中间件技术
Middleware Technology

第八章　数据存取中间件

赖永炫　教授/博士
厦门大学　软件工程系

大纲

- 开放数据库连接
  - ✓ ODBC
  - ✓ OLE DB
  - ✓ ADO
  - ✓ JDBC
- 对象关系映射ORM
  - ✓ Hibernate
  - ✓ MyBatis
- JPA介绍
  - ✓ 概念　编程
- 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的概念

- 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）。
  - ✓ 有了ODBC API，就不必为访问Sybase数据库专门写一个程序，为访问Oracle数据库又专门写一个程序，或为访问Informix数据库又编写另一个程序等等，程序员只需用ODBC API写一个程序就够了，它可向相应数据库发送结构化查询语言（SQL）调用。

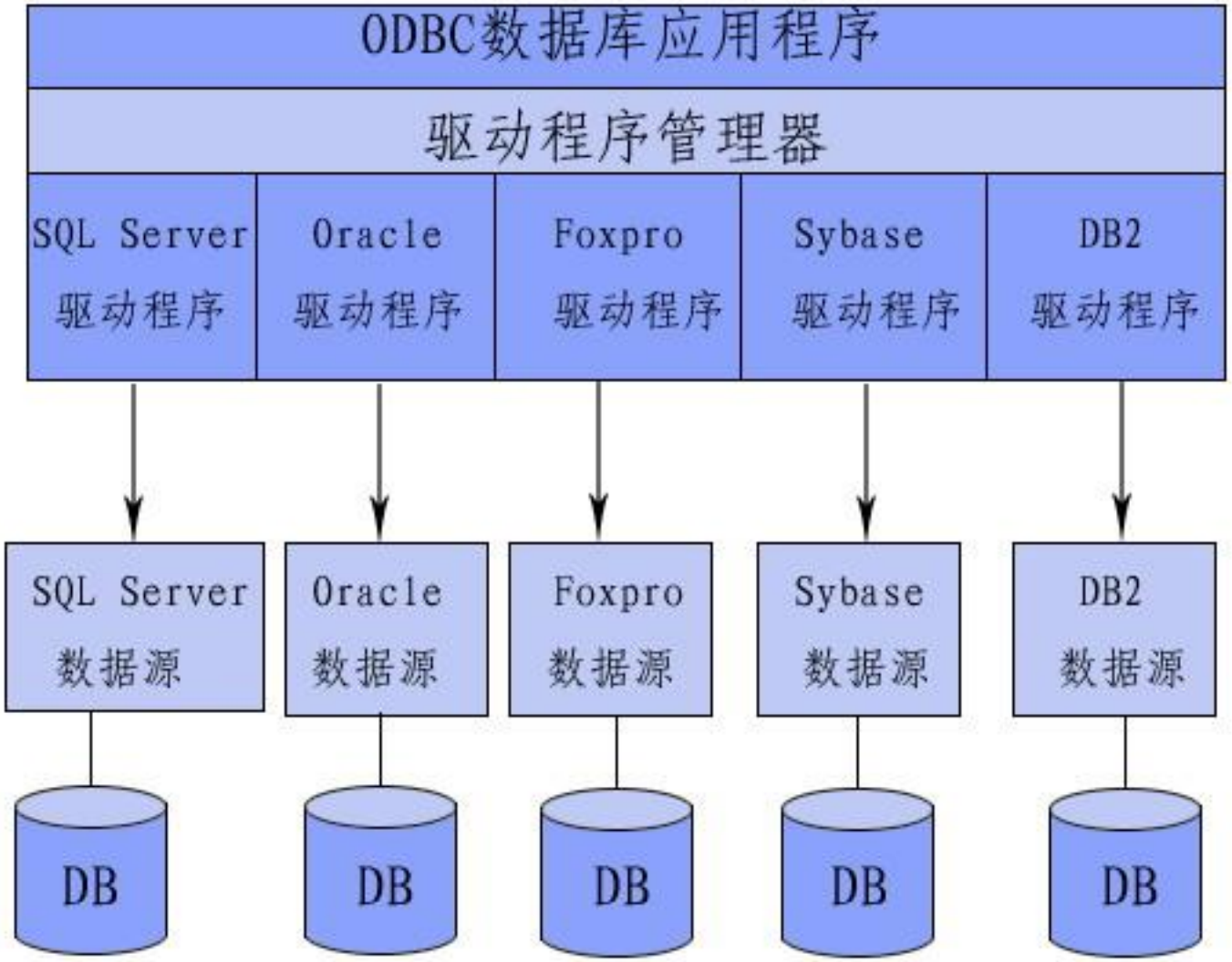
- ODBC提供了对SQL语言的支持，其API利用SQL来完成其大部分任务。
- 用户可以直接将SQL语句送给ODBC，通过ODBC的API应用程序可以存取保存在多种不同数据库管理系统（DBMS）中的数据，而不论每个DBMS使用了何种数据存储格式和编程接口。

ODBC的结构

- 包括四个主要部分：应用程序接口、驱动器管理器、数据库驱动器和数据源。
- 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。

ODBC的结构

- 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数据库应用程序
驱动程序管理器
SQL Server 驱动程序
Oracle 驱动程序
Foxpro 驱动程序
Sybase 驱动程序
DB2 驱动程序
SQL Server 数据源
Oracle 数据源
Foxpro 数据源
Sybase 数据源
DB2 数据源
DB
DB
DB
DB
DB

ODBC的体系结构

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的概念

- 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

- 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。

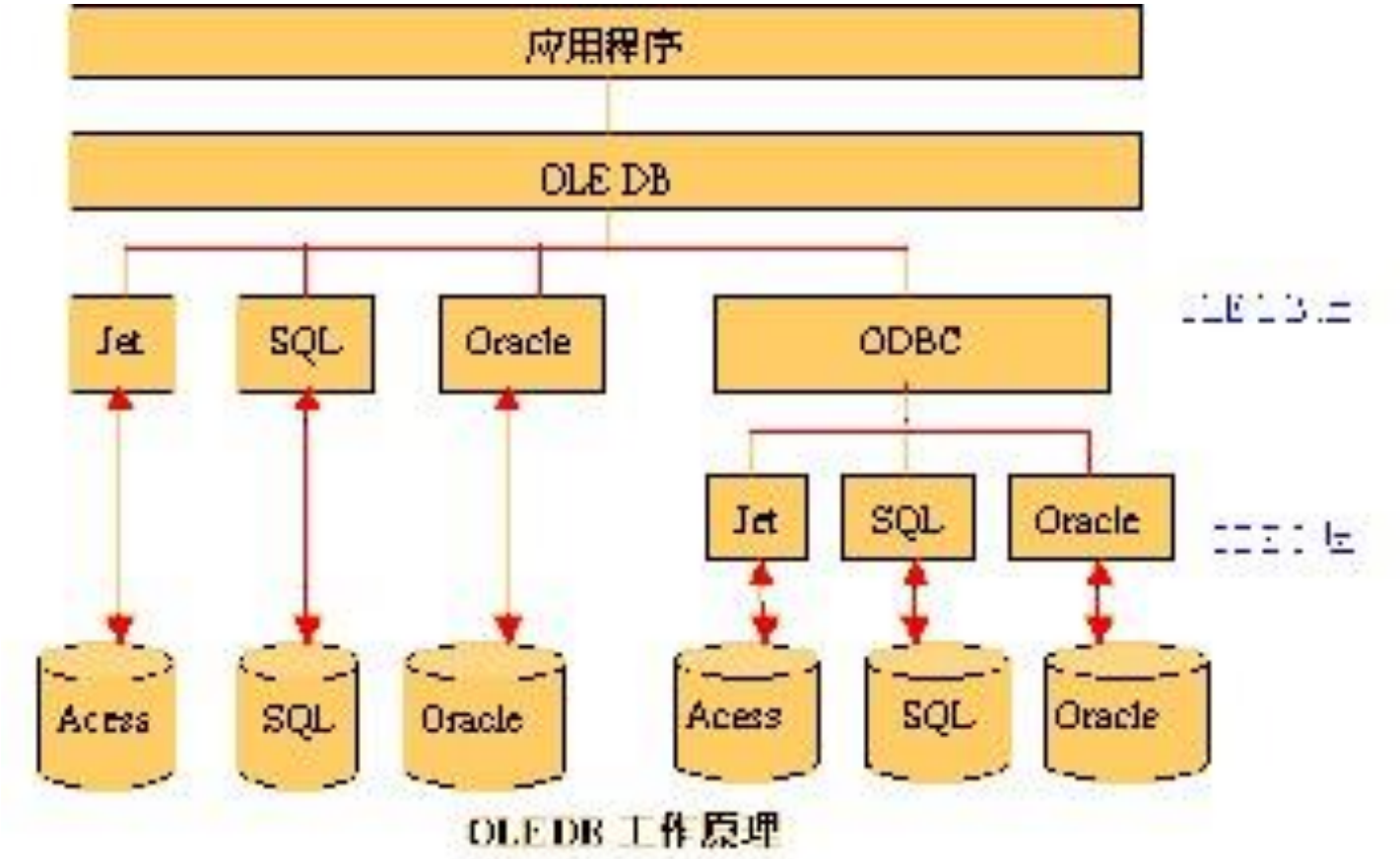
数据提供者

- 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。
- 数据提供者（Data Provider）：
  - ✓ 提供数据存储的软件组件，小到普通的文本文件、大到主机上的复杂数据库，或者电子邮件存储，都是数据提供者的例子。

