# Hibernate的框架搭建

Hibernate是一款优秀的持久化ORM框架

持久化(Persistence),即把数据(如内存中的对象)保持到可以永久保存的存储设备中(如硬盘)；

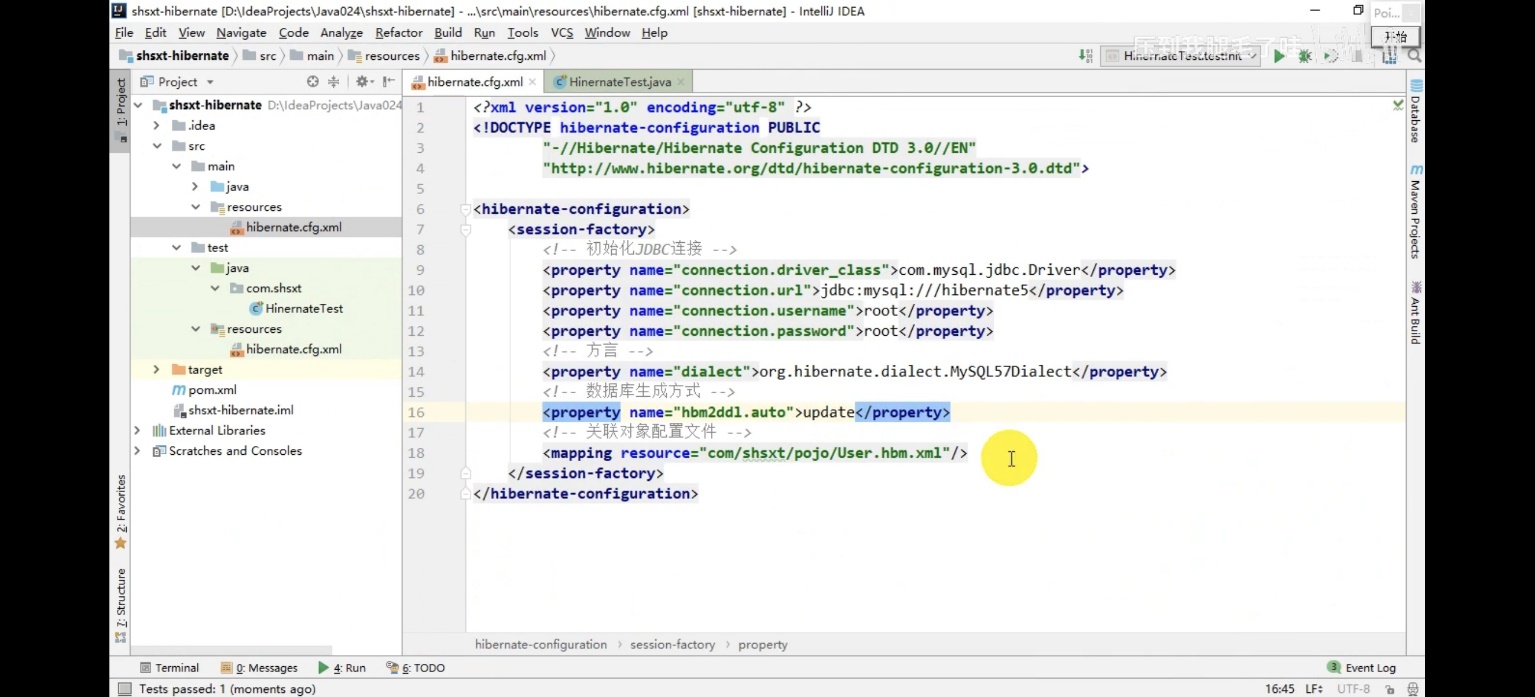
### ORM

O(OBject),R(Relation),M(Mapping),顾名思义就是对象(JavaBean)关系(关系型数据库)映射

## 1.创建一个Java项目，导入Hibernate框架相关依赖jar包

## 2.创建Hibernate的相关配置文件

1. Xxx.hbm.xml：它主要是用于描述类与数据库中的表的映射关系；
2. **hibernate.cfg.xml**：它是Hibernate框架的核心配置文件。



## 创建实体类和编写Xxx.hbm.xml

public class User implements Serializable {

private Integer id;

private String userName;

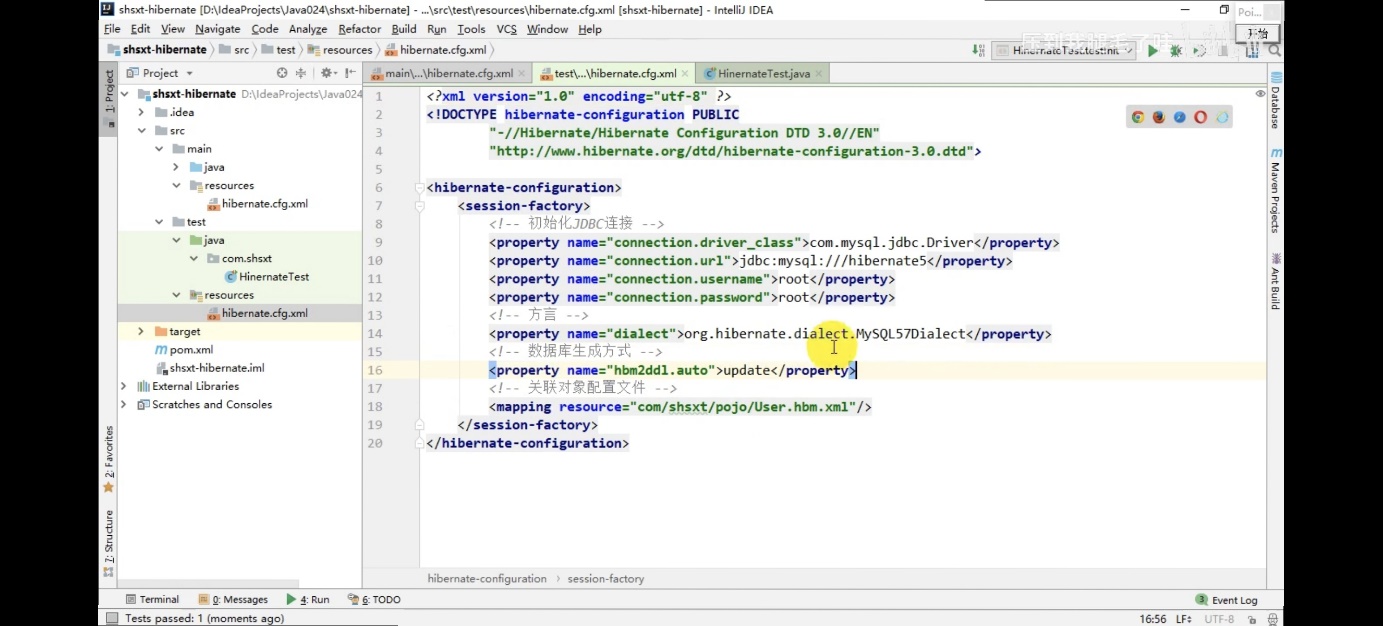
private String userPwd;

