第八章　数据访问中间件

李会格　博士/讲师

E-mail:1034434100@qq.com

数据访问中间件

- 数据访问中间件对业务开发人员屏蔽底层数据操作的繁琐细节，提供对多种数据源进行统一访问的接口。
- 数据访问中间件主要用于处理业务服务层和数据层之间的交互操作，目的是将业务和复杂的数据访问操作隔离开。
- 本章主要围绕数据库访问中间件展开，介绍ODBC、OLE DB和ADO的概念和它们之间的关系，其次介绍基于JDBC、对象-关系映射，以及JPA持久化框架等技术。

大纲

- 开放数据库连接
  - ODBC
  - OLE DB
  - Active Data Objects
  - JDBC
- 对象-关系映射ORM
  - ORM的概念
  - 对象与数据库间的映射
  - 对象-关系映射例子
  - Hibernate框架
- JPA持久化框架
- 其他持久化框架
- 小结

ODBC

- ODBC是Open Database Connectivity的英文简写。是一种用来在关系或非关系型数据库管理系统（DBMS）中存取数据的标准应用程序数据接口。
- 由微软倡导，被业界广泛接受。
- ODBC建立了一组规范并提供了一组对数据库访问的标准API（应用程序编程接口）。

ODBC

- 一个基于ODBC的应用程序对数据库的操作不依赖任何DBMS，不直接与DBMS打交道，所有的数据库操作由对应的DBMS的ODBC驱动程序完成。
- 因此，ODBC的最大优点是能以统一的方式处理所有的数据库，用它生成的程序与数据库或数据库引擎是无关的。
- ODBC提供了对SQL语言的支持，其API利用SQL来完成其大部分任务。

ODBC

- ODBC的结构
- 1) 应用程序接口：
  - ✓ 屏蔽不同的ODBC数据库驱动器之间函数调用的差别，为用户提供统一的SQL编程接口。
- 2) 驱动器管理器：
  - ✓ 为应用程序装载数据库驱动器。
- 3) 数据库驱动器：
  - ✓ 实现ODBC的函数调用，提供对特定数据源的SQL请求。如果需要，数据库驱动器将修改应用程序的请求，使得请求符合相关的DBMS所支持的文法。
- 4) 数据源：
  - ✓ 由用户想要存取的数据以及与它相关的操作系统、DBMS和用于访问DBMS的网络平台组成。

ODBC

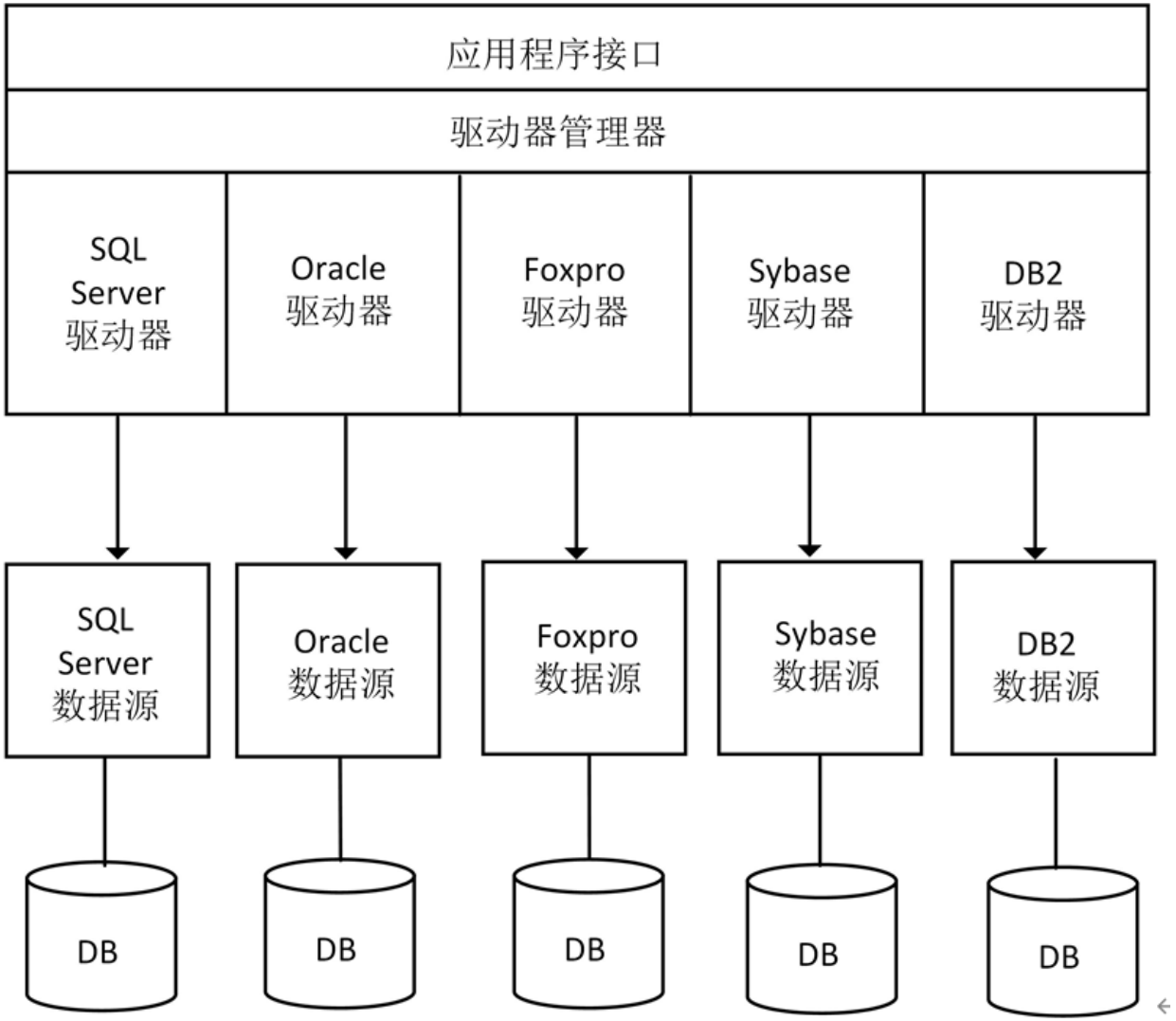


图 7-1 ODBC 的体系结构

ODBC

- ODBC的调用
- 应用程序要访问一个数据库，首先必须用ODBC管理器注册一个数据源，管理器根据数据源提供的数据库位置、数据库类型及ODBC驱动程序等信息，建立起ODBC与具体数据库的联系。

- 在ODBC中，ODBC API并不能直接访问数据库，而必须通过数据库驱动器与数据库交换信息。驱动器管理器负责将应用程序对ODBC API的调用传递给正确的数据库驱动器，而数据库驱动器在执行完相应的操作后，将结果通过驱动器管理器返回给应用程序。

ODBC

- ODBC的调用
- 当应用系统发出调用与数据源进行连接时，数据库驱动器能管理通信协议。

- 当建立起与数据源的连接时，数据库驱动器便能处理应用系统向DBMS发出的请求，对分析或发自数据源的设计进行必要的翻译，并将结果返回给应用系统。

大纲

- 开放数据库连接
  - ✓ ODBC
  - ✓ OLE DB
  - ✓ Active Data Objects
  - ✓ JDBC
- 对象-关系映射ORM
  - ✓ ORM的概念
  - ✓ 对象与数据库间的映射
  - ✓ 对象-关系映射例子
  - ✓ Hibernate框架
- JPA持久化框架
- 其他持久化框架
- 小结

OLE DB

- 随着数据源日益复杂化，应用程序很可能需要从不同的数据源取得数据。
  - ✓ 比如从Excel文件，Email，Internet/Intranet上的电子签名等信息。然而，ODBC仅支持关系数据库，以及传统的数据库数据类型。

- 1997年，微软公司引入了OLE DB技术。

OLE DB

- OLE DB（Object Link and Embed），即对象连接与嵌入，是微软的战略性的通向不同的数据源的低级应用程序接口。
- OLE DB不仅包括微软资助的标准数据接口开放数据库连通性（ODBC）的结构化问题语言（SQL）能力，还具有面向其他非SQL数据类型的通路。

