# 1. ORM和Hibernate框架

说明：前面讲的Struts框架是基于MVC模式的应用层框架。那么现在讲的Hibernate框架属于持久层框架，用于数据访问层。实质上Hibernate的底层还是JDBC实现的，因此Hibernate可用于任何操作数据库的Java项目，不管是Java SE还是Java EE。为了简单，开始学习时我们只用Java SE项目。

## 1.1 ORM和Hibernate的关系

ORM中文含义是对象关系映射，即Object Relation Mapping。在高级语言中我们使用面向对象的方式，而在关系型数据库中，数据是存放在二维表中的，ORM的目标就是实现关系型数据库中的数据能够映射成程序中的对象，让开发者能以面向对象的方式操作数据库。比如，程序实现了ORM后，可以直接从数据库中拿到一个Student“对象”，或者将一个对象“添加”到数据库中。

我们之间学习的DBUtils实现了对JDBC的封装，我们使用它也能实现一些简单的“数据 - 对象”映射。而Hibernate框架就是ORM的一个实现。Hibernate就实现了对象和表的映射、对象属性和表中字段的映射以及表中关系之间的映射。

## 1.2 JDBC和Hibernate的比较

直观的感受是Hibernate通过JDBC实现了ORM框架，能让开发者使用OOP思想操作数据库，使得操作简单高效。现在我们从诸多方面比较JDBC和Hibernate，以对它们有较全面的认识。

一、JDBC的优缺点

JDBC的最显著的优点是执行效率高、速度快，因为JDBC是Java连接关系型数据库最底层的东西。

JDBC有如下缺点（相比较于Hibernate，也是实际中比较看重的）：

（1）JDBC没有做数据的缓存（即把部分数据缓存在内存中提高效率，避免频繁查询）

（2）JDBC编程时较繁琐，需要考虑异常处理等。

（3）JDBC使用SQL时，跨平台性较差（例如不同数据库的分页查询、存储过程等SQL差异较大，不能实现跨平台）

二、Hibernate的优缺点

Hibernate的优点：

（1）ORM框架，完全的面向对象编程；

（2）Hibernate有一级缓存、二级缓存和查询缓存等（学习的重点）；

（3）调用框架的方法操作数据库，编程时比较简单；

（4）Hibernate能自动处理跨平台问题。

Hibernate的缺点：

（1）效率比较低，Hibernate框架本身对JDBC的封装比较“重量级”；

（2）如果数据库表中数据可能达到千万级别，则Hibernate不适合；

（3）如果数据库表之间的关系特别复杂，则Hibernate不适合。

三、总结

JDBC效率高，Java中关系型数据库的框架都是基于JDBC封装的。Hibernate能让程序员以面向对象的方式方便地操作数据库，但不适合数据量大且数据关系复杂的应用。因此Hibernate使用的场景就是企业内部系统（一般企业人力资源的数据不会太大），而一些互联网公司是不会采取使用Hibernate框架的。

题外话：如果有存储大数据量的需求，目前一般会使用非关系型数据库（NoSQL，Not Only SQL）。这种类型的数据库查询速度较关系型数据库快得多（大数据量时明显），但相应地NoSQL这些数据库对事务的控制较弱，因为效率和事务控制不可兼得，二者是矛盾的，这是牺牲事务提高效率的做法。

因此，一定要分清各种技术的使用场景，不要只会一味地使用一种技术，要会分场景使用技术。如果项目不适合用Hibernate，则可考虑使用Mybatis等框架。（NoSQL和Mybatis提后都会学习的）。

## 1.3 第一个Hibernate案例

首先搭建环境：在官网http://hibernate.org/orm/ 下载最新版的Hibernate，解压缩文件后，在Java项目中引入lib/required目录下所有的jar包。注意，Hibernate 5.2以上版本只支持JDK8及以上环境。主要jar包介绍如下：

（1）antlr-2.7.7.jar：一个语言转换工具，Hibernate使用它实现从HQL到SQL的转换。

（2）dom4j-1.6.1.jar：用于XML解析，这是我们学过的。

（3）hibernate-core-5.2.8.Final.jar：Hibernate的核心包。

（4）javassist-3.20.0-GA.jar：该包是用来创建代理对象的。我们之前学过的创建动态代理对象的方式是JDK接口代理和CGLIB代理，而这里的javassist可以说是创建代理对象的第三种方式。

