Hibernate3 Day1

讲师：BoBo老师

课程安排:4天(框架的使用+理论知识)

1. 第一天Hibernate快速入门(单表的CRUD)+核心api讲解(配置+接口)
2. 第二天:Hibernate一级缓存,对象持久化状态,多表映射操作
3. 第三天Hibernate各种查询的使用（三种方法）,抓取策略优化（lazy加载等）
4. 第四天Hibernate二级缓存\查询缓存,jpa注解的使用

内容安排：

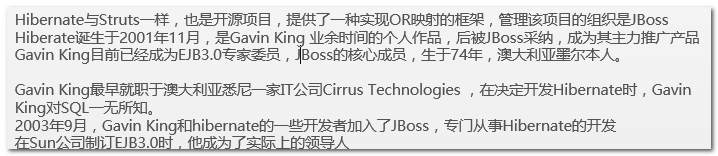
1. Hibernate概述、有什么优缺点等
2. Hibernate的快速入门：环境搭建、CRUD（增删改查）
3. ~~日志的使用。(LOG4J)~~
4. Hibernate核心配置常用属性详解
5. Hibernate核心API常用操作详解
6. 持久化对象的PO\OID的说明
7. hbm映射文件配置常用参数详解

学习目标：

快速入门，可应用Hibernate的常见的CRUD。

# Hibernate概述

## Hibernate的来历



关键词：开源框架，属于jboss公司产品，时间久远。可以不写SQL就可以操作数据库。

## 什么是Hibernate

|  |
| --- |
| Hibernate是**轻量级**JavaEE应用的**持久层**解决方案，是一个**关系数据库ORM**框架。 |

* 轻量级： 使用方便 （比Apache DbUtils 复杂很多倍 ）这个概念是个相对概念。（主要是对比EJB来说的，ejb是重量级的（内建组件很多，启动很慢，东西很多））
* 持久层： JavaEE分为 表现层、业务层(service)、持久层(dao)

Struts2 是一个表现层 框架

Hibernate 是一个持久层 框架 （操作数据库 ）--ssh

* 关系数据库： mysql、oracle 都是关系型数据库 （主流数据库 ）

最近 NOSQL 非关系型数据库开始流行 （mongodb、 redis）

* ORM ： Object Relational Mapping 对象关系映射 （一种思想）

将数据表的关系，映射为 类和对象之间关系:

|  |  |
| --- | --- |
| 关系型数据库 | Java |
| 表 | 类 |
| 表的字段 | 类的属性 |
| 表的一行数据 | 类的一个实例对象 |

ORM好处： java程序员只需要在程序中操作对象，因为存在映射关系，自动生成SQL语句。不需要你写sql语句，就可以操作数据库。

Java中常见的ORM实现（持久层框架）：

* JPA（ Java Persistence API）.Sun公司的规范，JPA通过JDK 5.0注解或XML描述对象－关系表的映射关系（只有接口规范）
* Hibernate 最流行ORM框架，通过对象-关系映射配置，可以完全脱离底层SQL
* MyBatis 本是apache的一个开源项目 iBatis，支持普通 SQL查询，存储过程和高级映射的优秀持久层框架
* 其他的持久层框架（非ORM）：Apache DBUtils 、Spring JDBCTemplate

【提示】

Jpa是一种规范，而Hibernate是它的一种实现，使用Jpa的一个好处是，可以更换实现而不必改动太多代码。Hibernate框架是基于JPA接口的实现，但又对其做了一些补充，因此，可以理解为Hibernate中有两套规范（JPA和自己的，事实上只是补充）。

## Hibernate的优缺点

优点：

|  |
| --- |
| * Hibernate完全是ORM，自动生成sql语句，极大简化jdbc的操作。 * 具有很强的反射和代理机制，生成代理对象，（可以代码增强）在数据获取可以进行延迟抓取、缓存的使用、达到优化的目的。 |

缺点：

|  |
| --- |
| * hibernate完全orm，自动生成sql，所以很多企业用户对hibernate性能存在质疑（任何持久层框架都不如jdbc速度快，开发效率和性能的平衡）。在hibernate框架中，提供了很多优化手段。 |

# hibernate的快速入门

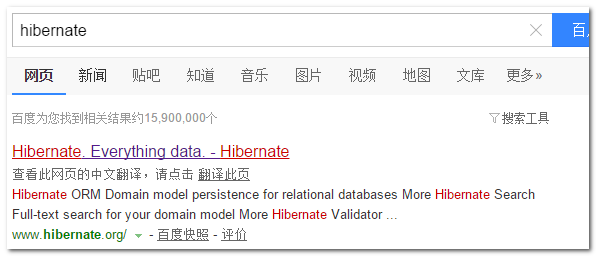
开发工具：MyEclipse 10.7(自带很多插件)

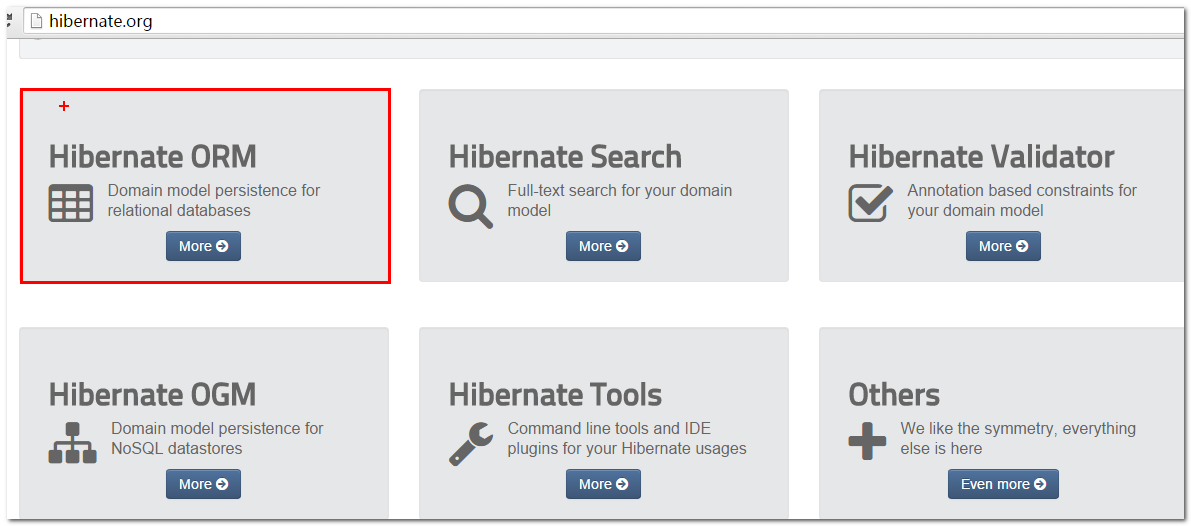
数据库层面：mysql5

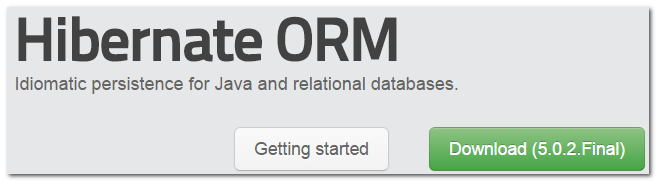
## hibernate的jar包下载和导入

### 开发包的下载：

百度搜索：







开源协议。



本次课程使用3.x系列的经典版本3.6.10

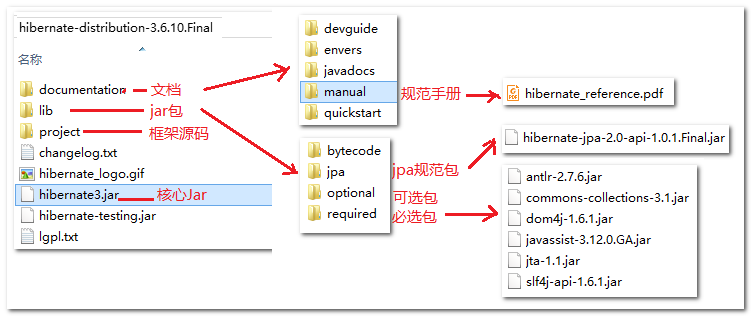


【提示】

框架的版本并不是越高越好，适合的最好；

不同的版本对jdk环境、不同框架的整合方面都有一定要求。

Hibernate开发包内容：



另外：Hibernate开发包中只是提供了一个日志接口的包slf4j，并没有提供日志的具体实现，因此，需要额外下载slf4j和log4j的开发包。



### Jar包的导入（开发环境的搭建）

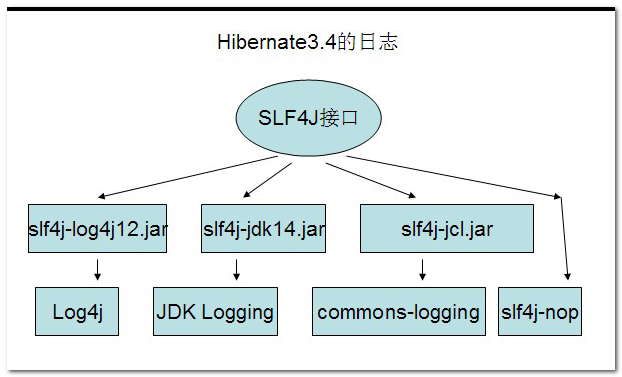
新建web工程hibernate\_day1

导入jar包到工程：共11个

* 核心jar：hibernate3.jar
* 必须jar：\lib\required\\*
* jpa规范jar：\lib\jpa---后面会用到jpa的注解。
* jdbc驱动包：mysql-connector-java-5.0.8-bin.jar—连接数据库
* 使用日志系统-slf4j系列（slf4j核心+ slf4j和log4j的整合包+log4j）—使用静态日志绑定体系+在src中添加log4j的核心配置文件：log4j.properties

