**Hibernate Study Notes**

**@author Count Monte Cristo**

**@create D20150610**

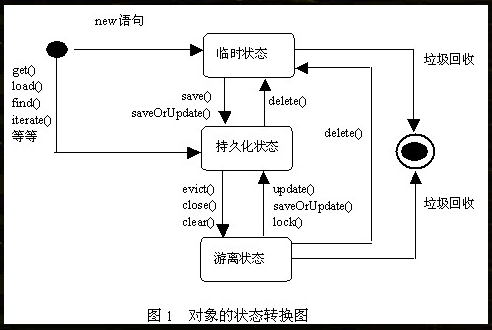
**@version ~~D20150812~~ D20151024**

**@description 记载了Hibernate中重要的知识，开发时可同时参照该文档和Hibernate官方文档**

**@reference 《尚学堂》马士兵J2EE Hibernate3.3.2框架技术视频**

**@motto VICTORY LOVES PREPARATION**

**@criterion 凯旋基诺**

1. **含义**
   1. [Hibernate](http://baike.baidu.com/view/7291.htm" \t "_blank)是一个开放源代码的[对象](http://baike.baidu.com/view/2387.htm" \t "_blank)关系映射[框架](http://baike.baidu.com/view/66971.htm" \t "_blank)，它对JDBC进行了非常轻量级的对象封装，使得Java[程序员](http://baike.baidu.com/view/39175.htm" \t "_blank)可以随心所欲的使用对象[编程](http://baike.baidu.com/view/3281.htm" \t "_blank)思维来操纵[数据库](http://baike.baidu.com/view/1088.htm" \t "_blank)。 Hibernate可以应用在任何使用JDBC的场合，既可以在Java的客户端[程序](http://baike.baidu.com/view/17674.htm" \t "_blank)使用，也可以在Servlet/JSP的Web应用中使用，最具革命意义的是，Hibernate可以在应用EJB的[J2EE](http://baike.baidu.com/view/1507.htm" \t "_blank)架构中取代CMP，完成[数据持久化](http://baike.baidu.com/view/4549557.htm" \t "_blank)的重任。
2. **核心接口和类**
   1. Hibernate的核心类和接口一共有6个，分别为:Session、SessionFactory、Transaction、Query、Criteria和Configuration。这6个核心和类接口在任何开发中都会用到。通过这些[接口](http://baike.baidu.com/view/159864.htm" \t "_blank)，不仅可以对持久化[对象](http://baike.baidu.com/view/2387.htm" \t "_blank)进行存取，还能够进行事务控制。
3. **Hibernate介绍**
   1. @百度百科
   2. @official Website<http://hibernate.org/orm/>
4. **Hibernate的HelloWorld Demo**
   1. @Pros\_Backups@Hibernate\_Demo\_HelloWorld
5. **Hibernate入门**
   1. @CSDN <http://blog.csdn.net/doodoofish/article/details/43207/>
6. **Maven Pom文件中的Hibernate配置**
   1. @CSDN <http://blog.csdn.net/lili72/article/details/8564634>
7. **此处暂时省略部分知识的记录，有待以后补齐**
   1. @create 2015-07-14
8. **课程内容**
   1. **HelloWorld**
      1. xml
      2. annotation
   2. **Hibernate原理模拟**
      1. 什么是O/R Mapping以及为什么要有O/R Mapping
   3. **常见的O/R框架**
   4. **Hibernate基础配置（重点）**
   5. **Hibernate核心开发接口介绍（重点）**
   6. **对象的三种状态**
   7. **ID生成策略**
   8. **关系映射（重点）**
   9. **Hibernate查询（HQL）（重点）**
   10. **在Struts基础上继续完善BBS2009**
   11. **性能优化**
   12. **补充话题**
9. **Hibernate入门：实体对象的三种状态以及CRUD方法**
   1. **@reference <http://blog.csdn.net/xiazdong/article/details/7574732>**
   2. **Hibernate实体的三种状态**
      1. Transient(瞬态)：刚刚被创建出来的状态
      2. Persistent(持久态、持久化)：被Session管理时的状态，能够和数据库同步(同步：就是实体内容修改了会导致数据库中数据的变化)
      3. Detached(托管态、托管)：过去被Session管理，但是现在Session已关闭，虽然存在与数据库对应的记录，但是不能同步
   3. **三种状态之间的转化**
      1. 转化图
      2. ****三种状态及转化详解
         1. **transient状态：**

当我们new一个对象的时候，就是transient状态，最显著的特点是在数据库中没有与之对应的记录。也没有纳入到Session 的管理，随时可以被垃圾回收处理掉。

* + - 1. **persistent状态：**

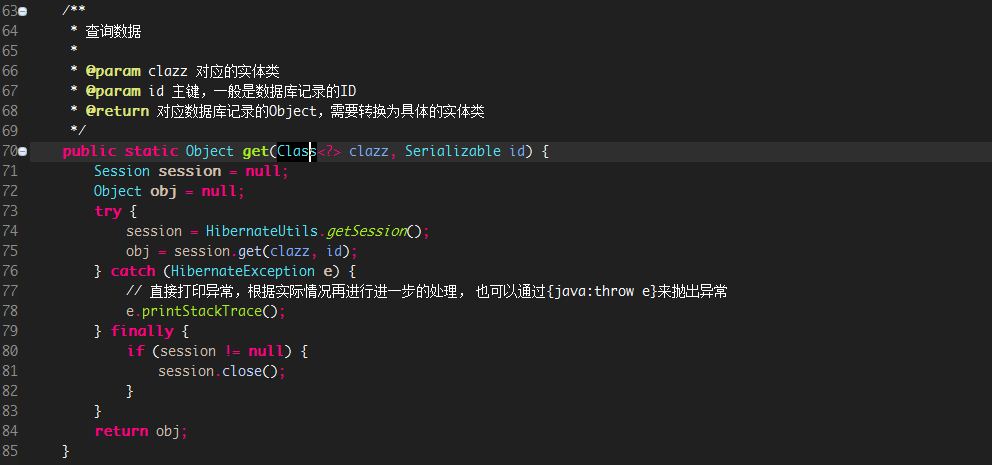
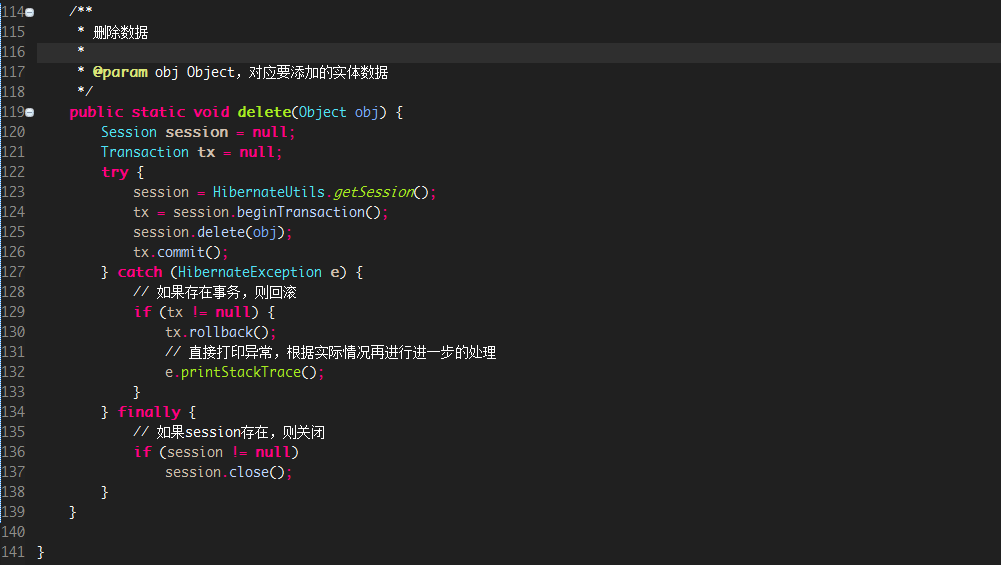
当我们对transient 对象进行save(),saveOrUpdate()之后对象进入persistent状态，在数据库中有与之对应的记录。纳入了session的管理，session和一级缓存绑定着，session级的缓存通常也叫一级缓存。当save()的时候会放到缓存里一份，可以理解为session里有个map，对象放到了map里面，有对象引用他了，垃圾回收机制是回收不了它的。调用delete()，进入

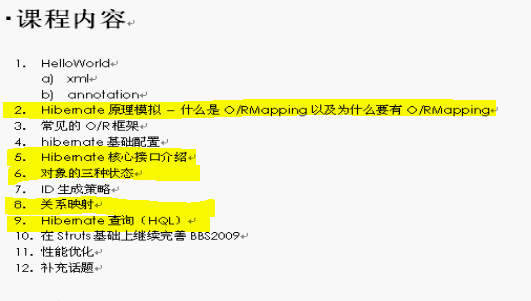
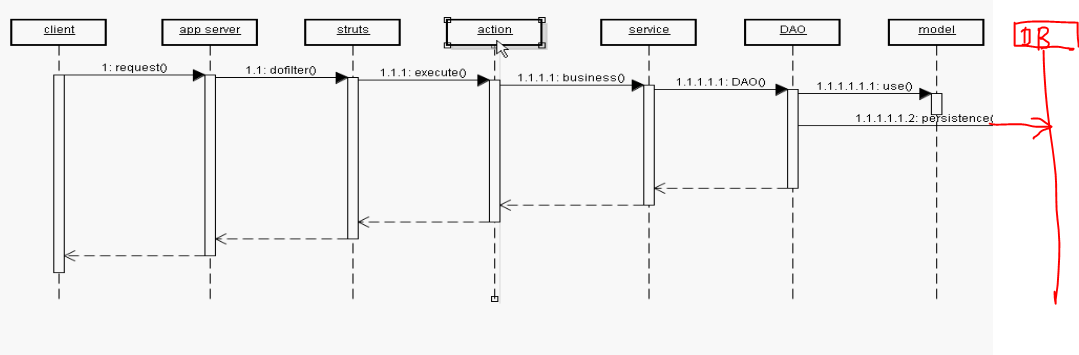
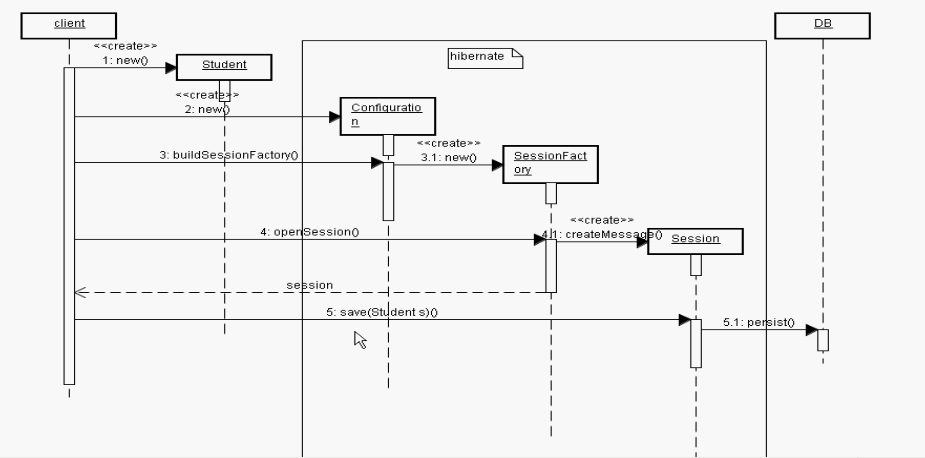
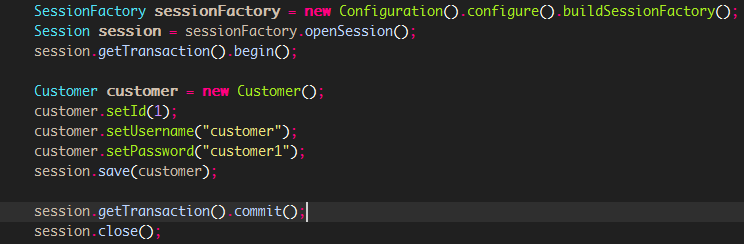
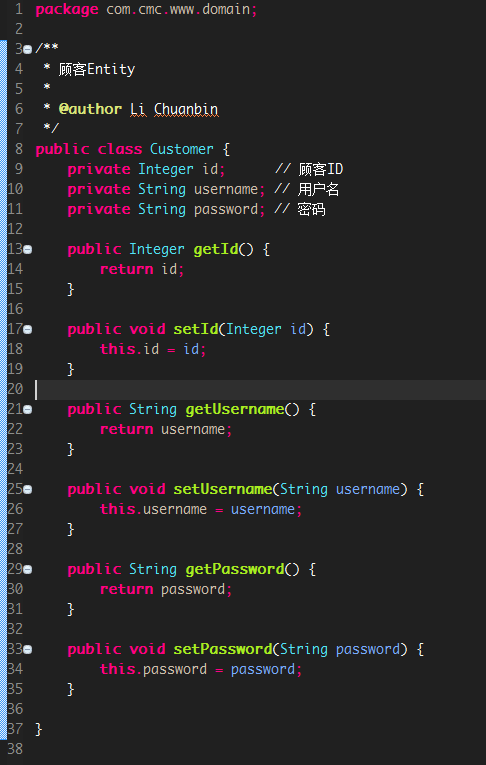
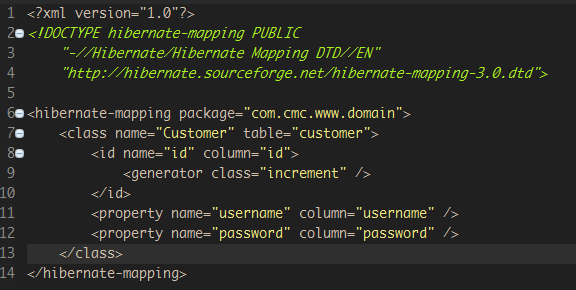
* + - 1. **detached状态：**

