计算机科学2002Vol. 29№. 6

持久性程序设计语言及其现状*)

A Survey of Persistent Programming Languages

卢 虹 徐宝文

(东南大学计算机科学与工程系 南京210096) (武汉大学软件工程国家重点实验室 武汉430072)

Abstract Persistent transaction system is an important branch of computer application, traditional programming language and database system are not suitable to construct persistent systems. In persistent programming language, data movement is inexplicit, persistent language has the advantages of both traditional language and database system, it is a powerful tool to construct persistent application systems. In this paper, we first address the problems that persistent language need to solve, then we present the design principles and some persistent models, at last, we discuss persistent extensions to some typical languages.

Keywords Persistent programming language, Object oriented database, Orthogonal persistence, Reachability

1. 引言

许多计算机系统如 MIS 系统、CAD/CAM 系统、CASE 系统、OA 系统、地理信息系统、医疗保健系统都遵循一个共同的模式，就是围绕持久性的数据进行计算(处理)，这类应用称为持久应用(persistent application)。在这类系统中，被处理数据分为瞬态数据和持久性数据。传统程序设计语言一般只提供了定义和操纵瞬态数据的设施，却不支持对持久性的抽象。传统的数据库系统可以定义和操纵持久数据，但却存在着计算不完备、数据建模能力差等问题。因此，在开发持久应用时，人们往往不得不同时使用数据库系统和传统程序设计语言：用数据库系统对持久性数据进行定义和存储，用传统语言进行其它辅助处理。但混合使用传统语言和数据库系统并不是一种理想的解决方案。首先，由于两种系统的数据是分离的，且具有不同的数据模型，它们之间的数据流动和转换就必须由程序员来完成；其次，程序员必须同时熟悉程序设计语言和数据库系统，这加重了程序员的开发负担。为此，人们提出了持久性语言的概念。持久性语言是一种不显式表达数据移动(在持久性存储设备和瞬态存储设备间移动)的语言。持久性语言吸取了传统语言和传统数据库系统二者的优点，是一种开发持久应用的有力工具。

本文首先对程序设计语言和数据库系统在开发持久性应用时的优劣进行比较，指出持久性语言需要解决的问题；然后详细讨论持久性语言的设计目标和几种主要的持久模型，最后讨论几种典型语言的持久性扩展。

2. 问题分析

数据库系统通常都配备一种说明式(declarative)查询语言。SQL 就是一种典型的数据库查询语言。由于 SQL 的使用非常广泛，下面我们就用 SQL 为例，从数据定义和数据计算两个方面对 SQL 和传统语言进行比较。

在数据定义方面，传统语言中的变量可以分为全局、自动和堆变量，它们的共同特点是都随程序运行的结束而消亡。传统语言缺乏一种方便可靠的持久变量，这种变量在数据库系统或文件系统中分配存储空间，其生存周期可以跨越程序的多次运行。传统程序设计语言的优势在于具有丰富的数据类型，不仅如此，程序员还可以使用结构(struct)、联合(union)等设施来描述预定义的数据类型所不能直接描述的对象。更重要的是，随着语言的发展，Ada、Modula-2、C++、Java、C#等语言进一步具备了定义抽象数据类型的能力，以及诸如继承、多态等面向对象的特征，这些语言的类型系统具有更良好的可扩展性和更强大的建模能力。相比之下，我们可以容易地用 SQL 语言定义持久数据，但是它却只支持一些基本简单数据类型，如 INTEGER、SMALLINT、DECIMAL、FLOAT、CHAR、VARCHAR，加上相当于集合类型的 TABLE，SQL 语言支持的数据类型也寥寥无几，所以，传统数据库系统的数据建模能力是相当有限的。

在数据计算方面，如所知，SQL 是一种说明式查询语言，它为大量的数据检索而设计和优化。这种说明式查询语言，其功能如果用面向过程的传统语言来实现，是非常不自然的。尽管说明式的 SQL 语言易于使用(尤其适于数据查询)，但它却是计算不完备的，它面向的问题域比传统语言的小得多，所以 SQL 不能替代传统语言。综上所述，数据库系统适合定义和存储持久数据，传统程序设计语言适合于数据计算和建模。

可以看出，在开发持久应用软件时，数据库系统和程序设计语言系统各有优势，综合运用二者似乎就能得到一个完美的解决方案。在这种解决方案中，数据库系统主要用来存储数据，传统程序设计语言则用来完成对这些数据的计算。然而，这种方法会产生两个严重的问题：通信障碍和类型失配。

