CH01_Hibernate框架的搭建

1.Hibernate.cfg.xml:Hibermate的配置文件，数据库的配置信息

2.创建持久化类

3.创建持久化类的映射文件

4.通过hibernateAPI操控数据，进行数据持久化

Hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>

 <session-factory>
  <!-- 数据库相关信息-->
  <!--方言，不同数据有不同的方言  -->
  <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
  <!-- 驱动 -->
  <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
  <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/hibernate?
useUnicode=true&amp;characterEncoding=UTF-8</property><!-- 设置字符编码集 -->
  <!-- 关于中文乱码
  1.创建eclipse项目编码方式，项目上右键、properties查看
  2.创建 数据库UTF-8
  3.创建表的字符集UTF-8
  4.表的中文字段字符集UTF-8
  5.hibernate连接数据库的url设置编码方式-->
  <property name="hibernate.connection.username">root</property>
  <property name="hibernate.connection.password"></property>
  <!--可选， 在控制台打印格式化sql语句 -->
  <property name="hibernate.show_sql">true</property>
  <property name="hibernate.format_sql">true</property>

  <!-- 设置映射文件 -->
  <mapping resource="com/hibernate/entity/Customer.hbm.xml"/>
  <mapping resource="com/hibernate/entity/User.hbm.xml"/>
</session-factory>
</hibernate-configuration>
```

User.java

```
package com.hibernate.entity;

public class User {
    private Integer id;
    private String username;
    private String password;
```

```
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    @Override
    public String toString() {
        return "User [id=" + id + ", username=" + username + ", password=" + password + "]";
    }

}
```

User.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping package="com.hibernate.entity">
    <class name="User" table="user">
    <id name="id" column="id">
    <generator class="increment"></generator>
    </id>
    <!-- type:引用类型进行权限命名，基本类型直接写
    column当字段名与属性名一致，可省略
    access:
    Property：默认值，表明Hibernate通过set和get方法访问实体类的属性，适用于没有属性但有get和set方法的
实体类映射；
    Field：表明Hibernate通过Java反射机制直接访问类的属性。适用于没有get和set方法的实体类映射；-->
    <property name="username" column="username" type="java.lang.String" not-null="true">
</property>
    <property name="password" column="password" access="property"></property>
    </class>
</hibernate-mapping>
```

hibernateUtil.java

```
public class HibernateUtil {

    private static SessionFactory sessionFactory;

    /**
     * 静态初始化Hibernate
     */
    static {
```

```
        //1.创建ServiceRegistry对象,读取hibernate.cfg.xml在根目录下，则不用写参数
        StandardServiceRegistry registry = new StandardServiceRegistryBuilder()
                .configure().build();

        try {
            //2.创建SessionFactory对象，简单了解
            sessionFactory = new MetadataSources(registry).buildMetadata()
                    .buildSessionFactory();
        }catch (Exception e) {
            e.printStackTrace();
            //手动释放StandardServiceRegistry对象
            StandardServiceRegistryBuilder.destroy(registry);
        }
    }

    /**
     * 创建Session对象,session是与数据库之间的会话
     */
    public static Session openSession() {
        return sessionFactory.openSession();
    }

    /**
     * 关闭SessionFactory
     */
    public static void closeSessionFactory() {
        sessionFactory.close();
    }

}
```

Test.java

```
package com.hibernate.ui;

import org.hibernate.Session;
import org.hibernate.Transaction;

import com.hibernate.entity.User;
import com.hibernate.util.HibernateUtil;

public class UserTest {

    public static void main(String[] args) {
//      saveUser();
//      getUserById();
//      updateUser();
        deleteUser();
        HibernateUtil.closeSessionFactory();
    }
    public static void saveUser() {
        Session session=HibernateUtil.openSession();
        Transaction transaction=session.beginTransaction();
        try {
            User user=new User();
            user.setUsername("张三");
            user.setPassword("123");
            session.save(user);
            transaction.commit();
        } catch (Exception e) {
```

```
            transaction.rollback();
        }finally {
            session.close();
        }
    }

    public static void getUserById() {
        Session session = HibernateUtil.openSession();
        //get()通过主键查询
        User user=session.get(User.class, new Integer(1));
        System.out.println(user);
    }

    public static void updateUser() {
        Session session=HibernateUtil.openSession();
        Transaction transaction=session.beginTransaction();
        try {
            //查询得到需要更新的User
            User user=session.get(User.class, 1);
            user.setPassword("456");
            session.update(user);
            transaction.commit();
        } catch (Exception e) {
            transaction.rollback();
        }finally {
            session.close();
        }
    }

    public static void deleteUser() {
        Session session=HibernateUtil.openSession();
        Transaction transaction=session.beginTransaction();
        try {
            //查询得到需要删除的User
            User user=session.get(User.class, 1);
            //删除
            session.delete(user);
            transaction.commit();
        } catch (Exception e) {
            transaction.rollback();
        }finally {
            session.close();
        }
    }

}
```

CH02_单实体映射

基本映射

```
<hibernate-mapping package="com.hibernate.entity">
    <class name="User" table="user">
    <id name="id" column="id">
    <generator class="increment"></generator>
    </id>
    <!-- type:引用类型进行权限命名，基本类型直接写
    column当字段名与属性名一致，可省略-->
    <property name="username" column="username" type="java.lang.String" not-null="true">
</property>
    <property name="password" column="password"></property>
    </class>
</hibernate-mapp
```

当属性与字段不一一对应：增加getter、setter方式

```
//采用单独添加getter,setter方法，添加复合属性
    public String getUsername() {
        return lastName+" "+firstName;

    }
    public void setUsername(String userName) {
        String[] names=userName.split(" ");
        lastName=names[0];
        firstName=names[1];
    }
```

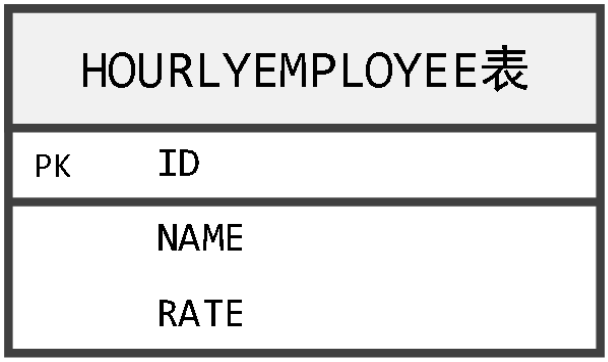
当属性有，而字段没有：增加formula Sql 语句

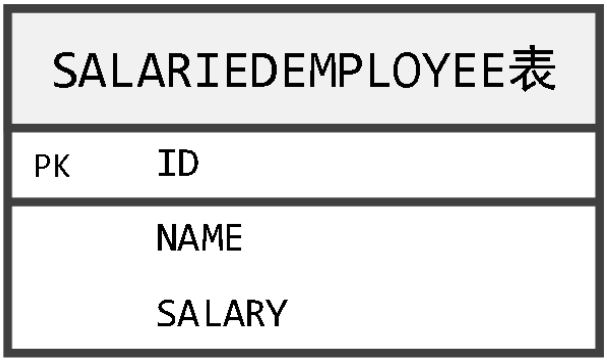
```
<!-- 把某一个属性映射到一条SQL语句
id为user表中的字段，因为现在的xml文件是user的映射文件-->
<property name="totalPrice" formula="(select sum(o.price) from orders as o where o.userId=id)">
</property>
```

