# Hibernate框架简介

ORM规则:

ORM:Object Relation Mapping 对象关系映射

目标:

通过orm,可以做到:

1. 不写sql,从对象直接保存到数据库中
2. 不写sql,从数据库中读取,直接获取的是封装好的对象

Hibernate是对ORM的具体实现,ORM是一种思想,或者说是一种规则的定义,它是基于ORM的持久层框架,对JDBC操作进行了封装,和具体数据库无关的技术

基于ORM的应用:

1).hibernate

2)mybatise

3)自定义

**Hibernate框架的整体分析**

1)APi:

Configuration

SessionFactory

Session

Transaction

Query/Criteria...

2)配置

主配置文件:hibernate.cfg.xml

映射配置文件:\*.hbm.xml

1. 集合映射
2. 1对多/多对1映射
3. 多对多映射
4. 组件映射/继承映射
5. 联合主键映射

3)其他细节

a.hibernate对象状态

b.session缓存(1级缓存)

c.二级缓存

d.lazy/inverse/cascase关键字用法

# Hibernate开发环境

版本:3.6.0 Final

整体步骤:

1. 下载源码,引入jar文件

hibernate3.jar + required目录中所有jar + jpa + 数据库驱动

1. 写配置

\*hibernate.cfg.xml主配置

\*Users.hbm.xml映射配置

3) Api

Configuration

SessionFactory

Transaction

Session

Query

# 简单示例

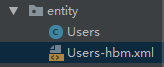
Users.java

|  |
| --- |
| package lw.pers.entity;  public class Users {  private int userId;  private String userName;  private int age;  public int getUserId() {  return userId;  }  public void setUserId(int userId) {  this.userId = userId;  }  public String getUserName() {  return userName;  }  public void setUserName(String userName) {  this.userName = userName;  }  public int getAge() {  return age;  }  public void setAge(int age) {  this.age = age;  }  @Override  public String toString() {  return "Users{" +  "userId=" + userId +  ", userName='" + userName + '\'' +  ", age=" + age +  '}';  }  } |

Users-hbm.xml

|  |
| --- |
| <?xml version="1.0"?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  <hibernate-mapping package="lw.pers.entity">  <class name="Users" table="users">  <id name="userId" column="id">  <generator class="native"></generator>  </id>  <property name="userName" column="name"></property>  <property name="age" column="age"></property>  </class>  </hibernate-mapping> |

注意:这个文件要和javabean同一个目录



src目录下建立文件hibernate.cfg.xml

|  |
| --- |
| <!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>  <property name="hibernate.connection.url">jdbc:mysql://47.96.27.61:3306/test</property>  <property name="hibernate.connection.driver\_class">com.mysql.jdbc.Driver</property>  <property name="hibernate.connection.username">lw</property>  <property name="hibernate.connection.password">linwei</property>  <!--通过数据库的方言,告诉hibernate如果产生sql,hibernate会根据配置的方言,生产符合当前数据库语言的sql-->  <property name="dialect">org.hibernate.dialect.MySQLDialect</property>  <!--查看hibernate生成的sql语句-->  <property name="hibernate.show\_sql">true</property>  <!--加载映射-->  <mapping resource="lw/pers/entity/Users-hbm.xml"></mapping>  </session-factory>  </hibernate-configuration> |

测试例子:

|  |
| --- |
| import lw.pers.entity.Users;  import org.hibernate.SessionFactory;  import org.hibernate.Transaction;  import org.hibernate.cfg.Configuration;  import org.hibernate.classic.Session;  public class Test {  @org.junit.Test  public void test()throws Exception{  //对象  Users users = new Users();  users.setUserName("jack");  users.setAge(30);  //1.创建配置管理对象  Configuration config = new Configuration();  //2.加载主配置文件,默认加载src/hibernate.cfg.xml  config.configure();  // config.configure("src/hibernate.cfg.xml");  //3.根据加载的主配置文件,创建对象  SessionFactory sessionFactory = config.buildSessionFactory();  //4.创建session对象,session就像是增强版的Connection,就是和数据库连接的一个会话信息  Session session = sessionFactory.openSession();  //5.开启事务  Transaction transaction = session.beginTransaction();  //保存  // session.save(users);  //查询,注意:第二个参数为主键,类型为Serializable类型的,2首先会自动转化为Interger,因为Interger继承了Number,而Number实现了Serializable,所以可以这么写  // Users users1 = (Users)session.get(Users.class, 2);  // System.out.println(users1);  //6.提交事务  transaction.commit();  session.close();  }  } |

# API

## 4.1增删改查

1. session.saveOrUpdate(Object)

作用:保存或者更新

如果Object没有设置主键,则保存

如果Object有设置主键,且主键存在,则更新,如果主键不存在则报错

2）session.save(Object)

作用:保存

如果Object没有设置主键,则保存

如果Object设置了主键,则还是保存,而不会去更新

1. session.delete(Object)

作用:删除

1. 先创建一个对象,设置一个存在的主键,再删除,

Users users1 = new Users();

Users1.setUserId(2);

session.delete(users1);

1. 先查询返回一个对象,再删除

Object object = session.get(Users.class,1);

if(object !=null){

session.delete(object);

}

1. session.update(Object)

作用:修改

和删除过程差不多

5）session.get()

作用:查询

Users users1 = (Users) session.get(Users.class, 2 );

当查询的主键不存在,则返回null

1. session.load()

作用:查询

Users users1 = (Users) session.load(Users.class, 2 );

当查询的主键不存在时,如果使用返回后的结果,则报错

load和geti相比,load是懒加载

## 4.2查询方式

### 4.2.1HQL（推荐）

查询的是对象,对象的属性,查询语句和SQL差不多,只是属性和字段替换了下

HQL和SQL的区别:

HQL查询的是对象,对象的属性,区分大小写

SQL查询的是表,字段,不区分大小写

|  |
| --- |
| //创建session对象  Session session = sessionFactory.openSession();  Transaction transaction = session.beginTransaction();  //得到hql查询结果  Query query = session.createQuery("from Users");  List<Users> list = query.list();  System.out.println(list);  transaction.commit();  session.close(); |

传递参数到sql中

session.createQuery(“update Dept set deptName=? where id=?”)

.setParameter(0,”HR”)

.setParameter(1,3);

这里从0开始

### 4.2.2QBC(了解)

HQL的查询语句和SQL差不多,但是QBC和SQL完全不一样,是面向对象的查询方式

|  |
| --- |
| //创建session对象  Session session = sessionFactory.openSession();  Transaction transaction = session.beginTransaction();  Criteria criteria = session.createCriteria(Users.class);  //设置条件  criteria.add(Restrictions.eq("userName","haha\_hello"));  criteria.add(Restrictions.eq("userId",2));  List<Users> list = criteria.list();  System.out.println(list);  transaction.commit();  session.close(); |

# hibernate主配置文件

hibernate.hbm2ddl.auto create 每次都先删除表,再创建表

hibernate.hbm2ddl.auto update 如果表不存在就创建,存在就不创建

hibernate.hbm2ddl.auto create-drop 每次在创建sessionFactory的时候创建表,执行sf.close()的时候删除表

hibernate.hbm2ddl.auto validate 检查映射配置与数据库结构是否一致,不一致就报错

代码方式创建表:

|  |
| --- |
| //创建配置管理器对象,加载配置文件  Configuration cfg = new Configuration().configure();  //自动建表工具类  SchemaExport export = new SchemaExport(cfg);  //第一个参数:是否打印建表的sql语句到控制台上  //第二个参数:是否创建表,true表示创建表,false表示不创建表  export.create(true,true); |