数据提供者

- 有的文档把这些软件组件的开发商也称为数据提供者。
  - ✓ 如果程序要访问Access 数据库中的数据，必须用ADO.NET透过OLE DB 来开启，而OLE DB了解如何和许多种数据源作沟通。

OLE DB


应用程序
OLE DB
Jet
SQL
Oracle
ODBC
Jet
SQL
Oracle
Access
SQL
Oracle
Access
SQL
Oracle

OLE-DB 的评价

- 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。由于 ADO成功地封装了OLE-DB大部分的功能，并且大量简化了数据存取工作。

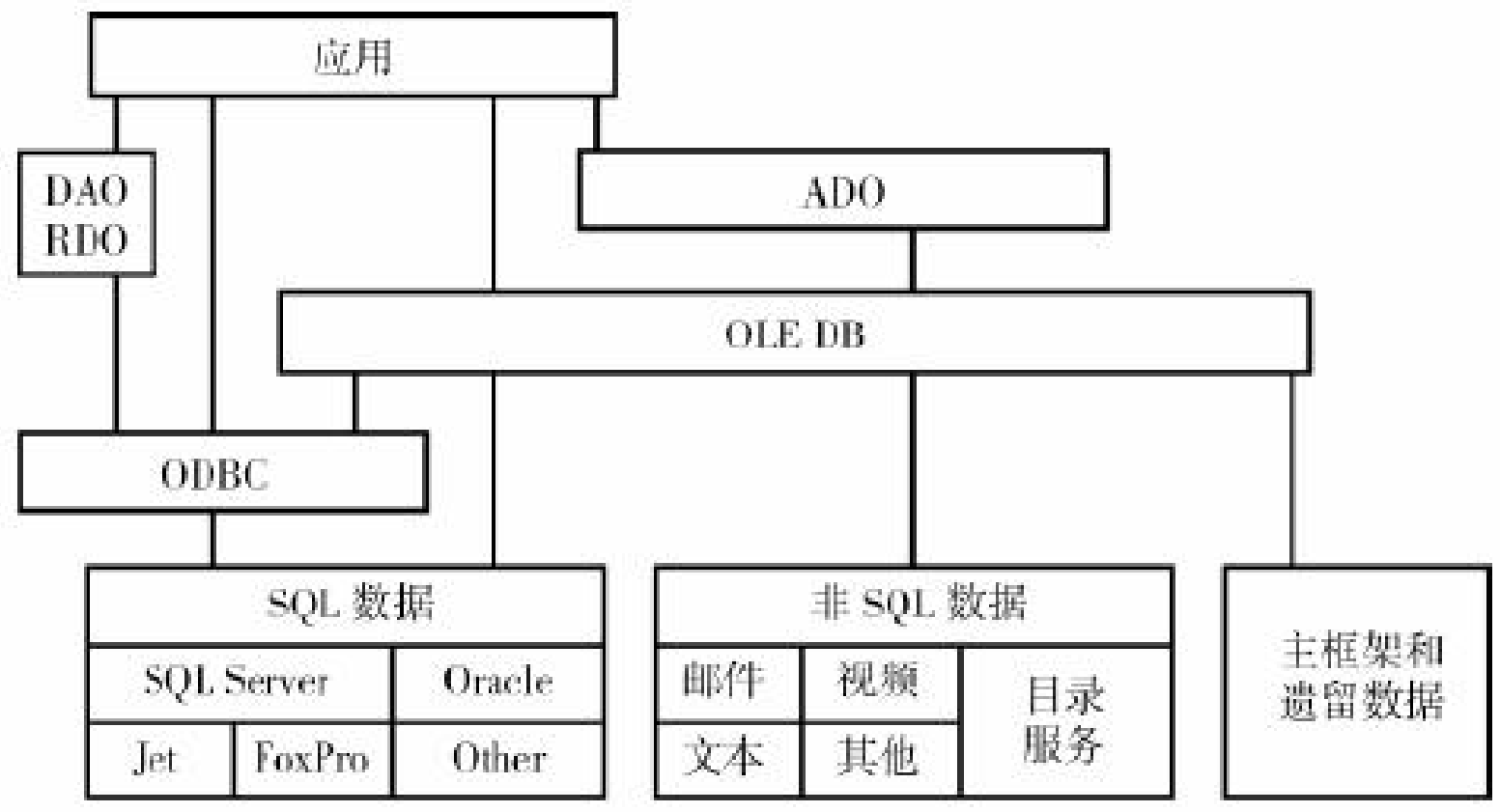
ADO

- 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。

ADO的概念

- 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。

ODBC、 OLE DB、ADO



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。

什么是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 vs ODBC

- 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的优势

- 相对ODBC来说，JDBC简易理解和使用，移植性也更好。
- JDBC包含了大部份基本数据操作功能，且面向对象的，完全遵循JAVA语言的优良特性。
- 程序员在短时间内即可了解JDBC驱动程序架构，容易上手。
- 采用JDBC数据库驱动程序只需选取适当的JDBC数据库驱动程序，无需求额外配置。

ORM：

当Relation 遇上Object Oriented

对象-关系映射ORM

- 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。

对象-关系映射

- 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。

ORM的好处

- 实际应用中，即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。

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

```
package com.bjpowernode.hibernate;
import java.util.Date;
 public class Student {
        private String id;
        private String name;

        public String getId() {
                return id;
        }
        public void setId(String id) {
                this.id = id;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
}
```