//省略getter、setter、构造器

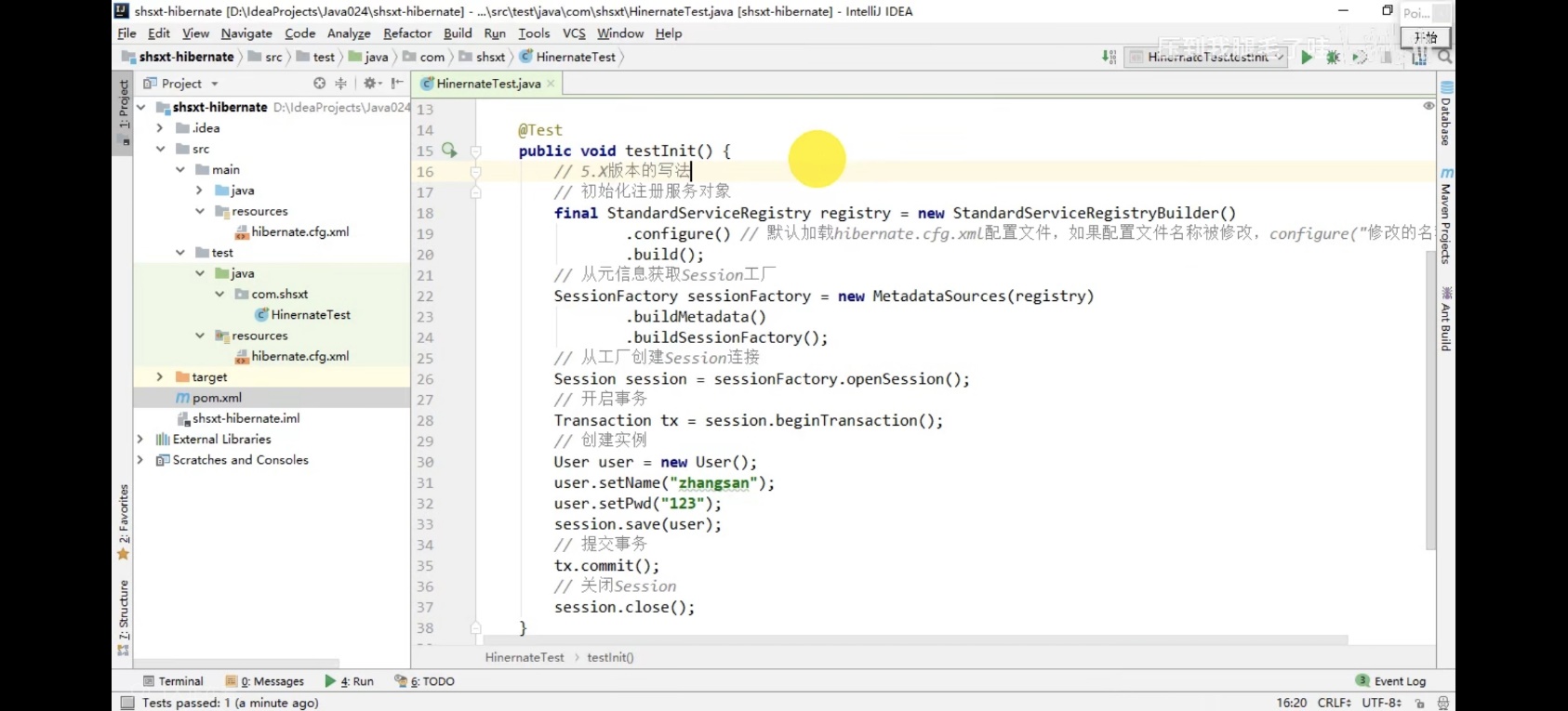
//一定要实现Serializable接口

}

**一般为POJO类名称，比如User.hbm.xml，该文件放在POJO类所在包下**。

1. 

## 4.测试类—生成数据库表



# 2．Hibernate五大关键接口

[Hibernate](https://so.csdn.net/so/search?q=Hibernate&spm=1001.2101.3001.7020)**五大关键接口**

在Hibernate的[API](https://so.csdn.net/so/search?q=API&spm=1001.2101.3001.7020)中有5个非常重要的接口：**Configuration、SessionFactory、Session、Transaction和Query**，它们是Hibernate组成的核心。

[Configuration](https://so.csdn.net/so/search?q=Configuration&spm=1001.2101.3001.7020" \t "_blank)

负责Hibernate配置工作，创建SessionFactory对象，在Hibernate启动过程中，Configuration类的实例首先定位在映射文件位置，读取配置，然后创建SessionFactory对象。

**SessionFactory**  
SessionFactroy接口负责初始化Hibernate。它充当数据存储源的代理，使用工厂模式创建[Session](https://so.csdn.net/so/search?q=Session&spm=1001.2101.3001.7020)对象。需要注意的是SessionFactory并不是轻量级的，一般情况下，一个项目通常只需要一个SessionFactory就够，当需要操作多个数据库时，可以为每个数据库指定一个SessionFactory。

