Hibernate\_day02总结

今日内容

* Hibernate持久化对象的状态
* Hibernate的一级缓存
* Hibernate操作持久化对象的方法
* Hibernate 关联关系映射

## 上次课内容回顾

Hibernate:

\* 第一天环境搭建,开发流程,配置和API

\* 第二天Hibernate持久化对象的状态,一级缓存,关联关系映射.

\* 第三天Hibernate的检索方式 抓取策略 二级缓存.

\* 第四天Hibernate练习:

1. Hibernate框架的概述:

\* 就是一个持久层的ORM框架.

\* **ORM**:对象关系映射.将Java中实体对象与关系型数据库中表建立一种关系.Hibernate中可以通过操作对象的方式,从而操作数据库的表.

\* 常见的持久层框架:

\* Hibernate

\* MyBatis

\* JPA

\* DBUtils

\* Hibernate3.x和Hibernate4.x

1. Hibernate快速入门:

2.1下载Hibernate开发包.

2.2Hibernate开发包目录结构:

\* 文档:

\* 开发包:

\* 操作字节码:

\* 必须的

\* 可选的

\* jpa

\* 项目:

2.3创建一个java项目 导入相应jar包.

2.4创建数据库表:

2.5创建一个实体类:

2.6建立映射:

2.7创建Hibernate核心配置文件:

2.8编写一个测试类:

1. Hibenrate的CRUD的操作:

\* save();

\* update();

\* delete();

\* get()/load();

**\* 区别:**

\* get :采用是立即检索,马上发送SQL查询.返回的是真实对象.检索一个找不到的对象的时候NullPointException

\* load :采用延迟加载技术,当真正使用这个对象的时候,才会发送SQL.返回代理对象.检索一个找不到的对象的时候ObjectNotFoundException

1. Hibernate的常见配置及API:

\* 4.1核心配置:

\* 两种方式;

\* 属性文件的方式:

\* 在src下创建一个hibernate.properties.

\*\*\*\*\* 手动加载映射文件.

\* XML格式的配置:

\* 在src下创建一个hibernate.cfg.xml

\* 必须的配置:

\* 数据库连接信息.

\* Hibernate的方言.

\* 可选配置:

\* 显示SQL

\* 格式化SQL

\* hbm2ddl.auto

\* create/create-drop/update/validate

\* 映射文件:

\* <mapping resource=””/>

\* 4.2映射配置:

\* <class>建立类与表映射

\* name :类的全路径

\* table :表名称

\* <id>

\* name

\* column

\* type

\* length

\* <property>

\* name

\* column

\* type

\* length

1. Hibernate的API

\* Configuration:管理配置信息.

\* 属性文件:

\* 直接创建.

\* XML

\* new Configuration().configure();

\* SessionFactory:

\* 维护二级缓存,线程安全的对象.

\* 抽取工具类.

\* Session:

\* 维护一级缓存,线程不安全的对象.

\* Transaction:

\*\*\*\*\* 事务是默认不自动提交,手动提交事务.

\* Query:

\* Criteria:

1. 持久化类编写:

\* 持久化类:就是一个实体类 + XML映射.

\* 编写原则:

\* 无参数构造:

\* 属性提供set/get方法

\* 提供为一个标识:

\* 尽量使用包装类型:

\* 这个类尽量不要使用final修饰.

\* 自然主键和代理主键:

\* Hibernate提供主键生成策略:

\* increment :

**\* identity :**

**\* sequence :**

**\* uuid :**

**\* native :**

\* assgined :

## Hibernate的持久化类状态:

### Hibernate的持久化类状态

持久化类:就是一个实体类 与 数据库表建立了映射.

\* Hibernate为了方便管理持久化类,将持久化类分成了三种状态.