首先，程序员为了操作数据库中的数据，必须经过如下繁琐的过程：①从数据库中读取数据；②对数据进行计算；③将数据送回数据库。为了让数据库和程序设计语言系统能交换数据，还必须建立额外的通信机制。途径之一是为传统语言配备一套庞大的程序库(函数库或类库)，程序员可以调用它们来操纵数据库，然而程序员将为学习这些程序库的使用方法付出艰苦的劳动。不仅如此，这些程序库的形式和用法，通常会随着所使用的数据库系统和程序设计语言的变化而变化，程序员的负担因此愈发沉重。当然，诸如 ODBC、JDBC 之类的

*)本研究得到国家自然科学基金(60073012)、江苏省自然科学基金、江苏省三三三人才基金、高等学校重点实验室访问学者基金、南京大学软件新技术国家重点实验室基金资助。卢 虹 博士生，主要研究领域为程序设计语言、程序分析、软件工程。徐宝文 教授，博导，主要从事程序设计语言、软件工程、并行程序设计等方面的教学与科研工作。

规范能抽象出各数据库系统的共同之处，为程序员提供一个通用的编程接口，一定程度上减轻了程序员的负担，但这仍不是一个彻底的解决方法。另一种解决途径是允许在程序中嵌入 SQL 语句，这种做法破坏了语言的单纯性，同时还需要为语言提供特殊的编译程序（或预编译程序），所以也不是理想的解决方法。以上论及的就是所谓的通讯障碍问题。

其次，由于数据库系统和程序设计语言都有各自的类型系统，从而我们不得不对同一个应用进行两次数据建模。程序员必须将程序设计语言中的数据结构（如树、集合、图等）映射为数据库系统中低级的、原始的数据类型（如整型、字符串型、浮点型等）或是相反，据统计，这些用于转换的代码占持久应用所有代码的30%左右[9]。这就是所谓的类型失配问题。

3. 设计目标

由于持久应用的传统开发方法具有诸多缺陷，人们一直在寻求更好的解决途径，其中之一就是持久性程序设计语言。M. P. Atkinson 等人是持久性语言研究的先驱者，他们研制了第一个持久语言 PS-Algol[1]。此后，人们又研制出了 Amber、Galileo、Napier88和 OPAL 等持久性语言。相比于这些学究气的语言，1996年问世的持久性语言 PJama 更加注重实用，它是一种基于 Java 的、面向对象的、类型安全的（type safe）、正交的持久性语言，关于这种语言的研究和开发仍然在进行之中。商业化的持久性语言往往和面向对象的数据库系统（OODBMS）捆绑在一起。比较有名的商业化 OODBMS 有 Versant、ObjectStore、Objectivity、O2、GemStone。这些系统往往绑定若干个编程语言，如 C++、Smalltalk、Java、Ada 等，并对这些语言进行持久性扩展。

上述各个语言对持久性的支持各有特点，那么，理想的持久性语言应该具有什么特征呢？理想的持久性语言应该满足所谓的正交性原则[2]，即：

·*形式无关原则* 程序的形式应和数据的生存期无关，即操纵持久性数据的代码和操纵瞬态数据的代码应是不可区分的。这一原则保证了代码的可重用性，它还隐含着：持久语言应能不显式地表达数据移动，读写数据库或文件系统对于程序员来说，应该是透明的。

·*类型无关原则* 任何类型的对象都可以被声明为持久的，一个对象是否是持久的与其类型无关。类型无关性之所以重要是因为：现代面向对象语言大多数都具有定义抽象数据类型的能力，用户可以根据自己的需要，灵活地扩展语言的类型系统，所以，除去语言预定义的少数几种原始数据类型外，用户定义的数据类型也应该可以是持久的。这一原则保证了类型的可重用性。

·*标识无关原则* 为了标识哪些数据是持久的、哪些数据是瞬态的，需要提供一定的机制，这种机制应该与系统的问题域无关。比如，我们不应该从对象的名称或类型来推断对象的持久性。

4. 持久性模型

由上面的讨论可以看出，传统的程序设计语言必须经过一定的扩展才能支持持久对象。持久性语言的一个核心问题是：如何确定一个特定的对象是否持久。通常，对象的属性可以分为静态属性（编译时属性）和动态属性（运行时属性）。在大多数编译型语言中，对象的静态属性主要指对象的名字、类型、存储类别等，对象的动态属性主要指对象的值。持久对象往往具有和瞬态对象不同的属性，比如特殊的名字、特殊的类型、特殊的值等等。目前的持久模型主要有三种：基于继承的持久模型、基于分配的持久模型和基于可达性的持久模型，它们分别从对象的类型、对象的存储类、对象和其它对象的关系入手，实现对持久性的支持。