注解配置

CH03_继承关系映射

每个具体类对应一整表





xml方式

```
<hibernate-mapping package="com.hibernate.entity">
    <class name="HourlyEmployee" table="hourly_employee">
    <id name="id"><generator class="identity"></generator></id>
    <property name="name"></property>
    <property name="rate"></property>
    </class>
</hibernate-mapping>
```

```
<hibernate-mapping package="com.hibernate.entity">
    <class name="SalariedEmployee" table="salaried_employee">
    <id name="id"><generator class="identity"></generator></id>
    <property name="name"></property>
    <property name="salary"></property>
    </class>
</hibernate-mapping>
```

注解方式

基类

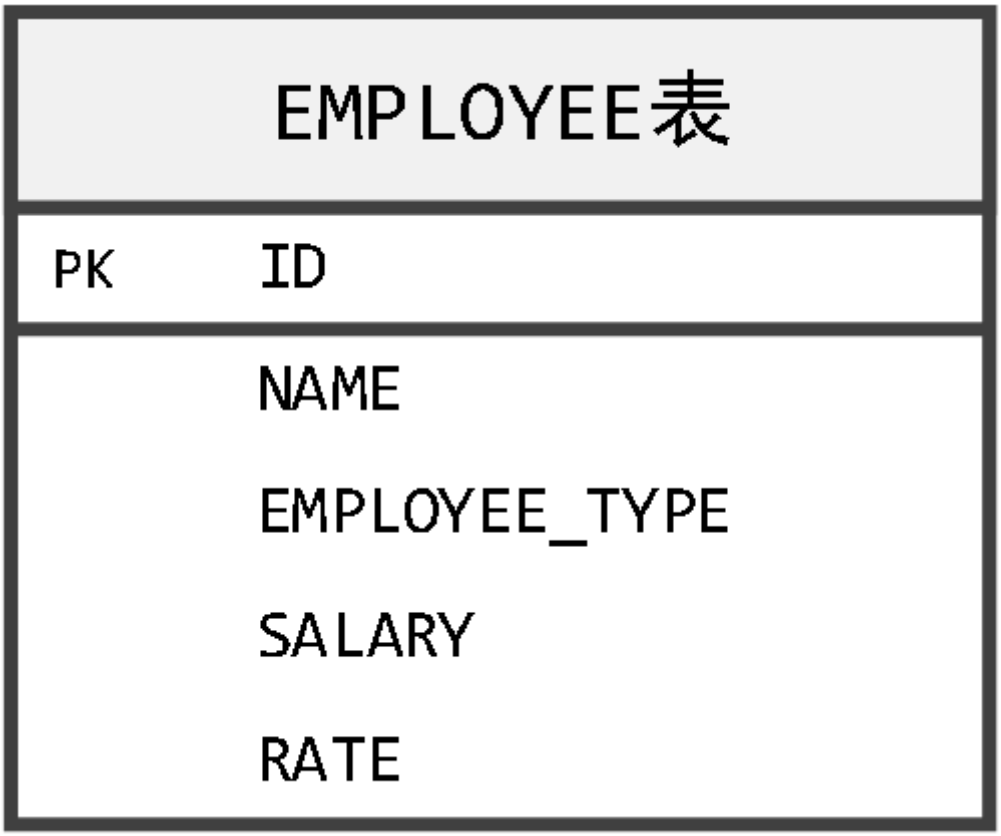
```
package com.hibernate.entity;

import javax.persistence.Column;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.MappedSuperclass;

import org.hibernate.annotations.GenericGenerator;

@MappedSuperclass
//不需要映射的基类配置
public class Employee {
    private Integer id;
    private String name;

    @Id
    @GeneratedValue(generator = "my_inc")
    @GenericGenerator(strategy = "increment",name="my_inc")
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }

    @Column
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
```

```
}
```

具体类

```
package com.hibernate.entity;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Table;

@Entity
@Table(name="hourly_employee")
public class HourlyEmployee extends Employee{
    private double rate;


    //属性名与字段名一致，不必添加@Column注解
    public double getRate() {
        return rate;
    }

    public void setRate(double rate) {
        this.rate = rate;
    }

    @Override
    public String toString() {
        return "HourlyEmployee [rate=" + rate + ",Id=" + getId() + ", name=" + getName() + "]";
    }

}
```

父类对应一张表



xml方式

```
<?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 package="com.hibernate.entity">
    <class name="Employee" table="employee">
    <id name="id">
    <generator class="increment"></generator>
    </id>
    <!-- 额外字段，区分某条记录属于那种 子类对象对应的数据，必须跟在<id>标签之下-->
    <discriminator column="employee_type"></discriminator>
    <property name="name"></property>
    <!-- 每个子类的映射 -->
    <subclass name="HourlyEmployee" discriminator-value="HE">
    <property name="rate"></property>
    </subclass>
    <subclass name="SalariedEmployee" discriminator-value="SE">
    <property name="salary"></property>
    </subclass>
    </class>
</hibernate-mapping>
```

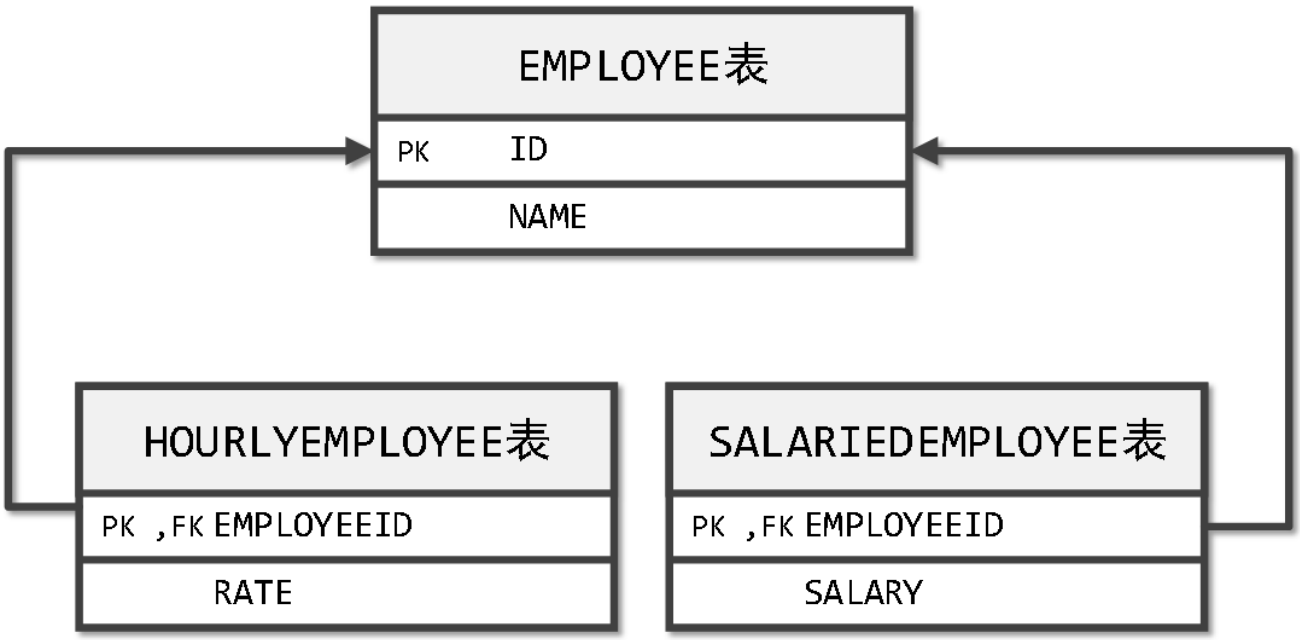
注解方式

```java
package com.hibernate.entity;

import javax.persistence.Column;
import javax.persistence.DiscriminatorColumn;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;
import javax.persistence.MappedSuperclass;

import org.hibernate.annotations.GenericGenerator;

@Entity
//继承关系的生成策略
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
//额外字段
@DiscriminatorColumn(name = "employee_type")
public class Employee {
    private Integer id;
    private String name;

    @Id
    @GeneratedValue(generator = "my_inc")
    @GenericGenerator(strategy = "increment",name="my_inc")
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }

    @Column
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

}
```

```java
package com.hibernate.entity;

import javax.persistence.Column;
import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;
import javax.persistence.Table;

@Entity
//额外字段的取值
@DiscriminatorValue(value = "HE")
public class HourlyEmployee extends Employee{
    private double rate;
```

```
    public double getRate() {
        return rate;
    }

    public void setRate(double rate) {
        this.rate = rate;
    }

    @Override
    public String toString() {
        return "HourlyEmployee [rate=" + rate + ",Id=" + getId() + ", name=" + getName() + "]";
    }

}
```

