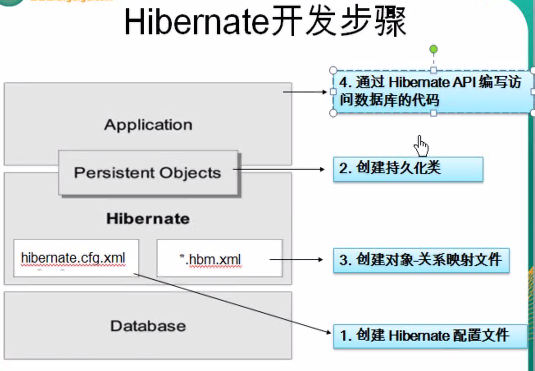
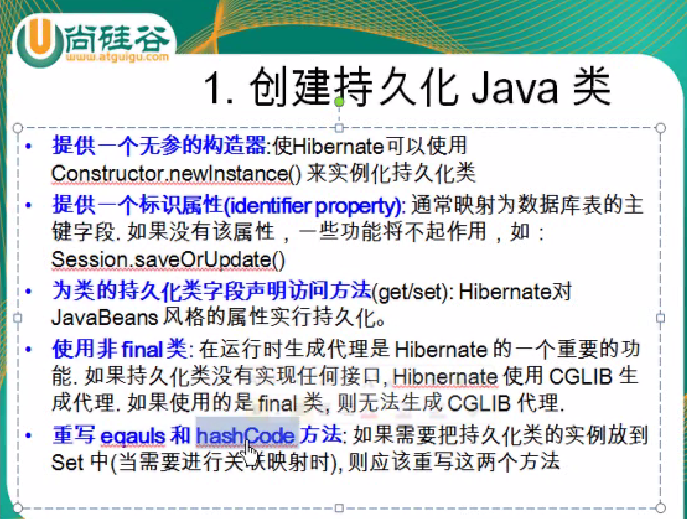
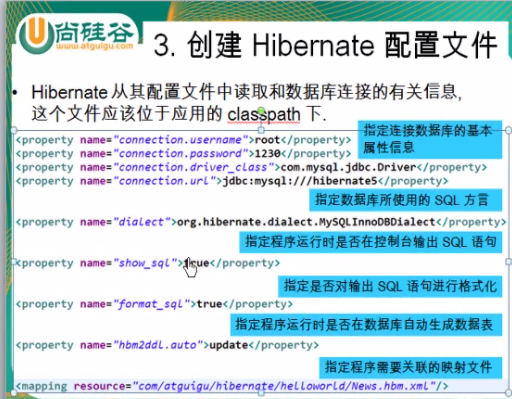
ideo by “atguigu”…

Video 1-4:

1. jar包 = hibernate-release-4.2.4.Final/lib/required + mysqlDriver
2. Hibernate四步曲



1. 注意

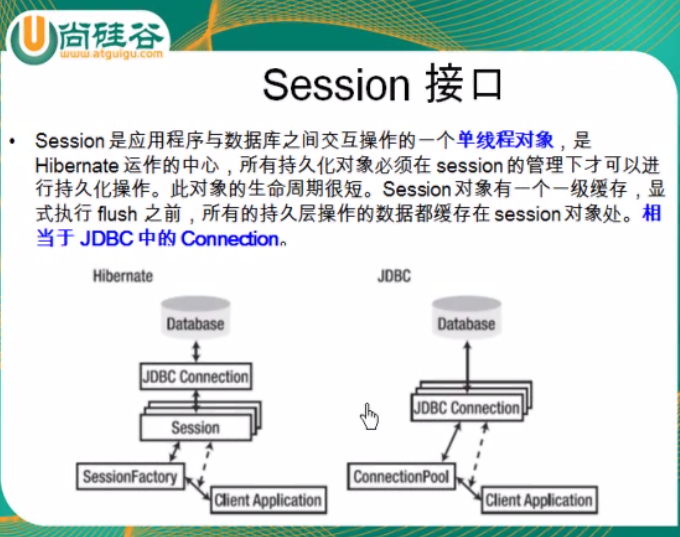


4.