4.1 基于继承的持久性模型

基于继承的持久性模型（persistence by inheritance）主要针对面向对象语言而言，在这种持久性模型中，类的持久性是从一个预定义的持久类继承而来的，持久类的对象既可以是持久对象也可以是瞬态对象，程序员必须显式地在持久性区域和瞬态区域间移动数据[8]，形式无关原则因而没有得到满足。这种模型最大的缺点是它难以保证数据的完整性：假定一个持久类的对象包含一个瞬态对象的引用，当此瞬态对象消亡时，它的引用却由于存在于持久对象中而得以保存下来，即，程序员可以用这个引用访问已经不存在的数据，这往往会导致严重的后果。这个问题和传统语言中的指针空挂问题是同质的。由于上述原因，基于继承的持久模型不是一种成功的持久模型。

4.2 基于分配的持久性模型

基于分配的持久性模型（persistence by allocation）也叫做基于创建的持久模型（persistence by creation）。这种模型引入了新的存储类：persistent 存储类。传统程序设计语言具备的存储类一般有局部类、全局类与动态类等几种：对于局部类，对应的变量为栈变量，在程序的栈上分配存储空间；对于全局类，对应的变量为静态变量，在程序的静态数据区分配存储空间；对于动态类，对应的变量为堆变量，在程序的堆空间中分配存储空间。而在持久语言中，变量还可以在持久性区域中分配存储空间。当为这些变量分配好存储空间即建立了这种变量之后，它们就具有了持久性。基于分配的持久语言往往引入 persistent 关键字用以描述对象的存储类，或是引入如 pnew 这样的运算符在持久区域中动态地创建对象[4]。

基于分配的持久模型有着与基于继承的持久模型类似的数据完整性问题。事实上，这两种模型都是静态模型，它们对持久性的支持都着眼于对象的静态属性（如对象的类型、对象的存储类），在静态模型中，对象的持久性是在编译时决定的，所以数据完整性问题是不可避免的。

4.3 基于可达性的持久性模型

基于可达性的持久模型（persistence by reachability）是一种动态持久模型。在这种模型中，持久对象要么是持久根（persistent root），要么是被其它持久对象引用的对象。可以想象，程序中的对象通过相互间的引用形成一张对象图，当这张图中的一个对象接收到一条特殊的消息时，此对象成为持久根。这条消息会在对象图中传播，结果是任何可从持久根到达的对象都成为持久对象。当一个持久对象从任何持久根都不可达时，它将被一个磁盘垃圾回收系统回收。基于可达性的模型成功地解决了数据完整性问题，PS-Algol、Napier88、PJama 都建立在这种模型之上。基于可达性的持久性也称为隐式持久性，这是因为程序员无需显式地标识对象的持久性，当然，这里要除去那些作为持久根的对象[5]。

事实上，只有基于可达性的持久模型才满足正交持久性的三条原则。然而在进行持久性改造时，并非所有语言都适合可达性模型。比如像 C 和 C++这类追求效率的语言，程序员可干涉堆空间的管理，堆对象由程序员负责（通过 delete 运算符）显式地销毁，而可达性模型却要求自动垃圾回收，因此象 C、C++这样的语言就不适合用可达性模型，本文的第6节将讨论一种基于分配的、C++的持久性扩展。Java 是一种新兴的语言，它在许多方面作了相当的创新，在持久性方面，它采用一种基于继承和基于可达性的混合模型，第7节将讨论 Java 持

久模型的优劣。而诸如 Ada 这样大型的语言，由于它的类型等价规则、与众不同的程序包、为支持并发而引入的任务等等特性，使得它的持久性扩展比较困难，第8节将详细讨论这些问题。下面我们将首先讨论第一个持久性语言 PS-Algol。

