制原理

在上一节给出的支撑数据语用的隐式持久化操作的基础上,本节描述 POD 方法中数据语用动态调整机制的原理.根据定义 7:$p(OBJ,\langle+\rangle,tab)$,类 *OBJ* 的数据语用策略的变化可以分为 3 类:(1) *OBJ* 从持久态转为非持久态,即 *tab* 被设定为 NULL,或者反之;(2) 目标数据表 *tab* 切换,由 tab_1 切换成 tab_2;(3) *OBJ* 和 *tab* 之间对应关系发生改变,即某个持久态属性转变为非持久态,或者反之;某个主键属性转变为非主键属性,或者反之.

当以上 3 类情况发生时,如何让 *OBJ* 的实例 *obj* 按照新设定的数据语用策略进行持久化,下面分别进行讨论.

2.3.1 对象持久态和非持久态之间相互转变

当正在运行中的应用的某个对象类 *OBJ* 从之前的持久态转变成非持久态时,用符号描述即 $p(OBJ,\langle+\rangle,tab)\Rightarrow p(OBJ,\langle+\rangle,\text{NULL})$,其中,符号"⇒"描述数据语用的变化.POD 采取的调整需将类 *OBJ* 正处于运行态的实例 *obj* 由持久态转入非持久态,此后,所有对 *obj* 的属性执行的访问动作 *write*(*obj.x*,*y*)及 *read*(*obj.x*)都将只在对象内部进行,不再触发任何持久化操作,整个执行策略的调整过程描述如下:

Adaptation: $p(OBJ,\langle+\rangle,tab)\Rightarrow p(OBJ,\langle+\rangle,\text{NULL})$;

根据定义 6 的映射规则,调整前,*tab* 的字段 *col* 和 *OBJ* 实例 *obj* 的属性 *x* 相对应,即 *obj.x*←→*tab.col*,于是:

对 *write*(*obj.x*,*y*):

Before adaptation: *ASSIGN*(*obj.x*,*y*)∧*persist*(*obj.x*,*y*,*tab.col*);

After adaptation: *ASSIGN*(*obj.x*,*y*);

对 *read*(*obj.x*):

Before adaptation: *query*(*obj.x*,*tab.col*)∧*RETURN*(*obj.x*);

After adaptation: *RETURN*(*obj.x*);

以上描述的是 $p(OBJ,\langle+\rangle,tab)\Rightarrow p(OBJ,\langle+\rangle,\text{NULL})$的情况.同理可以给出相反情况,即 $p(OBJ,\langle+\rangle,\text{NULL})\Rightarrow p(OBJ,\langle+\rangle,tab)$的整个执行策略的调整过程:

Adaptation: $p(OBJ,\langle+\rangle,\text{NULL})\Rightarrow p(OBJ,\langle+\rangle,tab)$;

根据定义 6 的映射规则,调整后的情况为 *obj.x*←→*tab.col*.于是:

对 *write*(*obj.x*,*y*):

Before adaptation: *ASSIGN*(*obj.x*,*y*);

After adaptation: *ASSIGN*(*obj.x*,*y*)∧*persist*(*obj.x*,*y*,*tab.col*);

对 *read*(*obj.x*):

Before adaptation: *RETURN*(*obj.x*);

After adaptation: *query*(*obj.x*,*tab.col*)∧*RETURN*(*obj.x*);

2.3.2 目标数据表 *Tab* 切换

当正在运行的应用的某个对象类的目标数据表发生了调整,即 $p(OBJ,\langle+\rangle,tab_1)\Rightarrow p(OBJ,\langle+\rangle,tab_2)$时,对于类 *OBJ* 正处于运行态的实例 *obj* 而言,从调整事件发生时刻起,当 *obj* 被访问时,需要调整为按照新的数据语用策略执行持久化操作.为了保证系统调整前后的数据持久状态的一致性,需要首先判断 *obj* 的状态是否在新目标数据表 tab_2 中存在:如果不存在,则应先将 *obj* 当前的状态持久化到新目标数据表 tab_2 中,然后才能执行对 *obj* 的属性的访问操作 *write*(*obj.x*,*y*)及 *read*(*obj.x*);否则,表 tab_2 将会丢失与 *obj* 的状态相应的数据.