# 6.映射配置文件

作用:描述”对象”与”表”的映射关系

命名:\*.hbm.xml

<property name=”birth” type=”timestamp”></property>

这里的timestamp,对应的数据库类型是datetime

|  |
| --- |
| <?xml version="1.0"?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  <!—  package:表示包名,可选的,如果没有写,后面必须指定类的全名  auto-import=”true”默认为true,即在写hql的时候,会自动引入包名,如果为false,则hql中要写完整的类名  class:表示一个javabean对象  name:表示映射的类的名称  table:(可选)类的名称,对应的表名,如果不写,默认是和类名称一样  length:表示字符长度,只对字符串有效  type:数据库中字段的类型  hibernate中指定的类型:小写开头  java中类型:写类的全名  unique:true表示唯一约束  id 表示映射的是主键  property:表示普通字段的映射  name:属性的名称  column:可选,如果没有指定,与属性名称name的值一样  generator:表示主键生成策略  class:  identity:主键自增长  sequence:以序列的方式实现自增长  native:主键自增长,根据底层数据库的能力选择identity,sequence其中一个  assigned:手动指定主键的值  uuid:uuid的值作为主键  -->  <hibernate-mapping package="lw.pers.entity">  <class name="Users" table="users">  <id name="userId" column="id">  <generator class="native"></generator>  </id>  <property name="userName" column="name"></property>  <property name="age" column="age"></property>  </class>  </hibernate-mapping> |

# 7.联合主键映射

概念:多列共同作为主键

1)创建连接主键类:

|  |
| --- |
| package lw.pers.entity;  import java.io.Serializable;  //联合主键对象  public class CompositeKeys implements Serializable{  private String name;  private String address;  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public String getAddress() {  return address;  }  public void setAddress(String address) {  this.address = address;  }  } |

2)创建实体类

|  |
| --- |
| package lw.pers.entity;  import java.util.Date;  public class Employee {  //名称,地址,部门,年龄  private CompositeKeys keys;  private String dept;  private Date birth;  public String getDept() {  return dept;  }  public void setDept(String dept) {  this.dept = dept;  }  public Date getBirth() {  return birth;  }  public void setBirth(Date birth) {  this.birth = birth;  }  public CompositeKeys getKeys() {  return keys;  }  public void setKeys(CompositeKeys keys) {  this.keys = keys;  }  @Override  public String toString() {  return "Employee{" +  "keys=" + keys +  ", dept='" + dept + '\'' +  ", birth=" + birth +  '}';  }  } |

3)Employee.hbm.xml映射配置

|  |
| --- |
| <?xml version="1.0"?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  <hibernate-mapping package="lw.pers.entity">  <class name="Employee" table="t\_employee">  <composite-id name="keys">  <key-property name="name"></key-property>  <key-property name="address"></key-property>  </composite-id>  <property name="dept" length="50"></property>  <property name="birth" type="date"></property>  </class>  </hibernate-mapping> |

4)测试

|  |
| --- |
| import lw.pers.entity.CompositeKeys;  import lw.pers.entity.Employee;  import org.hibernate.SessionFactory;  import org.hibernate.cfg.Configuration;  import org.hibernate.classic.Session;  import java.util.Date;  public class Test {  //因为SessionFactory是单例,所以这么写  private static SessionFactory sessionFactory;  static {  Configuration configuration = new Configuration();  sessionFactory = configuration.configure().buildSessionFactory();  }  //保存  @org.junit.Test  public void test\_save()throws Exception{  Session session = sessionFactory.openSession();  session.beginTransaction();  CompositeKeys keys = new CompositeKeys();  keys.setName("Jack");  keys.setAddress("湖北");  Employee emp = new Employee();  emp.setKeys(keys);  emp.setDept("研发部");  emp.setBirth(new Date());  session.save(emp);  //保存  session.getTransaction().commit();  session.close();  }  //获取联合主键  @org.junit.Test  public void test\_get()throws Exception{  Session session = sessionFactory.openSession();  session.beginTransaction();  CompositeKeys keys = new CompositeKeys();  keys.setName("Jack");  keys.setAddress("湖北");  Employee employee = (Employee) session.get(Employee.class,keys);  System.out.println(employee);  session.getTransaction().commit();  session.close();  }  } |

# 8.hibernate对象状态

临时状态

1. 直接new出来的状态
2. 不处理session的管理(即没有与一个session关联)
3. 对象在数据库中没有对象的记录

Employee e = new Employee();

e.setId(2);只要主键在数据库中存在,就说这个对象在数据库中有对应记录

持久化状态

1. 处理session的管理范围

当执行session的方法如:save/update/saveOrUpdate/get/load

对象就会自动转变为持久化状态

1. 在数据库中有对应的记录
2. 处于持久化状态的对象,当对对象属性进行更改的时候,提交更改会反映到数据库中

游离状态

1. 对象不处于session的管理范围;通常指session关闭后
2. 对应在数据库中有对应的记录

|  |
| --- |
| import lw.pers.crud.Employee;  import org.hibernate.Session;  import org.hibernate.SessionFactory;  import org.hibernate.cfg.Configuration;  import java.util.Date;  public class Test {  private static SessionFactory sf;  static {  sf = new Configuration().configure().buildSessionFactory();  }  //保存  @org.junit.Test  public void test\_save(){  Session session = sf.openSession();  session.beginTransaction();  //创建对象  //临时状态  Employee emp = new Employee();  emp.setName("Rose");  emp.setBirth(new Date());  //保存  session.save(emp);  //持久化状态  emp.setName("lucy...");//修改会反应到数据库中,这里会生成update语句  session.getTransaction().commit();  session.close();  //游离状态  emp.setName("hahah");//这里不会反应到数据库中  System.out.println(emp.getName());  }  //游离->持久  @org.junit.Test  public void test\_2(){  Session session = sf.openSession();  session.beginTransaction();  Employee employee = new Employee();//临时状态  employee.setId(3); //设置了主键后,数据库中就有了对应的数据,则成了游离状态  session.update(employee);//持久化了  employee.setName("你好啊");  session.getTransaction().commit();  session.close();  }  } |

# 9.Session缓存

概念:session缓存,也叫做一级缓存,当执行session的相关方法,如:save,update,get,load等方法时候,对象会自动放入一级缓存中,当session关闭后,一级缓存内容失效

**特点:**

1. 缓存有效范围,只在当前session范围内有效,缓存时间很短,作用范围小
2. 一级缓存,可以在短时间内多次操作数据库的时候,才会明显提升效率

一级缓存的结构:Map<主键,对象

1. 在提交事务的时候

Hibernate会用户缓存数据到数据库中,会对比缓存数据和数据库数据是否一致

,如果不一致,才提交更改到数据库(生成update)

4)hibernate提供的一级缓存由hibernate自身维护,如果想操作一级缓存的内容,必须通过hibernate提供的方法,即session.flush();

例如:

employee.setName("你好啊");

session.flush();

employee.setName("hahaah");

xxxxxxx.commit();

如果不写session.flush();则只有1条sql生成,如果加了session.flush()这一行,将会有两行

还有两个影响1级缓冲的操作:clear ,evict

Employee employee = (Employee) session.get(Employee.class,1);

session.evict(employee);//清空一级缓存中的指定对象

session.clear();//清空一级缓存中所有对象

Employee employee1 = (Employee) session.get(Employee.class,1);

