Hibernate入门

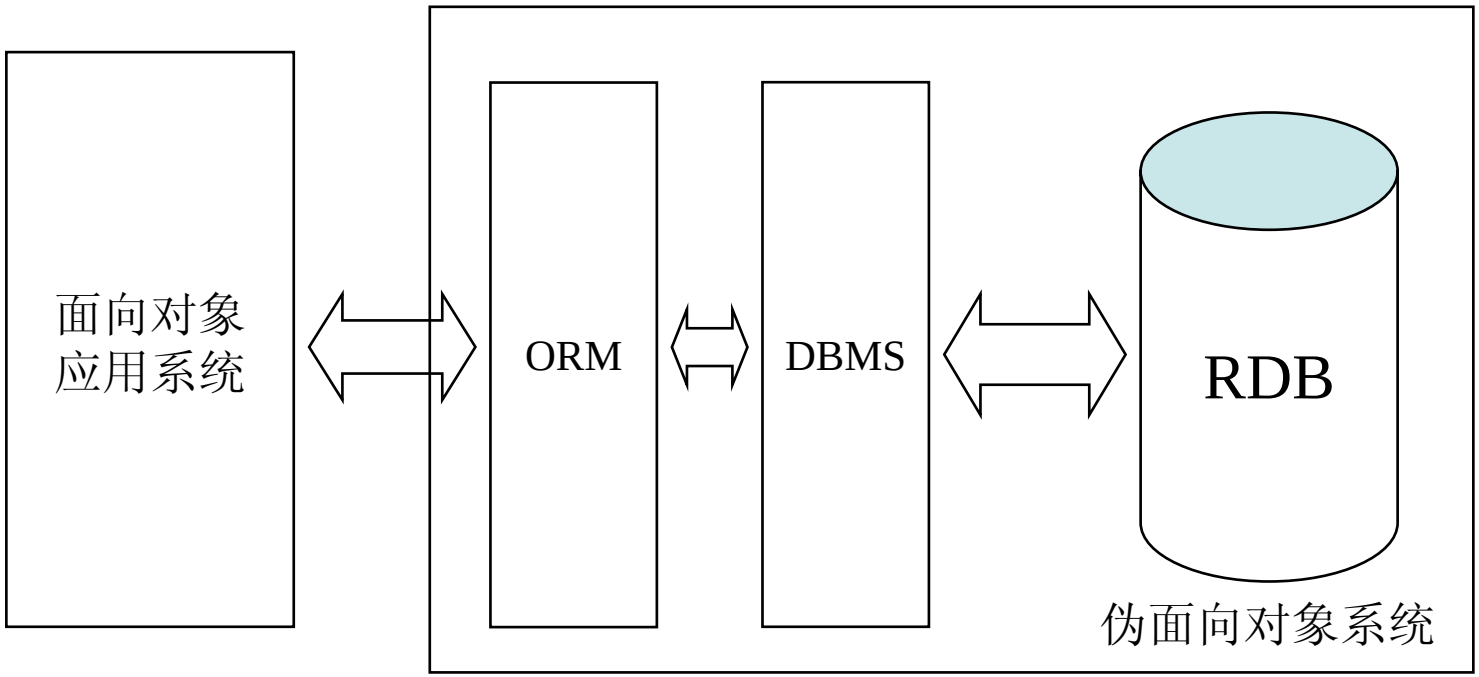
Hibernate作用

- ✓ 直接使用JDBC操作数据库的步骤很繁琐
- ✓ JDBC操作的是关系型数据库
- ✓ 我们用JAVA开发程序，则使用面向对象的思想
- ✓ Hibernate正是在这两种不同的模型之间建立关联，Hibernate给我们提供了利用面向对象的思想来操作关系型数据的接口
- ✓ Hibernate是基于伪面向对象数据库
- ✓ Hibernate主要的作用是做我们项目中的数据持久层
- ✓ Hibernate是ORM技术中的其中一种技术

ORM概念

- ✓ Object Relational Mapping（对象关系映射）
- ✓ ORM是一种为了解决面向对象与关系数据库存在的互不匹配的现象的技术。简单的说，ORM是通过使用描述对象和数据库之间映射的元数据，将java程序中的对象自动持久化到关系数据库中。本质上就是将数据从一种形式转换到另外一种形式
- ✓ 字母O起源于“对象”(Object),而R则来自于“关系”(Relational)。几乎所有的程序里面，都存在对象和关系数据库。在业务逻辑层和呈现层中，我们是面向对象的。当对象信息发生变化的时候，我们需要把对象的信息保存在关系数据库中
- ✓ 当你开发一个应用程序的时候(不使用O/R Mapping),你可能会写不少数据访问层的代码，用来从数据库保存，删除，读取对象信息，等等。而这些代码写起来总是重复的

伪面向对象数据库系统



Hibernate简单介绍

Hibernate 是一种对JDBC做了轻量级封装的对象-关系映射工具.所谓轻量级封装,是指Hibernate并没有完全封装JDBC,Java应用既可以通过Hibernate API访问数据库,还可以绕过Hibernate API,直接通过JDBC API来访问数据库.