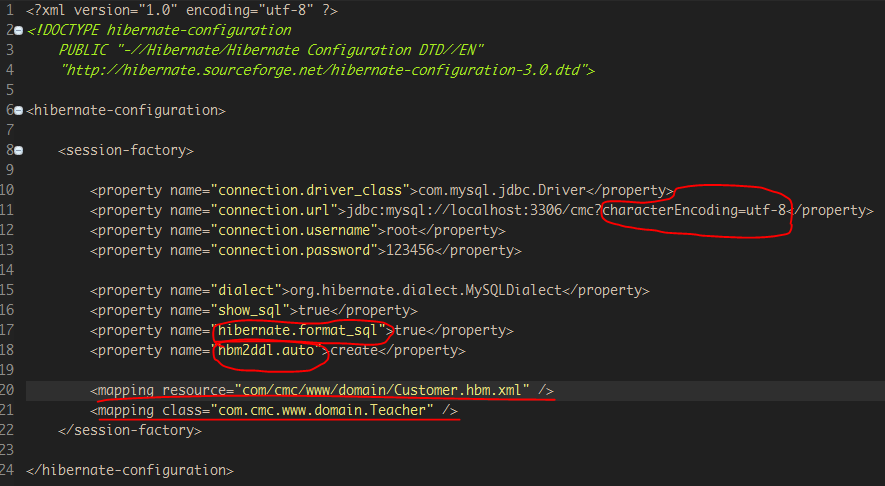
当persistent对象经过 evict()，close()，clear()方法处理，进入游离态也就是detached状态。detached对象之是将缓存里的记录清理了，数据库里的记录并没清理，没有纳入session管理，随时可以被垃圾回收掉。没有对象引用他～。因为在数据库里有引用记录没清理，所以可以调用update()，saveOrUpdate()，lock()等方法，调方法之后又回到transient状态。

* 1. **Session入门常用方法**
     1. Query query = session.createQuery(hql)
        1. 利用hql查询语句查询；
     2. Criteria critera = session.createCriteria(Class clazz);
     3. Transaction tx = session.beginTransaction();
        1. 开始事务
        2. tx.commit()提交事务；
     4. session.close();
        1. 关闭Session，此后被session管理的持久化对象变为脱管状态；
     5. session.save(Object obj);
        1. 添加
     6. session.update(Object obj);
        1. 更新
     7. session.delete(Object obj);
        1. 删除
     8. Object obj = session.get(Class clazz,Serialiazble id);
        1. 根据主键查找记录并返回
     9. Object obj = session.load(Class clazz,Serializable id);
        1. 和get方法效果一样，但是是懒加载，即在不使用他之前他不会返回对象；
  2. **HibernateUtils实例**

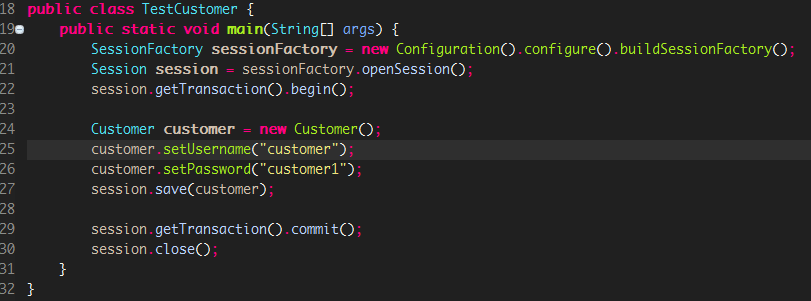
****



1. **《尚学堂》马士兵J2EE框架技术之Hibernate视频开始Notes**
2. **Annotation注意点**
   1. **关于“如果一个Entity没有使用@Table进行标注表的名称，则会默认使用类的名称，类开头小写”**
      1. 单独使用类的时候可能会这样生成，但是设计到关联的时候有时就不会默认表格的首字母小写了
      2. Best Practice：使用@Table(name = “表名称”)指定Entity具体的表格名称
3. **课程内容 pending !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!**
4. **Struts2（集成Hibernate和Spring）执行过程示意图**
5. **Hibernate执行过程示意图**
6. **hibernate.cfg.xml中的hbm2ddl.auto属性详解**
   1. **@reference** <http://www.cnblogs.com/feilong3540717/archive/2011/12/19/2293038.html>
   2. **含义**
      1. 主要用于：自动创建|更新|验证数据库表结构。如果不是此方面的需求建议set value="none"。
   3. **值及其作用**
      1. validate
         1. 每次加载hibernate时，验证创建数据库表结构，只会和数据库中的表进行比较，不会创建新表，但是会插入新值。
      2. update
         1. 最常用的属性，第一次加载hibernate时根据model类会自动建立起表的结构（前提是先建立好数据库），以后加载hibernate时根据 model类自动更新表结构，即使表结构改变了但表中的行仍然存在不会删除以前的行。要注意的是当部署到服务器后，表结构是不会被马上建立起来的，是要等应用第一次运行起来后才会。
      3. create
         1. 每次加载hibernate时都会删除上一次的生成的表，然后根据你的model类再重新来生成新表，哪怕两次没有任何改变也要这样执行，这就是导致数据库表数据丢失的一个重要原因。
      4. create-drop
         1. 每次加载hibernate时根据model类生成表，但是sessionFactory关闭的时候,表就自动删除。
7. **Hibernate中基本的方法**
   1. **图示**
   2. **说明：要特别注意类的上级包，因为同一个名字的类有可能在多个包中**
8. **Hibernate中的（多数）Annotation符合JPA标准，其实其他Annotation大多数都是符合JPA标准的，JPA努力实现Annotation的统一。**
9. **XML版本Hibernate\_Demo\_HelloWorld**
   1. Entity：普通Bean Entity+entity.hbm.xml实体配置文件
      1. Bean Entity(no annotation)
         1. 视图
      2. entity.hbm.xml
         1. 视图
         2. 说明：
            1. DOCTYPE定义文档类型，因为较熟，不赘述
            2. <id>标签用于指定主键，主键的generator有多个类的实现，详见Id Generator总结
            3. 其他的看一下也就知道什么意思，可参考Hibernate Reference Documetation
   2. hibernate.cfg.xml
      1. 视图

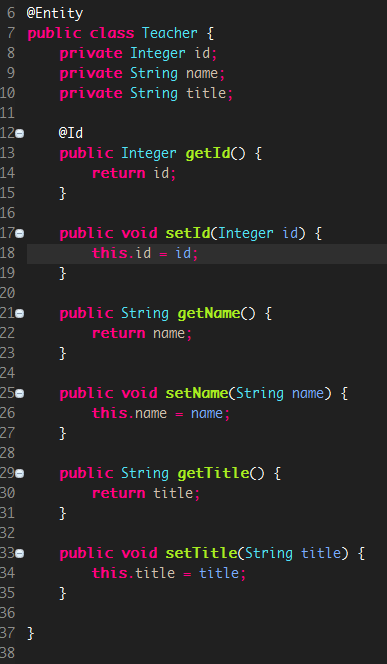


* + 1. 说明
       1. 此配置文件中配置了数据库、辅助编程的以及Bean Entity的信息
       2. characterEncoding=utf-8指定了存入数据库的时候文本的编码格式，可以解决中文乱码问题
       3. 其他辅助编程属性有利于编程的高效，可以参考Hibernate Reference Documentation
       4. Bean Entity的必须要在此文件中进行配置，resource属性对应xml形式的Bean Entity，class对应annotation形式的Bean Entity
  1. Controller
     1. 视图



* + 1. 说明
       1. 主要是加载配置文件找到Session
       2. 若hibernate.cfg.xml文件不在classes目录下面，可以通过configure(String value)进行配置

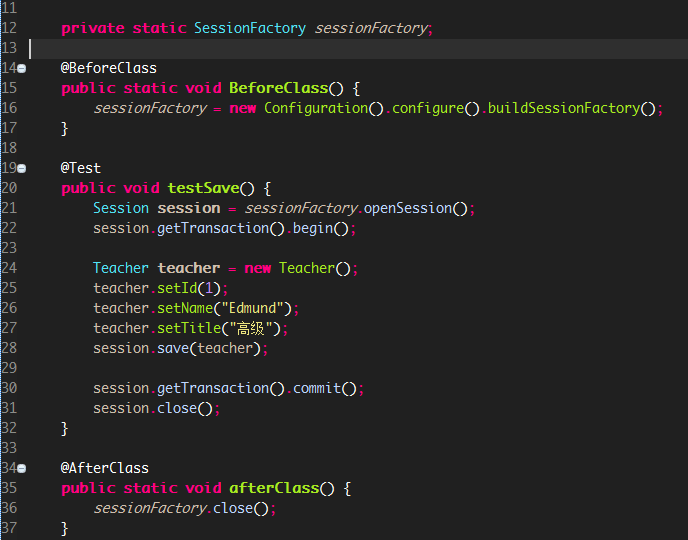
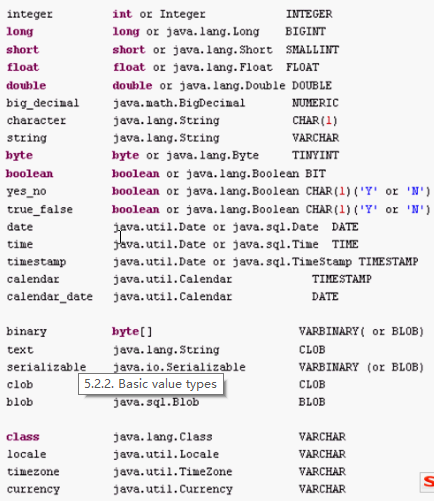
1. **Annotation版本Hibernate\_Demo\_HelloWorld**
   1. **Bean Entity**
      1. **视图**