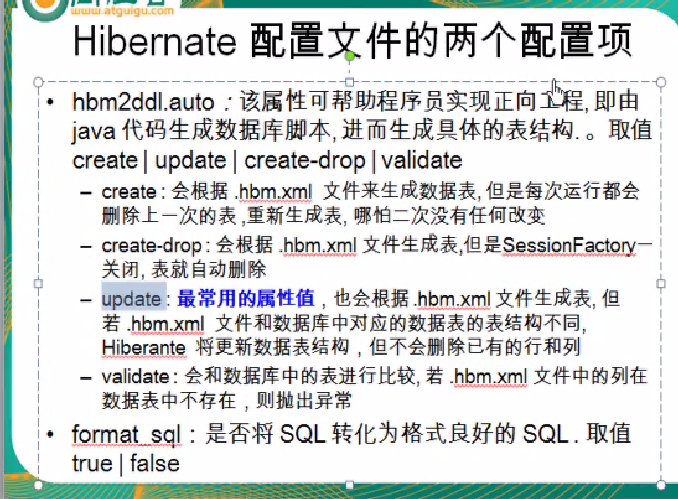
1）Configeration类管理Hibernate配置信息



2）线程安全

3）相当于对Connection封装

4）任何事务都要在Transaction下执行

5）

6）Session接口

7）用junit单元测试类完成之后的工作

**package** com.atguigu.hibernate.entities;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.Transaction;

**import** org.hibernate.cfg.Configuration;

**import** org.hibernate.service.ServiceRegistry;

**import** org.hibernate.service.ServiceRegistryBuilder;

**import** org.junit.After;

**import** org.junit.Before;

**import** org.junit.Test;

**public** **class** HibernateTest {

**private** SessionFactory sessionFactory;

**private** Session session;

**private** Transaction transaction;

@Before

**public** **void** init(){

Configuration configuration = **new** Configuration().configure();

ServiceRegistry serviceRegistry = **new** ServiceRegistryBuilder().

applySettings(configuration.getProperties())

.buildServiceRegistry();

sessionFactory = configuration.buildSessionFactory(serviceRegistry);

session = sessionFactory.openSession();

transaction = session.beginTransaction();

}

@After

**public** **void** destroy() {

transaction.commit();

session.close();

sessionFactory.close();

}

@Test

**public** **void** test() {

}

}

Video 5-6

1. Session的缓存
   1. Flush()：使得数据库中的记录和Session中一致，为了保持一致在Transaction的commit()方法中：先调用session的flush方法，再提交事务

Flush()方法可能会发送SQL语句(如果session里数据和数据库数据不一致才发)，不会提交事务

注意：在未提交事务或者显示调用session.flush()之前，也有可能会进行flush操作。。

1).执行HQL或者QBC查询，会进行flush操作,以得到数据表最新数据

2).若记录的ID是由底层数据库采用自增的方式生成的，则在调用save()方法之后，会立即发送Insert语句。 因为save方法之后，必须报账对象id存在。

@Test

**public** **void** testSessionFlush2(){

News news = **new** News("java", "SUN", **new** Date());

session.save(news);

}

@Test

**public** **void** testSessionFlush(){

News news = (News) session.get(News.**class**, 1);

news.setAuthor("SUN");

session.flush();

System.***out***.println("flush..");

}



b) refresh()方法 ：refresh会强制发送SQL语句

//刷新缓存 51行断点处手动修改mysql数据库记录

//1,若发现两个sysout语句结果一样，则是数据库隔离级别的问题，mysql的隔离级别默认是repeatable read，改成啥去了。。

@Test

//refresh会强制发送SQL语句

**public** **void** testRefresh(){

News news = (News) session.get(News.**class**, 1);

System.***out***.println(news);

session.refresh(news);

System.***out***.println(news);

}

* 1. clear()清理缓存

//清理缓存

@Test

**public** **void** testClear(){

News news = (News) session.get(News.**class**, 1);

session.clear();

News news2 = (News) session.get(News.**class**, 1);

}

Video 6：

1. 持久化对象的状态（脑海里形成一个公司与一个员工的关系）

临时对象：new出来的

持久化对象：save之后，位于Session缓存中，托管对象；持久化状态时，不允许程序随意修改其OID

删除对象：delete 曾经被session监管过，曾经在数据库中有记录

游离对象：save之后，close之后脱管对象，员工“请假”