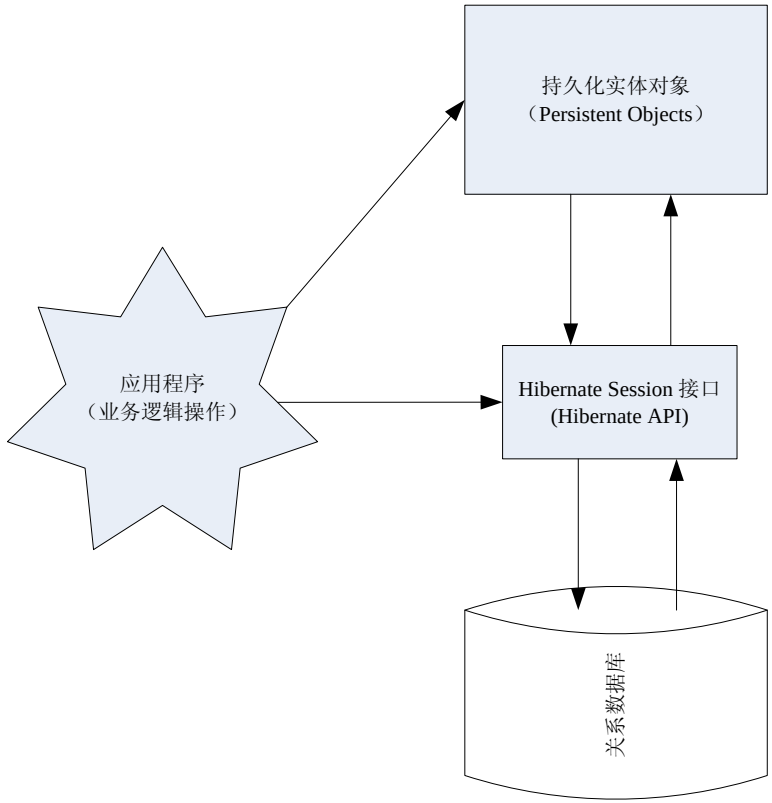
1.提供访问数据库的操作(如保存,更新,删除和查询对象)的接口.这些接口包括:Session、Transaction和Query接口。

2.用于配置Hibernate 的接口:Configuration.

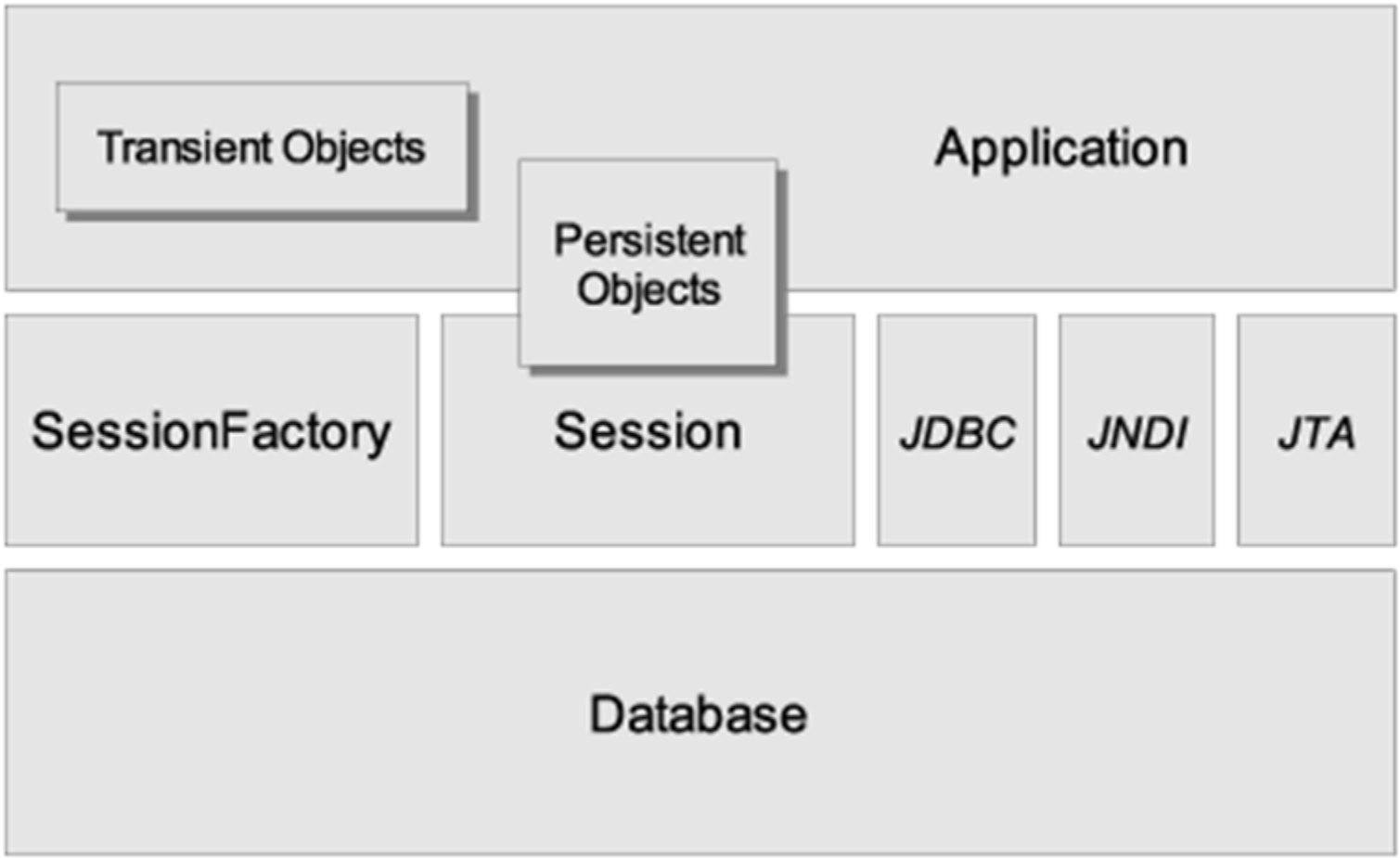
3.回调接口,使应用程序接受Hibernate内部发生的事件,并作出相关的回应.这些接口包括:Interceptor、Lifecycle和Validatable接口.