根据定义 6 中给出的映射规则,在调整发生之前,表 tab_1 的字段 col_1 与类 *OBJ* 的对象实例 *obj* 的属性 *x* 相对应,即 $obj.x\leftrightarrow tab_1.col_1$;而在调整发生之后,则是表 tab_2 的字段 col_2 与类 *OBJ* 的对象实例 *obj* 的属性 *x* 相对应,即 $obj.x\leftrightarrow tab_2.col_2$.于是整个执行策略的调整过程描述如下:

Adaptation: $p(OBJ,\langle+\rangle,tab_1)\Rightarrow p(OBJ,\langle+\rangle,tab_2)$;

对 *write*(*obj.x*,*y*):

Before adaptation: *ASSIGN*(*obj.x*,*y*)∧*persist*(*obj.x*,*y*,$tab_1.col_1$);

After adaptation: $ASSIGN(obj.x,y)\wedge((\neg\exists t)(t\in tab_{2set}\wedge obj[OBJ_{PK}]=t)\rightarrow INSERT(tab_{2set},t)\wedge t=obj[OBJ_{PK}])\wedge ASSIGN(t[col_2],y)\wedge persist(obj.x,y,tab_2.col_2)$;

对 *read*(*obj.x*):

Before adaptation: $query(obj.x,tab_1.col_1)\wedge RETURN(obj.x)$;

After adaptation: $((\neg\exists t)(t\in tab_{2set}\wedge obj[OBJ_{PK}]=t)\rightarrow INSERT(tab_{2set},t)\wedge t=obj[OBJ_{PK}])\wedge query(obj.x,tab_2.col_2)\wedge RETURN(obj.x)$;

2.3.3 类 *OBJ* 和目标数据表 *Tab* 之间的对应关系发生改变

从正在运行的应用中的某个对象类的属性 *x* 由持久态属性转变为非持久态属性时起,所有正在运行的 *OBJ* 实例 *obj* 的 *x* 属性的取值都将被非持久化处理.根据定义 6 的映射规则,调整前 *tab* 的字段 *col* 和 *OBJ* 的实例 *obj* 的属性 *x* 相对应,即 *obj.x*←→*tab.col*.调整后情况变为 *obj.x*←→NULL,于是,调整过程可以描述为:

Adaptation: *obj.x*←→*tab.col*⇒*obj.x*←→NULL;

对 *write*(*obj.x*,*y*):

Before adaptation: $ASSIGN(obj.x,y)\wedge persist(obj.x,y,tab.col)$;

After adaptation: $ASSIGN(obj.x,y)$;

对 *read*(*obj.x*):

Before adaptation: $query(obj.x,tab.col)\wedge RETURN(obj.x)$;

After adaptation: $RETURN(obj.x)$;

同理可描述相反情形:属性 *x* 由非持久态属性转变为持久态属性,该调整过程可以描述为:

Adaptation: *obj.x*←→NULL⇒*obj.x*←→*tab.col*;

对 *write*(*obj.x*,*y*):

Before adaptation: *ASSIGN*(*obj.x*,*y*);

After adaptation: $ASSIGN(obj.x,y)\wedge persist(obj.x,y,tab.col)$;

对 *read*(*obj.x*):

Before adaptation: *RETURN*(*obj.x*);

After adaptation: $query(obj.x,tab.col)\wedge RETURN(obj.x)$;

OBJ 和 *tab* 之间的对应关系发生改变的另一种情况是属性 *x* 由主键属性转变成非主键属性,从此时起,对该属性的访问策略也将会发生变化,即 $isPKAttr(x,OBJ)\Rightarrow\neg isPKAttr(x,OBJ)$,或者相反的情形:$\neg isPKAttr(x,OBJ)\Rightarrow isPKAttr(x,OBJ)$.对于第 1 种情形,在调整后将会按照非主键属性的规则执行对 *obj* 的属性 *x* 的访问动作 *write*(*obj.x*,*y*)和 *read*(*obj.x*).具体过程参见本文第 2.2.1 节中(Formula a)和第 2.2.2 节中(Formula a_2)部分的分支执行情况;对于第 2 种情形,在调整后将会按照主键属性的规则执行对 *obj* 的属性 *x* 的访问动作 *write*(*obj.x*,*y*)和 *read*(*obj.x*).具体过程参看本文第 2.2.1 节中(Formula b)~(Formula d)以及第 2.2.2 节中(Formula b_2)部分的分支执行情况.