类和表之间的关联

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
        <class name="com.bjpowernode.hibernate.User">
                <id name="id">
                        <generator class="uuid"/>
                </id>
                <property name="name"/>
        </class>
</hibernate-mapping>
```

Hibernate

- 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J2EE架构中完成数据持久化的重任。

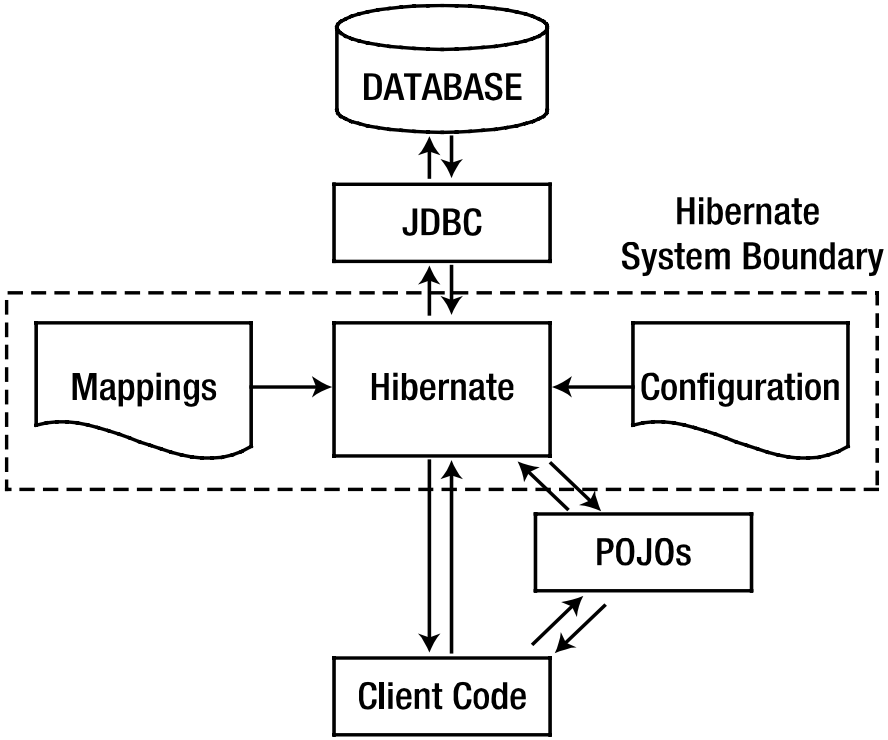


Figure 1-1. *The role of Hibernate in a Java application*

- 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做和一张数据库表的属性一一对应，当然也可以实现关系数据库的各种表件关联的对应。
  - ✓ 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的核心类和接口

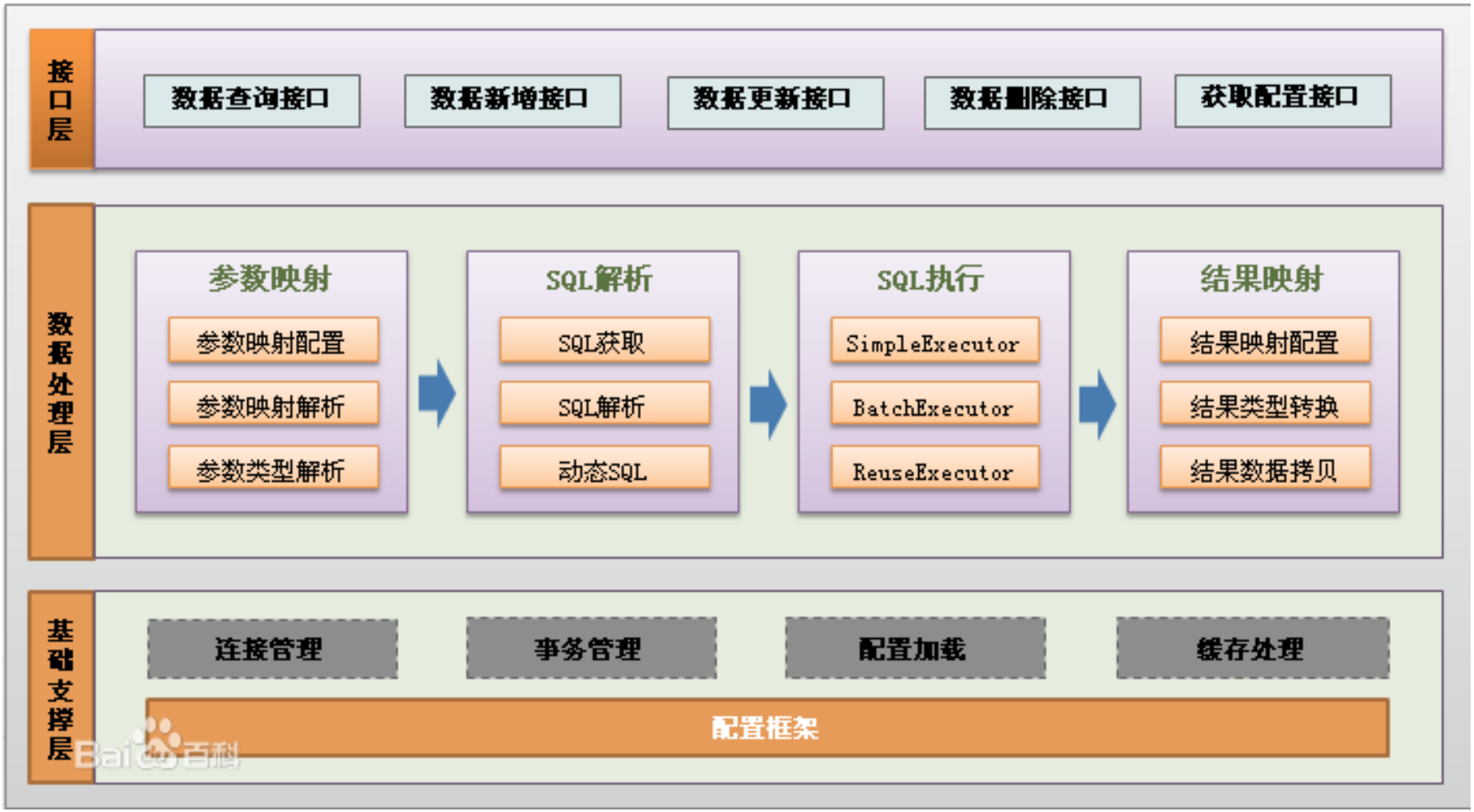
- 有6个：Session、SessionFactory、Transaction、Query、Criteria和Configuration。
  - ✓ 这6个核心类和接口在任何开发中都会用到。通过这些接口，不仅可以对持久化对象进行存取，还能够进行事务控制。

- Hibernate的具体内容，请参考附加课件

MyBatis

- 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。MyBatis 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。
- 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 Old Java Objects，普通的 Java对象）映射成数据库中的记录。

Mybatis功能架构



Mybatis的功能架构

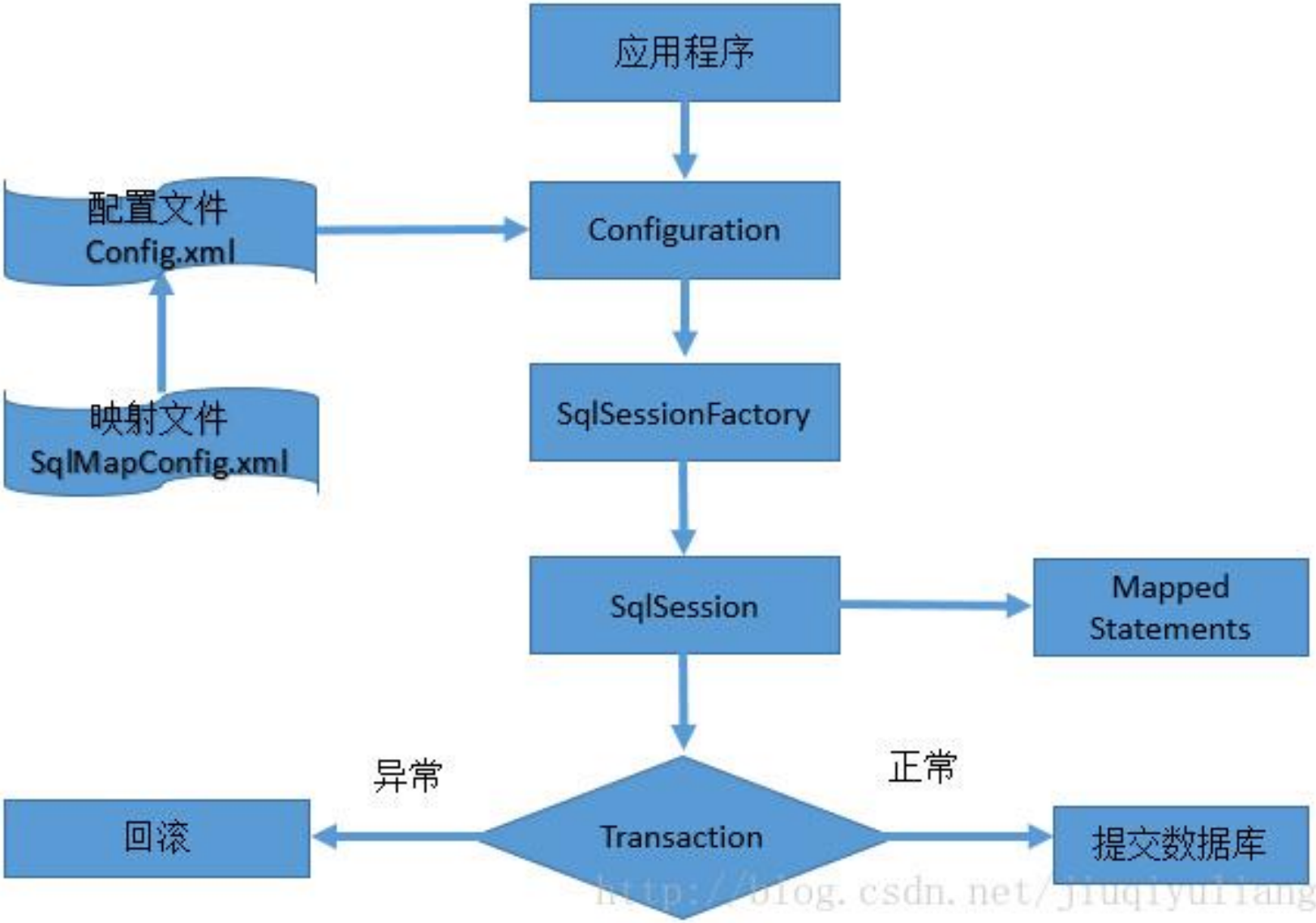
- (1)API接口层：提供给外部使用的接口API，开发人员通过这些本地API来操纵数据库。接口层一接收到调用请求就会调用数据处理层来完成具体的数据处理。
- (2)数据处理层：负责具体的SQL查找、SQL解析、SQL执行和执行结果映射处理等。它主要的目的是根据调用的请求完成一次数据库操作。
- (3)基础支撑层：负责最基础的功能支撑，包括连接管理、事务管理、配置加载和缓存处理，这些都是共用的东西，将他们抽取出来作为最基础的组件。为上层的数据处理层提供最基础的支撑。

MyBatis应用程序