4.用于扩展Hibernate的功能的接口,如UserType、CompositeUserType和IdentifierGenerator接口。如果需要的话,应用程序可以扩展这些接口.

Hibernate与O、R之间的关系



Hibernate体系架构



Hibernate核心接口

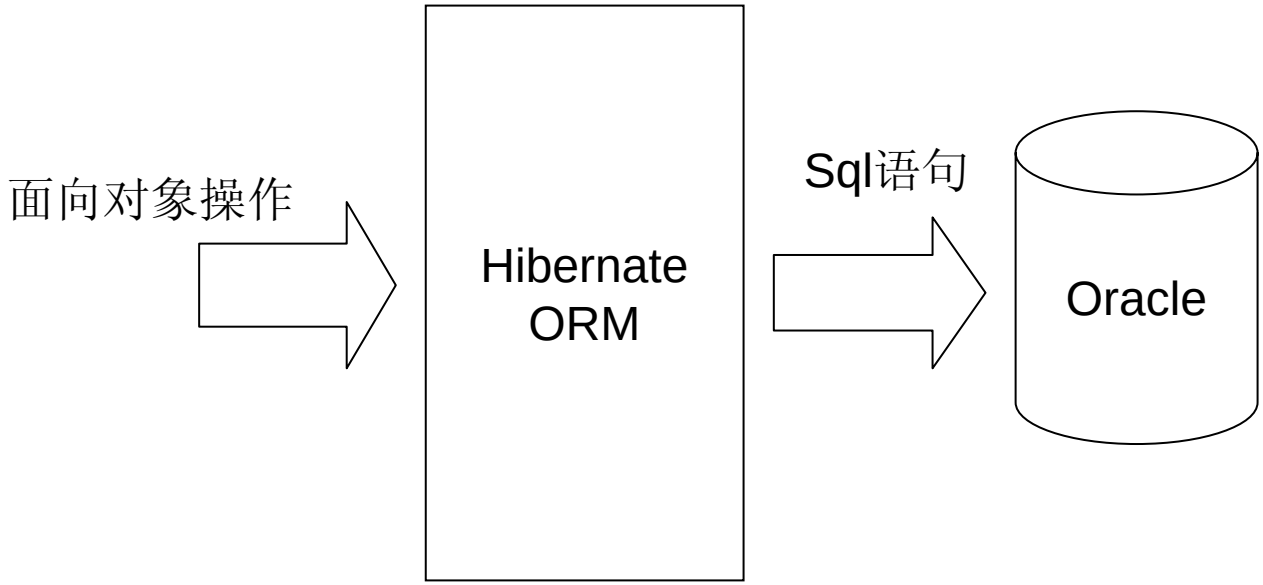
- ✓ 1.Configuration接口:Configuration对象用于配置并启动Hibernate .Hibernate应用通过Configuration实例来指定对象-关系映射文件的位置或者动态配置Hibernate的属性,然后创建SessionFactory实例.
- ✓ 2.SessionFactory接口: 一个SessionFactory实例对应一个数据存储源,应用从SessionFactory中获得Session实例.
- ✓ 3.Session接口:是Hibernate应用使用最广泛的接口.Session也被称为持久化管理器,它提供了和持久化相关的操作,如添加、更新、删除、加载和查询对象。
- ✓ 4.Transaction接口:是Hibernate的数据库事务接口,他对底层事务的接口进行了封装.
- ✓ 5.Query和Criteria接口:都是Hibernate的查询接口,用于向数据库查询对象,以及控制执行查询的过程.Query实例包装了HQL(Hibernate Query Language)查询语句.HQL查询语句是面向对象的,它引用类名和类的属性名,不是表名和表的字段名.Criteria接口

