第10章　数据库集成

CORBA和数据库技术可以集成来支持持久性 CORBA对象。ORB提供了一些重要特征，如分布性、互操作性和技术集成，也提供了面向对象的好处，如抽象和封装。另一方面，数据库提供了数据持久性和数据完整性、事务、并发数据访问和分布性。下面讨论如何把这两种技术结合起来，平衡但并不减少各自的优点。总体讨论系统结构之后，本章还将探讨当前的 CORBA标准，以及集成的总体方面。

10.1　系统体系结构

在讨论由CORBA和数据库集成产生的系统体系结构之前，先分别进行简要介绍。 CORBA消息范例相对简单： CORBA被设计用来使客户机能向对象发送请求并可能接受回应。类似地，许多数据库提供了API，使数据库客户机能向数据库服务器发出查询请求。数据库服务器将执行查询，并送回某种响应。从集成的观点来看，有两种明显的可能性：或者数据库本身通过使用CORBA接口输出服务，提供和 CORBA的紧密集成；或者集成发生在应用层次，由于缺乏可用的商业集成产品迫使我们在应用层进行集成[⊖]，这种应用层集成自然导致了三层结构的系统。

10.1.1　三层结构

三层结构的基本元素是客户机、应用程序服务器和数据库服务器。客户机通过应用程序服务器输出的 IDL接口来访问应用程序服务器。应用程序服务器扮演了双重角色，一方面是CORBA服务器，另一方面作为数据库客户机访问数据库服务器以获得持久状态信息。这个体系的三层结构如图10-1所示，通常定义为

客户机	图形用户界面
应用程序服务器	业务逻辑
数据库服务器	数据管理

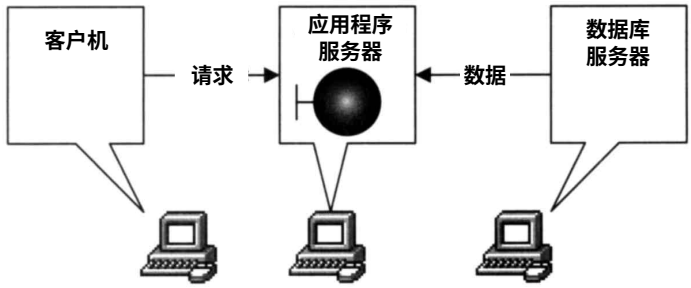

图10-1　三层CORBA结构

⊖ 最近公布的Oracle8i产品声称支持Enterprise Java Beans 和IIOP通信。这是有意义和令人鼓舞的发展，我们期待着这个产品的发行使用。

10.1.2 N层结构

三层结构仅仅是实际的基于组件计算的基础，后者通常称为 N层计算。用户接口层只是N层计算的一部分，如图 10-2所示。中间层本身通常是一组相互作用的组件，其中一个组件可以通过定义好的CORBA接口访问其他组件的服务。每个组件都不清楚其他组件内部的实现细节、数据库访问机制或者持久数据框架。封装组件内部复杂性和只暴露定义好的、基于 IDL的组件接口，通常就是集成CORBA和数据库技术的主要目的。下面，我们会知道这常常不是简单、直接的工作，但得到的好处会超过为组件化所付出的额外代价。