5. PS-Algol

PS-Algol 是第一个持久性程序设计语言。在 PS-Algol 诞生之前，有两个研究小组曾经尝试扩展 Algol-68和 Pascal，这两项工作遇到的困难促使 PS-Algol 的设计者们选择了 S-Algol(St Andrew 大学的一种教学语言)作为持久性改造的目标语言。在 S-Algol 中，对象的生存期和作用域没有明显的联系，对象的可用性取决于此对象的有效引用是否存在。S-Algol 有一个不错的垃圾收集器，负责无用对象的回收。S-Algol 的这些特性使得持久性改造相对比较容易。事实上，PS-Algol 的设计者没有对 S-Algol 的语法和编译器作任何修改，而仅仅为它扩展一套标准的库函数，就达到了目的。这套库函数如下所示：

```
procedure open. database ( string database. name, mode, user, pass-
    word -> pntr )
procedure get. root( pntr database -> pntr )
procedure set. root( pntr old. root, new. root )
procedure commit
procedure abandon
procedure close. database( pntr root. value )
```

其中，open. database 用于打开持久对象库，返回对象库的指针，第一次调用 open. database 的同时会开始一个新的事务，在 commit 或 abandon 之后调用 open. database 也会开始一个新事务。get. root 用于获取持久根，持久根是可达性闭包的入口点，通过它可以访问其它持久数据。set. root 用于建立一个持久根，但只有经过 commit 调用后，建立动作才真正完成。commit 用于向所有打开的对象库提交当前事务，从持久根可达的所有对象都会被写回持久存储设备中，abandon 用于撤消所有打开对象库的当前事务，将对象库的状态恢复为上一个 open. database 调用或 commit 调用后的状态。最后一个函数 close. database 用于关闭一个已打开的对象库。当通过一个指针引用持久对象时，如果该对象不在内存中，那么 S-Algol 的运行时系统会根据该对象的持久标识(PID)在持久对象库中找到它，并调入内存、存放在程序的堆空间中。

PS-Algol 的持久设施看似原始，但我们注意到程序员已经无须显式地移动数据，基于可达性的实现使 PS-Algol 避免了数据完整性问题，对象的持久性也与类型无关，因此，PS-Algol 是一种正交的持久性语言。有趣的是，实验数据表明，用 PS-Algol 来开发 CAD 软件时，总的代码量约缩小了三分之一，开发周期也以类似的比例缩短，可以预见，代码的维护代价也会相应地缩小。可见，PS-Algol 是持久性语言第一次成功的尝试。

6. C++的持久性扩展

O++是 ODE(Object Development Environment)系统的程序设计语言，它对传统的 C++语言进行了持久性扩展。O++采用基于分配的持久模型，它的设计力图遵循以下的原则[4]：

·持久性应该是对象的属性，而不是类型的属性；

·允许用户在瞬态存储区域和持久区域中分配任何类型的对象；

·持久对象和瞬态对象的访问方法应该没有区别；

·正如可以将对象从栈上移到堆上一样，我们也可以以同样的方式将对象从持久存储区移到瞬态存储区或是相反。

遵循上述设计原则，O++引入两个新的分配运算符 pnew 和 pdelete。pnew 与 pdelete 运算的功能与传统 C++中 new、delete 类似，它们能在持久存储区中分配对象。为了能够访问这些对象，O++又引入了持久指针的概念。持久指针是由关键字 persistent 修饰的指针，如以下程序段所示。其中，第1行的语义是：指针 pi 指向的对象是一个持久对象。persistent 不是一个存储类修饰符，而是一个类型修饰符。指针 pi 如普通指针一样，在栈上分配存储空间，而不是在持久存储区域中。持久对象可以和瞬态对象相互赋值，如第3行与第4行。

```
1  persistent int * pi = pnew int;
2  int i;
3  *pi = i;
4  i = *pi;
```

另外，由于持久指针和普通指针的具有不同的类型，我们无法以统一的方式处理瞬态对象和持久对象。例如，我们无法写出这样的子程序，它能够同时以持久指针和普通指针作为参数——类型检查系统会阻止我们这么做。为此，O++又引入了偶指针(dual pointer)的概念。偶指针既可以指向持久对象也可以指向瞬态对象，至于它到底指向什么则被推迟到运行时决定。

可以看出，O++的持久性满足类型无关原则。但显然，它不满足形式无关原则，最为明显的是我们必须以不同的方式建立和撤消持久对象和瞬态对象。尽管如此，O++的设计者们还是力图在一定程度上支持形式无关性，比如：持久对象和瞬态对象可以相互赋值，而且其形式与瞬态对象的相互赋值一样，还有，通过使用偶指针，我们就能以相同的方式操纵瞬态对象和持久对象。O++作为一种对 C++的扩展，为了效率的缘故，没有进行可达性分析，保证数据的完整性是程序员的任务，O++的编译系统和运行时系统对此不负责任，因此，O++也不满足标识无关原则。O++语言还有其它一些特性，比如对象查询设施、约束子(constraint)、触发子(trigger)等，不一一赘述。

