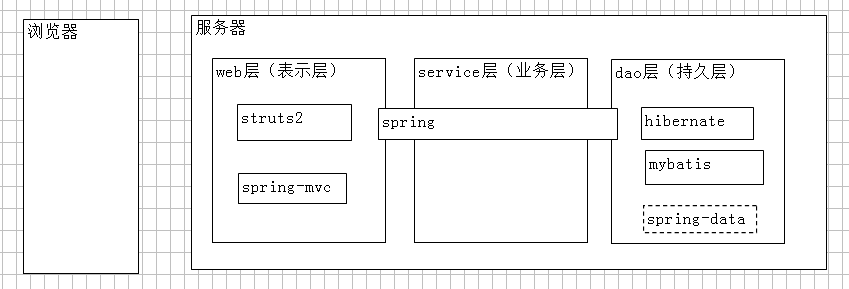
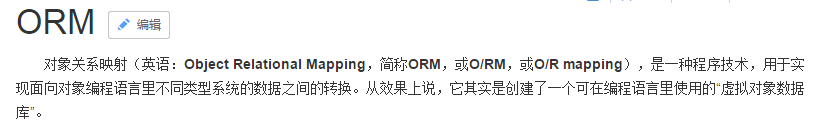
# 框架体系结构



# hibernate入门

## ORM框架

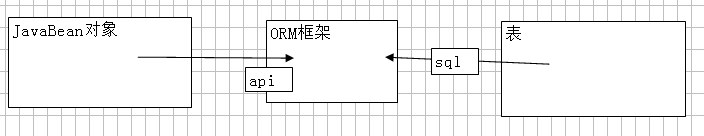


Hibernate是一个数据持久化层的ORM框架.

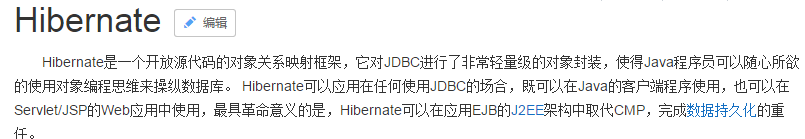
Object：对象，java对象，此处特指JavaBean

Relational：关系，二维表，数据库中的表。

映射|映射元数据：对象中属性，与表的字段，存在对应关系。



## 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
  + ORM 就是通过将Java对象映射到数据库表，通过操作Java对象，就可以完成对数据表的操作
* Hibernate提供了对关系型数据库增删改成操作

## 主流的ORM框架

* + JPA Java Persistence API.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
  + Apache DBUtils 、Spring JDBCTemplate

## 优点

|  |
| --- |
| * Hibernate对JDBC访问数据库的代码做了封装，大大简化了数据访问层繁琐的重复性代码 * Hibernate是一个基于jdbc的主流持久化框架，是一个优秀的orm实现，它很大程度的简化了dao层编码工作 session.save(User); * Hibernate使用java的反射机制 * Hibernate的性能非常好，因为它是一个轻量级框架。映射的灵活性很出色。它支持很多关系型数据库，从一对一到多对多的各种复杂关系 |

# 入门案例【掌握】

## 编写流程

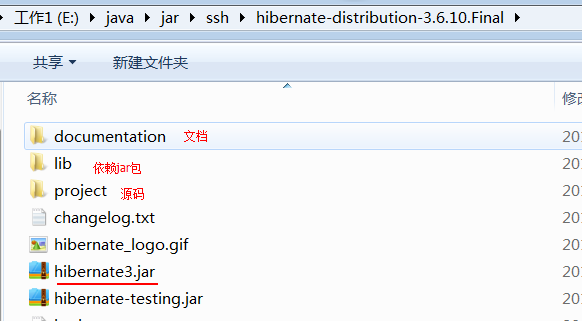
* 1. 导入jar包
* 2. 创建数据库和表
* 3. 编写核心配置文件（hibernate.cfg.xml）--> 配置获得链接等参数
* 4. 编写映射文件 hibernate mapping(\*.hbm.xml)
* 5 使用api测试