配置Hibernate开发环境

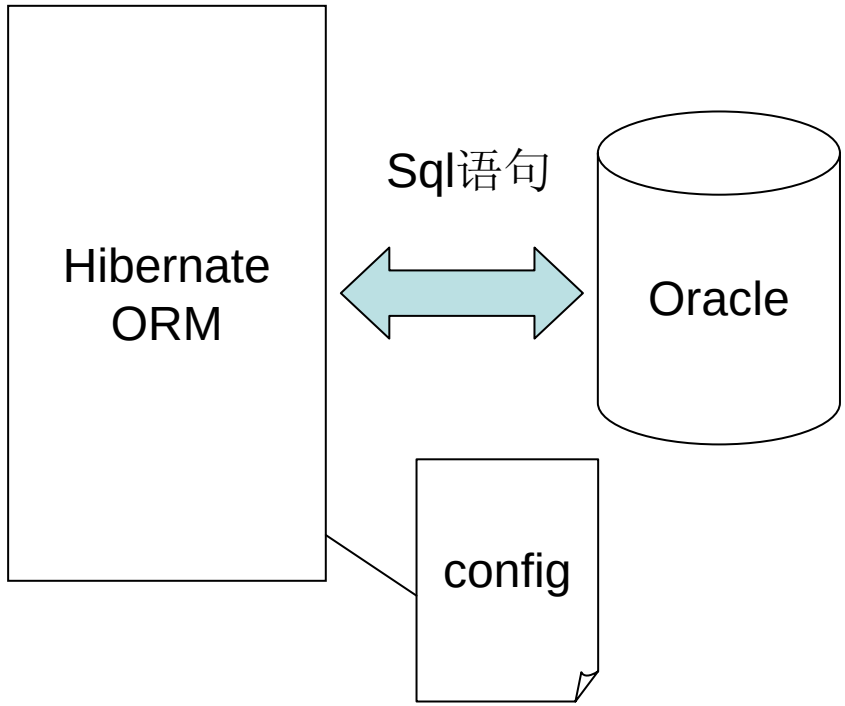
- ✓ 使用eclipse创建新项目，即可以是JAVA SE项目，也可以是JAVA EE项目，Hibernate本身和WEB无关
- ✓ 引入Hibernate及其依赖库（jar包）
- ✓ 引入Oracle(或者其他数据库)的驱动程序包，例如ojdbc14.jar
- ✓ 创建Hibernate配置文件 hibernate.cfg.xml