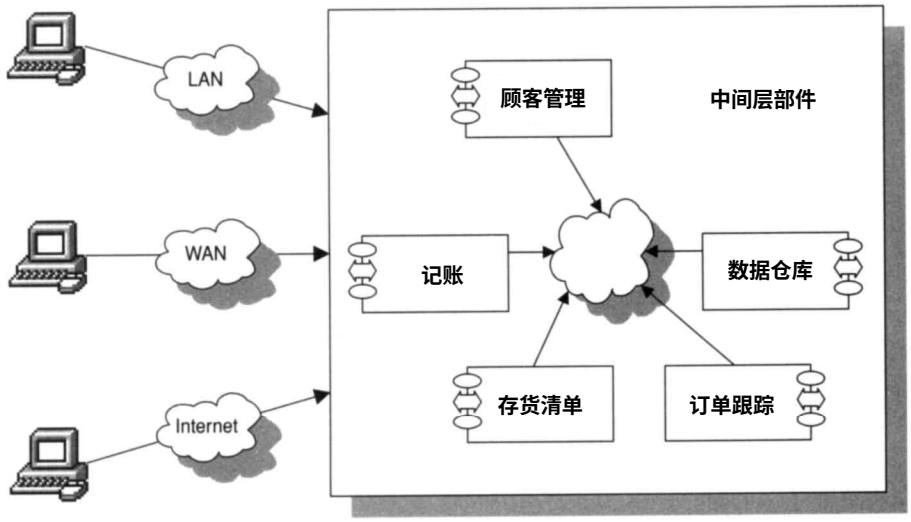

图10-2 N层结构

10.2 相关的OMG工作

有很多CORBA规范直接或间接地与 CORBA数据库集成领域相关。例如，对象事务服务和并发控制服务，还有查询服务、关系服务和生命周期服务等，以及对象值服务和可移植的对象适配器规范。但是这些服务都没有直接陈述 CORBA的数据库集成问题。第一个陈述这个问题的服务是CORBA持久对象服务（POS）。不幸的是，POS有一些明显缺点，持久状态的管理直接暴露给CORBA客户机。POS缺乏封装，有复杂的内部结构，并且不适于集成已有的储存设备，今天并没有 POS的商用实现。关于 POS实现问题的有趣讨论可以在 Plasil、Kleindienst和Tuma(1996)[⊖]中找到。因此，OMG决定彻底重写持久对象服务。这将产生一个全新的规范，甚至不同的名字：CORBA持久状态服务（PSS）。

PSS的规范还在发展，但是很明显，新的 PSS将是更加面向服务器的框架，不使对象持久机制暴露给CORBA客户机。其主要特征有：

⊖ Plasil,F.,Kleindienst, J., 和Tuma, P.,“实现CORBA持久对象服务的知识”，OOPSLA ’96学报,ACM, 1996年10月。

- 持久值 基于对象值规范(参见第5章“性能要求”中5.2.2节“按值传递对象”)，PSS将提供对储存持久值的支持。其API独立于数据储存，并基于对象值规范的语言映射。
- POA集成 PSS将和CORBA可移植对象适配器紧密集成(参见第2章“CORBA回顾”)。
- 事务管理 PSS支持分布式和非分布式的事务。对于分布式事务，PSS提供与CORBA对象事务服务的互操作机制。

因为PSS规范还没有最终确定，所以持久状态服务的商业实现尚需时日。既然今天对CORBA的集成有很大需求，就必须探讨现在可用的解决方案。本章剩余部分讨论CORBA或数据库集成，而不是基于将出现的PSS规范。

10.3 集成的各方面

正如我们所见的，已经有了处理持久性相关问题的OMG标准，人们也在为进一步标准化而努力。不幸的是，这些标准缺少可用的商用实现。例如，今天的常用数据库服务器并不通过CORBA接口直接输出服务。这意味着通常必须在应用层处理CORBA或数据库集成。下面考虑在企业系统中集成CORBA和数据库技术的关键方面。

IDL设计

IDL设计通常是语义的丰富性和性能间的权衡，因此，在设计IDL使CORBA客户机能访问持久数据时，要仔细考虑。一个重要问题是，我们实际上希望哪些持久对象能作为持久CORBA对象而直接被访问。四字节实体作为对象储存在ODBMS中，它通常不适合成为第一类CORBA成员。在对象/关系映射的讨论中，我们标识了由映射产生的对象的特点，它们通常是非常细粒度的数据对象。如果关系框架高度规范化，太直接的映射将很成问题。另一方面，由于CORBA体系的本性：CORBA是一个消息传递的结构，它并没有状态的概念；所以CORBA对象通常是以行为为中心的。例如，IDL属性隐含但是不明确需要其实现具有相当于IDL属性的成员变量。下面讨论通过IDL访问持久数据的三类不同方法：直接数据库访问、前端和CORBA业务对象。多数系统中使用了前端和业务对象的结合方法。

1. 直接数据库访问

为能直接通过IDL访问持久数据，我们通常使用某种查询处理接口，把单个查询字符串作为参数并以某种串或结构的序列形式返回结果。例如，可用以下IDL：

```
struct NameValue {
    string m_name;
    string m_value;
}
typedef sequence<NameValue> NameValueSeq;

interface QueryProcessor {
    NameValueSeq processQuery (in string queryExpression);
};
```

这个例子中，查询结果作为名/值对的序列返回。另一种办法返回any类型的序列，每个any拥有结果集的一个元素，这使我们能返回结果而不用把它们转换成串格式。但是，提供直接数

据库访问并没有充分利用CORBA的能力；结果是数据结构，能使用标准数据库访问机制（如前面讨论的嵌入SQL或SQL封装工具）来获得。CORBA的主要优点之一是它提供了以行为为中心的对象。下面，通过使用前端和业务对象，探讨如何用IDL来取得折衷。

2. 前端

前端的思想是提供CORBA对象作为持久数据的前端，以便能通过前端来控制数据。IDL接口的前端类型类似于RPC风格的接口，是十分过程化的。例如，使我们能控制证券对象的一个前端可能如下：

```
typedef string ID;

interface PortfolioFrontend {
    // Portfolio lifecycle events:
    ID createPortfolio (in string Name);
    void deletePortfolio (in ID portfolioID);

    // Portfolio manipulation:
    Money getCurrentValue (in ID portfolioID);
    ...
};
```

通常，前端提供过程风格的操作，这些操作允许客户机通过传递目标对象ID来控制数据或者对象。前端方法有明显好处。例如，在特定环境下它能提供更好的性能。回顾第5章“性能要求”中关于次级OID的讨论。本质上，前端可以实现为无状态伺服对象。因此，从内存管理的观点来看，实现前端十分容易。但是，这种十分过程化的方法的缺点是它缺乏语义表达性和清晰的面向对象设计所具有的其他好处。