## 数据库和表

|  |
| --- |
| create database h\_day01\_db;  use h\_day01\_db;  create table t\_user(  id int auto\_increment primary key,  username varchar(50),  password varchar(30)  ); |

## 导入jar包

* 版本：3.6.10 --> hibernate 4 建议注解开发，hibernate 4 对 3 不兼容。
* 目录结构



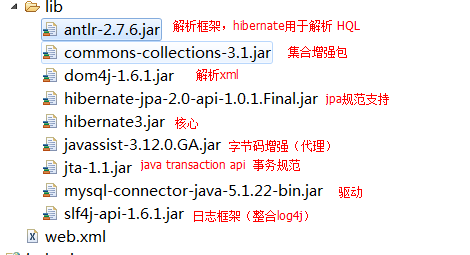
* jar介绍

核心：

必须：\lib\required

jpa规范：lib\jpa

mysql驱动： 



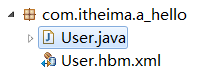
## 编写JavaBean + 映射文件

* 文件位置：javabean同包
* 文件名称：javabean同名
* 扩展名：\*.hbm.xml
* 内容：

添加约束







|  |
| --- |
| **public** **class** User {    /\*  \* create table t\_user(  id int auto\_increment primary key,  username varchar(50),  password varchar(30)  );  \*/  **private** Integer uid;  **private** String username;  **private** String password; |
| <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  <hibernate-mapping>  <class name=*"com.itheima.a\_hello.User"* table=*"t\_user"*>  <!-- 主键 -->  <id name=*"uid"*>  <!-- 固定值：主键生成策略 -->  <generator class=*"native"*></generator>  </id>    <!-- 普通属性 -->  <property name=*"username"*></property>  <property name=*"password"*></property>    </class>  </hibernate-mapping> |

## 编写核心配置文件

* 位置：类路径（classpath、src）-->WEB-INF/classes
* 名称：hibernate.cfg.xml
* 内容：

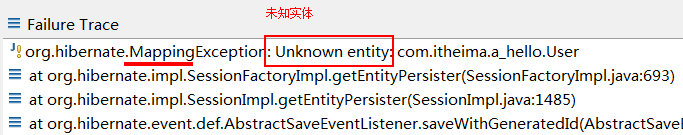
添加约束

|  |
| --- |
| <!DOCTYPE hibernate-configuration PUBLIC  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">  <hibernate-configuration>  <!-- SessionFactory，相当于之前学习连接池配置 -->  <session-factory>  <!-- 1 基本4项 -->  <property name=*"hibernate.connection.driver\_class"*>com.mysql.jdbc.Driver</property>  <property name=*"hibernate.connection.url"*>jdbc:mysql:///h\_day01\_db</property>  <property name=*"hibernate.connection.username"*>root</property>  <property name=*"hibernate.connection.password"*>1234</property>    <!-- 添加映射文件 -->  <mapping resource=*"com/itheima/a\_hello/User.hbm.xml"*/>  </session-factory>  </hibernate-configuration> |

## 测试

|  |
| --- |
| @Test  **public** **void** demo01(){  User user = **new** User();  user.setUsername("伟哥哥");  user.setPassword("1234");      //1 加载配置文件获得核心配置对象  Configuration config = **new** Configuration().configure();  //2 获得工厂 SessionFactory，相当于连接池  SessionFactory factory = config.buildSessionFactory();  //3获得会话session，相当于链接Connection  Session session = factory.openSession();  //4 开启事务  Transaction transaction = session.beginTransaction();    //操作  session.save(user);    //5 提交事务 | 回滚事务  transaction.commit();  //6 释放资源--关闭session  session.close();  //7 释放资源--关闭工厂factory  factory.close();  } |