OLE DB

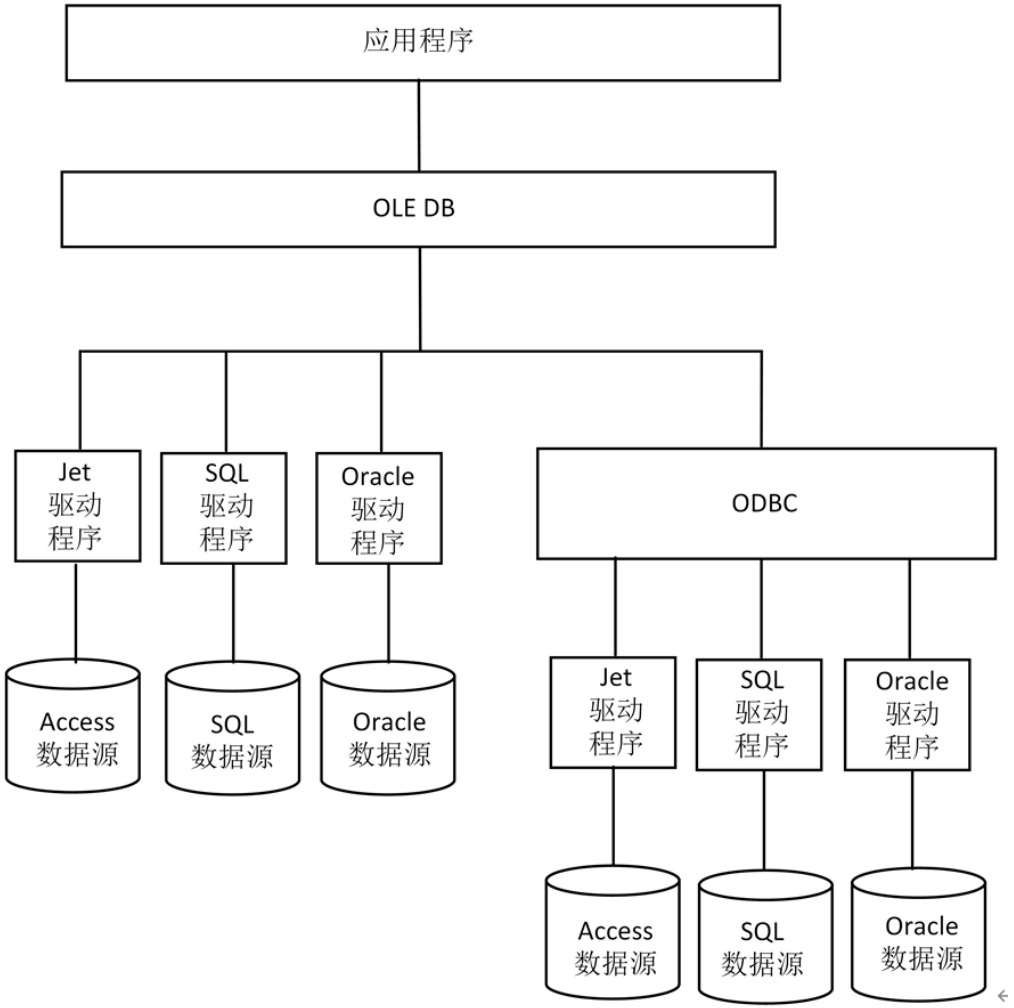


图 7-2 OLE DB 工作原理示意图

OLE DB

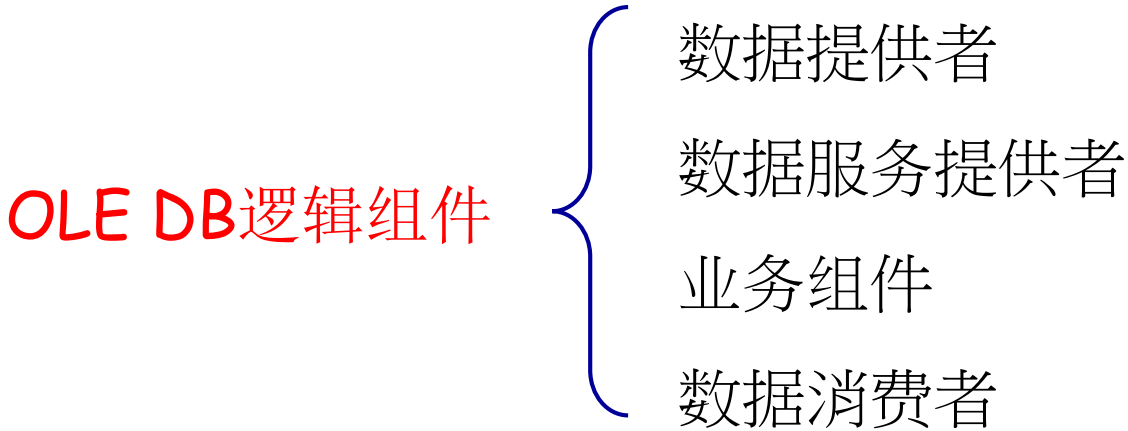
- 作为微软的组件对象模型（COM）的一种设计，OLE DB是一组读写数据的方法。

- OLD DB中的对象主要包括数据源对象、阶段对象、命令对象和行组对象。

- OLE DB的应用程序的请求序列：
  - ✓初始化、OLE 连接到数据源、发出命令、处理结果、释放数据源对象并停止初始化OLE。

OLE DB

- OLE DB逻辑组件
- OLE DB将传统的数据库系统划分为多个逻辑组件，这些组件之间相对独立又相互通信。



OLE DB

- OLE DB逻辑组件
- 数据提供者（Data Provider）
  - ✓ 提供数据存储的软件组件，如普通的文本文件、主机上的复杂数据库，或者电子邮件存储
- 数据服务提供者（Data Service Provider）
  - ✓ 位于数据提供者之上，使得数据提供者提供的数据不论是以何种物理方式组织和存储的，都能够以表的形式向外表示，并实现数据的查询和修改功能。
- 业务组件（Business Component）
  - ✓ 利用数据服务提供者来专门完成某种特定业务信息处理的可重用的功能组件。
- 数据消费者（Data Consumer）
  - ✓ 任何需要访问数据的系统程序或应用程序

OLE DB

- OLE DB与ODBC的区别
- OLE DB和ODBC标准都是为了提供统一的访问数据接口，但是OLE DB并不是替代ODBC的新标准。

- ODBC标准的对象是基于SQL的数据源（SQL-Based Data Source），而OLE DB的对象则是范围更为广泛的任何数据存储。

- 符合ODBC标准的数据源是符合OLE DB标准的数据存储的子集。符合ODBC标准的数据源要符合OLE DB标准，还必须提供相应的OLE DB服务程序。

大纲

- 开放数据库连接
  - ✓ ODBC
  - ✓ OLE DB
  - ✓ Active Data Objects
  - ✓ JDBC
- 对象-关系映射ORM
  - ✓ ORM的概念
  - ✓ 对象与数据库间的映射
  - ✓ 对象-关系映射例子
  - ✓ Hibernate框架
- JPA持久化框架
- 其他持久化框架
- 小结

Active Data Objects

- OLE DB是一个非常良好的架构，允许程序员存取各类数据。但是OLE DB太底层化，而且在使用上非常复杂，这让OLE DB无法广为流行。

- 为了解决这个问题，并且让VB和脚本语言也能够藉由OLE DB存取各种数据源，Microsoft同样以COM技术封装OLE DB为ADO对象，简化了程序员数据存取的工作。

Active Data Objects

- ADO (ActiveX Data Objects)是微软公司一个用于存取数据源的COM组件。
- 它提供了编程语言和统一数据访问方式OLE DB的一个中间层。
- 允许开发人员编写访问数据的代码而不用关心数据库是如何实现的，而只用关心到数据库的连接。
- 访问数据库的时候，关于SQL的知识不是必要的，但是特定数据库支持的SQL命令仍可以通过ADO中的命令对象来执行。

Active Data Objects

- ADO的工作原理
- ADO被设计来继承微软早期的数据访问对象层，包括RDO (Remote Data Objects)和DAO(Data Access Objects)。
- ADO包括了6个类：Connection，Command，Recordset，Errors，Parameters，Fields。
- ADO使得用户我们不用过多的关注OLE DB的内部机制，只需要了解ADO通过OLE DB创建数据源的几种方法即可，就可以通过ADO轻松地获取数据源。