Hibernate的执行原理

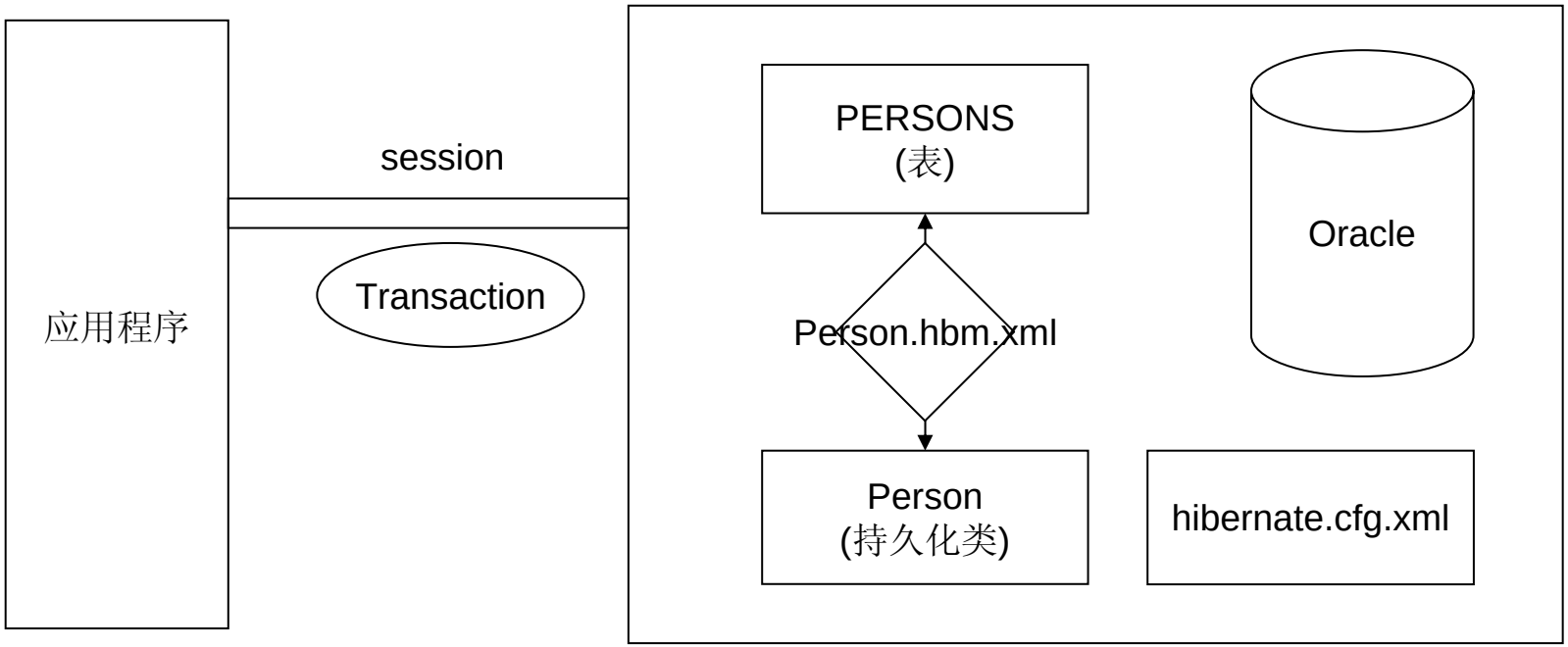
- ✓ 核心为一个HQL到SQL的实时编译器
- ✓ 对象缓存与同步组件



Hibernate基本配置



Hibernate伪对象数据库



一个Hibernate应用

- ✓ 基本配置文件
- ✓ 创建数据库表
- ✓ 创建持久化类
- ✓ 创建对象-关系映射文件
- ✓ 通过Hibernate API编写代码访问数据库

创建示例数据库用户

```
--创建用户
create user hb1 identified by hb1

--授予权限
grant connect,resource to hb1;
```

创建示例emp表

```
create table emp
(
 id number(6) primary key,  --oid，对象唯一标识
 empname varchar2(50), --员工姓名
 salary number(8,2), --员工工资
 hiredate date --入职日期
);
```

Hibernate配置文件

- ✓ 一个hibernate项目有一个配置文件
- ✓ 配置文件放置在src根目录下
- ✓ 配置文件可以是以下两种类型
  - ➢ hibernate.properties 不常用
  - ➢ hibernate.cfg.xml 常用
- ✓ 配置文件对应Configuration对象

配置文件的示例

hibernate.cfg.xml

```
<hibernate-configuration>
    <session-factory>
        <property name="connection.username">hb1</property>
        <property name="connection.url">jdbc:oracle:thin:@localhost:1521:orcl</property>
        <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property name="myeclipse.connection.profile">myoracle</property>
        <property name="connection.password">hb1</property>
        <property
      name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
        <property name="show_sql">true</property>
    </session-factory>
</hibernate-configuration>
```

配置文件解释

- ✓ <hibernate-configuration>声明Hibernate配置文件的开始
- ✓ <session-factory>表明以下的配置是针对session-factory（session工厂）配置的，SessionFactory是Hibernate中的一个类，这个类主要负责保存HIbernate的配置信息，以及对Session的操作
- ✓ <property>各种参数详细配置
  - ➢ connection.url 连接url
  - ➢ dialect 数据库方言，由于不同厂家的SQL语句不是完全相同，所以要指定使用的数据库方言类型，例如org.hibernate.dialect.Oracle10gDialect表示使用的是oracle 10g的数据库SQL语句
  - ➢ connection.driver_class数据库驱动类
  - ➢ connection.username 用户名
  - ➢ connection.password 密码
  - ➢ show_sql 设置为true表示运行时在控制台打印输出SQL以便于调试

创建实体类Emp.java

```
package com.pojo;
import java.util.Date;
public class Emp implements java.io.Serializable {
	private Integer id;
	private String empname;
	private Double salary;
	private Date hiredate;

	public Emp() {
	}
	public Emp(Integer id) {
		this.id = id;
	}
	public Emp(Integer id, String empname, Double salary, Date hiredate) {
		this.id = id;
		this.empname = empname;
		this.salary = salary;
		this.hiredate = hiredate;
	}

	get和set方法略
}
```

创建类的映射文件Emp.hbm.xml

```
<?xml version="1.0" encoding="utf-8"?>
<hibernate-mapping>
    <class name="com.pojo.Emp" table="EMP">
        <id name="id" type="java.lang.Integer">
            <column name="ID" precision="6" scale="0" />
            <generator class="increment" />
        </id>
        <property name="empname" type="java.lang.String">
            <column name="EMPNAME" length="50" />
        </property>
        <property name="salary" type="java.lang.Double">
            <column name="SALARY" precision="8" />
        </property>
        <property name="hiredate" type="java.util.Date">
            <column name="HIREDATE" length="7" />
        </property>
    </class>
</hibernate-mapping>
```

类的映射文件第二种格式

```
<hibernate-mapping>
    <class name=" com.pojo.Emp " table="emp">
        <id name="id" type="java.lang.Integer" column="id">
            <generator class="increment"/>
        </id>
        <property name="empname" type="java.lang.String" column="empname"/>
        <property name="salary" type="java.lang.Double" column="salary"/>
        <property name="hiredate" type="java.util.Date" column="hiredate"/>
    </class>
</hibernate-mapping>
```