## 常见异常

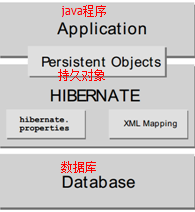


解决方案：

将映射文件添加到核心配置文件中 hbm.xml --> hibernate.cfg.xml

# api详解【多练】

## 体系结构

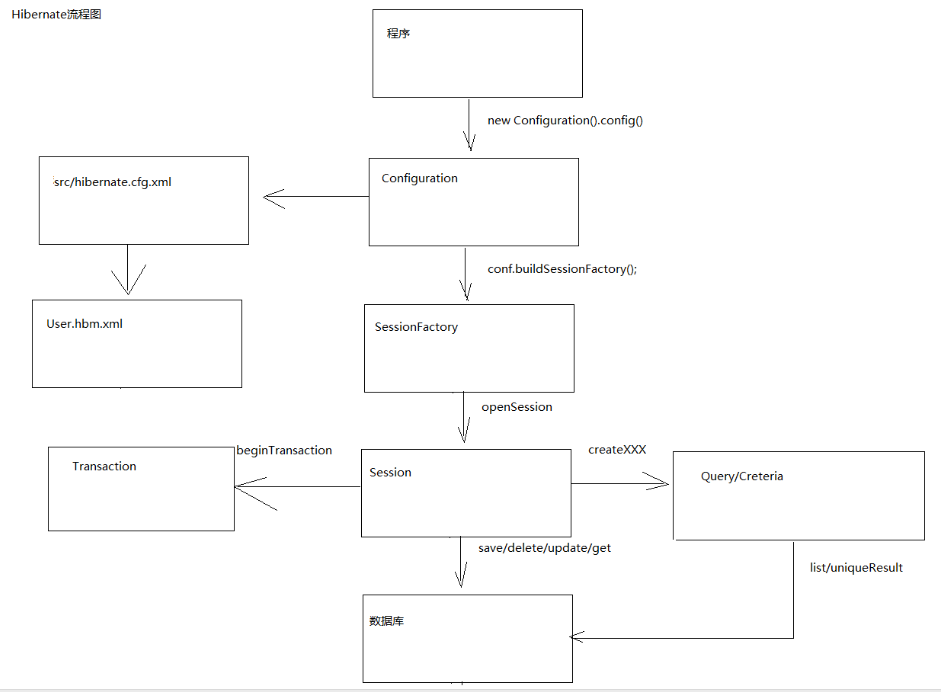


PO：persistent object ，用于与数据库交互数据。--dao层 （JavaBean + hbm ）

BO：Business object 业务数据对象。--service层

VO：Value Object 值对象。--web层

开发中：直接使用JavaBean 描述三个对象。



## Configuration 配置对象

* hibernate 核心配置文件种类

hibernate.cfg.xml 通常使用xml配置文件，可以配置内容更丰富。

hibernate.properties 用于配置key/value 形式的内容，key不能重复的。配置有很多的局限性。一般不用。

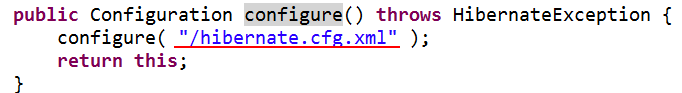
参考文件：hibernate-distribution-3.6.10.Final\project\etc\ hibernate.properties

提供了核心配置文件常用的配置项，及选择参数。

1.提供构造 new Configuration() hibernate将自动加载 hibernate.properties文件

hibernate.properties文件必须存放在类路径（src）下

2.提供方法 configure() 将加载src下的hibernate.cfg.xml



3.扩展api

configure(String) 加载指定目录下的 xml文件

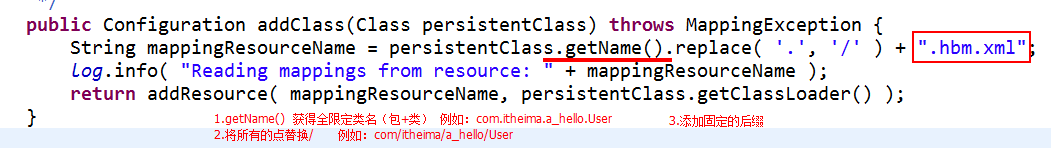
4.手动加载配置文件

// 手动加载指定的配置文件

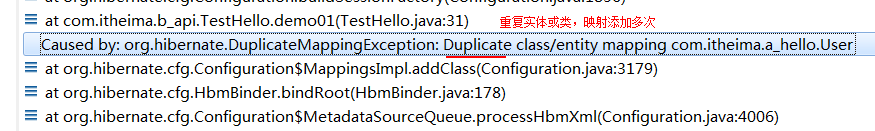
config.addResource("com/itheima/a\_hello/User.hbm.xml");