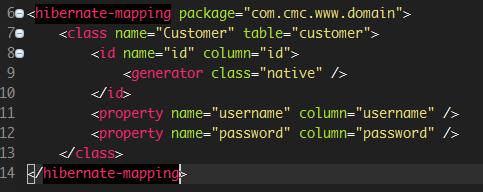
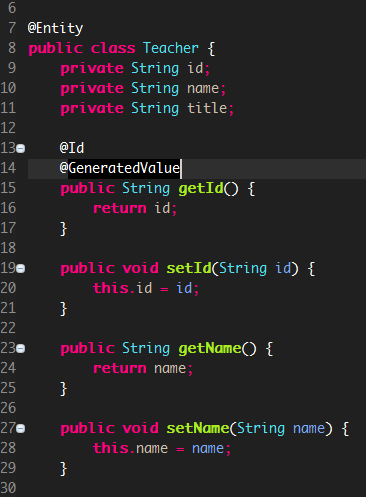
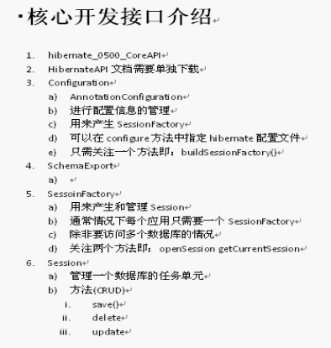
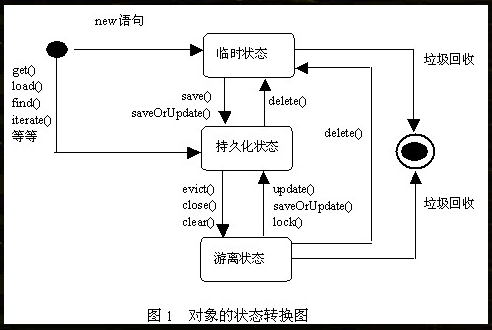
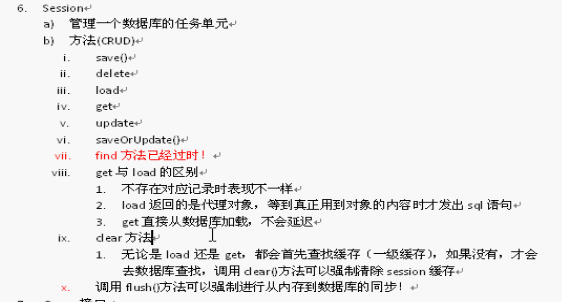
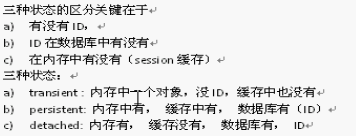
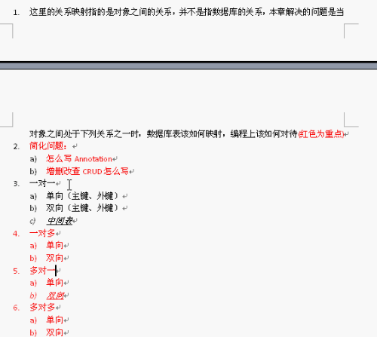
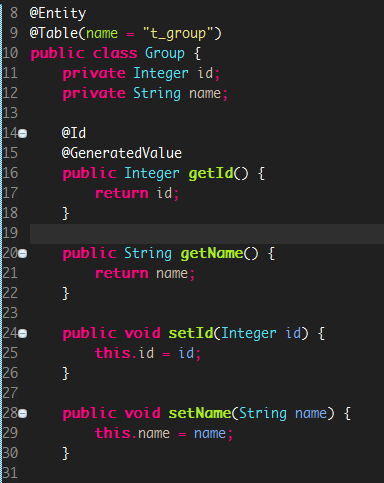
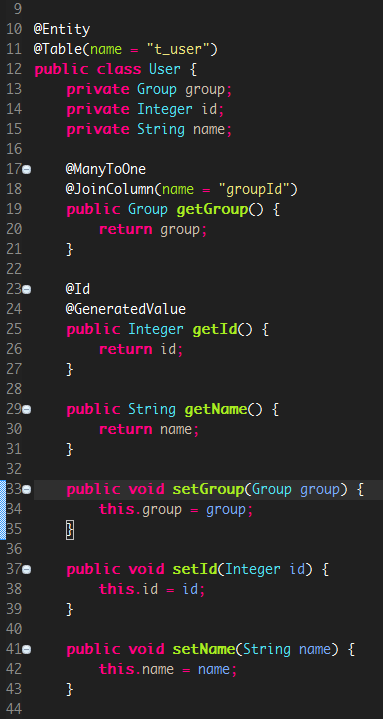
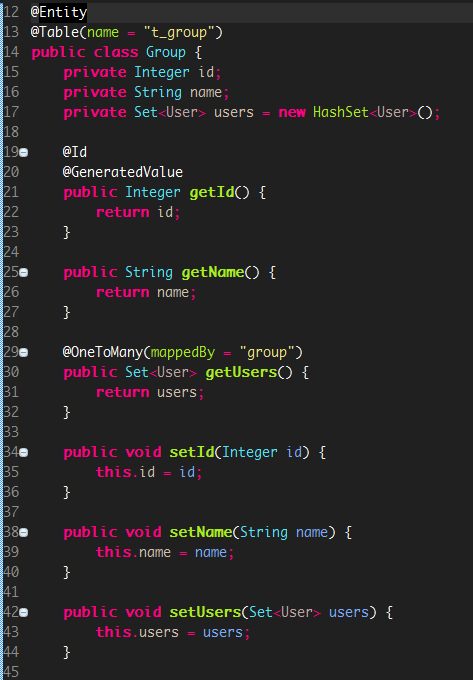
****

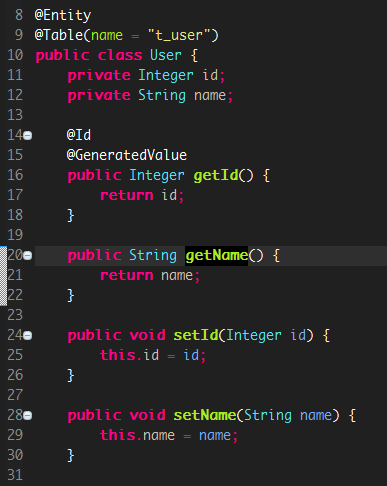
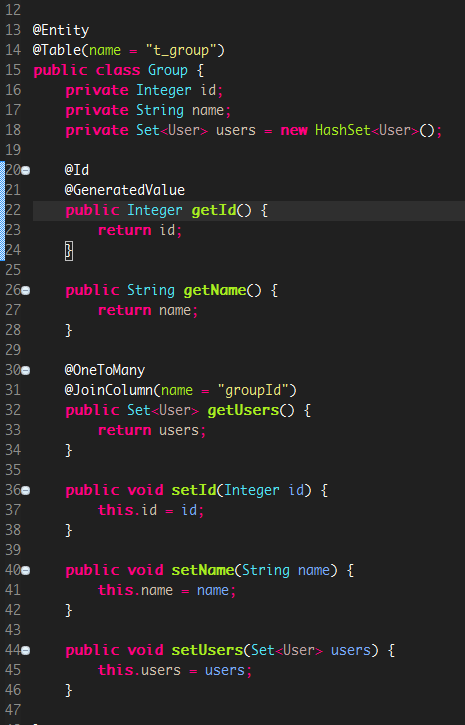
* + 1. **说明**
       1. @Entity：指定该类是Entity类，没有指定映射的表时候默认为类名对应的字段，首字母小写。例如，该类table为teacher
       2. @Id：指定数据库表的主键
       3. Bean Entity与数据库表映射的信息全部加到get方法上，不加在属性上是为了维持属性的私有化
       4. 没有指定具体的表格字段就会默认使用属性的名称（首字母小写，同时因为属性首字母也是小写，所有数据库表格中字段和属性一样）
  1. **hibernate.cfg.xml**
     1. 同XMl形式的hibernate.cfg.xml
  2. **Controller**
     1. 同XML形式的Controller

1. **What and Why O/R Mapping**
   1. O/R Mapping：Object and Relational Mapping(ORM)，对象（Bean Entity）关系（关系数据库）映射
   2. JDBC操作数据库很繁琐，通过ORM建立关联简化操作
   3. SQL编写不是面向对象的
   4. O/R Mapping跨越数据库平台，方言的使用
2. **常见ORM框架**
   1. Hibernate
   2. Toplink
   3. JDO
   4. iBatis
   5. MyBatis
   6. xorm(golang的ORM框架)
   7. JPA（标准）（想统一天下，JPA定义了标准：接口，其他框架实现了其标准，并扩展了自己的功能）
3. **Hibernate基础配置**
   1. **建表和建类顺序问题：实际开发中先建表后建类（设计到优化和习惯）**
   2. **搭建Log4J日志系统**
      1. @referenceHibernate\_Demo\_BasicConfiguration\_logging
      2. 日志框架
         1. slf接口(具体实现日志)
            1. slf4j-nodep
            2. log4j
            3. jdk-logging-api
            4. apache-commons-logging
      3. log4j中输入信息的级别
         1. debug
         2. info
         3. warn
         4. error
         5. fatal

注意：级别越低信息输入越详细，一般在开发的时候使用debug, 开发完成后使用error

* 1. **指定Bean Entity对应的的表，@reference javax.persistence.Table**
  2. **搭建JUnit环境**
     1. 在pom文件中导入JUnit的dependency
     2. 实例
        1. 视图
        2. 说明
           1. @BeforeClass，在类中方法执行之前执行
           2. @Test，标识方法是测试方法
           3. @AfterClass，在类中方法执行之后执行
  3. **若数据库字段和Entity中的属性相同，相当于在get方法上加上@Basic注解，不同的时候要在get方法上面写@Column(name = “数据库字段名称”)进行指定**
  4. **不想Bean Entity的属性映射到数据库的字段的时候可以加上@Transient注释**
  5. **Bean Entity中java.util.Date类型的数据默认保存到数据库的为日期+时分秒，可以使用@Temporary注释进行保存到数据库中字段的日期格式设置**
  6. **Hibernate中Bean Entity属性到数据库表格字段类型应映射**
  7. **Annotation映射位置：应该放在get方法上，如果放在field中会破坏field的私有化（通过Java反射机制）**

