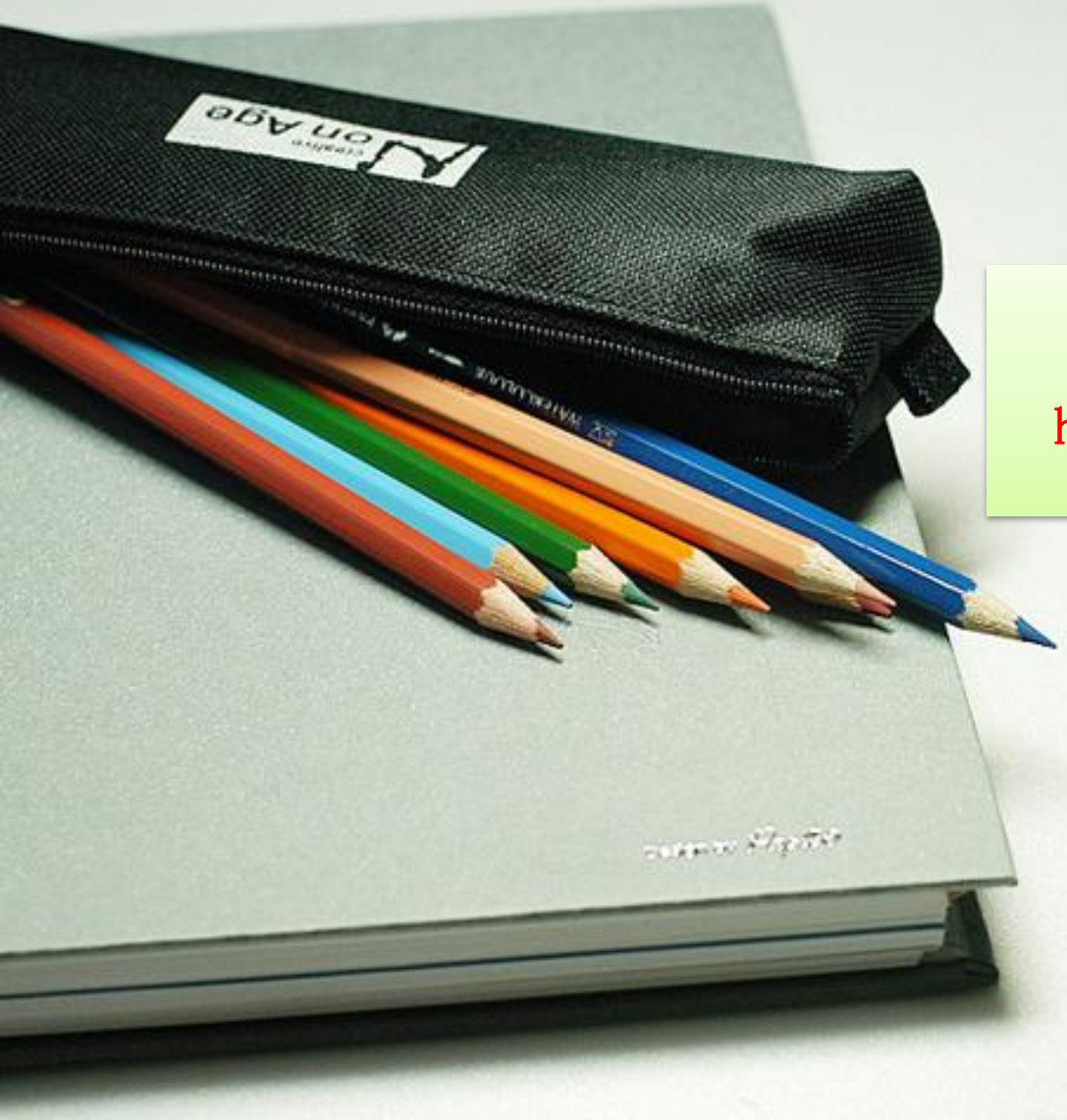


北京圣思园科技有限公司
http://www.shengsiyuan.com

主讲人：张龙

Hibernate应用开发详解

- 教学目标
  - 创建Hibernate的配置文件
  - 创建持久化类
  - 创建数据库Schema
  - 创建对象-关系映射文件
  - 映射文件的文档类型定义（DTD）
  - Hibernate与Struts集成
  - 练习使用MyEclipse进行Struts与Hibernate项目的开发

应用程序的分层体系结构发展

- 双层应用
  - 应用程序层
  - 数据库层
- 三层应用
  - 表述层
  - 业务逻辑层
  - 数据库层
- 四层应用
  - 表述层
  - 业务逻辑层
  - 持久化层
  - 数据库层
- N层应用

Hibernate开发实例

- 层与层之间存在自上而下的依赖关系，即上层组件会访问下层组件的API，而下层组件不应该依赖上层组件。例如：表述层依赖于业务逻辑层，而业务逻辑层依赖于数据库层。
- 每个层对上层公开API，但具体的实现细节对外透明。当某一层的实现发生变化，只要它的API不变，不会影响其他层的实现。

软件分层的优点

- 1．伸缩性

  伸缩性指应用程序是否能支持更多的用户。应用的层越少，可以增加资源（如CPU和内存）的地方就越少。层数越多，可以将每层分布在不同的机器上
- 2．可维护性

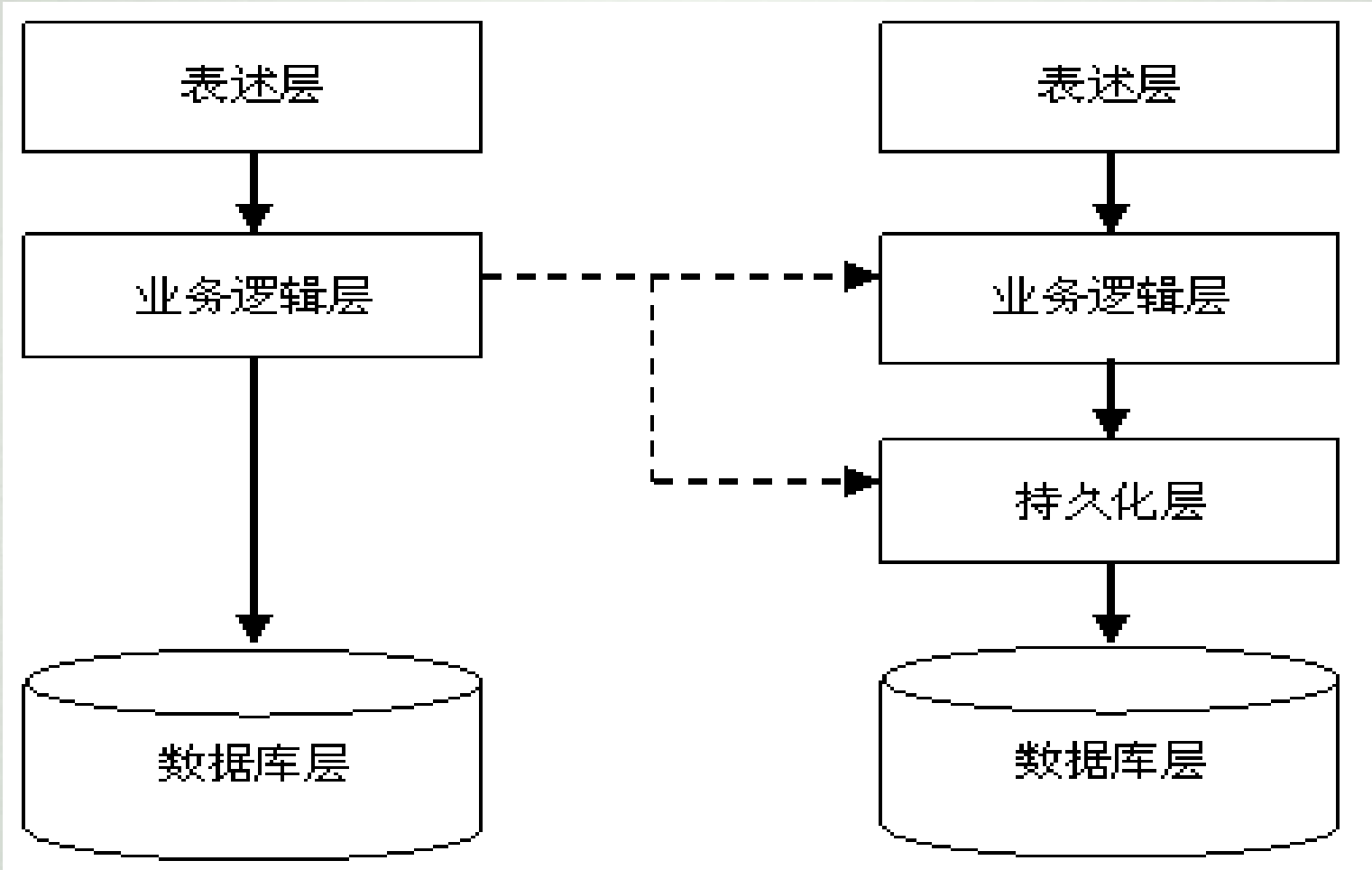
  可维护性指的是当发生需求变化，只需修改软件的某一部分，不会影响其他部分的代码。
- 3．可扩展性