// 手动加载指定类，对应的映射文件 User--> User.hbm.xml

config.addClass(User.**class**);



* 常见异常：



开发中：将hbm.xml映射 配置 hibernate.cfg.xml

学习中：可以使用 addClass 或 addResource

## SessionFactory工厂

* SessionFactory 相当于java web连接池，用于管理所有session
* 获得方式：config.buildSessionFactory();
* sessionFactory hibernate缓存配置信息 （数据库配置信息、映射文件，预定义HQL语句 等）
* SessionFactory线程安全，可以是成员变量，多个线程同时访问时，不会出现线程并发访问问题。
* 提供api：

//打开一个新的会话 session

factory.openSession();

//获得当前线程中绑定的会话session

factory.getCurrentSession();

hibernate支持，将创建的session绑定到本地线程中，底层使用ThreadLocal，在程序之间共享session。

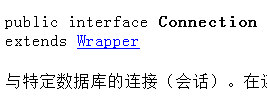
1.必须在hibernate.cfg.xml 配置

|  |
| --- |
| <!-- 2 与本地线程绑定 -->  <property name=*"hibernate.current\_session\_context\_class"*>thread</property> |

2.如果提交或回滚事务，底层将自动关闭session

## Session 会话

* Session 相当于 JDBC的 Connection -- 会话



* 通过session操作PO对象 --增删改查
* session单线程，线程不安全，不能编写成成员变量。
* session api

save 保存

update 更新

delete 删除

get 通过id查询，如果没有 null

load 通过id查询，如果没有抛异常

createQuery("hql") 获得Query对象

createCriteria(Class) 获得Criteria对象

## Transaction 事务

开启事务 beginTransaction()

获得事务 getTransaction()

提交事务：commit()

回滚事务：rollback()

|  |
| --- |
| try{  //开启  //session操作  //提交  } catch(e){  //回滚  }  扩展：不需要手动的管理事务，之后所有的事务管理都交予spring。 |

## Query对象

* hibernate执行hql语句
* hql语句：hibernate提供面向对象查询语句，使用对象（类）和属性进行查询。区分大小写。
* 获得 session.createQuery("hql")
* 方法：

list() 查询所有

uniqueResult() 获得一个结果。如果没有查询到返回null，如果查询多条抛异常。

setFirstResult(int) 分页，开始索引数startIndex

setMaxResults(int) 分页，每页显示个数 pageSize

## Criteria对象(了解)

* QBC（query by criteria），hibernate提供纯面向对象查询语言，提供直接使用PO对象进行操作。
* 获得方式：Criteria criteria = session.createCriteria(User.**class**);
* 条件

criteria.add(Restrictions.*eq*("username", "tom"));

// Restrictions.gt(propertyName, value) 大于

// Restrictions.ge(propertyName, value) 大于等于

// Restrictions.lt(propertyName, value) 小于

// Restrictions.le(propertyName, value) 小于等于

// Restrictions.like(propertyName, value) 模糊查询，注意：模糊查询值需要使用 % \_

## 工具类

|  |
| --- |
| **public** **class** H3Utils {    // 会话工厂，整个程序只有一份。  **private** **static** SessionFactory *factory*;  **static**{  //1 加载配置  Configuration config = **new** Configuration().configure();    //2 获得工厂  *factory* = config.buildSessionFactory();  //3 关闭虚拟机时,释放SessionFactory  Runtime.getRuntime().addShutdownHook(new Thread(new Runnable() {    @Override  public void run() {  System.out.println("虚拟机关闭!释放资源");  sf.close();  }  }));  }      /\*\*  \* 获得一个新的session  \* **@return**  \*/  **public** **static** Session openSession(){  **return** *factory*.openSession();  }    /\*\*  \* 获得当前线程中绑定session  \* \* 注意：必须配置  \* **@return**  \*/  **public** **static** Session getCurrentSession(){  **return** *factory*.getCurrentSession();  }      } |
|  |