【什么是slf4j】

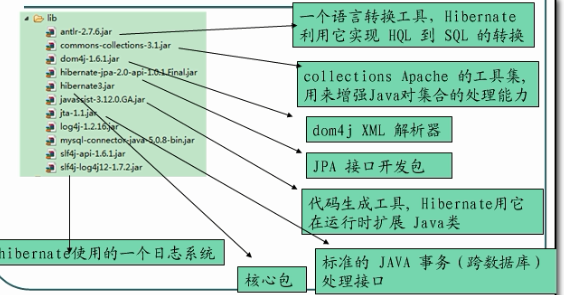




slf4j-api只是接口包（解耦合的思想，静态绑定思想），需要日志系统的实现log4j

* slf4j整合log4j的jar：slf4j-log4j12-1.7.2.jar
* log4j的开发jar：log4j-1.2.16.jar

【导入完成的包】共11个



## log4j的日志框架的使用(可选内容)

log4j是一个apache的日志的实现框架，企业开发中经常使用log4j记录日志。

问题：为什么要用log4j

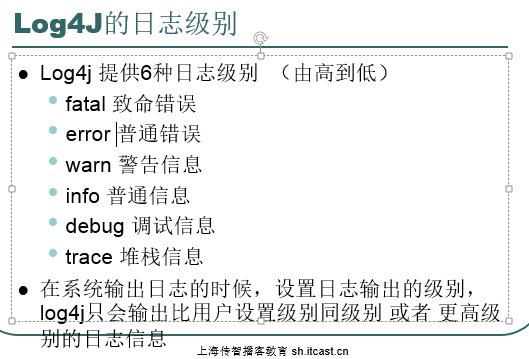
开发过程中，经常需要想控制台输出一些调试信息，传统的做法是system.out打印。

缺点：信息一定会输出在控制台中，可以将输出信息写入日志文件，调试信息会造成日志文件过于庞大。

日志技术的好处：存在日志级别的概念，通过配置（日志级别）来控制在不同的情况，输出不同级别的日志信息。--可控制输出的

log4j的日志级别：

提供6种日志级别（由高到低）

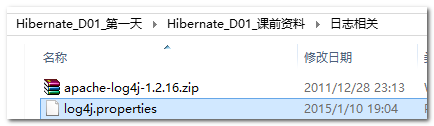


配置log4j有两种配置方式:

在classpath（src/）中下面配置

* log4j.xml配置方式
* log4j.properties配置方式.—推荐:键值对.

参考:



讲解这个文件的细节:



1、 记录器 Loggers—log4j大管家

语法： 记录器 = 日志级别, 输出源1, 输出源2 … （一个日志记录器配置多个输出源）

默认记录 log4j.rootLogger 根记录器

2、 输出源 Appender （将日志输出到哪里）

语法： log4j.appender.输出源名称 = 实现类全名

log4j.appender.输出源名称.属性 = 值



常用 ConsoleAppender 输出到控制台、 FileAppender 输出到文件

3、 输出源的属性配置：布局layout 日志信息格式

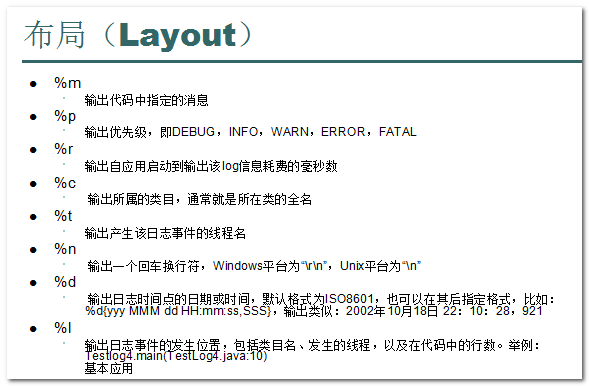


常用 PatternLayout 自定义模式字符串 输出日志

语法 ： log4j.appender.输出源名称.layout = 布局实现类全名

如果使用 自定义布局，设置 log4j.appender.A1.layout.ConversionPattern 属性

参考:



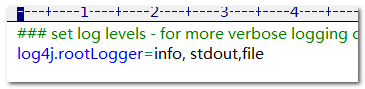


|  |
| --- |
| //用来测试日志的  **public** **class** Log4jTest {  //在class中定义日志记录器  //这个日志记录器会记录当前类的所有日志情况,没有线程问题,所以一个类就使用一个实例就行了  **private** **static** **final** Logger *LOG* = Logger.*getLogger*(Log4jTest.**class**);    @Test  **public** **void** testLog4j(){  //演示不同日志级别的使用效果  //6种级别:分别使用6种级别的方法，来写日志  *LOG*.fatal("致命严重错误级别的日志");  *LOG*.error("普通错误日志");  *LOG*.warn("警告错误日志");  *LOG*.info("普通信息日志");  *LOG*.debug("调试级别日志");  *LOG*.trace("堆栈调试信息");  // System.out.println("----systemout的输出");  //以后我们开发过程中的调试，都推荐使用log.debug();      }  } |

小结：

log4j的使用方法：我们在开发的时候,就可以选择不同的方法,来控制日志的输出.完全就不需要更改代码,只需要更改配置就行了.

如果你要写到文件中则加上输出源就行了:



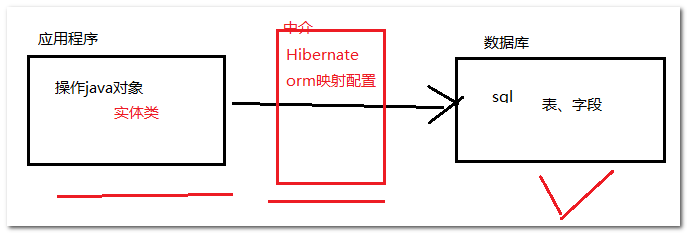
## 基于hibernate实现数据库表CRUD的操作.

### 开发准备和配置

第一步:准备数据库的表:

|  |
| --- |
| CREATE TABLE `t\_customer` (  `id` INT(11) NOT NULL AUTO\_INCREMENT,  `name` VARCHAR(20) DEFAULT NULL,  `age` INT(11) DEFAULT NULL,  `city` VARCHAR(30) DEFAULT NULL,  PRIMARY KEY (`id`)  ) ENGINE=INNODB DEFAULT CHARSET=utf8 |

第二步:编写实体类和ORM映射文件



【实体类和映射的编写前需要知道的】

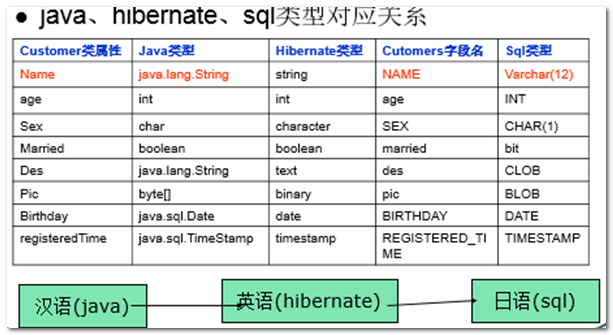
作为ORM框架，

Java和数据库之间的对应关系：类（POJO）--表, 属性--字段，对象--记录,

类就是普通的POJO实体类，必须提供getter和setter方法。

数据类型之间的对应关系：Java实体类、Hibernate、sql之间有数据类型的对应

对应关系:



【编写customer实体类】：

Customer.java

|  |
| --- |
|  |

【编写hbm（hibernate mapping ）映射】:

|  |
| --- |
| 习惯上,在实体类所在目录,创建类名.hbm.xml的数据表映射配置文件.—作用，让实体和数据库的具体字段和表进行对应映射。 |

|  |
| --- |
| 引入dtd的头信息:        <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  配置本地提示:    dtd文件的位置： |

Customer.hbm.xml

|  |
| --- |
| <?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">  <!-- hibernate-mapping标签用来编写java类和数据库表之间的对应关系 -->  <hibernate-mapping>  <!--class 类和表之间的关系  name:类的名字，一般是全类名  table:表的名字  -->  <class name=*"cn.itcast.hibernate.a\_quickstart.Customer"* table=*"t\_customer"*>  <!-- 属性和字段的对应关系 -->  <!-- 特殊的属性：id:用来配置主键的对应关系  name:类中的属性的名字  column:是数据库字段的名字，可以和属性一样，也可以不一样  -->  <id name=*"id"* column=*"id"*>  <!-- 主键策略：和数据库对应  class:主键策略配置，内置很多策略，mysql的自增长就对应identity  -->  <generator class=*"identity"*/>  </id>  <!-- 配置其他一般的属性  name:属性的名字  column:字段名  -->  <property name=*"name"* column=*"name"*/>  <!-- 如果字段名和属性名一样，则，可以省略column属性 -->  <property name=*"age"*/>  <property name=*"city"*/>  </class>  </hibernate-mapping> |