**Session**

Session 接口对于Hibernate 开发人员来说是一个最重要的接口。在Hibernate中，实例化的Session是一个轻量级的类，创建和销毁它都不会占用很多资源。这在实际项目中非常很重要，因为在程序中，可能会不断地创建以及销毁Session对象，如果Session 的开销太大，会给系统带来不良影响。但是Session对象是非线程安全的，因此在你的设计中，最好是一个线程只创建一个Session对象。 Session可以看作介于数据连接与事务管理一种中间接口。我们可以将Session想象成一个持久对象的缓冲区，Hibernate能检测到这些持久对象的改变，并及时刷新数据库。我们有时也称Session是一个持久层管理器，因为Session负责执行被持久化对象的增、删、改、查操作，类似于JDBC的Connection和Statement， 诸如存储持久对象至数据库，以及从数据库从获得它们。需要注意的是，Hibernate的Session不同于JSP 应用中的HttpSession。在Hibernate中，我们使用Session这个术语时，指的是Hibernate 自己的Session。

**Transaction**

负责事务相关的操作，它代表的是Hibernate事务，本质上也是数据库事务。一般在Hibernate的增删改中出现，但是使用Hibernate一般使用Spring去管理事务。

**Query**

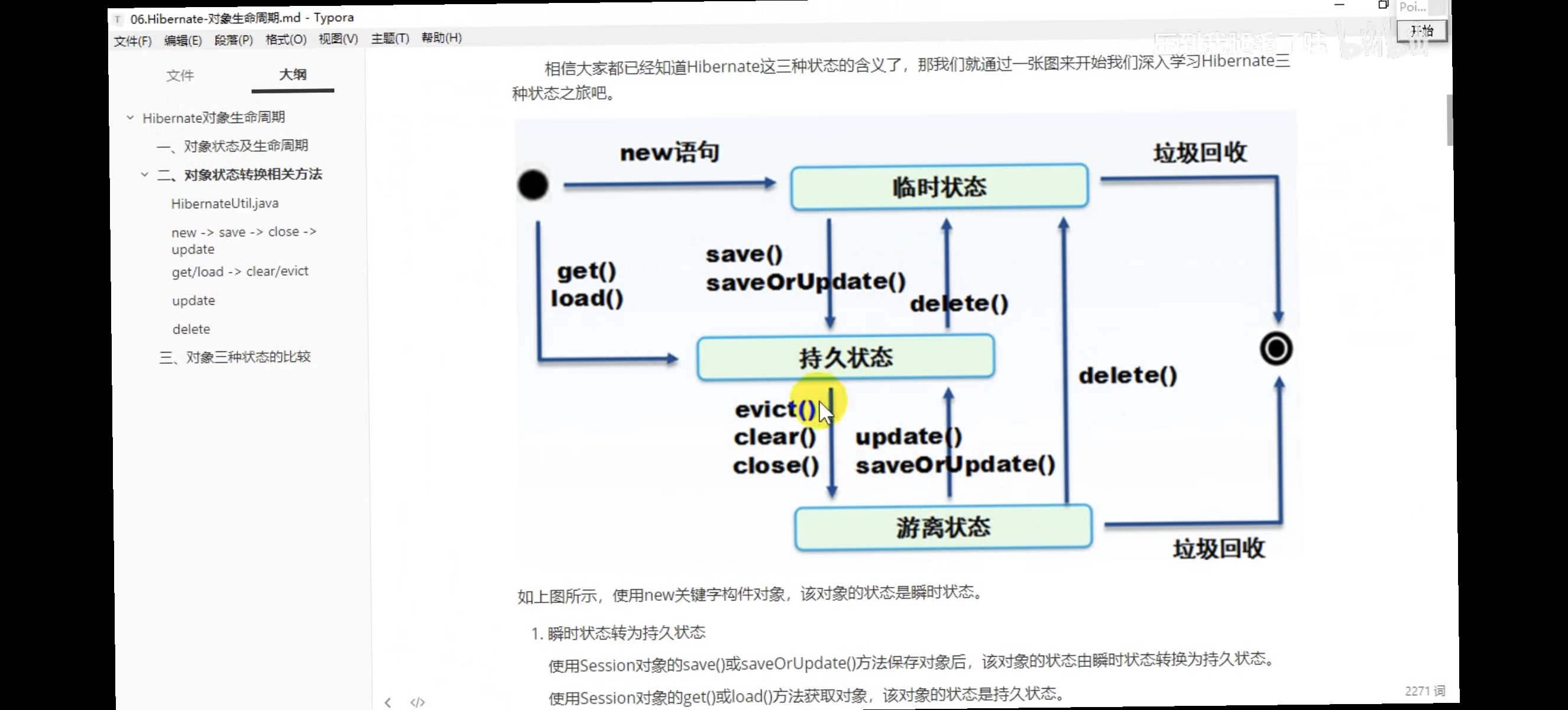
负责执行各种数据查询功能，它可以使用Hibernate特有的HQL语言和SQL语言两种方式。

# 3. Hibernate对象生命周期

Hibernate中对象有三种状态： 瞬时状态(Transient)、持久状态(Persistent)、游离状(Detached)。

* **瞬时状态**：刚刚使用new语句创建，还没有被持久化，不处于Session的缓存中。处于临时状态的Java对象被称为临时对象。**Session中没有，数据库中没有**。
* **持久化状态**：已经被持久化，加入到Session的缓存中。处于持久化状态的Java对象被称为持久化对象。**Session中有，数据库中有**。
* **游离状态**：已经被持久化，但不处于Session的缓存中。处于游离状态的Java对象被称为游离对象。**Session中没有，数据库中有**

## 3.1对象状态转换相关方法



1. 瞬时状态转为持久状态  
   使用Session对象的save()或saveOrUpdate()方法保存对象后，该对象的状态由瞬时状态转换为持久状态。使用Session对象的get()或load()方法获取对象，该对象的状态是持久状态。
2. 持久状态转为瞬时状态  
   执行Session对象的delete()方法后，对象由原来的持久状态变为瞬时状态，因为此时该对象没有任何的数据库数据关联。
3. 持久状态转为游离状态  
   执行了Session对象的evict()、clear()或close()方法，对象由原来的持久状态转为游离状态。
4. 游离状态转为持久状态  
   重新获取Session对象，执行Session对象的update()或saveOrUpdate()方法，对象由游离状态转为持久状态，该对象再次与Session对象相关联。
5. 游离状态转为瞬时状态  
   执行Session对象的delete()方法，对象由游离状态转为瞬时状态。处于瞬时状态或游离状态的对象不再被其他对象引用时，会被Java虚拟机按照垃圾回收机制处理。