Active Data Objects

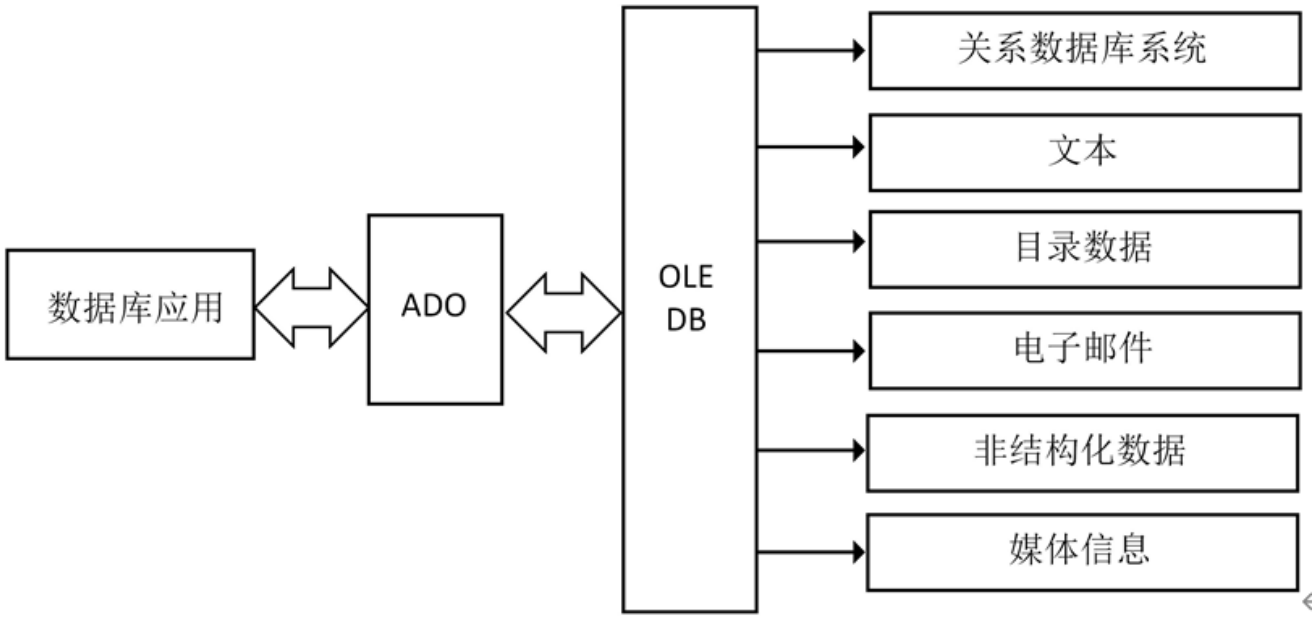


图 7-3 ADO 工作原理

Active Data Objects

- ADO对象模型
- 1) Connection对象
  - ✓ 在数据库应用里操作数据源都必须通过该对象，这是数据交换的环境。Connection对象代表了同数据源的一个会话，在客户/服务器模型里，这个会话相当于同服务器的一次网络连接。不同的数据提供者提供的该对象的集合、方法和属性不同。
  - ✓ Connection对象的Open和Close方法可以建立和释放一个数据源连接。Execute方法可以执行一个数据操作命令，使用BeginTrans、CommitTrans和RollbackTrans方法可以启动、提交和回滚一个处理事务。

Active Data Objects

- ADO对象模型
- 2) Command对象
  - ✓ Command对象是一个对数据源执行命令的定义，使用该对象可以查询数据库并返回一个Recordset对象，可以执行一个批量的数据操作，可以操作数据库的结构。不同的数据提供者提供的该对象的集合、方法和属性不同。
- 3) Parameter对象
  - ✓ Parameter对象在Command对象中用于指定参数化查询或者存储过程的参数。大多数数据提供者支持参数化命令，这些命令往往是已经定义好的，只是在执行过程中调整参数的内容。

Active Data Objects

- ADO对象模型
- 4) Recordset对象
  - ✓ 如果执行的命令是一个查询并返回存放在表中的结果集，这些结果集将被保存在本地的存储区里，Recordset对象是执行这种存储的ADO对象。通过Recordset对象可以操作来自数据提供者的数据，包括修改、插入和删除行。
- 5) Field对象
  - ✓ Recordset对象的一个行由一个或者多个Field对象组成，如果把一个Recordset对象看成一个二维网格表，那么Field对象就是这些列。这些列里保存了列的名称、数据类型和值，这些值是来自数据源的真正数据。为了修改数据源里的数据，必须首先修改Recordset对象各个行里Field对象里的值，最后Recordset对象将这些修改提交到数据源。

Active Data Objects

- ADO对象模型
- 6) Error对象
  - ✓ Error对象包含了ADO数据操作时发生错误的详细描述，ADO的任何对象都可以产生一个或者多个数据提供者错误，当错误发生时，这些错误对象被添加到Connection对象的Errors集合里。
  - ✓ 当另外一个ADO对象产生一个错误时，Errors集合里的Error对象被清除，新的Error对象将被添加到Errors集合里。

Active Data Objects

- ADO对象模型
- 7) Property对象
  - ✓ Property对象代表了一个由提供者定义的ADO对象的动态特征。
  - ✓ ADO对象有两种类型的Property对象：内置的和动态的。
  - ✓ 内置的Property对象是指那些在ADO里实现的在对象创建时立即可见的属性，可以通过域作用符直接操作这些属性。
  - ✓ 动态的Property对象是指由数据提供者定义的底层的属性，这些属性出现在ADO对象的Properties集合里