10.4 CORBA业务对象

对“什么是业务对象”的回答几乎像对“什么是CORBA对象”的回答一样困难——这有点抽象。我们用业务对象的概念来表示在处理的业务领域中具有某种意义的对象。CORBA业务对象能映射成单个持久对象或关系数据库中的一行。但是，也可能实际映射成更复杂的持久数据结构，尤其当处理规范化数据框架时。

如第5章“性能要求”中所讨论，决定哪些对象成为第一类CORBA成员并不容易。类似地，在IDL层次建模时，决定用哪些业务对象来提供对持久数据结构的访问也不容易。下面例子说明了如何把证券对象建模为合适的CORBA对象：

```
interface Portfolio
{
    void delete ();
    Money getCurrentValue ();
    ...
};
```

这个接口并不令人惊奇。注意，现在能通过直接访问来控制证券对象，而不用向前端对象的操作传递标识。通常业务对象需要某种管理对象，帮助控制对象创建和业务对象集合上的操作，例如查询操作。对于证券对象，我们需要一个有点像前端的证券管理器：

```
Interface PortfolioManager
{
 Portfolio createPortfolio (in string Name);
};
```

10.4.1 阻抗失配

不仅在持久数据框架和数据库客户机编程语言之间有阻抗，而且在持久对象和 CORBA对象之间也有阻抗。在IDL层，通常可以看到接口层次相对于基类层次有所偏移。这就是所谓的聚集层次中的向上偏移 —— 很多持久对象图形成某种聚集层次，这种层次通过实现此图的类的层次来表示。

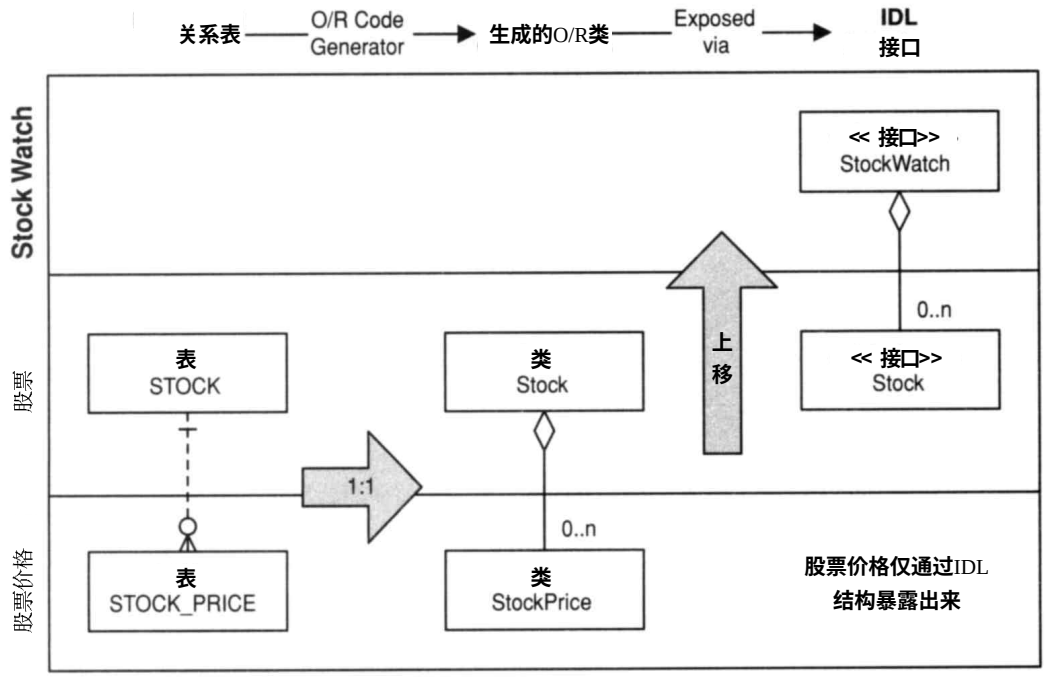

图10-3 聚集层次中的向上偏移

图10-3给出向上偏移的一个例子。回忆股票和股票价格信息的数据库框架：股票价格通过股票的主键与股票关联。例如，用对象 /关系代码生成器，可以得到 Stock类和StockPrice类作为直接映射的结果。但是，由于性能原因，把股票价格对象作为第一类 CORBA成员输出并无意义。相反，要在IDL层引入管理者接口（Stock Watch）。本例显示了前端和业务对象方法如何结合使用：持久股票对象成为第一类 CORBA成员。持久股票价格对象则只表示为 IDL数据结构，可以通过股票来访问（即，股票对于股票价格来说是前端）。这种向上偏移在现实世界中很普遍：CORBA业务对象通常是由细粒度数据对象组合成的。