总之,以上 3 种调整策略并未改变对象实例自身状态,只是改变了对象属性和目标数据表之间的对应关系及其取值和数据项之间的相互转换过程.因此,该调整对应用自身状态没有破坏性.但是,假如用户对数据语用的调整存在不合理之处,例如,某个原本为非持久态的类在转变成持久态之后对数据表 *tab* 进行了不合理的修改,而导致其他映射到 *tab* 的类找不到所期望的数据,就可能出现数据依赖和干扰,并由此引发数据一致性被破坏.但这并非是实施了 POD 数据语用调整策略的缘故,因为此时即便开发者以静态方式进行人工调整也将面临同样问题.问题的根源在于调整前后的数据语用策略之间存在冲突.而关于数据依赖和干扰导致数据状态一致性被破坏的问题,其基于中间件的自动化解决方案此前已在我们的另一部分工作中做过研究[4],因而不再赘述.

3 系统实现

应用数据语用在运行时刻能够被动态调整,需要满足两个要求:(1) 数据语用策略可以独立于应用源码进行配置;(2) 数据语用策略的调整能够对所有正在运行中的业务对象即时生效.对于第 1 个要求,POD 原型系统的做法是根据"关注点分离(separation of concerns)"的思想,将动态实现和调整数据语用的代码从业务逻辑中完整地抽取出来,然后,POD 原型系统可定制地将数据语用动态实现及调整所必须的代码覆盖到系统各个业务对象中,这些代码在运行时刻将根据开发者或部署者在配置文件中描述的数据语用策略进行工作,使得业务代码中无须出现任何持久化代码就能具备数据语用支撑的能力.对于第 2 个要求,为了让所有运行态对象实例都能及时按照新的数据语用策略运作,需要一个全局控制器来负责获取和维护对象实例状态信息以及应用的数据语用配置信息,以精确协调和控制每个对象实例的每一次状态改变和获取的行为,并在需要时调用持久化接口进行实际的持久化操作.这个控制器也必须同时监测数据语用配置文件的变动,及时发现和获取新设定的数据语用策略,动态地管理和切换被使用的数据语用策略,POD 原型系统的实现原理如图 1 所示.

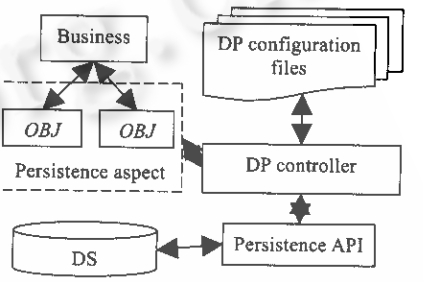

Fig.1 POD prototype illustrative diagram
图 1 POD 原型示意图

3.1 数据语用的隐式注入

POD 原型系统的实现基于"关注点分离"思想的典型技术:AOP(aspect-oriented programming,面向侧面编程).POD 将与数据语用相关的持久化代码从应用的业务逻辑中完整地抽取出来,这些代码在应用运行前由 POD 工具自动注入到类中预定的插入位置.由于这些持久化代码分散在应用的各个模块中,因此,我们称其为持久化侧面(aspect).在本文的工作中,持久化侧面是利用 AspectJ 实现的,AspectJ 是目前基于 Java 的最重要的 AOP 实现.持久化侧面用来描述对象类的数据语用逻辑.

POD 利用 AspectJ 的"类型间声明"技术定义了两个数据语用插入点:属性的 setter/getter 方法执行的前后,被注入的持久化侧面在 PODAspectJUtil 中定义,如图 2 所示.持久化侧面为对象类注入了持久化能力,但是,为了维护对象实例运行时与持久化相关的信息,还需要相应数据结构的支持.于是,本文定义了 PODBean 接口,如图 3 所示.实现这个接口的对象类具有数据语用动态支撑机制所需的数据结构.类不需要自己实现 PODBean 接口,该过程是由 POD 工具通过 AspectJ 的"类型间声明"技术对类文件的字节码进行处理完成的.

```
Pointcut setVal(PODBean p):
        Execution(**.set*(..)) && this(p);
Pointcut getVal(PODBean p):
        Eecution(*.get*(..)) && this(p);
```