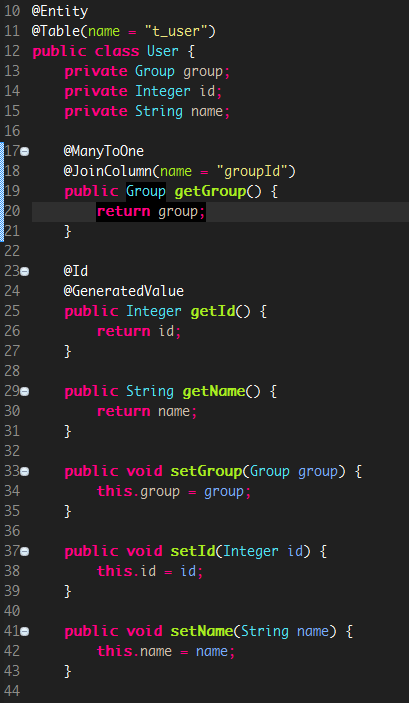
1. **ID的生成策略**
   1. **常用的生成策略**
2. increment：int类型，插入一条记录自增一，集群的情况不能使用
3. identity：int类型，数据库（部分）自动递增字段
4. sequence：int类型，数据库（部分）产生ID值，oracle常用
5. uuid：String类型，universal unique identification，全宇宙唯一的身份标识
6. native：int类型，数据库自动选择identity、sequence或者hilo算法生成ID
   1. **实例**
      1. Xml：<generator class=””>标签进行指定
      2. Annotation：@GeneratedValue注解，默认相当于native策略，其他（table、联合主键等）可以参照API和马士兵的Hibernate视频
7. **核心开发接口**
   1. **视图**
   2. **getCurrentSession() VS openSession()**
      1. 前者获取当前“上下文”中的session，如果没有创建一个新的，后者无论“上下文”中有没有新的session都会创建一个新的session
   3. **实例**
      1. @reference Hibernate\_Demo\_CoreAPI
   4. **核心开发接口图示及说明**
   5. **对象的三种状态**
      1. **视图**
   6. **三种状态的区分**
      1. 视图
      2. 说明
         1. 缓存是内存和数据库和数据库之间的一个映射集合，缓存会自动和数据库进行数据的同步
8. **关系映射（共7种，双向必须设置mappedBy，且其值为Many一方的属性）**
   1. **图示**
   2. **One2One**
      1. 单向主键关联（不重要）
         1. Annotation：@PrimaryKeyJoinColumn
         2. XML：直接参考Hibernate Reference Documentation和马士兵J2EE相关视频
      2. 联合主键关联
         1. 参考Hibernate Reference Documetation和马士兵J2EE相关视频
   3. **One2Many、Many2One**
      1. 设计思想：在“多”的一方加上外键（都是在多的一方加上外键的），不要使数据发生冗余
      2. 单向关联
         1. ManyToOne
            1. 实例（XML版本请参照Hibernate Reference Documentation和马士兵J2EE相关视频）
         2. OneToMany
            1. 实例

Annotation

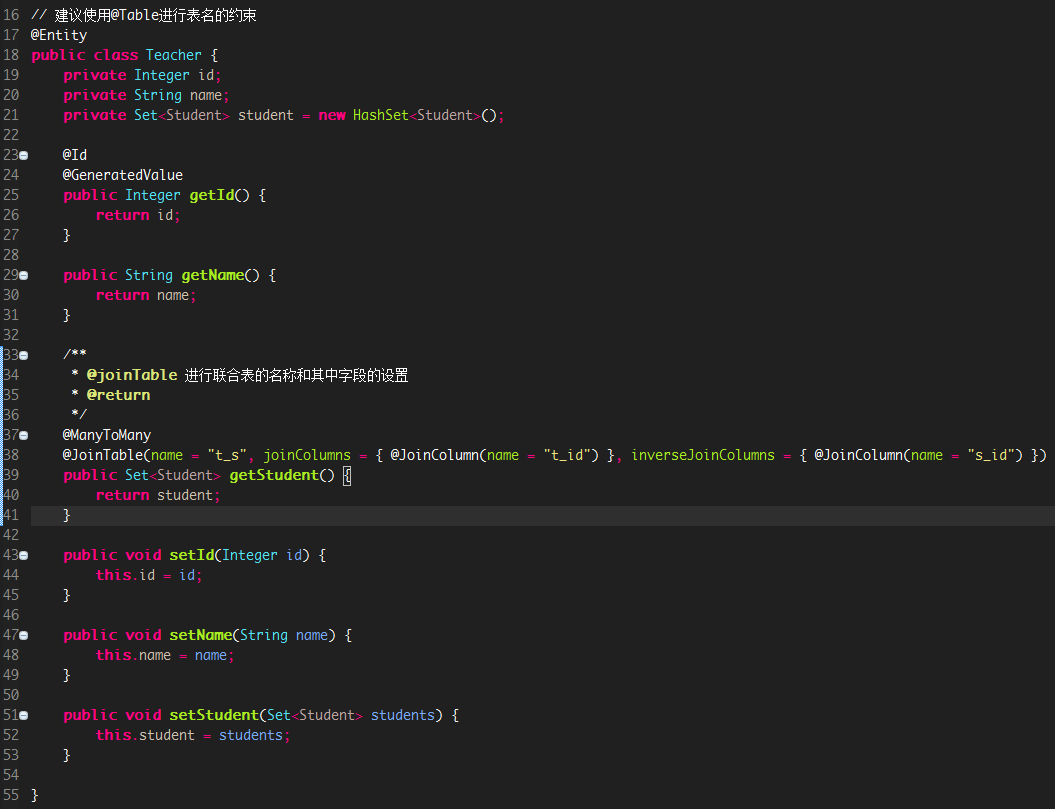
XML：直接参考Hibernate Reference Documentation和马士兵J2EE相关视频

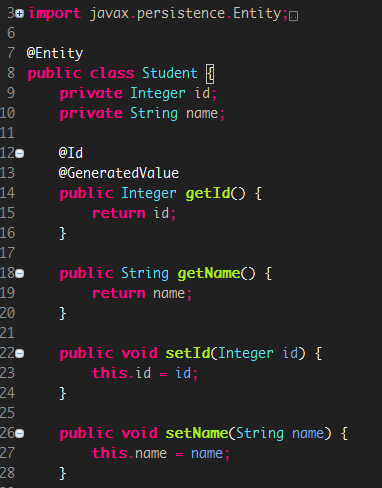
* + 1. 双向关联
       1. ManyToOne（OneToMany的双向和ManyToOne双向是一样的）

实例

Annotation

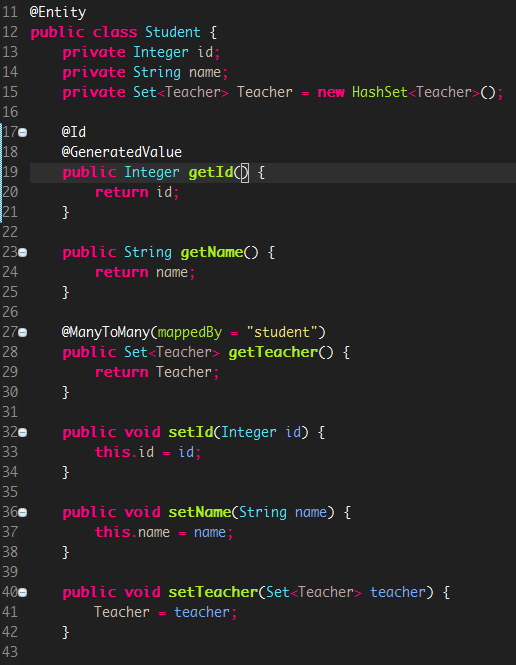
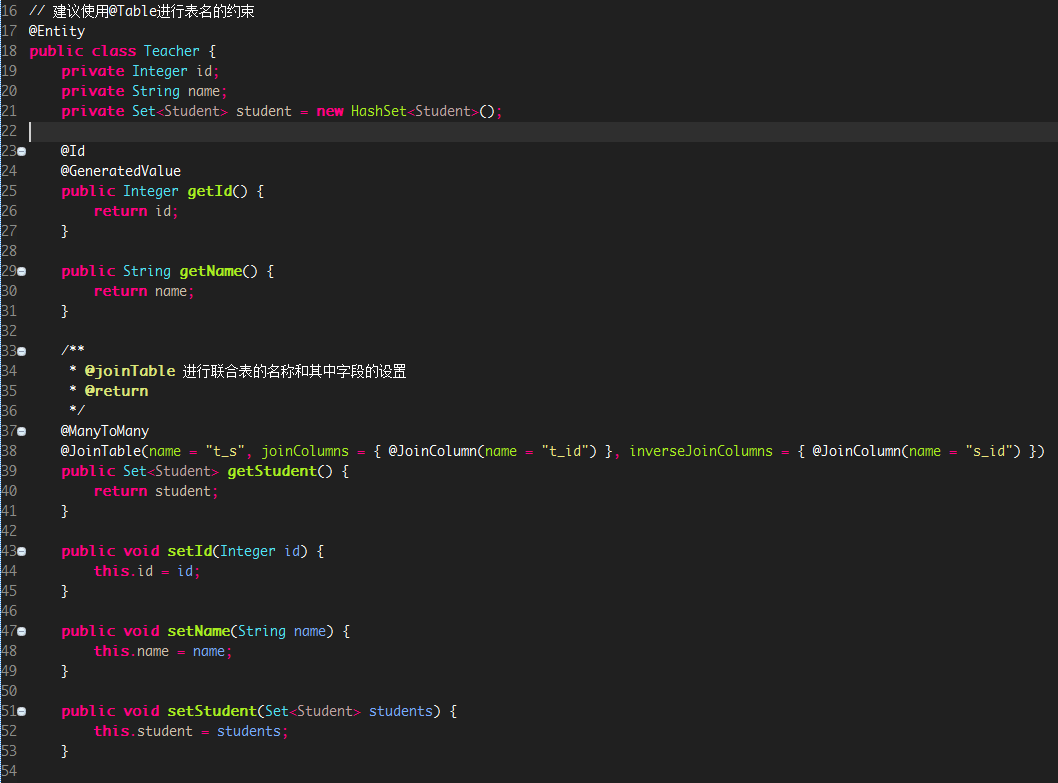
XML：直接参考Hibernate Reference Documentation和马士兵J2EE相关视频

* 1. **Many2Many**
     1. 设计思想：添加一张第三方表，用于存储两方的主键
     2. 单向关联
        + 1. 实例

Annotation

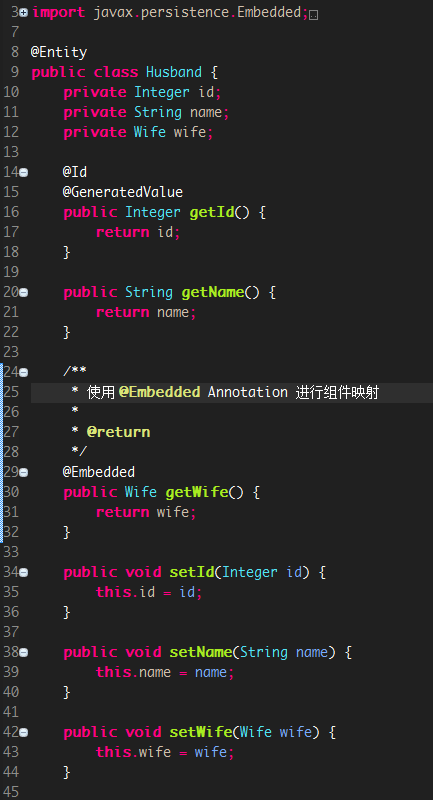
XML：直接参考Hibernate Reference Documentation和马士兵J2EE相关视频