  可扩展性指的是在现有系统中增加新功能的难易程度。层数越多，就可以在每个层中提供扩展点，不会打破应用的整体框架。
- 4．可重用性

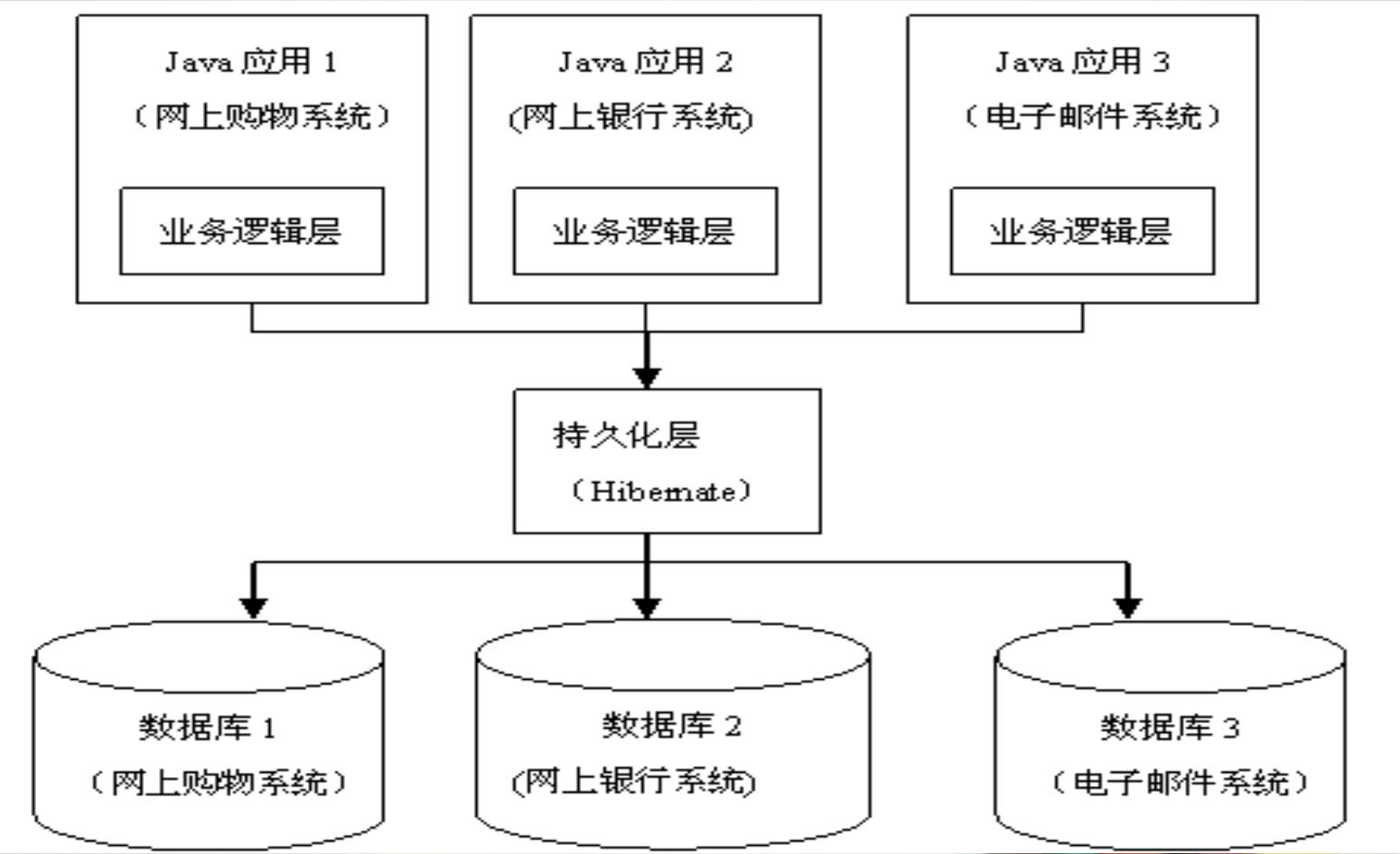
  可重用性指的是程序代码没有冗余，同一个程序能满足多种需求。例如，业务逻辑层可以被多种表述层共享。
- 5．可管理性

  可管理性指的是管理系统的难易程度。将应用程序分为多层后，可以将工作分解给不同的开发小组，从而便于管理。应用越复杂，规模越大，需要的层就越多

Java应用的持久化层



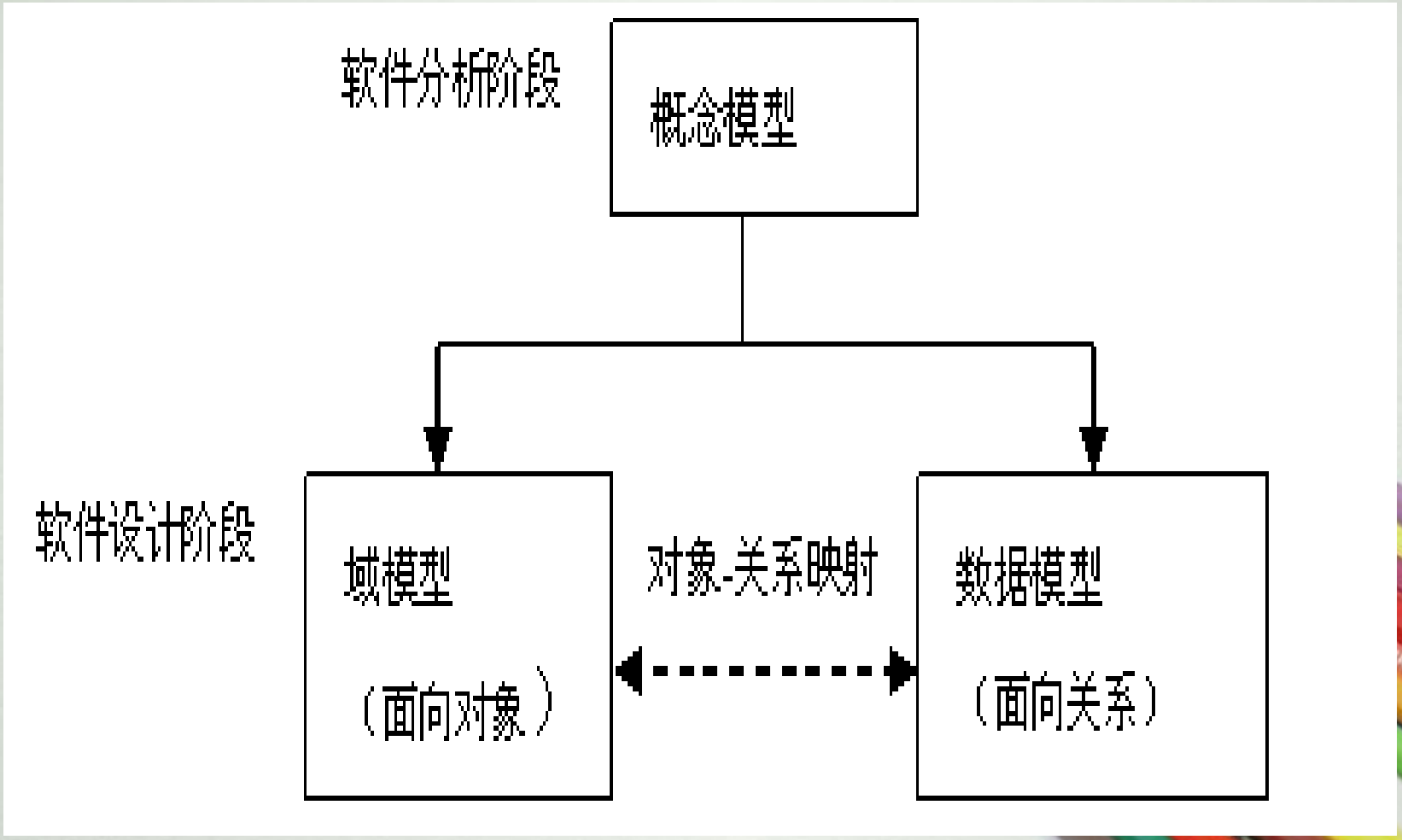
Hibernate是持久化层的一种实现方式



软件的模型

- 在软件开发领域，模型用来表示真实世界的实体。
- 在软件开发的不同阶段，需要为目标系统创建不同类型的模型：
  - 在分析阶段，需要创建概念模型。
  - 在设计阶段，需要创建域模型和数据模型。

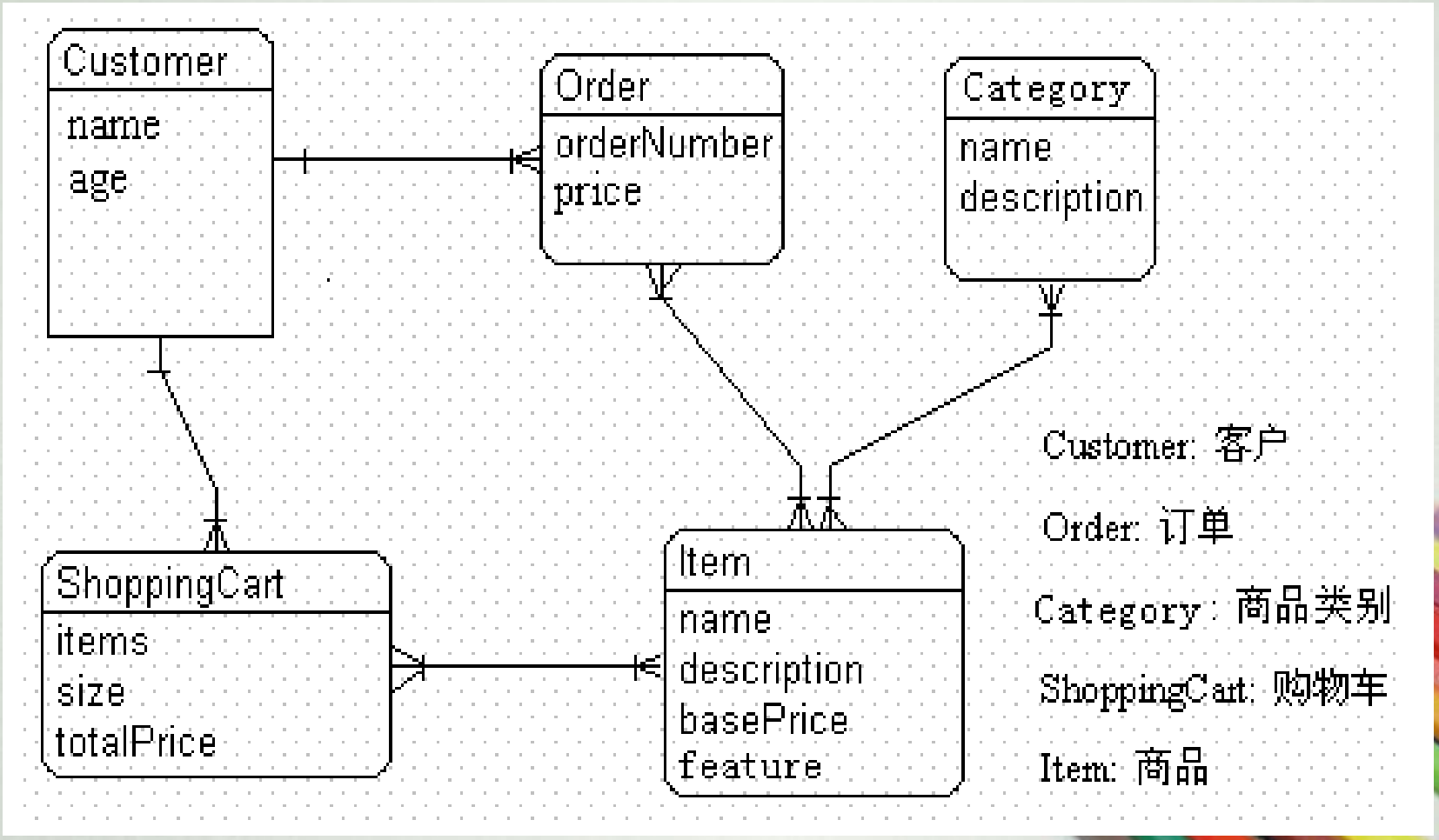
模型之间的关系



概念模型

- 概念模型用来模拟问题域中的真实实体。
- 概念模型描述了每个实体的概念和属性，以及实体之间的关系。
- 概念模型并不描述实体的行为。
- 不管是技术人员还是非技术人员都能看得懂概念模型，他们可以很容易的提出模型中存在的问题，帮助系统分析人员及早对模型进行修改。

购物网站应用的概念模型



实体与实体之间存在三种关系

- **Customer和Order实体：** 一对多。一个客户有多个订单，而一个订单只能属于一个客户。
- **Category和Item实体：** 多对多。一个商品类别包含多个商品，而一个商品可以属于多个商品类别。
- **Order和Item实体：** 多对多。一个订单包含多个商品，而一个商品可以属于多个订单。
- **Customer和ShoppingCart实体：** 一对多。一个客户有多个购物车，而一个购物车只能属于一个客户
- **ShoppingCart和Item实体：** 多对多。一个购物车包含多个商品，而一个商品可以属于多个购物车。