每个类对应一张表



xml方式

```
<hibernate-mapping package="com.hibernate.entity">
    <class name="Employee" table="employee">
    <id name="id">
    <generator class="increment"></generator>
    </id>
    <property name="name"></property>

    <!--映射子类-->
    <joined-subclass name="HourlyEmployee" table="hourly_employee">
     <!--hourly_id为子类表的为外键，参照父类表的主键-->
    <key column="hourly_id"></key>
    <property name="rate"></property>
    </joined-subclass>
```

```
    <joined-subclass name="SalariedEmployee" table="salaried_employee">
    <key column="salaried_id"></key>
    <property name="salary"></property>
    </joined-subclass>

    </class>
</hibernate-mapping>
```

注解方式

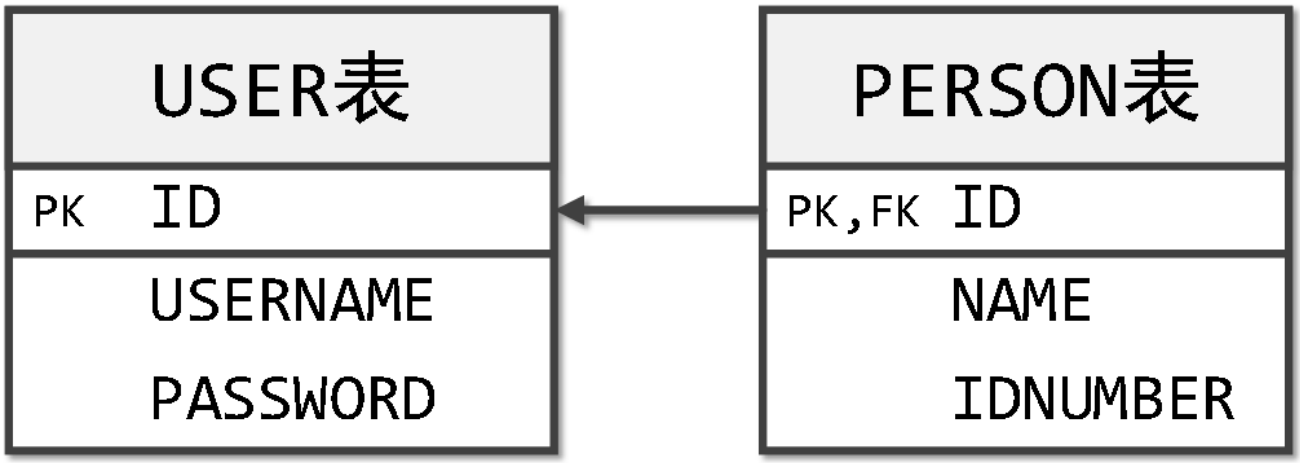
```
package com.hibernate.entity;

import javax.persistence.Column;
import javax.persistence.DiscriminatorColumn;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;
import javax.persistence.MappedSuperclass;

import org.hibernate.annotations.GenericGenerator;

@Entity
//继承关系的生成策略
@Inheritance(strategy = InheritanceType.JOINED)
public class Employee {
    private Integer id;
    private String name;

    @Id
    @GeneratedValue(generator = "my_inc")
    @GenericGenerator(strategy = "increment",name="my_inc")
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }

    @Column
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

}
```

```
package com.hibernate.entity;

import javax.persistence.Column;
import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;
```

```java
import javax.persistence.PrimaryKeyJoinColumn;
import javax.persistence.Table;

@Entity
@Table(name = "hourly_employee")
//制定子类表的主键列
@PrimaryKeyJoinColumn(name = "hourly_id")
public class HourlyEmployee extends Employee{
    private double rate;


    public double getRate() {
        return rate;
    }

    public void setRate(double rate) {
        this.rate = rate;
    }

    @Override
    public String toString() {
        return "HourlyEmployee [rate=" + rate + ",Id=" + getId() + ", name=" + getName() + "]";
    }

}
```

CH04_一对一关联映射与组合关系映射

主键关联-一对一方式1



注解

```
package com.hibernate.entity;

import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.OneToOne;
import javax.persistence.PrimaryKeyJoinColumn;

import org.hibernate.annotations.GenericGenerator;

@Entity
public class User {
    private Integer id;
    private String username;
    private String password;
    //关联到Person
    private Person person;

    @Id
    @GeneratedValue(generator = "my_gen")
    @GenericGenerator(strategy = "increment",name = "my_gen")
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }

    @OneToOne(cascade = CascadeType.ALL)//操作User的同时，级联操作Person
    //既是主键又是外键的person表列名
    @PrimaryKeyJoinColumn(name="id")
    public Person getPerson() {
        return person;
    }
    public void setPerson(Person person) {
        this.person = person;
    }
    @Override
    public String toString() {
        return "User [id=" + id + ", username=" + username + ", password=" + password + ",
person=" + person + "]";
    }

}
```

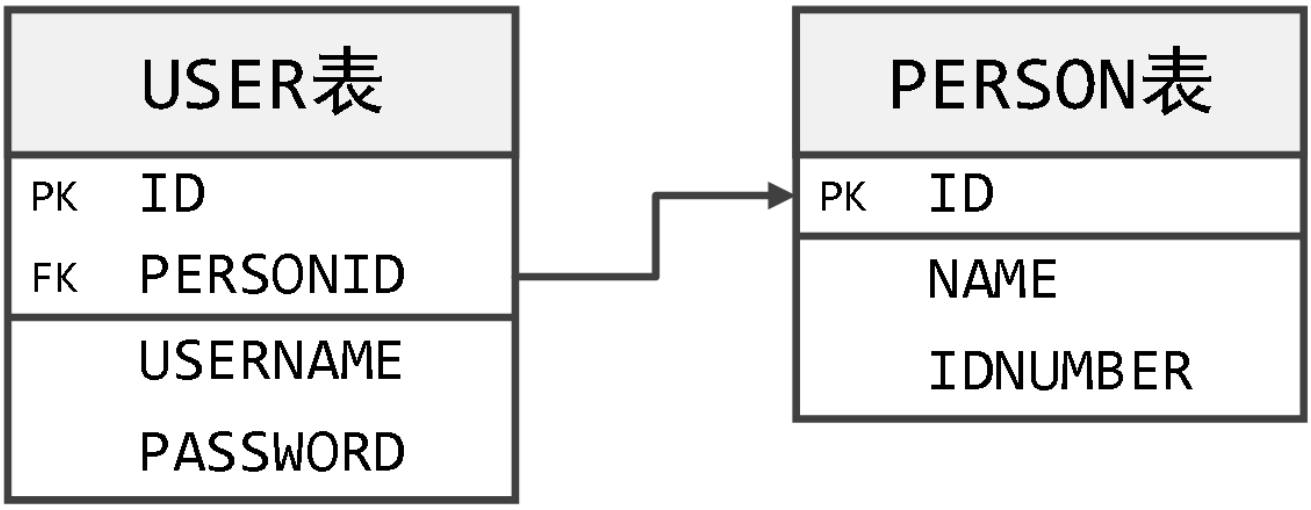
```java
package com.hibernate.entity;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.OneToOne;

import org.hibernate.annotations.GenericGenerator;
import org.hibernate.annotations.Parameter;

@Entity
public class Person {
    private Integer id;
    private String name;
    private String idNumber;//身份证号
    //关联到User
    private User user;

    @Id//表示OID属性
    @GeneratedValue(generator = "foreign")
    @GenericGenerator(strategy = "foreign",name = "foreign",
    parameters = {@Parameter(name="property",value = "user")})
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

    @Column(name = "id_number")
    public String getIdNumber() {
        return idNumber;
    }
    public void setIdNumber(String idNumber) {
        this.idNumber = idNumber;
    }

    //已经被person属性映射过了，注解映射只需要映射其中一方，另一方直接指定我在哪个属性上映射过了
    @OneToOne(mappedBy = "person")
    public User getUser() {
        return user;
    }
    public void setUser(User user) {
        this.user = user;
    }
    @Override
    public String toString() {
        return "Person [id=" + id + ", name=" + name + ", idNumber=" + idNumber + ", user=" +
user + "]";
    }

}
```