- 每个MyBatis应用程序主要都是使用SqlSessionFactory实例的，一个SqlSessionFactory实例可以通过SqlSessionFactoryBuilder获得。SqlSessionFactoryBuilder可以从一个xml配置文件或者一个预定义的配置类的实例获得。

MyBatis应用程序

- 用xml文件构建SqlSessionFactory实例，在这个配置中使用类路径资源（classpath resource)。可以使用任何Reader实例，包括用文件路径或file://开头的url创建的实例。
- MyBatis实用类：Resources有很多方法，可以方便地从类路径及其它位置加载资源。


应用程序
配置文件
Config.xml
Configuration
映射文件
SqlMapConfig.xml
SqlSessionFactory
SqlSession
Mapped
Statements
异常
正常
回滚
Transaction
提交数据库

JPA 简介

主要内容

JPA概述

使用JPA的步骤

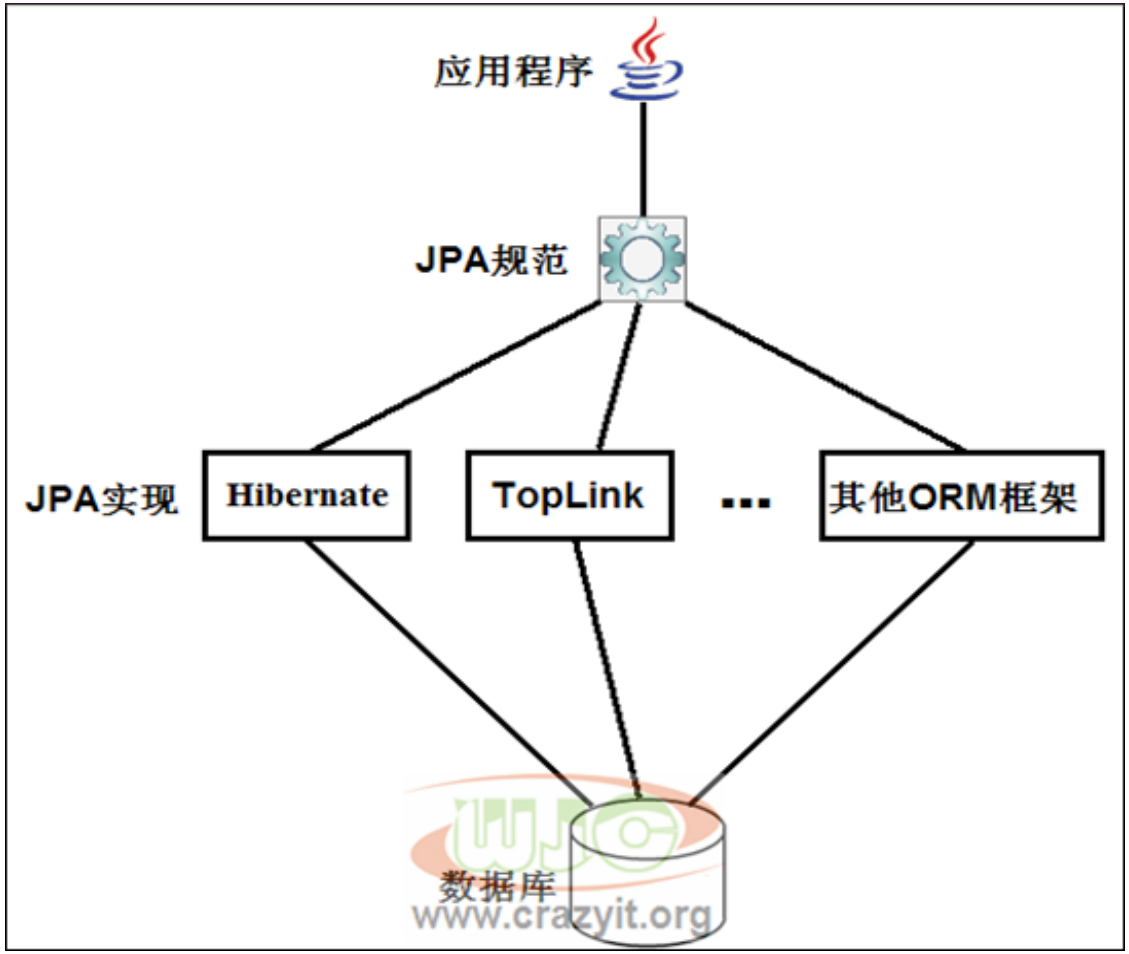
实体

JPA的API

JPQL

JPA 是什么

- Java Persistence API，用于对象持久化的API
- 是Java EE 5.0 平台标准的 ORM 规范，使得应用程序以统一的方式访问持久层

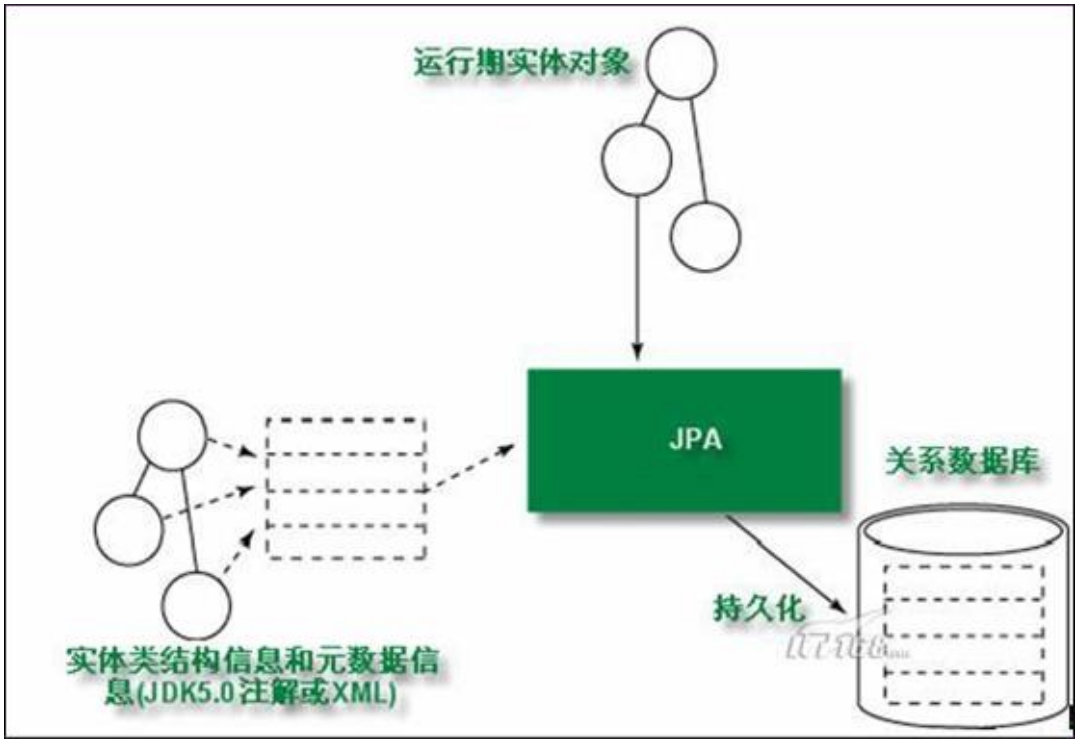


JPA 是什么

- JPA 是EJB3 Entity Bean，是一套从EJB3.0核心标准中分离出来的独立的标准文档
  - ✓ J2EE 4规范中最为人所熟悉的用来处理数据持久的Entity Bean，在Java EE5中被推到重来，取而代之的是java开发的通用持久化规范Java Persistence API 1.0，其实就是完全重新定义了的Entity Bean规范；
  - ✓ JPA作为java中负责关系数据持久化的组件已经完全独立出来成为一个单独的规范，而不再属于Enterprise Java Bean的范畴(EJB更多的是指Stateless/Stateful session bean和Message Driven Bean)。
  - ✓ 使用的 Java 的版本决定了实际是否可以应用 JPA。因为JPA 是 EJB 3.0 规范的一部分，而EJB 3.0 规范是 Java EE 5 版本的一部分。如果您未更新到 Java EE 5，则无法使用 JPA。

JPA 是什么

- 提供了以pojo编程模型为持久化对象的机制：通过 JDK 5.0 注解或 XML 描述对象－关系表的映射关系，并将运行期的实体对象持久化到数据库中，下图描述了 JPA 的结构：



JPA的产生

Sun引入新的 JPA ORM规范出于两个原因：

- 简化现有Java EE和 Java SE应用的对象持久化的开发工作：
  - ✓ 和实体Bean相比，不需要配置文件，提供了更好的开发体验
  - ✓ 不同于 EJB 3.0，JPA并不是J2EE环境专用，在J2SE的环境中也可以使用（不依赖J2EE容器，是java中的通用API）
- 整合ORM技术，统一各种ORM框架的规范，实现天下归一。

  （目前Hibernate 3.2、TopLink 10.1.3以及OpenJpa都提供了JPA的实现）

为什么要使用JPA

- Hibernate的DAO层代码：