映射文件中的核心标签和属性解释

- ✓ <hibernate-mapping>配置文件根标签
- ✓ <class>针对于一个OR的映射配置，通俗的说就是针对于一个类和表的映射配置，常用属性如下
  - ➢ name 映射的完整类名
  - ➢ table 映射的表名，如果类名和表名一致则可以省略
- ✓ <id>对象唯一标识，每个实体类都必须有唯一标识属性，例如员工id，部门id，常用属性如下：
  - ➢ name 类中唯一标识属性名
  - ➢ type 属性类型，如果和表中列类型一致，则可以省略
  - ➢ column 属性映射的表中列名，如果属性名称和列名一致，则可以省略
- ✓ <generator>生成唯一的标识的方式，class属性可以设置为assigned，increment，identity，sequence，native，uuid
- ✓ <property> 类中其他属性配置，其中name标识属性名，type,column属性同<id>中属性一致

将类的映射文件加入Hibernate

- ✓ 为了让Hibernate能够处理User对象的持久化，需要将它的映射信息加入到Hibernate中，让Hibernate能够找到实体和关系的映射信息
- ✓ 加入的方法很简单，在Hibernate配置文件中加入

```
<hibernate-configuration>
    <session-factory>
        ……
        <mapping resource="com/pojo/Emp.hbm.xml" />
    </session-factory>
</hibernate-configuration>
```

- ✓ resource属性指定了映射文件的位置和名称，注意路径之间使用斜杠分隔而不是用点，路径从顶层包开始

在做持久化对象操作之前需要创建的对象

```
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class Test {
    public static void main(String[] args) {
        //创建Hibernate配置对象
        //一个应用只需要一个Configuration对象实例
        Configuration  config = new Configuration().configure();

        //通过Configuration对象获得session工厂对象
        //一个应用只需要一个SessionFactory对象实例
        SessionFactory factory = config.buildSessionFactory();

        //通过SessionFactory对象获得一个Session对象
        //可以认为session对象就是JDBC中的一个Connection对象
        Session session = factory.openSession();
    }
}
```

通过HibernateSessionFactory类获得Session

```
import org.hibernate.Session;
import com.HibernateSessionFactory;

public class Test {

    public static void main(String[] args) {

        Session session = HibernateSessionFactory.getSession();

    }

}
```

Hibernate典型调用模式

```
Session sess = factory.openSession();//获得Session
Transaction tx = null;//声明事务对象
try {
    tx = sess.beginTransaction();//开启事务
     //执行增删改查
     tx.commit();//提交事务
}
catch (Exception e) {
    if (tx!=null) tx.rollback();//如果有异常就回滚事务，并且抛出异常
    throw e;
}finally {
    sess.close();//关闭Session
}
```

操作实体对象的方法

- ✓ Session中提供了对实体对象的操作方法：
  - save()方法——插入数据
  - update()方法——更新数据
  - delete()方法——删除数据
  - load()方法——返回数据
  - get()方法——返回数据

save()方法

- ✓ Session的save()用于将一个临时对象转变为持久化对象，也就是将一个新的业务实体保存到数据库中，其语法格式如下：
  - session.save(持久化对象);
- ✓ 通过Session的save()方法将持久化对象保存到数据库的具体步骤如下：
  - 根据持久化类对应映射文件指定的标识符生成方式为临时对象分配一个惟一的OID并赋值给持久化类的id属性。
  - 将临时对象加载到缓存中，使之转变为持久化对象。
  - 当提交事务时清理缓存，利用持久化对象包含的信息生成insert语句，真正的将持久化对象包含的信息保存到数据库中。

插入数据示例

```
public class TestInsert {
	public static void main(String[] args) {
		Session session = HibernateSessionFactory.getSession();//获得Session
		Transaction tx = session.beginTransaction();//开启事务
		try {
			//创建实体对象
			Emp emp = new Emp();
			emp.setEmpname("张三");
			emp.setSalary(2400.0);
			emp.setHiredate(new Date());
			//持久化实体对象插入数据
			session.save(emp);
			//提交事务
			tx.commit();
		} catch (Exception e) {
			e.printStackTrace();
			if (tx == null)
				tx.rollback();//回退事务
		} finally {
			session.close();//关闭连接
		}

	}
}
```

update()方法

- ✓ Session的update()方法用于将一个游离对象重新转变为持久化对象，也就是更新一个已经存在的业务实体到数据库中，其语法格式如下：
  - ➢ session. update(持久化对象);

更新数据示例

```
public class TestUpdate {
    public static void main(String[] args) {
        Session session = HibernateSessionFactory.getSession();//获得Session
        Transaction tx = session.beginTransaction();//开启事务
        try {
            //创建实体对象
            Emp emp = new Emp(1,"张三",3500.0,new Date());

            //持久化实体对象更新数据
            session.update(emp);
            //提交事务
            tx.commit();
        } catch (Exception e) {
            e.printStackTrace();
            if (tx == null)
                tx.rollback();//回退事务
        } finally {
            session.close();//关闭连接
        }
    }
}
```

delete()方法

- ✓ delete()方法的作用是删除与传入的持久化对象对应的数据库当中的记录，传入持久化类对象可以是持久化状态的，也可以是游离状态的，但不可以是临时状态的。如果是持久化状态的，则在销毁Session清理缓存时执行delete语句；如果是游离状态的，首先是将游离状态的对象转变为持久化状态，然后在销毁Session清理缓存时执行delete语句。
- ✓ Session的delete()方法的语法格式如下：
  - session.delete(持久化对象);