xml

```xml
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- package指定持久化类所在的包 -->
<hibernate-mapping package="com.hibernate.entity">
<!-- 类与表的映射 -->
    <class name="User" table="user">
        <id name="id" column="id" >
        <generator class="identity"></generator></id>
        <property name="username" column="username"  type="java.lang.String"></property>
        <property name="password" column="password"></property>

        <!-- 一对一关联,cascade表示user到person表的级联
           all:级联所有的操作
           null:所有操作都不级联
           save-update:保存或者更新级联
           delete:删除操作级联
         -->
        <one-to-one name="person" class="Person" cascade="all"></one-to-one>

    </class>
</hibernate-mapping>
```

```xml
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- package指定持久化类所在的包 -->
<hibernate-mapping package="com.hibernate.entity">
<!-- 类与表的映射 -->
    <class name="Person" table="person">
    <!-- 主键来自user表的id -->
        <id name="id" type="int" column="id" >
        <!-- 主键也是外键 -->
        <generator class="foreign">
        <!-- 当前主键来自某一个属性(user)的类型（User）所对应的表(user)的主键(id) -->
        <param name="property">user</param>
        </generator>
        </id>

        <property name="name" column="name"  type="java.lang.String"></property>
        <property name="idNumber" column="id_number"></property>

        <!-- constrained="true"如果是外键所对应的属性，必须加 -->
        <one-to-one name="user" class="User" constrained="true"></one-to-one>

    </class>
</hibernate-mapping>
```

唯一外键关联-一对一方式2



注解

```java
package com.hibernate.entity;

import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OneToOne;

import org.hibernate.annotations.GenericGenerator;

@Entity
public class User {
    private Integer id;
    private String username;
    private String password;
    //关联到Person
    private Person person;

    @Id
    @GeneratedValue(generator ="my_inc")
    @GenericGenerator(strategy = "increment",name = "my_inc")

    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
```

```
        this.username = username;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }

    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name="person_id")
    public Person getPerson() {
        return person;
    }
    public void setPerson(Person person) {
        this.person = person;
    }

    @Override
    public String toString() {
        return "User [id=" + id + ", username=" + username + ", password=" + password + ",
person=" + person + "]";
    }

}
```

```
package com.hibernate.entity;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.OneToOne;

import org.hibernate.annotations.GenericGenerator;

@Entity
public class Person {
    private Integer id;
    private String name;
    private String idNumber;//身份证号
    //关联到User
    private User user;

    @Id
    @GeneratedValue(generator ="my_inc")
    @GenericGenerator(strategy = "increment",name = "my_inc")

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

```
    @Column(name = "id_number")
    public String getIdNumber() {
        return idNumber;
    }
    public void setIdNumber(String idNumber) {
        this.idNumber = idNumber;
    }

    @OneToOne(mappedBy = "person")
    public User getUser() {
        return user;
    }
    public void setUser(User user) {
        this.user = user;
    }
    @Override
    public String toString() {
        return "Person [id=" + id + ", name=" + name + ", idNumber=" + idNumber + ", user=" +
user + "]";
    }

}
```

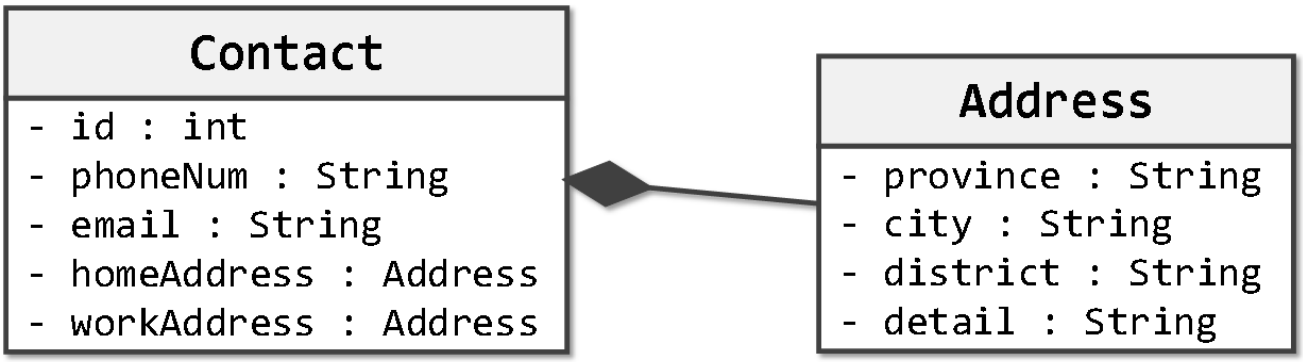
xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- package指定持久化类所在的包 -->
<hibernate-mapping package="com.hibernate.entity">
<!-- 类与表的映射 -->
    <class name="User" table="user">
        <id name="id" column="id" >
        <generator class="identity"></generator></id>
        <property name="username" column="username"  type="java.lang.String"></property>
        <property name="password" column="password"></property>

        <!-- many-to-one:使用唯一外键关联，实际是多对一的特殊情况
        unique="true"多对一关联,但是外键是唯一的，所以是一对一-->
        <many-to-one name="person" class="Person" unique="true" column="person_id"
cascade="all"></many-to-one>

    </class>
</hibernate-mapping>
```

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- package指定持久化类所在的包 -->
<hibernate-mapping package="com.hibernate.entity">
<!-- 类与表的映射 -->
    <class name="Person" table="person">
        <id name="id" type="int" column="id" >
        <generator class="identity">
        </generator>
```

```
        </id>
        <property name="name" column="name"  type="java.lang.String"></property>
        <property name="idNumber" column="id_number"></property>

        <!-- property-ref:外键列对应的属性名-->
        <one-to-one name="user" class="User" property-ref="person"></one-to-one>
    </class>
</hibernate-mapping>
```

组合关系映射



注解

```
package com.hibernate.entity;

import javax.persistence.Embeddable;

//嵌入类，这个类不需要单独映射
@Embeddable
public class Address {
    private String province;
    private String city;
    private String district;
    private String detail;
    public String getProvince() {
        return province;
    }
    public void setProvince(String province) {
        this.province = province;
    }
    public String getCity() {
        return city;
    }
    public void setCity(String city) {
        this.city = city;
    }
    public String getDistrict() {
        return district;
    }
    public void setDistrict(String district) {
        this.district = district;
    }
```

