# Hibernate第二天

## 本日重点

* **掌握常用的主键生成策略**
* **掌握持久化对象的三种状态**
* **掌握一级缓存的常用操作（save/update/saveOrUpdate/delete）**
* **掌握一对多关联映射**
* **掌握级联操作**

## 第一节课

### 介绍Hibernate持久化类编写规则（掌握）

* 提供无参的构造器
* 为每一个实体字段提供对应的getter/setter方法
* 每一个实体类都需要提供一个ID（标识属性）
* 实体类字段均使用包装类型
* 实体类不能使用final修饰

**重点**

get与load的区别：

get立即加载，load是懒加载（底层是使用动态代理来实现，生成的代理对象需要从实体类中继承）

### 介绍Hibernate主键生成类型（掌握）

* 自然主键：具有业务意义（省份证号、学号）
* **代理主键：没有业务意义**

### Hibernate主键生成策略

|  |  |
| --- | --- |
| 主键生成器 | 描述 |
| increment | 代理主键。由hibernate维护一个变量，每次生成主键时自动以递增。  问题：如果有多个应用访问一个数据库，由于每个应用维护自己的主键，所以此时主键可能冲突。建议不采用。  优点:可以方便跨平台  缺点:不适合高并发访问 |
| **identity** | 代理主键。由底层数据库生成表识符。条件是数据库支持自动增长数据类型。比如：mysql的自增主键，oracle不支持主键自动生成。  如果数据库支持自增建议采用。  优点:由底层数据库维护，和hibernate无关  缺点:只能对支持自动增长的数据库有效，例如mysql |
| **sequence** | 代理主键。Hibernate根据底层数据库序列生成标识符。条件是数据库支持序列。比如oracle的序列。  如果数据库支持序列建议采用。  优点:由底层数据库维护，和hibernate无关  缺点:数据库必须支持sequence方案例如oracle |
| native | 代理主键。根据底层数据库对自动来选择identity、sequence、hilo  由于生成主键策略的控制权由hibernate控制，所以不建议采用。  优点:在项目中如果存在多个数据库时使用  缺点:效率比较低 |
| **uuid** | 代理主键。Hibernate采用128bit位的UUID算法来生成标识符。该算法能够在网络环境中生成唯一的字符串标识符。  此策略可以保证生成主键的唯一性，并且提供了最好的数据库插入性能和数据库平台的无关性。建议采用。  优点:与数据库无关，方便数据库移植，效率高，不访问数据库就可以直接生成主键值，并且它能保证唯一性。  缺点:uuid长度大(32位),占用空间比较大，对应数据库中类型 char varchar |
| assigned | 自然主键。由java程序负责生成标识符。  不建议采用。  尽量在操作中避免手动对主键操作 |