Active Data Objects

- ADO与ODBC、OLE DB的关系

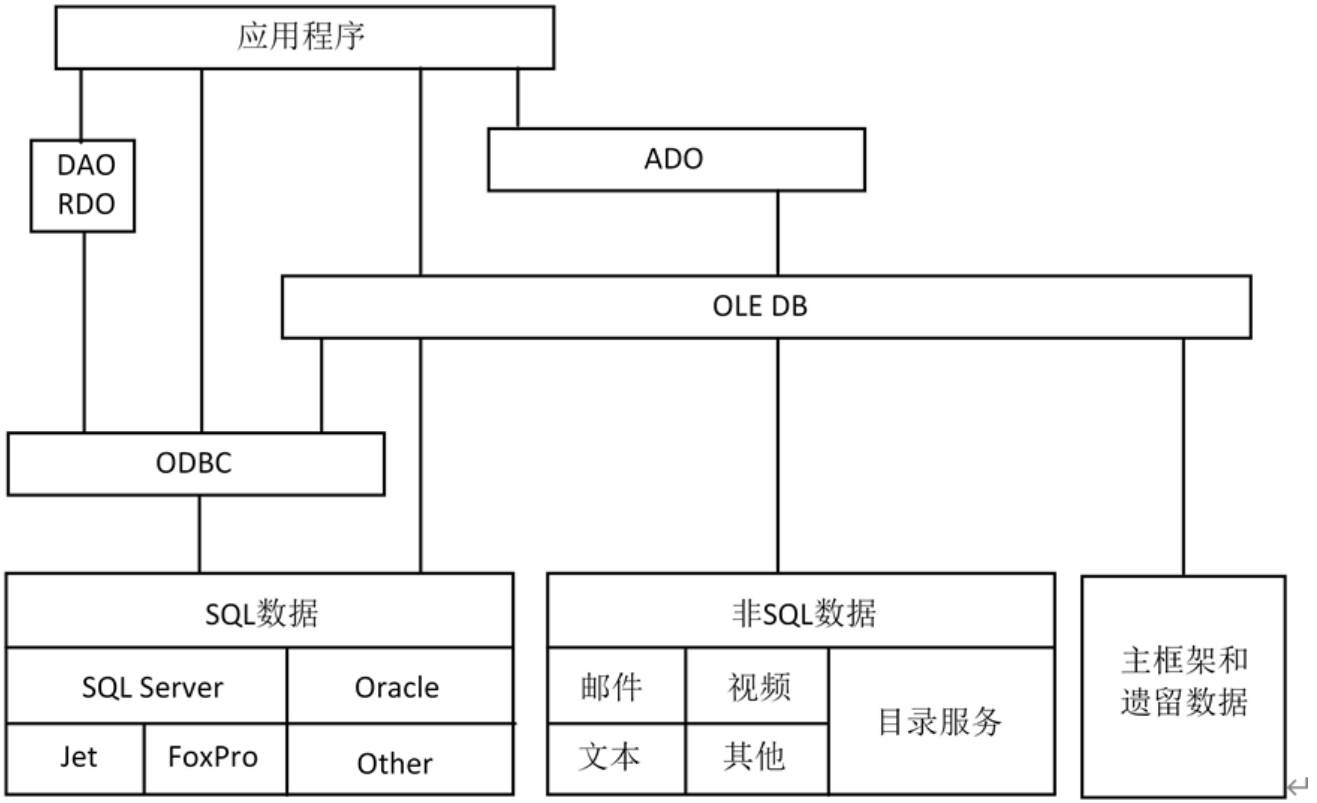


图 7-4 ODBC、OLE DB 和 ADO 关系图

大纲

- 开放数据库连接
  - ✓ ODBC
  - ✓ OLE DB
  - ✓ Active Data Objects
  - ✓ JDBC
- 对象-关系映射ORM
  - ✓ ORM的概念
  - ✓ 对象与数据库间的映射
  - ✓ 对象-关系映射例子
  - ✓ Hibernate框架
- JPA持久化框架
- 其他持久化框架
- 小结

JDBC

- ODBC仅支持关系数据库，以及传统的数据库数据类型，并且只以C/C++的API形式提供服务，因而无法符合日渐复杂的数据存取应用，也无法让脚本语言使用。
- 随着JAVA语言的流行， Sun公司推出了面向各关系型数据库厂商的数据库访问规格与标注JDBC。

JDBC

- JDBC(Java Data Base Connectivity)是Java与数据库的接口规范。
- JDBC定义了一个支持标准SQL功能的通用底层的应用程序编程接口(API)，它由Java 语言编写的类和接口组成，旨在让各数据库开发商为Java程序员提供标准的数据库API。

JDBC

- JDBC的设计在思想上沿袭了ODBC，同时在其主要抽象和SQL调用级接口实现上也沿袭了ODBC，这使得JDBC容易被接受。
  - ✓应用程序、驱动程序管理器、驱动程序和数据源。
- JDBC API定义了若干Java中的类：
  - ✓数据库连接、SQL指令、结果集、数据库元数据
- 允许Java程序员发送SQL指令并处理结果。通过驱动程序管理器，JDBC API可利用不同的驱动程序连接不同的数据库系统。

JDBC

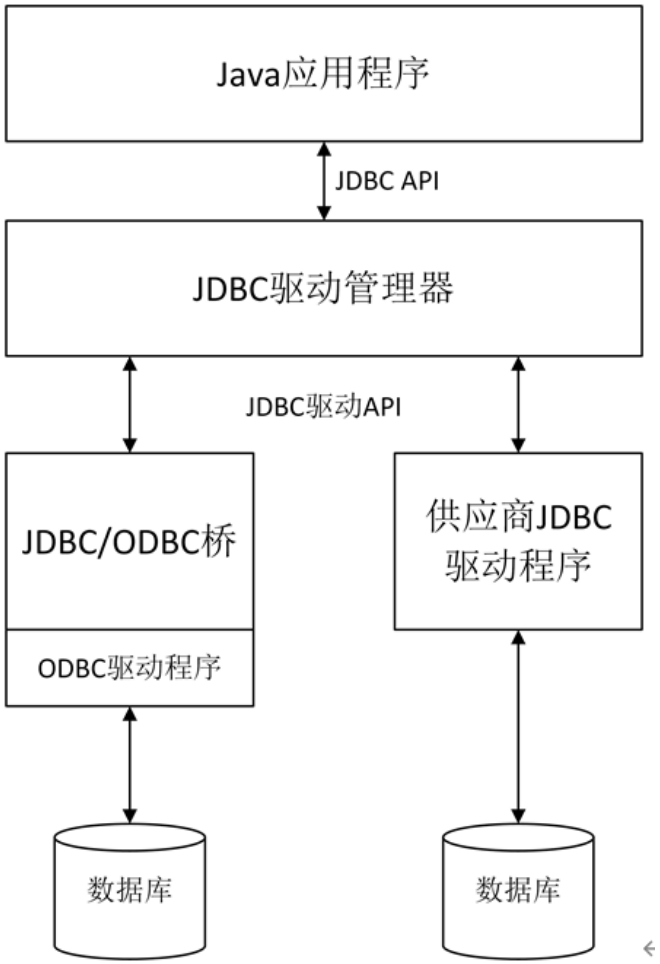


图 7-5 JDBC 总体结构

JDBC

- JDBC的常用接口
- Driver接口
  - Driver接口由数据库厂家提供，作为java开发人员，只需要使用Driver接口就可以了。在编程中要连接数据库，必须先装载特定厂商的数据库驱动程序，不同的数据库有不同的装载方法。
- Connection接口
  - Connection与特定数据库的连接（会话），在连接上下文中执行sql语句并返回结果。

大纲

- 开放数据库连接
  - ✓ ODBC
  - ✓ OLE DB
  - ✓ Active Data Objects
  - ✓ JDBC
- 对象-关系映射ORM
  - ✓ ORM的概念
  - ✓ 对象与数据库间的映射
  - ✓ 对象-关系映射例子
  - ✓ Hibernate框架
- JPA持久化框架
- 其他持久化框架
- 小结

ORM的概念

- 结构化查询语言SQL（STRUCTURED QUERY LANGUAGE）是数据库系统最重要的操作语言，并且影响深远。
- 然而，SQL是过程化的语言，这与大行其道的面向对象语言，存在某种程度的不协调和不匹配。数据库中用表格、行、列来表示数据，而在操作语言中则大多表示为类和对象。
- 随着面向对象的软件开发方法的发展，对象-关系映射（Object Relational Mapping，简称ORM）技术应运而生。

ORM的概念

- ORM系统一般以中间件的形式存在，主要实现程序对象到关系数据库数据的映射。
- ORM是通过使用描述对象和数据库之间映射的元数据，将程序中的对象自动持久化到关系数据库中。
- 实际应用中，ORM中间件在关系型数据库和业务实体对象之间作一个映射，使得用户在具体的操作业务对象的时候，就不需要再去和复杂SQL语句打交道，只要像平时操作对象一样操作它就可以了。

ORM的概念

- 常见的ORM框架有：Hibernate、iBATIS、TopLink、Castor JDO、Apache OJB等。

- 一般的ORM系统包括以下四个部分：
  - 一个对持久类对象进行CRUD操作的API；
  - 一个语言或API用来规定与类和类属性相关的查询；
  - 一个规定mapping metadata的工具；
  - 一种技术可以让ORM的实现同事务对象一起进行脏检查（dirty checking），惰性关联抓取（lazy association fetching）以及其他的优化操作。

大纲

- 开放数据库连接
  - ✓ ODBC
  - ✓ OLE DB
  - ✓ Active Data Objects
  - ✓ JDBC
- 对象-关系映射ORM
  - ✓ ORM的概念
  - ✓ 对象与数据库间的映射
  - ✓ 对象-关系映射例子
  - ✓ Hibernate框架
- JPA持久化框架
- 其他持久化框架
- 小结

对象与数据库间的映射

- 1) 类与数据库中表的映射：
  - ✓ 数据库中的每一张表对应编程语言中的一个类，当用户对类进行基本操作（如创建实现，修改对象的属性，删除一个实例）时，ORM框架会自动对数据库中的表进行相应的CRUD操作。
- 2) 对象与表中记录的映射：
  - ✓ 关系数据库中的一张表可能有多条记录，每条记录对应类的一个实例，当用户对一个对象进行修改时，ORM框架会自动对数据表中的相应记录进行修改。
- 3) 类的属性与数据库中表的字段的映射：
  - ✓ 数据库中表的字段的数据类型与类中的属性的类型也是一一对应的。

