**DAY01**

**☆抽取HibernateUtils 用来提供Session对象**

创建HibernateUtils

package cn.itcast.utils;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.cfg.Configuration;

/\*\*

\* 创建sessionFactory对象

\*/

public class HibernateUtils {

static Configuration cfg = null;

static SessionFactory sessionFactory = null;

//静态代码块

static {

cfg = new Configuration();

cfg.configure();

//创建sessionFactory

sessionFactory = cfg.buildSessionFactory();

}

//提供得到Session的方法

public static Session getSession() {

Session session = sessionFactory.openSession();

return session;

}

}

使用工具类得到session对象

package cn.itcast.hibernate.test;

import java.util.Date;

import java.util.List;

import org.hibernate.Query;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

import org.junit.Test;

import cn.itcast.entity.User;

import cn.itcast.utils.HibernateUtils;

/\*\*

\* 实现hibernate的crud的操作

\*/

public class TestDemo1 {

//实现添加操作

@Test

public void addTestSession() {

//根据工具类得到session对象

Session session = HibernateUtils.getSession();

//开启事务

Transaction tx = session.beginTransaction();

//写添加逻辑

User user = new User();

user.setUsername("jacktom");

user.setBirthday(new Date());

//调用session里面的方法 save方法

session.save(user);

//提交事务

tx.commit();

//关闭连接

session.close();

//sessionFactory不需要关闭了

// sessionFactory.close();

}

}

## Session接口（CRUD操作）

调用Session里面的方法，实现crud操作

Session是单线程对象，只能有一个操作时候，不能同时多个操作使用，不要把Session变量定义成成员变量，每次使用都创建新对象，相当于JDBC的Connection

Session是应用程序与数据库之间交互操作的一个单线程对象，是 Hibernate 运作的中心；Session是线程不安全的

所有持久化对象必须在session 的管理下才可以进行持久化操作

Session 对象有一个一级缓存，显式执行 flush 之前，所有的持久化操作的数据都缓存在 session 对象处

持久化类与 Session 关联起来后就具有了持久化的能力

### 常用方法

| **方法** | **作用** |
| --- | --- |
| save()/persist() 、update() 、saveOrUpdate() | 增加和修改对象 |
| delete() | 删除对象 |
| get()/load() | 根据主键查询 |
| createQuery()、createSQLQuery() | 数据库操作对象 |
| createCriteria | 条件查询 |

**Transaction接口(事务)**

代表数据库操作的事务对象

Transactiontransaction = session.beginTransaction();

提供事务管理的方法

commit():提交相关联的session实例

rollback():撤销事务操作

wasCommitted():检查事务是否提交

回顾：  
什么是事务

两个操作：提交和回滚

事务特性：原子性、一致性、隔离性、持久性

不考虑隔离性产生问题：脏读、不可重复读、虚读

在jdbc中，使用jdbc代码实现添加操作，默认事务自动提交

在hibernate中，事务默认不是自动提交的，需要手动配置，手动代码提交

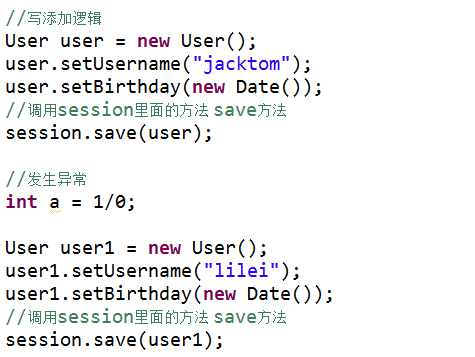
如果没有开启事务，那么每个Session的操作，都相当于一个独立的事务

让hibernate事务提交有两种：  
第一种： 通过配置方式设置hibernate是自动提交

http://upload-images.jianshu.io/upload_images/1540531-260091301a55c883.png?imageMogr2/auto-orient/strip%7CimageView2/2/w/1240

自动提交

**产生问题：**第一个save方法添加数据，但是添加之后出现异常，第二个添加不能完成



示例

第二种：手动代码方式控制事务

package cn.itcast.hibernate.test;

import java.util.Date;

import java.util.List;

import org.hibernate.Query;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

import org.junit.Test;

import cn.itcast.entity.User;

import cn.itcast.utils.HibernateUtils;

/\*\*

\*实现hibernate的crud的操作

\*

\*/

public class TestDemo1 {

//事务操作

@Test

public void testTx() {

//根据工具类得到session对象

Session session = null;