Fig.2 Code segment of PODAspectJUtil
图 2 PODAspectJUtil 代码段

```
Public interface PODBean extends serializable {
boolean hasPrimaryKey=false;
// to indicate whether obj has primary key attributes
boolean inPersist=false;   // to indicate whether obj is in persistence
Integer version=0; // to store the version of obj
HashMap<String,Integer> idsMap=new HashMap<String,Integer>();
// to store the value versions for all primary key attribute
boolean setNewPartId(String attr);
// to indicate whether attr is a primary key attribute
boolean isReadyForNewInsert();
// to indicate whether a create manipulation should be executed}
```

Fig.3 Code segment of PODBean
图 3 PODBean 代码段

3.2 数据语用控制器

期望被隐式注入了数据语用的对象实例能够根据当前的语用策略动态地执行持久化操作,需要一个全局数据语用控制器来负责维护对象运行时的状态信息,协调对象实例的持久化操作.在实例第 1 次被创建时会触

发数据语用控制器的初始化,执行以下操作:建立数据库连接,读入数据语用配置文件,获取并缓存用户对对象类的语用的配置,开启监控线程监测配置文件的改变.当某实例 *obj* 的属性访问方法被调用时,*obj* 首先联系数据语用控制器,询问下一步动作;控制器接到请求后,根据缓存的语用策略,并通过持久化接口对目标数据进行必要的查询,返回消息给 *obj*;*obj* 根据控制器的回复执行相应操作.该过程的实现规则已在第 2.2 节中详细描述.另一方面,在运行时刻,数据语用控制器会在后台执行一个监控线程,监测数据语用配置文件的变动,一旦有变动发生,控制器则立刻解析新的配置文件,废弃原有策略并使新策略生效,从而实现了数据语用策略的动态调整,该过程的实现规则已在第 2.3 节中进行了详细描述.

POD 使用了目前最流行的对象/关系映射框架 Hibernate 的持久化接口.用户需要提供以下配置文件:

(1) 数据库配置文件:数据库设置,连接池配置,缓存配置,映射文件路径.

(2) 对象/关系映射文件:类/表对应关系、属性/字段映射关系、对象主键属性设置.

POD 数据语用控制器的核心类 *DPController* 的功能是记录所有对象类当前的数据语用策略,监测配置文件的改变,维护数据库连接,负责调用 Hibernate 提供的持久化接口完成实际的持久化操作.*DPController* 的主要数据结构如图 4 所示.

```
Configuration configuration;   // to save configuration information
SessionFactory sessionFactory;   // to create database session
// store meta-info of primary key attributes
HashMap<Class,HashMap<String,Integer>>persistClassMap;
```

Fig.4 Code segment of DPController

图 4 DPController 代码段

4 实例研究

为了验证 POD 原型系统为开放、多变、难控的 Internet 环境下应用系统提供的数据语用动态支撑机制,本文选取了 Spring Framework 开源项目(http://www.springframework.org)中的示例应用:Java Pet Store(JPS)展示 POD 对应用系统数据语用的动态实现及适应能力.该 Web 程序基于 Spring 的 MVC 框架实现,提供了基于 Internet 环境的多层结构的在线企业应用"宠物商店"的基本功能.

4.1 数据语用的隐式支持

数据语用的隐式支持是数据语用动态适应机制的基础,首先,我们为 JPS 隐式实现数据语用.JPS 的购物车模块原本并无持久化功能,假定在运行一段时间后,业务需求发生变化:需将用户使用购物车进行购物的过程记录下来,以便进行用户购物信息的收集整理和统计.我们只需为 JPS 编写数据语用的配置文件,然后用 POD 工具对购物车类进行数据语用逻辑的注入,即可使本无持久化功能的 JPS 购物车模块从此具有持久化功能.

为该变化了的业务需求制定的数据语用策略为:购物车类 *Cart* 映射到目标数据表 *cart*,将使用该购物车的用户名设置为其主键属性;类 *CartItem* 映射到目标数据表 $cartItem_1$,并将其主键设置为由 POD 自动注入的属性 *cartItemID*,在配置文件中将二者建立关联.此时,根据数据语用的实现规则,用户登录后,如果表 *cart* 中没有相应记录,则 POD 会向其中插入一条记录;当用户向购物车中添加/修改商品时,POD 会向 $cartItem_1$ 中添加/修改记录.