对象与数据库间的映射

- 除了类与对象的映射，ORM还需要考虑引用完整性与关系约束检查、对象标识符等的映射。
- 由于面向对象设计的机制与关系模型的不同，面向对象设计与关系数据库设计之间是不匹配的。面向对象设计基于如耦合、聚合、封装等理论，而关系模型基于数学原理。
- 不同的理论基础导致了不同的优缺点。ORM系统需要一种映射方法来解决这种不匹配，从而获成功的设计。

大纲

- 开放数据库连接
  - ✓ ODBC
  - ✓ OLE DB
  - ✓ Active Data Objects
  - ✓ JDBC
- 对象-关系映射ORM
  - ✓ ORM的概念
  - ✓ 对象与数据库间的映射
  - ✓ 对象-关系映射例子
  - ✓ Hibernate框架
- JPA持久化框架
- 其他持久化框架
- 小结

对象-关系映射例子

- 作为一个简单的例子，以下定义一个学生Student类，同时定义一个简单的映射例子（基于Hibernate）

```
@Entity
public class Student {
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private long id;
   //学生类绑定到的关系数据库中 ID 生成为自增
   @Column(nullable = false)
   private String name;
   //学生姓名作为关系数据库表中的一列属性
   public Student(){ }
   public Student(String s){ name=s;}
   public String toString() {
      return  id + " -- " + name ;
   }
   public void setId(Long id) { this.id=id; }//为私有属性设置访问方法
   public long getId() { return id; }
   public void setName(String name) { this.name=name;}
   public String getName() { return name; }
}
```

对象-关系映射例子

```
public class StudentHibernate {
   SessionFactory factory;
   public void setup() {//在使用 Hibernate 之前需要调用 setup()方法
      Configuration configuration = new Configuration();
      //这里采用默认的配置，配置信息可以在 hibernate.cfg.xml 文件中修改
      configuration.configure();
      ServiceRegistryBuilder srBuilder = new ServiceRegistryBuilder();
      srBuilder.applySettings(configuration.getProperties());
      ServiceRegistry serviceRegistry = srBuilder.buildServiceRegistry();
      factory = configuration.buildSessionFactory(serviceRegistry);
      //根据配置信息拿到 sessionFactory
   }
   public void saveStudentRecord() {
      Student student1 = new Student("Wang Xiao");
      Student student2 = new Student("Zhang Hua");
      Session session = factory.openSession();//开启数据库会话
      Transaction tx = session.beginTransaction();//开启数据库事务
      session.persist(student1);//进行 ORM 映射，对象转换到关系数据库中
      session.persist(student2);
      tx.commit();//数据库事务提交
      session.close();
   }
```

对象-关系映射例子

```
    public void readStudentRecord() {
        Session session = factory.openSession();
        List<Student> list = (List<Student>) session.createQuery("from Student").list();
        //session.createQuery 通过类 SQL 语句作为参数，获得数据
        for (Student m : list) {
            System.out.println(m);
        }
        session.close();
    }

    public static void main(String[] args) {
        StudentHibernate sh=new StudentHibernate();
        sh.setup();
        sh.saveStudentRecord();
        sh.readStudentRecord();
    }
}
```

Hibernate框架

- Hibernate是一个开放源代码的对象关系映射框架，它对JDBC进行了非常轻量级的对象封装，使得Java程序员可以随心所欲的使用对象编程思维来操纵数据库。
- Hibernate可以应用在任何使用JDBC的场合，既可以在Java的客户端程序使用，也可以在Servlet/JSP的Web应用中使用。
- 最具革命意义的是，Hibernate可以在应用EJB的JavaEE架构中取代CMP （Container Managed Persistence），完成数据持久化的重任。

Hibernate框架

- 在Hibernate框架中，用户创建一系列的持久化类。每个类的属性都可以简单的看作是和一张数据库表的属性一一对应，也可以实现关系数据库的各种表关联的对应。
- 当需要相关操作时，用户不用再关注数据库表，无需再去一行行的查询数据库，只需要持久化类就可以完成增删改查的功能，使软件开发真正面向对象。据称使用Hibernate比JDBC方式减少了80%的编程量。

Hibernate框架

- Hibernate是传统Java对象和数据库服务器之间的桥梁，用来处理基于O/R映射机制和模式的那些对象。

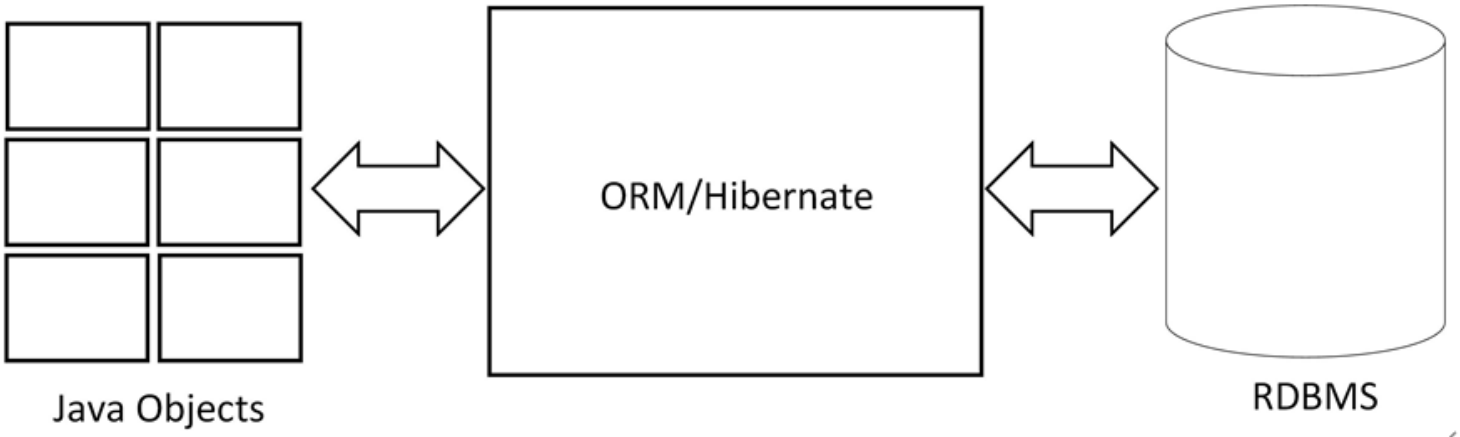


图 7-6 Hibernate 作用示意图

Hibernate框架

- Hibernate接口
- Hibernate使用不同的现存Java API，比如JDBC，Java事务API（JTA），以及Java命名和目录界面（JNDI）。

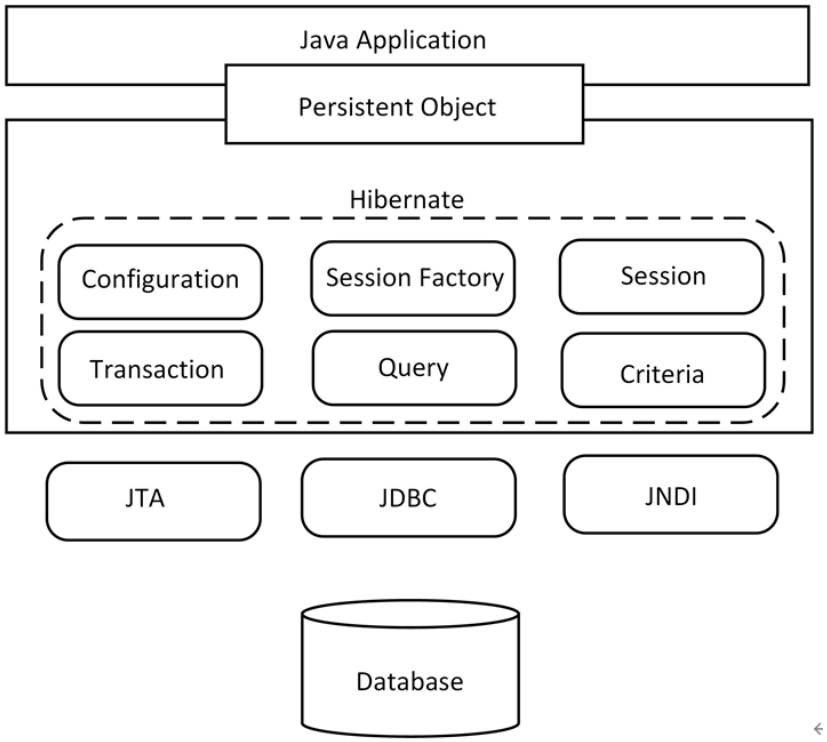


图 7-7 Hibernate 应用程序体系结构视图

Hibernate框架

- Hibernate核心类和接口
- 1) Session
  - Session接口负责执行被持久化对象的CRUD操作（CRUD的任务是完成与数据库的交流，包含了很多常见的SQL语句）。
  - 需要注意的是Session对象是非线程安全的。
  - Hibernate的session不同于JSP应用中的HttpSession。这里当使用session这个术语时，其实指的是Hibernate中的session，而以后会将HttpSession对象称为用户session。