（5）jboss-transaction-api\_1.2\_spec-1.0.1.Final.jar：JBoss提供的JTA（Hibernate现在是JBoss负责开发）。那么什么是JTA呢？JTA是“Java Transaction API”，它是sun公司给分布式事务处理制定的API规范（所谓规范说白了就是定义了JTA接口）。分布式事务就是在不同的JVM中实现事务控制。实际中很多项目的部署是分布式的，这样以提供高可用服务，如果用户两次请求到不同的服务器（即不同的JVM），那么如何实现它们的事务控制呢？这就是分布式事务需要解决的（这里不涉及）。

另外，由于Hibernate是依赖JDBC的，因此在项目中还必须引入JDBC驱动包。

此案例中只对学生进行操作，也就是说数据库中只有一张学生表（在MySQL中新建一个dbtest数据库）。对应的SQL为：

|  |
| --- |
| **create table** student(  **id int AUTO\_INCREMENT primary key**, *-- 学生编号是自增长主键* **name varchar**(20), *-- 姓名* **entranceTime datetime** *-- 入学时间* ) |

准备好数据库后，我们就在程序中进行Hibernate案例开发。步骤如下：

（1）写好Student实体类。比如在com.zhang.entity包下新建Student类，类中的属性可参考student数据库表。例如：

|  |
| --- |
| **package** com.zhang.entity;  **import** java.util.Date;  **public class** Student {  **private int id**; *// 编号* **private** String **name**; *// 姓名* **private** Date **entranceTime**; *// 入学时间  // 自行写getter和setter，以及toString等...* } |

（2）在实体类的同一个包下，应该给每个实体配置一个Hibernate的映射文件（Hibernate Mapping XML）。该映射文件的作用就是实现对象和数据库表的对应关系。映射文件命名方式为“实体类名.hbm.xml”。比如这里需要在com.zhang.entity下新建一个Student.hbm.xml文件用于配置Student类和student表之间的对应关系：

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"** *?>* **<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd"*>*** *<!-- 根节点中可指定要映射的类所在的包。若不指定，则下面用到类时，就要写类的包名了 -->* <**hibernate-mapping package="com.zhang.entity"**>  *<!-- class节点指明类和表的对应关系。name表示实体类，table就是对应的表名。 -->  <!-- 由于hibernate-mapping中指明了package包名，因此这里name直接写Student即可，否则要写上包名 -->* <**class name="Student" table="student"**>  *<!-- class标签中就来指定类属性和表字段的映射关系。 -->  <!-- 首先，用id标签来进行主键映射。name表示属性名，column表示对应的表的字段名 -->* <**id name="id" column="id"**>  *<!-- id标签中，通过generator标签指明主键生成策略，即通过class属性指定。这里写increment，表示会该主键将自动增长。 -->* <**generator class="increment"**></**generator**>  </**id**>  *<!-- 下面的property标签就是其他属性和字段的映射关系了 -->* <**property name="name" column="name"**></**property**>  <**property name="entranceTime" column="entranceTime"**></**property**>  </**class**> </**hibernate-mapping**> |

（3）在src目录下配置Hibernate的主配置文件，名字为hibernate.cfg.xml。主配置文件主要用于配置数据库连接以及指明加载上述的映射文件。如：