现在以用户 j2ee 登录网站,添加两项物品到购物车中,此时查看表 *cart* 和 $cartItem_1$,内容分别见表 1 和表 2,说明原本没有被持久化的购物车类 *Cart* 和 *CartItem* 现在已经可以将其实例的状态记录在数据表中了.

Table 1 Records in target table *cart*

表 1 目标数据表 *cart* 中的记录

Username	LastShoppingTime
j2ee	2007-08-20 15:30:05

Table 2 Records in target table $cartItem_1$

表 2 目标数据表 $cartItem_1$ 中的记录

Cartitemid	Itemid	qty	Instock	Hasprimarykey	Username	Cartidx
1	EST-18	2	1	1	j2ee	0
2	EST-8	1	1	1	j2ee	1

4.2 数据语用的动态调整

假定用户数据需求继续发生变化:购物车的某些信息如代表物品数量的属性 qty 不再需要被持久化,并且此时,类 *CartItem* 的目标数据表也需要由 $cartItem_1$ 切换成 $cartItem_2$.我们并不停止 JPS 的运行,而是直接在线修改配置文件:将类 *CartItem* 的目标数据表改为 $cartItem_2$;再将其属性 qty 从持久性属性映射列表中删除;然后用户 j2ee 再登录 JPS 进行购物.

在添加了两项商品到购物车中以后,查看新目标数据表 $cartItem_2$ 的内容,见表 3(与表 2 相比,表 3 没有 qty 列).说明购物车的状态已经写进了新的目标数据表,同时,对类 *CartItem* 的持久属性的调整也已生效.

Table 3 Records in target table $cartItem_2$ after the policy adaptation

表 3 策略调整后目标数据表 $cartItem_2$ 中的记录

Cartitemid	Itemid	Instock	Hasprimarykey	Username	Cartidx
1	EST-18	1	1	j2ee	0
2	EST-8	1	1	j2ee	1

本例说明,POD 原型系统成功地实现了对应用系统数据语用的动态适应.

4.3 数据语用动态支撑机制对业务的增强

原始的 JPS 应用的购物车模块并不具备持久化能力,因此,如果用户购物时未提交订单而中途退出,由于之前并未保存购物车中的内容,因而用户再次登录后将发现购物车为空,则购物过程必须从头开始.假定此时业务需求继续发生变化:如果用户没有提交订单就退出(技术故障或人为因素),那么要求在该用户下次登录时,退出前的购物信息仍然存在,用户可继续上次未完成的购物.于是,我们在购物过程中关闭浏览器,模拟用户非正常退出的情形.由于此时使用的是 POD 工具处理过后的 JPS,第 4.1 节的实验已经设定隐式数据语用将数据保存,于是我们在线修改配置文件,让类 *Cart* 和 *CartItem* 自动加载数据.之后,当用户 j2ee 再次登录查看购物车时,可以发现其中仍保留着浏览器关闭前已选购的商品清单.这是因为给类 *Cart* 设置主键后,POD 发现表 *cart* 中有对应的记录,且表 $cartItem_1$ 中存在与之关联的类 *CartItem* 的记录,会从数据表向对象进行同步.本例说明,POD 能够以在线的方式及时、动态地调整增强对象的数据语用,自动恢复应用系统的运行状态,从而在原有应用不改变、不感知的前提下提高应用的可靠性以及用户的满意度,使业务过程得以充分利用持久化带来的价值.

4.4 性能测试

为了测试 POD 原型系统的性能,针对 JPS,本文对比测试了 3 种情况:(1) 原始的未经修改的 JPS;(2) 修改后的使用 Hibernate 显式地对购物车类进行静态持久化的 JPS;(3) 利用 POD 原型系统实现了数据语用动态支撑的 JPS.

图 5(a)展示了 3 种情况下热部署 JPS 应用系统的耗时(各执行 10 次).我们发现,无论是使用 Hibernate 还是 POD 原型系统,持久化的引入都将导致部署时间增加.与 Hibernate 相比,使用 POD 原型系统部署 JPS 的时间更长,原因在于,POD 中数据语用动态支撑机制的初始化在部署时进行,而 Hibernate 是在运行时进行.