```
public class FwxxDaoHibImpl
                    extends BaseHibernateDAO
                    implements FwxxDao {

   public FWXX getFwxxDetail(int fwid) {
        super.get(FWXX.class, fwid);
   }
   …
}
```

代码精简易读

开发工作量小，可以将精力集中在业务逻辑的处理上

JPA就是用来解决这些问题的

遗憾：Hibernate是一种O/R映射框架，Java EE需要一个O/R映射规范

JPA和Hibernate的关系

- JPA是hibernate的一个抽象或者可以理解为接口（就像JDBC和JDBC驱动的关系）：
  - ✓ JPA是规范：JPA本质上就是一种ORM规范，不是ORM框架——因为JPA并未提供ORM实现，它只是制订了一些规范，提供了一些编程的API接口，但具体实现则由用用服务器厂商来提供实现
  - ✓ Hibernate 是实现：Hibernate除了作为ORM框架之外，它也是一种JPA实现
- 从功能上来说， JPA现在就是Hibernate功能的一个子集

JPA的供应商

JPA 的目标之一是制定一个可以由很多供应商实现的API，目前Hibernate 3.2、TopLink 10.1.3以及OpenJpa都提供了JPA的实现

- Hibernate
  - ✓ JPA的始作俑者就是Hibernate的作者
  - ✓ Hibernate 从3.2开始，就开始兼容JPA
- OpenJPA
  - ✓ OpenJPA 是 Apache 组织提供的开源项目
- TopLink
  - ✓ TopLink以前需要收费，如今开源了；
  - ✓ OpenJPA虽然免费，但功能、性能、普及性等方面更加需要加大力度。

JPA的实现

- JPA做为Java EE5里面的新成员,跟jdbc一样就是一接口,具体实现由服务器实现
- 你选择什么服务器就基本上决定了用哪个JPA实现
  - ✓Jboss/Wildfly就是用hibernate去实现的
  - ✓Weblogic是用OpenJPA
  - ✓Oracle是用TopLink

JPA的优势

- 标准化

  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。越来越多的提供商期待在不久的将来提供 JPA 实施。

- 对容器级特性的支持

  JPA 框架中支持大数据集、事务、并发等容器级事务

JPA的优势

- 简单易用，集成方便

  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，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，

- 可媲美JDBC的查询能力

  JPA的查询语言是面向对象的，JPA定义了独特的JPQL，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。

- 支持面向对象的高级特性

  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，最大限度的使用面向对象的模型

JPA的不足

- JPA是一个规范而不是一个产品。
  需要提供商提供一个实施，才能获得这些基于标准的API 的优势。
- JPA是Hibernate、TopLink，JDO等ORM框架真子集，只提供其中最好的功能，如果应用程序中需要的功能，但是规范中没有提供的功能，则使用供应商特有的API，移植起来比较麻烦，所以要尽可能地使用JPA API。

JPA包括 3方面的技术

- ORM映射元数据：JPA支持XML和 JDK 5.0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。

- JPA的API：用来操作实体对象，执行CRUD操作，框架在后台替我们完成所有的事情，开发者从繁琐的JDBC和 SQL代码中解脱出来。

- 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。

主要内容

JPA概述

使用JPA的步骤

实体

JPA的API

JPQL

使用JPA持久化对象的步骤

创建persistence.xml,在这个文件中配置持久化单元(Hibernate中的hibernate.cfg.xml);

需要指定跟哪个数据库进行交互;

需要指定JPA使用哪个持久化的框架;(因为他本身没有持久化能力);

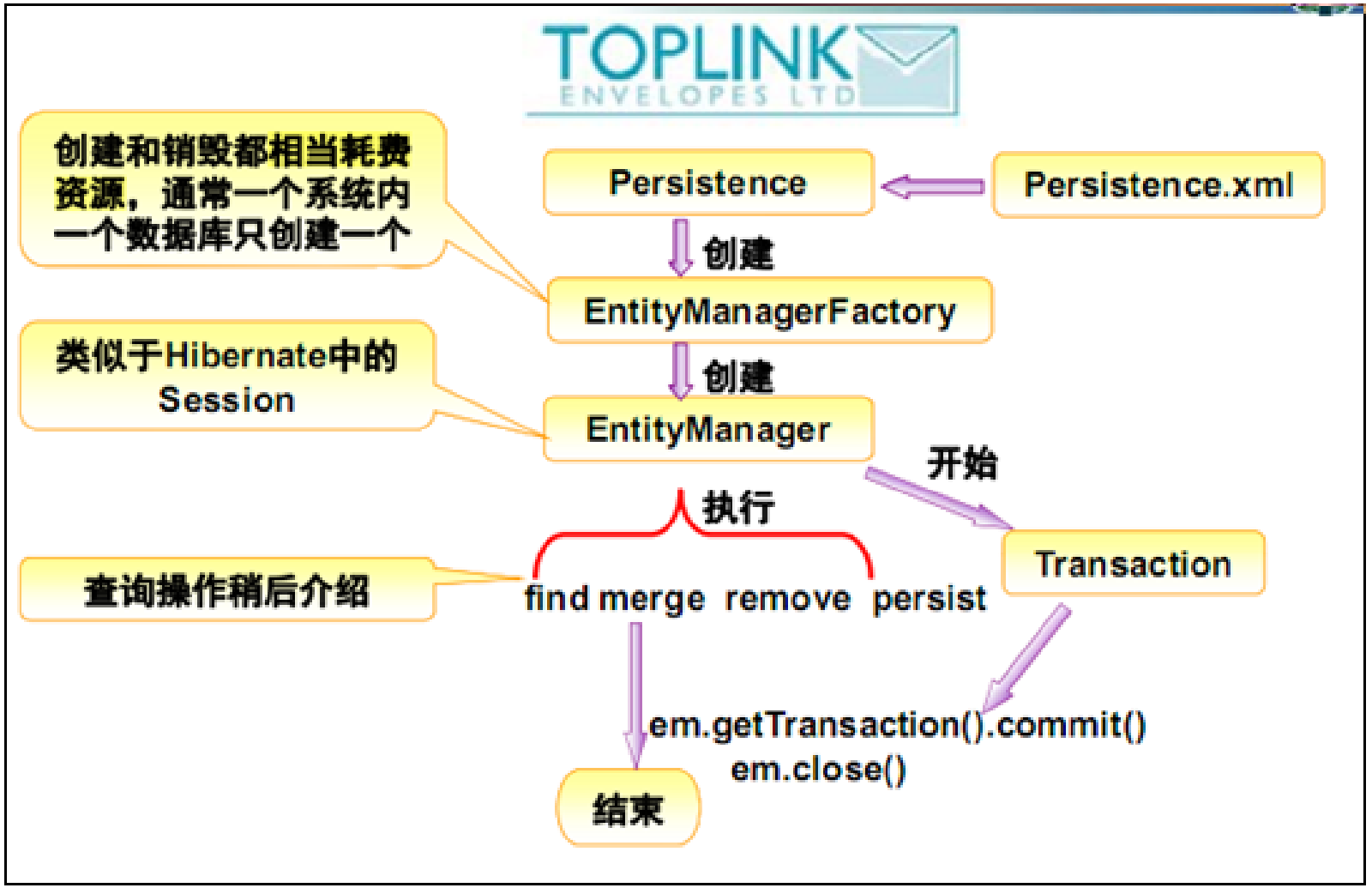
创建EntityManagerFactory(Hibernate中的SessionFactory);

创建EntityManager(实体管理器)(Hibernate中的Session);

创建实体类,使用annotation来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

JPA的执行过程


TOPLINK
ENVELOPES LTD
创建和销毁都相当耗费资源，通常一个系统内一个数据库只创建一个
Persistence
Persistence.xml
创建
EntityManagerFactory
类似于Hibernate中的Session
创建
EntityManager
开始
执行
Transaction
查询操作稍后介绍
find merge remove persist
em.getTransaction().commit()
em.close()
结束

persistence.xml

persistence.xml