7. Java 的持久性

从第1.1版开始，Java 就支持一种称之为序列化(serialization)的轻度持久性[11]。序列化的技术早已有之，在 Modula3中它称之为 pickling。Java 规定，只要有一个对象继承了 Serializable 接口，就可以用几条简单的方法调用，将它转换成一个字节序列，这个过程称之为序列化。序列化的字节流便于在持久性介质上存储，也便于在网络上传输。值得一提的是，秉承 Java 语言一贯的传统，序列化是与平台无关的，也就是说我们可以在 Windows 机器上建立一个对象，将之序列化，然后通过网络传输到 Unix 机器上，在那里重新装配这个对象，程序员无须关心数据在不同机器上如何表示，也不必关心字节的顺序。从字节流中恢复的是一个全新的对象，它和原来的对象有完全一致的状态和类型，仅仅是对象标识(object identity)不同。Java 不仅仅序列化单个的对象，而且能追踪对象内包含所有句柄(对象标识在 Java 中的叫法)并序列化那些对象，接着又能对每个对象内包含的句柄进行追踪，依此类推，最终的结果是：从一个对象可以到达的所有对象都会被序列化，Java 的这种持久策略可以避免数据完整性问题[10]。可以看出，Java 实际上采用的是一种基于继承和基于可达性的混合模型。

序列化非常容易使用，Java 的运行时系统会为程序员照看一切繁琐的细节，它非常适用于在不同进程和机器间传递

对象，因此序列化也是 Java RMI 的基础。另外，序列化是一个原子操作，所以可以用来保存数据的快照(snapshot)。尽管 Java 的序列化有这些优点，但它依然不是一种优良的持久性机制。首先，它不符合形式无关原则，程序员必须在程序中显式地序列化一个对象或是相反。其次，由于序列化仅仅保存对象的状态和类型，而丢失了对象的标识信息，如果两个对象具有公共的子结构，那么它们序列化之后将不再共享公共的子结构而是分别持有一个拷贝。第三，由于 Java 的反序列化是原子操作，尽管有时程序员只想从序列化的对象群中恢复一小部分对象，他也不得不等待 Java 运行时系统将所有的对象反序列化，由于牵涉到磁盘读写，序列化和反序列化往往会耗费大量的时间，这就是所谓的 big inhale 问题。

由于序列化机制的缺陷，在 Glasgow 进行的 PJama 项目[3]试图对 Java 做比较彻底的持久性改造。PJama 基于可达性模型，对象的持久性由运行时系统来决定(通过可达性分析)，持久存储区和瞬态存储区之间的数据移动由系统自动完成。在 PJama 中，对象的类型信息和对象的值一起存于持久对象库中，这使得系统可以对持久对象进行类型检查，所以，PJama 保持了 Java 的类型安全性。PJama 中，持久对象仅在需要的时候才被调入，这种渐增式的调度策略避免了 Java 的 big inhale 问题。PJama 一定程度上克服了 Java 的缺点，其研究和开发还在进展当中。

8. Ada 的持久性扩展

由美国国防部主持研制的 Ada 语言适于用来开发大型、可靠、易维护的软件[13]，是一种优秀的通用程序设计语言，适合于用来解决各种应用领域(如嵌入式、实时)中的问题。然而，由于在设计的时候没有考虑持久性，Ada 并不特别适用于开发持久应用。为此人们提出了一些改进建议，也开发了一些持久性 Ada 原型，这些工作包括 Green 的扩展建议[12]和 PGRAPHITE 系统[7]等。Ada 的许多特性都使我们难以对它进行持久性扩展，Crawley 指出，如果不对核心语言作适当的修改，Ada 是无法支持正交持久性的[6]。

首先，在 Ada 语言中，我们需要考虑更多程序实体(如程序包、任务)的持久性。Napier88和 PS-Algol 等语言不支持封装的概念，只有当程序地址空间中的对象能够从持久根访问到的时候，这些对象才具有持久性。Ada 则可以通过程序包来封装变量和常量，也正是它们构成了程序包的状态。这些变量和常量可能是持久的，这就要求程序包也可以具有持久性，文[6]举了一个很好的例子来证明这一点。此外，任务是一种 Ada 的基本对象，也是 Ada 类型体系的一部分，很自然地，任务应该可以持久化。事实上，如果 Ada 不支持持久任务，那么一个具有任务的程序包就无法持久化了。在持久化一个任务的时候，我们需要保存任务的所有运行上下文，包括它的栈，程序指针等等。任务持久化的一个主要问题是，哪些任务需要持久化?此时，可达性原则已经不适用了，因为一个从持久根不可达的任务也完全有可能修改持久对象。