图 5(b)比较了三者执行购物操作的速度(各执行 10 次).我们发现,使用 Hibernate 和 POD 原型系统,持久化的引入都会导致操作耗时的增加.另外,使用 POD 的数据语用动态支撑机制与在程序中直接调用 Hibernate 进行持久化相比,除了第 1 次以外,其余操作的耗时大致相当.这是因为当直接使用 Hibernate 时,程序第 1 次操作结束时需要加载持久化机制,而 POD 在部署时刻就已经加载了持久化机制,所以在第 1 次执行时 POD 耗时更少.

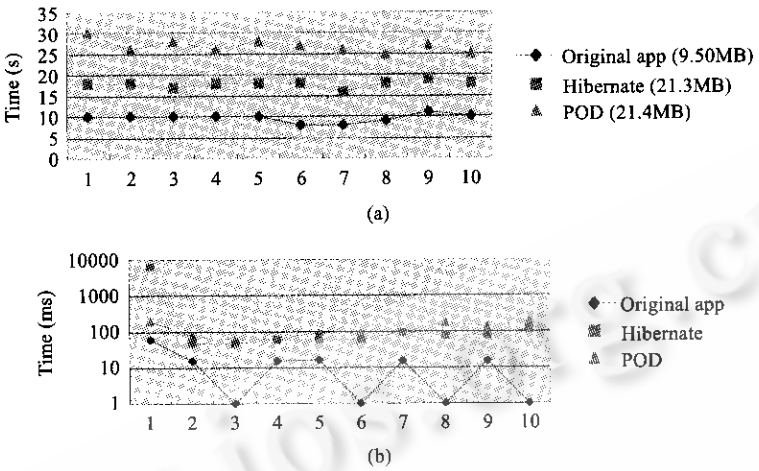

Fig.5 Performance test

图 5 性能测试

5 讨 论

本文的工作还存在着一些局限与不足之处,主要包括:对象关联属性的持久化,可定制事务的支持、查询和删除接口的提供等等,这些都是我们未来工作的重点.

目前,POD 方法的数据语用动态支撑机制尚未支持对象关联属性,对象关联属性的持久化也是数据语用支撑机制的一部分,是正交持久化完整性规则[5,6]的要求,这是我们未来工作的一个重点.

在数据语用操作的事务性方面,由于数据语用被隐式地注入业务代码,业务代码中显然不应该出现显式的事务调用.但是,应用的某些业务方法是需要事务性的,以保证其数据访问操作的原子性.POD 原型系统目前采取的策略是默认将每一个持久化操作都运行在独立的事务中.在我们下一步的工作中,POD 将为用户提供可定制的事务服务,用户可以在语用配置文件中指定某个类的哪些业务方法必须运行在同一个事务中,然后,POD 原型系统会将这些业务方法中涉及的所有对象的状态访问操作放在同一个事务上下文中进行.

本文中数据语用的基本操作并未包括查询和删除.我们认为,查询和删除是数据库针对持久数据为用户提供的功能,并非面向对象模型中定义的对象行为,对象操作其状态的行为只有读和写.由于没有对应的对象行为,查询和删除持久数据必然在代码中显式出现.而一旦代码中出现了显式查询和删除操作,则说明在开发时该应用就已确定了其数据语用策略,失去了动态适应能力.而对需求多变的网构软件而言,对象是否需要持久化以及采取何种数据语用策略,是难以完全在开发时刻确定的.但是,为了支持某些具有明确持久化目标的特定应用的需求,使得 POD 方法更具实用性,POD 原型系统事实上提供了显式的查询和删除接口.

6 相关工作

Internet 的发展对数据持久化技术提出了新要求,带来了新挑战.为了满足应用对数据统一管理、无缝集成、可用性、易用性以及动态适应性等方面的需求,研究人员在数据库和中间件领域都作出了卓有成效的努力,但是,在处理网构软件不确定的数据需求时,在数据语用的动态适应性方面仍存在不足:

在数据库研究领域,文献[7−9]从数据合并的思想出发,研究如何通过集成关系、XML 等异构数据来构建集中式的数据仓库.但是,数据仓库中的数据是被集成数据的静态复制,不能动态更新,难以及时反映数据源的变动[10],因而在数据需求持续变化的 Internet 环境中会面临严重问题.而文献[11−13]则从联邦数据库系统的联合查询优化、跨多个异构数据源的事务和安全实现等多个角度出发,研究基于数据联邦思想的数据集成.联邦数据库系统是由一些完全独立和自治的成员数据库组成的集合,各成员数据库和联邦互相协作以提供全局数据

操作.联邦数据库系统集中关注解决的是异构数据集成问题,但是,它最严重的缺陷在于使用联邦数据库系统的应用必须在全局查询里明确地指定所要使用的成员数据库.这意味着当数据需求或环境发生变化时,应用本身不得不作出调整后才能使用新的成员数据库[10],因此难以动态适应不断变化着的数据语用.