5)list查询和iterate查询,都是Query的方法

Query from\_employee = session.createQuery("from Employee");

iterate查询：先从缓存中查找,如果缓存中没有,则直接查询,并进行缓存

总共查询n+1条,即n+1条sql

1. 先查询所有的主键,总共1次
2. 再高扭矩每一个主键,进行查询,总共n次
3. iterate查询:n+1
4. iterate查询,在使用数据的时候,才会进行b步骤（懒加载），即不打印时,是不会进行查询的

|  |
| --- |
| Iterator<Employee> iterate = from\_employee.iterate();  while (iterate.hasNext()){  Employee employee = iterate.next();  System.out.println(employee);  } |

list查询:查询的结果会放入1级缓存,但不会从1级缓存中获取

一次性查询所有满足需要的数据即总共1次

|  |
| --- |
| Query from\_employee = session.createQuery("from Employee");  List<Employee> list = from\_employee.list();  for(Employee employee : list){  System.out.println(employee);  } |

# 10.lazy属性

Hibernate为了提升程序运行效率,提供了懒加载

lazy属性表示懒加载

true:支持懒加载

false:关闭懒加载

extra:懒加载,在真正使用数据的时候,才向数据库发送查询的sql,当执行集合的isEmpty()/size()方法的时候,只是统计

懒加载位置:

类级别:默认支持懒加载,但只有在使用load时才会应用这个特性

字段级别:普通字段无效,大数据类型会有影响(long/longtext)

集合属性:默认支持懒加载

懒加载异常: org.hibernate.LazyInitializationException: could not initialize proxy - no Session

出现异常原因:由于是懒加载,所以在使用数据时,才发送sql,这是session不能close,

|  |
| --- |
| Employee emp = (Employee) session.load(Employee.class, 3);  session.getTransaction().commit();  session.close();  System.out.println(emp);//在session关闭后 |

像这样,就会出现懒加载异常

解决方法:

1. 在session关闭之前,先使用一次数据,那么session关闭后再使用对象数据,就不是懒加载数了
2. 强迫代理初始化对象

其实这两种方式都是在session关闭前使用一次对象数据,这样就会发送sql,来设置对象,此时对象就有了数据

|  |
| --- |
| Session session = sf.openSession();  session.beginTransaction();  Employee emp = (Employee) session.load(Employee.class, 3);  session.getTransaction().commit();  // System.out.println(emp); //方式1  Hibernate.initialize(emp);//方式2  session.close();  System.out.println(emp); |

# 11.集合映射

1.需求:购物填写地址,一个用户对应多个地址

2.数据库设计:

t\_users

id name address1 address2 ...

100 jack 湖北潜江 湖北天门 ...

这么设计是有问题的,因为你不知道地址有几个,所以无法确定地址字段有多少个,如果你默认就10个地址字段,那么如果某个用户只有一个地址字段,那么剩下9个地址段的内容都为空.这样表的设计是不合理的。

更改后的设计成了两个表:

t\_users t\_address

id name id address set集合映射

100 jack 100 广州

100 天津

t\_users t\_address

id name id address idex\_ list集合映射

100 jack 100 广州 0

100 天津 1

t\_users t\_address

id name id address shortName\_ Map集合映射

100 jack 100 广州 gz

100 天津 sz

javabean:

|  |
| --- |
| package lw.pers.crud;  import java.util.\*;  public class Users {  private int id;  private String name;  //用户对应的多个地址  //set集合  private Set<String> addressSet;  //list集合  private List<String> addressList = new LinkedList<>();  //Map集合  private Map<String,String> addressMap = new HashMap<>();  public Map<String, String> getAddressMap() {  return addressMap;  }  public void setAddressMap(Map<String, String> addressMap) {  this.addressMap = addressMap;  }  public int getId() {  return id;  }  public void setId(int id) {  this.id = id;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public Set<String> getAddressSet() {  return addressSet;  }  public void setAddressSet(Set<String> addressSet) {  this.addressSet = addressSet;  }  public List<String> getAddressList() {  return addressList;  }  public void setAddressList(List<String> addressList) {  this.addressList = addressList;  }  } |

映射配置:

|  |
| --- |
| <?xml version="1.0"?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  <hibernate-mapping package="lw.pers.crud">  <class name="Users" table="t\_users">  <id name="id" column="id">  <generator class="native"></generator>  </id>  <property name="name" length="50"></property>  <!--用户对应的多个地址-->  <!--集合属性的映射-->  <!--  要映射的属性:集合属性addressSet  映射到的表:t\_address  指定外键字段:user\_id  -->  <set name="addressSet" table="t\_address">  <key column="user\_id"></key>  <!--这里的element必须指定type类型,因为没有t\_address表对应的javabean-->  <element column="address" type="string"></element>  </set>  <!--list集合映射-->  <list name="addressList" table="t\_addressList">  <key column="user\_id"></key>  <list-index column="idx\_"></list-index>  <element column="address" type="string"></element>  </list>  <!--Map集合的映射-->  <map name="addressMap" table="t\_addressMap">  <key column="user\_id"></key>  <map-key type="string" column="showName"></map-key>  <element column="address" type="string"></element>  </map>  </class>  </hibernate-mapping> |

测试:

|  |
| --- |
| import lw.pers.crud.Users;  import org.hibernate.Session;  import org.hibernate.SessionFactory;  import org.hibernate.cfg.Configuration;  import java.util.HashSet;  public class Test {  private static SessionFactory sf;  static {  sf = new Configuration().configure().buildSessionFactory();  }  @org.junit.Test  public void test\_3(){  Session session = sf.openSession();  session.beginTransaction();  //用户  Users users = new Users();  users.setName("老张");  //地址对象  HashSet<String> addressSet = new HashSet<>();  addressSet.add("上海");  addressSet.add("北京");  //关系  users.setAddressSet(addressSet);  //保存  session.save(users);  session.getTransaction().commit();  session.close();  }  //只要配置好映射,在获取苏话剧的时候,会获取到关联对象的数据  @org.junit.Test  public void test\_4(){  Session session = sf.openSession();  session.beginTransaction();  Users users =(Users)session.get(Users.class, 1);  System.out.println(users.getAddressSet()); //关联的数据懒加载  session.getTransaction().commit();  session.close();  }  @org.junit.Test  public void test\_5(){  Session session = sf.openSession();  session.beginTransaction();  Users users = new Users();  users.setName("老许");  users.getAddressList().add("b\_深圳福田...");  users.getAddressList().add("a\_深圳宝安...");  session.save(users);  session.getTransaction().commit();  session.close();  }  @org.junit.Test  public void test\_6(){  Session session = sf.openSession();  session.beginTransaction();  Users users = (Users) session.get(Users.class, 2);  System.out.println(users.getAddressList());  session.getTransaction().commit();  session.close();  }  @org.junit.Test  public void test\_7(){  Session session = sf.openSession();  session.beginTransaction();  Users users = new Users();  users.setName("老许");  users.getAddressMap().put("gz","深圳福田...");  users.getAddressMap().put("sz","深圳宝安...");  session.save(users);  session.getTransaction().commit();  session.close();  }  @org.junit.Test  public void test\_8(){  Session session = sf.openSession();  session.beginTransaction();  Users users = (Users) session.get(Users.class, 4);  System.out.println(users.getAddressMap());  session.getTransaction().commit();  session.close();  }  } |