# 核心配置文件详解

## 详细配置【多读】

|  |
| --- |
| <!-- SessionFactory，相当于之前学习连接池配置 -->  <session-factory>  <!-- 1 基本4项 -->  <property name=*"hibernate.connection.driver\_class"*>com.mysql.jdbc.Driver</property>  <property name=*"hibernate.connection.url"*>jdbc:mysql:///h\_day01\_db</property>  <property name=*"hibernate.connection.username"*>root</property>  <property name=*"hibernate.connection.password"*>1234</property>    <!-- 2 与本地线程绑定 -->  <property name=*"hibernate.current\_session\_context\_class"*>thread</property>    <!-- 3 方言：为不同的数据库，不同的版本，生成sql语句（DQL查询语句）提供依据  \* mysql 字符串 varchar  \* orcale 字符串 varchar2  -->  <property name=*"hibernate.dialect"*>org.hibernate.dialect.MySQL5Dialect</property>    <!-- 4 sql语句 -->  <!-- 显示sql语句 -->  <property name=*"hibernate.show\_sql"*>true</property>  <property name=*"hibernate.format\_sql"*>true</property>    <!-- 5 自动创建表（了解） ，学习中使用，开发不使用的。  \* 开发中DBA 先创建表，之后根据表生产 PO类  \* 取值：  update：【】  如果表不存在，将创建表。  如果表已经存在，通过hbm映射文件更新表（添加）。（映射文件必须是数据库对应）  表中的列可以多，不负责删除。  create ：如果表存在，先删除，再创建。程序结束时，之前创建的表不删除。【】  create-drop：与create几乎一样。如果factory.close()执行，将在JVM关闭同时，将创建的表删除了。(测试)  validate：校验 hbm映射文件 和 表的列是否对应，如果对应正常执行，如果不对应抛出异常。(测试)  -->  <property name=*"hibernate.hbm2ddl.auto"*>create</property>    <!-- 6 java web 6.0 存放一个问题  \* BeanFactory 空指针异常  异常提示：org.hibernate.HibernateException: Unable to get the default Bean Validation factory  \* 解决方案：取消bean校验  -->  <property name=*"javax.persistence.validation.mode"*>none</property>    <!-- 添加映射文件  <mapping >添加映射文件  resource 设置 xml配置文件 （addResource(xml)）  class 配置类 (addClass(User.class)) 配置的是全限定类名  -->  <mapping resource=*"com/itheima/a\_hello/User.hbm.xml"*/>  </session-factory> |

# Hibernate中持久化类

## 编写规则

* + 提供一个无参数 public访问控制符的构造器
  + 提供一个标识属性，映射数据表主键字段
  + 所有属性提供public访问控制符的 set get 方法(javaBean)
  + 标识属性应尽量使用基本数据类型的包装类型
  + 不要用final修饰实体 （将无法生成代理对象进行优化）

## 持久化对象的唯一标识 OID

* Java按地址区分同一个类的不同对象.
* 关系数据库用主键区分同一条记录
* Hibernate使用OID来建立内存中的对象和数据库中记录的对应关系

结论: 对象的OID和数据库的表的主键对应。为保证OID的唯一性，应该让Hibernate来为OID付值

## 区分自然主键和代理主键

* 主键需要具备: 不为空/不能重复/不能改变

自然主键: 在业务中,某个属性符合主键的三个要求.那么该属性可以作为主键列.

代理主键: 在业务中,不存符合以上3个条件的属性,那么就增加一个没有意义的列.作为主键.