但是，向上偏移只是CORBA对象和关系表示之间失配效果的一个例子。阻抗失配可能有比简单的层次向上偏移更严重的影响，尤其当关系数据高度规范化时，如图 10-4所示，我们经常会遇到CORBA业务对象与一个行集合相关的情况。这时，服务器端的直接对象/关系映射在多大程度上有助于实现CORBA业务对象是一个问题。

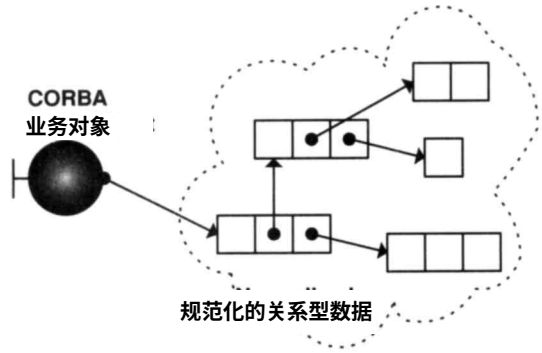

图10-4 强阻抗失配示例

10.4.2 对象/关系代码生成

通常，对象/关系代码生成器产生的类将直接映射到表，以及处理集合等事物的额外的辅助类。把CORBA和对象/关系代码生成结合的思想扩展了这种方法，它生成映射成表的 CORBA IDL接口。通常，这样的IDL 对象/关系代码生成器将生成IDL接口，生成的IDL接口的实现类，以及服务器主线程 —— 即一个完整CORBA服务器实现，这种工具的一个例子是 Persistence Software的DOCK产品。

例如，DOCK工具生成IDL接口Stock和StockFactory，可被CORBA客户机用来远程访问Stock表。StockFactory提供创建新股票的操作和查询操作，如querySQLWhere()操作，类似于在第9.3.6节“对象/关系代码生成”中讨论的。容易发现，用这种直接方法来进行基于 IDL的对象/关系代码生成是非常局限的。我们不想把SQL暴露给CORBA客户机，而想要封装数据框架的组件。但是，这种方法似乎向正确方向迈进了一步，远离了手工代码映射，并走向更有效的代码生成。很可能将来的对象/关系代码生成器将提供对映射的更精细的控制。例如，可能有一个工具允许选择把表映射成IDL接口还是IDL数据结构。这种结构允许在单个调用中获取对象状态。另外，我们通常只想通过IDL暴露表的一个子集，并需要通过增加操作来修改映射的灵活性。在我们能够自动生成持久CORBA事件服务的实现，来精确地支持事件服务规范中所定义的 IDL之前还有很长的路要走。对于需要高度规范化的表的那些部分，能否达到这个目标还成问题。但是系统的大部分通常是由具有简单的 CRUD操作的对象组成，在这样的情况下，代码生成看来还能很好地工作。

10.4.3 对象关系

在RDBMS中，关系是用键表示的。一行可有一个外键属性来表达对另一行的关系。键表达了双向的关系，因为总能从主键到达外键，反之亦然。这对于保证引用完整性（例如，基于数据库触发器）是重要的。

在CORBA环境下，关系通常通过CORBA对象引用表示。对象引用是单向的，并且不为对象引用直接保证引用完整性，因为服务器无法控制输出到客户机的引用。

CORBA关系服务在较高抽象层次定义了关系，关系被显式建模为 CORBA对象，这意味着关系和关系参与者被表示为IDL接口。关系服务支持双向关系，可以通过一般的 CORBA远程方

法调用来遍历。问题是CORBA关系只在第二和第三层处理关系，如果必须导航大量关系时，这种服务表现如何呢？通过比较ODBMS和关系服务在实现对象图遍历方法上的差异，我们来进一步探讨这个问题。

10.5 ODBMS

ODBMS为遍历对象关系提供了高度优化方法，这是由于它实现了成熟的缓冲策略，并且常常通过提供一些方法，把对象分成簇以使应用程序能优化缓冲工具。

图10-5显示了典型的ODBMS客户机如何遍历一个对象图，客户机第一次访问对象时，ODBMS把对象状态取回到由ODBMS管理的客户机端缓冲区。多数ODBMS被优化，以减少从数据库服务器下载数据的次数，下载不仅是下载请求的对象，还下载下次可能访问的其他对象的状态和关系信息。

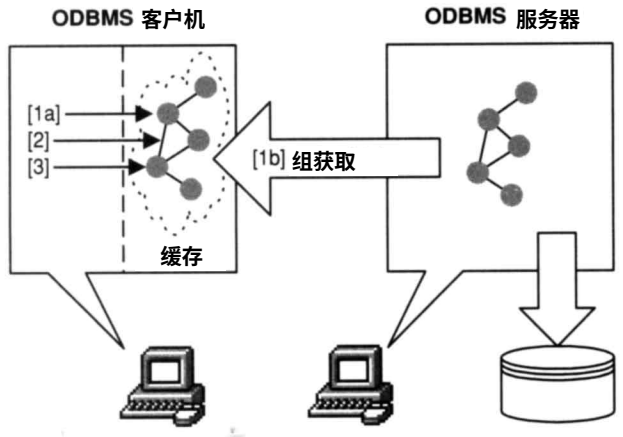

图10-5 ODBMS对象图导航

在图10-5中，客户机访问第一个对象(1a)导致获取整个一组相关对象进入缓冲区（1b)。如果现在客户机想导航关系（2）以访问相关对象（3)，因为它已活动于缓冲区，故无需再次请求数据库服务器。ODBMS提供的集簇技术越好，对缓冲区的利用就越好，在导航复杂对象图时，这对于保证高性能是重要的。