瞬时态 transient :(临时态)

持久态 persistent :

脱管态 detached :(离线态)

### 三种持久化对象的状态:

#### Transient瞬时态:

特点:**持久化对象没有唯一标识OID.没有纳入Session的管理.**

#### Persistent持久态:

特点:**持久化对象有唯一标识OID.已经纳入到Session的管理.**

**\*\*\*\*\* 结论:持久化持久态对象有自动更新数据库的能力.**

#### Detached脱管态:

特点:**持久化对象有唯一标识OID,没有纳入到Session管理.**

### 区分三种持久化对象的状态:

@Test

// 区分持久化对象的三种状态:

public void demo1(){

// 1.创建Session

Session session = HibernateUtils.openSession();

// 2.开启事务

Transaction tx = session.beginTransaction();

// 向数据库中保存一本图书:

Book book = new Book(); // 瞬时态:没有唯一标识OID,没有与session关联.

book.setName("Hiernate开发");

book.setAuthor("孙XX");

book.setPrice(65d);

session.save(book); // 持久态:有唯一标识OID,与session关联.

// 3.事务提交

tx.commit();

// 4.释放资源

session.close();

book.setName("Struts2开发"); // 脱管态:有唯一的标识,没有与session关联.

}

### 三种状态对象转换:

#### 瞬时态:

获得:

Book book = new Book();

瞬时--->持久

\* save(book);

\* save()/saveOrUpdate();

瞬时--->脱管

\* book.setId(1);

#### 持久态:

获得:

Book book = (Book)session.get(Book.class,1);

\* get()/load()/find()/iterate();

持久--->瞬时:

\* delete(book);

\* 特殊状态:删除态.(被删除的对象,不建议去使用.)

持久--->脱管:

\* session.close();

\* close()/clear()/evict();

#### 脱管态:

获得:

Book book = new Book();

book.setId(1);

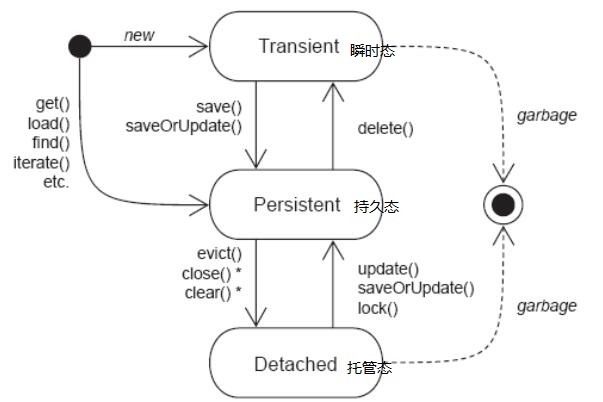
脱管--->持久:

\* session.update();

\* update()/saveOrUpdate()/lock()

脱管--->瞬时:

\* book.setId(null);



### 持久态对象有自动更新数据库的能力;

@Test

// 测试持久态的对象自动更新数据库

public void demo2(){

// 1.创建Session

Session session = HibernateUtils.openSession();

// 2.开启事务

Transaction tx = session.beginTransaction();

// 获得一个持久态的对象.

Book book = (Book) session.get(Book.class, 1);

book.setName("Struts2开发");

// session.update(book);

// 3.提交事务

tx.commit();

// 4.关闭资源

session.close();

}

\*\*\*\*\*\* 自动更新数据库的能力依赖了Hibernate的一级缓存.

## Hibernate的一级缓存:(重要)

### Hibernate的一级缓存:

什么是缓存:

\* 缓存将数据库/硬盘上文件中数据,放入到缓存中(就是内存中一块空间).当再次使用的使用,可以直接从内存中获取.

缓存的好处:

\* 提升程序运行的效率.缓存技术是Hibernate的一个优化的手段.

Hibernate分成两个基本的缓存:

\* 一级缓存:Session级别的缓存.一级缓存与session的生命周期一致.自带的.不可卸载.

\* 二级缓存:SessionFactory级别的缓存.不是自带的.