# 12.各种映射

在1对多或者反过来多对1的关系中,最好通过多的放一放来维护关系,即多对1（推荐）,可以少生成sql语句

需求:保存用户信息,用户对应多个地址

小张 年龄25 广州天河,gzth,51000

数据库设计:

javabean

t\_users t\_address

id name age id address shortName code user\_id

100 小张 25 1 广州天河 gzth 51000 100

用户和地址的关系:

1对多:一个用户对应多个地址

地址和用户的关系:

多对1:多个地址对应1个用户

**1对多**

|  |
| --- |
| class Users  int id;  String name;  int age;  Set<Address> address //1对多 |

**多对1**

|  |
| --- |
| class Address  int id;  String name;  String shortName;  String code;  Users user; |

## 12.1多对1(推荐)

Users.java:

|  |
| --- |
| package lw.pers.crud;  public class Users{  private int id;  private String name;  private int age;  public int getId() {  return id;  }  public void setId(int id) {  this.id = id;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public int getAge() {  return age;  }  @Override  public String toString() {  return "Users{" +  "id=" + id +  ", name='" + name + '\'' +  ", age=" + age +  '}';  }  public void setAge(int age) {  this.age = age;  }  } |

Address.java:

|  |
| --- |
| package lw.pers.crud;  public class Address {  private int id;  private String shortName;  private String code;  private Users users;  private String name;  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  //地址和用户的关系,是多对1的关系  public int getId() {  return id;  }  public void setId(int id) {  this.id = id;  }  public String getShortName() {  return shortName;  }  public void setShortName(String shortName) {  this.shortName = shortName;  }  public String getCode() {  return code;  }  public void setCode(String code) {  this.code = code;  }  public Users getUsers() {  return users;  }  public void setUsers(Users users) {  this.users = users;  }  } |

Users.hbm.xml:

|  |
| --- |
| <?xml version="1.0"?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  <!--用户映射文件-->  <hibernate-mapping package="lw.pers.crud">  <class name="Users" table="t\_users">  <id name="id" column="id">  <generator class="native"></generator>  </id>  <property name="name" length="50"></property>  <property name="age"></property>  </class>  </hibernate-mapping> |

Address.hbm.xml:

|  |
| --- |
| <?xml version="1.0"?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  <!--用户映射文件-->  <hibernate-mapping package="lw.pers.crud">  <class name="Address" table="t\_address">  <id name="id" column="id">  <generator class="native"></generator>  </id>  <property name="name" length="50"></property>  <property name="shortName" length="20"></property>  <property name="code" length="20"></property>  <!--  多对1配置:  1.映射的对象  2.对象对应的外键字段  3.对象的类型  注意:Uesrs一定要有映射文件  -->  <many-to-one name="users" column="user\_id" class="Users"></many-to-one>  </class>  </hibernate-mapping> |

## 12.2.1对多

Users.java:

|  |
| --- |
| package lw.pers.crud;  import java.util.HashSet;  import java.util.Set;  public class Users{  private int id;  private String name;  private int age;  //用户和地址,是1对多的关系[这里一定要用接口接受]  private Set<Address> addresses = new HashSet<>();  public Set<Address> getAddresses() {  return addresses;  }  public void setAddresses(Set<Address> addresses) {  this.addresses = addresses;  }  public int getId() {  return id;  }  public void setId(int id) {  this.id = id;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public int getAge() {  return age;  }  @Override  public String toString() {  return "Users{" +  "id=" + id +  ", name='" + name + '\'' +  ", age=" + age +  '}';  }  public void setAge(int age) {  this.age = age;  }  } |

Address.java:

|  |
| --- |
| package lw.pers.crud;  public class Address {  private int id;  private String shortName;  private String code;  private String name;  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  //地址和用户的关系,是多对1的关系  public int getId() {  return id;  }  public void setId(int id) {  this.id = id;  }  public String getShortName() {  return shortName;  }  public void setShortName(String shortName) {  this.shortName = shortName;  }  public String getCode() {  return code;  }  public void setCode(String code) {  this.code = code;  }  } |

Users.hbm.xml:

|  |
| --- |
| <?xml version="1.0"?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  <!--用户映射文件-->  <hibernate-mapping package="lw.pers.crud">  <class name="Users" table="t\_users">  <id name="id" column="id">  <generator class="native"></generator>  </id>  <property name="name" length="50"></property>  <property name="age"></property>  <!--1对多映射  -->  <set name="addresses" table="t\_addressSet">  <key column="user\_id"></key>  <one-to-many class="Address"></one-to-many>  </set>  </class>  </hibernate-mapping> |

Address.hbm.xml:

|  |
| --- |
| <?xml version="1.0"?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  <!--用户映射文件-->  <hibernate-mapping package="lw.pers.crud">  <class name="Address" table="t\_address">  <id name="id" column="id">  <generator class="native"></generator>  </id>  <property name="name" length="50"></property>  <property name="shortName" length="20"></property>  <property name="code" length="20"></property>  <!--通过一方维护关系-->  </class>  </hibernate-mapping> |

Test:

|  |
| --- |
| import lw.pers.crud.Address;  import lw.pers.crud.Users;  import org.hibernate.Session;  import org.hibernate.SessionFactory;  import org.hibernate.cfg.Configuration;  public class Test {  private static SessionFactory sf;  static {  sf = new Configuration().configure().buildSessionFactory();  }  @org.junit.Test  public void test\_7(){  Session session = sf.openSession();  session.beginTransaction();  Address address = new Address();  address.setName("上海");  address.setCode("51000");  address.setShortName("sh");  //用户  Users uses = new Users();  uses.setName("小张");  uses.setAge(25);  //关系  // address.setUsers(uses);  //保存用户地址,当然保存顺序可以调换,如果调换了,可能会多生成sql语句  session.save(uses);  session.save(address);  session.getTransaction().commit();  session.close();  }  @org.junit.Test  public void test\_8(){  Session session = sf.openSession();  session.beginTransaction();  Address address = (Address) session.get(Address.class,1);  // System.out.println(address.getUsers());  session.getTransaction().commit();  session.close();  }  @org.junit.Test  public void test\_9(){  Session session = sf.openSession();  session.beginTransaction();  Address address = new Address();  address.setName("上海");  address.setCode("51000");  address.setShortName("sh");  //用户  Users uses = new Users();  uses.setName("小张");  uses.setAge(25);  //关系  uses.getAddresses().add(address);  session.save(address);  session.save(uses);  session.getTransaction().commit();  session.close();  }  @org.junit.Test  public void test\_10(){  Session session = sf.openSession();  session.beginTransaction();  Users user = (Users) session.get(Users.class,1);  System.out.println(user.getAddresses());  session.getTransaction().commit();  session.close();  }  } |

## 12.3解除关联关系

|  |
| --- |
| @Test  public void test\_1() {  Session session = sf.openSession();  session.beginTransaction();  Dept dept = (Dept)session.get(Dept.class,1);  dept.getEmployees().clear(); //t\_employee的user\_id就为null了  session.getTransaction().commit();  session.close();  } |

## 12.4多对多

项目和开发人员的关系就是多对多的关系:

t\_project 项目表

id name

100 CRM

101 OA

class Project:

int id;

String name;

Set<Person> personSet;

t\_person 开发人员表

personId name

1. 小王
2. 小李

class Project:

int id;