删除数据示例

```
public class TestDelete {
    public static void main(String[] args) {
        Session session = HibernateSessionFactory.getSession();//获得Session
        Transaction tx = session.beginTransaction();//开启事务
        try {
            //创建实体对象
            Emp emp = new Emp(1);
            //持久化实体对象删除数据
            session.delete(emp);
            //提交事务
            tx.commit();
        } catch (Exception e) {
            e.printStackTrace();
            if (tx == null)
                tx.rollback();//回退事务
        } finally {
            session.close();//关闭连接
        }

    }
}
```

Session中的查询方法

- ✓ 在Hibernate3.0以上的版本，提供了二种通过OID检索方式，一种是通过Session的load()方法实现，另一种是通过Session的get()方法实现。这二个方法都是通过指定的OID加载对应的持久化对象，首先是检索缓存，在没有检索到的情况下才检索数据库，但是他们在使用的细节上着如下细微的差别。
- ① 当数据库不存在与指定OID对应的持久化对象时，load()方法会抛出异常，而get()方法则返回null;
- ② get()方法总是立即加载对象，而load()方法支持延迟加载。

- 延迟加载就是并不立即执行SQL语句，而是在使用到持久化对象的数据时再执行SQL语句返回数据

根据ID返回一条记录示例

```
public class TestGet {
	public static void main(String[] args) {
		Session session = HibernateSessionFactory.getSession();// 获得Session
		Transaction tx = session.beginTransaction();// 开启事务

		try {
			// 根据ID查询返回一条数据
			Emp emp = (Emp) session.get(Emp.class, 1);
			// 打印输出员工信息
			System.out.println(emp.getId() + "," + emp.getEmpname() + ","
					+ emp.getSalary() + "," + emp.getHiredate());
			// 提交事务
			tx.commit();
		} catch (Exception e) {
			e.printStackTrace();
			if (tx == null)
				tx.rollback();// 回退事务
		} finally {
			session.close();// 关闭连接
		}

	}
}
```

更新数据示例2

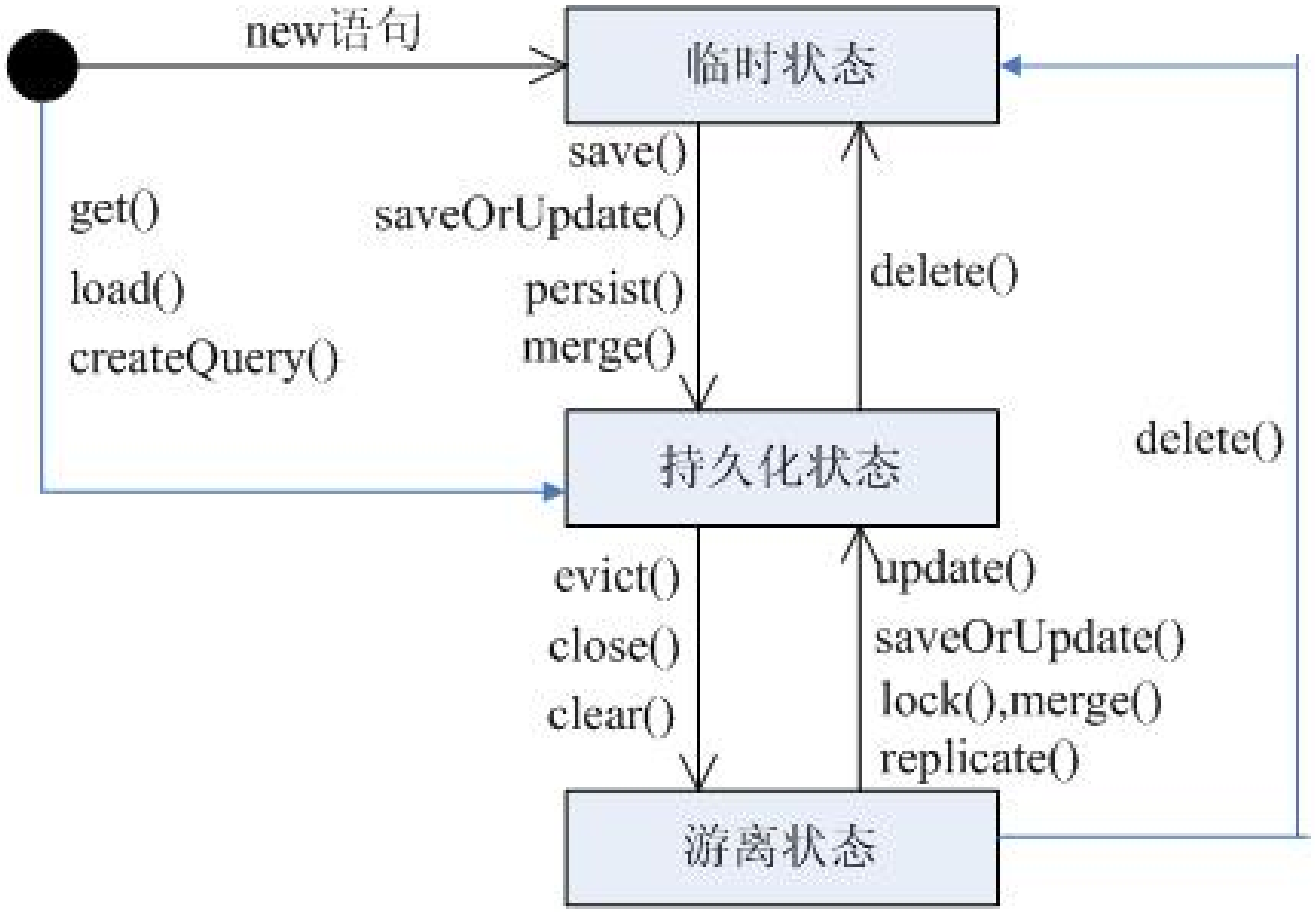
```
public class TestUpdate2 {
	public static void main(String[] args) {
		Session session = HibernateSessionFactory.getSession();// 获得Session
		Transaction tx = session.beginTransaction();// 开启事务
		try {
			//根据ID查询返回一条数据
			Emp emp = (Emp) session.get(Emp.class, 1);
			//修改属性值
			emp.setSalary(4600.0);
			// 提交事务
			tx.commit();
		} catch (Exception e) {
			e.printStackTrace();
			if (tx == null)
				tx.rollback();// 回退事务
		} finally {
			session.close();// 关闭连接
		}

	}
}
```