|  |
| --- |
| *<?***xml version='1.0' encoding='utf-8'***?>* **<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd"*>*** <**hibernate-configuration**>  <**session-factory**>  *<!-- 所有配置在session-factory节点中 -->  <!-- property标签来配置数据库连接等信息 -->* <**property name="connection.driver\_class"**>com.mysql.jdbc.Driver</**property**>  <**property name="connection.url"**>jdbc:mysql://localhost:3306/dbtest?useSSL=false**&amp;**useUnicode=true**&amp;**characterEncoding=utf-8</**property**>  <**property name="connection.username"**>root</**property**>  <**property name="connection.password"**>123456</**property**>  *<!-- 设置数据库方言 -->* <**property name="dialect"**>org.hibernate.dialect.MySQLDialect</**property**>  *<!-- 执行数据库操作时，是否显示SQL语句。默认是不显示的 -->* <**property name="show\_sql"**>true</**property**>  *<!-- 显示SQL语句时，是否格式化。默认不会格式化。这两个配置都是为了方便调试的 -->* <**property name="format\_sql"**>true</**property**>  *<!-- 最后，要用mapping标签指明加载的映射文件 -->* <**mapping resource="com/zhang/entity/Student.hbm.xml"**/>  </**session-factory**> </**hibernate-configuration**> |

这里再提一下数据库的方言（Dialect）。讲数据库时我们就提到每种数据库是有不同的方言的，就是它们针对标准SQL扩展的部分，说白了就是各个数据库厂商提供的“不符合标准、无法跨数据库平台通用”的SQL语言。Hibernate的封装为我们实现了“跨平台”，但需要先告诉Hibernate我们所使用数据库的方言，上述的dialect就是在Hibernate中配置数据库方言属性的。

（4）这样就可以写测试类来使用Hibernate提供的数据库操作了。首先加载默认的hibernate.cfg.xml文件，然后通过SessionFactory创建一个Session用来操作数据库。这里的Session和HttpSession是不一样的，Hibernate中的Session就相当于是一个和数据库会话的连接，主要用它来进行数据操作。实例代码如下：

|  |
| --- |
| **package** com.zhang.test;  **import** com.zhang.entity.Student; **import** org.hibernate.Session; **import** org.hibernate.SessionFactory; **import** org.hibernate.Transaction; **import** org.hibernate.cfg.Configuration; **import** java.util.Date;  **public class** Demo {  **public static void** main(String[] args) {  *// 创建配置对象* Configuration configuration = **new** Configuration();  configuration.configure(); *// 此方法默认加载classpath中的hibernate.cfg.xml文件  // 用配置对象创建一个session工厂对象。* SessionFactory sessionFactory = configuration.buildSessionFactory();  *// 使用工厂拿到一个session* Session session = sessionFactory.openSession();  *// 下面就用session向数据库中保存一个student对象。* Student student = **new** Student(10, **"张三"**, **new** Date());  *// Hibernate事物默认不是自动提交的。因此操作前先开始事物，使用beginTransaction()开启，并返回事物对象。* Transaction ta = session.beginTransaction();  *// 直接调用save方法保存对象* session.save(student);  *// 完了就提交事物。* ta.commit();  *// 最后关闭连接* session.close();  sessionFactory.close();  } } |

运行程序后我们刷新数据库中的student表，就能看到新增的数据了。发现Hibernate确实实现了ORM，上述我们只调用了save()方法，只要传递对象进去，Hibernate就能自动根据映射关系，找到对象的表并插入正确的数据。

说明：

（1）保存时，Student对象不用设置id属性值，因为已设置该主键生成策略是自增的，由Hibernate完成主键设置即可。

（2）我们的“入学时间”在数据库中使用了datetime类型。datetime类型既可保存日期，又可保存时间，而date类型只能保存日期，time类型只能保存时间。