10.6 CORBA关系服务

CORBA关系服务使用IDL表示关系：两个CORBA对象间的关系由实现接口Relationship和Role的CORBA对象来表示。导航IDL表示的对象关系存在的问题是导航(1-3)的每一步都是一次远程访问，并且每次远程访问都附加一定的开销。另外，关系涉及了太多相对重量级CORBA对象。因此，遍历一个由CORBA关系服务表示的关系，将比遍历一个使用具有高度优化的缓冲和集簇机制的ODBMS慢几个数量级，本规范对这个问题的解释很模糊。例如，规范称get_other_related_object()操作能“通过缓冲其他角色和相关对象的对象引用来实现”。这并没有

说明如何为相关对象的簇而管理关系信息的缓冲。规范模糊地提出了通过选择对象工厂来管理集簇，但是没有说明如何在单个行为中实际地缓冲集簇的关系信息。最后，即使规范允许实现有效缓冲机制，IDL编译器生成的标准桩也是不够的:我们需要特定于客户机的方案来实现缓冲管理。

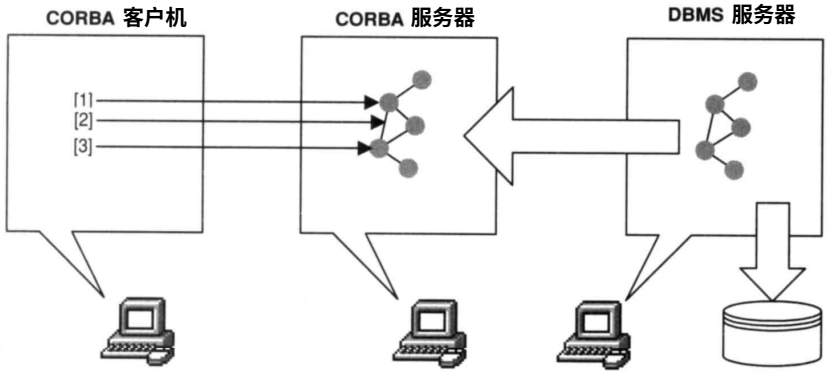

图10-6 使用关系服务来遍历关系

这样，CORBA关系服务看来主要适用于需要独立于数据储存的关系访问和高层抽象的系统，这些系统对性能和可伸缩性没有太高的要求。在性能和可伸缩性非常重要时，很难找到有效方法来用CORBA客户机从第三层有效地遍历关系，这一点必须反映在IDL设计上。

查询处理

和关系紧密相关的问题是查询处理。我们先考虑CORBA查询服务，然后在讨论不同的折衷办法前，看看可替换的方法。

10.7 CORBA查询服务

CORBA查询服务实际说明的是一般数据操纵，包括选择、插入、更新和删除，所以它更像是数据操纵服务，而不是简单的查询服务。查询服务使用谓词来处理查询，并设计为独立于任何特定的查询语言。但是，需要特别指出的是，实现必须至少支持SQL92或者OQL93。提供在集合上操纵数据，并执行于服务器端的操作的查询服务方式看来很有前途，尤其在远程导航关系问题上。理想的情况是，查询服务的实现应和DBMS紧密集成。不幸的是，查询服务还缺乏可用的商用实现。

10.8 特定于应用程序的解决方案

特定于应用程序的解决方案的产生不仅是因为查询服务缺乏可用的商用实现，而且在查询服务会带来太多开销或复杂性的地方，通常需要更简单的解决方案。特定于应用程序的解决方案的极其简单的一个例子可用StockWatch接口来实现，StockWatch接口提供了

getStockBySymbol()操作一这是查询接口的一个典型的、非常简单的形式，稍微复杂的查询操作允许传递包含通配符的字符串，返回匹配的集合。一般方法是使查询操作接受字符串名 /值对的序列，相当于查询的WHERE语句。CORBA商业对象服务，如第6章“对象定位”所述，提供了另一种使客户机能查询对象的有趣方法。

通常，一定类型业务对象的查询过程可在两个不同层次处理，业务对象本身提供访问和修改其状态的操作，而业务对象集合上的查询操作由作为该集合前端的管理者来提供。

10.9 权衡

不同方案间通常有所权衡，最好地封装底层数据框架的方法使得能够通过输出业务对象和关系的接口来执行访问，以便客户机能使用访问远程对象的一般 CORBA方式来查询和导航。这种方法虽然提供了高层的封装和灵活性，但是却不太可伸缩，以致可能不适用于大量对象或复杂关系。