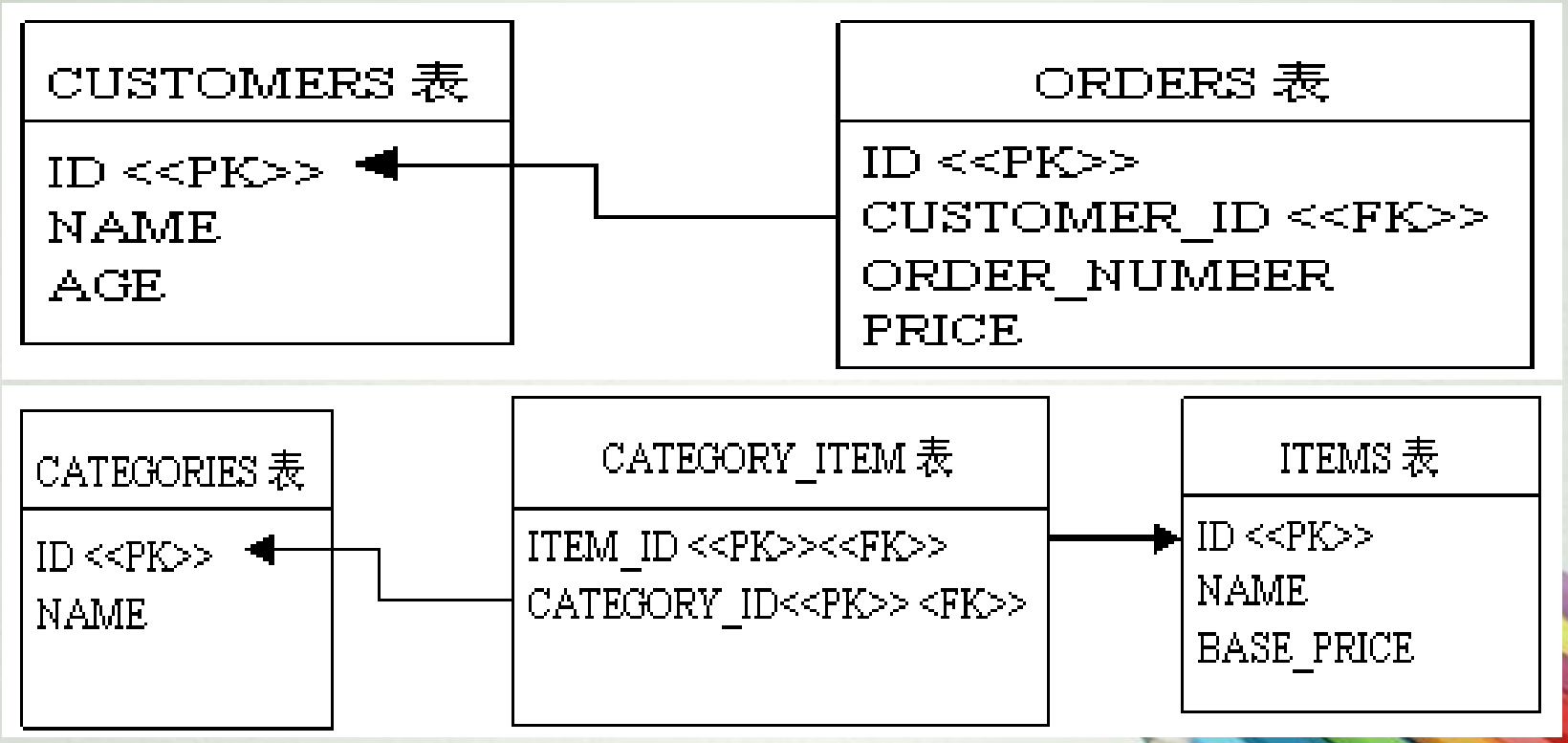
关系数据模型

- 关系数据模型是在概念模型的基础上建立起来的，用于描述这些关系数据的静态结构，它由以下内容组成：
  - 一个或多个表
  - 表的所有索引
  - 视图
  - 触发器
  - 表与表之间的参照完整性

表的主键

- 在关系数据库表中，用主键来识别记录并保证每条记录的惟一性。作为主键的字段必须满足以下条件：
  - 不允许为null。
  - 每条记录具有惟一的主键值，不允许主键值重复。
  - 每条记录的主键值永远不会改变。
- 使用代理主键机制，代理主键不具有业务含义，不会被改变。（auto increment,uniqueidentifier）

表与表之间的参照完整性



用连接表表示多对多关系(重要)

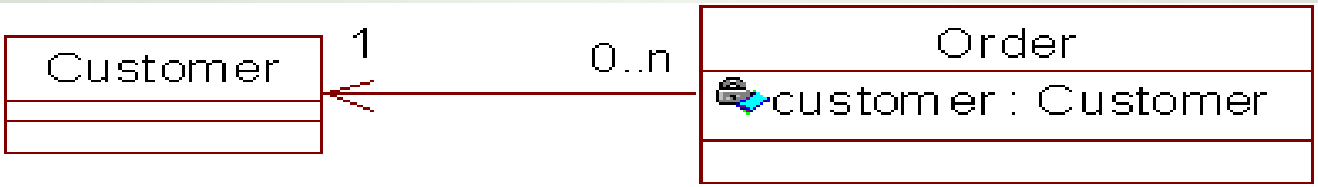
域模型

- 域模型是面向对象的。在面向对象术语中，域模型也可称为设计模型。域模型由以下内容组成：
  - 具有状态和行为的域对象
  - 域对象之间的关系
    - 关联
    - 依赖
    - 聚集
    - 一般化（泛化）

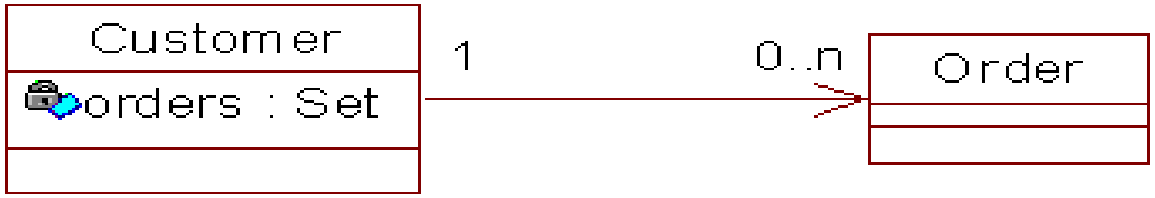
域对象之间的关系

- 关联(Association)
- 依赖(Dependency)
- 聚集(Aggregation)
- 一般化(Generalization)