1. save方法：

将临时对象变为持久化对象，为对象分配ID

在flush缓存时会发送一条insert语句

在save方法之前的id是无效的

1. persist方法：也会执行insert操作

和save的区别：在调用 persist方法之前，若对象已经有ID了，则不会执行insert，会抛出异常。



Video7- 依然是讲session核心方法

1. update方法
2. /\*
3. \* update
4. \* 1.若更新一个持久化对象，不需要显示调用update，调用setXXX即可。
5. \* 2.更新一个游离对象，需要显示敲update方法,
6. \* 可以把游离对象变为持久化对象
7. \* 需要注意的是：
8. \* 1.如果把news.setAuthor("DB4")；注释，还是会发update语句。。。
9. \* 如何能够让update方法不盲目发送update语句呢？
10. \* 在.hbm.xml文件class节点设置 select-before-update = true
11. \* 但通常不需要设置。
12. \* 2.news.setId(1000);行 对于数据表中不存在的对象，但还调用了update方法，会StaleStateException
13. \* 3.当update方法关联一个有游离对戏那个
14. \*如果在session缓存中已经存在OID相同的，会异常，因为在session缓存中不能存在两个IOD相同的对象。
15. \*
16. \*/

@Test

**public** **void** testUpdate(){

News news = (News) session.get(News.**class**, 1);

transaction.commit();

session.close();

session = sessionFactory.openSession();

transaction = session.beginTransaction();

news.setAuthor("SUN");

session.update(news);

}

1. saveOrUpdate()方法

判断对象为临时对象的方法：

OID是否为null（一般情况下）

极端情况下，。。。<id> 设置了unsaved-value属性

/\*

\* 注意：

\* 1.若OID不为空，但数据表中还没有和它对应的记录，会异常

\* 2.了解：若OID=id的unsaved-value 属性值的对象，也认为是游离对象

\*/

@Test

**public** **void** testSaveOrUpdate(){

News news = **new** News("FF", "fffggadsg", **new** Date());

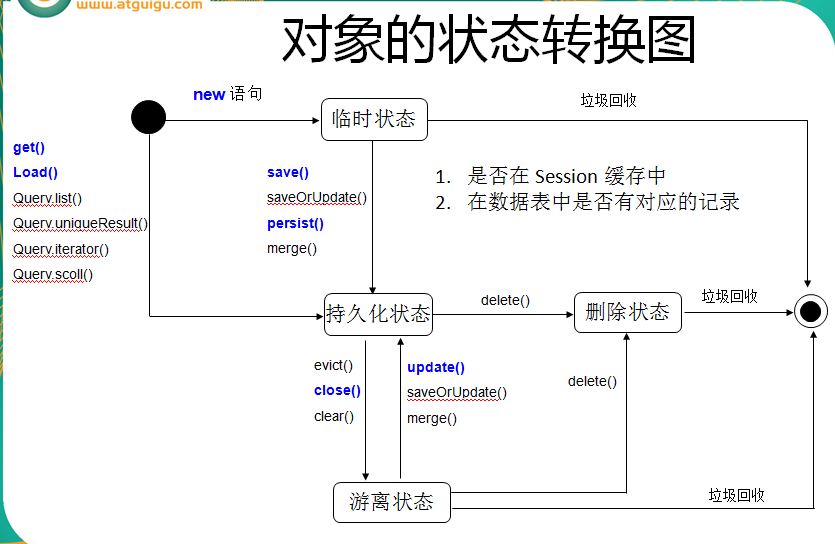
news.setId(1);

session.saveOrUpdate(news);

