重点内容

```
1.Set集合的映射
2.关联关系的维护
3.Hiberbate中对象的四种状态以及相互转换
4.session对象方法
```

笔记整理

一、集合的映射

1.值类型的集合

```
1.1 Set集合：
    HashSet：无序、不重复
    TreeSet
    LinkedHashSet
1.2 List集合
    有序、可重复
1.3 Map集合
    数据结构：键值对
    HashMap：无序、不重复（指的是Key）
1.4 数组

1.5 Bag
    无序、可重复（与Set集合类似）
    List

1.6 注意：使用集合属性时，一定要使用接口来声明而不能使用具体的实现类声明。因为经过session的操作后，集合就变成了Hibernate自己的集合实现类。
```

四个集合的映射配置示例：

```
<?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="cn.itcast.e_hbm_collection">

<class name="User" table="user">
    <id name="id">
        <generator class="native"></generator>
    </id>
    <property name="name"/>

    <!-- addressSet属性，Set集合
        table属性：集合表的名称
        key子元素：集合外键的列名
        element子元素：存放集合元素的列的信息
        sort属性："unsorted|natural|comparatorClass"
            默认为：unsorted
        order-by属性：写的是order by 子句，是SQL语句，是操作的集合表。
            这是在查询数据时指定orderby子句。
    -->
    <set name="addressSet" table="user_addressSet" order-by="address ASC">
        <key column="userId"></key>
        <element type="string" column="address"></element>
    </set>

    <!-- addressList属性，List集合
```

```
        list-index：用于存放索引的列
    -->
    <list name="addressList" table="user_addressList">
        <key column="userId"></key>
        <list-index column="idx"></list-index>
        <element type="string" column="address"></element>
    </list>

    <!-- addressArray属性，数组。与List的映射基本一致 -->
    <array name="addressArray" table="user_addressArray">
        <key column="userId"></key>
        <list-index column="idx"></list-index>
        <element type="string" column="address"></element>
    </array>

    <!-- addressMap属性，Map集合 -->
    <map name="addressMap" table="user_addressMap">
        <key column="userId"></key>
        <map-key type="string" column="key_"></map-key>
        <element type="string" column="address"></element>
    </map>

    <!-- addressBag属性，Bag集合：无序，可重复。与Set集合的映射基本一致 -->
    <bag name="addressBag" table="user_addressBag">
        <key column="userId"></key>
        <element type="string" column="address"></element>
    </bag>
</class>
</hibernate-mapping>
```

2.实体类型的集合

集合的元素就是另一个实体

2.1 一对多（oneToMany）

就是设置外键列的值：

```
以员工和部门为例：
1.实体的设计：
Department：
private Integer id;
private String name;
private Set<Employee> employees = new HashSet<Employee>(); // 关联的很多员工
----------------------
Employee：
private Integer id;
private String name;
private Department department; // 关联的部门对象
=====================================
2.映射的配置
<hibernate-mapping package="cn.itcast.f_hbm_oneToMany">

<class name="Employee" table="employee">
    <id name="id">
        <generator class="native"></generator>
    </id>
    <property name="name" type="string" column="name"/>

    <!-- department属性，表达的是本类与Department的多对一
        class属性：关联的实体类型
        column属性：外键列（引用关联对象的表的主键）
    -->
    <many-to-one name="department" class="Department" column="departmentId"></many-to-one>
</class>
</hibernate-mapping>
```

```
------------------------------
<hibernate-mapping package="cn.itcast.f_hbm_oneToMany">

<class name="Department" table="department">
    <id name="id">
        <generator class="native"></generator>
    </id>
    <property name="name"/>

    <!-- employees属性，Set集合，表达的是本类与Employee的一对多
        class属性：关联的实体类型
        key子元素：对方表中的外键列（多方的那个表）

        inverse属性:
            默认为false，表示本方维护关联关系。
            如果为true，表示本方不维护关联关系。
            只是影响是否能设置外键列的值（设成有效值或是null值），对获取信息没有影响。

        cascade属性:
            默认为none，代表不级联。
            级联是指操作主对象时，对关联的对象也做相同的操作。
            可设为：delete, save-update, all, none ...
    <set name="employees" cascade="all">
        <key column="departmentId"></key>
        <one-to-many class="Employee"/>
    </set>
    -->
</class>
==================================================
3.操作
// 保存，有关联关系
@Test
public void testSave() throws Exception {
    Session session = sessionFactory.openSession();
    session.beginTransaction();
    // ----------------------------------------

    // 新建对象
    Department department = new Department();
    department.setName("开发部");

    Employee employee1 = new Employee();
    employee1.setName("张三");

    Employee employee2 = new Employee();
    employee2.setName("李四");

    // 关联起来
    employee1.setDepartment(department);
    employee2.setDepartment(department);
    department.getEmployees().add(employee1);
    department.getEmployees().add(employee2);

    // 保存
//  session.save(employee1);
//  session.save(employee2);
    session.save(department); // 保存部门

    // ----------------------------------------
    session.getTransaction().commit();
    session.close();
}

----------------------------------
// 解除关联关系
@Test
public void testRemoveRelation() throws Exception {
```

```
    Session session = sessionFactory.openSession();
    session.beginTransaction();
    // ------------------------------------------

    // // 从员工方解除
    // Employee employee = (Employee) session.get(Employee.class, 1);
    // employee.setDepartment(null);

    // 从部门方解除（与inverse有关系，为false时可以解除）
    Department department = (Department) session.get(Department.class, 1);
    department.getEmployees().clear();

    // ------------------------------------------
    session.getTransaction().commit();
    session.close();
}
-----------------------------------------------
// 删除对象，对关联对象的影响
@Test
public void testDelete() throws Exception {
    Session session = sessionFactory.openSession();
    session.beginTransaction();
    // ------------------------------------------

    // // 删除员工方（多方），对对方没有影响
    // Employee employee = (Employee) session.get(Employee.class,2);
    // session.delete(employee);

    // 删除部门方（一方）
    // a，如果没有关联的员工：能删除。
    // b，如果有关联的员工且inverse=true，由于不能维护关联关系，所以会直接执行删除，就会有异常
    // c，如果有关联的员工且inverse=false，由于可以维护关联关系，他就会先把关联的员工的外键列设为null值，再删除自己。
    Department department = (Department) session.get(Department.class, 4);
    session.delete(department);

    // ------------------------------------------
    session.getTransaction().commit();
    session.close();
}
```