在 Session 接口的实现中包含一系列的 Java 集合, 这些 Java 集合构成了 Session 缓存. 只要 Session 实例没有结束生命周期, 存放在它缓存中的对象也不会结束生命周期.

### 证明Hibernate的一级缓存的存在:

@Test

// 证明一级缓存的存在

public void demo3(){

// 1.创建Session

Session session = HibernateUtils.openSession();

// 2.开启事务

Transaction tx = session.beginTransaction();

// save方法可以向一级缓存中存放数据的.

/\*Book book = new Book();

book.setName("JQuery开发");

book.setAuthor("张XX");

book.setPrice(45d);

Integer id = (Integer) session.save(book);

Book book2 = (Book) session.get(Book.class, id);

System.out.println(book2);\*/

// 分别用get执行两次查询.

Book book1 = (Book) session.get(Book.class, 1);// 马上发生SQL去查询

System.out.println(book1);

Book book2 = (Book) session.get(Book.class, 1);// 不发生SQL,因为使用一级缓存的数据

System.out.println(book2);

// 3.提交事务

tx.commit();

// 4.关闭资源

session.close();

}

### 深入理解一级缓存中快照区:

@Test

// 深入理解一级缓存结构:快照区:

public void demo4(){

// 1.创建Session

Session session = HibernateUtils.openSession();

// 2.开启事务

Transaction tx = session.beginTransaction();

// 获得一个持久态的对象.

Book book = (Book) session.get(Book.class, 1);

book.setName("Spring3开发");

// 3.提交事务

tx.commit();

// 4.关闭资源

session.close();

}

结论:向一级缓存存入数据的时候,放入一级缓存区和一级缓存快照区,当更新了一级缓存的数据的时候,事务一旦提交,比对一级缓存和快照区,如果数据一致,不更新,如果数据不一致,自动更新数据库.

### Hibernate管理一级缓存:

一级缓存是与session的生命周期相关的.session生命周期结束,一级缓存销毁了.

\* clear()/evict()/flush()/refresh()管理一级缓存.

\* clear() :清空一级缓存中所有的对象.

\* evict(Object obj) :清空一级缓存中某个对象.

\* flush() :刷出缓存.

\* refresh(Object obj):将快照区的数据重新覆盖了一级缓存的数据.

### Hibernate一级缓存的刷出时机:(了解.)

FlushMode:

\* 常量:

\* ALWAYS :每次查询的时候都会刷出.手动调用flush.事务提交的时候.

\* AUTO :**默认值**.有些查询会刷出.手动调用flush.事务提交的时候.

\* COMMIT :在事务提交的时候,手动调用flush的时候.

\* MANUAL :只有在手动调用flush才会刷出.

严格程度:MANUAL > COMMIT > AUTO > ALWAYS

## 操作持久化对象的方法:

#### save():

保存一条记录:将瞬时态对象转成持久态对象.

#### update()

更新一条记录:将脱管态对象转成持久态对象.

在<class>标签上设置select-before-update="true" 在更新之前先去查询

#### saveOrUpdate():

根据对象状态的不同执行不同的save获得update方法.

\* 如果对象是一个瞬时态对象:执行save操作.

\* 如果对象是一个脱管态对象:执行update操作.

\* 设置id不存在,就会报错,可以在<id>上设置一个unsaved-value=”-1”,执行保存的操作.（-1为不存在的ID，可以为其他值）

#### delete():

将持久态对象转成瞬时态.

#### get()/load():

获得一个持久态对象.

## Hibernate关联关系的映射:

### 实体之间的关系:

实体之间有三种关系:

\* 一对多:

\* 一个用户,生成多个订单,每一个订单只能属于一个用户.

\* 建表原则:

\* 在多的一方创建一个字段,作为外键,指向一的一方的主键.

\* 多对多:

\* 一个学生可以选择多门课程,一个课程可以被多个学生选择.

\* 建表原则:

\* 创建第三张表,中间表至少有两个字段,分别作为外键指向多对多双方主键.

\* 一对一:(特殊.最少.)