//开启事务

Transaction tx = null;

try {

session = HibernateUtils.getSession();

tx = session.beginTransaction();

//实现两次添加操作

//写添加逻辑

User user = new User();

user.setUsername("jacktom");

user.setBirthday(new Date());

//调用session里面的方法 save方法

session.save(user);

//发生异常

int a = 1/0;

User user1 = new User();

user1.setUsername("lilei");

user1.setBirthday(new Date());

//调用session里面的方法 save方法

session.save(user1);

//提交

tx.commit();

}catch(Exceptione) {

e.printStackTrace();

//回滚

tx.rollback();

}finally {

//关闭session

session.close();

}

}

}

## Query接口(HQL语句查询操作)

Query代表面向对象的一个Hibernate查询操作

session.createQuery 接受一个HQL语句

HQL是Hibernate Query Language缩写，语法很像SQL语法，但是完全面向对象的

### 使用Query对象步骤

获得Hibernate Session对象

编写HQL语句

​ 调用session.createQuery 创建查询对象

​ 如果HQL语句包含参数，则调用Query的setXXX设置参数

​ 调用Query对象的list() 或uniqueResult() 方法执行查询

Query还包含两个方法 用于控制返回结果

​ setFirstResult(intfirstResult) 设置返回结果从第几条开始

​ setMaxResults(intmaxResults) 设置本次返回结果记录条数

实现查询的操作

Query query = session.createQuery(“hql语句”)

query.list(); 返回集合

package cn.itcast.hibernate.test;

import java.util.Date;

import java.util.List;

import org.hibernate.Query;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

import org.junit.Test;

import cn.itcast.entity.User;

import cn.itcast.utils.HibernateUtils;

/\*\*

\* 实现hibernate的crud的操作

\*/

public class TestDemo1 {

//query使用

@Test

public void testQuery() {

Session session = null;

Transaction tx = null;

try {

session = HibernateUtils.getSession();

tx = session.beginTransaction();

//查询表中所有的记录

//hql语句 User是实体类的名称

Query query = session.createQuery("from User");

List<User> list = query.list();

for (User user : list) {

System.out.println(user);

}

tx.commit();

}catch(Exception e) {

tx.rollback();

}finally {

session.close();

}

}

}

# Hibernate实现crud操作（重点）

## 功能具体介绍

### （1）添加操作save

调用Session里的save（实体类对象），不需要设置对象的id值

User user = new User();

user.setUsername("jackTom");

user.setBirthday(new Date());

session.save(user);//调用Session里的save方法

### （2）根据主键查询操作

调用Session里面的get方法

get方法里面有两个参数：

| **参数** | **值** |
| --- | --- |
| 参数一 | 返回实体类的Class |
| 参数二 | id值 |

User user = (User)session.get(User.class, 1);//调用Session里面的get方法

### （3）修改操作update

调用session里面的方法update方法  
update方法里面传递参数有一个：  
修改之后的值，封装到实体类对象，这个对象中必须有id值

做修改操作时候实现的步骤： 首先 根据id把原始信息查询出来 其次，设置要修改的值 最后，调用update方法实现

User user = new User();

user.setId(2);

user.setUsername("东方不败");

session.update(user);

http://upload-images.jianshu.io/upload_images/1540531-31fc51c7351ec659.png?imageMogr2/auto-orient/strip%7CimageView2/2/w/1240

img

做修改操作时候实现的步骤：  
首先 根据id把原始信息查询出来  
其次，设置要修改的值  
最后，调用update方法实现

//修改：根据id修改

//修改id=1的username，修改为其他值

//根据id查询出来对象

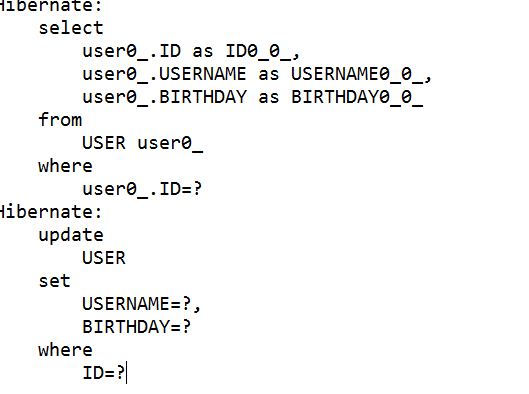
User user = (User)session.get(User.class, 1);

//设置对象中，要修改的值

user.setUsername("大腹便便");