实体对象的三种状态

- ✓ 对于需要被持久化的Java对象，在它的生命周期中，可处于三个状态之一。
  - ➢ 临时状态(transient)：也称为瞬时状态，刚刚用new 语句创建，还没有被持久化， 不处于session的缓存中，处于临时状态下的Java对象被称为临时对象。临时状态的对象在表中没有匹配的记录。
  - ➢ 持久化状态(persistent)：已经被持久化，加入到session缓存中。处于持久化状态的Java对象被称为持久化对象。持久化状态的对象在表中有匹配的记录。
  - ➢ 游离状态(detached)：已经被持久化，但不再处于session的缓存中。处于游离状态下的对象被称为游离对象。当一个对象被持久化之后，在session关闭后就脱离了session的管理，但是在表中有匹配的记录，可以用session的方法重新建立这个对象和session的关联

三种状态图


new语句
临时状态
save()
saveOrUpdate()
persist()
merge()
get()
load()
createQuery()
delete()
持久化状态
delete()
evict()
close()
clear()
update()
saveOrUpdate()
lock(),merge()
replicate()
游离状态

三种状态示例

```
public class TestStatus {

    public static void main(String[] args) {

        Session session = HibernateSessionFactory.getSession();//获得Session
        Transaction tx = session.beginTransaction();//开启事务

        Emp emp = new Emp("李四",5500.0,new Date());//emp对象为临时状态

        session.save(emp);//emp对象为持久状态

        tx.commit();

        session.close();//emp对象为脱管状态

        System.out.println(emp);

    }

}
```

Query对象简单使用

- ✓ 使用session对象的get方法和load方法只能查询返回单条数据，如果想查询多条记录，需要使用Query接口
- ✓ 示例

```
public class TestQuery {
    public static void main(String[] args) {
        Session session = HibernateSessionFactory.getSession();// 获得Session
        Transaction tx = session.beginTransaction();// 开启事务

        String hql = "from Emp";// 注意Emp是类名不是表名，区分大小写
        Query query = session.createQuery(hql);// 创建Query对象传入HQL语句
        List list = query.list();// 执行HQL查询，返回数据形成一个List集合，元素是Object类型

        // 遍历集合
        for (Object obj : list) {
            Emp emp = (Emp) obj;//转换为Emp类型
            System.out.println(emp.getId() + "," + emp.getEmpname() + ","
                    + emp.getSalary() + "," + emp.getHiredate());
        }
        session.close();// 关闭连接
    }
}
```

正向工程和反向工程

- ✓ 做OR映射建模的时候有两种顺序
  - ➢ 先创建实体类和映射文件，然后通过正向工程生成表
  - ➢ 先创建表，然后通过反向工程生成实体类和映射文件
- ✓ 按照规范应该先建立实体对象然后再建立关系
- ✓ 而习惯性的是先建立关系然后建立实体对象
- ✓ 我的意思是：其实都行！

正向工程示例

```
package com;

import org.hibernate.cfg.Configuration;
import org.hibernate.tool.hbm2ddl.SchemaExport;

public class ExportDB {
    public static void main(String[] args) {

        // 读取hibernate.cfg.xml文件
        Configuration cfg = new Configuration().configure();
        //获得到处对象
        SchemaExport export = new SchemaExport(cfg);
        //导出生成表
        export.create(true, true);
    }
}
```

注意：此方法会删除掉所有表然后重新创建，以前如果有数据会全部丢失