}

1. delete方法：只要在数据库中有记录和session中OID对应，就删除；

没有的话就异常

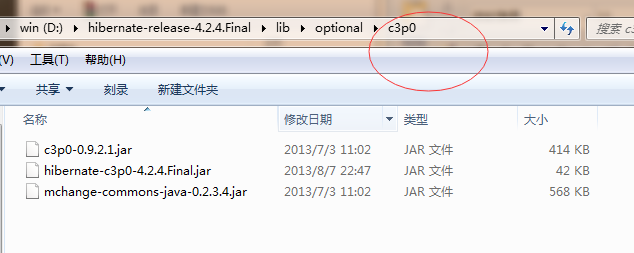
4. evict从缓存中移除对象

Video 8 hibernate.cfg.xml其他配置项

hibernate.cfg.xml：包括数据库连接+hibernate所需的其他两部分

1. 在hibernate中使用c3p0数据源

1).导入jar包



2).加入配置

* C3P0 数据库连接池属性
  + hibernate.c3p0.max\_size: 数据库连接池的最大连接数
  + hibernate.c3p0.min\_size: 数据库连接池的最小连接数
  + hibernate.c3p0.timeout: 数据库连接池中连接对象在多长时间没有使用过后，就应该被销毁
  + hibernate.c3p0.max\_statements: 缓存 Statement 对象的数量
  + hibernate.c3p0.idle\_test\_period: 表示连接池**检测线程**多长时间检测一次池内的所有链接对象是否超时. 连接池本身不会把自己从连接池中移除，而是专门有一个线程按照一定的时间间隔来做这件事，这个线程通过比较连接对象最后一次被使用时间和当前时间的时间差来和 timeout 做对比，进而决定是否销毁这个连接对象。
  + hibernate.c3p0.acquire\_increment: 当数据库连接池中的连接耗尽时, 同一时刻获取多少个数据库连接
  + hibernate.jdbc.fetch\_size：例如一次查询1万条记录，对于Oracle的JDBC驱动来说，是不会 1 次性把1万条取出来的，而只会取出 fetchSize 条数，当结果集遍历完了这些记录以后，再去数据库取 fetchSize 条数据。因此大大节省了无谓的内存消耗。Fetch Size设的越大，读数据库的次数越少，速度越快；Fetch Size越小，读数据库的次数越多，速度越慢。Oracle数据库的JDBC驱动默认的Fetch Size = 10，是一个保守的设定，根据测试，当Fetch Size=50时，性能会提升1倍之多，当 f**etchSize=100**，性能还能继续提升20%，Fetch Size继续增大，性能提升的就不显著了。并不是所有的数据库都支持Fetch Size特性，例如MySQL就不支持
* hibernate.jdbc.batch\_size：**设定对数据库进行批量删除，批量更新和批量插入的时候的批次大小，类似于设置缓冲区大小的意思。batchSize 越大，批量操作时向数据库发送sql的次数越少，速度就越快。**
* **测试结果是当Batch Size=0的时候，使用Hibernate对Oracle数据库删除1万条记录需要25秒，Batch Size = 50的时候，删除仅仅需要5秒！Oracle数据库 batchSize=30 的时候比较合适。**

**Video9: .hbm.xml文件：hibernate mapping == hbm**

1. **建议1个hbm对应一个类**
2. **Hibernate建议使用代理主键：不具备业务意义的字段，通常为整型**

****

1. **各种generator特征：**

**increment：会发送select语句查询max(ID)，容易出现并发问题**

**identity：需要底层数据库自己设置主键自增，mysql可以采取这种，oracle不行；**

**OID需要long/int/short,不能为byte.**

**sequence：需要底层数据库设置，DB2,ORACLE可以，mysql不行,因为要底层数据库支持序列。。OID需要long/int/short,不能为byte.**

**hilo: high/low算法，不依赖任何数据库，通用. OID需要long/int/short,不能为byte.**

**Native:根据当前数据库支持能力选择identity/sequence/hilo的一种，跨平台~！**

1. **type属性：有时候(日期+时间都需要时)需要指定完整的hibernate映射类型，但一般写全类名也可以。**

****

**5: formula：设置一个 SQL 表达式, Hibernate 将根据它来计算出派生属性的值.**

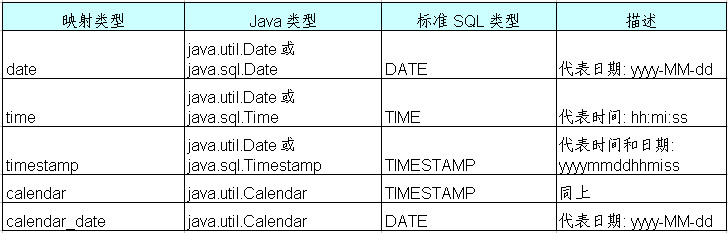
* + **派生属性: 并不是持久化类的所有属性都直接和表的字段匹配, 持久化类的有些属性的值必须在运行时通过计算才能得出来, 这种属性称为派生属性**

**6：映射java的时间日期类型**

* **基础知识：**

**在 Java 中, 代表时间和日期的类型包括: java.util.Date 和 java.util.Calendar. 此外, 在 JDBC API 中还提供了 3 个扩展了 java.util.Date 类的子类: java.sql.Date, java.sql.Time 和 java.sql.Timestamp, 这三个类分别和标准 SQL 类型中的 DATE, TIME 和 TIMESTAMP 类型对应**