（3）执行session.get()方法时，会利用Java的反射机制创建一个对象，利用的是默认的构造函数。所以一个持久化类中必须有一个默认的构造函数。

（4）为什么实现持久化类的序列化：分布式环境中（即不同的JVM间）对象的网络传输需要把对象数据转成二进制传输，序列化主要的作用就是在网络上传输。

# 2. Hibernate基本操作（CRUD）

这里主要讲使用Hibernate进行增删改查等操作。

## 2.1 基础API

（1）配置管理类对象Configuration：

configuration.configure()：加载主配置文件，默认加载hibernate.cfg.xml。

configuration.configure(“com/config/hibernate.cfg.xml”)：加载指定目录下的配置文件。

configuration.buildSessionFacroty()：获得session的工厂对象。

（2）SessionFactory：session工厂类，用于创建session对象：

sessionFactory.openSession()：创建一个session对象。

sessionFactory.getCurrentSession()：创建或取出当前session对象（使用该方法需要以后进行配置）。

（3）Session对象：该对象维护了一个与数据库连接的会话，只要进行数据库操作，都会用到这个对象。

session中提供了很多方法，主要用于：控制事物、获得session连接的一些信息，以及常用的基于主键的CRUD操作。比如：

session.beginTransaction()：开启事物；

session.getTransaction.commit()：拿到事物并提交，这样就不用一个对象专门接收Transaction对象了。

session.save(Object obj)：向数据库中保存一个对象。

session.update(Object obj)：基于主键更新一个对象。

session.saveOrUpdate(Object obj)：基于主键保存或者更新一个对象。

session.get(Class<T> clazz, Serializable id)：按照主键查询。Serializable是一个接口，像整型和字符串类型都实现了此接口，因此可传递整型和字符串等。

session.load(Class<T> clazz, Serializable id)：按照主键查询。支持懒加载。

session.delete(Object obj)：根据主键删除。

由于session中提供的方法一般是基于主键的，因此对象的主键属性一定要有值。例如删除操作，只需要给出主键属性即可，其他属性无需添加。

例如：

|  |
| --- |
| **public class** Demo {  **public static void** main(String[] args) {  Configuration config = **new** Configuration();  config.configure();  SessionFactory sf = config.buildSessionFactory();  Session session = sf.openSession();  *// 1. 准备更新数据。* Student student = **new** Student();  student.setId(3); *// 要更新的是主键为3的学生。下面设置的属性就是将要更新进去的数据了* student.setName(**"李四"**);  student.setEntranceTime(**new** Date());  *// 一定要设置事物，否则不会真正对数据库进行操作。* session.beginTransaction();  session.update(student);  session.getTransaction().commit(); *// 提交  // 2. 删除数据* Student student1 = **new** Student();  student1.setId(2); *// 只要设置好主键即可删除* session.beginTransaction();  session.delete(student1);  session.getTransaction().commit();  *// 3. 查询数据* session.beginTransaction();  Student student2 = session.get(Student.**class**, 1); *// 获得主键为1的学生* System.***out***.println(student2);  session.getTransaction().commit();  session.close(); *// 一定要关闭session和sf资源，否则程序不会退出。* sf.close();  } } |

需要注意：当使用update()或者delete()时，若数据库中不存在对应的数据，比如没有主键是2的学生数据，但是你要更新或删除它，那么Hibernate就会报错，报错信息为：

Batch update returned unexpected row count from update [0]; actual row count: 0; expected: 1。

而调用get()和load()方法获得数据时，若数据库中无此数据，那么也不会报错，而是返回null。因此，我们在用session进行更新或删除之前，最好先调用get()或load()方法先获取一下数据，若获得的数据不为null，再更新/删除本对象。

当然，对于更新，也能使用saveOrUpdate()方法，也就是说，若数据库中不存在，则直接添加此对象。

## 2.2 其他查询方式

上述我们看到，使用session提供的基础操作方法，可用的功能较有限，只能实现基本的增删改查。有时，我们进行的数据库操作较复杂，那么Hibernate也提供了多种操作数据库的方法，比如下面的HQL等。

当然，提供的这些查询仍然是通过session获得的，比如session.createQuery()等。

### 2.2.1 了解HQL

HQL是Hibernate Query Language的缩写，也就是hibernate提供的面向对象的查询语言。HQL语句和SQL语法很类似，但是HQL中不是写表和字段的名字，而是写对应实体类的名字和属性名，因此HQL中，类和属性名是区分大小写的。