Hibernate框架

- Hibernate核心类和接口
- 2) SessionFactory
  - SessionFactory接口负责初始化Hibernate。它充当数据存储源的代理，并负责创建Session对象。这里用到了工厂模式。
  - 需要注意的是SessionFactory并不是轻量级的，因为一般情况下，一个项目通常只需要一个SessionFactory就够，当需要操作多个数据库时，可以为每个数据库指定一个SessionFactory。

Hibernate框架

- Hibernate核心类和接口
- 3) Transaction
  - Transaction 接口是一个可选的API，可以选择不使用这个接口，取而代之的是Hibernate 的设计者自己写的底层事务处理代码。
  - Transaction 接口是对实际事务实现的一个抽象，这些实现包括JDBC的事务、JTA 中的UserTransaction、甚至可以是CORBA 事务。
  - 之所以这样设计是能让开发者能够使用一个统一事务的操作界面，使得自己的项目可以在不同的环境和容器之间方便地移植。

Hibernate框架

- Hibernate核心类和接口
- 4) Query
  - Query接口让你方便地对数据库及持久对象进行查询，它可以有两种表达方式：HQL语言或本地数据库的SQL语句。
  - Query经常被用来绑定查询参数、限制查询记录数量，并最终执行查询操作。
- 5) Criteria
  - Criteria接口与Query接口非常类似，允许创建并执行面向对象的标准化查询。值得注意的是Criteria接口也是轻量级的，它不能在Session之外使用。

Hibernate框架

- Hibernate核心类和接口
- 6) Configuration
  - Configuration 类的作用是对Hibernate 进行配置，以及对它进行启动。
  - 在Hibernate 的启动过程中，Configuration 类的实例首先定位映射文档的位置，读取这些配置，然后创建一个SessionFactory对象。
  - Configuration是启动hibernate 时所遇到的第一个对象。

Hibernate框架

- Hibernate的优势
- Hibernate实现了OR映射，同时提供访问数据的CRUD方法，代替手工书写SQL语句的繁琐，降低拼写错误。这样在很大程度上减少了开发过程人工使用SQL和JDBC处理数据的时间。

- 此外Hibernate对JDBC访问数据库的代码进行了轻量级封装，大大简化了数据访问层繁琐的重复性代码，并且减少了内存消耗，加快了运行效率。

Hibernate框架

- Hibernate的优势
- Hibernate还具有可扩展性强的特点，由于源代码的开源以及API的开放，当本身功能不够用时，可以自行编码进行扩展。

- 此外，Hibernate采用了POJO对象，代码没有侵入性，移植性比较好。

大纲

- 开放数据库连接
- 对象-关系映射ORM
- JPA持久化框架
  - ✓ JPA的概念
  - ✓ JPA持久化对象
  - ✓ JPA的优势
  - ✓ JPA编程范例
  - ✓ JPA与Hibernate的关系
- 其他持久化框架
- 小结

JPA的概念

- Java Persistence API（JPA）是Java EE 5平台上的标准的对象-关系映射和持久管理接口。

- 作为EJB 3规范成果的一部分，它得到了所有主要的Java供应商的支持。JPA的一个主要特性是任何的持久性提供程序都可以在上面做插拔。

- JPA是EJB 3规范的组成部分，代替了实体bean。它是基于POJO标准的ORM持久化模型，为POJO提供持久化标准规范，可以在Web和桌面应用中使用。

JPA的概念

- JPA包括以下3方面的技术：
- 1) ORM映射元数据
  - JPA支持XML和JDK5.0注解两种元数据的形式，元数据描述对象和表之间的映射关系，框架据此将实体对象持久化到数据库表中。
- 2) 调用接口API
  - 用来操作实体对象，执行CRUD操作，框架在后台完成所有的事情，开发者从繁琐的JDBC和SQL代码中解脱出来。
- 3) 查询语言
  - 这是持久化操作中很重要的一个方面，通过面向对象而非面向数据库的查询语言查询数据，避免程序的SQL语句紧密耦合。

大纲

- 开放数据库连接
- 对象-关系映射ORM
- JPA持久化框架
  - ✓ JPA的概念
  - ✓ JPA持久化对象
  - ✓ JPA的优势
  - ✓ JPA编程范例
  - ✓ JPA与Hibernate的关系
- 其他持久化框架
- 小结

JPA持久化对象

- 利用JPA进行持久化对象，主要包括以下几个步骤：
  - ✓ 创建persistence.xml，在这个文件中配置持久化单元(Hibernate中的hibernate.cfg.xml)。通过该文件，指定跟哪个数据库进行交互，指定JPA使用哪个持久化的框架；
  - ✓ 创建实体管理器的工厂EntityManagerFactory（类似于Hibernate中的SessionFactory）；
  - ✓ 创建实体管理器EntityManager(类似于Hibernate中的Session)；
  - ✓ 创建实体类，使用注解annotation描述实体类跟数据库表之间的一一映射关系；
  - ✓ 使用JPA API完成数据增加、删除、修改和查询操作；
  - ✓ 使用结束后，关闭EntityManager。

JPA持久化对象

- 实体管理器（Entity Manager）用于管理系统中的实体，它是实体与数据库之间的桥梁。通过调用实体管理器的相关方法可以把实体持久化到数据库中，同时也可以把数据库中的记录打包成实体对象。
- 实体生命周期

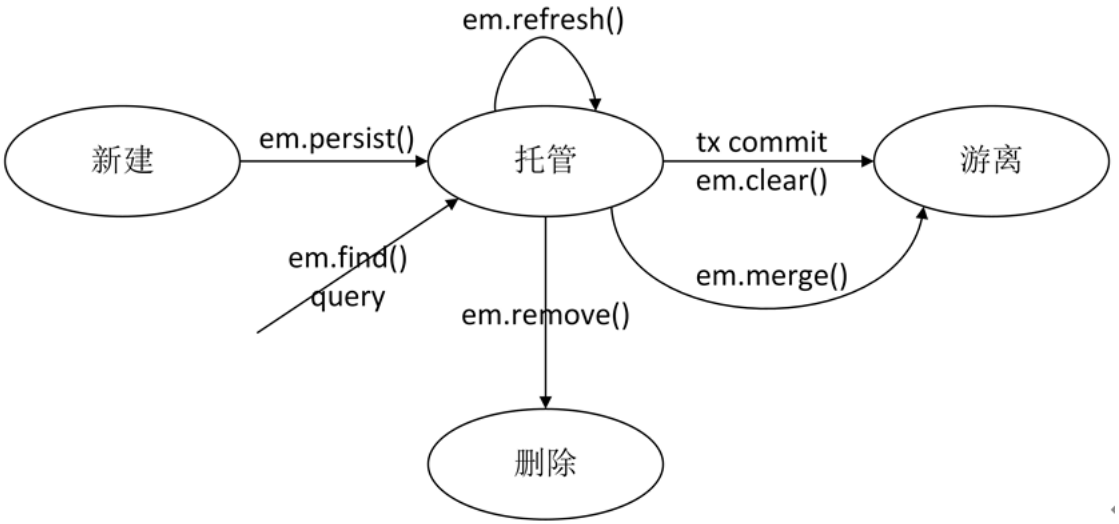


图 7-8 JPA 实体生命周期状态转换图

JPA持久化对象

- 实体生命周期的四种状态
- 1) 新建状态（New）：
  - ✓ 对象在保存进数据库之前为临时状态。此时数据库中没有该对象的信息，该对象的ID属性也为空。如果没有被持久化，程序退出时临时状态的对象信息将丢失。
- 2) 托管状态（Managed）：
  - ✓ 对象在保存进数据库后或者从数据库中加载后、并且没有脱离Session时为持久化状态。这时候数据库中有对象的信息，该对象的id为数据库中对应记录的主键值。由于还在 Session 中，持久化状态的对象可以执行任何有关数据库的操作，例如获取集合属性的值等。