```
<persistence-unit name="AddressBookPU" transaction-type="RESOURCE_LOCAL">
  <provider>oracle.toplink.essentials.PersistenceProvider</provider>
  <class>addressbook.Address</class>
  <class>addressbook.Person</class>
  <properties>
    <property name="toplink.jdbc.user" value="sa"/>
    <property name="toplink.jdbc.password" value="sa"/>
    <property name="toplink.jdbc.url" value="jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=zf"/>
    <property name="toplink.jdbc.driver" value="com.microsoft.jdbc.sqlserver.SQLServerDriver"/>
    <property name="toplink.ddl-generation" value="drop-and-create-tables"/>
...
```

持久化单元名称

Provider名称

需要持久化的类，可以有多个

配置数据库链接

数据库驱动

运行时先删除，再创建表

持久化单元

持久化单元： persistence-unit

数据库相关的信息

持久化提供者信息（Hibernate、Toplink…）

厂商的一些客户化属性

<property name="hibernate.show_sql"
value="true"/>

其他可选的一些元数据

如果使用不同的持久化提供程序，那么需要指定提供程序类

附加的ORM映射文件 附加的ORM映射文件

附加实体的JAR文件

一个持久化单元可以创建一个EntityManagerFactory

持久化提供者

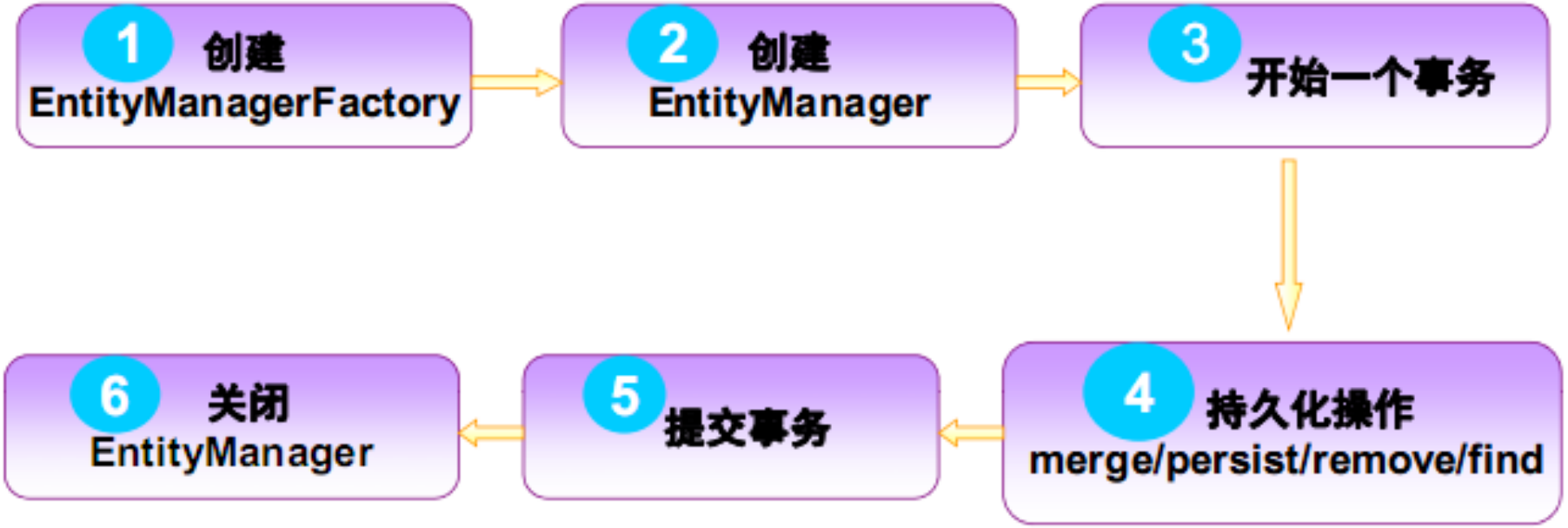
Persistence Provider：持久化提供者

JPA只是一套标准的持久化API，它需要通过一个ORM框架,才能进行持久化，我们把这个能和JPA集成起来的ORM框架称为持久化的提供者;

每一个支持ebj3.0 JPA的ORM框架必须提供一个实现javax.persistence.spi.PersistenceProvider接口的实现类，通过它创建EntityManager对象

使用JPA实现增加、删除、修改和查询操作

- 使用JPA的6个步骤：



EntityManagerFactory

实体管理器的工厂，类似于Hibernate 中的SessionFactory

通过持久化提供者Persistence Provider创建这个对象应用程序管理

用来创建EntityManager实例

一个数据库创建一个EntityManagerFactory对象

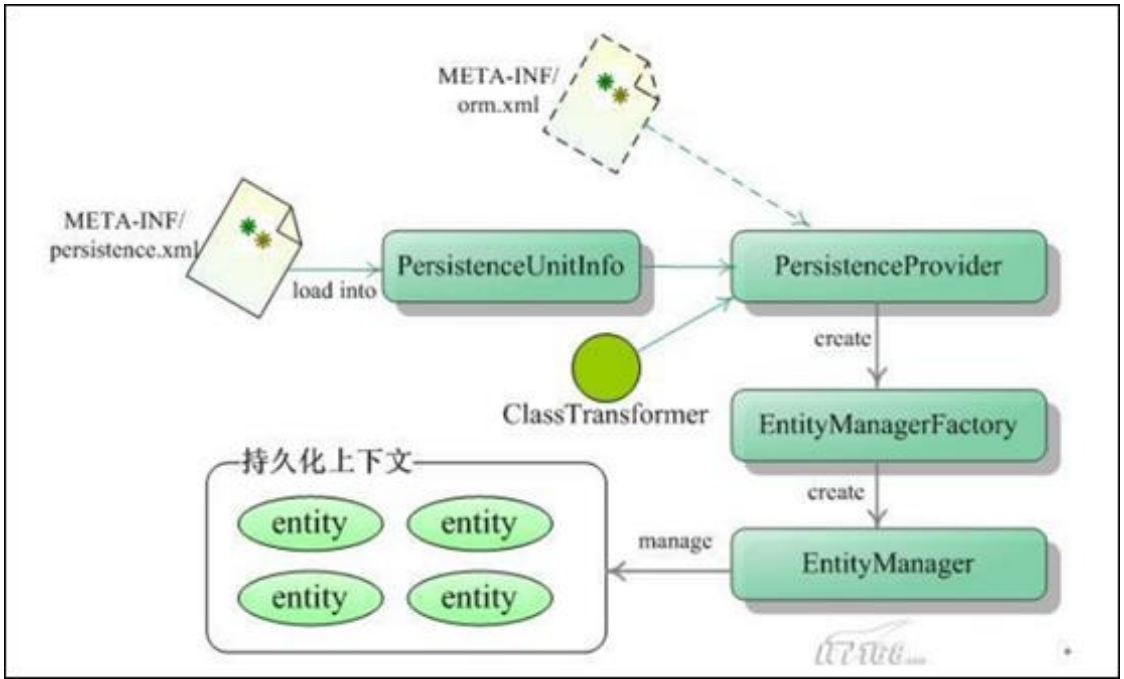
EntityManager

实体管理器，类似于Hibernate的Session

由EntityManagerFacotry创建

用来访问持久化上下文中实体对象的一个接口，管理实体对象与底层数据源之间进行O/R映射(增删改查)

一个线程一个EntityManager对象，



EntityManager

获取EntityManager对象的方式

1/应用程序管理EntityManager

应用程序通过javax.persistence.EntityManagerFactory的createEntityManager创建EntityManager实例

2/容器管理EntityManager

通过依赖注入获取

```
@Stateless
public class HRAgentBean implements HRAgentRemote{
    @PersistenceContext(unitName="salesemployee")
    private EntityManager manager;
```

Persistence Context：持久化上下文

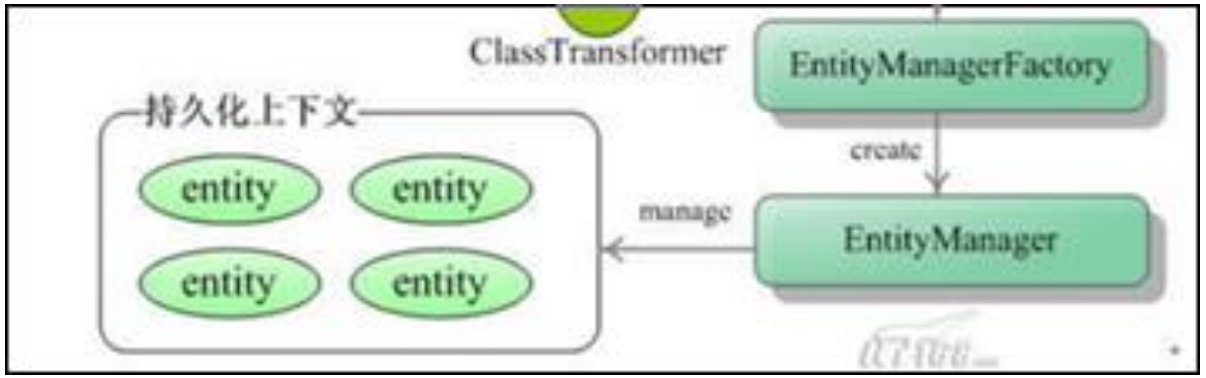
其实就是JPA的一级缓存,是一系列实体的管理环境，实体对象集合。被EntityManager管理着的，通过EntityManager和持久化上下文进行交互。

持久化上下文类型

应用程序管理

应用程序来管理所需资源

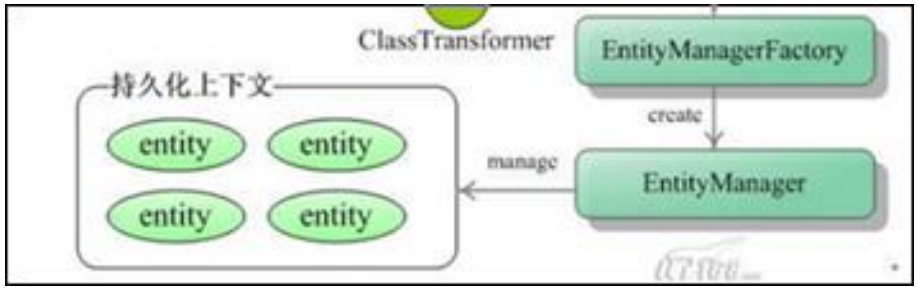
持久化上下文是绑定到当前EntityManager



Persistence Context：持久化上下文

容器管理，又分为两种：

- 和事务范围相关持久化上下文
  - 事务启动就会创建，事务结束也会结束；
  - 在无状态session bean中使用；
- 扩展的持久化上下文
  - 和有状态session bean结合使用，它的生命周期和事务无关；
  - 当Session bean对象创建，持久化上下文就创建；
  - 当Session bean对象销毁，持久化上下文就结束



JPA的使用步骤