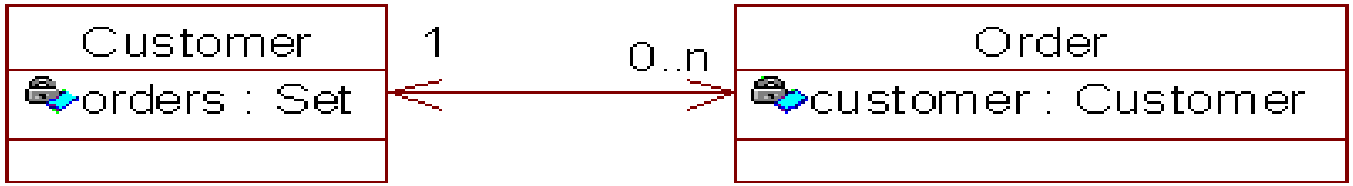
关联关系



从 Order 到 Customer 的多对一单向关联

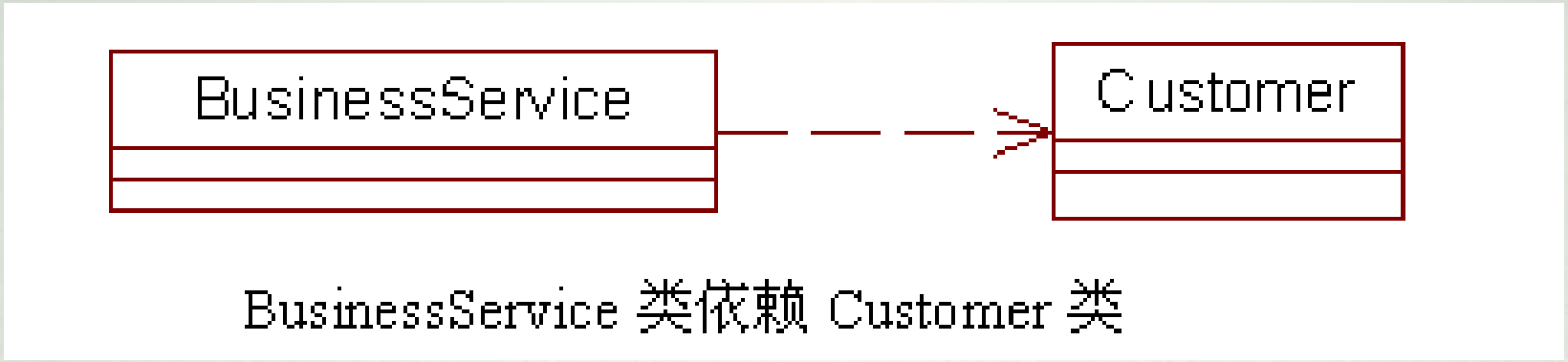


从 Customer 到 Order 的一对多单向关联



从 Customer 到 Order 的一对多双向关联

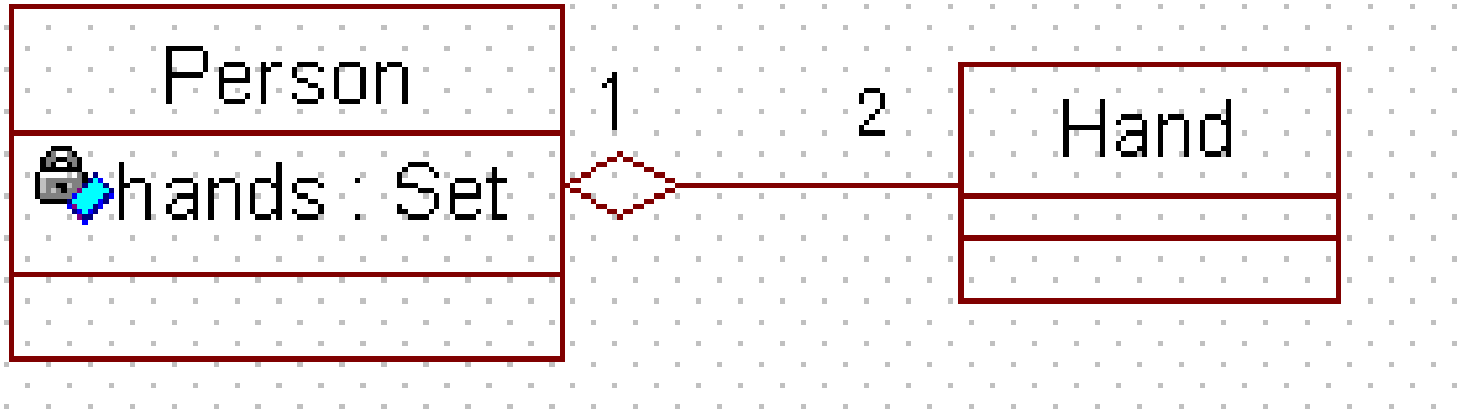
依赖关系



在BusinessService类中访问Customer类的方法，并且构造Customer类的实例

聚集关系

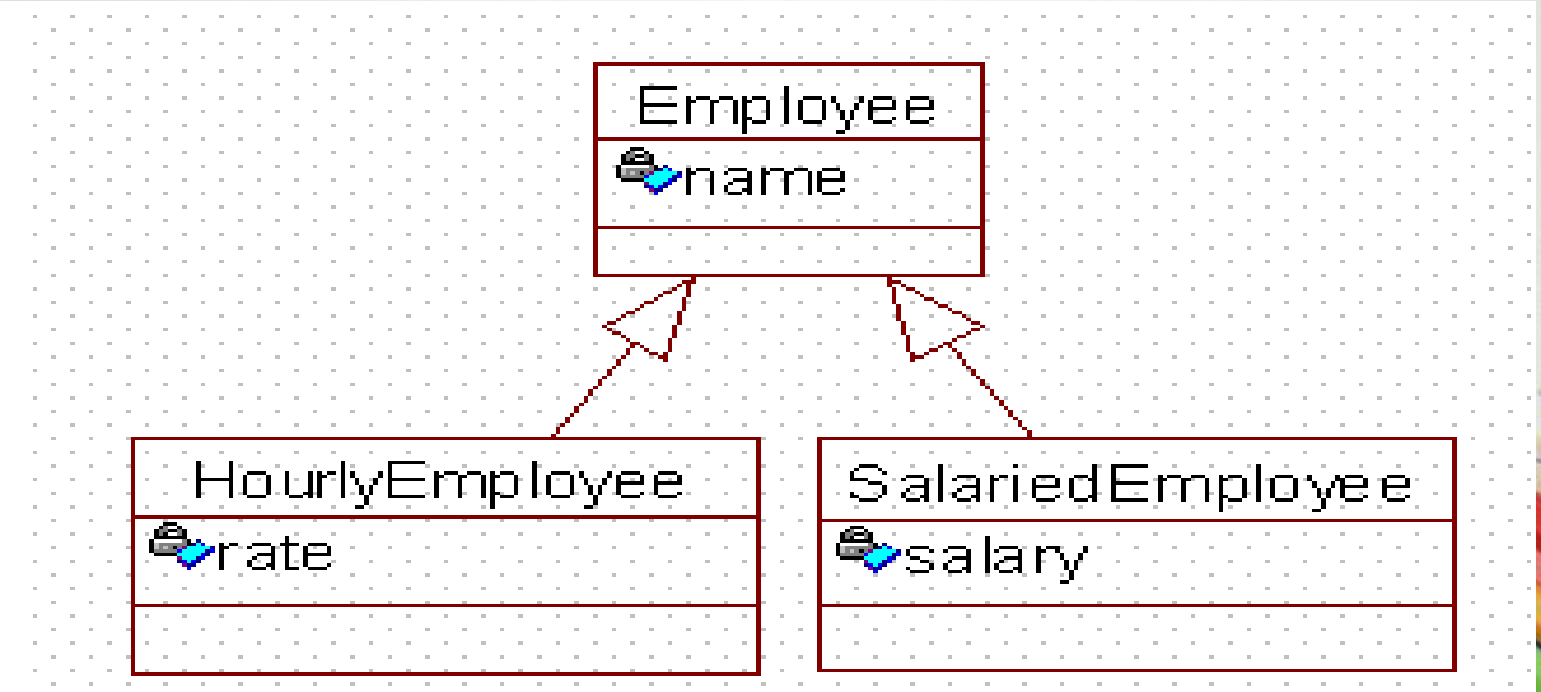
- 聚集指的是整体与部分之间的关系，在实体域对象之间很常见



Person 类与 Hand 类之间的聚集关系

一般化关系

- 一般化指的是类之间的继承关系。



Employee 类之间的继承关系

域对象

- 域对象可以代表业务领域中的人、地点、事物或概念。域对象分为以下几种：
  - 实体域对象：业务领域的名词
  - 过程域对象：业务领域的动词
  - 事件域对象：业务领域中的事件

实体域对象

- 实体对象可以代表人、地点、事物或概念。例如客户、订单、商品等作为实体域对象。
- 对于J2EE Web应用，这些名词可以作为包含状态和行为的JavaBean。采用JavaBean形式的实体域对象也称为POJO（Plain Old Java Object）。
- 为了使实体域对象与关系数据库表中记录对应，可以为每个实体域对象分配惟一的OID（Object Identifier，即对象标识符），OID是关系数据库表中的主键（通常为代理主键）在实体域对象中的等价物。

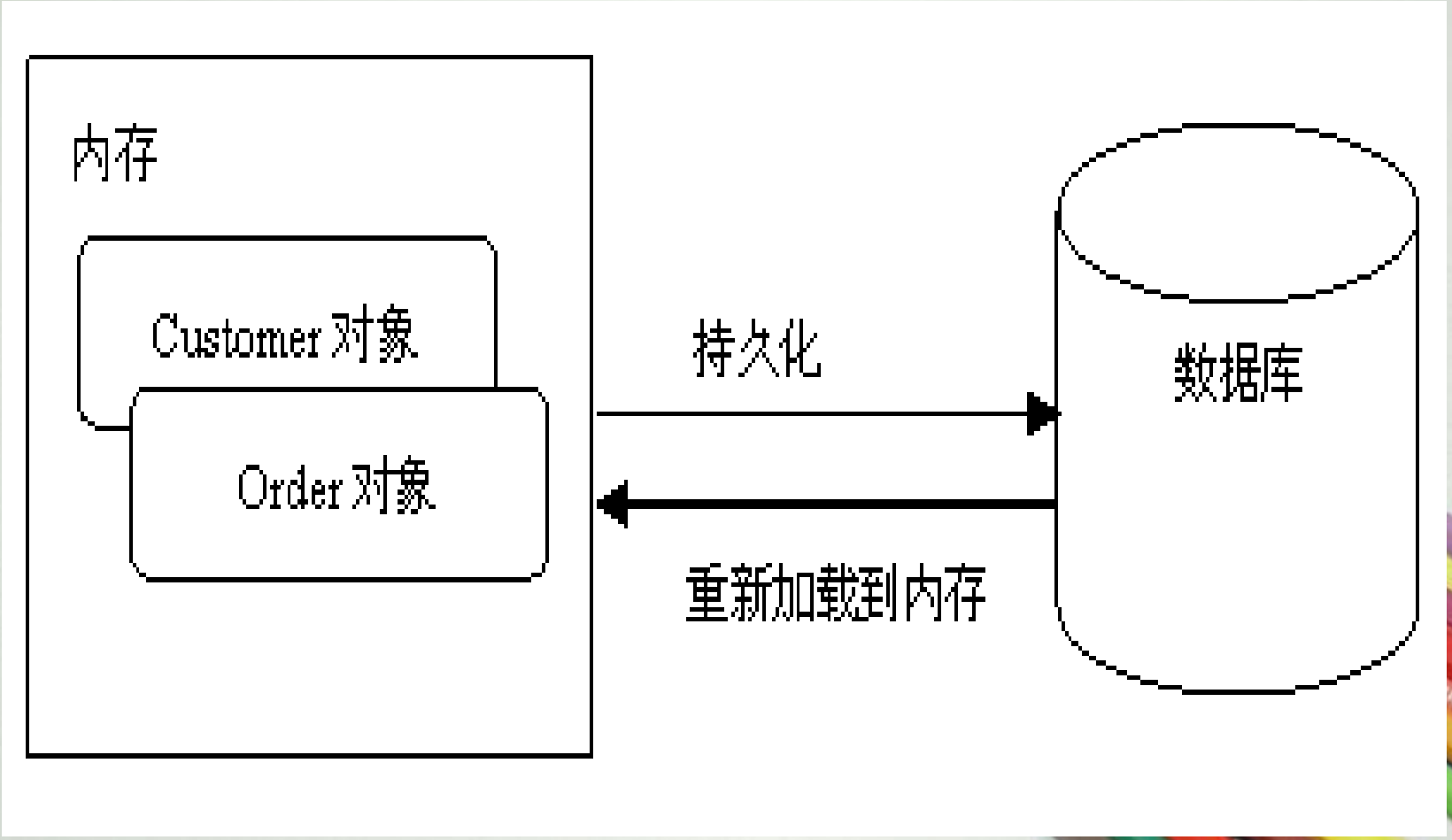
过程域对象

- 过程域对象代表应用中的业务逻辑或流程。它们通常依赖于实体域对象。
- 可以把业务领域中的动词，例如客户发出订单、登入应用等作为过程域对象。
- 在J2EE Web应用中，它们可作为常规的JavaBean，具有管理和控制应用的行为。

事件域对象

- 事件域对象代表应用中的一些事件（如异常、警告或超时）。这些事件通常由系统中的某种行为触发。
- 例如在多用户环境中，当一个客户端程序更新了某种实时数据，服务器端程序会创建一个事件域对象，其他正在浏览相同数据的客户端程序能够接受到这一事件域对象，随即同步刷新客户界面。