JPA持久化对象

- 实体生命周期的四种状态
- 3) 游离状态（Detached）：
  - ✓ 是对象曾经处于持久化状态、但是现在已经离开Session了。虽然分离状态的对象有id值，有对应的数据库记录，但是已经无法执行有关数据库的操作。例如，读取延迟加载的集合属性，可能会抛出延迟加载异常。
- 4) 删除状态（Removed）：
  - ✓ 删除的对象，有id值，尚且和 Persistence Context 有关联，但是已经准备好从数据库中删除。

JPA持久化对象

- 创建实体管理器
- 所有实体管理器都来自类型javax.persistence.EntityManagerFactory的工厂。
  - ✓ 以下示例演示为名为"EmployeeService"的持久性单元创建一个EntityManagerFactory：

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("EmployeeService");
```

  - ✓ 通过工厂创建实体管理器：

```
EntityManager em = emf.createEntityManager();
```

JPA持久化对象

- 保存实体
- 通过使用实体管理器，可以持久化实例，如下面的例子，通过前面创建的实体管理器em来持久化Employee类的实例：

```
Employee emp = new Employee(15);
em.persist(emp);
```

- 查找实体
- 如果实体在数据库中，下一行代码显示如何找到它：

```
Employee emp = em.find(Employee.class, 15);
```

JPA持久化对象

- 删除实体
- 要从数据库中删除实体，可以调用EntityManager的remove方法：

```
Employee emp = em.find(Employee.class, 15);
em.remove(emp);
```

- 更新实体
- 要更新实体，可以在被管实体上调用setter方法。被管实体是从EntityManager返回的实体。

```
Employee emp = em.find(Employee.class, 15);
emp.setName("new Name");
```

JPA持久化对象

- 事务
- 以下代码显示如何启动和提交事务

```
em.getTransaction().begin();
Employee emp = new Employee(15);
em.persist(emp);
em.getTransaction().commit();
```

大纲

- 开放数据库连接
- 对象-关系映射ORM
- JPA持久化框架
  - ✓ JPA的概念
  - ✓ JPA持久化对象
  - ✓ JPA的优势
  - ✓ JPA编程范例
  - ✓ JPA与Hibernate的关系
- 其他持久化框架
- 小结

JPA的优势

- 1) 标准化
  - JPA 是 JCP 组织发布的 Java EE 标准之一，因此任何声称符合 JPA 标准的框架都遵循同样的架构，提供相同的访问API，这保证了基于JPA开发的企业应用能够经过少量的修改就能够在不同的JPA框架下运行。

- 2) 容器级特性的支持
  - JPA框架中支持大数据集、事务、并发等容器级事务，这使得 JPA 超越了简单持久化框架的局限，在企业应用发挥更大的作用。

JPA的优势

- 3) 简单方便
  - ✓ JPA的主要目标之一就是提供更加简单的编程模型：在JPA框架下创建实体和创建Java 类一样简单，没有任何的约束和限制，只需要使用 javax.persistence.Entity进行注释。
  - ✓ JPA的框架和接口也都非常简单，没有太多特别的规则和设计模式的要求，开发者可以很容易的掌握。
  - ✓ JPA基于非侵入式原则设计，因此可以很容易的和其它框架或者容器集成。

JPA的优势

- 4) 查询能力
  - ✓ JPA的查询语言是面向对象而非面向数据库的，它以面向对象的自然语法构造查询语句，可以看成是Hibernate HQL的等价物。
  - ✓ JPA定义了独特的JPQL（Java Persistence Query Language），JPQL是EJB QL的一种扩展，它是针对实体的一种查询语言，操作对象是实体，而不是关系数据库的表，而且能够支持批量更新和修改、JOIN、GROUP BY、HAVING 等通常只有 SQL 才能够提供的高级查询特性，甚至还能够支持子查询。

JPA的优势

- 5) 高级特性
  - JPA 中能够支持面向对象的高级特性，如类之间的继承、多态和类之间的复杂关系，这样的支持能够让开发者最大限度的使用面向对象的模型设计企业应用，而不需要自行处理这些特性在关系数据库的持久化。

大纲

- 开放数据库连接
- 对象-关系映射ORM
- JPA持久化框架
  - ✓ JPA的概念
  - ✓ JPA持久化对象
  - ✓ JPA的优势
  - ✓ JPA编程范例
  - ✓ JPA与Hibernate的关系
- 其他持久化框架
- 小结

JPA编程范例

- 使用JPA进行数据持久化首先需要配置好persistant.xml文件，在配置文件中指定好持久化使用的框架、框架相关联的配置信息以及所连接的数据库的信息。
- 实际开发中在源码部分创建META-INF文件夹，文件夹下创建persitant.xml文件，并填写配置信息，下面是样例，例子中采用Hibernate作为持久化框架。

JPA编程范例

- 下面是样例，例子中采用Hibernate作为持久化框架

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence"
                           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                           xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
                           http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd ">
  <persistence-unit name="EmployeeService">
  <!--注意配置文件名称，要与后面代码中获取 EntityManagerFactory 的方法参数一致-->
    <!-- 四要素 org.hibernate.cfg.Environment-->
    <properties>
      <!-- 如果使用 Hibernate 实现的 JPA，使用的就是 Hibernate 的环境参数 -->
      <property name="hibernate.connection.driver_class" value="com.mysql.jdbc.Driver" />

      <property name="hibernate.connection.url"
                value="jdbc:mysql://localhost:3306/mytest" />
      <!--指定数据库链接-->

      <property name="hibernate.connection.username" value="root" />

      <property name="hibernate.connection.password" value="" />
      <!--填写数据库访问的用户信息-->
```

JPA编程范例

```
↵
        <!--可选配置-->↵
        <!--控制台打印 sql 语句-->↵
        <property name="hibernate.show_sql" value="true" />↵
↵
        <!-- 格式化输出 SQL -->↵
        <property name="hibernate.format_sql" value="true" />↵
↵
    </properties>↵
↵
  </persistence-unit>↵
↵
</persistence>↵
↵
```

JPA编程范例

- 以下代码显示了一个简单的完全功能类，可用于对Employee实体发出典型的创建，读取，更新和删除（CRUD）操作。

```
public class EmployeeService {//EmployeeService 封装 Employee 对象与数据库的关联操作
    protected EntityManager em;
    public EmployeeService(EntityManager em) {
        this.em = em;//EnityManager 对象提供 Hibernate 的服务
    }
    public Employee createEmployee(int id, String name, long salary) {
        Employee emp = new Employee(id);//Employee 类是 ORM 映射中的实体类
        emp.setName(name);
        emp.setSalary(salary);
        em.persist(emp);
        return emp;
    }

    public void removeEmployee(int id) {
        Employee emp = findEmployee(id);
        if (emp != null) {
            em.remove(emp);
        }
    }
```

JPA编程范例

```
    public Employee raiseEmployeeSalary(int id, long raise) {
        Employee emp = em.find(Employee.class, id);
        if (emp != null) {
            emp.setSalary(emp.getSalary() + raise);
        }
        return emp;
    }

    public Employee findEmployee(int id) {
        return em.find(Employee.class, id);
    }

    public List<Employee> findAllEmployees() {
        TypedQuery<Employee> query = em.createQuery("SELECT e FROM Employee e",
                                                    Employee.class);
        return query.getResultList();
    }
}
```