通常必须提供特殊的查询操作，通过委托服务器端工作来处理性能问题。虽然这种方法似乎更可伸缩，但是也更不灵活，因为它严格要求客户机具有特殊的查询操作提供的功能，这可能会暴露对象内部细节而削弱封装。

SQL和OQL使客户机能向服务器发送查询，以便在服务器端执行。这种方法的好处是很可能有最好的性能。不幸的是，它也把数据的内部结构完全暴露给客户机。把数据框架暴露给CORBA客户机违背了用CORBA封装复杂性的思想。这些权衡如图10-7所示。

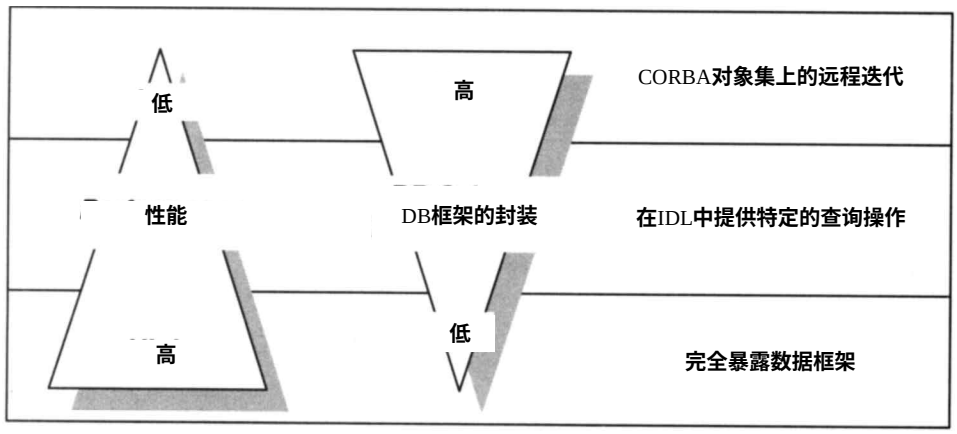

图10-7 通过CORBA查询对象的各种权衡

10.9.1 做实际工作的层次

设计过程中有一个问题，即在哪里执行实际工作。关系数据库管理系统设计是在第一层执行所有工作：客户机发送查询并接收结果，但是实际工作是由数据库服务器做的。关系数据库管理系统实现了高级的缓冲和集簇机制，使客户机能在第二层有效地遍历复杂对象图。当应用

程序逻辑结构复杂，并在对象图上需要大量导航，而这些无法用静态的数据操纵语言（如 SQL）来表达时,通常使用ODBMS。注意，除了客户机端的对象导航功能外，多数 ODBMS还提供像OQL一类查询语言，OQL能用来向数据库服务器发送请求，所以能在第一层执行。 OQL和SQL的一个重要区别是：OQL返回对象引用，而SQL只返回数据。

可以看到，这里有一些冲突的需求。在第一层执行工作通常是最有效的方法，因为第一层可以直接访问储存卷。在第二层执行工作涉及和第一层的相互通信，这增加了性能开销。传统上，特定于应用程序的业务逻辑在第二层执行，把数据库服务器功能和应用程序功能分离开来。目前，关系数据库厂商正试图较紧密地集成数据服务器逻辑和应用程序逻辑。 Oracle 的盒式磁盘(cartridge)是在第二层内使查询更灵活的一个例子，而 Informix的数据刀片(date blade)向应用程序开放了第一层。

有时不用这些对象/关系混合方法，而用对象/关系代码生成器来桥接对象/关系失配。一些对象/关系代码生成器用缓冲机制来向客户机提供数据访问和导航机制，其性能比单纯的 SQL更好。问题是这些工具必须与关系数据库服务器一道工作，而后者并没有为这些目的而优化，以致它们的执行情况未必比在数据库服务器端提供了支持客户机端缓冲的专用逻辑的 ODBMS 或者对象/关系混合方法好。

只要商用数据库不提供对 CORBA服务(如关系服务或者查询服务)的直接支持，就必须在应用层，即第二层或者第三层进行控制。显然，把解决方案上移至第二或第三层很可能会在性能和可伸缩性方面带来负面影响。但是，在很多情况下，独立于数据储存对业务对象访问是极其重要的。例如，很多企业资源计划（ ERP）系统开始开放其接口，并通过 CORBA输出业务对象和关系。如果应用程序需要跨越两个 ERP系统（如SAP 和BAAN）执行业务事务处理，那么这些系统把需要的对象和关系作为 CORBA对象输出将更为理想。显然，接口的设计必须十分小心，以保证这种处理将来能够调整。