```
    public String getDetail() {
        return detail;
    }
    public void setDetail(String detail) {
        this.detail = detail;
    }
    @Override
    public String toString() {
        return "Address [province=" + province + ", city=" + city + ", district=" + district +
", detail=" + detail
                + "]";
    }

}
```

```
package com.hibernate.entity;

import javax.persistence.AttributeOverride;
import javax.persistence.AttributeOverrides;
import javax.persistence.Column;
import javax.persistence.Embedded;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

import org.hibernate.annotations.GenericGenerator;

@Entity
public class Contact {
    private Integer id;
    private String phone;
    private String email;
    private Address homeAddress;
    private Address workAddress;

    @Id
    @GeneratedValue(generator ="my_inc")
    @GenericGenerator(strategy = "increment",name = "my_inc")
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getPhone() {
        return phone;
    }
    public void setPhone(String phone) {
        this.phone = phone;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }

    //嵌入的属性
    @Embedded
```

```
    @AttributeOverrides(value= {
            @AttributeOverride(name = "province",column = @Column(name="home_province")),
            @AttributeOverride(name = "city",column = @Column(name="home_city")),
            @AttributeOverride(name = "district",column = @Column(name="home_district")),
            @AttributeOverride(name = "detail",column = @Column(name="home_detail")),
    })
    public Address getHomeAddress() {
        return homeAddress;
    }
    public void setHomeAddress(Address homeAddress) {
        this.homeAddress = homeAddress;
    }

    @Embedded
    @AttributeOverrides(value= {
            @AttributeOverride(name = "province",column = @Column(name="work_province")),
            @AttributeOverride(name = "city",column = @Column(name="work_city")),
            @AttributeOverride(name = "district",column = @Column(name="work_district")),
            @AttributeOverride(name = "detail",column = @Column(name="work_detail")),
    })
    public Address getWorkAddress() {
        return workAddress;
    }
    public void setWorkAddress(Address workAddress) {
        this.workAddress = workAddress;
    }
    @Override
    public String toString() {
        return "Contact [id=" + id + ", phone=" + phone + ", email=" + email + ", homeAddress="
+ homeAddress
                + ", workAddress=" + workAddress + "]";
    }

}
```

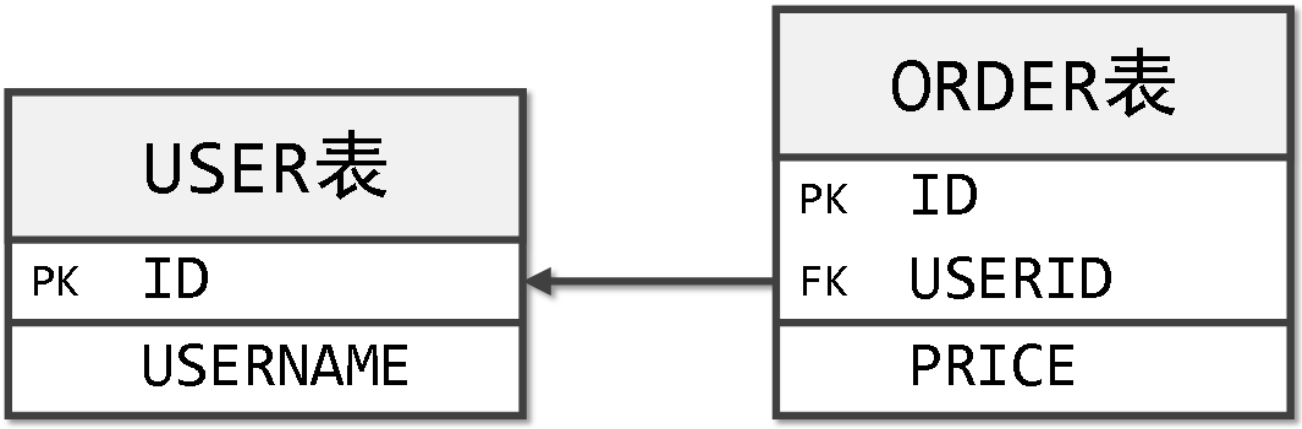
xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- package指定持久化类所在的包 -->
<hibernate-mapping package="com.hibernate.entity">
<!-- 类与表的映射 -->
    <class name="Contact" table="contact">
    <!-- 主键的映射 -->
        <id name="id" type="int" column="id" >
        <!-- 主键生成器 -->
        <generator class="identity"></generator>
        </id>
        <!-- 一般属性的映射 -->
        <property name="phone" column="phone"  type="java.lang.String"></property>
        <property name="email" column="email"></property>
        <!-- 组合关系中被包含类的映射 -->
        <component name="homeAddress" class="Address">
            <property name="province" column="home_province"></property>
            <property name="city" column="home_city"></property>
```

```
            <property name="district" column="home_district"></property>
            <property name="detail" column="home_detail"></property>
        </component>
        <component name="workAddress" class="Address">
            <property name="province" column="work_province"></property>
            <property name="city" column="work_city"></property>
            <property name="district" column="work_district"></property>
            <property name="detail" column="work_detail"></property>
        </component>
    </class>
</hibernate-mapping>
```

CH05_一对多关联映射

many参照one



单向一对多

注解

```
package com.hibernate.entity;

import java.util.HashSet;
import java.util.Set;

import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.OneToMany;

import org.hibernate.annotations.GenericGenerator;

@Entity
public class User {

    private Integer id;
```

```
    private String userName;
    private String password;
    //域模型中用来表示从用户能够导航到订单数据（从用户到订单的一对多关联）
    private Set orderSet = new HashSet<>();

    @Id
    @GeneratedValue(generator = "my_inc")
    @GenericGenerator(strategy = "increment",name = "my_inc")
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getUserName() {
        return userName;
    }
    public void setUserName(String userName) {
        this.userName = userName;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    @Override
    public String toString() {
        return "User [id=" + id + ", userName=" + userName + ", password=" + password + "]";
    }

    //映射一对多,需要特殊指定集合的属性的get方法的返回值Set的存储的属性类型，其他如getUser,直接返回User类型，所以不需要特殊指出
    //mappedby表示在Order类的user属性映射过了
    @OneToMany(mappedBy = "user",targetEntity = Order.class,cascade = CascadeType.REMOVE)
    public Set getOrderSet() {
        return orderSet;
    }
    public void setOrderSet(Set orderSet) {
        this.orderSet = orderSet;
    }


}
```

```
package com.hibernate.entity;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;

import org.hibernate.annotations.GenericGenerator;

@Entity
@Table(name="orders")
public class Order {
```

```
    private Integer id;
    private Double price;
    private User user;

    @Id
    @GeneratedValue(generator = "my_inc")
    @GenericGenerator(strategy = "increment",name = "my_inc")
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public Double getPrice() {
        return price;
    }
    public void setPrice(Double price) {
        this.price = price;
    }

    //映射多对一关系
    @ManyToOne
    @JoinColumn(name = "user_id")
    public User getUser() {
        return user;
    }
    public void setUser(User user) {
        this.user = user;
    }

    @Override
    public String toString() {
        return "Order [id=" + id + ", price=" + price + "]";
    }


}
```

xml重点

User.hbm.xml

```
    <class name="User" table="user">
        <id name="id" type="int" column="id">
            <generator class="identity"></generator>
        </id>
        <property name="userName" type="java.lang.String" column="username" ></property>
        <property name="password"/>

        <!-- 映射User与Order之间的一对多关系
            set用来映射Set类型的属性
            key:指定外键字段名字
            one-to-many表示一对多关系 -->
        <set name="orderSet" cascade="delete">
            <key column="USER_ID"/>
            <one-to-many class="Order"/>
```

```
        </set>

    </class>
```


```
<!-- 映射User与Order之间的一对多关系
            list用来映射List类型的属性
            key:指定外键字段名字
            index:指定保存插入顺序的字段
            one-to-many表示一对多关系 -->
        <list name="orderList" cascade="delete">
            <key column="USER_ID"/>
            <index column="ORDER_index"></index>
            <one-to-many class="Order"/>
        </list>
```