## 第二节课

### 介绍持久化对象的三种状态（掌握）

* 持久态：session中有，有OID
* 瞬时态（临时态）：没有OID、session中不存在
* 脱管态（游离态）：有OID、session中不存在

### 演示Hibernate持久化对象的三种状态

// 此时Customer就是一个瞬时态

Customer customer **=** **new** Customer**();**

// 此时customer是一个持久态

customer **=** session**.**get**(**Customer**.**class**,** 1**);**

// 脱管态

Customer customer **=** **new** Customer**();**

customer**.**setId**(**1**);**

### 持久化对象三种状态切换（掌握）

重点要记住：对象是否存在OID、在Session中是否存在

* 瞬时态

瞬时态 → 持久态：session.save/saveOrUpdate

瞬时态→脱管态：setId

* 持久态

持久态→脱管态：session.close/clear/evict

clear：清空session缓存，这里并不关闭session缓存

evict：从session里清除指定的一个持久化对象

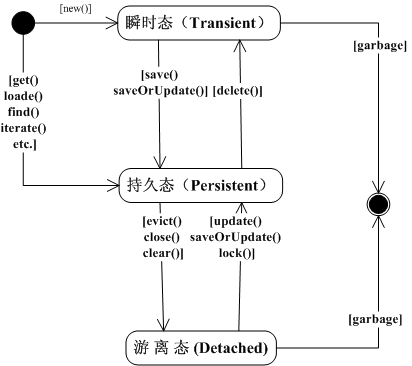
refresh：重新去数据库中查询数据

持久态→瞬时态：session.delete

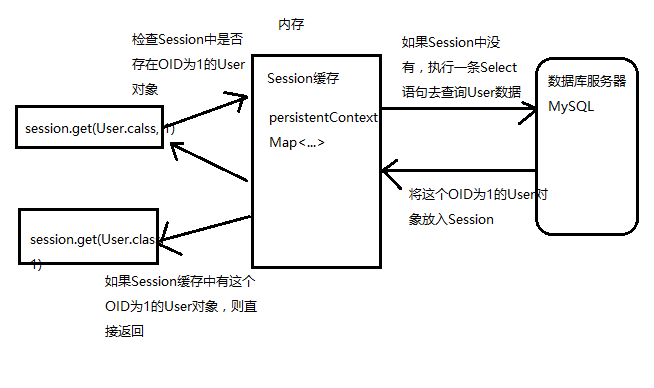
* 脱管态

脱管态→瞬时态：setId(null)

脱管态→持久态：session.update



### 一级缓存介绍



### 一级缓存测试

## 第三节课

### 持久化对象修改数据库

在session缓存中，维护有Map，这个Map的key是持久化的对象(Customer)，Value是对应Custoer的一系列属性（快照）。

注意：当事务提交或者sesssion关闭的时候，会自动将持久化对象与快照**（key与value）**进行比对，如果发现有改变的地方,Hibernate自动执行Update语句。

### 一级缓存常用操作（掌握）

### Session中的update、saveOrUpdate、delete方法

* update

主要用来更新脱管对象。

Session中只允许有一个唯一的OID的对象。update会将脱管态的对象转换为持久态（要将对象放在session缓存），如果session缓存中已经存在同样的OID的对象，这时候就会出现异常。

* saveOrUpdate

当对象是一个瞬时态，save

当对象是一个脱管态，调用的是update

* delete

先删除session缓存的对象，然后执行delete语句删除数据库的记录

/\*\*

\* 测试session.delete()

\* 可直接将游离对象删除

\*/

@Test

**public** **void** test03() {

Session session = HibernateUtils.*getSession*();

session.beginTransaction();

// 这里直接New了一个对象，然后设置ID

User u1 = **new** User();

u1.setId(1);

session.delete(u1);

session.getTransaction().commit();

session.close();

HibernateUtils.*close*();