10.9.2 事务和并发

在单个商用事务中访问驻留在 SAP和BAAN组件中的业务对象直接引出了一个更一般的问题，即如何在CORBA和数据库集成的上下文中处理事务。第 11章“ CORBA环境下的事务 ”，一般地论述了事务，第 12章“ 分布式事务处理 ”集中于分布式事务处理，第 13章“ 用户会话 ”主要讨论长期事务。这些章也从不同角度讨论了并发问题。

10.9.3 对象标识

前面关于与数据库集成相关的 IDL设计问题的讨论假定了一些对象标识。在前端的例子中使用了次级OID来标识证券。对业务对象的例子，我们只是假定 CORBA的对象概念解决了对象标识的问题。

标识实际上是持久性的一个关键元素。关系数据库管理系统使用数据库键来实现标识。对象数据库管理系统通常提供系统生成的对象标识（ OID)。问题是CORBA没有为对象标识提供有力的模型，至少从数据库的观点看是这样。 CORBA对象引用（IOR）包含了身份标识数据，但是这仅仅相对于一个服务器（或一个对象适配器）标识了对象。而且，一个 IOR可表示多个

CORBA对象（在一对多的通信中），一个CORBA对象也能由多个不同IOR表示。

至少，多数ORB提供了方法，让应用程序为CORBA对象定义一些标识数据，并将其封装在对象IOR内。这使我们有机会在IOR内嵌入足够的信息，以在整个系统中唯一地标识CORBA业务对象的持久数据。注意，为在系统全局唯一地标识应用层对象而选择一个框架是重要的：依靠封装在IOR中的其他信息，如主机名或端口号等是毫无意义的，因为它们可能改变。所以，即使对象标识符只在服务器端阐释，它们在系统环境下也必须是独立于服务器的。

10.9.4 伺服对象管理

标识请求目标的对象ID能够被有效地映射成伺服对象，并且通过伺服对象池模式实现请求，这在第2章“CORBA回顾”中讨论过了。除了一般资源管理和清除外，池模式还有助于通过保留池中的常用对象来减少对象故障的数量。数据库可能包含数十亿的对象（或表示CORBA对象状态的数据结构）。之所以不想在内存中保留所有对象的原因很明显，首先，要使进程保持最小，以避免换页和不必要的虚拟内存消耗。其次，如果没有成熟的同步机制的话，从数据库载入大量对象到内存将给应用程序并发行为带来负面后果。

10.10 无状态伺服对象

无状态伺服对象作为暂时的实体，代表了使用SQL向数据库服务器输入的IIOP请求，这意味着无状态伺服对象在IDL接口操作语义和SQL查询间提供了某种转换。

很多BOA代的ORB并不提供对无状态伺服对象的很好支持，因为它们在OID和伺服对象之间实现了一对一映射，即，一个伺服对象只能为一个特定的CORBA对象处理请求。是否值得用对象池模式来减少对象故障数量，取决于伺服对象激活和冻结的代价。如果ORB运行时模块不能有效地管理伺服对象的激活和冻结，那么，把无状态伺服对象保留在池中以减少对象激活和冻结的频率就可能很有意义。如果ORB运行并不附加很大开销，那么，可能在预请求的基础上激活和冻结伺服对象更好些，减少了池管理的开销。

对POA代的ORB，管理无状态伺服对象较为容易，例如使用缺省伺服对象能控制所有输入的请求，而不考虑目标对象标识。缺省伺服对象包含了来自POA的目标对象标识，并用它来为此服务构造合适的查询。

无状态伺服对象是使用CORBA来输出那些用底层数据库实现的服务的很普通方法。对比一下今天成千上万Internet应用程序使用的、结合了HTML、HTTP和CGI的方法，Internet客户机向HTTP服务器发送HTTP请求，激活了CGI脚本来执行请求。CGI脚本是典型无状态的，并用SQL把HTTP请求委托给数据库服务器。使用CORBA无状态伺服对象在思想上类似于这种方法，但是还提供了CORBA的其他好处，包括面向对象系统提供的强大的抽象，CORBA服务的更高层框架和请求参数的自动打包和解包。

10.11 有状态伺服对象

使用有状态伺服对象时，首要问题是伺服对象状态是否绑定于事务边界。如果伺服对象状态仅在单个事务期间保留于内存，这种方法很少有问题，如果试图在事务外把状态信息保留在

内存，就可能很困难。通常，这种方法需要成熟的缓冲同步机制。但是，很少有商用数据库适于支持跨事务的缓冲区同步 —— 例如，提供有效的触发机制来同步缓冲区。

除了事务边界，还必须看看进程边界。在进程级，必须保证缓冲区在应用程序服务器间同步：很多CORBA系统有镜像应用程序服务器。缓冲区同步有很多方法，如用多点传送技术来公布更改。

有时，通过实现十分简单的缓冲策略，仅缓冲只读的值或不常改变的值，可能获得显著的性能增益。例如，股票对象实现能在缓冲区内保留股票的符号，因为股票符号很少改变，即使在公司合并时也是如此。这时，缓冲区能够被有效地管理，例如，在发生改变时简单地使应用程序的所有缓冲区无效。