```
public class JPAJUnitTest {
    private static EntityManagerFactory emf;
    private Person p;
    private Address a ;
    @BeforeClass
     public static void setUpClass() throws Exception {
       //1. 创建EntityManagerFactory
       emf =  Persistence.createEntityManagerFactory("AddressBookPU");
     }
    @Test
    public void createPersons() throws Exception {
       add(p);
    }
    …
}
```

JPA的使用步骤

```
public class JPAJUnitTest {
        private static EntityManagerFactory emf;
        ...
        @Test
        public void createPersons() throws Exception {
             add(p);
         }

        public static void add(Object object){
           //2. 创建EntityManager
            javax.persistence.EntityManager em = emf.createEntityManager();
           ...
         }
}
```

JPA的使用步骤

```
public class JPAJUnitTest {
    …
    public static void add(Object object){
        javax.persistence.EntityManager em = emf.createEntityManager();
        em.getTransaction().begin(); //3. 开始事务
        try {
            em.persist(object); //4. 持久化操作
            em.getTransaction().commit(); //5. 提交事务
        } catch (Exception e) {
            e.printStackTrace();
            em.getTransaction().rollback();
        } finally {
            em.close(); //6. 关闭EntityManager
        }
…
```

主要内容

JPA概述

使用JPA的步骤

实体

JPA的API

JPQL

实体

- 添加实体类和使用JPA的注记标注持久化特性

```
@Entity
@Table(name="ACCP50NT_ADDRESS")
public class Address implements Serializable {
   private static final long serialVersionUID = 1L;
   private Long id;
   private String street;
   private String city;
   private String country;

   public Address() { }  // 默认构造方法
   // Getter and setter方法
   ...
}
```

实体类注记

映射的表，缺省与类同名

POJO属性

Address.java

实体

- 添加实体类和使用JPA的注记标注持久化特性

```
@Entity
@Table(name="ACCP50NT_ADDRESS")
public class Address implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name="AddressID")
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
```

主键

写在getter方法前

属性到字段的映射，缺省与属性同名

主键生成器：
AUTO – 由系统选择生成策略
IDENTITY – 使用标识列

实体

- 添加导入的类和接口

```
package addressbook;

import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

...
```

JPA必需的接口

实体

- 使用注记描述对象之间的关联

每个Person具有一个地址Address

OneToOne关联

```
@Entity
@Table(name="ACCP50NT_PERSON")
public class Person implements Serializable {
  …
   @OneToOne(cascade=CascadeType.ALL)
   Address address;
  …
}
```

ALL--对Person的全部操作将级联到Address对象

实体—实体的定义

实体具备以下的条件：

- ✓ 必须使用 javax.persistence.Entity注解或者在 XML映射文件中有对应的元素；
- ✓ 必须具有一个不带参的构造函数，类不能声明为final，方法和需要持久化的属性也不能声明为 final；

- 如果游离状的实体对象需要以值的方式进行传递，如通Session bean的远程业务接口传递，则必须实现Serializable接口；

- 需要持久化的属性，其访问修饰符不能是public，它们必须通过实体类方法进行访问。

实体—Annotation —基本注解

- @Entity
  - ✓将对象标注为一个实体，表示需要保存到数据库中
  - ✓默认情况下类名即为表名，通过name属性显式指定表名
- @Id　对应的属性是表的主键

- @EmbeddedId或@IdClass
  - ✓组合关键字
- @Column
  - ✓属性对应的表字段

实体—Annotation —基本注解

- @GeneratedValue
  - 主键的产生策略，通过strategy属性指定；
  - 默认情况下，JPA自动选择一个最适合底层数据库的主键生成策略可供选择的策略
    - IDENTITY：表自增键字段（SqlServer对应策略，Oracle不支持这种方式）
    - AUTO： JPA自动选择合适的策略，是默认选项（MySql对应策略）；
    - SEQUENCE：通过序列产生主键，通过@SequenceGenerator注解指定序列名（如Oracle的Sequence， MySql不支持这种方式）；
    - TABLE：通过表产生主键，使用该策略可以使应用更易于数据库移植。不同的 JPA实现商生成的表名是不同的。

实体—Annotation —继承关系

继承关系：

对于继承的实体，在javax.persistence.InheritanceType定义了3种映射策略

- SINGLE_TABLE

  父子类都保存到同一个表中，通过字段值进行区分。

- JOINED

  父子类相同的部分保存在同一个表中，不同的部分分开存放，通过表连接获取完整数据。

- TABLE_PER_CLASS

  每一个类对应自己的表（一般不推荐采用这种方式）

实体—Annotation —关联关系

关联关系：

- @OneToOne
- @OneToMany
- @ManyToOne
- @ManyToMany

主要内容

JPA概述

使用JPA的步骤

实体

JPA的API

JPQL

JPA的API

下面是 EntityManager的一些主要的接口方法

- void persist(Object entity)
  - ✓新实体实例将转换为受控状态
- void remove(Object entity)
  - ✓删除某个实体对象,也就是删除数据库中某条记录
- void flush()
  - ✓将受控态的实体数据同步到数据库中
- T merge(T entity)
  - ✓游离态的实体持久化到数据库中，并转换为受控态的实体
- T find(Class entityClass, Object primaryKey)
  - ✓以主键查询实体对象，entityClass是实体的类，primaryKey是主键值

JPA的API--修改

使用JPA实现数据的加载/修改

- 程序代码

```
public void updatePerson() throws Exception {
        ...
        Person p = em.find(Person.class, new Long(2));
        p.setName("董");
        p.getAddress().setStreet("海淀16");
        ...
        em.getTransaction().begin();
        em.merge(p);
        em.getTransaction().commit();
        p = em.find(Person.class, new Long(2));
        assertEquals(p.getName(),"董");
        assertEquals(p.getAddress().getStreet(),"海淀16");
```

根据主键加载

先加载，再更新
不再需要繁琐的逐字段编码

更新数据需要进行事务控制

演示示例

JPA的API——删除

使用JPA实现数据的加载/删除

- 程序代码