* + 1. 双向关联
       - 1. 实例

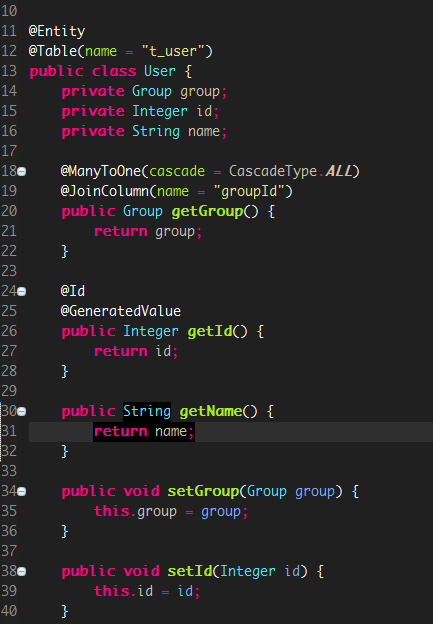
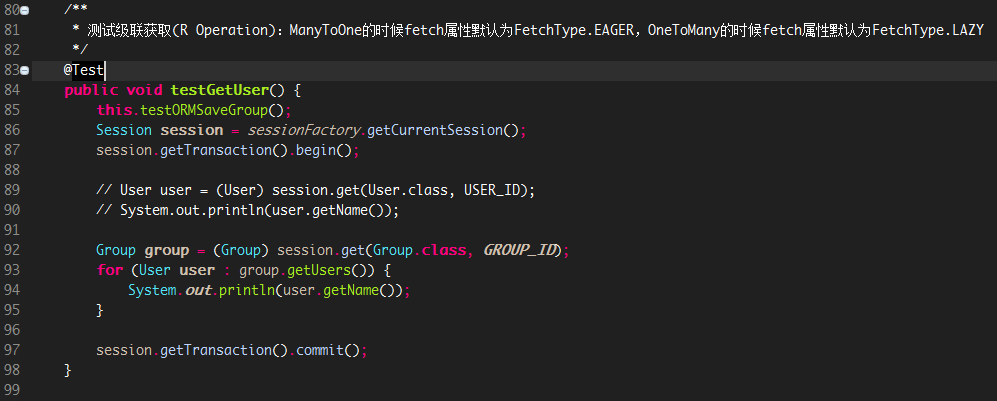
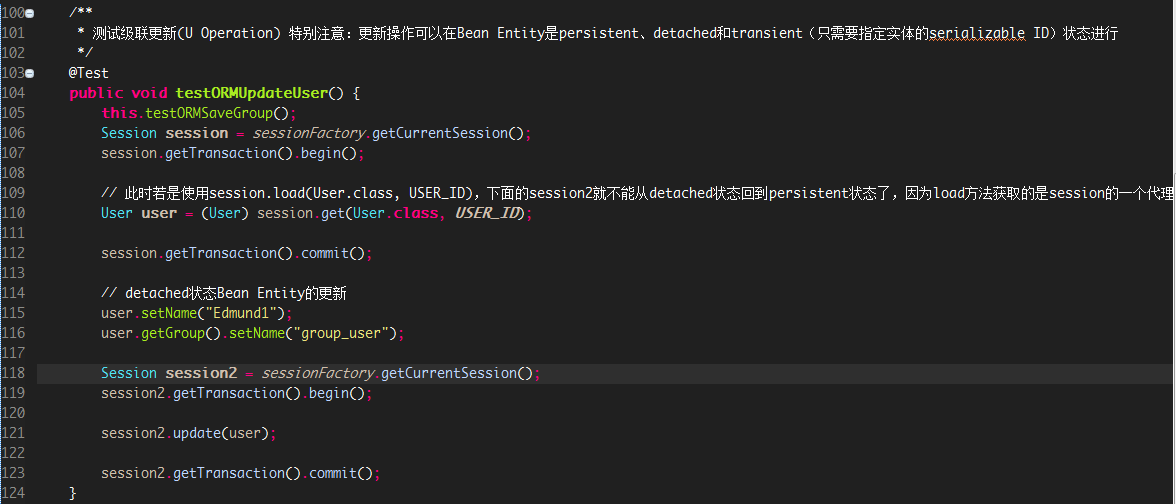
Annotation

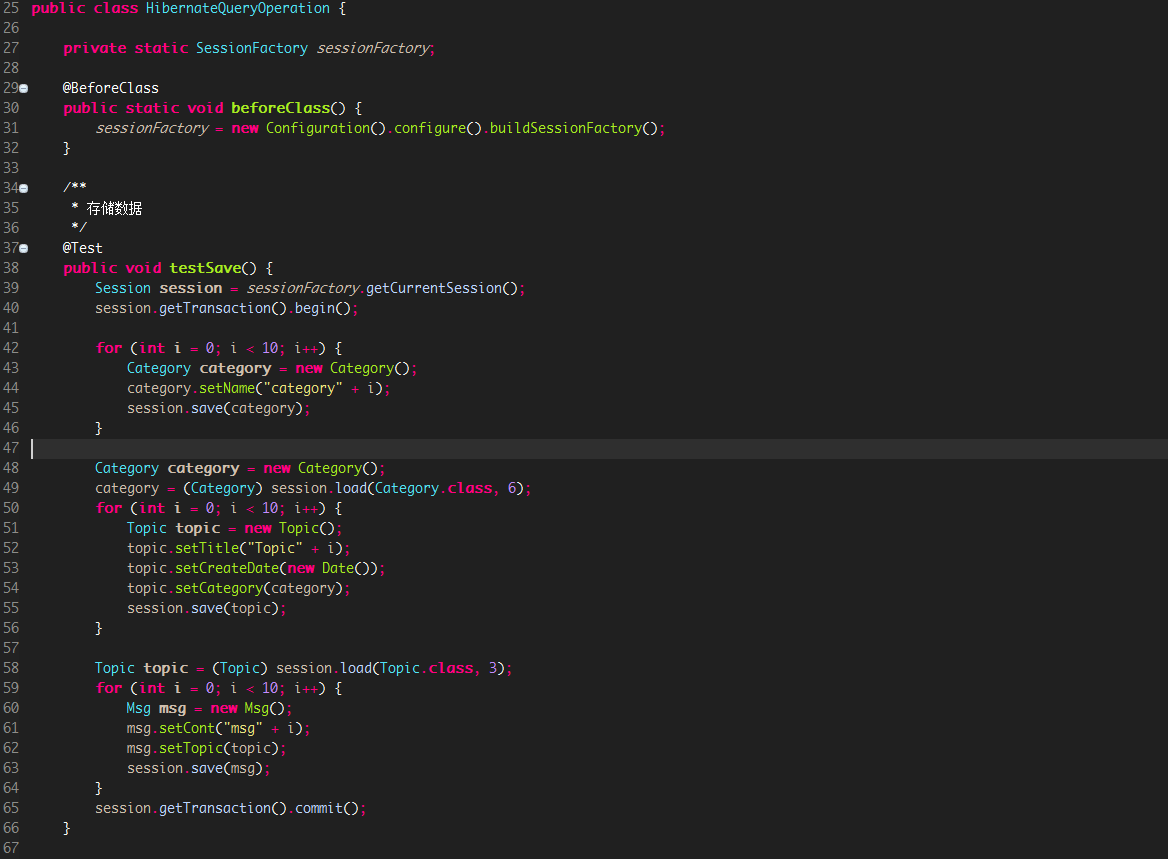
XML：直接参考Hibernate Reference Documentation和马士兵J2EE相关视频

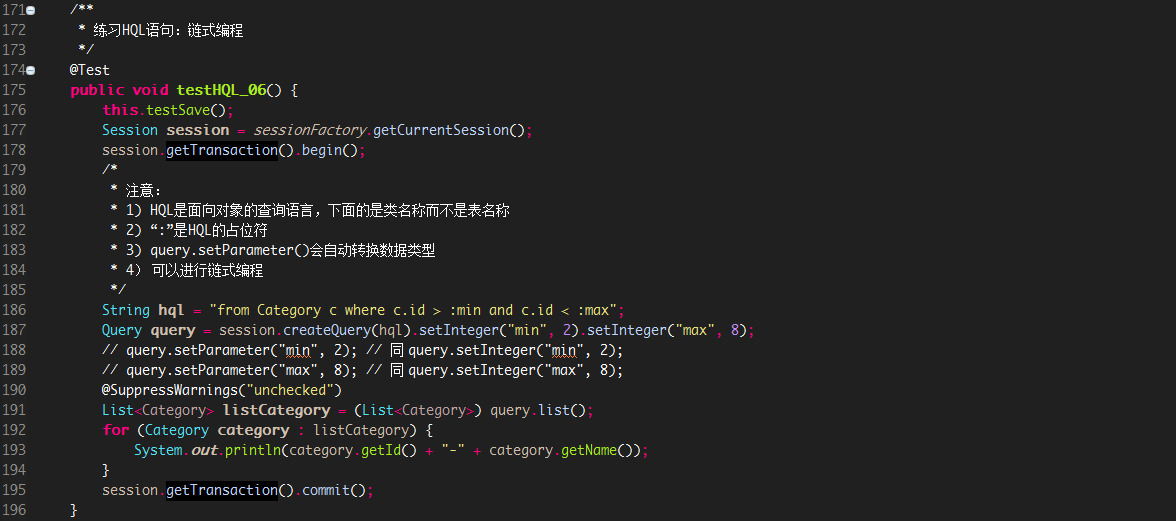
* 1. **组件映射（一个类中的属性指向另一个类的对象，是一种组合关系，可以使用组件映射。如果是一对一的话，数据库设计可以使用一张表）**
     1. Annotation：@Embedded
     2. XML：直接参考Hibernate Reference Documentation和马士兵J2EE相关视频
     3. 实例