第三步:编写hibernate核心配置.—jdbc连接数据库\hbm映射管理等等

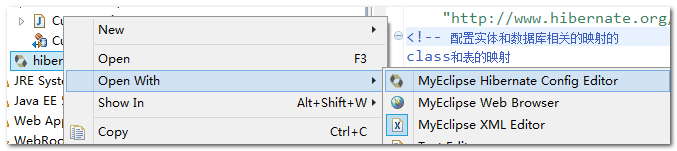
习惯上在src下创建hibernate.cfg.xml(必须叫这个名字)

|  |
| --- |
| 引入头部信息： |

|  |
| --- |
| <?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">  <!-- 在这里配置Hibernate的核心的一些配置，比如jdbc、映射文件加载相关的 -->  <hibernate-configuration>  <!-- 所有的配置都在会话工厂中配置，会话工厂缓存了这些配置 -->  <session-factory>  <!-- jdbc相关的东东 :4个东西-->  <property name=*"hibernate.connection.driver\_class"*>com.mysql.jdbc.Driver</property>  <property name=*"hibernate.connection.url"*>jdbc:mysql:///itcast24</property>  <property name=*"hibernate.connection.username"*>root</property>  <property name=*"hibernate.connection.password"*></property>  <!-- 配置方言: 作用不同的数据库的sql是不同，自动发sql会根据方言来发sql-->  <property name=*"hibernate.dialect"*>org.hibernate.dialect.MySQL5Dialect</property>  <!-- 配置其他属性 -->  <!-- 自动建表 -->  <property name=*"hibernate.hbm2ddl.auto"*>update</property>  <!-- 打印sql -->  <property name=*"hibernate.show\_sql"*>true</property>  <!-- 格式化sql -->  <property name=*"hibernate.format\_sql"*>true</property>    <!-- 引入hbm配置文件 -->  <!-- 注意：包中间的分隔符是点，但资源文件的分隔符是斜杠 -->  <mapping resource=*"cn/itcast/hibernate/a\_quickstart/Customer.hbm.xml"*/>    </session-factory>  </hibernate-configuration> |

【提示】

建议打开核心配置文件使用的编辑器，会有更多提示：



第四步:java编程实现CRUD，基本示例如下：



### 保存（插入）数据

保存数据,调用save方法

|  |
| --- |
| //插入保存数据  @Test  **public** **void** testSave(){    //手动new一个实体类对象  Customer customer = **new** Customer();  // customer.setId(id)//id是自增长的不需要赋值  customer.setName("Rose");  customer.setAge(18);  customer.setCity("上海");    //得到配置对象:加载核心配置文件hibernate.cfg.xml,,--可以认为制造工厂的材料  Configuration configuration = **new** Configuration().configure();    //创建会话工厂(类似于连接池)  SessionFactory sessionFactory = configuration.buildSessionFactory();    //通过会话工厂得到会话session(该会话和servlet中的会话没关系)（类似于Connection连接）  Session session=sessionFactory.openSession();    //开启事务  Transaction transaction = session.beginTransaction();  //操作数据库  //保存数据，不需要写sql，只需要操作实体类对象  session.save(customer);    //提交事务  transaction.commit();    //释放资源  session.close();  sessionFactory.close();  }    只要你在控制台看到下面这句话，没有报异常的话，就知道数据已经插入数据库： |

### 修改（更新）数据,

更新数据，通过session的update方法

|  |
| --- |
| //更新操作  @Test  **public** **void** testUpdate(){  //配置对象  Configuration configuration = **new** Configuration().configure();  //会话工厂  SessionFactory sessionFactory = configuration.buildSessionFactory();  //会话  Session session = sessionFactory.openSession();  //开启事务  Transaction transaction = session.beginTransaction();  //数据库操作    Customer customer = **new** Customer();  customer.setId(1);//更新的时候必须给id赋值  customer.setName("Jack");  customer.setAge(88);  customer.setCity("北京");    //更新  //更操作是根据id属性来更新!!!  session.update(customer);    //提交事务  transaction.commit();  //释放资源  session.close();  sessionFactory.close();    }  控制台：    说明操作了数据库了。 |
| 【补充】:注意update默认会更新所有字段.    city就会置空！  注意：在使用update的时候，一定要保证属性都有值。没值的会被置空。 |

### 删除数据,

删除数据，通过session的delete方法(具有id的对象)

|  |
| --- |
| //删除  @Test  **public** **void** testDelete(){  //配置对象  Configuration configuration = **new** Configuration().configure();  //会话工厂  SessionFactory sessionFactory = configuration.buildSessionFactory();  //会话  Session session = sessionFactory.openSession();  //开启事务  Transaction transaction = session.beginTransaction();  //数据库操作  Customer customer = **new** Customer();  customer.setId(1);//删除操作必须给id属性值，否则异常  //其他属性可有可无  customer.setName("Rose");//删除时除id属性外，其他属性都会被忽略。  //删除（对象）  //删除方法底层根据id来删除数据。  session.delete(customer);    //提交事务  transaction.commit();  //释放资源  session.close();  sessionFactory.close();  } |
|  |

【注意】：删除是根据主键删除的,与其他字段没关系

【提示】：session包的引入：

要引入：

不要引入：

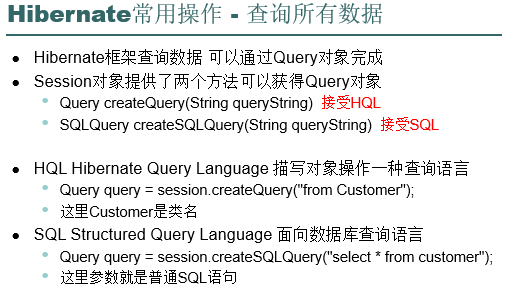
### 通过主键来查询数据：

根据主键查找数据,通过session的get或load

|  |
| --- |
| //查询：根据id来查询  //根据主键查找数据,通过session的get或load  @Test  **public** **void** testQueryById(){  //配置对象  Configuration configuration = **new** Configuration().configure();  //会话工厂  SessionFactory sessionFactory = configuration.buildSessionFactory();  //会话  Session session = sessionFactory.openSession();  //开启事务  Transaction transaction = session.beginTransaction();  //数据库操作  //根据id属性（主键）查询  //参数1：实体类的class：自动去查找该类对应的表的数据,还有个作用，将查询的结果自动封装到实体类中  //参数2：主键属性的值  // Customer customer = (Customer) session.get(Customer.class, 2);//如果数据不存在，会返回null  Customer customer = (Customer) session.load(Customer.**class**, 3);//如果数据不存在，load会报错  System.*out*.println("customer:"+customer);    //提交事务  transaction.commit();  //释放资源  session.close();  sessionFactory.close();  } |