**在标准 SQL 中, DATE 类型表示日期, TIME 类型表示时间, TIMESTAMP 类型表示时间戳（时间+日期）, 同时包含日期和时间信息.**

****

**那具体如何进行映射呢？**

**Java.util.Date是父接口，所以可以映射一切。。于是，在设置持久化类的类型时，设置为java.util.Date….**

**具体：通过property的type属性。。。**

<property name=*"date"* type=*"timestamp"* >

<column name=*"DATE"* />

</property>

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

<column name=*"DATE"* />

</property>

<property name=*"date"* type=*"time"* >

<column name=*"DATE"* />

</property>

其中，timestamp、date、time既不是Java类型也不是标准SQL类型，而是Hibernate类型，像一个扁担

**Video10：重点：映射组成关系 Worker 与Pay**

1. **blob大文件处理（这一集中说的方法在开发中基本不用，忽略）**
2. **映射组成关系**

Video11.多对一映射

1. 单向n to 1

推荐先插入1的一段，后插入n的一段。。。

@Test

**public** **void** testMany2One(){

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

customer.setCustomerName("BB");

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

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

order1.setOrderName("Order-3");

order2.setOrderName("Order-4");

//设定关联关系

order1.setCustomer(customer);

order2.setCustomer(customer);

//先插入customer，后插入order，3条insert

// session.save(customer);

// session.save(order1);

// session.save(order2);

//先插入customer，后插入order"

//3条insert语句，2条update

session.save(order1);

session.save(order2);

session.save(customer);

}

注意1：

延迟加载：

@Test

**public** **void** testMany2OneGet(){

Order order = (Order) session.get(Order.**class**, 1);

System.***out***.println(order.getOrderName());

Customer customer = order.getCustomer();

System.***out***.println(customer.getCustomerName());

}

以下是上述程序产生的SQL语句分析:当需要真正用到cutomer的时候才发送select语句。。。。

Hibernate:

select

order0\_.ORDER\_ID as ORDER\_ID1\_2\_0\_,

order0\_.ORDER\_NAME as ORDER\_NA2\_2\_0\_,

order0\_.CUSTOMER\_ID as CUSTOMER3\_2\_0\_

from

ORDERS order0\_

where

order0\_.ORDER\_ID=?

Order-1

Hibernate:

select

customer0\_.CUSTOMER\_ID as CUSTOMER1\_0\_0\_,

customer0\_.CUSTOMER\_NAME as CUSTOMER2\_0\_0\_

from

CUSTOMERS customer0\_

where

customer0\_.CUSTOMER\_ID=?

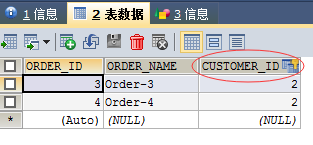
AA

注意2：

获取Order对象时候，其关联的对象是一个代理对象（lazy。。）

注意3：delete对象时候，如果customer还有被引用，则删不了，解除掉特定的引用就可以删除特定的customer。。

（如图，清理了Order表的CUSTOMER\_ID为1的之后就可以删除CUSTOMER表里CUSTOMER\_ID为1的那一行记录了。）





@Test

**public** **void** testDelete(){

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

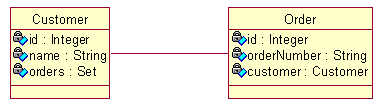
session.delete(customer);

}

Video12.双向1-n

1. 双向 1-n 与 双向 n-1 是完全相同的两种情形

* 双向 1-n 需要在 1 的一端可以访问 n 的一端, 反之依然.
* 域模型:从 Order 到 Customer 的多对一双向关联需要在Order 类中定义一个 Customer 属性, 而在 Customer 类中需定义存放 Order 对象的集合属性



2. 在没有设置 inverse=true 的情况下，父子两边都维护父子

关系

* 在 1-n 关系中，将 n 方设为主控方将有助于性能改善(如果要国家元首记住全国人民的名字，不是太可能，但要让全国人民知道国家元首，就容易的多)
* 在 1-N 关系中，若将 1 方设为主控方

**1).会额外多出 update 语句**。

2).插入数据时无法同时插入外键列，因而无法为外键列添加非空约束