String name;

Set<Project > projectSet;

t\_relation 关系表

project\_id person\_id

100 1

100 2

101 1

Project.java:

|  |
| --- |
| package lw.pers.aa;  import java.util.HashSet;  import java.util.Set;  public class Project {  private int id;  private String name;  private Set<Person> personSet = new HashSet<>();  public int getId() {  return id;  }  public void setId(int id) {  this.id = id;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public Set<Person> getPersonSet() {  return personSet;  }  public void setPersonSet(Set<Person> personSet) {  this.personSet = personSet;  }  @Override  public String toString() {  return "Project{" +  "id=" + id +  ", name='" + name + '\'' +  ", personSet=" + personSet +  '}';  }  } |

Person.java:

|  |
| --- |
| package lw.pers.aa;  import java.util.HashSet;  import java.util.Set;  public class Person {  private int personId;  private String name;  private Set<Project> projectSet = new HashSet<>();  public int getId() {  return personId;  }  public int getPersonId() {  return personId;  }  public void setPersonId(int personId) {  this.personId = personId;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public Set<Project> getProjectSet() {  return projectSet;  }  public void setProjectSet(Set<Project> projectSet) {  this.projectSet = projectSet;  }  @Override  public String toString() {  return "Person{" +  "personId=" + personId +  ", name='" + name + '\'' +  ", projectSet=" + projectSet +  '}';  }  } |

Project.hbm.xml:

|  |
| --- |
| <?xml version="1.0"?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  <hibernate-mapping package="lw.pers.aa">  <class name="Project" table="t\_project">  <id name="id">  <generator class="native"></generator>  </id>  <property name="name" length="50"></property>  <!--  多对多的映射配置  table:集合元素映射的中间表  key:中间表外键字段  many-to-many:中的column表示当前外键字段project\_id中对应的字段，class表示对应字段的类型  -->  <set name="personSet" table="t\_relation">  <key column="project\_id"></key>  <many-to-many column="person\_id" class="Person"></many-to-many>  </set>  </class>  </hibernate-mapping> |

Person.hbm.xml:

|  |
| --- |
| <?xml version="1.0"?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  <hibernate-mapping package="lw.pers.aa">  <class name="Person" table="t\_person">  <id name="personId">  <generator class="native"></generator>  </id>  <property name="name" length="50"></property>  <set name="projectSet" table="t\_relation">  <key column="person\_id"></key>  <many-to-many column="project\_id" class="Project"></many-to-many>  </set>  </class>  </hibernate-mapping> |

test:

|  |
| --- |
| package lw.pers.aa;  import org.hibernate.Session;  import org.hibernate.SessionFactory;  import org.hibernate.cfg.Configuration;  import org.junit.Test;  public class App {  private static SessionFactory sf;  static {  sf = new Configuration().configure().buildSessionFactory();  }  //保存数据  @Test  public void back(){  Session session = sf.openSession();  session.beginTransaction();  //创建对象  Project p\_oa = new Project();  p\_oa.setName("OA项目");  Project p\_crm = new Project();  p\_crm.setName("CRM项目");  Person p1 = new Person();  p1.setName("小王");  Person p2 = new Person();  p2.setName("小李");  //关系  p\_crm.getPersonSet().add(p1);  p\_crm.getPersonSet().add(p2);  p\_oa.getPersonSet().add(p1);  // p1.getProjectSet().add(p\_crm);错误,不能重复维护,因为t\_relation是联合主键,违反主键唯一的约束  //保存  session.save(p\_crm);  session.save(p\_oa);  session.save(p1);  session.save(p2);  session.getTransaction().commit();  session.close();  }  } |

如果Project.hbm.xml中配置了<set>则可以Project对象来配置Person数据,相反也一样.

## 12.4.组件映射

对象之间的关系:

组合关系:一个类中包含对另一个类的引用,这2个类就是组合关系

继承关系:一个类继承另外一个类

组件映射:组件类和包含的组件类同事映射到一个表

需求:

汽车与车轮

Car.java:

|  |
| --- |
| package lw.pers.component;  public class Car {  private int id;  private String type;  private Wheel wheel;  public int getId() {  return id;  }  public void setId(int id) {  this.id = id;  }  public String getType() {  return type;  }  public void setType(String type) {  this.type = type;  }  public Wheel getWheel() {  return wheel;  }  public void setWheel(Wheel wheel) {  this.wheel = wheel;  }  } |

Wheel.java:

|  |
| --- |
| package lw.pers.component;  public class Wheel {  //车轮个数  private int count;  //尺寸  private double size;  public int getCount() {  return count;  }  public void setCount(int count) {  this.count = count;  }  public double getSize() {  return size;  }  public void setSize(double size) {  this.size = size;  }  } |

Car.hbm.xml:

|  |
| --- |
| <?xml version="1.0"?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  <hibernate-mapping package="lw.pers.component">  <class name="Car" table="t\_car">  <id name="id">  <generator class="native"></generator>  </id>  <property name="type"></property>  <!--组件映射-->  <component name="wheel" class="Wheel">  <property name="count"></property>  <property name="size"></property>  </component>  </class>  </hibernate-mapping> |

test:

|  |
| --- |
| package lw.pers.component;  import org.hibernate.Session;  import org.hibernate.SessionFactory;  import org.hibernate.cfg.Configuration;  import org.junit.Test;  public class App {  private static SessionFactory sf;  static {  sf = new Configuration().configure().buildSessionFactory();  }  //保存数据  @Test  public void save(){  Session session = sf.openSession();  session.beginTransaction();  //车轮  Wheel wheel = new Wheel();  wheel.setCount(10);  wheel.setSize(15.6);  //汽车  Car car = new Car();  car.setType("BWM");  car.setWheel(wheel);  //保存  session.save(car);  session.getTransaction().commit();  session.close();  }  } |

+----+------+-------+------+

| id | type | count | size |

+----+------+-------+------+

| 1 | BWM | 10 | 15.6 |

+----+------+-------+------+

## 12.5.1对1映射

需求:存储用户,身份证信息

用户:姓名,性别,年龄,身份证编号

身份证:身份证编号,签发地,有效日期

它们的关系就是一对一的关系

t\_idcard 身份证信息

cardNo place date

3412 上海 2020-09-11

对于t\_user表的设计有两种方式:

**方式1: card\_id**

t\_user 用户

id name sex card\_id 注意:card\_id这个字段是外键,引用的是t\_idcard的cardNo字段,且外键要唯一

1 jack 男 3412

**方式2:**

t\_user用户

card\_id name sex 注意:这个car\_id是引用的t\_idcard的表的主键cardNo,这里主键和外键是同一个字段 car\_id

3412 jack 男

方式1代码:

User.java:

|  |
| --- |
| package lw.pers.bb;  //用户对象  //1对1映射,方式1  public class User {  private int id;  private String name;  private char sex;  //用户关联的身份证信息,在数据库中是外键  private IdCard idCard;  public int getId() {  return id;  }  public void setId(int id) {  this.id = id;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public char getSex() {  return sex;  }  public void setSex(char sex) {  this.sex = sex;  }  public IdCard getIdCard() {  return idCard;  }  public void setIdCard(IdCard idCard) {  this.idCard = idCard;  }  } |