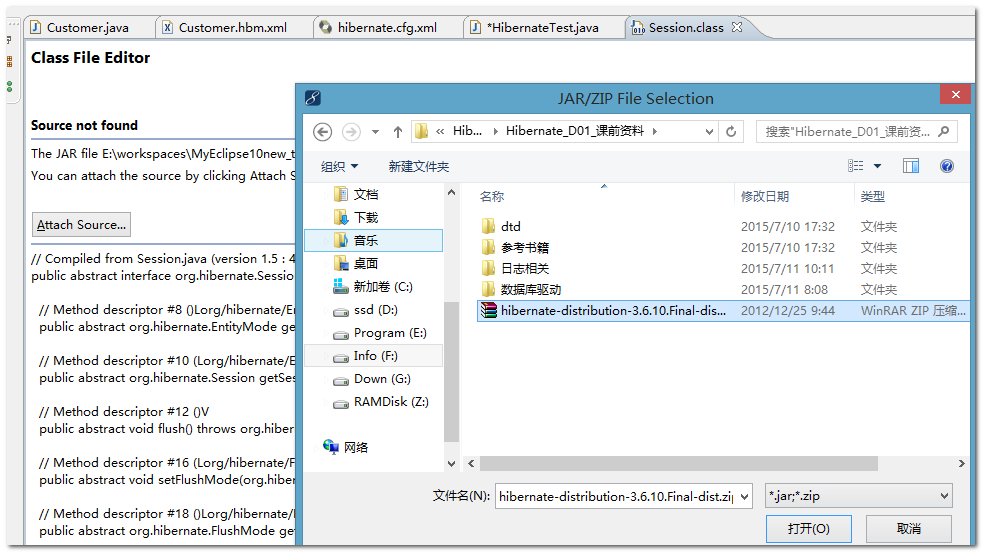
### 查询所有数据

使用hibernate提供的Query接口来完成,使用HQL(hibernate Query language)语句完成





【提示】为了方便学习，建议大家关联源码，关联方式：



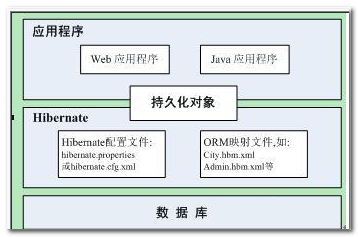
|  |
| --- |
| //测试查询所有数据对象  @Test  **public** **void** testQueryAll(){  //获取配置对象，用来加载核心配置文件hibernate.cfg.xml--材料  Configuration configuration = **new** Configuration().configure();  //2.使用配置对象获取会话工厂,用来生产会话，有点像连接池  SessionFactory sessionFactory = configuration.buildSessionFactory();  //3.通过工厂来获取一个会话,（和servlet没有半毛钱关系）作用：用来操作数据库对应的对象的,有点像connection  Session session = sessionFactory.openSession();  //4.开启事务（可以省略）  Transaction transaction = session.beginTransaction();  //操作数据库（通过操作对象来操作）  //一。Query对象的  String hql="from Customer";//Hibernate查询语句，该语句的语法类似于sql:select \* from t\_customer  //hql是面向对象的，这里的Customer是类名，不是表名！，自动生成sql的  Query query = session.createQuery(hql);  //查询列表操作  //Hibernate会自动将结果封装为实体类中  List<Customer> list = query.list();  System.*out*.println(list);    //二。SqlQuery接口对象，是Query接口的子接口,可以接受sql语句  String sql="select \* from t\_customer";  SQLQuery sqlQuery = session.createSQLQuery(sql);  //查询列表操作  //默认情况下，不会自动将结果封装回实体中，装到了Object[]  //如果想自动装回实体类，必须绑定一个实体class  List list2 = sqlQuery.addEntity(Customer.**class**).list();  System.*out*.println(list2);    //提交事务  transaction.commit();  //释放资源  session.close();  sessionFactory.close();  } |

## hibernate的基本运行流程分析（图解）



# hibernate核心配置

## Hibernate的基本体系结构



通过上面的体系结构图得知：应用程序通过持久化对象（po）间接的完成对数据库的操作（crud）。

持久化对象（persistent object），简写为PO,这里就是Customer这个实体类

## Hibernate核心配置文件分类

Hibernate框架支持properties和xml两种方式的Hibernate属性的配置，对应的两种核心配置文件为：

* hibernate.properties 配置常用的属性,必须手动编程加载hbm文件或持久化类。（了解）
* hibernate.cfg.xml配置常用的属性,配置加载hbm映射,配置缓存策略等。（推荐）

[提示]配置Hibernate的属性很多，可以参考hibernate解压包的project/etc/hibernate.properties。

## hibernate.properties配置示例（了解）

目的：通过properties格式配置最小化的hibernate属性(连接和方言)

【配置示例】

hibernate.properties:

|  |
| --- |
| hibernate.dialect org.hibernate.dialect.Oracle10gDialect  hibernate.connection.driver\_class oracle.jdbc.driver.OracleDriver  hibernate.connection.username hb15  hibernate.connection.password hb15  hibernate.connection.url jdbc:oracle:thin:@localhost:1521:orcl |

方言：用来生成对应数据库的sql的。

mysql：官方默认org.hibernate.dialect.MySQLDialect，我们现在推荐使用org.hibernate.dialect.MySQL5Dialect

提示：上述配置是Hibernate的最小化配置。

【示例测试】

建立包:cn.itcast.hibernate.b\_coreapi

|  |
| --- |
| //测试properties核心配置的使用  //业务：根据id查询  @Test  **public** **void** testCoreConfigProperties(){  //配置对象:new Configuration()默认会加载hibernate.properties文件  Configuration configuration = **new** Configuration();  //手动编码加载hbm映射  configuration.addResource("cn/itcast/hb/a\_quickstart/Customer.hbm.xml");  //会话工厂  SessionFactory sessionFactory = configuration.buildSessionFactory();  //会话  Session session = sessionFactory.openSession();  //开启事务  Transaction transaction = session.beginTransaction();  //数据库操作  //根据id查询  Customer customer =(Customer) session.get(Customer.**class**, 3);  System.*out*.println(customer);    //提交事务  transaction.commit();  //释放资源  session.close();  sessionFactory.close();  } |

【提示】

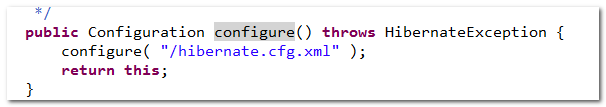
properties方式，主要缺点：不能配置加载hbm映射文件，注解可以使用，功能较弱，xml方式可以完全替代properties方式，因此后面都使用xml的方式进行配置和使用。

【思考】

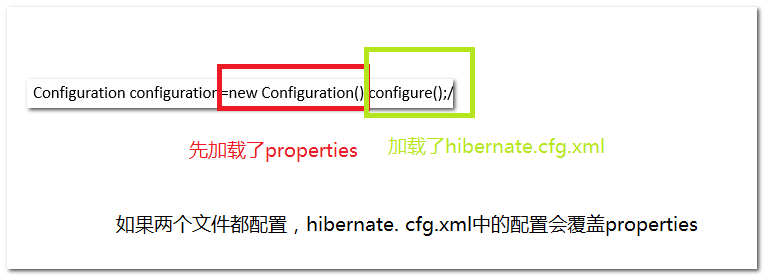
如果两个配置文件都存在，hibernate会加载使用哪个？

new Configuration();只加载properties文件

Configuration configuration = new Configuration().configure();会先加载properties文件，再加载cfg.xml文件。



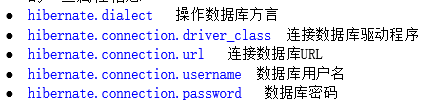
后加载的会覆盖先加载的属性的值



## hibernate.cfg.xml配置示例+框架常用属性

### 数据库连接参数和数据库方言

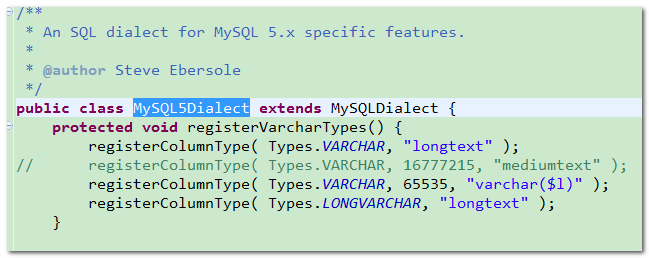
connection连接参数、数据库方言的配置



方言是用来将实体类的映射转换为不同数据库的sql的一种配置策略.

操作hibernate必须配置的5个属性—最小化配置

方言：



【了解】

在配置hibernate.cfg.xml时,hibernate属性:hibernate.dialect中的“hibernate”可以省略.

在配置hibernate.properties时, hibernate前缀不能省略,但用cfg.xml可以省略.

### 连接池

* Hibernate默认连接池(了解):

默认情况下，我们有没有用连接池呢?

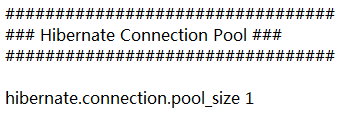
查看日志发现Hibernate使用了一个默认的连接池：



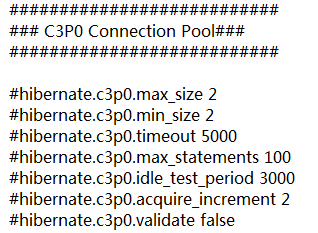
【提示】

不建议在生产环境使用。

默认的连接池大小配置（了解）：

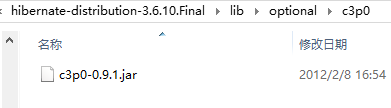


* c3p0连接池的配置：---生产环境



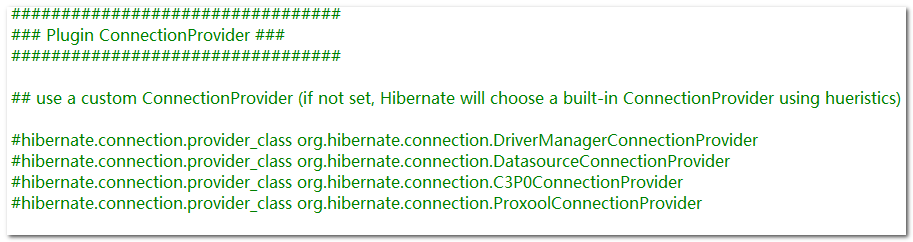
更换C3P0的连接池：

1. 解压目录的lib中



导入c3p0的jar

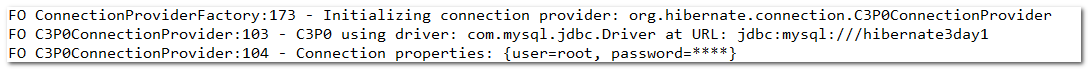
1. 配置更改连接池提供者（插件机制，可以随时换，可插拔）



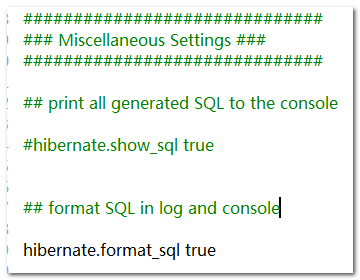
切换到c3p0连接池:



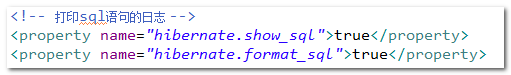
再次执行查看结果：

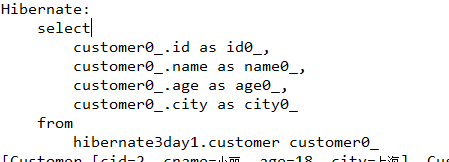


### 控制台打印sql



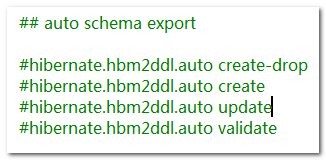
默认是false，建议在测试环境下，都将sql打开，便于调试。





### 自动建表

该配置是让hibernate是否自动建立相应的表。也就是说，你数据库可以没有表，hibernate可以自动给你建立一个。



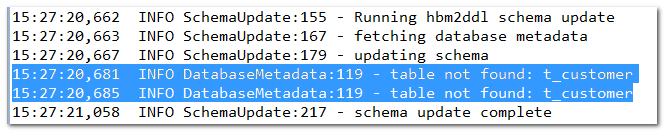
* create-drop:在程序运行时,（创建session工厂的时候）会自动建表,在程序停止时(关闭sessionFactory时),表删除

测试前先删除数据库的表！

编写建表的代码：

|  |
| --- |
| //使用Hibernate来建表的方法：只需要得到session工厂即可。  @Test  **public** **void** createTable(){  Configuration configuration = **new** Configuration().configure();  SessionFactory sessionFactory = configuration.buildSessionFactory();//该动作可以自动建表    sessionFactory.close();//释放资源  System.*out*.println("建表结束");  } |

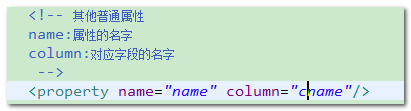
控制台：



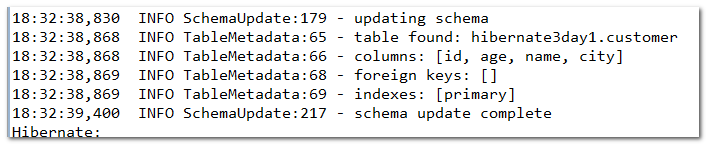
这个提示，就说明建表了。。

* create:在程序运行时,如果表不存在,自动建表,如果存在,则不进行任何操作(mysql会先删除再创建)
* update:在程序运行时,如果表不存在,自动建表,如果表存在,则检查表和类的结构是否一致,如果不一致,则更新表结果.(推荐)

update:







思考为什么？hibernate怎么知道你要将name改成cname

如果数据库没有的字段，则自动增加

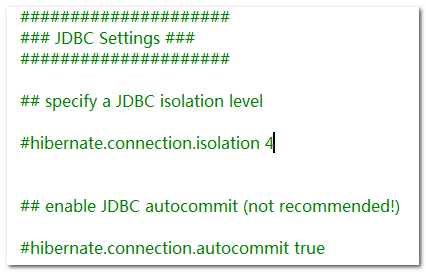
提示：数据库中字段可以有冗余，但冗余的字段无法被Hibernate管理（因为没有映射）

* validate:在程序运行时,如果表结构和类不一致,则报错!（表中没有相应的字段，则报错，表中有冗余字段，不报错）





### jdbc的其他相关设置

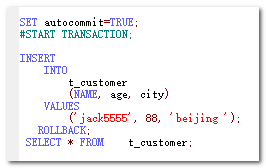


默认事务提交的设置是false，即默认不自动提交事务。（手动事务）

|  |
| --- |
| //测试自动提交属性  @Test  //Hibernate默认是手动事务  **public** **void** testAutoCommit(){  //配置对象:  Configuration configuration = **new** Configuration().configure();  //会话工厂  SessionFactory sessionFactory = configuration.buildSessionFactory();  //会话  Session session = sessionFactory.openSession();  //开启事务  // Transaction transaction = session.beginTransaction();  //数据库操作  //保存一个客户  Customer customer = **new** Customer();  customer.setName("Rose1");  customer.setAge(18);  customer.setCity("上海1");  //保存操作  session.save(customer);    //提交事务  // transaction.commit();  //释放资源  session.close();  sessionFactory.close();  } |

【扩展1】

没有提交的事务，数据库中这条记录是否存在？(在运行的过程中,曾经存在过（内存中，事务中）.如果没有提交,只是没有持久化数据而已)



【扩展2】

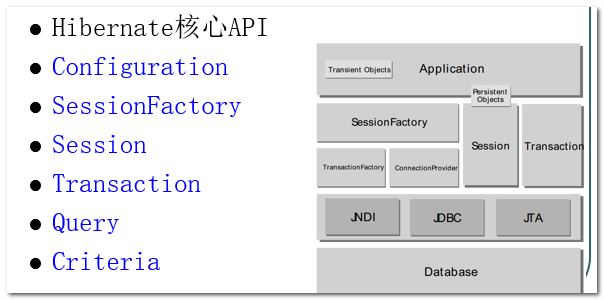
oracle默认是false,即使改成true也没效果！



原因：Oracle不支持自动提交事务,不能自动提交（Oracle数据库必须手动提交，即使你在hibernate中设置了自动提交，也没用。）,但msyql是可以自动提交的.

# Hibernate核心API

## 概述



通过该体系结构图，可以看到Hibernate的核心API接口，它们之间的关系：

Hibernate通过Configuration来读取核心配置文件，SessionFactory缓存配置并用来提供Session，而Session是用来操作PO的，让Hibernate生成crud的sql语句，在操作过程中使用Transaction来管理事务。Query和Criteria是Hibernate提供的查询的两种不同方式。

下面详解：

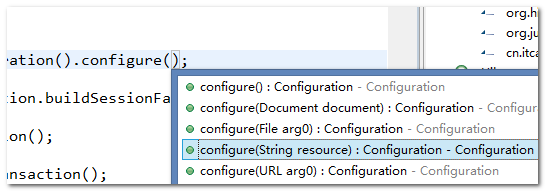
## Configuration配置对象

作用:加载Hibernate的相关配置(Hibernate.cfg.xml配置\Hibernate.properties配置[不支持配置映射])

第一种:加载Hibernate.cfg.xml

Configuration configuration=new Configuration().configure();//加载默认类路径的Hibernate.cfg.xml

也可以加载其他命名的配置文件new Configuration().configure(“文件路径”);



~~第二种:加载Hibernate.properties~~

Configuration configuration=new Configuration();//加载默认的Hibernate.properties

需要手动加载hbm映射

configuration.addResource(xxx.hbm.xml)

## SessionFactory会话工厂

作用:用来缓存一些配置和数据的

可以缓存:

* Hibernate常用属性(包括连接池)
* hbm类和数据表映射信息
* 预定义SQL语句(命名查询语句)
* 二级缓存

获取SessionFactory的方式： configuration.buildSessionFacotry()获取对象

【提示】

|  |
| --- |
| SessionFactory是线程安全的.即**当多个线程访问同一个SessionFactory是不存在线程问题的。**构造SessionFactory很消耗资源，一般情况下，一个应用（项目）只需要初始化一个，即操作一个全局的SessionFactory对象。 |

SessionFactory和Session的关系有点像连接池和连接的关系，我们操作数据库时主要使用的连接和session，连接池和SessionFactory只需要创建后从里面获取连接和session即可。

【练习示例】

抽取HibernateUtils来提供Session对象。

|  |
| --- |
| //Hibernate工具类：直接获取session  **public** **class** HibernateUtils {  //全局的session工厂  **private** **static** SessionFactory *sessionFactory*;    **static**{  //初始化session工厂  *sessionFactory*=**new** Configuration().configure().buildSessionFactory();  //关闭工厂：在虚拟机结束的时候关闭  //addShutdownHook可以在虚拟机关闭之前，做一些事情（释放资源）  Runtime.*getRuntime*().addShutdownHook(**new** Thread(){  //覆盖方法  @Override  **public** **void** run() {  //关闭session工厂代码  *sessionFactory*.close();  System.*out*.println("session工厂已经关闭！");  }    });    }    //获取工厂  **public** **static** SessionFactory getSessionFactory(){  **return** *sessionFactory*;  }    //获取session  **public** **static** Session openSession(){  **return** *sessionFactory*.openSession();  }      } |

使用工具类来进行操作数据（查询示例）:

|  |
| --- |
| //测试工具类  @Test  **public** **void** testUtils(){  // SessionFactory sessionFactory = HibernateUtils.getSessionFactory();  // Session session = sessionFactory.openSession();    Session session=HibernateUtils.*openSession*();  //操作数据库  session.beginTransaction();    //查询  Customer customer=(Customer)session.get(Customer.**class**, 2);  System.*out*.println(customer);    session.getTransaction().commit();  session.close();  // sessionFactory.close();  // HibernateUtils.getSessionFactory().close();  } |

## session会话

作用：Session对象相当于JDBC的Connection，主要用于操作数据的（CRUD）。

|  |
| --- |
| * 其内部实际上也确实持久化了一个Connection对象，即一个Session对应一个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的运作中心。 * Session是线程不安全的。 |

【问题】：如何解决Session线程不安全问题?