### 3.1.1 封装一个可重复调用的HibernateUtil.java工具类

public class HibernateUtil {

private static final ThreadLocal<Session> THREAD\_LOCAL = new ThreadLocal<Session>();

private static SessionFactory sessionFactory = null;

private static StandardServiceRegistry serviceRegistry = null;

// 通过静态块初始化对象

static{

try {

serviceRegistry = new StandardServiceRegistryBuilder().configure().build();

sessionFactory = new MetadataSources(serviceRegistry).buildMetadata().buildSessionFactory();

}catch (Exception e ){

System.out.println("创建SessionFactory失败");

e.printStackTrace();

}

}

// 获取session

public static Session getSession(){

Session session = THREAD\_LOCAL.get();

if (null == session || !session.isOpen()){

if (null ==sessionFactory){

rebuildSessionFactoty();

}

session = (null != sessionFactory) ? sessionFactory.openSession() : null;

THREAD\_LOCAL.set(session);

}

return session;

}

// 初始化SessionFactory

public static void rebuildSessionFactoty(){

try {

serviceRegistry = new StandardServiceRegistryBuilder().configure().build();

sessionFactory = new MetadataSources(serviceRegistry)

.buildMetadata().buildSessionFactory();

} catch (Exception e) {

System.out.println("创建SessionFactory失败");

e.printStackTrace();

}

}

// 关闭session

public static void closeSession(){

Session session = THREAD\_LOCAL.get();

THREAD\_LOCAL.set(null);

if (null != session && session.isOpen()){

session.close();

}

}

}

### 3.1.2 new -> save -> close -> update

@Test

public void testSave() {

Session session = null;

Transaction tx = null;

User user = null;

try {

session = HibernateUtil.getSession();

tx = session.beginTransaction();

**// 构造对象 -> 瞬时状态，session中没有，数据库中没有**

user = new User();

user.setUsername("刘德华");

user.setPassword("123");

**// 调用save() -> 持久状态，user被session管理，session中有，数据库中有**

**session.save(user);**

/\*

\* **在持久状态下,脏数据检查:当提交事务时或清理缓存时,发现session中的数据和**

**\* 数据库中的数据不一致时,将会把session中的数据更新到数据库中**

\*/

user.setUsername("张学友");

**// 在保存以后再修改对象将会产生多条sql语句,效率较低,建议在save前修改**

session.flush();

tx.commit();

} catch (Exception e) {

e.printStackTrace();get/load -> clear/evict

tx.rollback();

} finally {

HibernateUtil.closeSession();

}

**// session被关闭 -> 游离状态，session中没有，数据库中有**

System.out.println("姓名:" + user.getUsername());

user.setUsername("梁朝伟");

try {

session = HibernateUtil.getSession();

tx = session.beginTransaction();

**// 调用update() -> 持久状态，user被session管理，session中有，数据库中有**

**// 如果此时先get()|load()获取到user -> 持久状态，session中有，数据库中有**

**// 再调用delete() -> 瞬时状态，sesison中没有，数据库中没有**

session.update(user);

tx.commit();

} catch (Exception e) {

e.printStackTrace();

tx.rollback();

} finally {

HibernateUtil.closeSession();

}

// 游离状态

}

### 3.1.3 get/load -> clear/evict

@Test

public void testGet1() {

Session session = null;

Transaction tr = null;

User user = null;

try {

session = HibernateUtil.getSession();

tr = session.beginTransaction();

**// get() -> 持久状态，user被session管理，session中有，数据库中有**

// get()会立即查询该对象：范围从session，SessionFactory，数据库

user = (User) session.get(User.class, 1);

System.out.println("姓名:" + user.getUsername());

tr.commit();

**// clear()清除session缓存中所有对象，evict()清除指定对象**

session.clear();

// session.evict(user);

// clear()|evict() -> 游离状态，不被session管理，数据库中不会被更改

user.setUsername("张国荣");

System.out.println(user.getUsername());

} catch (Exception e) {

e.printStackTrace();

tr.rollback();

} finally {

HibernateUtil.closeSession();

}

/\***get/load的区别：get会立即去查询对象，load在使用才去查询（懒加载），get找不到对象时返回null，load找不到对象时抛异常**。\*/

}

@Test

public void testGet2() {

Session session = null;

Transaction tr = null;

User user = null;

try {

session = HibernateUtil.getSession();

tr = session.beginTransaction();

**// get() -> 持久状态，user被session管理，session中有，数据库中有**

**// get()会立即查询该对象：范围从session，SessionFactory，数据库**

**// get()如果找不到对象不会抛异常，返回null**

user = (User) session.get(User.class, 10);

System.out.println("姓名:" + user.getUsername());

tr.commit();

} catch (Exception e) {

e.printStackTrace();

tr.rollback();

} finally {

HibernateUtil.closeSession();

}

}

@Test

public void testLoad() {

Session session = null;

Transaction tr = null;

User user = null;

try {

session = HibernateUtil.getSession();

tr = session.beginTransaction();

**// load() -> 持久状态**

**// load()不会立即去查询对象，到使用时才会查询(懒加载)：范围从session，SessionFactory，数**

**//据库**

**// load()当对象不存在时会抛出org.hibernate.ObjectNotFoundException异常**

user = (User) session.load(User.class, 10);

System.out.println("姓名:" + user.getUsername());

tr.commit();

} catch (Exception e) {

e.printStackTrace();

tr.rollback();

} finally {

HibernateUtil.closeSession();

}

}

**get/load的区别：get会立即去查询对象，load在使用才去查询（懒加载），get找不到对象时返回null，load找不到对象时抛异常**。

### 3.1.4 update

@Test

public void testUpdate() {

Session session = null;