在中间件领域,数据语用支撑机制是数据管理中间件研究的核心内容,而目前最为重要和流行的数据管理中间件就是对象/关系映射(object/relational mapping,简称 ORM)持久化框架.文献[14,15]给出了经典 ORM 持久化框架的一个通用设计方案.本文的 POD 原型系统就是基于 ORM 的.POD 本质上属于正交持久化理念的范畴[5,6].通过数据语用的隐式注入机制,实现了持久化的正交性和独立性的目标.典型的 ORM 框架包括:

- EJB2.0[16]:使用 CMP(container-managed persistence,容器管理的持久化)将数据访问细节从业务代码中抽取出来,在部署描述符中指定持久化策略.其缺点是应用 CMP 实体 EJB 构件必须实现诸多生命周期管理的容器回调接口,而且它只能在容器中运行.更为严重的是,容器不提供数据语用的动态支撑机制.
- EJB3.0[17,18]:摒弃了 EJB2.0 中招致严重性能开销的实体 EJB 类、home 接口、remote 及 local 接口,取而代之的是一个处理 POJO(plain old Java object)[19]对象的持久化管理器.其缺点是,应用对象仍然必须运行在容器中,并且需要在业务代码中显式地调用持久层接口,因而无法动态支撑应用的数据语用.
- Hibernate[20]:是目前最流行的 ORM 框架,具有轻量级的优点,无需重量级的容器支持,但是也需要在业务代码中显式地调用持久层的编程接口,因而也不具备数据语用的动态支撑能力.
- JDO[21,22]:是一个轻量级 Java 对象持久化规范,它能够透明地将 POJO 类和底层数据源中的数据对应起来,其能力不局限于 ORM,它可以支持多种数据源类型,如文本、XML 数据库、对象数据库等等.为了让需要持久化的 POJO 类与底层数据源挂钩,需要对编译生成的.class 文件进行字节码级改造,这个过程称作增强.但是,增强仅仅为 POJO 类加入了持久化能力,而并未注入数据语用动态适应的机制.

开源组织 Source Forge 的项目 JPA(Java persistence aspect)[23],用 AspectJ 为 POJO 类实现一个持久化侧面,侧面调用持久层接口进行持久化.但由于缺乏中间件的协调和监控,因而也不具备数据语用的动态适应能力.

7 结束语

本文旨在通过一种动态数据语用支撑方法解决网构软件的数据语用不确定性的问题.本文的主要贡献在于:首先,提出了数据语用的概念,从数据语用的角度分析了目前的持久化技术在适应网构软件动态变化的数据需求时所面临的问题;其二,提出了一种数据语用动态支撑方法 POD,以形式化的方式阐述其基本原理;其三,在 POD 方法指导下,实现了一个基于主流 ORM 框架的提供数据语用动态支撑机制的持久化框架原型系统;最后,在 POD 原型系统上进行了实例研究,给出并分析了性能测试结果,验证了 POD 方法的有效性与可行性.

References:

[1] Lü J, Ma XX, Tao XP, Xu F, Hu H. Internetware: Research and progress. Science in China (Series E), 2006,36(10):1037−1080 (in Chinese with English abstract).

[2] Mei H, Huang G, Zhao HY, Jiao WP. A software architecture centric engineering approach for Internetware. Science in China (Series E), 2006,49(6):702−730 (in Chinese with English abstract).

[3] He ZX. A New Introduction to Pragmatics. Shanghai: Shanghai Foreign Language Education Press, 2000 (in Chinese).

[4] Teng T, Huang G, Chen X, Mei H. Towards automated resolution of undesired interactions induced by data dependency. In: Ouabdesselam F, Bousquet LD, eds. Proc. of the 9th Int'l Conf. on Feature Interactions in Telecommunications and Software Systems. Amsterdam: IOS Press, 2007. 55−60.