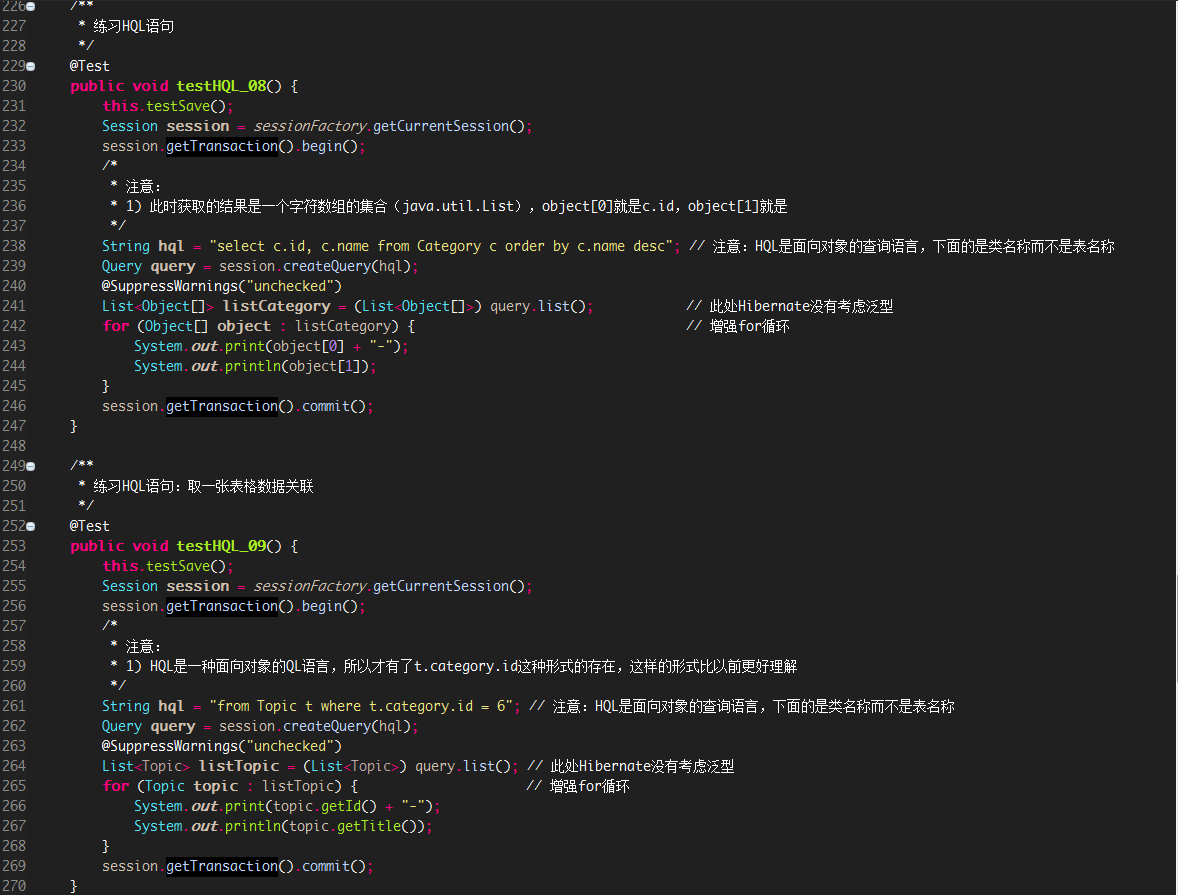


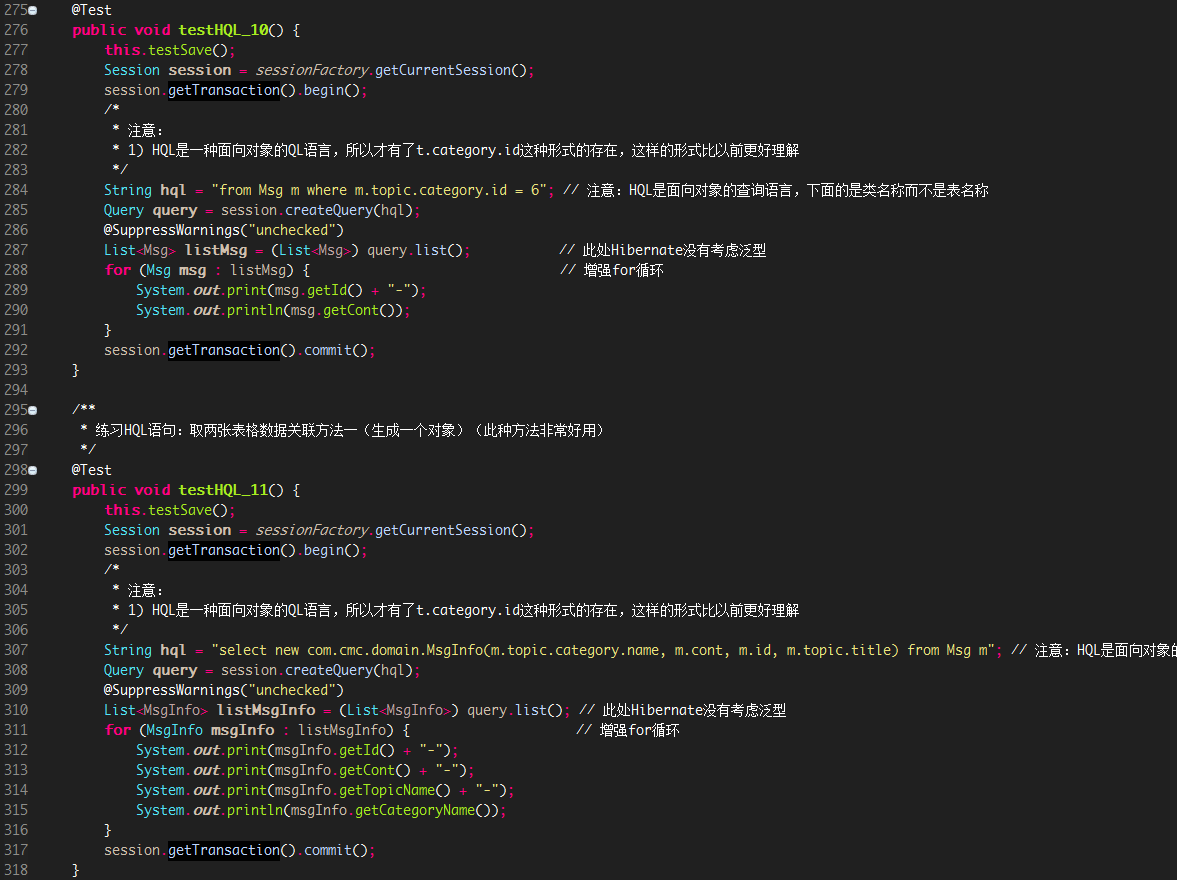
* + 1. @demo Hibernate\_Demo\_ORM\_Component

1. **O/R Mapping CRUD操作**
   1. **@reference Hibernate\_Demo\_ORM\_One2Many\_Many2One\_Bi\_CRUD**
   2. **CRUD实例**
      1. save
         1. 实例
         2. 说明
            1. 一定要在@ManyToOne中设置cascade 相应属性才可以级联保存相关对象，此例中级联保存Group对象
            2. 如果从One级联保存Many此时需要设置Many中包含One的内容
      2. **Retrieve**
         1. 实例
         2. 说明
            1. cascade属性只是在CUD操作中进行级联，R操作的级联需要设置fetch属性
            2. ManyToOne的fetch属性默认为FetchType.EAGER，即会立即加载级联数据，OneToMany的fetch属性默认为FetchType.LAZY，即当使用到该数据的时候才会加载
      3. **Update**
         1. 实例
         2. 说明
            1. 见实例注释
      4. **Delete**
         1. 说明：见注意事项viii
   3. **集合映射（不重要）**
      1. @reference Hibernate\_Demo\_ORM\_Collections
      2. 不太重要，需要时参照Hibernate开发手册和尚学堂Hibernate视频的集合映射
      3. 经典例题（**重要**）
         1. 树状结构的映射（极其重要+最好背诵+面试必备）
            1. 有待进一步完成(@version D20150811)
         2. 学生、课程、分数设计
            1. 有待进一步完成(@version D20150811)
   4. **继承映射（不重要）**
      1. 数据表的设计
         1. 一张总表
         2. 每个类分别一张表
         3. 每个子类分别一张表
      2. 不太重要，需要时参照Hibernate开发手册和尚学堂Hibernate视频的继承映射
   5. **注意事项**
      1. cascade：设定cascade可以设定在持久化操作对于关联对象的操作，即保存对象的时候可以保存与其级联对象；
      2. 铁律：双向关系在程序中要设置双向关联
      3. 铁律：双向关联要设置mappedBy属性
      4. cascade和fetch：
         1. cascade关联CUD操作
         2. fetch关联R操作
      5. 双向关系手动设置的时候中不要都设置fetch属性的值为FetchType.EAGER，默认Hibernate已经设置好了fetch属性的默认值，特殊情况的时候才需要手动设置fetch属性
      6. 在做删除或者更新操作的时候最好先load一下，在persistent状态进行操作（实际操作也应该是这样的，先判断数据库是否有相应的数据）
      7. O/R Mapping 编程模型
         1. **映射模型**
            1. JPA Annotation
            2. Hibernate Annotation Extension
            3. Hibernate XML
            4. JPA XML
         2. **编程接口**
            1. JPA
            2. Hibernate
         3. **数据查询**
            1. HQL
            2. EJBQL(JPAQ)
      8. 删除对象的时候如果不消除关联关系，则会删除和该对象有关的所有对象以及和该对象有关的所有对象有关的对象，一直级联下去，所以要删除某个对象的时候应该先删除级联关系，具体做法：
         1. 先设定关系为null，再删除记录，如果不删除记录，则该记录就会变成垃圾数据
         2. 直接使用HQL（或者EJBQL）语句，例如“delete from User u where u.id = 1”
2. **Hibernate查询**
   1. **查询类型** @description 按顺序排序，越靠前范围越大
      1. NativeSQL：本地查询语句，不好跨平台服务，不常用但是很重要，因为是最基本的
      2. HQL：Hibernate的查询语言，设置Dialect之后可以适用不同的数据库，面向对象的数据库查询方法
      3. EJBQL(JPQL 1.0)
      4. QBC：Query By Criterion
      5. QBE：Query By Example，QBC的一部分
   2. **查询实例** 
      1. @reference Hibernate\_Demo\_ORM\_Query\_Operation