}

## 第四节课

### 介绍数据对象的一对一关系

* **外键关联（掌握）**

就是在一对一的任意一方，添加一个列，通过这个列就可以查询到另一个表中的数据

* 主键关联

通过两个表的主键来进行关联，两个表的主键是一样的。

**public** **class** **Employee** {

**private** Archive archive;

}

**public** **class** **Archive** {

**private** Employee employee;

}

### 介绍数据对象的一对多关系（掌握）

外键在多的一方维护。在多的一方添加一个列用来关联另一个表。

**public** **class** **Customer** {

**private** Set<Order> orders;

}

**public** **class** **Order** {

*// 缩写是将原因字母删除，将重复的字母删除*

*// message, msg*

**private** Customer cstm;

}

### 数据对象的多对多关系（掌握）

通过中间表来完成多对多关联。中间表中保存了两个张表中的主键。

**public** **class** **Student** {

**private** Set<Teacher> teachers;

}

**public** **class** **Teacher** {

**private** Set<Student> students;

}

### 创建实体类并完成关联映射（掌握）

* Customer

**public** **class** Customer {

**private** Integer id;

**private** String name;

**private** String address;

// 一个Customer有多个订单，这里使用Set来表示一对多的关系

**private** Set<Order> orders = **new** HashSet<Order>();

**public** Set<Order> getOrders() {

**return** orders;

}

**public** **void** setOrders(Set<Order> orders) {

**this**.orders = orders;

}

**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;

}

**public** String getAddress() {

**return** address;

}

**public** **void** setAddress(String address) {

**this**.address = address;

}

}

* Order

**public** **class** Order {

**private** String id;

**private** Date createTime;

// 多个Order订单可以对应一个Customer

**private** Customer customer;

**public** Customer getCustomer() {

**return** customer;

}

**public** **void** setCustomer(Customer customer) {

**this**.customer = customer;

}

**public** String getId() {

**return** id;

}

**public** **void** setId(String id) {

**this**.id = id;

}

**public** Date getCreateTime() {

**return** createTime;

}

**public** **void** setCreateTime(Date createTime) {

**this**.createTime = createTime;

}

}

### 在映射配置文件中配置关联关系（掌握）

找谁，怎么找？

（Customer查找订单Order了，通过Customer的id和t\_order表中的外键cid来查找）

（Order查找客户Customer，也是通过t\_order表的外键cid和Customer的id来查找）

Customer类中有一个Set<Order>，所以在Customer.hbm.xml映射文件中，需要使用set标签来完成映射。

name表示的是要映射到的实体类中的哪个属性。

Customer要去查找到它的订单，是通过订单中外键cid来进行关联。<one-to-many class=””>配置的是查找的Order的完全限定名称。

<set name=*"orders"*>

<key column=*"cid"*></key>

<one-to-many class=*"com.itheima.Order"*/>

</set>

column属性就是指定用哪个列（外键）去查找对应的表。class配置的是要查找的表的完全限定名称

<many-to-one name=*"customer"* class=*"com.itheima.Customer"* column=*"cid"* ></many-to-one>

* Customer.hbm.xml

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

<class name=*"com.itheima.Customer"* table=*"t\_d2h02\_customer"*>

<id name=*"id"*>

<generator class=*"identity"*></generator>

</id>

<property name=*"name"* length=*"20"*></property>

<property name=*"address"* length=*"50"*></property>

<!-- name=”orders”表示Customer中的orders属性，可以理解为orders属性映射为一个集合 -->

<set name=*"orders"*>

<!-- 外键cid,这个外键是存放在t\_order表中 -->

<key column=*"cid"*></key>

<!-- 多方的配置 -->

<one-to-many class=*"com.itheima.Order"*/>

</set>

</class>

</hibernate-mapping>

* Order.hbm.xml

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

<class name=*"com.itheima.Order"* table=*"t\_d2h02\_order"*>

<id name=*"id"* length=*"32"*>

<generator class=*"uuid"*></generator>

</id>

<property name=*"createTime"* type=*"date"*></property>

<!-- 一方的配置，也就是根据cid去查找客户Customer -->

<many-to-one name=*"customer"* class=*"com.itheima.Customer"* column=*"cid"*></many-to-one>

</class>

</hibernate-mapping>

* 在hibernate.cfg.xml中导入映射文件

<mapping resource=*"com/itheima/Customer.hbm.xml"*/>

<mapping resource=*"com/itheima/Order.hbm.xml"*/>

## 第五节课

### 单向关联保存测试

* 保存测试代码

@Test

**public** **void** test02() {

Customer cstm = **new** Customer();

cstm.setName("李三思");

cstm.setAddress("上海");

Order order1 = **new** Order();

order1.setCreateTime(**new** Date());

order1.setCustomer(cstm);

Order order2 = **new** Order();

order2.setCreateTime(**new** Date());

order2.setCustomer(cstm);

cstm.getOrders().add(order1);

cstm.getOrders().add(order2);

Session session = HibernateUtils.*getSession*();

session.beginTransaction();

session.save(order1);

session.save(order2);

session.save(cstm);

session.getTransaction().commit();

session.close();

HibernateUtils.*close*();

}

* Hibernate执行的SQL语句

Hibernate:

insert

into

t\_d2h02\_order

(createTime, cid, id)

values

(?, ?, ?)

Hibernate:

insert

into

t\_d2h02\_order

(createTime, cid, id)

values

(?, ?, ?)

Hibernate:

insert

into

t\_d2h02\_customer

(name, address)

values

(?, ?)

Hibernate:

update

t\_d2h02\_order

set

createTime=?,

cid=?

where

id=?

Hibernate:

update

t\_d2h02\_order

set

createTime=?,

cid=?

where

id=?

Hibernate:

update

t\_d2h02\_order

set

cid=?

where

id=?

Hibernate:

update

t\_d2h02\_order

set

cid=?

where

id=?