其次，为了支持持久程序包，with 语句的语法和语义也必须修改。在 Ada83和 Ada95中，with 语句的语义是将程序包的名称和程序包的体静态地绑定在一起，而持久 Ada 必须支持动态的 with 语句，这是因为在持久程序包中的变量或函数可以使用之前，必须将程序包从持久区域中装入程序的地址空间，而这一动作必须在运行时完成。经过持久性扩展的 with 语句，其用法如下所示：

```
with dynamic My_Pack from"/home/luhong/pack"。
```

第三，Ada 的类型等价性规则必须修改。在许多语言中，类型是按结构等价的，Ada 则不同，它采用一种更强的类型等价规则：按名等价。按名等价规则的优点是，它可以区分物理表示相同但逻辑上不同的类型。然而，按名等价的规则却不适用于持久语境。假设有一个 Ada 程序定义了类型 T，在两台不同的机器上编译这个程序会产生两个类型标识 idT1和 idT2，当我们试图将一台机器上的、类型为 T 的对象名字和另一台机器上持久对象库中的对象绑定时，就会发生类型不匹配错误。因此，在绑定持久对象时，Ada 必须使用按结构等价的规则。

综上所述，Ada 的许多特性都使得对它进行持久性扩展非常困难，因此，迄今为止，还没有出现支持正交持久性的 Ada 实现。

结束语 持久性系统是计算机应用的一个非常庞大的分支，持久性语言是开发持久应用强有力的工具。本文介绍了持久性语言的简短历史，讨论了持久语言的设计目标和几种典型语言的持久性扩展。持久性语言领域的研究方兴未艾，新概念、新思想不断提出，如反演式程序设计(reflective programming)，超链程序设计(hyper programming)等。反演式程序设计的基本思想如下：程序可以由程序本身来编写。在持久性系统中，持久数据往往具有未知的数据类型，反演式程序可以生成代码来安全地处理这些类型。超链技术允许程序的源代码中包含持久对象(位于持久对象库中)的链接。用超链程序设计技术写出的程序更加简洁、稳定可靠和易于理解。除了持久性语言，开发持久应用还需要得到持久操作系统、面向对象的数据库以及其它一些运行时系统的支持。目前，持久应用的主要开发手段依然是传统程序设计语言和数据库系统，但集成持久操作系统、面向对象数据库和持久性语言的全新系统必然会取而代之，成为持久应用领域主流发展方向。

参 考 文 献

1 Atkinson M P,et al. PS-algol: a language for persistent programming. 10th Australian Computer Conference, Melbourne, Sept. 1983. 70～79

2 Atkinson M P, et al. Orthogonal persistent object systems. VLDB Journal,1995,4(3)

3 Atkinson M P,el al. An orthogonally persistent Java. ACM SIGMOD Record,1996,25(4):68～75

4 Agrawal R,et al. ODE (object database and environment): The language and the data model. In: Proc. 1989 ACM-SIGMOD Conf. on Management of Data,Portland,Oregon,June 1989

5 Printezis T,et al. The Design of a New Persistent Object Store for PJama. In:Proc. of the Second Intl. Workshop on Persistence and Java,Sun Microsystems Technical Report TR-97-63,1997

6 Crawley S C,et al. Orthogonal persistence and Ada. In: Proc. TRI-Ada'94,Baltimore MD,ACM,Nov. 1994. 298～308

7 Wileden J C,et al. PGRAPHITE: An experiment in persistent typed object management. In:Proc. of SIGSOFT '88:Third Symposium of Software Development Environments,Sep. 1988. 130～142

8 Models and languages of Object/Oriented Databases, Addison-Wesley,Harlow,England,ISBN:0-201-62431-1

9 Joseph,et al. Object-Oriented Database:Design and Implementation. Proceedings of the IEEE,1991,79(1)

10 Eckel B . Thinking in Java. President,MindView Inc. ,Prentice Hall PTR,1998

11 Riggs R, et al. A Pickling state in the Java system. In:Proc. of the second Intl. conf. on object-oriented technologies (COOT'96),Toronto,Canada,June 1996

12 Green G. Access values pointing to any type. ACM Ada Letters, 1990,10:101～109

13 徐宝文. Ada 95语言评述. 计算机研究与发展,1997,34(1):53～57