例如，要查询学生表中编号为1的学生，那么SQL为：“from Student where id = 1”。代码实例：

|  |
| --- |
| **package** com.zhang.test;  **import** com.zhang.entity.Student; **import** org.hibernate.Session; **import** org.hibernate.cfg.Configuration; **import** org.hibernate.query.Query;  **import** java.util.List;  **public class** Demo {  **public static void** main(String[] args) {  *// 链式编程直接获得session。* Session session = **new** Configuration().configure().buildSessionFactory().openSession();  session.beginTransaction();  *// 用session.createQuery()方法创建HQL查询，方法参数就是HQL语句。同样，是可以使用参数化查询的。这里使用“:参数名”来作为一个占位符。* Query<Student> query = session.createQuery(**"from Student where id = :id"**, Student.**class**);  query.setParameter(**"id"**, 1); *// 设置参数。根据占位符名称设置。* List<Student> studentList = query.list(); *// query.list()得到的是查询到的所有student对象。* **if** (studentList.size() <= 0) {  *// 若没有查到数据，则List大小为0* System.***out***.println(**"未查询到数据"**);  } **else** {  System.***out***.println(**"数据为："** + studentList);  }  session.getTransaction().commit();  session.close();  session.getSessionFactory().close(); *// 还要获得sessionFactory关闭，否则程序不结束* } } |

说明：

（1）createQuery()方法中也可直接传递HQL，而不传递相关类的字节码文件。这样的话，list()返回的数据类型就是不带泛型的，可以自己强制转换为带泛型的。但是不推荐这样做。

（2）即便查询结果只有一条或者查询不到结果，那也会封装在List中，只是长度为1或者0。

（3）HQL和SQL类似，也能用HQL进行增删改操作，比如使用HQL：“update Student set name = ? where id = ?”可进行更新。这时应该调用query.executeUpdate()方法来执行增删改操作。方法返回受影响行数。

### 2.2.2 使用本地SQL

Hibernate也支持使用原生SQL语句进行查询，这适用于复杂的查询。使用session.createNativeQuery()来进行SQL查询。例如查询指定的学生：

|  |
| --- |
| **public class** Demo {  **public static void** main(String[] args) {  *// 链式编程直接获得session。* Session session = **new** Configuration().configure().buildSessionFactory().openSession();  session.beginTransaction();  *// 获得有泛型的NativeQuery。* NativeQuery<Student> query = session.createNativeQuery(**"select \* from student where id = :id"**, Student.**class**);  query.setParameter(**"id"**, 1);  *// 获得学生集合* System.***out***.println(query.list());  session.getTransaction().commit();  session.close();  session.getSessionFactory().close();  } } |

上述的查询不太复杂，很容易将查询结果封装成实体对象。当查询较复杂，查询的数据并不好封装成实体时，那么我们就不用泛型，直接进行查询。这时，query.list()返回的数据还是一个List集合，只不过集合的每个元素都是一个Object数组。这时我们自行处理数据即可。每个Object数组就代表“一行”数据，多行数据组成了集合。例如（虽然这里的查询也不复杂，但我只想获得学生的姓名和入学时间）：

|  |
| --- |
| **public class** Demo {  **public static void** main(String[] args) {  *// 链式编程直接获得session。* Session session = **new** Configuration().configure().buildSessionFactory().openSession();  session.beginTransaction();  *// 获得有泛型的NativeQuery。* NativeQuery query = session.createNativeQuery(**"select name, entranceTime from student"**);  List<Object[]> list = query.list();  *// 遍历* **for** (**int** i = 0; i < list.size(); i++) {  System.***out***.println(**"姓名："** + list.get(i)[0]);  System.***out***.println(**"入学时间："** + list.get(i)[1]);  }  session.getTransaction().commit();  session.close();  session.getSessionFactory().close();  } } |

注意，如果只查询一列数据，比如“select name from student”，那么List集合中就不是Object数组了，而直接是Object对象，因为一列数据没必要用数组封装了，这是需要注意的。