Transaction tr = null;

User user = null;

try {

session = HibernateUtil.getSession();

tr = session.beginTransaction();

// 手动构造的瞬时状态对象也可以修改，但是需要指定所有属性，不建议使用

//user = new User();

//user.setId(3);

//user.setUsername("李四");

// get() -> 持久状态，user被session管理，session中有，数据库中有

user = (User) session.get(User.class, 2);

// 通过从数据库中加载该对象然后再修改可以进行判断进而避免异常，提高程序的健壮性

if (null != user) {

user.setUsername("老王");

**// update() -> 持久状态，user被session管理，session中有，数据库中有**

session.update(user);

}

tr.commit();

} catch (Exception e) {

e.printStackTrace();

tr.rollback();

} finally {

HibernateUtil.closeSession();

}

}

**注意：先获取对象进行判断再更新，可以避免异常，提高程序的健壮性**

### 3.1.5 delete

@Test

public void testDelete() {

Session session = null;

Transaction tr = null;

User user = null;

try {

session = HibernateUtil.getSession();

tr = session.beginTransaction();

// 手动构造的瞬时状态对象，指定主键也是可以删除该对象的，但是不建议这么用

//user = new User();

//user.setId(5);

// get() -> 持久状态，user被session管理，session中有，数据库中有

user = (User) session.get(User.class, 10);

// 通过从数据库中加载该对象然后删除可以进行判断进而避免异常,提高程序的健壮性

if (null != user) {State Mem Session DB

**// delete() -> 瞬时状态，session中没有，数据库中没有**

session.delete(user);

}

tr.commit();

} catch (Exception e) {

e.printStackTrace();

tr.rollback();

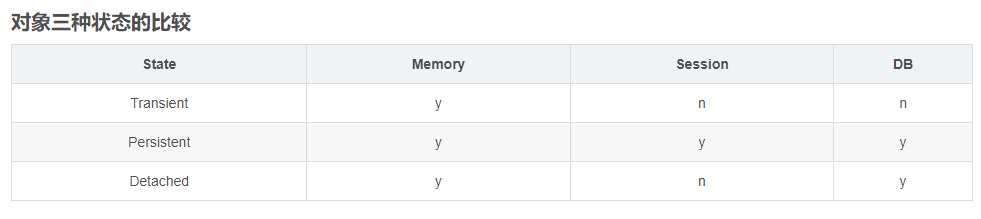
} finally {

HibernateUtil.closeSession();

}

}

**注意：先获取对象进行判断再删除，可以避免异常，提高程序的健壮性。**

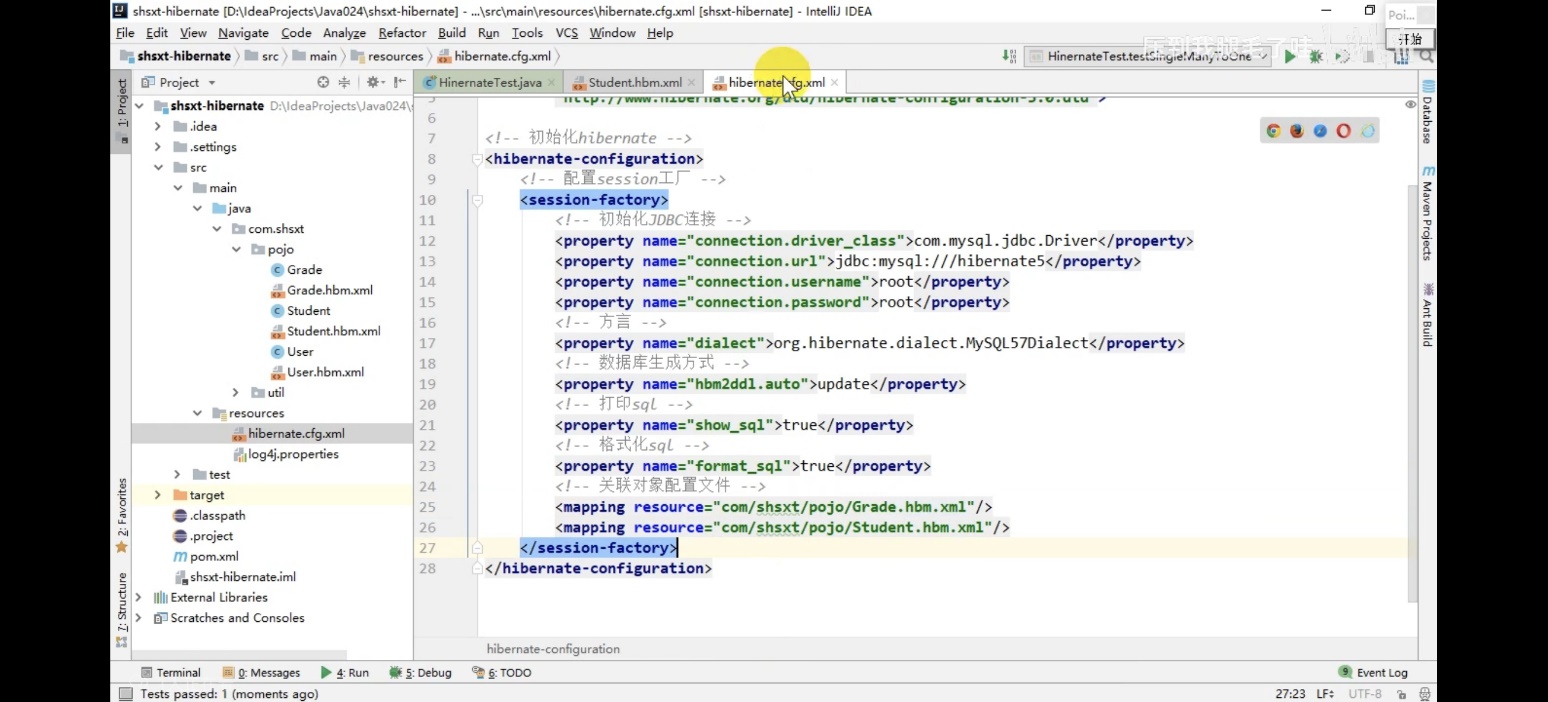


[(183条消息) Hibernate五大关键接口\_小白说丶的博客-CSDN博客](https://blog.csdn.net/ch19940812/article/details/84639244?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522167823887816800213069315%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=167823887816800213069315&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-1-84639244-null-null.142%5ev73%5econtrol,201%5ev4%5eadd_ask,239%5ev2%5einsert_chatgpt&utm_term=Hibernate%E4%BA%94%E5%A4%A7%E5%85%B3%E9%94%AE%E6%8E%A5%E5%8F%A3&spm=1018.2226.3001.4187)