\* 一个公司只能有一个注册地址,一个注册地址,只能被一个公司使用.(否则将两个表建到一个表.)

\* 建表原则:

\* 唯一外键:

\* 一对一的双方,假设一方是多的关系.需要在多的一方创建一个字段,作为外键.指向一的一方的主键.但是在外键添加一个unique.

\* 主键对应:

\* 一对一的双方,通过主键进行关联.

### Hibernate中一对多的配置:(\*\*\*\*\*)

第一步:

\* 创建两个实体:

\* 客户实体:

public class Customer {

private Integer cid;

private String cname;

// 一个客户有多个订单.

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

public Integer getCid() {

return cid;

}

public void setCid(Integer cid) {

this.cid = cid;

}

public String getCname() {

return cname;

}

public void setCname(String cname) {

this.cname = cname;

}

public Set<Order> getOrders() {

return orders;

}

public void setOrders(Set<Order> orders) {

this.orders = orders;

}

}

\* 订单实体:

public class Order {

private Integer oid;

private String addr;

// 订单属于某一个客户.放置一个客户的对象.

private Customer customer;

public Integer getOid() {

return oid;

}

public void setOid(Integer oid) {

this.oid = oid;

}

public String getAddr() {

return addr;

}

public void setAddr(String addr) {

this.addr = addr;

}

public Customer getCustomer() {

return customer;

}

public void setCustomer(Customer customer) {

this.customer = customer;

}

}

第二步:建立映射:

Customer.hbm.xml

<hibernate-mapping>

<class name="cn.itcast.hibernate3.demo2.Customer" table="customer">

<!-- 配置唯一标识 -->

<id name="cid" column="cid">

<generator class="native"/>

</id>

<!-- 配置普通属性 -->

<property name="cname" column="cname" length="20"/>

<!-- 建立映射 -->

<!-- 配置一个集合 <set>的name Customer对象中的关联对象的属性名称. -->

**<set name="orders">**

**<!-- <key>标签中column:用来描述一对多多的一方的外键的名称. -->**

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

**<!-- 配置一个<one-to-many>标签中class属性:订单的类的全路径 -->**

**<one-to-many class="cn.itcast.hibernate3.demo2.Order"/>**

**</set>**

</class>

</hibernate-mapping>

Order.hbm.xml

<hibernate-mapping>

<class name="cn.itcast.hibernate3.demo2.Order" table="orders">

<!-- 配置唯一标识 -->

<id name="oid" column="oid">

<generator class="native"/>

</id>

<!-- 配置普通属性 -->

<property name="addr" column="addr" length="50"/>

<!-- 配置映射 -->

**<!--**

**<many-to-one>标签**

**name :关联对象的属性的名称.**

**column :表中的外键名称.**

**class :关联对象类的全路径**

**-->**

**<many-to-one name="customer" column="cno" class="cn.itcast.hibernate3.demo2.Customer"/>**

</class>

</hibernate-mapping>

第三步:将映射放到核心配置文件中.

### Hibernate中级联保存的效果:

级联:操作当前对象的时候,关联的对象如何处理.

在映射关系上配置：

cascade=”save-update”

级联方向性:

\* 保存客户的时候,选择级联订单.

\* 保存订单的时候,选择级联客户.

### Hibernate中级联删除的效果:

cascade=”delete”

### Hibernate中的级联取值:

none :不使用级联

save-update :保存或更新的时候级联

delete :删除的时候级联

all :除了孤儿删除以外的所有级联.

delete-orphan :孤儿删除(孤子删除).

\* 仅限于一对多.只有一对多时候,才有父子存在.认为一的一方是父亲,多的一方是子方.

\* 当一个客户与某个订单解除了关系.将外键置为null.订单没有了所属客户,相当于一个孩子没有了父亲.将这种记录就删除了.

all-delete-orphan :包含了孤儿删除的所有的级联.