域对象的持久化概念



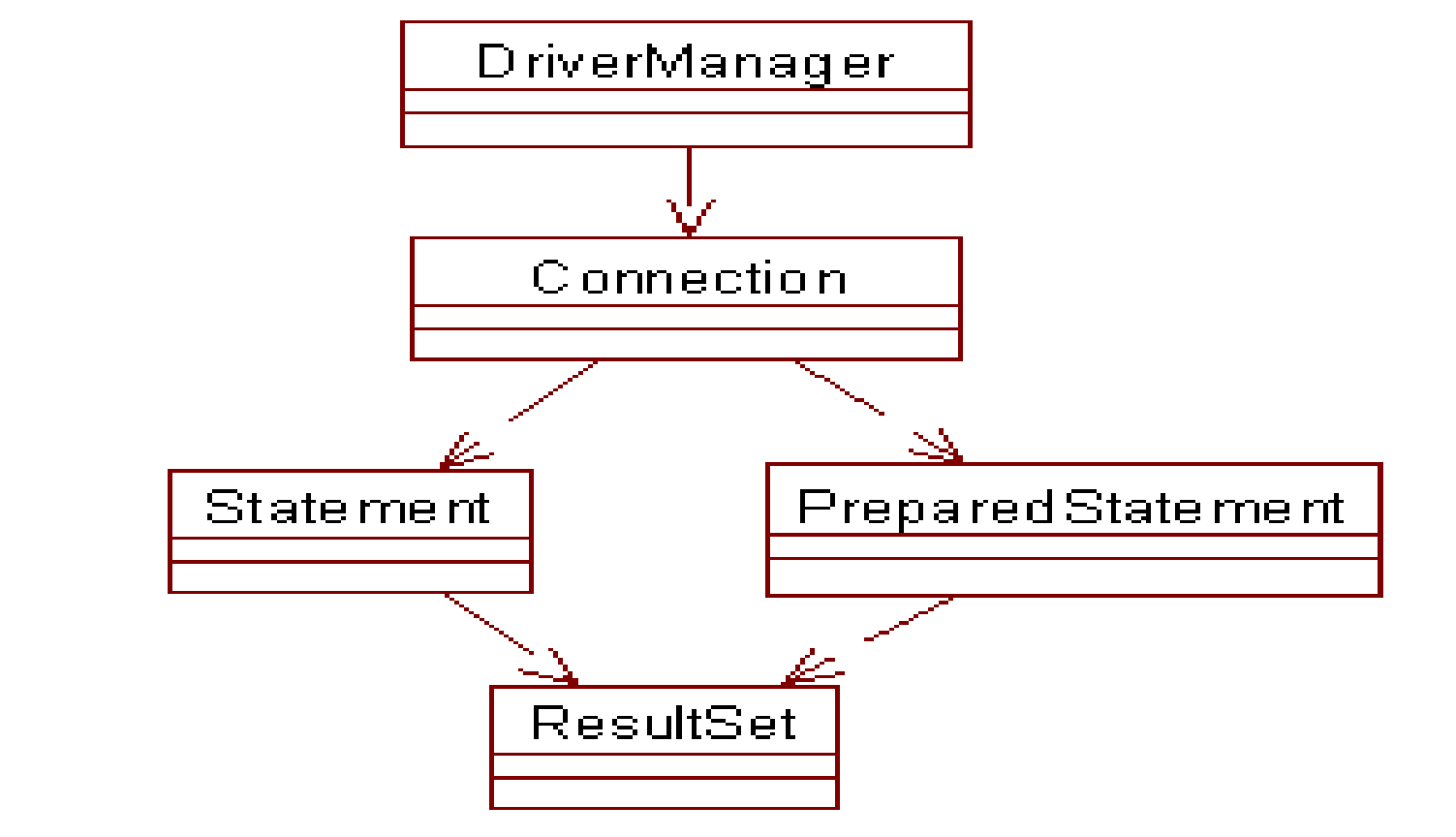
域对象的持久化概念

- 狭义的理解，“持久化”仅仅指把域对象永久保存到数据库中
- 广义的理解，“持久化”包括和数据库相关的各种操作：
  - 保存：把域对象永久保存到数据库中。
  - 更新：更新数据库中域对象的状态。
  - 删除：从数据库中删除一个域对象。
  - 加载：根据特定的OID，把一个域对象从数据库加载到内存中。
  - 查询：根据特定的查询条件，把符合查询条件的一个或多个域对象从数据库加载到内存中。

通过JDBC API来持久化实体域对象

- Java应用访问数据库的最直接的方式就是直接访问JDBC API
- java.sql包提供了JDBC API。在java.sql包中常用的接口和类包括：
  - DriverManager：驱动程序管理器，负责创建数据库连接。
  - Connection：代表数据库连接。
  - Statement：负责执行SQL语句。
  - PreparedStatement：负责执行SQL语句，具有预定义SQL语句的功能。
  - ResultSet：代表SQL查询语句的查询结果集。

JDBC API


DriverManager
Connection
Statement
PreparedStatement
ResultSet

JDBC编程的缺点

- 实现业务逻辑的代码和数据库访问代码掺杂在一起，使程序结构不清晰，可读性差。
- 在程序代码中嵌入面向关系的SQL语句，使开发人员不能完全运用面向对象的思维来编写程序。

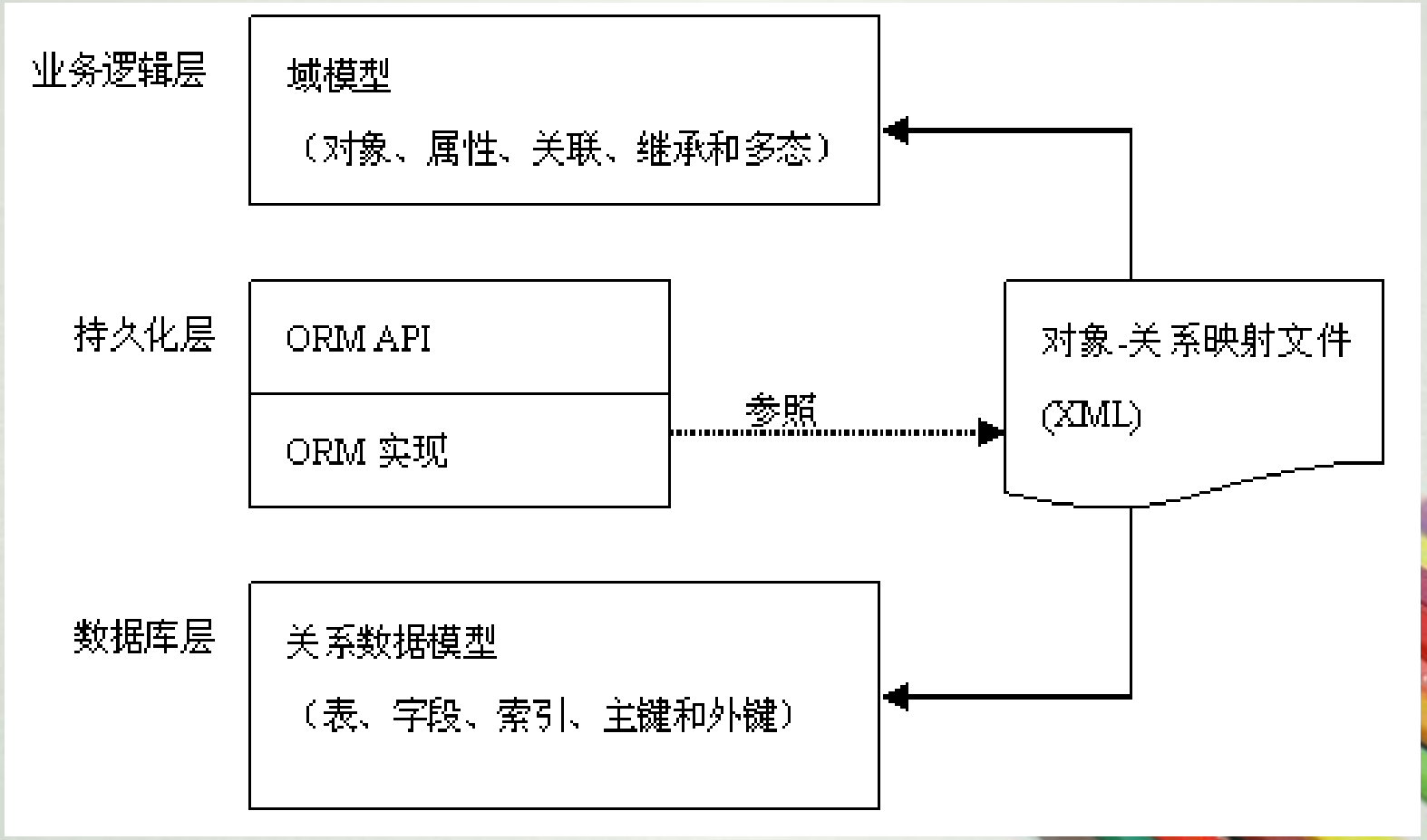
JDBC编程的缺点

- 业务逻辑和关系数据模型绑定，如果关系数据模型发生变化，例如修改了CUSTOMERS表的结构，那么必须手工修改程序代码中所有相关的SQL语句，这增加了维护软件的难度。
- 如果程序代码中的SQL语句包含语法错误，在编译时不能检查这种错误，只有在运行时才能发现这种错误，这增加了调试程序的难度。

对象—关系映射

- ORM（object—relationship mapping）模式：在单个组件中负责所有实体域对象的持久化，封装数据访问细节。
- Hibernate是ORM的一个实现

ORM模式


业务逻辑层
域模型
（对象、属性、关联、继承和多态）
持久化层
ORM API
ORM 实现
参照
对象-关系映射文件
(XML)
数据库层
关系数据模型
（表、字段、索引、主键和外键）

对象-关系映射(Object-Relation Mapping)的概念

- ORM解决的主要问题就是对象-关系的映射。域模型和关系模型都分别建立在概念模型的基础上。域模型是面向对象的，而关系数据模型是面向关系的.
- 一般情况下，一个持久化类和一个表对应，类的每个实例对应表中的一条记录。

对象-关系映射(Object-Relation Mapping)的概念

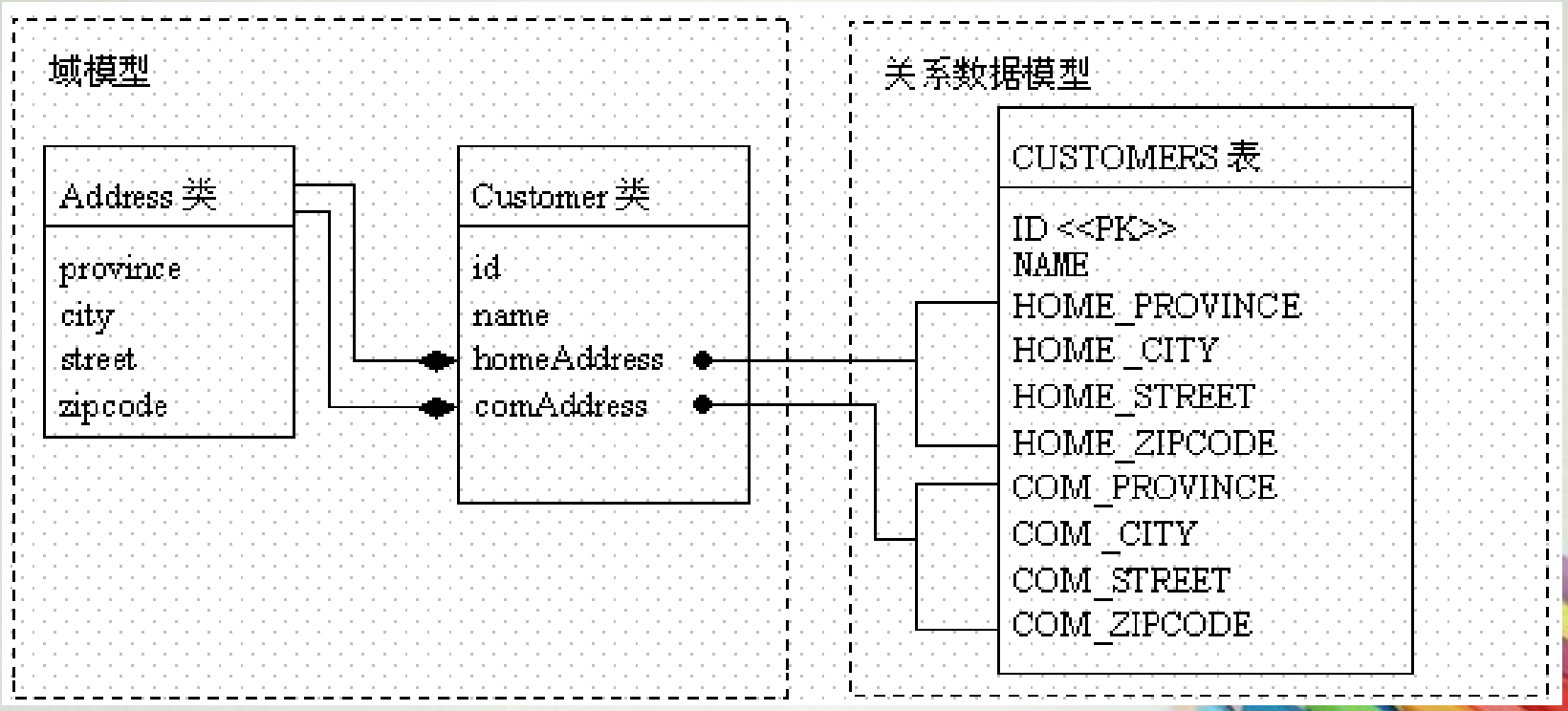
面向对象概念	面向关系概念
类	表
对象	表的行（即记录）
属性	表的列（即字段）