IdCard.java:

|  |
| --- |
| package lw.pers.bb;  import java.util.Date;  //身份证对象  public class IdCard {  private String cardNo;  private String place;  private Date date;  //身份证对应的用户信息  private User user;  public User getUser() {  return user;  }  public void setUser(User user) {  this.user = user;  }  public String getCardNo() {  return cardNo;  }  public void setCardNo(String cardNo) {  this.cardNo = cardNo;  }  public String getPlace() {  return place;  }  public void setPlace(String place) {  this.place = place;  }  public Date getDate() {  return date;  }  public void setDate(Date date) {  this.date = date;  }  } |

User.hbm.xml:

|  |
| --- |
| <?xml version="1.0"?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  <hibernate-mapping package="lw.pers.bb">  <class name="User" table="t\_user">  <id name="id">  <generator class="native"></generator>  </id>  <property name="name" length="50"></property>  <property name="sex" type="character" length="2"></property>  <!--1对1,unique表示card\_id这个字段是唯一的-->  <many-to-one name="idCard" column="card\_id" class="IdCard" unique="true"></many-to-one>  </class>  </hibernate-mapping> |

IdCard.hbm.xml:

|  |
| --- |
| <?xml version="1.0"?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  <hibernate-mapping package="lw.pers.bb">  <class name="IdCard" table="t\_idcard">  <id name="cardNo">  <generator class="assigned"></generator>  </id>  <property name="place" length="200"></property>  <property name="date" type="date"></property>  <!--1对1映射-->  <one-to-one name="user" class="User"></one-to-one>  </class>  </hibernate-mapping> |

Test:

|  |
| --- |
| package lw.pers.bb;  import org.hibernate.Session;  import org.hibernate.SessionFactory;  import org.hibernate.cfg.Configuration;  import org.junit.Test;  import java.util.Date;  public class App {  private static SessionFactory sf;  static {  sf = new Configuration().configure()  .addClass(User.class)  .addClass(IdCard.class)  .buildSessionFactory();  }  //保存数据  @Test  public void save(){  Session session = sf.openSession();  session.beginTransaction();  //对象  User user = new User();  user.setName("Jack");  user.setSex('男');  IdCard idCard = new IdCard();  idCard.setCardNo("3412");  idCard.setPlace("上海");  idCard.setDate(new Date());  //关系  user.setIdCard(idCard);  //保存  session.save(idCard);  session.save(user);  session.getTransaction().commit();  session.close();  }  } |

方式2:

User.java:

|  |
| --- |
| package lw.pers.cc;  //用户对象  //1对1映射,方式1  public class User {  private String id;  private String name;  private char sex;  //用户关联的身份证信息,在数据库中是外键  private IdCard idCard;  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;  }  public char getSex() {  return sex;  }  public void setSex(char sex) {  this.sex = sex;  }  public IdCard getIdCard() {  return idCard;  }  public void setIdCard(IdCard idCard) {  this.idCard = idCard;  }  } |

IdCard.java: 和方式1一样

|  |
| --- |
| package lw.pers.cc;  import java.util.Date;  //身份证对象  public class IdCard {  private String cardNo;  private String place;  private Date date;  //身份证对应的用户信息  private User user;  public User getUser() {  return user;  }  public void setUser(User user) {  this.user = user;  }  public String getCardNo() {  return cardNo;  }  public void setCardNo(String cardNo) {  this.cardNo = cardNo;  }  public String getPlace() {  return place;  }  public void setPlace(String place) {  this.place = place;  }  public Date getDate() {  return date;  }  public void setDate(Date date) {  this.date = date;  }  } |

User.hbm.xml:

|  |
| --- |
| <?xml version="1.0"?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  <hibernate-mapping package="lw.pers.cc">  <class name="User" table="t\_user">  <id name="id">  <!--指定了主键生成策略为外键策略,即当前表把别的表的主键作为当前表的主键-->  <generator class="foreign">  <!--这个idCard就是对应的是one-to-one标签的name属性的值-->  <param name="property">idCard</param>  </generator>  </id>  <property name="name" length="50"></property>  <property name="sex" type="character" length="2"></property>  <!--特殊的1对1-->  <!--constrained=true,表示在主键上添加外键约束-->  <one-to-one name="idCard" class="IdCard" constrained="true"></one-to-one>  </class>  </hibernate-mapping> |

Idcard.hbm.xml: 和方式1中的一样

|  |
| --- |
| <?xml version="1.0"?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  <hibernate-mapping package="lw.pers.cc">  <class name="IdCard" table="t\_idcard">  <id name="cardNo">  <generator class="assigned"></generator>  </id>  <property name="place" length="200"></property>  <property name="date" type="date"></property>  <!--1对1映射-->  <one-to-one name="user" class="User"></one-to-one>  </class>  </hibernate-mapping> |

Test:

|  |
| --- |
| package lw.pers.cc;  import org.hibernate.Session;  import org.hibernate.SessionFactory;  import org.hibernate.cfg.Configuration;  import org.junit.Test;  import java.util.Date;  public class App {  private static SessionFactory sf;  static {  sf = new Configuration().configure().buildSessionFactory();  }  //保存数据  @Test  public void save(){  Session session = sf.openSession();  session.beginTransaction();  //对象  User user = new User();  user.setName("Jack");  user.setSex('男');  IdCard idCard = new IdCard();  idCard.setCardNo("3412");  idCard.setPlace("上海");  idCard.setDate(new Date());  //关系  user.setIdCard(idCard);  //保存  session.save(idCard);  session.save(user);  session.getTransaction().commit();  session.close();  }  } |

1对1:

方式1:

有外键方:

<many-to-one name=”” column =”” class=”” unique=”true”>

无外键方:

<one-to-tone name=”” class=””>

方式2:

有外键方:

<id name="id">

<generator class="foreign">

<param name="property">idCard</param>

</generator>

</id>

<one-to-one name=”idCard” class=””>

无外键方:

<one-to-tone name=”” class=””>

## 12.6.继承映射

Animal类为基类,Dog和Cat分别为子类

**推荐第四种方式**

### 方式1:只写子类的配置文件:

缺点:有多少个子类,就要写多少个映射文件,映射文件太多了

有多少个子类就要用多少张表来存储

Animal.java:

|  |
| --- |
| package lw.pers.dd;  public class Animal {  private int id;  private String name;  public int getId() {  return id;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public void setId(int id) {  this.id = id;  }  } |

Dog.java:

|  |
| --- |
| package lw.pers.dd;  //狗  public class Dog extends Animal{  //与主人玩耍  private String play;  public String getPlay() {  return play;  }  public void setPlay(String play) {  this.play = play;  }  } |

Dog.hbm.xml:

|  |
| --- |
| <?xml version="1.0"?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  <hibernate-mapping package="lw.pers.dd">  <class name="Dog" table="t\_dog">  <!--继承父类的属性,直接写-->  <id name="id">  <generator class="native"></generator>  </id>  <property name="name" length="20"></property>  <!--子类属性-->  <property name="play" length="20"></property>  </class>  </hibernate-mapping> |

test:

|  |
| --- |
| package lw.pers.dd;  import org.hibernate.SessionFactory;  import org.hibernate.cfg.Configuration;  import org.hibernate.classic.Session;  import org.junit.Test;  public class App {  private static SessionFactory sf;  static {  sf = new Configuration().configure()  .addClass(Dog.class)  .buildSessionFactory();  }  //保存  @Test  public void test(){  Session session = sf.openSession();  session.beginTransaction();  Dog dog = new Dog();  dog.setName("花皮狗");  dog.setPlay("接飞盘");  //保存  session.save(dog);  session.getTransaction().commit();  session.close();  }  //查询  @Test  public void test\_1(){  Session session = sf.openSession();  session.beginTransaction();  Dog dog = new Dog();  dog = (Dog)session.get(Dog.class,1);  System.out.println(dog);  session.getTransaction().commit();  session.close();  }  } |