# 4.Hibernate的映射关系

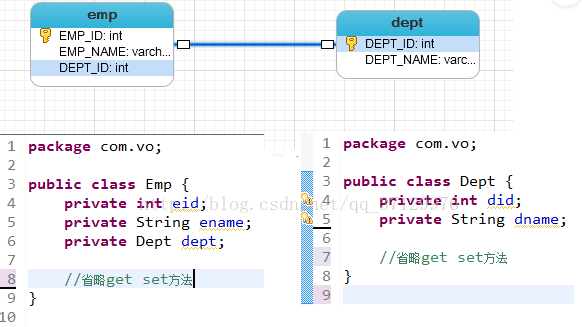
**单向关联**：单向关联是指只有一方有另一方的关联信息而另一方没有关联信息

**双向关联**：双向关联是指两方都有另一方的关联信息



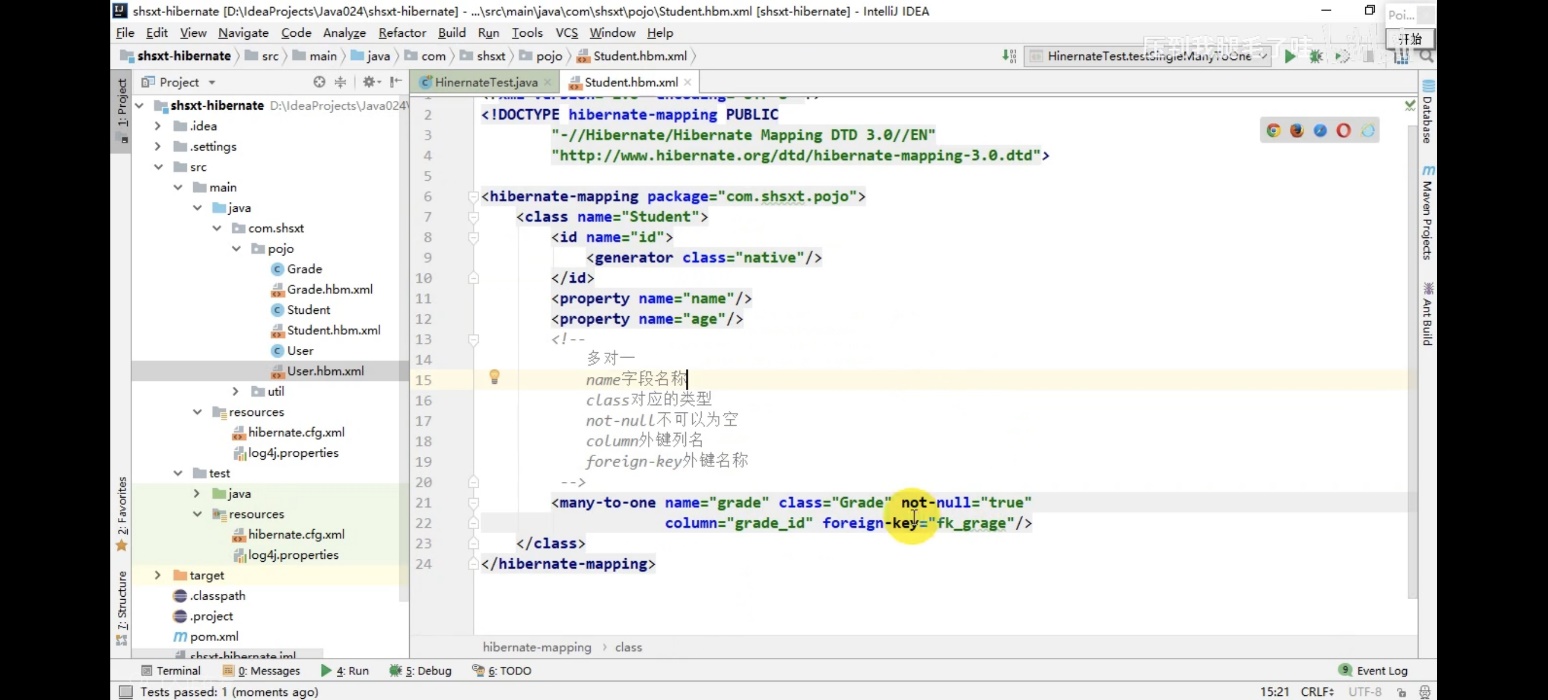
## 4.1单向多对一

两个对象之间多对一的关系，比如员工（emp）和部门（dept），多个员工可以在一个部门，映射原理就是**在多的一端添加一个外键指向一的一端的主键**。



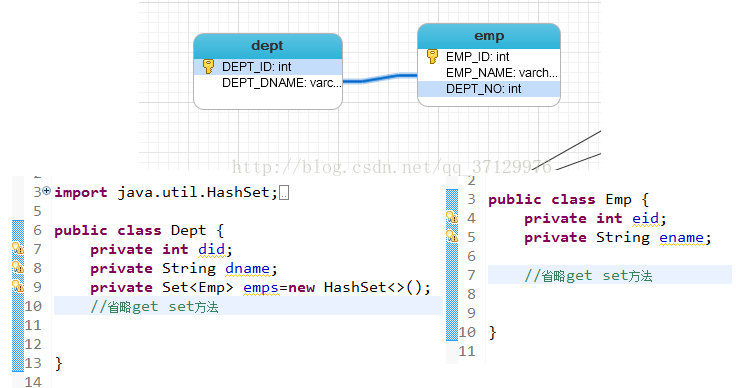
在**多的一端**的映射文件加入下面映射标签：





## 4.2 单向一对多

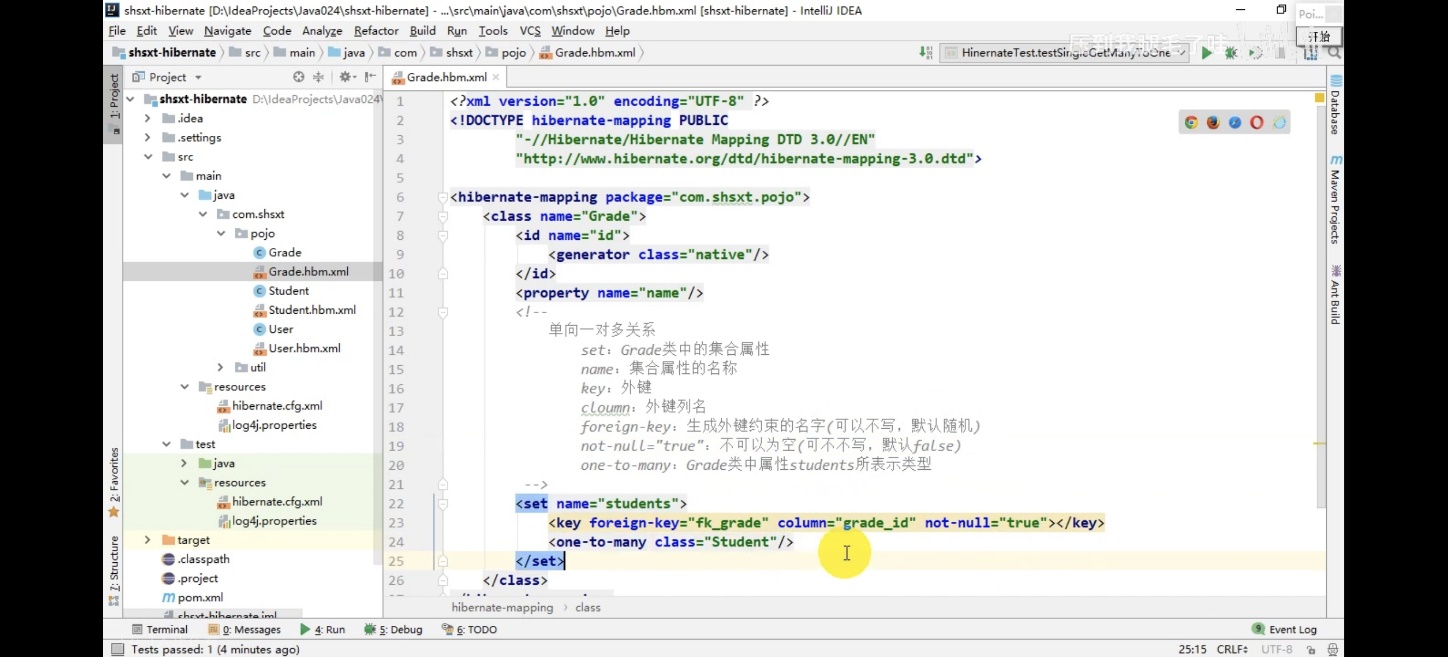
单向一对多的映射和单向多对一的映射原理相同，**都是在多的一端加入外键指向一的一端**



  注意：它与多对一的区别是维护的关系不同  
  
  
 \*多对一维护的关系是：多指向一的关系，有了此关系，加载多的时候可以将一加载上来  
  
  
 \*一对多维护的关系是：一指向多的关系，有了此关系，在加载一的时候可以将多加载上来