同样，如果需要执行增删改操作，还是调用query.executeUpdate()方法。

# 3. Hibernate基础部分详解

## 3.1 主配置文件详解

主配置文件中主要配置：数据库连接信息、其他参数和映射信息。详细讲解的例子：

|  |
| --- |
| *<?***xml version='1.0' encoding='utf-8'***?>* **<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd"*>*** <**hibernate-configuration**>  *<!-- 一个session-factory节点代表一个数据库 -->* <**session-factory**>  *<!-- 数据库连接配置。每个属性的name值都有前缀hibernate，该前缀可省略 -->* <**property name="connection.driver\_class"**>com.microsoft.sqlserver.jdbc.SQLServerDriver</**property**>  <**property name="connection.url"**>jdbc:sqlserver://localhost:1433;databaseName=mytest;</**property**>  <**property name="connection.username"**>sa</**property**>  <**property name="connection.password"**>afan</**property**>  *<!--设置数据库方言，hibernate会根据具体的数据库来生成适应这个数据库的SQL语句 -->* <**property name="dialect"**>org.hibernate.dialect.SQLServerDialect</**property**>  <**property name="show\_sql"**>true</**property**>  <**property name="hibernate.format\_sql"**>true</**property**>  *<!-- 自动建表 -->* <**property name="hbm2ddl.auto"**>update</**property**>  *<!--  有关自动建表的说明：Hibernate能够根据映射文件自动在数据库中建立表，这就是自动建表。有以下选项：  create-drop：每次创建sessionFactory时创建表；当调用sesisonFactory的close方法时删除表。  create：每次都重新建表；如果表已经存在就先删除表及其数据再重新创建表。  update：如果表不存在就创建表；若表存在就不创建。  validate: 当映射文件与数据库表结构不一样时报错。(生成环境时可用于数据库验证)   -->  <!-- 加载映射文件 -->* <**mapping resource="com/zhang/test/Employee.hbm.xml"**></**mapping**>  </**session-factory**> </**hibernate-configuration**> |

## 3.2 映射文件详解

用例子详细讲解映射文件配置：

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"** *?>* **<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd"*>*** *<!--  auto-import：可省略，默认为tru。表示在写HQL语句时自动导入包名，若此项设置为false，在HQL语句中遇到类名还要加上包名  如：session.createQuery("from com.zhang.entity.Employee");  -->* <**hibernate-mapping package="com.zhang.test" auto-import="true"**>  *<!--  一个class节点映射一个实体(一般，一个对象就写一个映射文件，即一个映射文件只写一个class节点)  name 指定要映射的对象的类型  table 指定对应的表名；如果不指定表名table，默认与该实体类名称一样  -->* <**class name="Employee" table="employee"**>  <**id name="empId" column="id"**>  <**generator class="increment"**></**generator**>  </**id**>  *<!--  普通字段映射写在property节点中  name 指定对象的属性名称  column 指定对象属性对应的表字段名称，如果不写默认与对象属性一致。  length 指定该字段数据的长度, 默认为255。在自动建表时有用。  type 指定映射表的字段类型，如果不指定会匹配实体属性的类型，有两种方式指定类型  1.写java类型：必须写全名，比如写java.lang.String  2.写hibernate类型：直接写类型，都是小写，比如int  一般采用第一种方式，或者直接默认不指定类型。   如果列名称为数据库关键字，需要用反引号[Tab键上面的键]或改列名。  -->* <**property name="empName" column="name" length="20" type="java.lang.String"**></**property**>  <**property name="empWorktime" column="worktime" type="java.util.Date"**></**property**>  </**class**> </**hibernate-mapping**> |

## 3.3 主键生成策略

主键生成策略用于指定Hibernate以什么样的方式处理主键字段。在Hibernate中，常用以下几种方式（配置在映射文件的id标签中的generator标签的class属性）。

（1）increment。之前我们的主键生成策略就是用的increment，应用该策略表示主键从1开始自增。如果使用increment，每次向表中添加数据时，Hibernate会先执行如下语句（显示执行的SQL则会看见）：

|  |
| --- |
| select max(主键字段) from 表 |

也就是说，Hibernate向表中添加数据时先查询出表中主键的最大值，然后在最大值基础上加1设置主键保存到数据库中，保证表中数据主键是自增的。因此使用increment时要求主键类型是数字类型。

可以看出，每增加一条记录，Hibernate会执行两条SQL（先查询主键最大值再保存数据），效率较低。

（2）assigned：指定主键生成策略为手动指定主键的值。这时保存对象数据时，需要设置对象的主键，否则报错，因此该设置要求自己指定主键的值。

（3）identity：指定主键生成策略为自增长。与increment不同的是，identity是利用数据库进行自增长，即要求在数据库中设置主键为自增长，如果不设置的话，则保存数据时程序出错。identity适用于mysql和sqlserver等，是直接支持自增长的。这也要求主键是数值类型的。