+----+-----------+-----------+

| id | name | play |

+----+-----------+-----------+

| 1 | 花皮狗 | 接飞盘 |

+----+-----------+-----------+

### 方式2:所有子类配置写到一个文件中

用一张表存储所有子类信息,一个映射文件

总结:生成的表,不符合设计的原则,因为所有子类都用一张表存储,存在很多冗余字段

t\_animal

id name catching play type\_

1 猫 抓老鼠 null lw.pers.dd.Cat

2 狗 null 接飞盘 lw.pers.dd.Dog

type\_字段(鉴别器):区分不同子类的信息

Animal.java:

|  |
| --- |
| package lw.pers.dd;  public class Animal {  private int id;  private String name;  public int getId() {  return id;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public void setId(int id) {  this.id = id;  }  } |

Dog.java:

|  |
| --- |
| package lw.pers.dd;  //狗  public class Dog extends Animal{  //与主人玩耍  private String play;  public String getPlay() {  return play;  }  public void setPlay(String play) {  this.play = play;  }  @Override  public String toString() {  return "Dog{" +  "play='" + play + '\'' +  '}';  }  } |

Cat.java:

|  |
| --- |
| package lw.pers.dd;  public class Cat extends Animal{  //抓老鼠  private String catching;  public String getCatching() {  return catching;  }  public void setCatching(String catching) {  this.catching = catching;  }  } |

Animal.hbm.xml:

|  |
| --- |
| <?xml version="1.0"?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  <!--  整个继承结构一张表  -->  <hibernate-mapping package="lw.pers.dd">  <class name="Animal" table="t\_animal">  <!--继承父类的属性,直接写-->  <id name="id">  <generator class="native"></generator>  </id>  <!--鉴别器字段必须写,不然报错-->  <discriminator column="type\_"></discriminator>  <property name="name" length="20"></property>  <!--子类:猫-->  <subclass name="Cat">  <property name="catching"></property>  </subclass>  <!--子类:狗-->  <!--如果没有指定鉴别器字段的,则默认是类的全名-->  <subclass name="Dog" discriminator-value="dog\_">  <property name="play"></property>  </subclass>  </class>  </hibernate-mapping> |

Test:

|  |
| --- |
| package lw.pers.dd;  import org.hibernate.SessionFactory;  import org.hibernate.cfg.Configuration;  import org.hibernate.classic.Session;  import org.junit.Test;  public class App {  private static SessionFactory sf;  static {  sf = new Configuration().configure()  .addClass(Animal.class)  .buildSessionFactory();  }  @Test  public void test(){  Session session = sf.openSession();  session.beginTransaction();  Dog dog = new Dog();  dog.setName("花皮狗");  dog.setPlay("接飞盘");  Cat cat = new Cat();  cat.setCatching("抓老鼠");  cat.setName("黑猫");  //保存  session.save(dog);  session.save(cat);  session.getTransaction().commit();  session.close();  }  @Test  public void test\_1(){  Session session = sf.openSession();  session.beginTransaction();  Dog dog = new Dog();  dog = (Dog)session.get(Dog.class,1);  System.out.println(dog);  session.getTransaction().commit();  session.close();  }  } |

### 方式3:每个类一张表

总共三个表,1个基类两个子类,1一个映射文件

特点:完全按照面向对象设计表结构,符合数据库设计原则

缺点:但是表结构,关系边的复杂,影响数据库访问效率

t\_animal

id name

105 猫

106 狗

t\_dog t\_cat

id play id catching

106 接飞盘 105 抓老鼠

t\_dog有一个id外键,t\_cat有一个外键引用的是t\_animal表中的id

Animal.java:

|  |
| --- |
| package lw.pers.dd;  public class Animal {  private int id;  private String name;  public int getId() {  return id;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public void setId(int id) {  this.id = id;  }  } |

Dog.java:

|  |
| --- |
| package lw.pers.dd;  //狗  public class Dog extends Animal{  //与主人玩耍  private String play;  public String getPlay() {  return play;  }  public void setPlay(String play) {  this.play = play;  }  @Override  public String toString() {  return "Dog{" +  "play='" + play + '\'' +  '}';  }  } |

Cat.java:

|  |
| --- |
| package lw.pers.dd;  public class Cat extends Animal{  //抓老鼠  private String catching;  public String getCatching() {  return catching;  }  public void setCatching(String catching) {  this.catching = catching;  }  } |

Animal.hbm.xml:

|  |
| --- |
| <?xml version="1.0"?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  <!--  整个继承结构一张表  -->  <hibernate-mapping package="lw.pers.dd">  <class name="Animal" table="t\_animal">  <!--继承父类的属性,直接写-->  <id name="id">  <generator class="native"></generator>  </id>  <property name="name"></property>  <!--子类:猫-->  <joined-subclass name="Cat" table="t\_cat">  <!--指定id为外键-->  <key column="id"></key>  <property name="catching"></property>  </joined-subclass>  <!--子类:狗-->  <joined-subclass name="Dog" table="t\_dog">  <!--指定id为外键-->  <key column="id"></key>  <property name="play"></property>  </joined-subclass>  </class>  </hibernate-mapping> |

test:

|  |
| --- |
| package lw.pers.dd;  import org.hibernate.SessionFactory;  import org.hibernate.cfg.Configuration;  import org.hibernate.classic.Session;  import org.junit.Test;  public class App {  private static SessionFactory sf;  static {  sf = new Configuration().configure()  .addClass(Animal.class)  .buildSessionFactory();  }  @Test  public void test(){  Session session = sf.openSession();  session.beginTransaction();  Dog dog = new Dog();  dog.setName("花皮狗");  dog.setPlay("接飞盘");  Cat cat = new Cat();  cat.setCatching("抓老鼠");  cat.setName("黑猫");  //保存  session.save(dog);  session.save(cat);  session.getTransaction().commit();  session.close();  }  @Test  public void test\_1(){  Session session = sf.openSession();  session.beginTransaction();  Dog dog = new Dog();  dog = (Dog)session.get(Dog.class,1);  System.out.println(dog);  session.getTransaction().commit();  session.close();  }  } |

### 方式4:每个子类用一张表

共两张表,一个映射文件

Animal.java:

|  |
| --- |
| package lw.pers.dd;  public class Animal {  private int id;  private String name;  public int getId() {  return id;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public void setId(int id) {  this.id = id;  }  } |

Dog.java:

|  |
| --- |
| package lw.pers.dd;  //狗  public class Dog extends Animal{  //与主人玩耍  private String play;  public String getPlay() {  return play;  }  public void setPlay(String play) {  this.play = play;  }  @Override  public String toString() {  return "Dog{" +  "play='" + play + '\'' +  '}';  }  } |

Cat.java:

|  |
| --- |
| package lw.pers.dd;  public class Cat extends Animal{  //抓老鼠  private String catching;  public String getCatching() {  return catching;  }  public void setCatching(String catching) {  this.catching = catching;  }  } |