域模型与关系模型之间存在许多不匹配之处。

域模型与关系模型之间存在许多不匹配之处

- 域模型中有继承关系，关系模型不能直接表示继承关系
- 域模型中有多对多关联关系，关系模型通过连接表来表示多对多关联关系
- 域模型中有双向关联关系，关系模型只有单向参照关系，而且总是many方参照one方。
- 域模型提倡精粒度模型，而关系模型提倡粗粒度模型

域模型与关系模型之间的不匹配举例



精粒度域模型和粗粒度关系模型

在Java应用中使用Hibernate的步骤

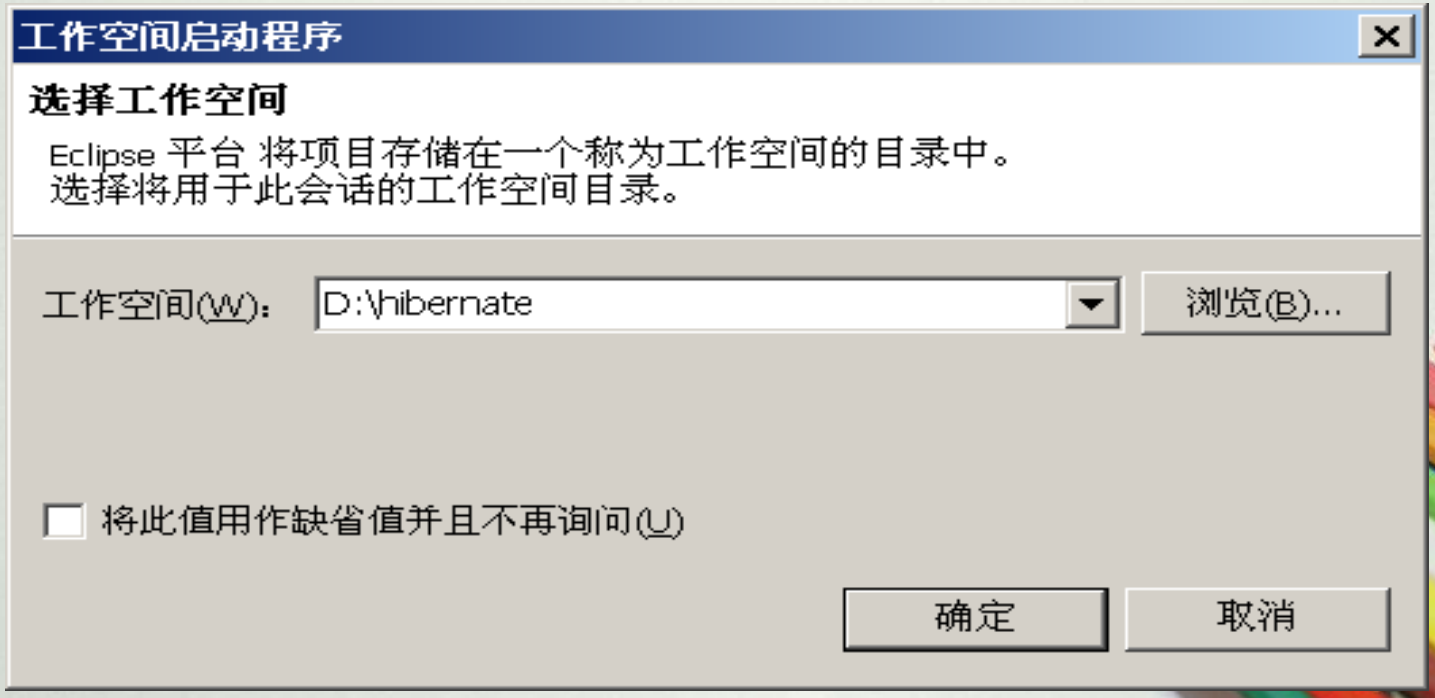
- 创建Hibernate的配置文件
- 创建持久化类
- 创建对象-关系映射文件
- 通过Hibernate API编写访问数据库的代码

开始Hibernate之旅

- 整合Struts与Hibernate进行应用的开发
- 开发环境：MyEclipse
- 软件的分层体系结构
  - 表示层
  - 业务逻辑层
  - 数据持久层
  - 数据库层

开始Hibernate之旅

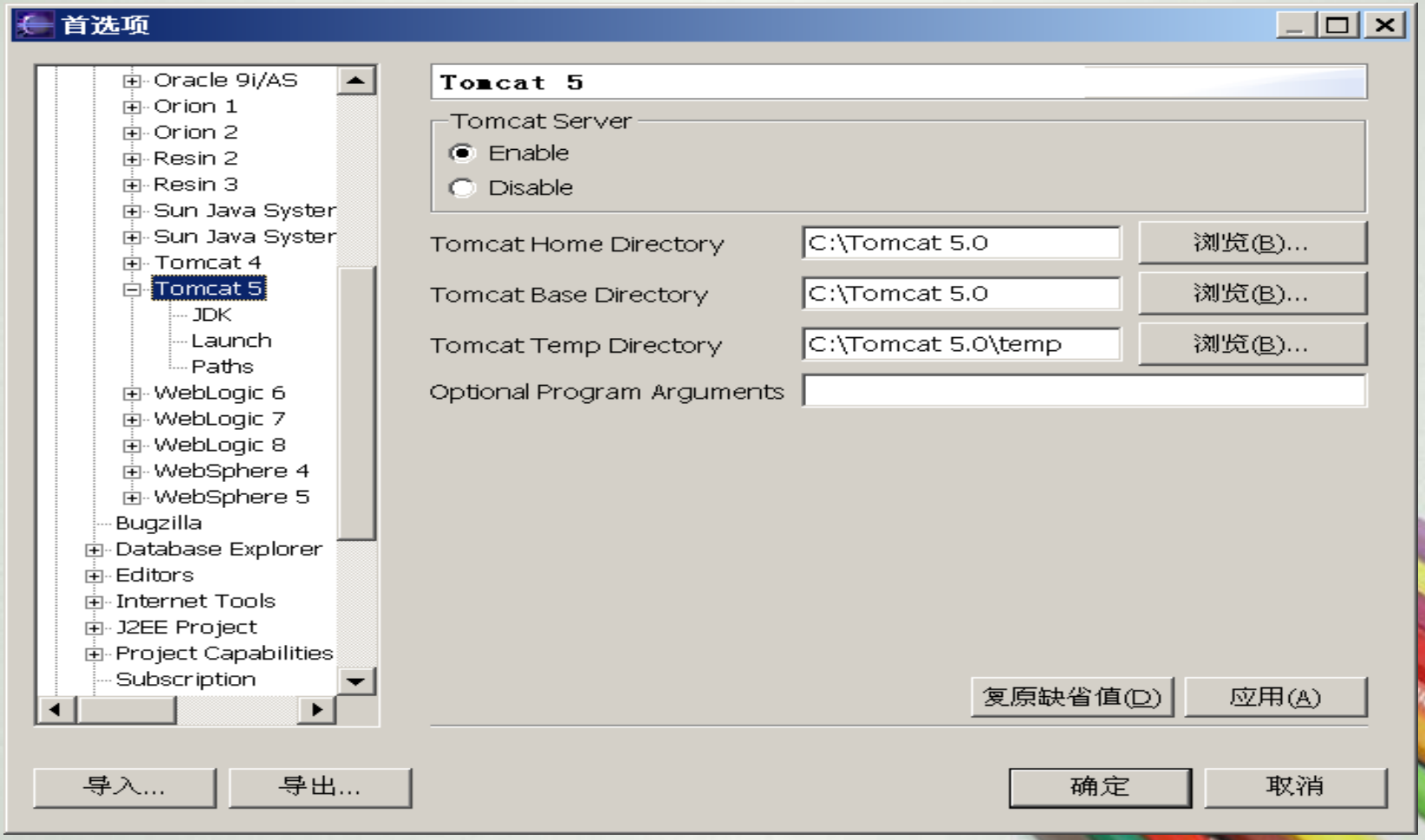
- 安装MyEclipse
- 启动Eclipse,工作空间位置：D:\hibernate



开始Hibernate之旅

- 在Eclipse中配置Tomcat服务器，方便在Eclipse中启动
- 窗口→首选项→MyEclipse→Application Servers→Tomcat 6