[5] Atkinson MP, Daynes L, Jordan MJ, Printezis T, Spence S. An orthogonally persistent Java. ACM SIGMOD Record, 1996,25(4): 68−75.

[6] Atkinson M, Morrison R. Orthogonally persistent object systems. VLDB Journal, 1995,4(3):319−401.

[7] Chaudhuri S, Dayal U. An overview of data warehousing and OLAP technology. ACM SIGMOD Record, 1997,26(1):65−74.

[8] Torlone R, Panella I. Design and development of a tool for integrating heterogeneous data warehouses. In: Tjoa AM, Trujillo J, eds. Proc. of the 7th Int'l Conf. on Data Warehousing and Knowledge Discovery (DaWaK 2005). LNCS 3589, Springer-Verlag, 2005.

[9] Tseng FSC, Chen CW. Integrating heterogeneous data warehouses using XML technologies. Journal of Information Science, 2005, 31(3):209-229.

[10] Raman V, Narang I, Crone C, Haas L, Malaika S, Mukai T, Wolfson D, Baru C. Data access and management services on grid. 2002. http://www.ggf.org/Meetings/ggf5/pdf/dais/document2.pdf

[11] Haas LM, Lin ET, Roth MA. Data integration through database federation. IBM Systems Journal, 2002,41(4):578-596.

[12] Deshpande A, Hellerstein JM. Decoupled query optimization for federated database systems. In: Ngu AHH, ed. Proc. of the 18th Int'l Conf. on Data Engineering (ICDE 2002). San Jose: IEEE Computer Society, 2002. 716-732.

[13] Mattos NM. Integrating information for on demand computing. In: Freytag JC, Lockemann PC, Abiteboul S, Carey MJ, Selinger PG, Heuer A, eds. Proc. of the 29th Int'l Conf. on Very Large Data Bases (VLDB). Berlin: Morgan Kaufmann Publishers, 2003. 8-14.

[14] Ambler S. The design of a robust persistence layer for relational databases. Ambysoft Inc., 2000. http://www.ambysoft.com/downloads/persistenceLayer.pdf

[15] Ambler S. Mapping objects to relational databases: O/R mapping in detail. Ambysoft Inc., 2003. http://www.agiledata.org/essays/mappingObjects.html

[16] SUN Microsystems. Enterprise JavaBeans specification, Version 2.0. 2001. http://jcp.org/aboutJava/communityprocess/final/jsr019/index.html

[17] SUN Microsystems. Enterprise JavaBeans specification, Version 3.0. 2006. http://jcp.org/aboutJava/communityprocess/final/jsr220/index.html

[18] SUN Microsystems. Java persistence API. 2006. http://javashoplm.sun.com/ECom/docs/Welcome.jsp?StoreId=22&PartDetailId=ejb-3_0-fr-eval-oth-JSpec&SiteId=JCP&TransactionId=noreg

[19] Fowler M. An acronym for: Plain old Java object. 2000. http://www.martinfowler.com/bliki/POJO.html

[20] Hibernate reference documentation, Version 3.0.5. 2005. http://www.hibernate.org/

[21] Java data objects specification, Version 1.0.1. 2003. http://jcp.org/aboutJava/communityprocess/final/jsr012/index2.html

[22] Java data objects specification, Version 2.0. 2006. http://jcp.org/aboutJava/communityprocess/final/jsr243/index.html

[23] Forge S. Java persistence aspect, Version 0.6. 2004. http://sourceforge.net/projects/jpa

附中文参考文献:

[1] 吕建,马晓星,陶先平,徐锋,胡昊.网构软件的研究与进展.中国科学(E辑),2006,36(10):1037-1080.

[2] 梅宏,黄罡,赵海燕,焦文品.一种以软件体系结构为中心的网构软件开发方法.中国科学(E辑),2006,49(6):702-730.

[3] 何兆熊.新编语用学概要.上海:上海外语教育出版社,2000.

滕腾(1977-),男,湖南东安人,博士,主要研究领域为分布计算与中间件技术,数据持久化技术.

黄罡(1975-),男,博士,副教授,CCF 会员,主要研究领域为分布计算与中间件技术,软件工程,软件体系结构.

陈兴润(1985-),男,硕士生,主要研究领域为中间件技术,数据持久化技术.

梅宏(1963-),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件工程,软件复用,软件构件技术,分布对象技术.