关于使用级联（cascade）的建议:

- 多对多或者多对一关系建议不用级联
- 一对多或者一对一的关系建议使用级联

2.2 多对多（manyToMany）

就是插入或者删除中间表的记录

```
以学生和老师为例：
2.2.1 实体的设计
Teacher：
private Long id;
private String name;
private Set<Student> students = new HashSet<Student>(); // 关联的学生们
-----------------------
Student:
private Long id;
private String name;
private Set<Teacher> teachers = new HashSet<Teacher>(); // 关联的老师们
================================================
2.2.2 映射配置
<class name="Teacher" table="teacher">
    <id name="id">
        <generator class="native"></generator>
    </id>
```

```
    <property name="name" type="string" column="name"/>

    <!-- students属性，Set集合。
        表达的是本类与Student的多对多。
     -->
    <set name="students" table="teacher_student" inverse="true">
        <key column="teacherId"></key>
        <many-to-many class="Student" column="studentId"></many-to-many>
    </set>
</class>
---------------------------------
<class name="Student" table="student">
    <id name="id">
        <generator class="native"></generator>
    </id>
    <property name="name"/>

    <!-- teachers属性，Set集合。
        表达的是本类与Teacher的多对多。

        table属性：中间表（集合表）
        key子元素：集合外键（引用当前表主键的那个外键）
     -->
    <set name="teachers" table="teacher_student" inverse="false">
        <key column="studentId"></key>
        <many-to-many class="Teacher" column="teacherId"></many-to-many>
    </set>

</class>
================================================
2.2.3 操作
@Test
public void testSave() throws Exception {
    Session session = sessionFactory.openSession();
    session.beginTransaction();
    // ----------------------------------------

    // 新建对象
    Student student1 = new Student();
    student1.setName("王同学");

    Student student2 = new Student();
    student2.setName("李同学");

    Teacher teacher1 = new Teacher();
    teacher1.setName("赵老师");

    Teacher teacher2 = new Teacher();
    teacher2.setName("蔡老师");

    // 关联起来
    //关联关系的指定由老师或者学生其中一方来指定就行了
    student1.getTeachers().add(teacher1);
    student1.getTeachers().add(teacher2);
    student2.getTeachers().add(teacher1);
    student2.getTeachers().add(teacher2);

    teacher1.getStudents().add(student1);
    teacher1.getStudents().add(student2);
    teacher2.getStudents().add(student1);
    teacher2.getStudents().add(student2);

    // 保存
    session.save(student1);
    session.save(student2);
    session.save(teacher1);
    session.save(teacher2);
```

```
    // ------------------------------------------
    session.getTransaction().commit();
    session.close();
}
------------------------------------------
// 解除关联关系
@Test
public void testRemoveRelation() throws Exception {
    Session session = sessionFactory.openSession();
    session.beginTransaction();
    // ------------------------------------------

    // 如果inverse=false就可以解除，如果为true就不可以解除
    Teacher teacher = (Teacher) session.get(Teacher.class, 3L);
    teacher.getStudents().clear();

    // ------------------------------------------
    session.getTransaction().commit();
    session.close();
}
------------------------------------------
// 删除对象，对关联对象的影响
@Test
public void testDelete() throws Exception {
    Session session = sessionFactory.openSession();
    session.beginTransaction();
    // ------------------------------------------

    // a，如果没有关联的对方：能删除。
    // b，如果有关联的对方且inverse=false，由于可以维护关联关系，他就会先删除关联关系，再删除自己。
    // c，如果有关联的对方且inverse=true，由于不能维护关联关系，所以会直接执行删除自己，就会有异常。
    Teacher teacher = (Teacher) session.get(Teacher.class, 9L);
    session.delete(teacher);

    // ------------------------------------------
    session.getTransaction().commit();
    session.close();
}
```