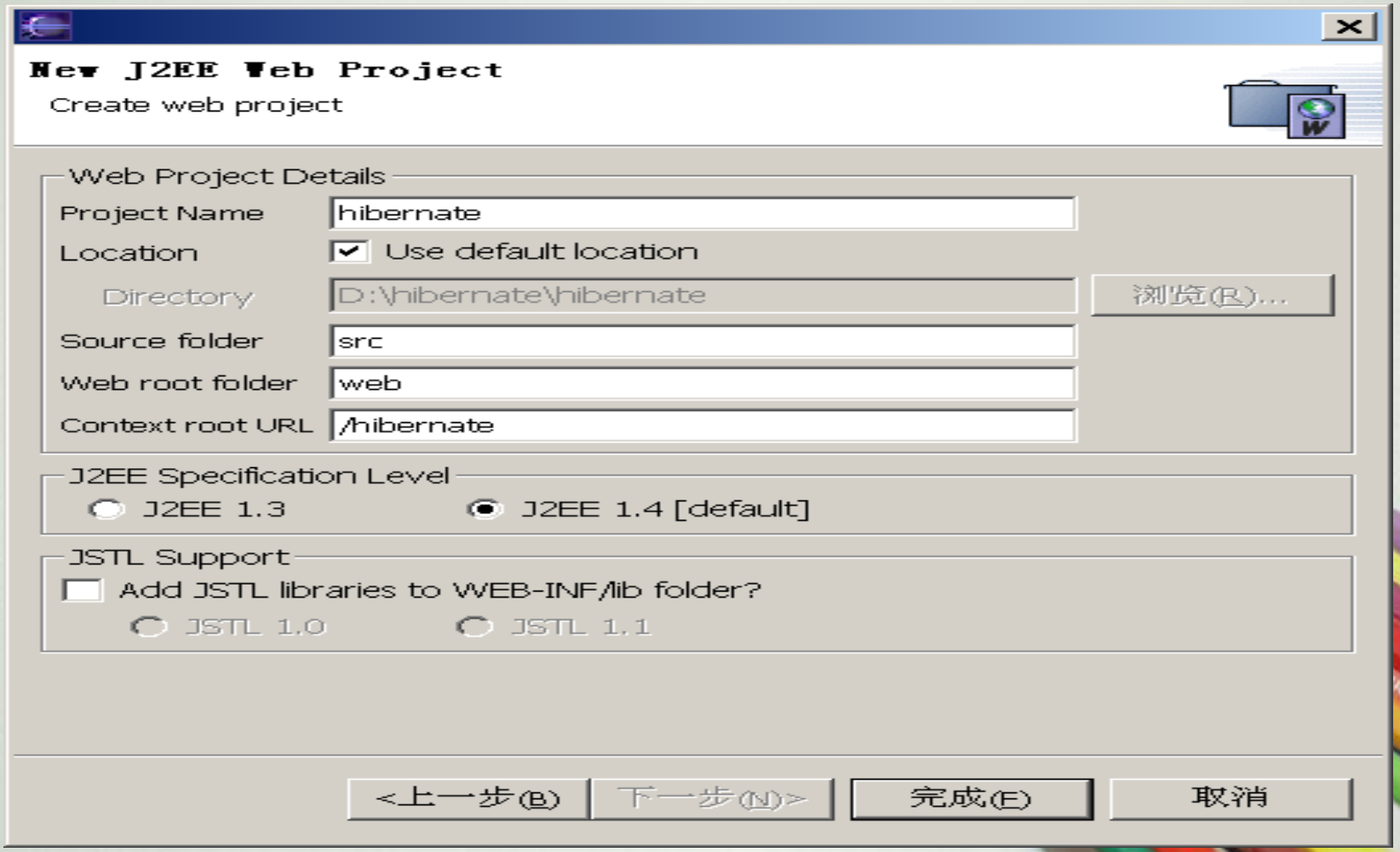
开始Hibernate之旅



开始Hibernate之旅

- 新建→项目→J2EE→Web Project
- Project Name:hibernate
- Source Folder:src
- Web Root Folder:web
- OK

开始Hibernate之旅



开始Hibernate之旅

- 增加对Struts的支持
- 添加Struts依赖的jar包

开始Hibernate之旅

- MyEclipse也同样提供了对Hibernate的支持，由于我们是第一次进行Hibernate项目的开发，所以所有工作我们完全手工完成来加强对Hibernate的理解

开始Hibernate之旅

- 首先需要在lib目录下导入Hibernate需要的所有jar包（很多）

开始Hibernate之旅

- 新建包：com.hibernate.util
- 在该包下新建类：HibernateUtil.java
- 编写该类的代码

开始Hibernate之旅

- 新建包com.hibernate.model
- 在该包下新建类：Person.java→持久化类
- 编写该类的代码

开始Hibernate之旅

- 新建包：com.hibernate.persistence
- 在该包下新建类：DBPerson.java→数据库访问类
- 编写该类的代码

创建持久化类

- 持久化类符合JavaBean的规范，包含一些属性，以及与之对应的getXXX()和setXXX()方法。
- 持久化类有一个id属性，用来惟一标识Person类的每个对象。在面向对象术语中，这个id属性被称为对象标识符（OID，Object Identifier）
- Hibernate要求持久化类必须提供一个不带参数的默认构造方法

开始Hibernate之旅

- 对Person.java文件创建一个Hibernate映射文件Person.hbm.xml
- Java的实体类是通过配置文件与数据表中的字段相关联。Hibernate在运行时解析配置文件，根据其中的字段名生成相应的SQL语句
- 将该文件存放在src目录下

<id>元素映射OID

- <generator>子元素用来设定标识符生成器。Hibernate提供了多种内置的实现。

标识符生成器	描述
increment	适用于代理主键。由 Hibernate 自动以递增的方式生成标识符，每次增量为 1。
identity	适用于代理主键。由底层数据库生成标识符。前提条件是底层数据库支持自动增长字段类型。
sequence	适用于代理主键。Hibernate 根据底层数据库的序列来生成标识符。前提条件是底层数据库支持序列。
hilo	适用于代理主键。Hibernate 根据 high/low 算法来生成标识符。Hibernate 把特定表的字段作为“high”值。默认情况下选用 hibernate_unique_key 表的 next_hi 字段。
native	适用于代理主键。根据底层数据库对自动生成标识符的支持能力，来选择 identity、sequence 或 hilo。
uuid.hex	适用于代理主键。Hibernate 采用 128 位的 UUID（Universal Unique Identification）算法来生成标识符。UUID 算法能够在网络环境中生成惟一的字符串标识符。这种标识符生成策略并不流行，因为字符串类型的主键比整数类型的主键占用更多的数据库空间。
assigned	适用于自然主键。由 Java 应用程序负责生成标识符，为了能让 Java 应用程序设置 OID，不能把 setId() 方法声明为 private 类型。应该尽量避免使用自然主键。

<property>元素映射值类型属性

- name属性：指定持久化类的属性的名字。
- type属性：指定Hibernate或Java映射类型。Hibernate映射类型是Java类型与SQL类型的桥梁。
- column属性：指定与类的属性映射的表的字段名。

Java类型、Hibernate映射类型以及SQL类型之间的对应关系

Java类型	Hibernate类型	Sql类型
java.lang.String	string	Varchar
int	int	int
char	character	char(1)
boolean	boolean	bit
java.lang.String	text	text
byte[]	binary	blob
java.sql.Date	date	date
java.sql.Timestamp	timestamp	timestamp

开始Hibernate之旅

- 编写Hibernate的描述文件 hibernate.cfg.xml
- Hibernate的描述文件中存放数据库连接驱动程序类，登陆数据库的用户名/密码，映射实体类配置文件的位置等信息。
- 将该配置文件存放在src目录下

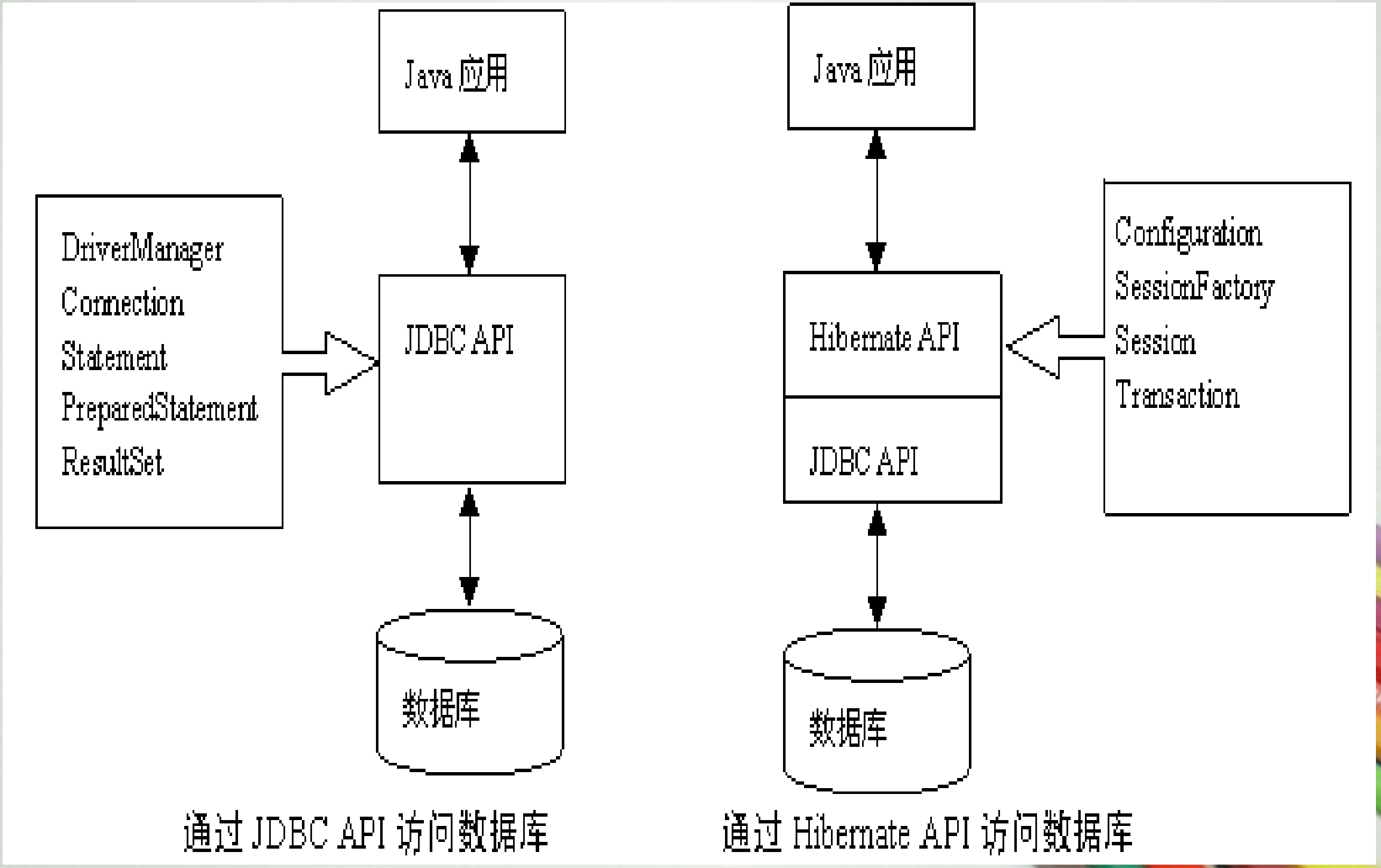
Hibernate配置文件的属性

属性	描述
hibernate.dialect	指定数据库使用的 SQL 方言
hibernate.connection.driver_class	指定数据库的驱动程序
hibernate.connection.url	指定连接数据库的 URL
hibernate.connection.username	指定连接数据库的用户名
hibernate.connection.password	指定连接数据库的口令
hibernate.show_sql	如果为 true，表示在程序运行时，会在控制台输出 SQL 语句，这有利于跟踪 Hibernate 的运行状态。默认为 false。在应用开发和测试阶段，可以把这个属性设为 true，以便跟踪和调试应用程序，在应用发布阶段，应该把这个属性设为 false，以便减少应用的输出信息，提高运行性能。

采用XML文件来配置对象-关系映射的优点

- Hibernate既不会渗透到上层域模型中，也不会渗透到下层数据模型中。
- 软件开发人员可以独立设计域模型，不必强迫遵守任何规范。
- 数据库设计人员可以独立设计数据模型，不必强迫遵守任何规范。
- 对象-关系映射不依赖于任何程序代码，如果需要修改对象-关系映射，只需修改XML文件，不需要修改任何程序，提高了软件的灵活性，并且使维护更加方便。