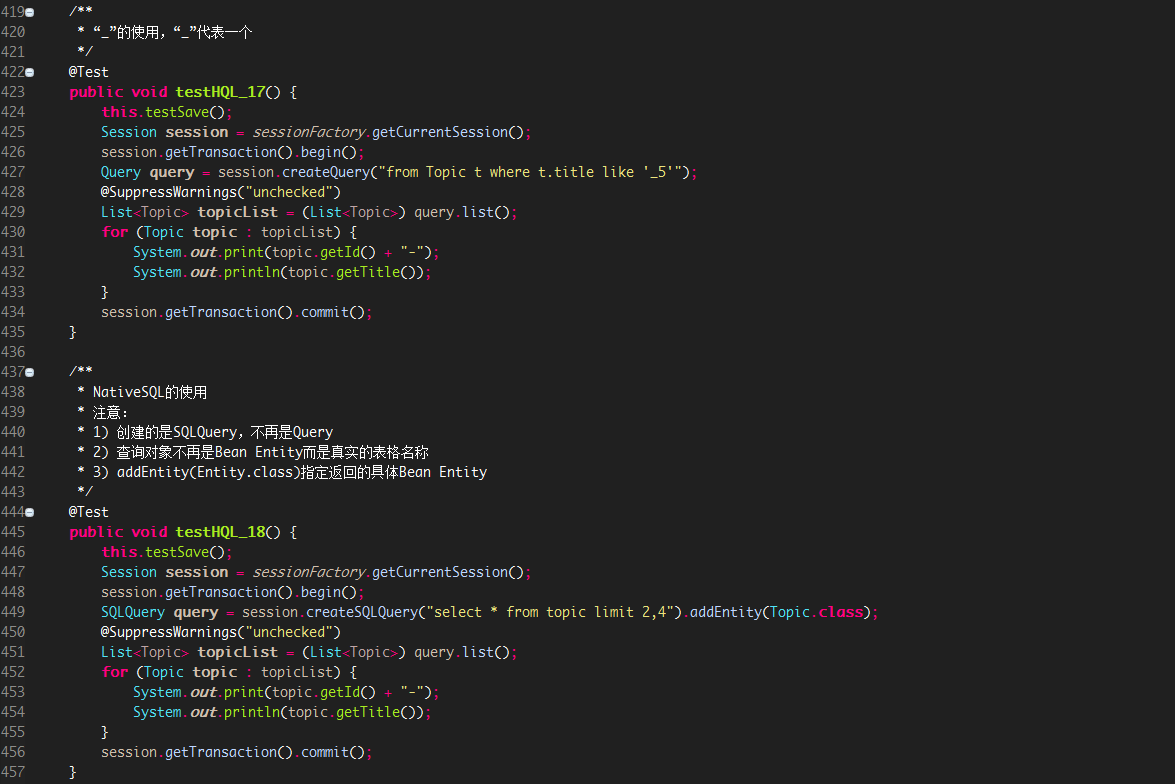
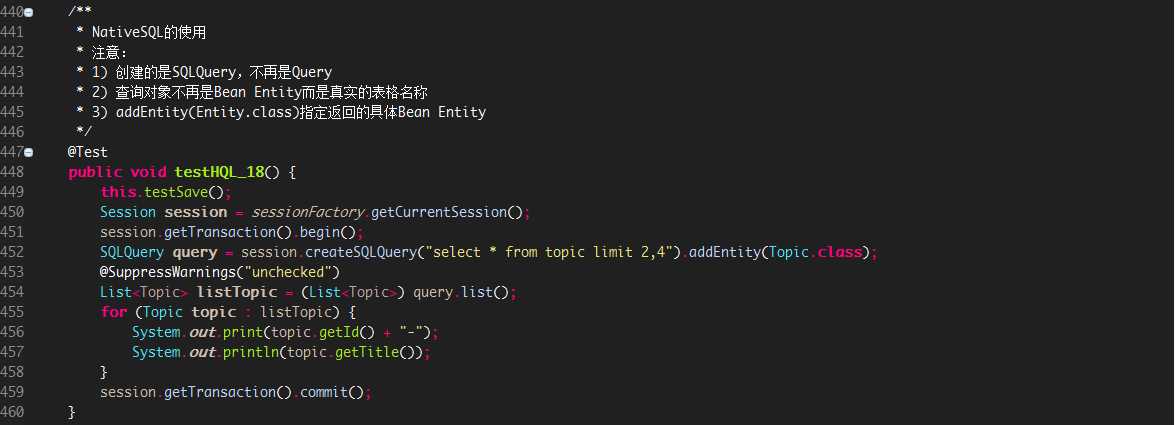
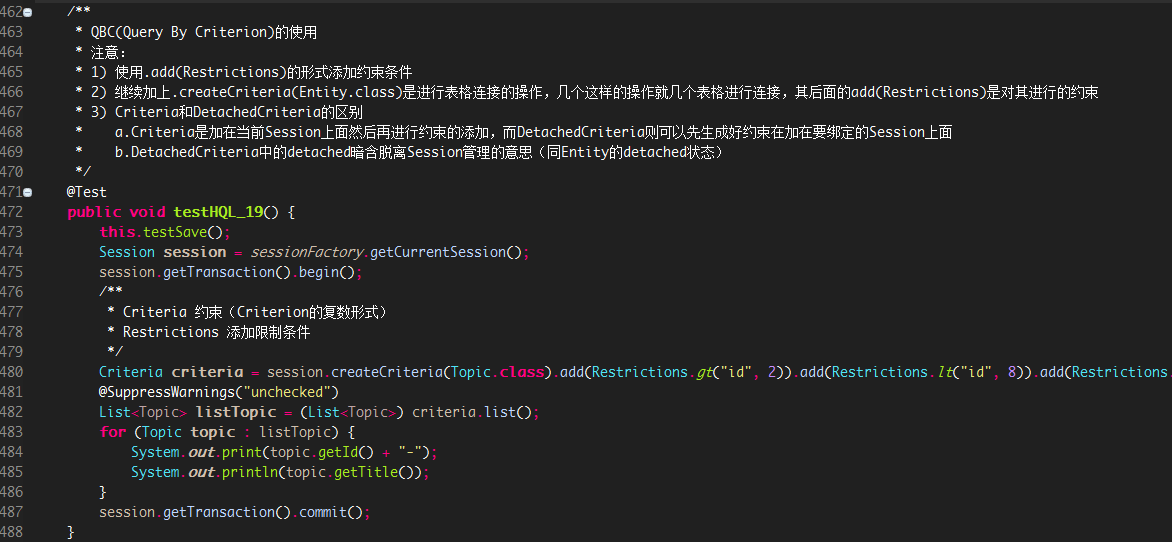


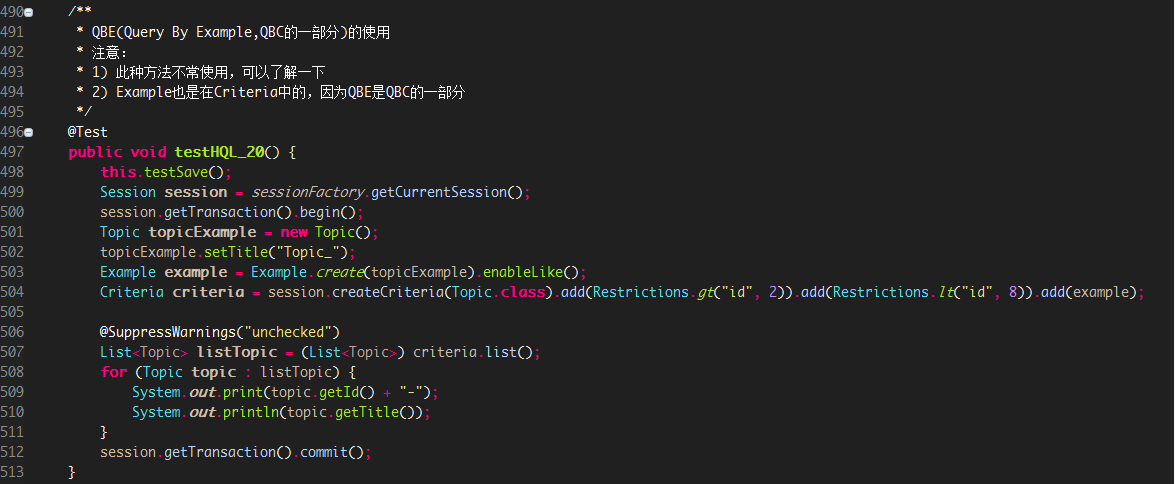
****

****

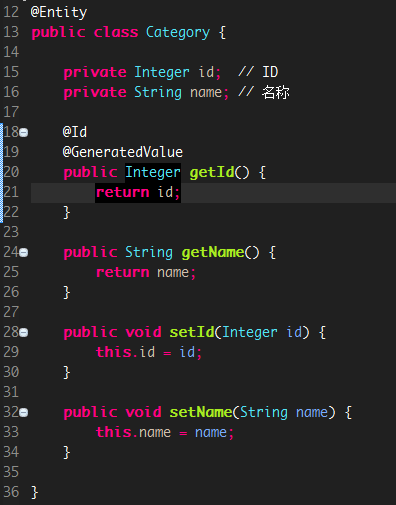
****

****

****

****

* 1. **注意**
     1. 部分注意事项请见代码注释
     2. 使用HQL的时候要使用导航关系（强烈建议）
     3. 命名查询不在此介绍，需要的的时 候查文档
     4. 应用驱动学习：发现问题去查询文档进行解决

1. **性能问题** @versionD20151024
   1. **1+N问题【典型的面试题】**（@description 1+N是指预期进行一个查询，结果是进行了1+N此查询）
      1. **含义**
         1. Hibernate中用户要取出两个关联的对象的表格数据的时候，如本例中的Topic和Category，此时用户本想只取出Topic的信息，但是因为Topic和Category是ManyToOne的关系，所以导致了在取Topic的同时去除了大量的Category信息，造成了无用的查询。
      2. **实例**
         1. Bean Entity:Topic
         2. Bean Entity:Category
         3. TestCase
      3. **解决方案**
         1. ManyToOne设置迟加载（慢加载）
         2. 在Entity上添加@BatchSize(size = int)注解进行批量操作，自己规定批量处理的处理单元大小
         3. 使用join fetch
            1. 使用session.createCriteria(Entity.class)，因为createCriteria()默认使用表连接进行数据的查询操作
            2. 使用session.createQuery(“from Topic t left join fetch t.category c”)，自己写join fetch的sql语句
2. **list和iterate比较** @versionD20151024
   1. **实例**
      1. list
      2. iterate
   2. **解析**
      1. 参考注释
      2. list不能利用缓存（一级缓存）、iterate可以利用缓存（首先查找session级的缓存）
      3. iterate更为高级，list更为常用
3. **一级缓存（Session级别）、二级缓存（SessionFactory级别）和查询缓存（QueryCache）(三级缓存)** @versionD20151024
   1. **缓存**
      1. **含义**
         1. 为了把本来应该存在硬盘上的信息存在内存中所开辟的一块内存空间
      2. **类别（hibernate中）**
         1. **一级缓存**
            1. 就是session级别的缓存
            2. 一级缓存默认提供，不需要在配置文件中进行启用
         2. **二级缓存**
            1. sessionFactory级别的缓存，跨越session存在，就是所有session级别的缓存的集合，为了使session级别缓存之间能够共享缓存
            2. 二级缓存需要在hibernate.cfg.xml中进行配置（启用以及启用哪个缓存，查询文档）
            3. 存入的内容需要符合的条件

经常被访问

改动不大且不会经常改动（以为改动了就要设计到访问数据库的操作，然后再存入缓存效率不如直接进行数据库操作）

数量有限

* + - * 1. **二级缓存要使用@Cache注解对Bean Entity进行标注**
        2. load和iterate默认使用二级缓存（包含添加和查询等操作）
        3. list默认使用二级缓存进行添加数据，查询的时候不使用
        4. query要使用不了二级缓存（因为每次的查询条件很可能不同，缓存结果没有太大的用处），需要打开查询缓存（三级缓存）
        5. 实例