在**一的一端**的映射文件加入下面映射标签：







## 4.3 双向一对多

单向多对一，查询多能把一查出来，而查询一不能把多查出来，双向多对一就解决了这个问题

**在单向多对一的基础上修改，在Dept封装类中加入一个set集合属性**，并初始化private Set<Emp> emps=new HashSet<>();然后生成get set方法，之后修改Dept类对应的映射文件，加入下列代码：



其中set节点的name属性值为集合对应的属性名，table为set中的元素对应的记录放在哪一个数据表中，key中column指定多的表中的外键列的名字，one-to-many中class值是指定映射类型，（类的路径）

## 4.4 cascade(级联)

<set>的三个属性：

**cascade(级联)**

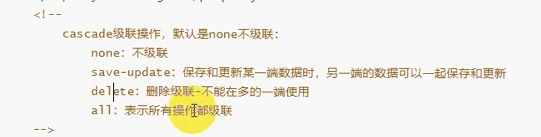
1.none：当Session操纵当前对象时，忽略其他关联的对象。它是cascade属性的默认值

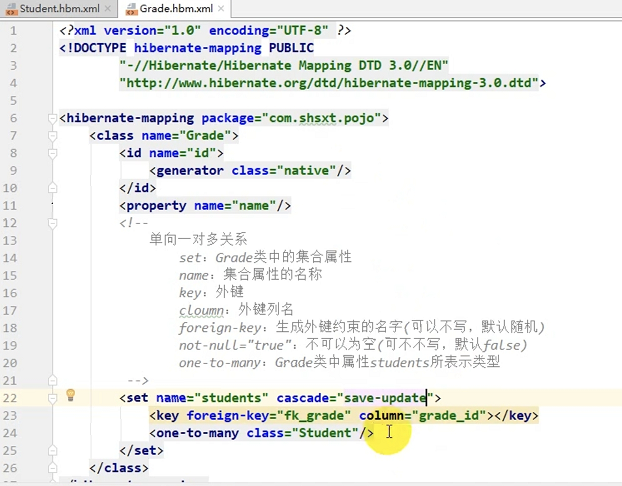
2.**save-update**：当通过Session的save()、update()及saveOrUpdate()方法来保存或更新当前对象时。