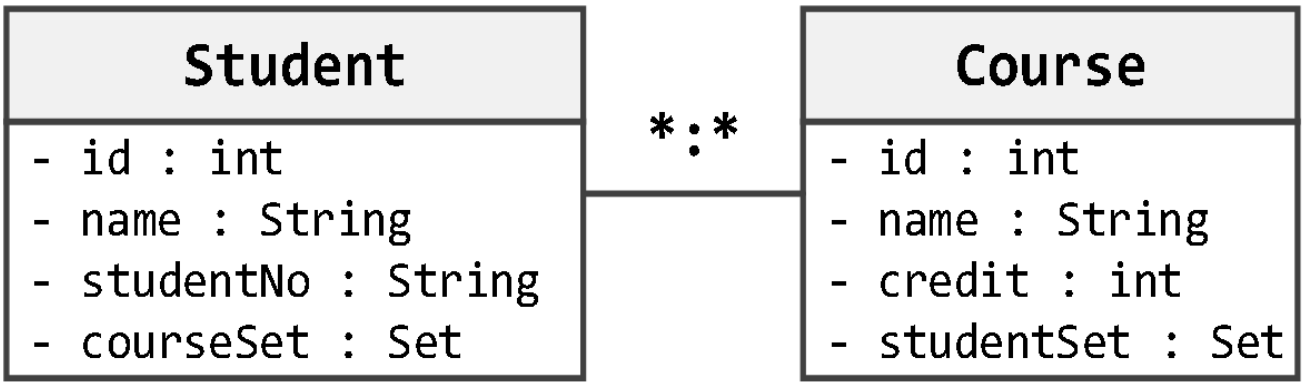

```
<!-- 映射User与Order之间的一对多关系
            map用来映射Map类型的属性
            key:指定外键字段名字
            index:指定保存Map中key值的字段
            one-to-many表示一对多关系 -->
        <map name="orderMap" cascade="delete">
            <key column="USER_ID"/>
            <index column="ORDER_KEY" type="string"></index>
            <one-to-many class="Order"/>
        </map>
```

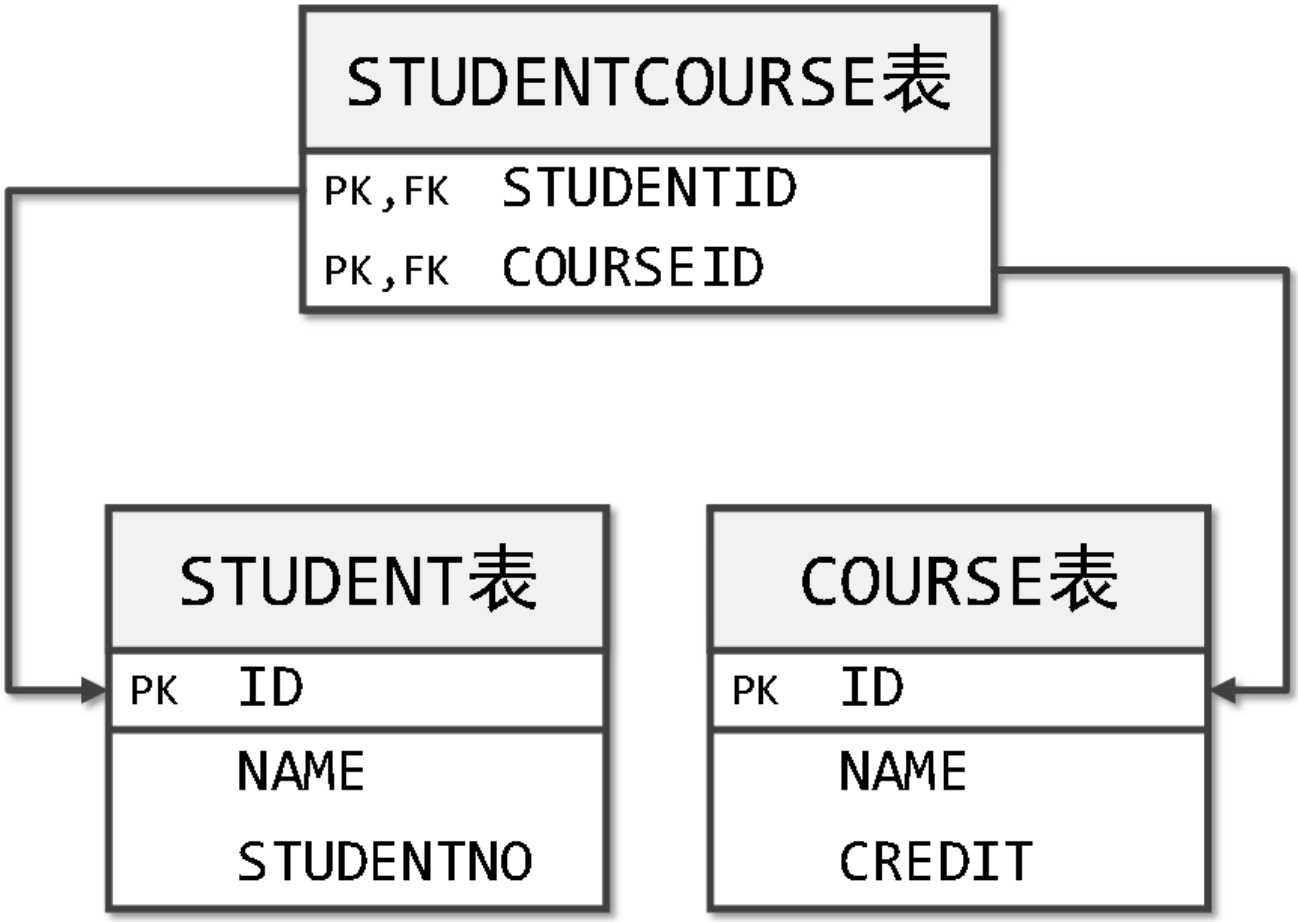
Order.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping package="com.hibernate.entity">
    <class name="Order" table="orders">
        <id name="id" type="int" column="id">
            <generator class="identity"></generator>
        </id>
        <property name="price" ></property>
    </class>
</hibernate-mapping>
```

双向一对多

CH06_多对多关联映射





CH08_Hibernate检索方式

```
package com.hibernate.ui;

import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;

import org.hibernate.Criteria;
```

```java
import org.hibernate.Hibernate;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.criterion.Criterion;
import org.hibernate.criterion.Restrictions;
import org.hibernate.query.NativeQuery;
import org.hibernate.query.Query;

import com.hibernate.entity.Order;
import com.hibernate.entity.User;
import com.hibernate.util.HibernateUtil;
/**
 * HQL,QBC
 * 1.HQL语句
 * 1.1.HQL查询语句
 * 1.2.HQL更新语句
 * 2.QBC语句
 *
 * @author 雨
 *
 */
public class Test {

    public static void main(String[] args) {
//      saveUserAndOrder();

//      addOrder();

//      getUserAndOrders();

//      deleteUser();

//      testHQL();
//      testHQLUpdate();
//      testHQLDelete();

//      pagingQuery(2,2);

//      testNamedQuery();
//      testQBC();

//      findUserByNameOrAge("张三",0);

//      testSQL();

//      getUserById();

        findOrderById();