//调用update方法实现

session.update(user);



img

### （4）删除操作delete

调用session里面的方法 delete方法

Delete方法中传递一个参数： 实体类对象，对象中至少要有一个id值

实现方式有两种  
第一种 先查询出来对象，再调用delete方法删除

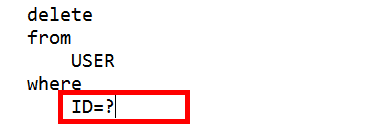
//删除id是4的记录

//查询id是4的记录

User user = (User)session.get(User.class, 4);

//调用delete方法

session.delete(user)



img

### （5）查询操作(3种)hql sql qbc

查询表中所有记录

第一种 使用hql查询

//创建query对象

Query query = session.createQuery("from User");

//调用query里面list方法返回list集合

List<User> list = query.list();

for(User user : list){

System.out.println(user);

}

第二种 使用本地sql查询（会用）

//创建对象

SQLQuery query = session.createSQLQuery("select \* from user");

//调用query里面的方法

//返回list集合，每部分不是对象形式，而是数据形式

List<Object[]> list = query.list();

for(Object[] objects : list) {

System.out.println(Arrays.toString(objects));

}

## crud代码演示:

package cn.itcast.hibernate.test;

import java.util.Arrays;

import java.util.List;

import org.hibernate.Query;

import org.hibernate.SQLQuery;

import org.hibernate.Session;

import org.hibernate.Transaction;

import org.junit.Test;

import cn.itcast.entity.User;

import cn.itcast.utils.HibernateUtils;

/\*\*

\* 实现hibernate的crud的操作

\*/

public class TestDemo2 {

//根据主键查询

@Test

public void testGet() {

Session session = null;

Transaction tx = null;

try {

session = HibernateUtils.getSession();

tx = session.beginTransaction();

//调用session里面get方法

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

System.out.println(user);

tx.commit();

}catch(Exception e) {

tx.rollback();

}finally {

session.close();

}

}

//本地sql查询

@Test

public void testSQL() {

Session session = null;

Transaction tx = null;

try {

session = HibernateUtils.getSession();

tx = session.beginTransaction();

//创建对象

SQLQuery query = session.createSQLQuery("select \* from user");

//调用query里面的方法

//返回list里面，每部分不是对象形式，而是数组形式

List<Object[]> list = query.list();

for (Object[] objects : list) {

System.out.println(Arrays.toString(objects));

}

tx.commit();

}catch(Exception e) {

tx.rollback();

}finally {

session.close();

}

}

//hql语句查询 Query

@Test

public void testHQL() {

Session session = null;

Transaction tx = null;

try {

session = HibernateUtils.getSession();

tx = session.beginTransaction();

//创建query对象

Query query = session.createQuery("from User");

//调用query里面list方法返回list集合

List<User> list = query.list();

for (User user : list) {

System.out.println(user);

}

tx.commit();

}catch(Exception e) {

tx.rollback();

}finally {

session.close();

}

}

//第二种直接删除操作(不建议使用)

@Test

public void testDeleteDemo2() {

Session session = null;

Transaction tx = null;

try {

session = HibernateUtils.getSession();

tx = session.beginTransaction();

//删除id是3的记录

//创建对象

User user = new User();

//设置id值

user.setId(2);

//调用delete方法删除

session.delete(user);

tx.commit();

}catch(Exception e) {

tx.rollback();

}finally {

session.close();

}

}

//第一种删除操作(先查询在删除方法)建议使用

@Test

public void testDeleteDemo1() {

Session session = null;

Transaction tx = null;

try {

session = HibernateUtils.getSession();

tx = session.beginTransaction();

//删除id是4的记录

//查询id是4的对象

User user = (User) session.get(User.class, 4);

//调用delete方法

session.delete(user);

tx.commit();

}catch(Exception e) {

tx.rollback();

}finally {

session.close();

}

}

//修改操作(先查询在修改)

@Test

public void testUpdateSuccess() {

Session session = null;

Transaction tx = null;

try {

session = HibernateUtils.getSession();

tx = session.beginTransaction();

//修改：根据id修改

// 修改id=1的username，修改为 其他值

//根据id查询出来对象

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

//设置对象中，要修改成的值

user.setUsername("大腹便便");

//调用update方法实现

session.update(user);

tx.commit();

}catch(Exception e) {

tx.rollback();

}finally {

session.close();