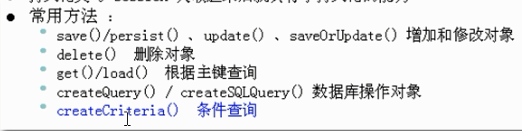
不要将Session定义为成员变量,只在方法中的局部变量使用.

(方法内部为什么不存在线程安全问题: 用的私有栈空间,不存在线程共享问题)

(每个方法使用栈空间 是线程私有空间,不存在多线程并发问题)

另外：Session还提供了一级缓存和持久化对象的常用操作。

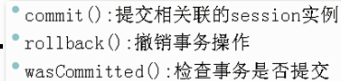
Session的常用方法:



## Transaction事务

作用：用于事务的管理（提交或回滚等）

常用操作:

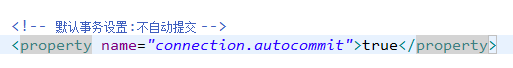


代码中操作事务

|  |
| --- |
| Transaction transaction=session.beginTransation()开启事务  //提交事务  transaction.commit()  //提交事务改进  session.getTransaction().commit();//引用型的操作 |

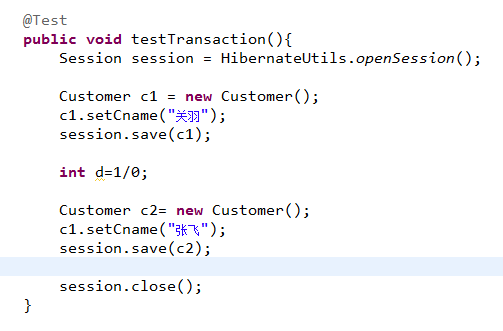
【注意】

|  |
| --- |
| 如果没有手动开启事务，那么每个Session的操作，都相当于一个独立的事务。  默认情况下，JDBC的事务是自动提交的，而hibernate的事务默认是不提交的，但可以通过配置的值为true，变成自动事务提交(mysql) |



【思考题面试题】：

问题：数据库中保存了几条数据？



【题目分析】

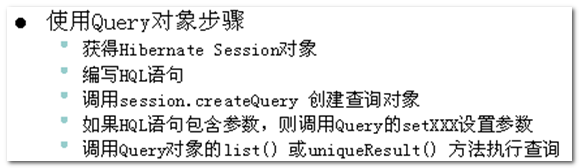
* 1. 自动事务是否开启，如果自动事务开启，关羽就能保存进去了。原因每个session操作都是一个独立的事务
  2. 如果不是自动事务提交，一般得手动管理事务。两个都保存不进去。

## Query查询接口

作用：Query对象主要用于执行查询操作。

Query接口用于接收HQl语句执行查询。

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



Hibernate为照顾和兼容原生的SQL，Hibernate提供了Query的子接口SqlQuery来执行SQL语句，用法和query接口的一样。

【示例】

通过Query接口执行查询，分别查询所有数据列表、根据条件查询一条数据（分别不使用和使用占位符）、

|  |
| --- |
| @Test  //测试Query接口的各种基本查询：查询所有，条件查询  **public** **void** testQuery(){  Session session = HibernateUtils.*openSession*();  session.beginTransaction();  //操作  //1.Query接口  //1.1查询所有  // List<Customer> list = session.createQuery("from Customer").list();  //扩展：from后面跟的是个类名，应该是全类名，如果你写短类名，那么Hibernate会自动反射出全类名  // List<Customer> list = session.createQuery("from cn.itcast.hibernate.a\_quickstart.Customer").list();  //\*是sql的语法  // List<Customer> list = session.createQuery("select \* from Customer").list();  List<Customer> list = session.createQuery("select c from Customer c").list();  System.*out*.println(list);  //1.2条件查询:按照姓名查询,这里面的name是属性，不是字段  List<Customer> list12 = session.createQuery("from Customer where name='Rose'").list();  System.*out*.println(list12);  //扩展：如果你能保证只有一个对象返回，可以使用另外一个方法,少写一段代码，list.get(0)  Customer customer = (Customer) session.createQuery("from Customer where name='Rose'").uniqueResult();  System.*out*.println(customer);  //1.3:条件查询：占位符  Customer customer3 = (Customer) session.createQuery("from Customer where name=?")  .setParameter(0, "Rose").uniqueResult();  System.*out*.println(customer3);    //2.sqlquery:面向sql的  List<Customer> list2 = session.createSQLQuery("select \* from t\_customer").addEntity(Customer.**class**).list();  System.*out*.println(list2);  //2.1条件查询  // List<Customer> list21 = session.createSQLQuery("select \* from t\_customer where name='Rose'").addEntity(Customer.class).list();  Customer customer21 = (Customer)session.createSQLQuery("select \* from t\_customer where name='Rose'").addEntity(Customer.**class**).uniqueResult();  System.*out*.println(customer21);  //占位符条件查询  Customer customer22 = (Customer)session.createSQLQuery("select \* from t\_customer where name=?")  // .setParameter(0, "Rose")  .addEntity(Customer.**class**)  //先绑定再设置参数，顺序问题  .setParameter(0, "Rose")  .uniqueResult();  System.*out*.println(customer22);      session.getTransaction().commit();  session.close();  } |

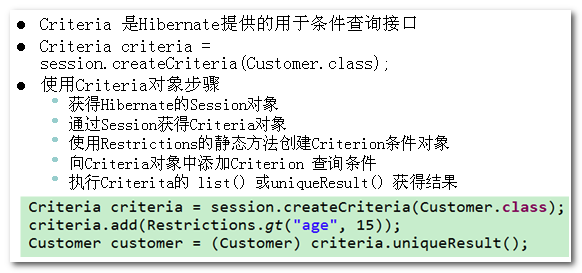
【注意】

SqlQuery.list 默认是生成的object[]，必须使用绑定实体后，Hibernate才会将结果自动封装成实体对象。

## Criteria查询接口

作用：用于执行查询操作。

特点：完全面向对象的方式添加条件,不需要写hql或sql



【示例】

使用Criteria接口对象执行查询，分别查询所有数据列表、根据条件查询一条数据

|  |
| --- |
| @Test  //Criteria接口查询  //使用Criteria接口对象执行查询，分别查询所有数据列表、根据条件查询一  **public** **void** testCriteria(){  Session session = HibernateUtils.*openSession*();  session.beginTransaction();  //操作  //获取Criteria接口对象：完全面向对象,最大的特点，不需要关心sql怎么写,自动生成sql。  Criteria criteria = session.createCriteria(Customer.**class**);//参数就是要查询的对象（对应表）  //查询所有  // List<Customer> list = criteria.list();  // System.out.println(list);  //条件查询(特点：要啥条件随便加，语句怎么写，不用管)  //参数是属性，不是字段  criteria.add(Restrictions.*eq*("name", "Rose"));//等值查询  criteria.add(Restrictions.*like*("city", "%上%"));//模糊查询  // Criterion  //查询  // List<Customer> list = criteria.list();  // System.out.println(list);    Customer customer = (Customer) criteria.uniqueResult();  System.*out*.println(customer);    session.getTransaction().commit();  session.close();    } |

## 小结回顾

常用的Hibernate核心API：

* Configuration核心配置对象,用来加载Hibernate配置文件，用来创建会话工厂
* SessionFactory 会话工厂,用来创建Hibernate会话Session,缓存了一些配置。（连接池等）
* Session 会话,相当于一个数据库连接(持久了一个connection),进行数据库表CRUD操作，它是hibernate的运作核心。
* Transaction 事务,对操作进行事务管理
* Query查询接口,接收HQL语句,子接口SQLQuery接受sql语句
* Criteria条件查询（QBC：query by Criteria）,完全面向对象方式进行多条件的组合查询，无需关心sql的写法。

# 持久化对象PO

## 理解持久化对象PO

### PO对象是什么

全称：Persistent Object。

持久化（Persistence），即把数据（如内存中的对象）保存到可永久保存的存储设备中（如磁盘）。持久化的主要应用是将内存中的对象存储在关系型的数据库中，当然也可以存储在磁盘文件中、XML数据文件中等等



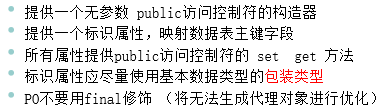
称呼上:

实体类----持久化类-

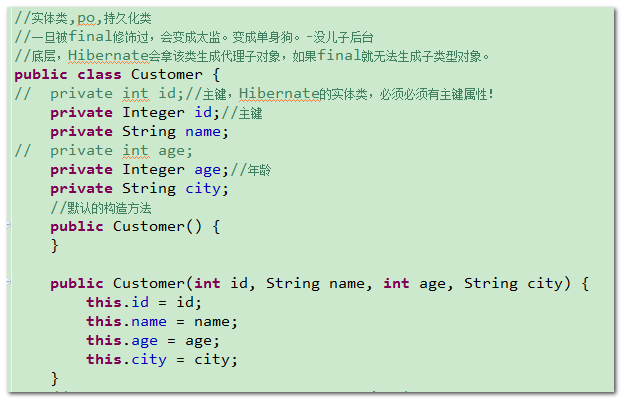
new出来的实体对象---》po（持久化对象）

### 编写规则

PO类的要求如下：



【示例】



### 属性类型的问题讨论

问题思考：PO属性的数据类型使用包装类型还是原始类型？

【场景模拟阅读】

|  |
| --- |
| //需求：有个表存放学生的分数，假如有个人没有参加考试，那么这个分数的字段该是什么值？这个字段值是0还是null合适呢?看需求,但用null更合适.  基本数据类型double默认值是0，Double默认值是null  //如果说,使用基本的数据类型,则,无法通过这个字段来区分是否参加了考试(假如有个人考试了但是0分.)。假如要求参加考试的学生必须有分数，而没参加考试的学生没有分数（为空），double无法识别了。一般我们都用包装类 |

**尽量使用包装类型，不要使用原始类型**。

【示例】

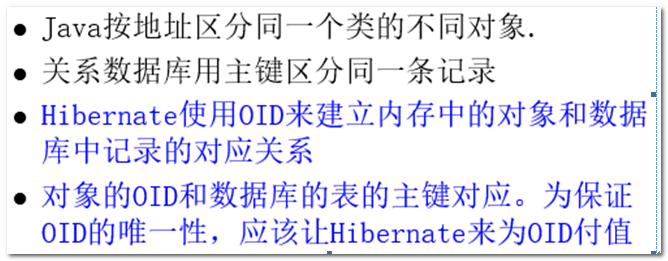
PO的属性的数据类型分别使用原始类型和包装类型的测试。

|  |
| --- |
| @Test  **public** **void** testPO(){  Session session = HibernateUtils.*openSession*();  session.beginTransaction();    // Customer customer=(Customer)session.get(Customer.class, 2);  // System.out.println(customer);    //保存数据  Customer customer = **new** Customer();  customer.setName("xiaohong");  customer.setCity("南京");  // customer.setAge(28);//问题：数据库有对应字段的值么？    session.save(customer);    session.getTransaction().commit();  session.close();  } |

## 理解OID-持久化对象的唯一标识

### 什么是OID

OID（Object identifier），即对象标识，用来标识唯一对象的，因此，OID在PO中被称为持久化对象的唯一标识。



Hibernate框架根据OID标识，是否为同一个对象

【课后阅读】

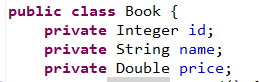
|  |
| --- |
| 在关系数据库中，主键用来识别记录，并保证每条记录的唯一性。数据库中的主键最重要的3个基本要素就是不允许为null,不允许有重复值,主键永远不会改变. 所以通常我们设计表都会设计主键的值为自动增加,没有业务逻辑含义的一组数字,当然针对每个数据库,设置的方法也不同.  在Java语言中，通过比较两个变量所引用对象的内存地址是否相同，或者比较两变量引用的对象是否相等。  Hibernate为了解决两者之间的不同，使用对象标识符（OID）来标识对象的唯一性。OID是关系数据库中主键在Java对象模型中的等价物。在运行时，Hibernate根据OID来维持Java对象和数据库中的对应关系。  OID具有唯一性和不变性，一般是由HIBERNATE或数据库来给其赋值。  OID(Object ID)应当没有任何业务相关含义，OID绝对不应当具有任何业务含义。 |

# hbm映射配置详解

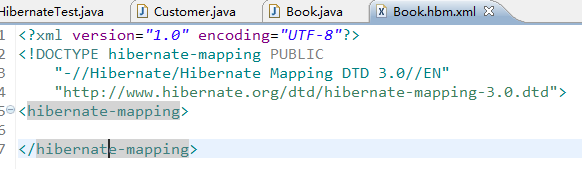
## 准备工作（全天复习）

目标：建立PO和hbm映射（加入到核心配置文件中）、建表（构建工厂）。

建立book的po类:



建立hbm映射文件：



填充基本的映射文件内容：



加载hbm映射

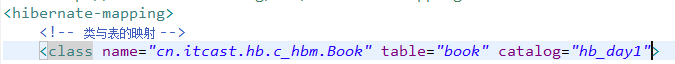


建表测试：

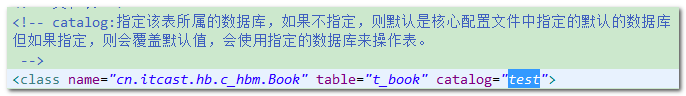
|  |
| --- |
| @Test  //建表测试  **public** **void** testCreateTabele(){  HibernateUtils.*getSessionFactory*();    } |

## class元素

作用：指定类与表的映射，用来映射表的一些属性。



* name：类名
* table：表名
* catalog：数据库名—用于mysql



* schema：命名空间（方案—等同于用户，类似于“域”）—用于Oracle 

注意： catalog和schema 都可以不写，使用核心配置文件中配置的默认数据库和名称空间

如果你在hbm中配置的catalog和核心配置文件中的不一样?以hbm中配置的catalog为主。比如，catalog是test

【示例】

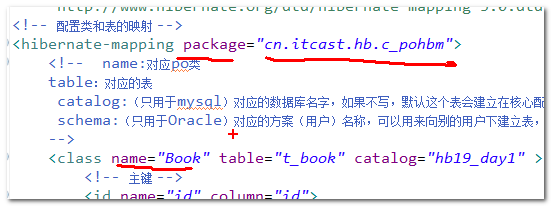
更改catalog会将表生成到其他的位置

如果是mysql的root用户登录的情况下,可以指定写入任何的库.

如果是Oracle用户登录的情况下,schema指定的用户和核心配置文件中的用户不一样,会自动建立表和操作表么?不一定,看核心配置文件中的用户是否有访问其他用户的权限.

[扩展了解]

可以将包名写在映射标签上，也可以：



## id元素

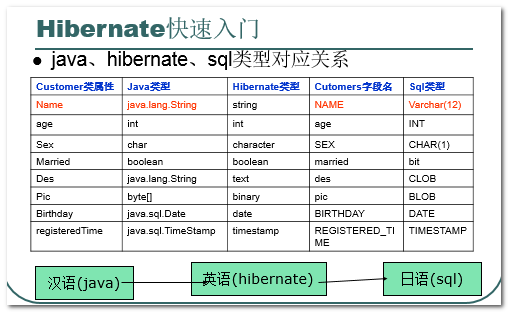
作用：主键策略，设定持久化类的OID和表主键的映射

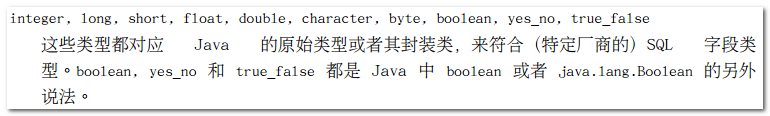


属性：

* name: 标识持久化类 OID 的属性名
* column: 设置标识属性所映射的数据列的列名(主键字段的名字)，可省略。
* length：字段的长度，如果不指定，则使用数据库字段的默认最大长度。
* type:指定 Hibernate 映射类型. Hibernate 映射类型是 Java 类型与 SQL 类型的桥梁。如果没有为某个属性显式设定映射类型, Hibernate 会运用反射机制先识别出持久化类的特定属性的 Java 类型, 然后自动使用与之对应的默认的 Hibernate 映射类型。

Java 的基本数据类型和包装类型对应相同的 Hibernate 映射类型. 基本数据类型无法表达 null, 所以对于持久化类的 OID 推荐使用包装类型（integer，long，string等）。





## generator元素

作用：持久化类的标识符(OID)生成器，用来指定主键的生成策略。



【主键策略知识】

数据库主键分为两种：代理主键和自然主键。

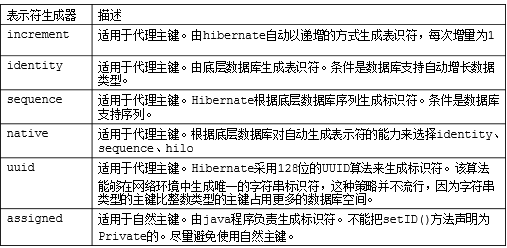
两者的区别为：

* 代理主键：不具有业务含义的字段作为主键的，---例如自增长id，uuid
* 自然主键：具有业务含义的字段作为主键的， ---例如订单编号(有些表的设计中订单编号是有规则.2015011002998)

两者的选择：企业开发中，会根据业务需要选择，但使用代理主键居多。

属性：

* class属性：指定使用的标识符生成器全限定类名或其缩写名。



选择：最常用的：native，其次：identity,sequence,uuid,assigne

### native

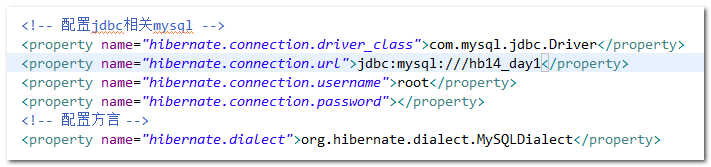
标识符生成器依据底层数据库对自动生成标识符的支持能力, 底层其实子自动选择使用 identity, sequence 或 hilo 标识符生成器.