3.delete：当通过Session的delete()方法删除当前对象时，会级联删除所有关联的对象。

4.delete-orphan：删除孤儿。

5.all:包含save-update，delete的行为。







## 4.5 inverse(反转)

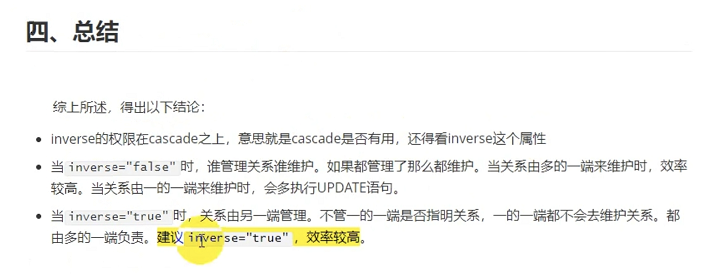
**inverse(反转)----在一端设置即可**

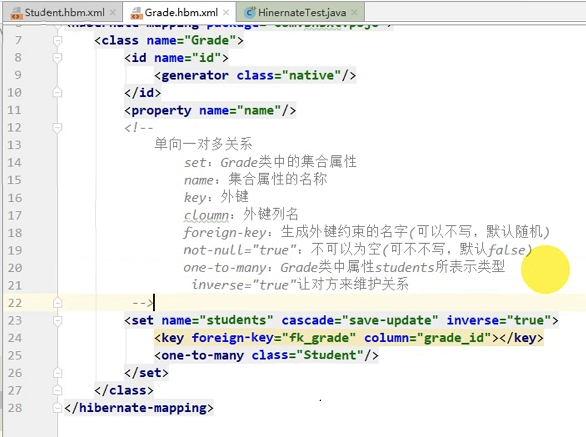
1.**inverse 决定由双向关联哪一方来维护关联关系**

2.inverse设置为false，则为主动方，主动负责维护关联关系，默认是false 。

3.inverse设置为true，则为被动方，由多的一端负责维护关联关系。

**一般将一的一端设为true，以使多的一方维护关联关系，有效减少update语句**





order-by

在查询时，对集合(set)中的元素进行排序，使用表的字段名，而不是属性名

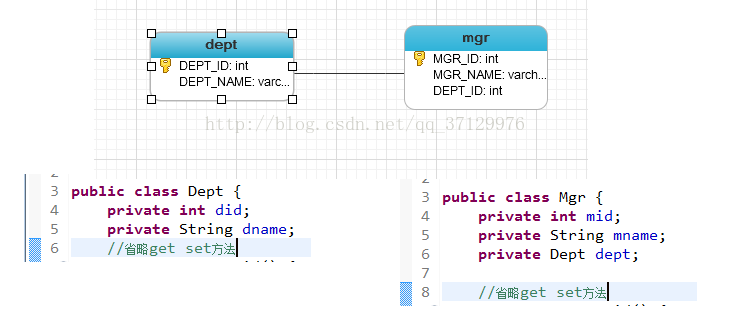
## 4.6单向一对一

**单向一对一：通过控制单向多对一中多的一端唯一，有此便形成了单向一对一**

一对一关联关系也是很常见的关联关系，如部门--部门经理，汽车--车位,人--身份证等等。

有**两种策略**可以实现一对一的关联映射：

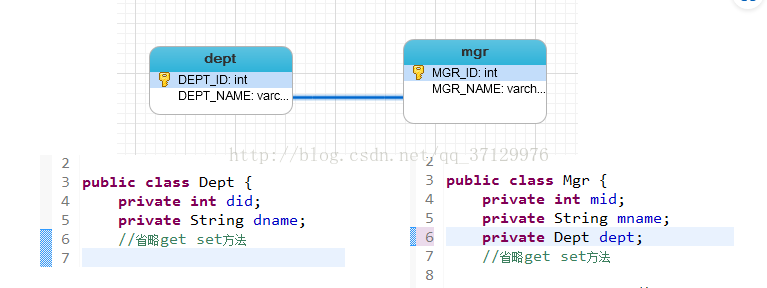
**4.6.1唯一外键关联：**外键关联，本来是用于多对一的配置，但是如果加上唯一的限制之后，也可以用来表示一对一关联关系



**在mrg类对应映射文件加入下列代码（多的）**

**采用<many-to-one>标签来映射，指定多的一端unique为true，这样就限制了多的一端的多重性为一。**

**4.6.2主键关联**：即让两个对象具有相同的主键值，一个表中的外键列即主键列，以表明它们之间的一一对应的关系；数据库表不会有额外的字段来维护它们之间的关系，仅通过表的主键来关联



在mrg类对应映射文件加入下列代码（多的）

<hibernate-mapping>

<class name="com.vo.Mgr" table="MGR">

<id name="mid" column="MGR\_ID">

<!-- 使用外键的方式生成主键 -->

<generator class="foreign">

<!-- property属性指定使用当前哪个持久化类的哪一个主键生成本表的主键，与one-to-one连用 -->

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

</generator>

</id>

<property name="mname" column="MGR\_NAME"></property>

<!-- 采用foreign主键生成器的一端使用one to one节点映射关联属性， -->

<!-- constrained="true"在当前主键上加上外键约束 -->

<one-to-one name="dept" class="com.vo.Dept" **constrained="true"**></one-to-one>

</class>

</hibernate-mapping>

## **4.7双向一对一**

**4.7.1对比单向一对一**进行映射，需要在dept类中加入emp类对象作为成员属性，然后在dept映射文件中加入<one-to-one><one-to-one>标签

**双向一对一唯一外键映射关键映射代码——在dept端新加入如下标签映射**：



**注意：一对一唯一外键关联双向采用<one-to-one>标签映射，必须指定<one-to-one>标签中的property-ref属性为关系字段的名称**

**4.7.2双向一对一主键映射**关键映射代码——在dept端新加入如下标签映射：



[(184条消息) hibernate映射关系\_hibernate定义映射关系\_Kayyz丶的博客-CSDN博客](https://blog.csdn.net/qq_37129976/article/details/78273799?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522167826406816782428659685%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=167826406816782428659685&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-1-78273799-null-null.142%5ev73%5econtrol,201%5ev4%5eadd_ask,239%5ev2%5einsert_chatgpt&utm_term=Hibernate%E7%9A%84%E6%98%A0%E5%B0%84%E5%85%B3%E7%B3%BB&spm=1018.2226.3001.4187)