（4）sequence：这是适用于Oracle数据库的自增长。因为Oracle中没有自增，只有序列（sequence），了解。

（5）uuid：指定表的主键是随机生成的UUID序列。保存数据时，不需要手动设置主键，由Hibernate自动设置UUID主键。这种使用UUID的方式是比较常用的。注意的是，UUID是32位字符串，因此主键的长度length不能小于该值。

（6）native：指示Hibernate自动根据数据库类型选择合适的自增长方法，一般不使用，而是自己具体指定。

（7）foreign：外键方式，后面讲。

## 3.4 关于SessionFactory

SessionFactory是一个重量级别的类，其中存放了主配置文件、映射文件和持久化类的信息。SessionFactory是单例的，一般，SessionFactory对象在应用启动时就会根据配置信息创建完毕（只创建一次），SessionFactory中的信息提供给程序使用以便操作数据库。Hibernate文档中说SessionFactory的属性是不可修改的（修改也不起作用），这保证了信息的可靠性，而且修改SessionFactory事实上是没有必要的。

SessionFactory中的信息是共享的，但由于其中的信息只在应用启动时初始化一次，且不能被修改，因此不存在线程安全问题，SessionFactory本身就是线程安全的。

## 3.5 关于Session和Hibernate的事务处理

Hibernate中的一个Session会话相当于是一次数据库的连接，Hibernate中对数据库的操作都是通过Session对象来完成的。（Hibernate的Session对象和Web中的Session没有一点关系，一个可以理解为数据库连接的会话，一个是浏览器客户端与服务器的会话）

Hibernate中的事务默认是手动提交的（总之Hibernate是通过设置connection.setAutoCommit(false)来实现的），因此对于非查询操作，需要控制事务，否则对数据库操作不生效。

控制事务的前提是在同一个连接中，因此Session中就提供了beginTransaction等方法来控制本会话的事务。

## 3.6 Hibernate的执行流程

Hibernate开发的一般步骤如下：

（1）使Hibernate加载主配置文件，继而映射文件等信息都会被加载；

（2）获取SessionFactory，继而获得Session对象

（3）通过Session对象操作数据库。对于非查询操作，注意要控制事务。

（4）关闭Session。

Hibernate的内部执行原理是怎样的呢？例如为何调用session.save(student)方法就能执行插入SQL操作数据库呢？

很简单，通过student所在持久类的映射文件信息，Hibernate能知道操作的是哪张表，又根据映射文件中类属性和表字段的对应关系，Hibernate能知道操作的是哪些字段，而要插入的数据都在student对象中，因此插入数据的必要信息都有了，最终Hibernate通过拼接SQL可实现执行SQL。一句话，这些信息都是我们提供的，Hibernate自然能通过这些信息操作数据库（利用反射等）。

# 4. 对象的状态

在学习Hibernate缓存之前，先研究Hibernate中对象的状态。

先通过下面保存数据的主要代码研究对象的状态：

|  |
| --- |
| 1 Student student = **new** Student(); 2 student.setName(**"Tom"**); 3 session.save(student); 4 transaction.commit(); 5 session.close(); |

（1）当程序执行完第1、2行时，只是在内存中创建了一个对象，此时和Hibernate是没有关系的。

（2）当执行完第3行代码，student对象将被Hibernate所管理，但此时数据还没有被保存到数据库中（可用断点验证）。

（3）第4行代码提交事务，此行执行完毕后数据会被保存到数据库中。

（4）第5行代码关闭session，即断开与数据库的会话连接。

在Hibernate中，把第1、2行时对象的状态称为临时状态；把第3、4行时对象的状态称为持久化状态；把第5行时对象的状态称为托管状态（脱离了Hibernate的管理）。