对象池模式之所以称为对象池，是因为其目的不是维持对象缓冲区，而是解决资源管理问题。即使此模式能够用于为持久对象实现特定于应用程序的缓冲策略，这似乎也是一个复杂的任务，尤其在频繁改变的值需要细粒度的缓冲区同步时。下面简单讨论ORB如何与ODBMS集成，以便不必在应用层处理缓冲，而能依靠ODBMS缓冲区管理。

10.12 数据库适配器

一些ORB厂商指出：通过提供自己专有的数据库适配器来具体说明对象数据库适配器的结构，这一点还缺乏CORBA标准。例如，IONA Technologies提供建立数据库适配器的类属框架，可用来集成Orbix ORB和ODBMS(如Versant和ObjectStore)。这个框架的关键概念是把CORBA对象实现和持久对象相分离：ORB向临时伺服对象发送输入调用。此伺服对象只有一个成员变量，即一个灵巧指针。伺服对象使用这个灵巧指针简单地把每个调用委托给持久对象。多数 ODBMS支持灵巧指针的思想，它负责保证在访问缓冲对象前目标对象存在于本地缓冲区[⊖]，如图10-8所示。

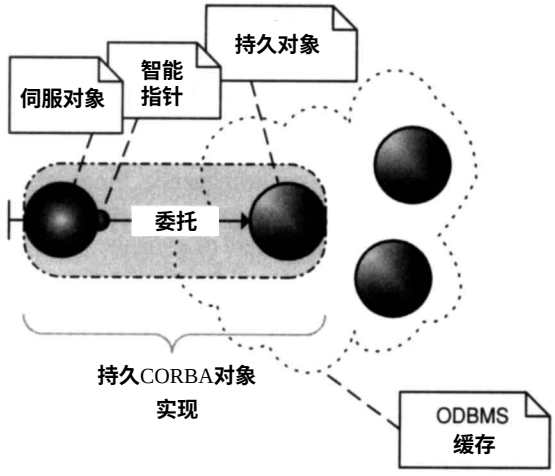

图10-8 ORB/ODBMS集成

⊖ 注意ObjectStore是一个例外。ObjectStore的虚拟管理体系允许应用程序使用标准虚拟存储器指针访问持久对象，这是使用ObjectStore的标准方式。在页面错误时，ObjectStore负责捕获错误并载入需要的页面。但ObjectStore也支持智能指针，智能指针可以用于上面描述的情况。

临时伺服对象和持久对象的完全分离是重要的，这样避免了在持久对象中和临时的特定ORB的数据成员有关的问题。这里陈述的解决方法基本能用于任何 ORB，问题只是如何实现伺服对象：是手工写，还是数据库适配器提供自动生成的工具？另一个要对象数据库适配器解决的问题是CORBA对象标识符和灵巧指针之间的映射：通常适配器会支持后期绑定，所以适配器必须能把CORBA对象标识符转换成灵巧数据库指针。幸运的是，多数 ODBMS提供了灵巧指针和字符串之间相互转换的工具。这种串形式的 ODBMS对象ID能嵌入于CORBA对象ID。最后，对象适配器需要处理事务。通常 ODBMS不允许应用程序在事务外访问对象。因此，适配器必须管理事务。在第12章“分布式事务处理”中，将进一步探讨不同风格的事务处理。

CORBA 数据库适配器帮助我们集成了持久对象和 CORBA对象实现，以建立持久CORBA对象。但是，这只是 CORBA和数据库集成的更大问题中的一小部分。例如，这里讨论的 CORBA数据库适配器，并没有解决如何把 CORBA IDL数据类型映射成可在 DBMS中存储的数据类型的问题，反之亦然。这是将出现的持久状态服务要解决的问题。

10.13 海量对象CORBA系统

企业系统通常需要支持数千兆的数据（或者按 CORBA说法，海量对象）。ORB是封装内部数据结构和使数据可在更高抽象层（如 CORBA业务对象形式）访问的有力工具。技术上，多数ORB能使用后期绑定轻易地处理支持海量对象的需求，特别是 POA对缺省伺服对象的支持，使它可通过使用无状态伺服对象的 CORBA来直接访问大量数据。对于有状态伺服对象，系统将依靠有效的对象池实现，在内存中保存频繁使用的对象，并抛弃不常用的对象以避免应用程序服务进程占用巨大的空间。

海量对象CORBA系统的奥妙通常在于正确的 IDL设计。IDL应该恰当地结合业务对象和前端风格的服务器对象，以允许客户机在业务对象集合上执行操作。回想在第 5章“性能要求”中讨论的CORBA远程访问的开销和对象访问模式的性能影响，在为系统设计 IDL时应注意，让数千兆的数据可访问比设计复杂的服务器端对象池和缓冲策略更为重要，应尽量设计 IDL以使CORBA客户端能处理大量的远程激发，还应尽量依赖数据库服务器提供的可伸缩性和查询功能。