### 级联保存操作（掌握）

在保存一个实体对象时，如果这个实体对象关联了另一个实体对象且这个实体对象是一个瞬时态的，配置级联保存后，它也会同时将这个瞬时态的实体对象保存到数据库中。  
如果这个实体对象是持久态的，这时会发出一个update语句来更新外键（cid）。

如果这个实体对象是一个脱管态的，且这个脱管态的实体对象的OID是不存在的，Hibernate会直接抛出异常。

### 解决单向关联保存操作问题（掌握）

* 映射文件配置

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

<class name=*"com.itheima.Order"* table=*"t\_d2h02\_order"*>

<id name=*"id"* length=*"32"*>

<generator class=*"uuid"*></generator>

</id>

<property name=*"createTime"* type=*"date"*></property>

<!-- 级联保存配置 -->

<many-to-one name=*"customer"* class=*"com.itheima.Customer"* column=*"cid"* cascade=*"save-update"*></many-to-one>

</class>

</hibernate-mapping>

* 级联保存代码

@Test

**public** **void** test03() {

Customer cstm = **new** Customer();

cstm.setName("李三思");

cstm.setAddress("上海");

Order order1 = **new** Order();

order1.setCreateTime(**new** Date());

order1.setCustomer(cstm);

Order order2 = **new** Order();

order2.setCreateTime(**new** Date());

order2.setCustomer(cstm);

Session session = HibernateUtils.*getSession*();

session.beginTransaction();

// 建立关联关系，如果没有建立关联关系，及时配置了级联保存，Hibernate也是不会自动保存Customer的

order1.setCustomer(cstm);

order2.setCustomer(cstm);

// 因为配置了级联保存，Hibernate会自动执行一条Insert语句来保存Customer

session.save(order1);

session.save(order2);

session.getTransaction().commit();

session.close();

HibernateUtils.*close*();

}

* Hibernate执行的SQL语句

Hibernate:

insert

into

t\_d2h02\_customer

(name, address)

values

(?, ?)

Hibernate:

insert

into

t\_d2h02\_order

(createTime, cid, id)

values

(?, ?, ?)

Hibernate:

insert

into

t\_d2h02\_order

(createTime, cid, id)

values

(?, ?, ?)

### 双向关联维护分析与问题解决（掌握）

在保存实体的时候，尽量避免建立双方双向关联映射。

Customer cstm = **new** Customer();

Order order1 = **new** Order();

Order order2 = **new** Order();

// 这里已经建立order与custoer的关联

order1.setCustomer(cstm);

order2.setCustomer(cstm);

// 这里的代码其实是多余的，开发中要尽量避免

cstm.getOrders().addOrder(order1);

cstm.getOrders().addOrder(order2);

* inverse

order.setCustomer // 在建立order与customer的关联关系，数据库中是通过外键来建立关联。

如果Inverse为false的话，它会执行一个update语句来维护数据库的外键关联关系。

如果inverse为true的话，它会放弃对外键的维护。也就是说它不再会发起update语句。

* Customer.hbm.xml

配置inverse=true，表示外键交给另一方维护（也就是说保存/更新Customer时，Hiberante不再发出Update语句去更新外键）

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

<class name=*"com.itheima.Customer"* table=*"t\_d2h02\_customer"*>

<id name=*"id"*>

<generator class=*"identity"*></generator>

</id>

<property name=*"name"* length=*"20"*></property>

<property name=*"address"* length=*"50"*></property>

<set name=*"orders"* cascade=*"save-update"* inverse=*"true"*>

<key column=*"cid"*></key>

<one-to-many class=*"com.itheima.Order"*/>

</set>

</class>

</hibernate-mapping>

## 第六节课

### 一对多关联映射（掌握）

### 级联删除（掌握）

删除一个实体对象时，会将它所关联的对象也一并删除。

cascade=”delete”

### 总结级联操作

|  |  |
| --- | --- |
| none | 默认值。不使用级联操作 |
| save-update | save update或save-update完成操作，级联保存临时对象，如果是游离对象，会执行update |
| delete | 级联删除 |
| delete-ophan | 删除与当前对象解除关系的对象 |
| all | save-update delete操作 |
| all-delete-orphan | delete-orphan与all操作 |

## 作业

* 创建一个Customer、Order，建立Customer与Order的一对多关联关系
* 编写代码完成双向关联
* 编写代码完成级联保存
* 编写代码完成级联删除