通过Hibernate API操纵数据库



SessionFactory接口

- 一个SessionFactory实例对应一个数据存储源，应用从SessionFactory中获得Session实例。SessionFactory有以下特点：
  - 它是线程安全的，这意味着它的同一个实例可以被应用的多个线程共享。
  - 它是重量级的，这意味着不能随意创建或销毁它的实例。如果应用只访问一个数据库，只需要创建一个SessionFactory实例，在应用初始化的时候创建该实例。如果应用同时访问多个数据库，则需要为每个数据库创建一个单独的SessionFactory实例。

Session接口

- Session接口是Hibernate应用使用最广泛的接口。
- Session也被称为持久化管理器，它提供了和持久化相关的操作，如添加、更新、删除、加载和查询对象。
- Session有以下特点：
  - 不是线程安全的，因此在设计软件架构时，应该避免多个线程共享同一个Session实例。
  - Session实例是轻量级的，所谓轻量级是指它的创建和销毁不需要消耗太多的资源。这意味着在程序中可以经常创建或销毁Session对象，例如为每个客户请求分配单独的Session实例，或者为每个工作单元分配单独的Session实例。

注意：此Session非彼Session(HttpSession)

Session接口操纵数据库的方法

- Session接口提供了操纵数据库的各种方法，如：
  - save()方法：把Java对象保存数据库中。
  - update()方法：更新数据库中的Java对象。
  - delete()方法：把Java对象从数据库中删除。
  - get()方法：从数据库中加载Java对象。

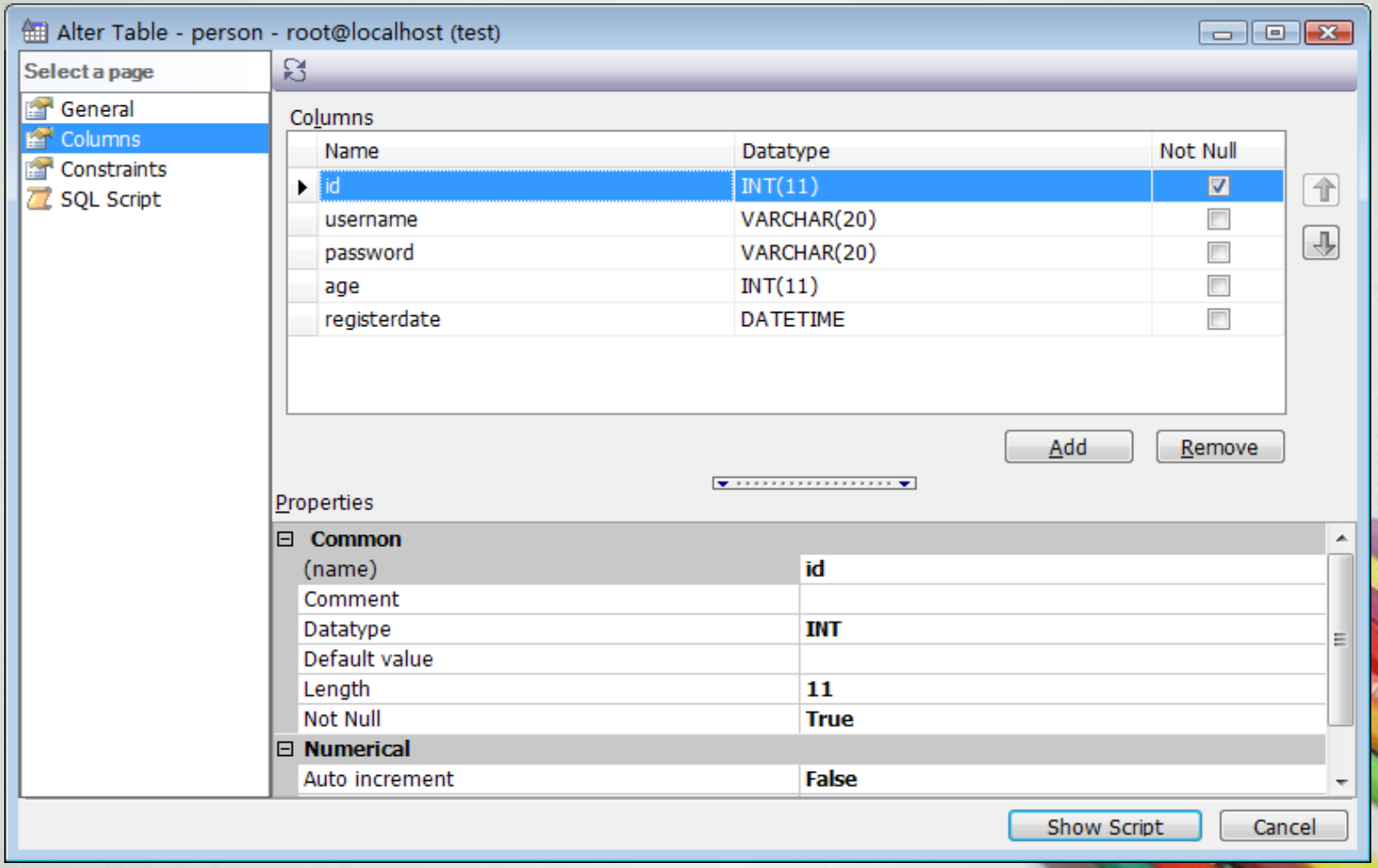
用Session来执行事务的流程

```
Session session = factory.openSession();
 Transaction tx;
 try {
    //开始一个事务
    tx = session.beginTransaction();
    //执行事务
    ...
    //提交事务
    tx.commit();
 }
 catch (Exception e) {
    //如果出现异常，就撤销事务
    if (tx!=null) tx.rollback();
    throw e;
 }
 finally {
    //不管事务执行成功与否，最后都关闭Session
    session.close();
 }
```

开始Hibernate之旅

- 创建数据库:hibernate
- 建立表：person
- 设计表的字段(注意：不将id设置为自动增加类型，让hibernate为我们做这些，体会hibernate的作用)

开始Hibernate之旅



开始Hibernate之旅

- 新建页面文件：register.jsp

开始Hibernate之旅

编写页面文件：listAll.jsp代码，用来显示注册用户的信息

开始Hibernate之旅

- 将MySql数据库驱动类放到lib目录下

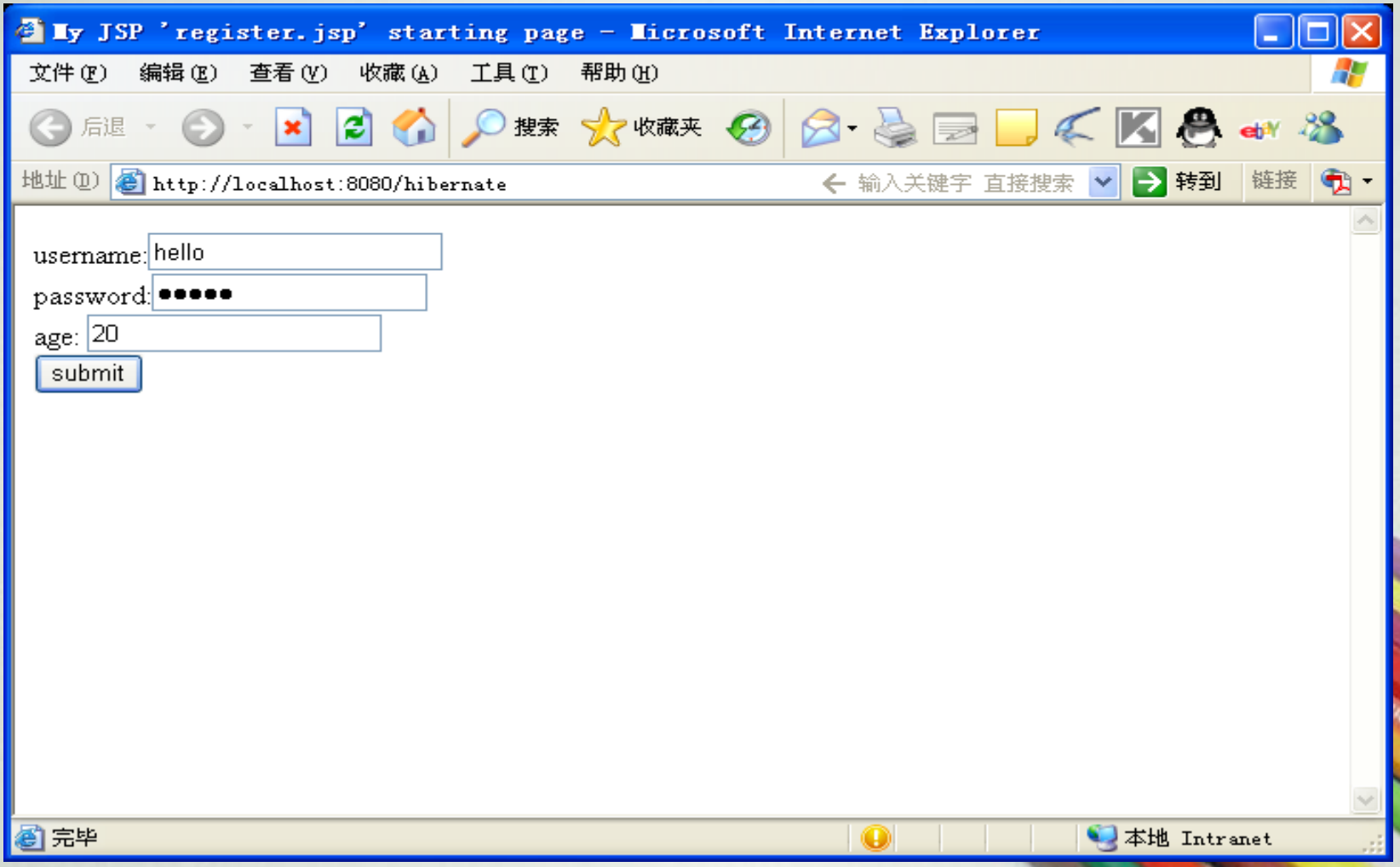
开始Hibernate之旅

- 配置好Tomcat的server.xml文件，增加一个Context path
- <Context path="/hibernate" reloadable="true" docBase="D:\hibernate\hibernate\web" />

开始Hibernate之旅

- 配置我们应用的web.xml，在文件的最后增加如下xml片段
- <welcome-file-list>
  <welcome-file>register.jsp</welcome-file>
  </welcome-file-list>
- 在Eclipse中启动Tomcat
- 输入如下网址：http://localhost:8080/hibernate

开始Hibernate之旅



开始Hibernate之旅

- 观察Tomcat命令行的输出
  - 显示Hibernate执行的sql语句
- 修改PersonAction.java代码，增加修改部分
- 修改struts.xml文件，增加相应的action片段
- 增加updateUser.jsp页面

开始Hibernate之旅

- 修改PersonAction.java代码，增加查看部分
- 修改struts.xml代码，增加相应的action片段
- 增加listUser.jsp页面

开始Hibernate之旅

- 修改PersonAction.java代码，增加删除部分
- 修改struts.xml代码，增加相应的action片段

开始Hibernate之旅