}

}

//修改操作(直接修改)

@Test

public void testUpdate() {

Session session = null;

Transaction tx = null;

try {

session = HibernateUtils.getSession();

tx = session.beginTransaction();

//修改：根据id修改

// 修改id=2的username，修改为 东方不败

//创建实体类对象

//操作实现修改，但是问题：

//调用update方法之后，把表里面所有字段都修改

//只是设置id和username值，根据id修改username和birthday

User user = new User();

user.setId(2);

user.setUsername("东方不败");

session.update(user);

tx.commit();

}catch(Exception e) {

tx.rollback();

}finally {

session.close();

}

}

}

# 基本数据类型和包装类使用

回顾：  
Java中基本数据类型 有8个 byte short int long float double char boolean  
对应的包装类：  
​ int---Integer char---Character，其他是首字母大写

​ 在实体类中使用基本数据类型还是使用包装类？  
都是使用包装类。

考试成绩。  
可以使用类型 int 和 包装类 Integer  
一般都使用 包装类

表示 学生得分是 0 分  
int score = 0;  
Integer score = 0;

表示学生是否参加考试  
int score = 0;  
Integer score = null;

# Hibernate的主键oid

什么是oid

在java中，区分不同的对象，根据对象的地址  
在数据库中，区别不同的记录，根据主键id值

现在是实体类和数据库映射关系  
在java中没有主键，在数据库里面没有地址

使用hibernate的OID区别不同的对象，如果OID相同的就是一个对象

自然主键和代理主键  
自然主键：  
Oid和业务相关的属性或者字段  
比如人的身份证号码

代理主键：  
Oid和业务不相关的属性和字段  
比如写id

## 配置Hibernate的oid

<id name="id" column="ID">

<generator class="native"></generator>

</id>

使用id标签

有name属性 值写 实体类属性名称

Column属性值 写数据库表字段

在id标签里面使用 generator，配置oid

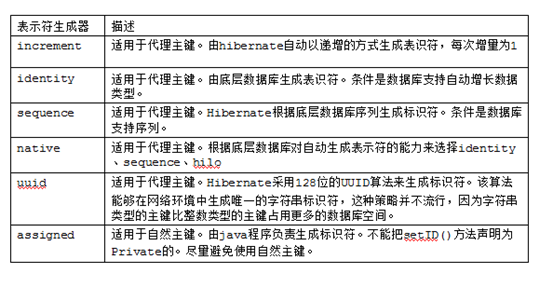
有class属性值 oid主键的生成策略

**Hibernate的oid主键生成策略**

Increment和identity

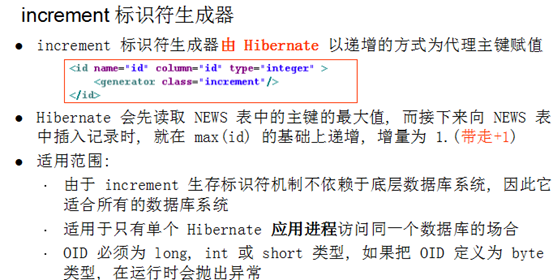
使用increment作为生成策略，生成建表语句中没有AUTO\_INCREMENT自动增长

使用identity作为生成策略，生成的语句中 有AUTO\_INCREMENT自动增长

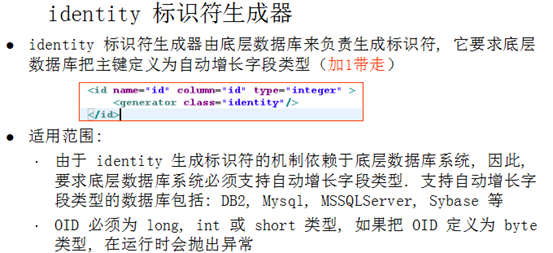


hibernate的主键生成策略

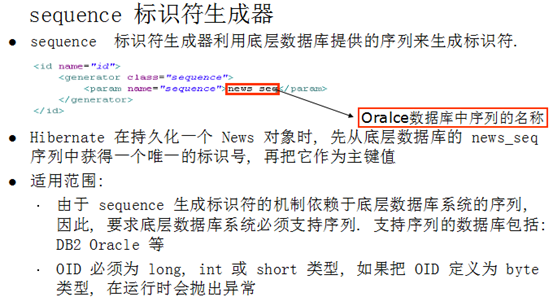
### 详解:



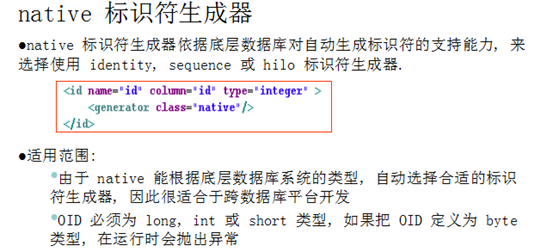
increment标识符生成器



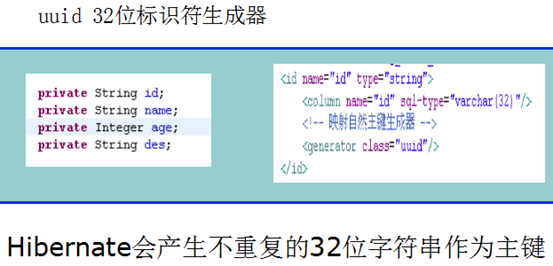
identity标识符生成器



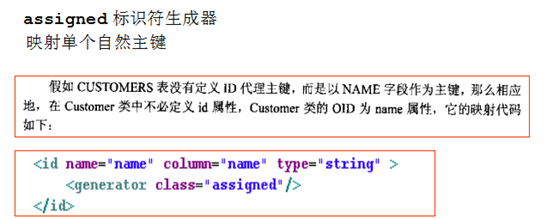
sequence标识符生成器



native标识符生成器



uuid32位标识符生成器



assigned标识符生成器

**DAY02**

# 持久化类的状态

## 持久化类三种状态

持久化类有三种状态，区分不同的状态，使用两种操作区分：

​ 第一个 判断对象是否有oid

​ 第二个 判断对象是否与session相关联

### 1、瞬时态

没有oid，没有与session相关联

User user = new User();

user.setUsername("xxx");

user.setBirthday(new Date());

### 2、持久态

有oid，与session相关联

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

### 3、托管态

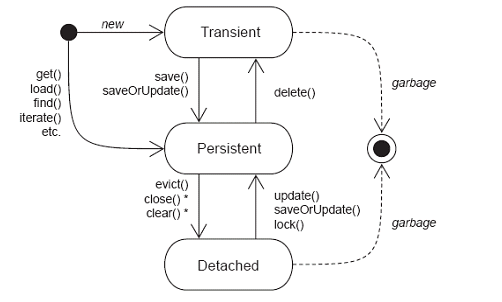
有oid，没有与session相关联

User user = new User();

user.setId(1);

user.setUsername("大黄蜂");

**三种状态之间转换**



三种状态之间转换

### 瞬时态

​ 转换成持久态：调用save方法，saveOrUpdate方法实现

​ 状态成脱管态：设置oid值

### 持久态

​ 转换成瞬时态：调用delete方法实现

​ 转换成脱管态：调用session里面的close方法实现

### 脱管态

​ 转换成瞬时态：设置oid值为空

​ 转换成持久态：调用update和saveOrUpdate方法实现

# Hibernate的一级缓存

## 什么是缓存

把数据不放到文件系统中，放到系统内存中，可以直接从内存中获取数据，提高获取数据的效率

## Hibernate的缓存

### Hibernate的一级缓存

在Hibernate中的一级缓存默认是打开的，一级缓存使用的范围是session范围的。从session创建，到session关闭的过程。如果session关闭了，那么一级缓存也就关闭了。

### Hibernate的二级缓存

在Hibernate中的二级缓存默认不是打开的，手动设置才可以使用。二级缓存使用范围是sessionFactory范围的二级缓存。

## 验证一级缓存的存在

### 第一个操作：首先添加记录，添加完成后，根据oid进行查询

//添加操作

User user = new User();

user.setUsername("岳不群");

user.setBirthday(new Date());

//调用save方法

Serializable id = session.save(user);

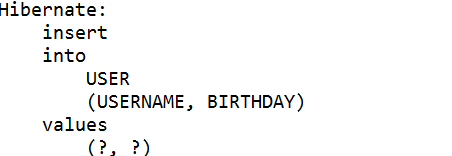
//根据返回的id值查询

User u = (User) session.get(User.class, id);

System.out.println(u);

第一次执行添加操作，发送sql语句，实现添加数据到数据库

第二次根据添加之后返回的id进行查询，没有查询数据库，而是查询一级缓存内容



sql语句

### 第二个操作：第一次查询id是1的记录，第二次再次查询id值是1的记录

//第一次查询id值是1的记录

User user1 = (User) session.get(User.class, 1);