        HibernateUtil.closeSessionFactory();
    }

    //执行SQL语句
    private static void testSQL() {
        Session session = HibernateUtil.openSession();
        NativeQuery query = session.createNativeQuery("select * from user", User.class);
        System.out.println(query.list());
    }

    private static void findUserByNameOrAge(String name, int age) {
        Session session = HibernateUtil.openSession();
```

```
        //HQL实现动态查询
//      StringBuffer hql = new StringBuffer("from User u ");
//      if(name != null && age !=0 ) {
//          hql.append("where u.userName = :name and u.age = :age");
//      }
//      if(name != null && age == 0) {
//          hql.append("where u.userName = :name");
//      }
//      if(name == null && age != 0) {
//          hql.append("where u.age = :age");
//      }
//
//      Query query = session.createQuery(hql.toString());
//      if(name != null) {
//          query.setParameter("name", name);
//      }
//      if(age != 0) {
//          query.setParameter("age", age);
//      }
//
//      System.out.println(query.list());

        //QBC实现动态查询
        Criteria criteria = session.createCriteria(User.class);
        if(name != null) {
            Criterion c1 = Restrictions.eq("userName", name);
            criteria.add(c1);
        }
        if(age != 0) {
            Criterion c2 = Restrictions.eq("age", age);
            criteria.add(c2);
        }

        System.out.println(criteria.list());

        session.close();
    }

    //QBC检索方式
    private static void testQBC() {
        Session session = HibernateUtil.openSession();

        //创建Criteria对象
        Criteria criteria = session.createCriteria(User.class);

        //添加查询条件
        Criterion c1 = Restrictions.eq("userName", "张三");
        Criterion c2 = Restrictions.ge("id", 118);

        //将条件添加到查询中
        criteria.add(c1);
        criteria.add(c2);

        System.out.println(criteria.list());
    }

    /**
    一、HQL检索方式
     */
    private static void testHQL() {
        Session session = HibernateUtil.openSession();
```

```
        //1.基础查询
//      String hql = "from User";//检索表中所有记录的所有字段
//      String hql = "select u from User u";//与上一句等价
//      String hql = "select u.id from User u";//查询某个字段的值
//      String hql = "select u.userName, u.password from User u";//查询多个字段
        //在HQL中可以直接调用构造方法(与上一句等价)
//      String hql = "select new User(u.userName, u.password) from User u";
//      Query query = session.createQuery(hql);
        //当查询结果中包含多条记录时，使用list方法执行HQL查询
//      List<User> userList = query.list();
//      System.out.println(userList);
        //当检索所有记录的某个字段时，结果中List集合元素的类型是该字段类型
//      List<Integer> userIds = query.list();
//      System.out.println(userIds);
        //当检索多个字段时，结果中List集合元素的类型是Object[]
//      List<Object[]> users = query.list();
//      for(Object[] o : users) {
//          System.out.println("用户名：" + o[0]);
//          System.out.println("密码：" + o[1]);
//      }


        //2.where子句
        //SQL注入代码
//      String name = "1 or 1 = 1";
//      String password = "1 or 1 = 1";
//      String hql = "from User where userName = " + name + " and password=" + password;

//      String hql = "from User where userName = '张三'";
//      String hql = "from User as u where u.userName like '张%'";//起别名使用as关键字，该关键字可
以省略
//      String hql = "from User where userName='Tom'";
//      Query query = session.createQuery(hql);
//      List<User> users = query.list();
//      System.out.println(users);

//      User user = (User) query.uniqueResult();//查询结果中只有一条记录，直接返回单个对象
//      System.out.println(user);

        //3.排序order by
//      String hql = "from User u order by u.userName desc";
//      String hql = "select u.userName, count(u) from User u group by u.userName having
u.userName != 'Tom'";
//      Query query = session.createQuery(hql);
//      List<User> users = query.list();
//      System.out.println(users);
//      List<Object[]> users = query.list();
//      for(Object[] o : users) {
//          System.out.println("用户名：" + o[0]);
//          System.out.println("个数：" + o[1]);
//      }

        //4.HQL传参方式
        //(1)?占位符
//      String hql = "from User where userName = ? and password = ?";
//      Query query = session.createQuery(hql);
//      query.setParameter(0, "张三");
//      query.setParameter(1, "111");
//      System.out.println(query.list());
```

```
        //(2)按照参数名称进行传参，参数名称不一定要与属性名相同
//      String hql = "from User where userName = :username and password = :password";
//      Query query = session.createQuery(hql);
//      query.setParameter("username", "张三");
//      query.setParameter("password", "111");
//      System.out.println(query.list());

        //(3)使用命名参数
        //第一种绑定方法，使用对象
//      String hql = "from User where userName = :userName and password = :password";
//      Query query = session.createQuery(hql);
//      User u = new User();
//      u.setUserName("张三");
//      u.setPassword("111");
//      query.setProperties(u);//HQL语句中命名参数的名称同setProperties方法参数中传入的对象的属性名称
要一致
//      System.out.println(query.list());

        //第二种绑定方法：使用map
//      String hql = "from User where userName = :userName and password = :password";
//      Query query = session.createQuery(hql);
//      Map<String, Object> pro = new HashMap<>();
//      pro.put("userName", "张三");
//      pro.put("password", "111");
//      query.setProperties(pro);//HQL语句中命名参数的名称同setProperties方法参数中传入的Map对象的
key名称要一致
////        System.out.println();
//      query.list();

        //where子查询(订单数量大于0的用户)
        //select * from user u where (select count(o) from orders o where u.id=o.userId )>0

        //等价的，相关子查询
//      String hql = "select u from User u where (select count(o) from Order o where o.user =
u)>0";
//      String hql1 ="select u from User u where (select count(o) from u.orderSet o) > 0";

        //使用size函数
//      String hql = "select u from User u where size(u.orderSet) > 0";

        //无关子查询（查询出订单价格大于所有订单平均价格的订单）？？？为什么要分相关、无关子查询
//      String hql = "select o from Order o where o.price > (select avg(o1.price) from Order
o1)";

        //查询订单中存在一条价格>100
//      String hql = "select u from User u where 100 < some(select o.price from u.orderSet o)";
        String hql = "select u from User u where u in(select o.user from Order o group by
o.user)";
        Query query = session.createQuery(hql);
        System.out.println(query.list());