@reference Hibernate\_Demo\_3Kinds\_Cache

@reference <http://cxl2012.iteye.com/blog/1944489>

配置

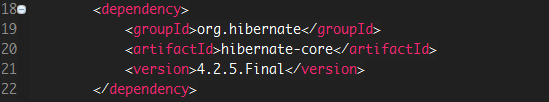
Hibernate.cfg.xml

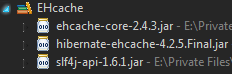


ehcache.xml

hibernate-core-4.2.5.Final的ehcache.xml版本

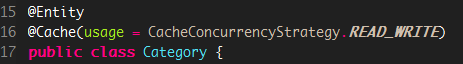
pom.xml



其他包

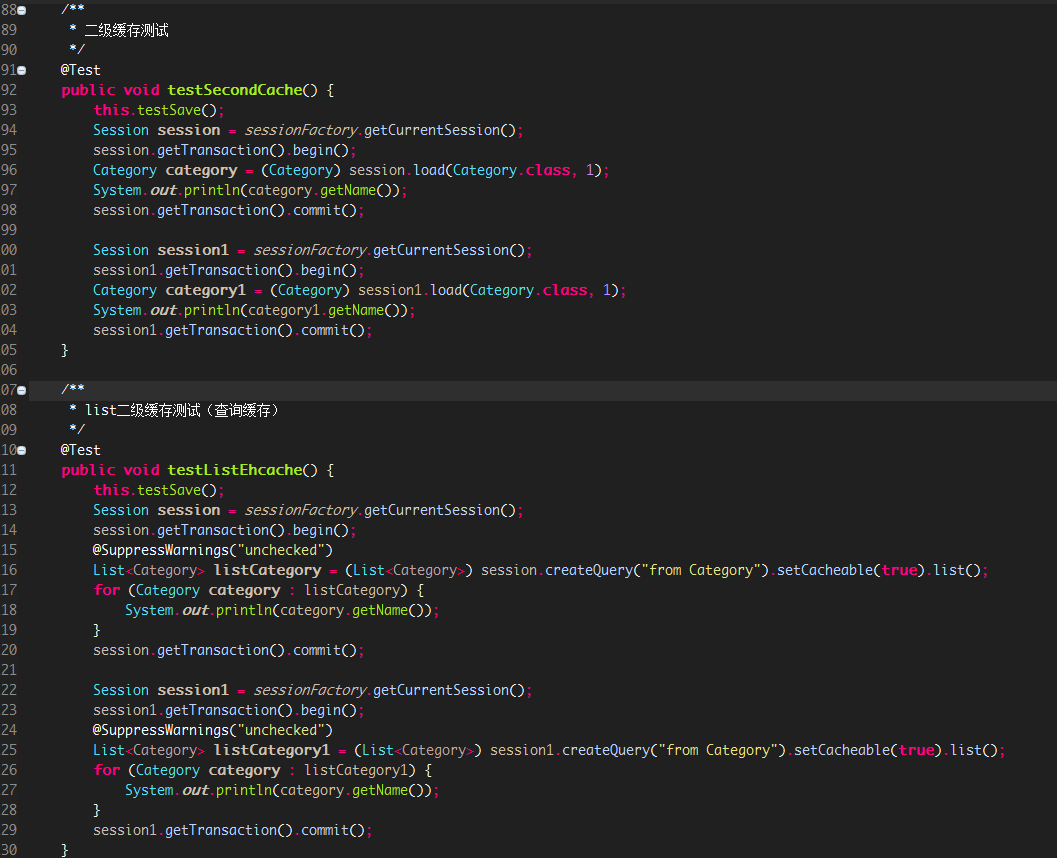
Entity

Category.java



Test

HibernateQueryOperation.java



* + - 1. **查询缓存（三级缓存）** 
         1. 查询缓存只有在两个查询语句一样的情况下使用
         2. 需要在hibernate.cfg.xml中进行配置，其依赖于二级缓存
         3. 调用Query的setCacheable(true)方法指明使用查询缓存
         4. 缓存算法（缓存中数据饱和了之后哪个对象需要被丢弃的算法）（纯为了面试）

LRU、LFU、FIFO（经常使用的三种）

LRU

Least Recently Used

最近很少使用的被丢弃（时间&频率方面）

LFU

Least Frequently Used（频率高低）

最少被使用（频率方面）

FIFO

First In First Out

早来的先丢弃（顺序方面）

ehcache中的cache模块指定具体的缓存算法

memoryStoreEvictionPolicy = “LRU”

1. **事务的并发处理（面试方面意义比较大）** @versionD20151024 Pending到数据库事务部分
   1. **事务（Transaction）**
      1. **含义**
         1. 是访问并可能更新数据库中各种数据项的一个程序执行单元
      2. **特性（ACID）**
         1. Atomic（原子性：要么处理、要么不处理）
            1. 一个事务是一个不可分割的工作单位，事务中包括的诸操作要么都做，要么都不做。
         2. Consistent（一致性：不改变约束）
            1. 事务必须是使数据库从一个一致性状态变到另一个一致性状态。一致性与原子性是密切相关的。
         3. Isolation（独立性：事务之间不能有约束）
            1. 一个事务的执行不能被其他事务干扰。即一个事务内部的操作及使用的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰。
         4. Durability（持久性：不能无缘无故进行回滚）
            1. 持久性也称永久性（permanence），指一个事务一旦提交，它对数据库中数据的改变就应该是永久性的。接下来的其他操作或故障不应该对其有任何影响。
   2. **数据并发操作存在的问题**
      1. **丢失更新（Lost Update）**
         1. **第一类丢失更新(First Lost Update)**
            1. 第一类丢失更新一般是指同时发生了两个事务，不仅丢了第一个事务的数据还丢了第二个事务的数据
         2. **第二类丢失更新(Second Lost Update)**
            1. **概念**

当两个或多个事务选择同一行，然后基于最初选定的值更新该行时，会发生丢失更新问题。每个事务都不知道其它事务的存在。最后的更新将重写由其它事务所做的更新，这将导致数据丢失

* + - * 1. **实例**

事务A和事务B同时修改某行的值

事务A将数值改为1并提交

事务B将数值改为2并提交

这时数据的值为2，事务A所做的更新将会丢失

* + - * 1. **解决方法**

对行加锁，只允许并发一个更新事务

* + 1. **脏读（Dirty Read）（未确定的相关性）**
       1. **概念**
          1. 当第二个事务选择其它事务正在更新的行时，会发生未确认的相关性问题。第二个事务正在读取的数据还没有确认并且可能由更新此行的事务所更改
       2. **实例**
          1. Mary的原工资为1000, 财务人员将Mary的工资改为了8000(但未提交事务)
          2. Mary读取自己的工资 ,发现自己的工资变为了8000，欢天喜地
          3. 而财务发现操作有误，回滚了事务,Mary的工资又变为了1000
          4. 像这样,Mary记取的工资数8000是一个脏数据
       3. **解决方法**
          1. 如果在第一个事务提交前，任何其他事务不可读取其修改过的值，则可以避免该问题
    2. **不可重复读（两次结果不一样，导致不能重复读取相同的数据）(Non-repeated Read)**
       1. **概念**
          1. 当第二个事务多次访问同一行而且每次读取不同的数据时，会发生不一致的分析问题。不一致的分析与未确认的相关性类似，因为其它事务也是正在更改第二个事务正在读取的数据。然而，在不一致的分析中，第二个事务读取的数据是由于已进行了更改的事务提交的。而且，不一致的分析涉及多次（两次或更多）读取同一行，而且每次信息都由其它事务更改；因而该行被非重复读取。在一个事务中前后两次读取的结果并不致，导致了不可重复读
       2. **实例**
          1. 在事务1中，Mary 读取了自己的工资为1000,操作并没有完成
          2. 在事务2中，这时财务人员修改了Mary的工资为2000,并提交了事务
          3. 在事务1中，Mary 再次读取自己的工资时，工资变为了2000
          4. 导致两次读取的工资不一致
       3. **解决方法**
          1. 如果只有在修改事务完全提交之后才可以读取数据，则可以避免该问题
    3. **幻读（针对以添加和删除的不可重复读）（Phantom Read）**
       1. **概念**
          1. 当对某行执行插入或删除操作，而该行属于某个事务正在读取的行的范围时，会发生幻像读问题。事务第一次读的行范围显示出其中一行已不复存在于第二次读或后续读中，因为该行已被其它事务删除。同样，由于其它事务的插入操作，事务的第二次或后续读显示有一行已不存在于原始读中
       2. **实例**
          1. 目前工资为1000的员工有10人
          2. 事务1,读取所有工资为1000的员工
          3. 这时事务2向employee表插入了一条员工记录，工资也为1000
          4. 事务1再次读取所有工资为1000的员工 共读取到了11条记录
       3. **解决方法**
          1. 如果在操作事务完成数据处理之前，任何其他事务都不可以添加新数据，则可避免该问题
    4. **一般出现的问题分为脏读和非重复读两种**
  1. **数据库的事务隔离机制（不仅仅局限于Hibernate）**
     1. @reference java.sql.Connection
     2. **种类**
        1. **read-uncommitted（对应的静态数值：1=0001，左移效率高，且可以同时设置权限，1100之类）**
           1. 能够读取没有提交的数据
           2. 会出现：脏读、非重复读、幻读
        2. **read-committed（对应的静态数值：2=0010）**
           1. 提交了数据之后才能够进行读取
           2. 会出现：非重复读、幻读
        3. **repeatable-read（对应的静态数值：4=0100）**
           1. MySQL引擎默认设置
           2. 读出数据之后给数据加锁，这样其他用户就不能修改该数据的数值
           3. 会出现：幻读
        4. **serializable（对应的静态数值：8=1000）**
           1. 序列化的执行，这样就没有了并发机制，不会出现事务的问题
           2. 效率较低
           3. 出现情况：检查数据库与之相关的问题
     3. **其他说明**
        1. 只要数据库支持事务（因为事务有ACID特性），就不可能出现第一类丢失更新
        2. 事务的级别越高，效率越低。因为级别越高加的锁越多越复杂
        3. 一般数据库引擎为read-committed（折中的办法，即高层度地解决了事务并发问题，又提高了查询效率），然后通过代码解决非重复读（幻读的解决方式基本上只有设置隔离级别为serializable，即级别8）
  2. **Hibernate的事务隔离级别**
     1. **配置**
        1. 设置hibernate.cfg.xml中的属性hibernate.connection.isolation = 2 (read-committed，默认事务隔离级别)
        2. 如果不进行事务隔离级别的设置，则是依赖于使用的数据库引擎的默认设置
     2. **解决“非重复读”问题的方法（使用悲观锁和乐观锁）**
        1. **悲观锁**
           1. 读数据的时候给数据加锁，这样其他用户在该施加锁的用户还没有处理完的时候就不能对该数据进行修改操作
           2. 但是依赖于使用的数据库引擎的加锁机制
           3. select .. for update（数据库原始加锁机制SQL，使用for update进行加锁声明）
           4. Hibernate中的加锁语句：load(xxx.class, i, LockMode.Upgrade)

LockMode.None

没有锁的机制，Transaction结束的时候回自动切换到此模式

LockMode.read

在查询（R操作）的时候，Hibernate会自动获取该锁

LockMode.write

在添加（C）、修改（U）、删除（D）的时候，Hibernate会自动获取该锁

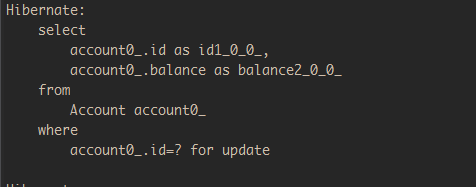
以上3重加锁的模式是Hibernate内部使用的，也就是说在做数据库操作的时候会自动上锁和下锁

LockMode.Upgrade

告诉数据库在本用户读取数据的时候请其给该数据加锁

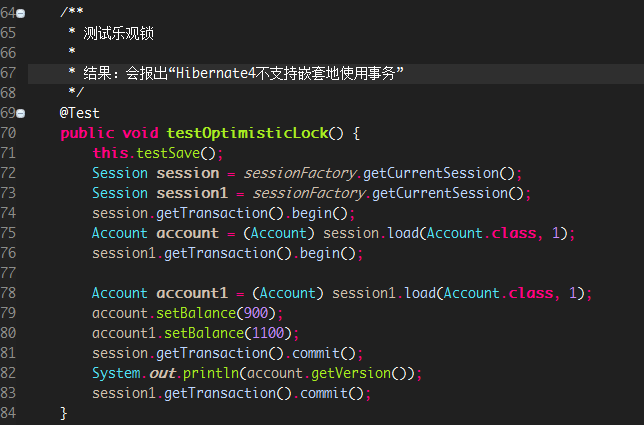
程序实例

测试代码

结果（使用SQL的“for update”加上锁）

* + - 1. **乐观锁**
         1. 给每个对象添加一个version(int)对象，如果其他用户修改了该对象version的值会自动加一，然后本用户在提交之前比较version的前后数值，如果不同则提交失败，其他操作解决之，相同则提交成功
         2. 程序实例

实体（java.persistence.Version就是为了使用乐观锁而定义的一个注解，记录了版本号）

测试实例