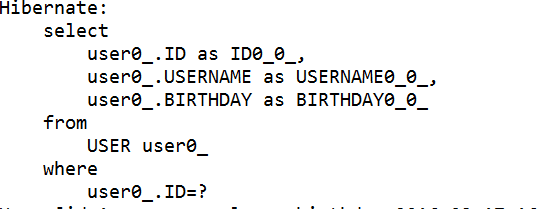
System.out.println(user1);

//第二次查询id值是1的记录

User user2 = (User) session.get(User.class, 1);

System.out.println(user2);

第一次查询数据库获取数据，第二次没有查询数据库，查询一级缓存的内容



sql语句

## 持久态自动更新数据库

//查询id是2的记录

//持久态

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

//设置要修改的值

user.setUsername("哈哈");

//调用update

//session.update(user);

不需要调用update方法实现修改

## 一级缓存的快照区(副本)

//查询id是2的记录

//持久态

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

//设置要修改的值

user.setUsername("哈哈");

//调用update

//session.update(user);

首先根据id查询用户数据，返回user是持久态对象。

首先把返回的持久态user对象放到一级缓存中，另外，把user对象复制一份，再放到一级缓存对应的快照区。

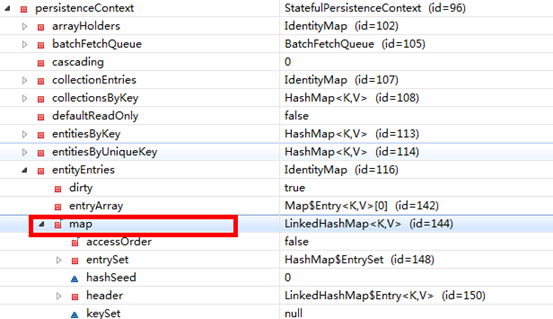
设置了持久态对象里面的值的时候（修改了user里面的值），执行之后

首先，同步更新一级缓存中的内容；其次，但是不会更新对应快照区的内容

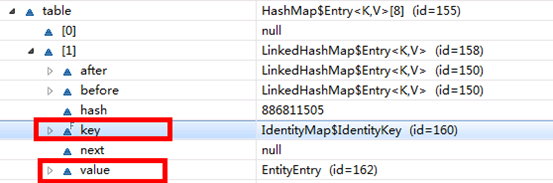
提交了事务之后，实现比较一级缓存和快照区的内容是否相同，如果不相同，把一级缓存中的内容更新到数据库里

一级缓存使用Java的集合存储，使用map集合

Key是一级缓存，Value是快照区



图一



图二

# 操作持久化类的方法

## 常用的方法

| **描述** | **方法** |
| --- | --- |
| 添加 | save() |
| 修改 | update() |
| 删除 | delete() |
| 根据id查询 | get()、load() |
| 添加或修改 | saveOrUpdate() |

## saveOrUpdate方法使用（保存、修改）

这个方法可以实现添加操作，也可以实现修改操作。根据实体类的不同状态实现不同的操作

### 第一个 实现添加操作

当实体类状态是瞬时态时候，调用这个方法实现添加操作

//实现添加操作

//瞬时态

User user = new User();

user.setUsername("东方不败");

user.setBirthday(new Date());

//执行saveOrUpdate方法

session.saveOrUpdate(user);

### 第二个 实现修改操作

持久化类状态是持久态和脱管态时，实现修改操作

//实现修改操作

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

user.setUsername("令狐冲");

//调用方法

session.saveOrUpdate(user);

//脱管态

User user = new User();

user.setId(2);

user.setUsername("任盈盈");

user.setBirthday(new Date());

//调用方法

session.saveOrUpdate(user);

## load方法使用（延迟查询）

load方法和get方法相同，根据id查询数据，返回对象

区别：

​ get方法：立刻查询，执行方法之后，马上查询数据库

​ load方法：延迟查询，执行load方法之后，不会马上查询数据库，得到返回对象里面的值的时候才会去查询数据库

//get方法查询

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

System.out.println(user);

执行get方法之后，立刻发送语句查询数据库

//load方法查询

User user = (User) session.load(User.class, 3);

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

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

执行load方法之后，没有发送sql语句查询数据库，返回实体类对象中只有一个id值，但是得到实体类对象中除了id值之外其他值的时候，才会发送sql语句查询数据库