下面讲讲具体的方法对对象的状态产生的影响。

（1）save方法：可以把一个对象从临时状态转变为持久化状态。

（2）get方法：从数据库中根据主键提取出一个对象，获得的该对象就是一个持久化状态的对象。

需要说明的是，一般，一个持久化状态的对象在只要进行了事务的提交，就能保存到数据库中。例如更新对象数据时，之前的做法是先用get方法得到对象数据，然后更新对象，最后调用update方法保存到数据库中。但实际上，update方法是不需要调用的，只要修改对象进行提交事务即可，例如：

|  |
| --- |
| Session session = **sessionFactory**.openSession(); Transaction transaction = session.beginTransaction(); Student student = session.get(Student.**class**, 1); *// 获得主键为1的学生对象* student.setName(**"Jack"**); *// 更新对象数据* transaction.commit(); *// 直接提交事务即可更新对象* session.close(); |

上述代码就可实现对象数据的更改保存到数据库中。为什么不调用update方法也可以呢？因为调用get方法拿到的对象已经是持久化状态的对象了，对于持久化对象，提交事务即可将变更保存到数据库中。其实update方法的作用就是把一个对象变成持久化状态，因此这里就没有必要再调用update方法了。

（3）update方法：把一个对象变成持久化状态。

（4）evict方法：把一个对象从持久化状态转变为脱管状态。如果在事务提交之前执行evict，那么将不会对数据库产生影响，因为对象已经不再是Hibernate管理了。如果想转变成持久化对象，只要再调用update方法即可。

（5）clear方法：把所有的hibernate中的持久化对象都转换成脱管状态的对象。注意这里转换的是所有的持久化对象。

# 5. 副本的概念

例如，数据库中有学生信息为：

|  |  |  |
| --- | --- | --- |
| id | name | age |
| 1 | Tom | 12 |

用get方法获取后，“更改”了对象信息（实际数据没有更改），再进行事务提交：

|  |
| --- |
| Session session = **sessionFactory**.openSession(); Transaction transaction = session.beginTransaction(); Student student = session.get(Student.**class**, 1); *// 获得主键为1的学生对象* student.setName(**"Tom"**); *// 更新对象数据。实际上还是原来的数据，并没有真正更新* transaction.commit(); *// 提交事务* session.close(); |

Hibernate会向数据库发送更新SQL进行操作吗？实践说明，Hibernate并没有向数据库发送更新语句。

也就是说，Hibernate会自动感知对象的数据是否变化了，如果变化了，就发出更新语句，否则什么也不做。这样就提高了效率，因为没有变化的数据是没有必要耗费连接资源再去数据库更新的。

Hibernate是怎么做到这一点的呢？其实很简单，当get等查询方法获得对象时，会产生这个持久化对象的一个副本（产生的方式是克隆，而不是对象的引用，因为引用的话还是同一个对象），最后更新数据时，Hibernate会比对持久化对象和副本对象，如果持久化对象改变了则执行更新语句。

以后使用别的数据库缓存框架时，也可以考究一下它有没有类似的设计，以此评判。

# 6. flush方法

在保存或者更新对象时，SQL语句的发送是在提交事务时完成的。其实SQL的发送是在调用Session对象的flush方法时完成的。如果我们在提交事务前手动调用flush方法，则执行完flush方法后，Hibernate就会执行在该事务中的所有保存或更新操作，例如：

|  |
| --- |
| Session session = **sessionFactory**.openSession(); Transaction transaction = session.beginTransaction(); *// 1. 对象1，修改* Student student = session.get(Student.**class**, 1); *// 获得主键为1的学生对象* student.setName(**"Tom"**); *// 修改对象 // 2. 对象2，保存* Student student1 = **new** Student(); student1.setName(**"Jack"**); session.save(student1); *// 调用flush方法，会发送SQL* session.flush(); transaction.commit(); *// 提交事务* session.close(); |

如果不手动调用flush方法，在transaction.commit()方法中也会自动判断之前的代码有没有调用过flush方法，如果没有调用过flush方法，则会自动调用flush方法。

flush方法会完成如下工作：

（1）flush方法会检查所有的持久化对象；

（2）如果持久化对象是由临时状态转变过来的，则会发送insert语句保存持久化对象数据；

（3）如果持久化对象是由get等方法得到的，则先把持久化对象和副本对象进行比对，如果一致，则什么都不做，如果不一致，则执行update语句更新对象数据。

flush方法就用于处理对象的更新或保存，从而决定是发送update还是insert语句。