JPA编程范例

- 主类：

```
public class Main {
    public static void main(String[] args) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("EmployeeService");
        //以 JPA 配置文件 persistence.xml 中的持久化单元名为参数
        EntityManager em = emf.createEntityManager();//通过 factory 获取到管理实例对象
        EmployeeService service = new EmployeeService(em);
        em.getTransaction().begin();//开启数据库事务
        Employee emp = service.createEmployee(1, "Tom", 5000);
        em.getTransaction().commit();//事务提交
        System.out.println("Persisted " + emp);

        emp = service.findEmployee(1);
        System.out.println("Found " + emp);

        List<Employee> emps = service.findAllEmployees();
        for (Employee e : emps)
            System.out.println("Found employee: " + e);
        em.getTransaction().begin();
        emp = service.raiseEmployeeSalary(1, 1000);
        em.getTransaction().commit();
        System.out.println("Updated " + emp);

        em.getTransaction().begin();
        service.removeEmployee(15);
        em.getTransaction().commit();
        System.out.println("Removed Employee 15");

        em.close();
        emf.close();//使用完毕，释放资源
    }
}
```

大纲

- 开放数据库连接
- 对象-关系映射ORM
- JPA持久化框架
  - ✓ JPA的概念
  - ✓ JPA持久化对象
  - ✓ JPA的优势
  - ✓ JPA编程范例
  - ✓ JPA与Hibernate的关系
- 其他持久化框架
- 小结

JPA与Hibernate的关系

- JPA需要供应商来实现其持久化的功能，而Hibernate可以看作是JPA中的一个优秀实现。

- 从功能上来说JPA是Hibernate功能的一个子集。但其实，在2006年J2EE 5.0标准正式发布以后，持久化框架标准JPA基本上是参考Hibernate实现的。

- 从Hibernate在3.2版本开始已经完全兼容JPA标准。Hibernate3.2获得了Sun TCK的 JPA（Java Persistence API）兼容认证。

JPA与Hibernate的关系

- Hibernate主要通过3个组件来实现JPA，包括hibernate-annotation、hibernate-entity manager和hibernate-core。
  - hibernate-annotation是Hibernate支持annotation方式配置的基础，它包括了标准的JPA annotation以及Hibernate自身特殊功能的annotation。
  - hibernate-core是Hibernate的核心实现，提供了Hibernate所有的核心功能。
  - hibernate-entitymanager实现了标准的JPA，可以把它看成hibernate-core和JPA之间的适配器。它并不直接提供ORM的功能，而是对hibernate-core进行封装，使得Hibernate符合JPA的规范。

JPA与Hibernate的关系

- 总的来说，JPA是规范，Hibernate是框架，JPA是持久化规范，而Hibernate实现了JPA。

大纲

- 开放数据库连接
- 对象-关系映射ORM
- JPA持久化框架
  - ✓ JPA的概念
  - ✓ JPA持久化对象
  - ✓ JPA的优势
  - ✓ JPA编程范例
  - ✓ JPA与Hibernate的关系
- 其他持久化框架
- 小结

其他持久化框架

- 除了Hibernate以外，各大厂商也推出其他的持久化框架，它们具有各自的特点和特性，能够适用于不同的场景。典型的对象-关系映射框架有myBatis，Nhibernate，OpenJPA，TopLink，等。

其他持久化框架

- MyBatis
- MyBatis前身是apache的一个开源项目iBatis。2010年这个项目由apache software foundation 迁移到了google code，并且改名为MyBatis。2013年11月迁移到Github。iBATIS一词来源于“internet”和“abatis”的组合，是一个基于Java的持久层框架。
- iBATIS提供的持久层框架包括SQL Maps和Data Access Objects（DAO）。MyBatis是支持普通SQL查询，存储过程和高级映射的优秀持久层框架，并且消除了几乎所有的JDBC代码和参数的手工设置以及结果集的检索。

其他持久化框架

- MyBatis
- MyBatis使用简单的XML或注解用于配置和原始映射，将接口和Java的POJOs（Plain Ordinary Java Objects，普通的Java对象）映射成数据库中的记录。
- 对于具体的数据操作，Hibernate会自动生成SQL语句，而Mybatis则要求开发者编写具体的SQL语句。
- 相对Hibernate等"全自动"的ORM机制而言，Mybatis以SQL开发的工作量和数据库移植性上的让步，为系统设计提供了更大的自由空间。

其他持久化框架

- NHibernate
- NHibernate 是一个基于.Net 的针对关系型数据库的对象持久化类库。
- Nhibernate 来源于非常优秀的基于Java的Hibernate关系型持久化工具。NHibernate 从数据库底层来持久化你的.Net 对象到关系型数据库。
- NHibernate 让开发者的代码仅仅和对象关联，NHibernat 自动产生 SQL 语句，并确保对象提交到正确的表和字段中去。

其他持久化框架

- NHibernate
- 特性：
- 1）Visual Studio 友好，Visual Studio 中轻松映射常规 C# 或 VB.NET 对象模型，不需要特殊的基类或属性，完全支持继承、组件和枚举。
- 2）快速开发周期，从域模型生成数据库表，支持所有流行的关系数据库，支持复杂的旧方案。
- 3）大量插件与工具，包括全文搜索、使用 Microsoft Velocity 和Memcached进行集群范围的缓存、业务验证规则、ReSharper插件等。

其他持久化框架

- OpenJPA
- OpenJPA 是 Apache 组织提供的开源项目，它实现了 EJB 3.0 中的 JPA 标准，为开发者提供功能强大、使用简单的持久化数据管理框架。
- OpenJPA 封装了和关系型数据库交互的操作，让开发者把注意力集中在编写业务逻辑上。
- OpenJPA 可以作为独立的持久层框架发挥作用，也可以轻松的与其它 Java EE 应用框架或者符合 EJB 3.0 标准的容器集成。

其他持久化框架

- OpenJPA
- 除了对 JPA 标准的支持之外，OpenJPA 还提供了非常多的特性和工具支持让企业应用开发变得更加简单，减少开发者的工作量，包括允许数据远程传输/离线处理、数据库/对象视图统一工具、使用缓存（Cache）提升企业应用效率等。

其他持久化框架

- TopLink
- TopLink原为WebGain公司的产品，现在被Oracle收购，并重新包装为Oracle AS TopLink。
- TopLink为在关系数据库表中存储Java对象和企业Java组件（EJB）提供高度灵活和高效的机制。
- TopLink几乎能够处理持久性方面的任何情况，可以消除在构造持久层时所涉及的内在风险。
- TopLink提供一个持久性基础架构，使开发人员能够将来自多种体系结构的数据（包括EJB（CMP和BMP）、常规的Java对象、servlet、JSP、会话bean和消息驱动bean）集成在一起。

其他持久化框架

- TopLink
- TopLink允许Java应用程序访问作为对象存储在关系数据库中的数据，从而极大地提高了开发人员的工作效率。
- TopLink还具有通过最大限度地降低数据库命中率和网络流量及利用由JDBC和数据库提供的最优化来提升应用程序性能的特性。

其他持久化框架

- TopLink
- TopLink是通过创建一个元数据"描述符"（映射）集来实现上述特性的，这些元数据描述符定义了以一个特定的数据库模式存储对象的方式。
- TopLink在运行时使用这些映射动态地生成所需的SQL语句。这些元数据描述符（映射）独立于语言和数据库，开发人员能够在无需对它们所表示的类进行重编译的情况下修改它们。

本章小结

- 数据访问中间件对业务开发人员屏蔽底层数据操作的繁琐细节，提供对多种数据源进行统一访问的接口。其作用于处理业务服务层和数据层之间的交互操作，将业务和复杂的数据访问操作隔离开。

- 本章首先介绍了数据访问应用程序接口，包括ODBC，OLE DB，ADO，并介绍了以Java的API形式提供服务的JDBC。然后介绍了对象-关系映射的概念和Hibernate框架，并用案例演示对象-关系映射的编程。再然后介绍JPA持久化框架的概念和持久化对象的过程，并阐述了具体框架与JPA之间的关系。最后，对其他的持久化框架进行了简要的介绍。

第4次作业

基于关系数据库，建立一个基于JPA和Hibernate（或mybatis）的数据存取服务，实现以下功能。

- (1)建立用户类和表User,包含UserId、UserName等信息。
- (2)建立新闻类News,包括ItemId、UserId、Title、Conent、PublisTime等信息。(3)用户发布新闻News,并存人数据库中。
- (4)用户对该新闻进行增加、修改和删除，并可根据关键字进行查询。
- (5)基于JPA建立News和User的join操作查询某个用户所发布的新闻。