二、关联关系的分类

```
    双向关联
单向关联：
    单向多对一
    单向一对多
    单向多对多
```

三、Hibernate中对象的4中状态

```
3.1 4中状态：
3.1.1 临时状态 (Transient)
与数据库没有对应，跟Session没有关联。一般是新new出的对象。
======================
3.1.2 持久化状态（Persist）
对象在Session的管理之中，并且有对应的数据库记录。
特点：
    1，有OID()
    2，对对象的修改会同步到数据库。
======================
3.1.3 游离状态（Detached）
数据库中有对应记录，但对象不在Session管理之中。
修改此状态对象时数据库不会有变化。
======================
```

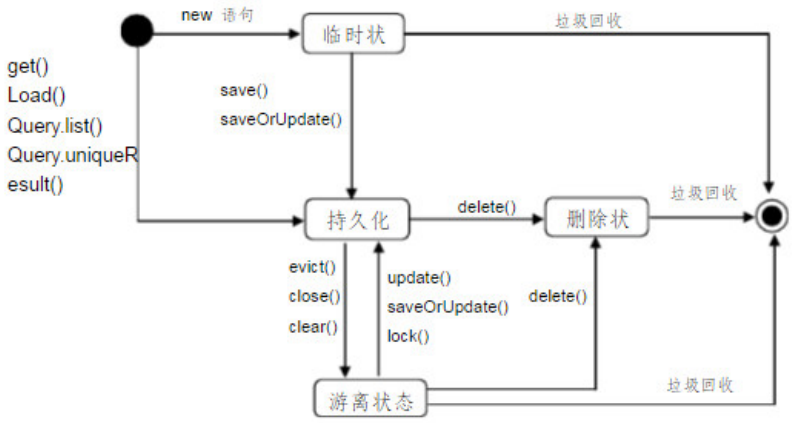
```
3.1.4 删除状态(Removed)
执行了delete()后的对象。

========================
========================
3.2 实现4中状态相互转换的session方法:
3.2.1 操作实体对象的
save()
update()
saveOrUpdate()
delete()

3.2.2操作缓存的
clear()
evict()
flush()

3.2.3查询实体对象的
get()
load()
createQuery()
createCriteria()

========================
========================
3.3 get()和load()方法的对比
        加载方式          返回值       如果数据不存在
-----------------------------------------------------------
get      立即加载     真实对象或null        返回null
load     延迟加载     代理对象              抛异常
```



4种状态的转换

四、session四种对象状态转换方法解释

```
4.1// save()：把临时状态变为持久化状态（交给Sessioin管理）
    // 生成的sql语句：insert into ...
4.2// update()：把游离状态变为持久化状态
// 生成的sql语句：update ...
// 在更新时，对象不存在就报错
4.3// saveOrUpdate()：把临时或游离状态转为持久化状态
// 会生成：insert into 或 update ...
// 在更新时，对象不存在就报错
// 本方法是根据id判断对象是什么状态的：如果id为原始值（对象的是null，原始类型数字是0）就是临时状态，如果不是原始值就是游离状态。
4.4// delete()：把持久化或游离转为删除状态
// 会生成：delete ...
// 如果删除的对象不存在，就会抛异常
4.5// get()：获取数据，是持久化状态
// 会生成：select ... where id=?
// 会马上执行sql语句
// 如果数据不存在，就返回null
```

```
4.6// load()：获取数据，是持久化状态
// 会生成：select ... where id=?
// load()后返回的是一个代理对象，要求类不能是final的，否则不能生成子类代理，就不能使用懒加载功能了。
// 让懒加载失效的方式：一、把实体写成final的；二、在hbm.xml中写<class ... lazy="false">
// 不会马上执行sql语句，而是在第1次使用非id或class属性时执行sql。
// 如果数据不存在，就抛异常：ObjectNotFoundException
        <!--
            lazy属性：默认为true，默认可以懒加载。
         -->
        <class name="User" table="user" lazy="true">
            <id name="id">
                <generator class="native"></generator>
            </id>
            <property name="name"/>
        </class>
```

五、 对于使用hibernate操作大数据出现的问题的解决方案

```
出现的问题:session空间过大，造成内存溢出
解决方案：使用if语句设置固定值清除session中保存的对象
示例：
// 操作大量数据，要防止Session中对象过多而内存溢出
@Test
public void testBatchSave() throws Exception {
    Session session = sessionFactory.openSession();
    session.beginTransaction();
    // ------------------------------------------

    for (int i = 0; i < 30; i++) {
        User user = new User();
        user.setName("测试");
        session.save(user);

        if (i % 10 == 0) {
            session.flush(); // 先刷出
            session.clear(); // 再清空
        }
    }

    // ------------------------------------------
    session.getTransaction().commit();
    session.close();
}
```

个人总结

在使用关联关系时，比如老师与学生的多对多关系，在创建实体重写toString（）方法时，不要相互打印对方的信息，这样在打印对象信息时会造成相互调用，引起内存溢出。

操作