如果数据库 mysql ---- identity

如果数据库 oracle ---- sequence （hibernate 会自动创建一个新的序列）

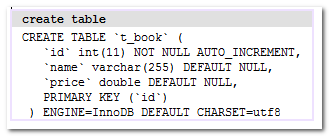
### Identity--mysql演示

切换到mysql: (前提是要导入mysql的jar包)

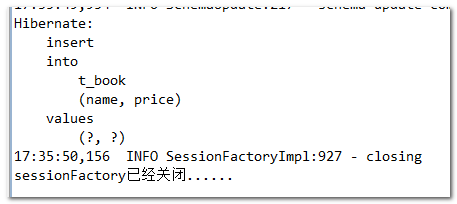


依赖数据表底层自增长 ，用于DB2, Mysql, MSSQLServer, Sybase

创建表后,生成的sql:



插入数据:

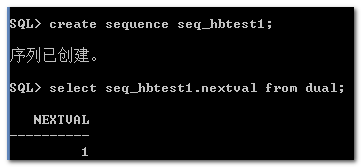


采取抢占的形式进行id的赋值。测试.

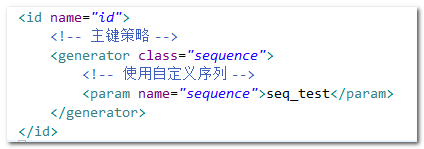
### sequence—oracle演示

依赖于底层数据库，采用序列的机制，主要用于oracle，db2等。

可以指定自定义的序列;



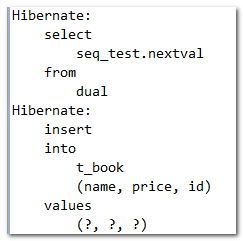
修改策略：



测试：



结果：

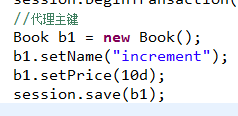


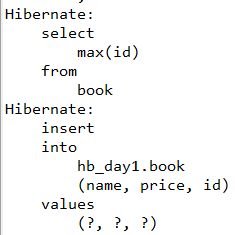
经过测试，使用了自定义的序列。

如果不指定自定义的序列,会使用默认的序列(hibernate生成)

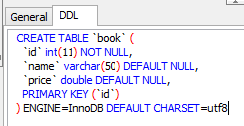
### increment

由hibernate来生成OID和维护的，原理是select max(id) +1



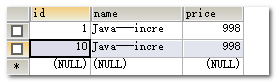


如果数据表中没有数据，则初始的时候,hibernate给值是1,再次给值2

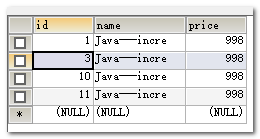


表主键没有自增长了

问题:如果手动将主键值改大了,会如何?



再次保存会:



适用场景：



问题：可能出现多线程冲突问题 ，两个线程同时查询max(id)， 同时+1 ,insert

【示例】

debug来控制两个线程。（试试）



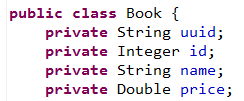
### uuid

用于String类型，生成代理主键， 采用uuid （32位）作为主键值

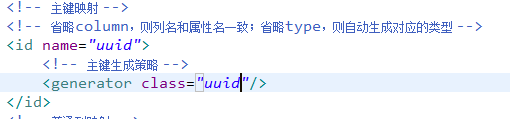
Hibernate会产生不重复的32位字符串作为主键

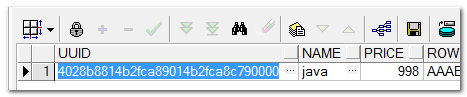
【示例】

增加一个string类型的字段：



更改映射的主键：





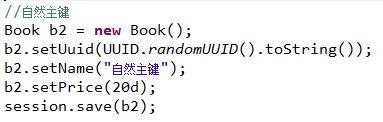
### assigned

唯一的一个自然主键设置方式，必须手动设置主键的值。

如果不指定会出现：



指定示例：

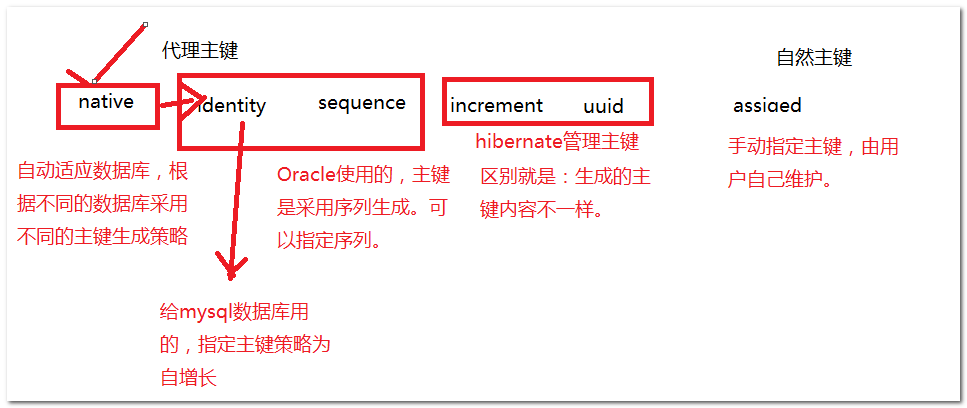




如果不指定主键,则报错:



### 小结



## property元素

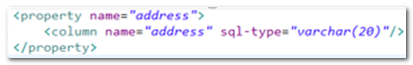
作用：配置非主键的属性和表字段的映射

属性（和id元素一样）：

* name：类中属性名
* column: 设置标识属性所映射的数据列的列名(字段的名字)，可省略。
* length：字段的长度，如果不指定，则使用数据库字段的默认最大长度。
* type:指定 Hibernate 映射类型. Hibernate 映射类型是 Java 类型与 SQL 类型的桥梁。如果没有为某个属性显式设定映射类型, Hibernate 会运用反射机制先识别出持久化类的特定属性的 Java 类型, 然后自动使用与之对应的默认的 Hibernate 映射类型。

Java 的基本数据类型和包装类型对应相同的 Hibernate 映射类型. 基本数据类型无法表达 null, 所以对于持久化类的 OID 推荐使用包装类型（integer，long，string等）。

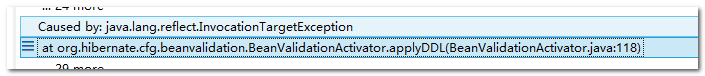
另外了解：也可以使用子元素<column>来配置数据库中的字段类型（不推荐）：



【扩展】type：字段类型（是hibernate类型）：扩展（如果想写数据库类型，则需要子元素column：）

# 其他：jee6的bean校验错误

如果你用myeclise2014版本的时候,默认建立的web工程是用jee6,那么在运行hibernate的时候,会报错:bean校验的错误.

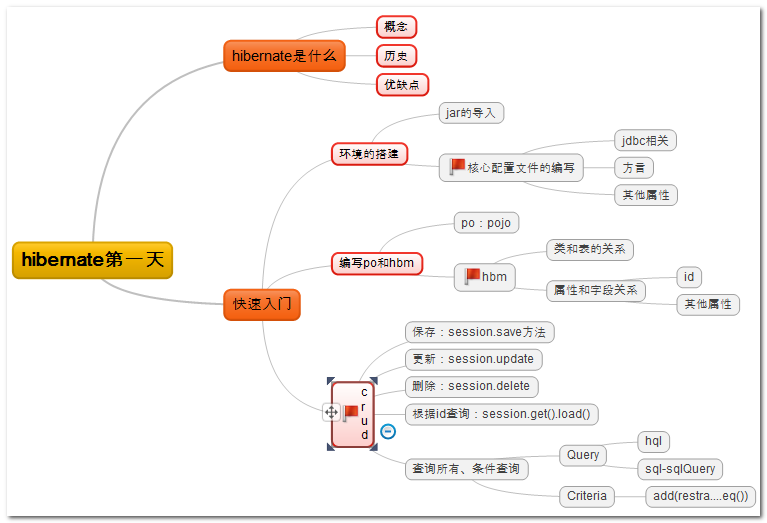


解决方案:在核心配置文件(hibernate.cfg.xml)中,配置一个阻止bean校验的一个属性





# 小结+重点



# 作业

【作业一】

课程中的快速入门，即基本的增删改查(CRUD)，写一遍。

【作业二】

编写一个登录小程序。

业务逻辑：页面输入用户名和密码，在后台验证，是否在数据库中存在，如果存在，则提示登录成功，否则，提示登录失败。

技术要点：将以前的练习中的dbutil改造为Hibernate，用Hibernate作为dao（持久层）的技术与数据库打交道。

参考步骤：

1. 搭建环境：导入hb的jar，核心配置文件，工具类
2. 编写PO和hbm映射
3. 直接在程序中编写crud