### 双向维护产生多余的SQL:

配置inverse=”true”:在那一端配置.那么那一端放弃了外键的维护权.

\* 一般情况下,一对多中的一方去放弃，多的一方去维护.

cascade:操作关联对象.

inverse:控制外键的维护.

### Hibernate的多对多的配置:

第一步:创建实体类:

学生的实体:

public class Student {

private Integer sid;

private String sname;

// 一个学生选择多门课程:

private Set<Course> courses = new HashSet<Course>();

public Integer getSid() {

return sid;

}

public void setSid(Integer sid) {

this.sid = sid;

}

public String getSname() {

return sname;

}

public void setSname(String sname) {

this.sname = sname;

}

public Set<Course> getCourses() {

return courses;

}

public void setCourses(Set<Course> courses) {

this.courses = courses;

}

}

课程的实体:

public class Course {

private Integer cid;

private String cname;

// 一个课程被多个学生选择:

private Set<Student> students = new HashSet<Student>();

public Integer getCid() {

return cid;

}

public void setCid(Integer cid) {

this.cid = cid;

}

public String getCname() {

return cname;

}

public void setCname(String cname) {

this.cname = cname;

}

public Set<Student> getStudents() {

return students;

}

public void setStudents(Set<Student> students) {

this.students = students;

}

}

第二步建立映射:

Student.hbm.xml

<hibernate-mapping>

<class name="cn.itcast.hibernate3.demo3.Student" table="student">

<!-- 配置唯一标识 -->

<id name="sid" column="sid">

<generator class="native"/>

</id>

<!-- 配置普通属性 -->

<property name="sname" column="sname" length="20"/>

<!-- 配置关联映射 -->

<!-- <set>标签 name:对应学生中的课程集合的名称 table:中间表名称. -->

<set name="courses" table="stu\_cour">

<!-- <key>中column写 当前类在中间表的外键.-->

<key column="sno"></key>

<!-- <many-to-many>中class:另一方类的全路径. column:另一方在中间表中外键名称-->

<many-to-many class="cn.itcast.hibernate3.demo3.Course" column="cno"/>

</set>

</class>

</hibernate-mapping>

Course.hbm.xml

<hibernate-mapping>

<class name="cn.itcast.hibernate3.demo3.Course" table="course">

<!-- 配置唯一标识 -->

<id name="cid" column="cid">

<generator class="native"/>

</id>

<!-- 配置普通属性 -->

<property name="cname" column="cname" length="20"/>

<!-- 配置与学生关联映射 -->

<!-- <set>中name:对应当前类中的学生的集合的名称 table:中间表的名称 被动的一方放弃外键维护-->

<set name="students" table="stu\_cour" inverse="true">

<!-- <key>中column:当前类在中间表中外键 -->

<key column="cno"></key>

<!-- <many-to-many>中class:另一方的类全路径. column:另一方在中间表中外键名称 -->

<many-to-many class="cn.itcast.hibernate3.demo3.Student" column="sno"/>

</set>

</class>

</hibernate-mapping>

第三步:将映射文件加入到核心配置文件中:

今天的内容总结:

Hibernate持久化对象的状态:

\* 瞬时态:

\* 持久态:

\* 脱管态:

\* 状态之间的转换:

Hibernate一级缓存:

**\* Hibernate的一级缓存的存在.**

**\* get/load/save**

**\* Hibernate的一级缓存的结构:**

**\* 快照区.**

\* Hibernate一级缓存的管理:

\* clear()/evict()/flush()/refresh();

\* 刷出时机(了解)

Hibernate操作持久化对象常用方法:

\* save();

\* update();

\* saveOrUpdate();

\* 瞬时:执行save

\* 脱管:执行update

\* delete();

\* get()/load();直接获得持久对象.

**Hibernate的关联关系:(\*\*\*\*\*)**

**\* 一对多的配置:(手动配置)**

**\* 级联.**

**\* 多对多的配置:(手动配置)**

**\* 级联:**