        session.close();
    }

    //执行引用查询
    private static void testNamedQuery() {
        Session session = HibernateUtil.openSession();
        Query query = session.createNamedQuery("getUsers", User.class);
        System.out.println(query.list());
    }
```

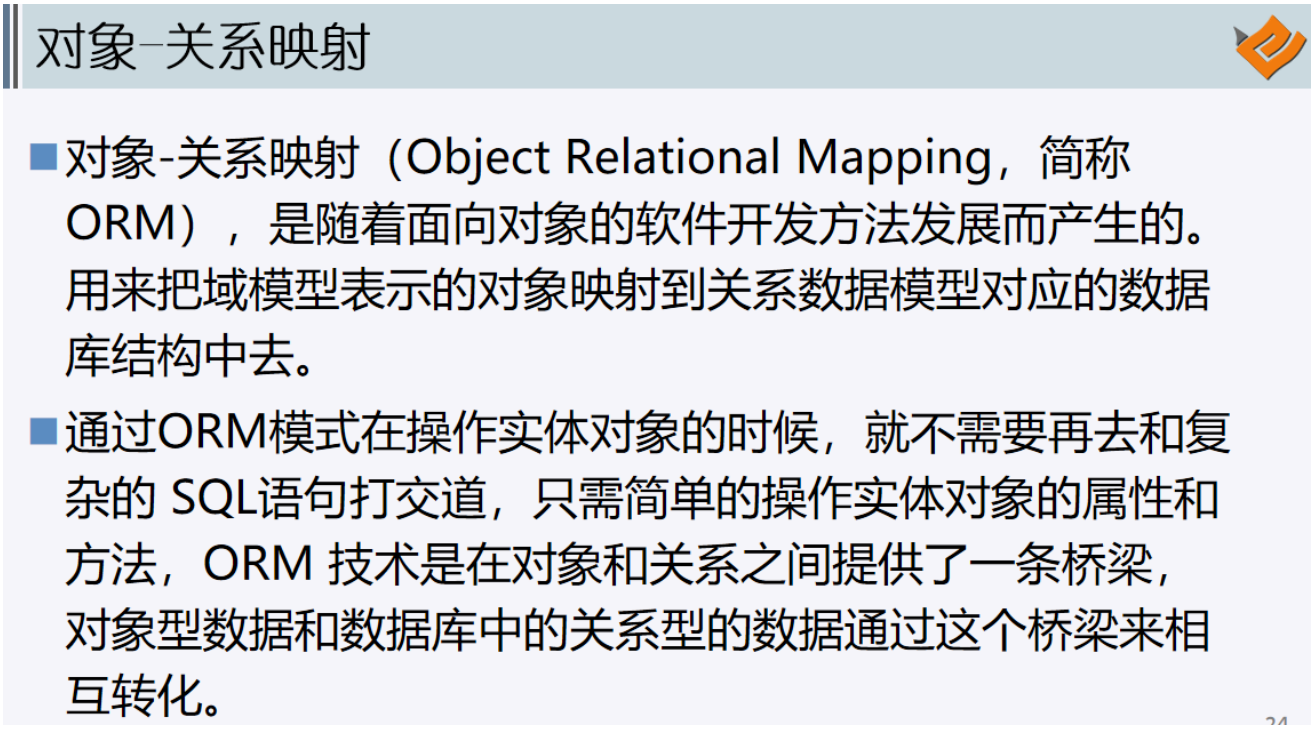
```
//分页查询:pageNum当前需要显示的页数，maxCount每页显示的数量
private static void pagingQuery(int pageNum, int maxCount) {
    Session session = HibernateUtil.openSession();
    Query query = session.createQuery("from User");
    query.setFirstResult((pageNum - 1) * maxCount);//此次查询的起始位置
    query.setMaxResults(maxCount);//此次查询的记录的条数
    System.out.println(query.list());
}

/**
 * 2.更新
 */
private static void testHQLUpdate() {
    Session session = HibernateUtil.openSession();
    Transaction tx = session.beginTransaction();
    // HQL实现更新
    String hql = "update User u set u.password = :password where u.userName=:username";
    Query query = session.createQuery(hql);
    query.setParameter("password", "qwerty");
    query.setParameter("username", "Tom");
    int row = query.executeUpdate();// 返回最终执行更新的记录的条数
    System.out.println("更新了" + row + "条");

    tx.commit();
    session.close();
}


private static void testHQLDelete() {
    Session session = HibernateUtil.openSession();
    Transaction tx = session.beginTransaction();
    // HQL实现删除
    String hql = "delete from User where userName=:username";
    Query query = session.createQuery(hql);
    query.setParameter("username", "Tom");
    int row = query.executeUpdate();// 返回最终执行删除的记录的条数
    System.out.println("删除了" + row + "条");

    tx.commit();
    session.close();
}

//保存
private static void saveUserAndOrder() {
    Session session = HibernateUtil.openSession();
    Transaction tx = session.beginTransaction();

    User u = new User();
    u.setUserName("李四");
    u.setPassword("77777");

    Order o = new Order();
    o.setPrice(300.0);

    //建立关联
    Set set = new HashSet();
    set.add(o);
    u.setOrderSet(set);

    session.save(u);
    session.save(o);
```

```
        tx.commit();
        session.close();
    }

    //给已经存在的用户添加新的订单
    private static void addOrder() {
        Session session = HibernateUtil.openSession();
        Transaction tx = session.beginTransaction();

        User u = session.get(User.class, new Integer(111));

        //新的订单对象
        Order o = new Order();
        o.setPrice(800.0);

        //建立关联
        u.getOrderSet().add(o);
        o.setUser(u);

        session.save(o);
        tx.commit();
        session.close();
    }

    //检索用户数据，并通过用户导航到它所关联的订单
    private static void getUserAndOrders() {
        Session session = HibernateUtil.openSession();
        //一对多关联映射默认情况下查询用户时，不会连接订单表查询
        User u = session.get(User.class, new Integer(109));

        //通过上述用户对象导航到它关联的订单(映射关联关系的优势)
        System.out.println(u.getOrderSet());

        session.close();
    }

    //由于配置了级联删除操作，所以删除用户时，会自动删除同用户关联的订单
    private static void deleteUser() {
        Session session = HibernateUtil.openSession();
        Transaction tx = session.beginTransaction();

        User u = session.get(User.class, new Integer(109));
        session.delete(u);

        tx.commit();
        session.close();
    }

    private static void getUserById() {
        Session session = HibernateUtil.openSession();
        //get的检索策略：立即检索（不可修改）
        User u = session.get(User.class, new Integer(110));
        //load默认的检索策略：延迟检索(可通过配置去修改检索策略)
//      User u = session.load(User.class, new Integer(110));
        System.out.println(u.getClass());
        //延迟检索返回对象OID属性是有值,访问代理对象的OID属性时，不会执行查询语句
        System.out.println(u.getId());

        //使用u时：访问对象的某个非OID属性时，执行查询语句
//      u.getUserName();
```

```
        //手动初始化代理类对象
//      Hibernate.initialize(u);

        session.close();

    }

    private static void findOrderById() {
        Session session = HibernateUtil.openSession();
        Order o = session.get(Order.class, new Integer(9));
        session.close();
    }

}
```

简答题

ORM



OID属性生成策略

- Hibernate 自带了很多种标识符生成器：
  - **increment** 采用 Hibernate 数值递增的方式；
  - **identity** 采用数据库提供的自增长方式；
  - **assigned** 主键由应用逻辑产生；
  - sequence 采用数据库提供的序列方式；
  - hilo 通过hi/lo算法 // Hibernate 5.0 以后不支持；
  - seqhilo 通过hi/lo算法；
  - native 自动选择合适的标识符生成器；
  - uuid.hex 通过uuid算法。

Session缓存作用

Session缓存的作用

- Session缓存有三大作用：
  - 减少数据库访问次数，提高数据访问的效率；
  - 保证缓存中的对象与数据库中相关的记录同步；
  - 当缓存中的持久化对象存在循环关联关系时，Session会保证不出现死循环，以及由死循环引起的堆栈溢出异常。

Hibernate缓存清理时期

Session清理缓存

- Session在某一时间点按照缓存中对象的属性变化来同步更新数据库的这一过程被称为 **Session 清理缓存。**
- 缓存清理的时间点：
  - 当调用 transaction.commit() 方法时，会先清理缓存，再向数据库提交事务；
  - 当显式调用 Session.flush() 方法时，会清理缓存；
  - 当调用 Session 的查询（不包括 load() 和 get())方法时，如果缓存中对象的属性有变化则清理缓存。

Hibernate实体对象生命周期

Hibernate中的实体对象

临时对象 (Transient Objects)	持久化对象 (Persist Objects)	游离对象 (Detached Objects)	被删除对象 (Removed Objects)
• 处于临时状态的对象称为临时对象 • 在数据库中不存在与之相对应的记录	• 处于持久化状态的对象称为持久化对象 • 在数据库中存在与之相对应的记录	• 处在游离状态的对象称为游离对象 • 在数据库中可能存在与之相对应的记录(前提是没有其他Session实例删除该记录)	• 处在删除状态的对象称为被删除对象 • 数据库中存在与之对应的记录(已经计划从数据库中删除)