```
@Test
public void removePerson() throws Exception {
    javax.persistence.EntityManager em =  emf.createEntityManager();
    Person p = em.find(Person.class, new Long(2));
    try {
        em.getTransaction().begin();
        em.remove(p);
        em.getTransaction().commit();
        p = em.find(Person.class, new Long(2));
        assertNull(p);
    } catch (Exception e) {
        ...
```

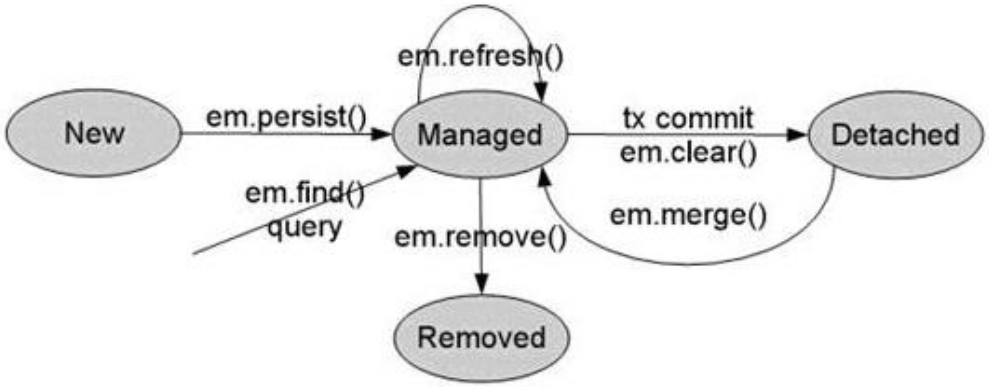
先加载，再删除

删除数据需要进行事务控制

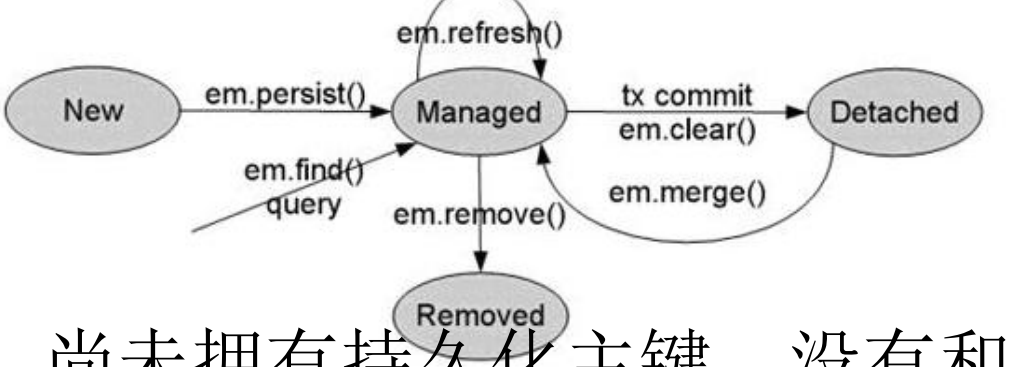
级联删除

JPA的生命周期


does not exist
POJO construction
Object obj = new Object();
自由状态
new
1. Database remove
2. @PostRemove
1. EntityManager.persist(obj)
2. @PrePersist
3. Database insert
4. @PostPersist
分离状态
1. EntityManager.remove(obj)
2. @PreRemove
3. Pending removal in database
serialized to another tier
detached
managed
removed
EntityManager.merge(obj)
1. Update query
2. @PreUpdate
3. Database update
4. @PostUpdate
1. EntityManager.refresh(obj)
2. @PostLoad
删除状态
1. Entity public API
2. @PreUpdate
3. Database update
4. @PostUpdate
持久状态


em.refresh()
New
em.persist()
Managed
tx commit
em.clear()
Detached
em.find()
query
em.remove()
em.merge()
Removed

实体的状态



新建态new

新创建的实体对象，尚未拥有持久化主键，没有和一个持久化上下文关联起来

受控态managed

已经拥有持久化主键并和持久化上下文建立了联系

游离态detached

拥有持久化主键，但尚未和持久化上下文建立联系

删除态removed

拥有持久化主键，已经和持久化上下文建立联系，但已经被安排从数据库中删除

主要内容

JPA概述

使用JPA的步骤

实体

JPA的API

JPQL

JPQL

- Java Persistence Query Language（Java持久化查询语言）
- 是一种可移植的查询语言，可以被编译成所有主流数据库服务器上的SQL
- JPQL是面向对象的，通过面向对象而非面向数据库的查询语言查询数据，在Java空间对类和对象进行操作，避免程序的SQL语句紧密耦合
- 使用 javax.persistence.Query接口代表一个查询实例

JPQL—创建Query实例

- 通过EntityManager来生成Query实例：

EntityManager提供的使用JPQL（或原生SQL）创建Query的方法：

方法	用途
public Query createQuery(String sqlString);	使用 JPQL 语句创建动态查询
public Query createNamedQuery(String name);	在命名查询的基础上创建查询实例。此方法可以用于 JPQL 和原生 SQL 查询
public Query createNativeQuery(String sqlString);	使用包含 UPDATE 或 DELETE 的原生 SQL 语句创建动态查询
public Query createNativeQuery(String sqlString, Class result-class);	使用原生 SQL 语句创建检索单一实体类型的动态查询
public Query createNativeQuery(String sqlString, String result-setMapping)；	使用原生 SQL 语句创建检索多个实体结果集合的动态查询

JPQL—使用Query接口

Query接口执行数据查询的部分方法：

- getSingleResult()：单一查询结果
- getResultList()：多个查询结果
- setParameter：
  - Query setParameter(int position, Object value)：通过参数位置号绑定查询语句中的参数
  - Query setParameter(String name, Object value)：绑定命名参数
- setMaxResults：设置返回的最大结果数
- executeUpdate：新增、删除或更改的语句，通过该方法执行

Query

使用JPA-QL

- 返回单一查询结果的程序代码 JPAQLJunitTest.java

```
@Test
public void findPerson() throws Exception {
    javax.persistence.EntityManager em = emf.createEntityManager();
    Person p = (Person)em.createQuery(
                "select p from Person p where p.name = :name")
                .setParameter("name", "董")
                .getSingleResult();
    assertEquals(p.getName(),"董");
    assertEquals(p.getSurname(),"一");
}
...
```

类似HQL

设定查询的参数值

返回单一查询结果

表 "ACCP50NT_PERSON" 中的数据。位置是 "zf" 中。 "(...

PersonID	SURNAME	NAME	ADDRESS_AddressID
2	一	董	3
4	二	张	<NULL>
5	三	李	<NULL>

Query

使用JPA的QL

- 返回多个查询结果的程序代码

```
@Test
public void findPersons() throws Exception {
   javax.persistence.EntityManager em = emf.createEntityManager();
   List<Person> p = (List<Person>)em.createQuery("select p from
                          Person p where not(p.name = :name)")
                    .setParameter("name", "董")
                    .getResultList();
   assertEquals(p.size(),2);
   assertEquals(p.get(0).getName(),"张");
   assertEquals(p.get(1).getName(),"李");
}
...
```

设定查询的参数值

返回查询结果列表

表 "ACCP50NT_PERSON" 中的数据，位置是 "zf" 中、"(...

PersonID	SURNAME	NAME	ADDRESS_AddressID
2	一	董	3
4	二	张	<NULL>
5	三	李	<NULL>

JPQL

JPQL支持三种语句类型，可以在查询中使用JPQL执行选择、更新、删除操作：

语句类型	描述
SELECT	检索实体或与实体有关的数据
UPDATE	更新一个或多个实体
DELETE	删除一个或多个实体

定义和使用Select：

```
SELECT c
FROM Category c
WHERE c.categoryName LIKE :categoryName
ORDER BY c.categoryId
```

该 JPQL 查询具有（或可能具有）如下内容：

- □SELECT 子句，指定要检索的对象类型、实体或值；
- □FROM 子句，指定其它子句是使用的实体声明；
- □可选的 WHERE，过滤查询返回的结果；
- □可选的 ORDER BY 子句，排序查询检索到的结果；
- □可选的 GROUP BY 子句，执行聚合；
- □可选的 HAVING 子句，执行与聚合结合的过滤。

JPQL--条件表达式和操作

操作符类型	操作符
定位	.
一元符号	+、-
算术	*、/、+、-
关系	=、>、>=、<、<=、<>、 [NOT] BETWEEN、[NOT] LIKE、[NOT] IN IS [NOT] NULL、IS [NOT] EMPTY、[NOT] MEMBER OF
逻辑	NOT AND OR

JPQL--使用JPQL函数

字符串函数

1. 字符串函数

可以在 JPQL 查询的 SELECT 子句中使用字符串函数，这些函数的唯一用途是过滤查询结果。如果希望对数据执行任何字符串操作，就必须使用 Java 语言中可用的函数。主要原因在于应用程序在内存中维护字符串比在数据库中进行同样操作要快得多。

CONCAT(string 1, string 2): 返回两个字符串或字面量连接后的值

SUBSTRING(string, position, length): 返回从 position 开始长度为 length 的子字符串

LOWER(string):将一个字符串 string 转换成小写形式。

UPPER(string):将一个字符串 string 转换成大写形式。

LENGTH(string):返回字符串 string 的长度，为整数。

TRIM([[LEADING|TRAILING|BOTH] [trim_character] FROM] string_to_trimmed):剪切特定字符，得到新字符串。剪切方式可以是 LEADING、TRAILING 或[BOTH]。如果没有指定 trim_character，则剪切空白

LOCATE(searchString,stringToBeSearched [initialPosition]): 返回给定字符串在另一个字符串中的位置。如果没有指定 initialPosition，则从 position 1开始检索

JPQL--使用JPQL函数

算术函数

算术函数	描述
ABS (simple_arithmetic_expression)	返回绝对值
SORT (simple_arithmetic_expression)	返回平方根的双精度值
MOD (num, div)	返回对 num、div 执行取摸操作的结果
SIZE (collection_value_path_expression)	返回集合中项目的数量

时间函数

时间函数	描述
CURRENT_DATE	返回当前日期
CURRENT_TIME	返回当前时间
CURRENT_TIMESTAMP	返回当前时间标记

JPQL--投影

在 SELECT 子句中使用构造器表达式

可以在 SELECT 子句中使用构造器返回一个或多个 Java 实例。当你希望在用从子查询中检索到的数据初始化的查询中创建实例时，这一特性特别有用：

SELECT NEW actionbazaar.persistence.ItemReport(c.categoryID, c.createdBy)

FROM Category c

指定的类不必映射到数据库，也不必是实体。

例：

HomeroomClassMembershipLifecycleImpl.findStudentCountBySchoolClassAndDateGroupBySex

JPQL

使用聚合：

聚合函数：AVG、COUNT、MAX、MIN和SUM

通过GROUP BY和HAVING分组

排序：ORDER BY

使用子查询

可以在WHERE和HAVING子句中使用子查询，EJB3.1的FROM子句中不支持；

可以在子查询中使用IN、EXISTS、ALL、ANY和SOME

联接实体

关联联结：INNER JOIN

外联结：LEFT JOIN或LEFT OUTER JOIN

获取联结：JOIN FETCH

JPQL-- Update和Delete

大数据量的更新、删除，可以用 EntityManager进行实体的更新操作，也可以通过查询语言执行数据表的字段更新，记录删除的操作

```
UPDATE PollOption p SET p.optionItem =
:value
WHERE p.optionId = :optionId
DELETE FROM PollOption p
WHERE p.optionId = :optioned
```

本章小结

- 开放数据库连接
  - ✓ ODBC
  - ✓ OLE DB
  - ✓ ADO
  - ✓ JDBC
- 对象关系映射ORM
  - ✓ Hibernate
  - ✓ MyBatis
- JPA介绍
  - ✓ 概念　编程