Animal.hbm.xml:

|  |
| --- |
| <?xml version="1.0"?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  <!--  整个继承结构一张表  -->  <hibernate-mapping package="lw.pers.dd">  <!--抽象类表示当前类不生成对应的表-->  <class name="Animal" abstract="true">  <!--继承父类的属性,直接写-->  <id name="id">  <!--注意使用union-subclass不能使用主键自动增长生成策略-->  <generator class="assigned"></generator>  </id>  <property name="name"></property>  <!--子类:猫-->  <union-subclass name="Cat" table="t\_cat">  <property name="catching"></property>  </union-subclass>  <!--子类:狗-->  <union-subclass name="Dog" table="t\_dog">  <property name="play"></property>  </union-subclass>  </class>  </hibernate-mapping> |

test:

|  |
| --- |
| package lw.pers.dd;  import org.hibernate.SessionFactory;  import org.hibernate.cfg.Configuration;  import org.hibernate.classic.Session;  import org.junit.Test;  public class App {  private static SessionFactory sf;  static {  sf = new Configuration().configure()  .addClass(Animal.class)  .buildSessionFactory();  }  @Test  public void test(){  Session session = sf.openSession();  session.beginTransaction();  Dog dog = new Dog();  dog.setId(100);  dog.setName("花皮狗");  dog.setPlay("接飞盘");  Cat cat = new Cat();  cat.setId(101);  cat.setCatching("抓老鼠");  cat.setName("黑猫");  //保存  session.save(dog);  session.save(cat);  session.getTransaction().commit();  session.close();  }  @Test  public void test\_1(){  Session session = sf.openSession();  session.beginTransaction();  Dog dog = new Dog();  dog = (Dog)session.get(Dog.class,1);  System.out.println(dog);  session.getTransaction().commit();  session.close();  }  } |

MariaDB [test]> select \* from t\_dog;

+-----+-----------+-----------+

| id | name | play |

+-----+-----------+-----------+

| 100 | 花皮狗 | 接飞盘 |

+-----+-----------+-----------+

MariaDB [test]> select \* from t\_cat;

+-----+--------+-----------+

| id | name | catching |

+-----+--------+-----------+

| 101 | 黑猫 | 抓老鼠 |

+-----+--------+-----------+

# 13.Inverse属性

Inverse:表示控制权是否转移

true:控制权转移,当前方没有控制权

false:控制权没有转移,当前方有控制权(默认为false)

这个属性是在1的一方,而不是多的一方:如<set> <list> ...，当然也多对多中也能使用

比如部门和员工的关系(部门id作为员工表的外键和员工表的user\_id是一样的)

Inverse:

为true:总体的意思是:只能保存各自的数据

1. 保存数据时,员工表的user\_id是为空的而不是等于部门id
2. 对获取数据没什么影响
3. 不能解除关系
4. 如果要删除1这边的数据的话,会删除失败.

# 14.cascade级联操作

cascade:表示级联操作,在1对多,多对1,1对1都可以设置

save-update 级联保存或更新

delete 级联删除

save-update,delete 级联保存,更新,删除

all 同上

none 不级联(默认值)

比如1对多的关系中 用户和地址是一对多的关系,如果是保存了用户,但是没有保存关联的地址,那么就会报错.如果在用户一方设置了cascade=”save-update” 那么如关联的一方地址不保存，只保存用户,那么地址会被自动保存

|  |
| --- |
| <?xml version="1.0"?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  <!--用户映射文件-->  <hibernate-mapping package="lw.pers.crud">  <class name="Users" table="t\_users">  <id name="id" column="id">  <generator class="native"></generator>  </id>  <property name="name" length="50"></property>  <property name="age"></property>  <!--1对多映射  -->  <set name="addresses" table="t\_addressSet" cascade="save-update">  <key column="user\_id"></key>  <one-to-many class="Address"></one-to-many>  </set>  </class>  </hibernate-mapping> |

|  |
| --- |
| Address address = new Address();  address.setName("上海");  address.setCode("51000");  address.setShortName("sh");  //用户  Users uses = new Users();  uses.setName("小张");  uses.setAge(25);  uses.getAddresses().add(address);  session.save(uses);  // session.save(address); |

如果想要修改用户的id,则必须先将用户关联的地址表的外键字段user\_id设置为Null,再改变用户表的id的值,最后在重新设置关联的地址的user\_id的值,级联可以帮我们简化此操作

可以这么写:

cascade=”save-update,delete”

# 15.二级缓存使用

一级缓存:

基于session的缓存

特点:

1. 在短时间内多次操作数据库情况下,缓存效果比较明显
2. session关闭后,就不能使用缓存内容

总结:

缓存只在当前session有效,缓存时间短,作用范围小,总体看缓存效果不明显

面试题:

不同的session,能否共享一级缓冲? 答案:当然不可以

二级缓存:

基于应用程序的缓存,基于sessionFactory级别的缓存

缓存数据可以被多个session共享,但需要指定哪些对象要放入二级缓存中

放入二级缓存中对象的特点:

1. 经常使用
2. 不会被经常修改

Hibernate提供的二级缓存是以缓存框架形式提供,hibernate提供了二级缓存框架默认

的实现,也支持其他二级缓存框架,如果要更换缓存,只要更换配置中具体的二级缓存框架使用的核心类即可,可×配的缓存框架。

开启二级缓存的步骤:

1. 开始二级缓存
2. 指定二级缓存具体实现框架
3. 哪些类要加入二级缓存
4. 测试

hibernate.cfg.xml中配置

|  |
| --- |
| <!--二级缓存配置-->  <!--1)开启二级缓存-->  <property name="hibernate.cache.use\_second\_level\_cache">true</property>  <!--2)指定使用哪一种二级缓存-->  <property name="hibernate.cache.provider\_class">org.hibernate.cache.HashtableCacheProvider</property>  <!—开启查询缓存,即使用hql查询也能进行二级缓存-->  <property name=”hibernate.cache.use\_query\_cache”>true</property>  <!--3)哪些类加入二级缓存的类-->  <class-cache class="lw.pers.crud.Users" usage="read-only"></class-cache>  <!—集合缓存-->  <collection-cache usage=”read-only” collection=”lw.pers.crud.Users. addresses”> |

<!--3)哪些类加入二级缓存的类-->

<class-cache class="lw.pers.crud.Users" usage="read-only"></class-cache>

这一行也可以这么写（不推荐这么写）

|  |
| --- |
| <class name="Users" table="t\_users">  <cache usage="read-only"></cache>  <id name="id" column="id">  <generator class="native"></generator>  </id>  <property name="name" length="50"></property>  <property name="age"></property>  <!--1对多映射  -->  <set name="addresses" table="t\_addressSet">  <key column="user\_id"></key>  <one-to-many class="Address"></one-to-many>  </set>  </class> |

缓存策略:

usage=”read-write” 二级缓存的数据可以读写

usage=”read-only” 二级缓存的数据只读

usage=”nonstrict-read-write” 非严格读取

usage=”transactional” 基于事务的策略

hql查询时如何开启二级缓存？

1）<property name=”hibernate.cache.use\_query\_cache”>true</property>

2）Query q = session1.createQuery(“from Dept”).setCacheable(true);

Query q = session2.createQuery(“from Dept”).setCacheable(true)

setCacheable(true):放入二级缓存或者从二级缓存中获取

更新数据:会不会通知一级缓存、二级缓存？

更新数据:二级缓存会检测到最新的数据,会通知二级缓存,但是不会通知一级缓存