## 基本数据与包装类型

* 基本数据类型和包装类型对应hibernate的映射类型相同
* 基本类型无法表达null、数字类型的默认值为0。
* 包装类默认值是null。当对于默认值有业务意义的时候需要使用包装类。

## 类型对应

|  |  |  |
| --- | --- | --- |
| Java数据类型 | Hibernate数据类型 | 标准SQL数据类型 (PS:对于不同的DB可能有所差异) |
| byte、java.lang.Byte | byte | TINYINT |
| short、java.lang.Short | short | SMALLINT |
| int、java.lang.Integer | integer | INGEGER |
| long、java.lang.Long | long | BIGINT |
| float、java.lang.Float | float | FLOAT |
| double、java.lang.Double | double | DOUBLE |
| java.math.BigDecimal | big\_decimal | NUMERIC |
| char、java.lang.Character | character | CHAR(1) |
| boolean、java.lang.Boolean | boolean | BIT |
| java.lang.String | string | VARCHAR |
| boolean、java.lang.Boolean | yes\_no | CHAR(1)('Y'或'N') |
| boolean、java.lang.Boolean | true\_false | CHAR(1)('Y'或'N') |
| java.util.Date、java.sql.Date | date | DATE |
| java.util.Date、java.sql.Time | time | TIME |
| java.util.Date、java.sql.Timestamp | timestamp | TIMESTAMP |
| java.util.Calendar | calendar | TIMESTAMP |
| java.util.Calendar | calendar\_date | DATE |
| byte[] | binary | VARBINARY、BLOB |
| java.lang.String | text | CLOB |
| java.io.Serializable | serializable | VARBINARY、BLOB |
| java.sql.Clob | clob | CLOB |
| java.sql.Blob | blob | BLOB |
| java.lang.Class | class | VARCHAR |
| java.util.Locale | locale | VARCHAR |
| java.util.TimeZone | timezone | VARCHAR |
| java.util.Currency | currency | VARCHAR |

## 普通属性

|  |
| --- |
| <hibernate-mapping>  package 用于配置PO类所在包  例如： package="com.itheima.d\_hbm"  <class> 配置 PO类 和 表 之间对应关系  name：PO类全限定类名  例如：name="com.itheima.d\_hbm.Person"  如果配置 package，name的取值可以是简单类名 name="Person"  table : 数据库对应的表名  dynamic-insert="false" 是否支持动态生成insert语句  dynamic-update="false" 是否支持动态生成update语句  如果设置true，hibernate底层将判断提供数据是否为null，如果为null，insert或update语句将没有此项。  普通字段  <property>  name : PO类的属性  column : 表中的列名，默认name的值相同  type:表中列的类型。默认hibernate自己通过getter获得类型，一般情况不用设置  取值1： hibernate类型  string 字符串  integer 整形  取值2： java类型 （全限定类名）  java.lang.String 字符串  取值3：数据库类型  varchar(长度) 字符串  int 整形  <property name="birthday">  <column name="birthday" sql-type="datetime"></column>  </property>  javabean 一般使用类型 java.util.Date  jdbc规范提供3中  java类型 mysql类型  java.sql.Date date  java.sql.time time  java.sql.timestamp timestamp  null datetime    以上三个类型都是java.util.Date子类    length : 列的长度。默认值：255  not-null : 是否为null  unique : 是否唯一  access：设置映射使用PO类属性或字段  property : 使用PO类属性，必须提供setter、getter方法  field : 使用PO类字段，一般很少使用。  insert 生成insert语句时，是否使用当前字段。  update 生成update语句时，是否使用当前字段。  默认情况：hibernate生成insert或update语句，使用配置文件所有项  注意：配置文件如果使用关键字，列名必须使用重音符 |

## 主键

|  |
| --- |
| 主键  <id>配置主键  name:属性名称  access="" 设置使用属性还是字段  column="" 表的列名  length="" 长度  type="" 类型  <generator> class属性用于设置主键生成策略  1.increment 由hibernate自己维护自动增长  底层通过先查询max值，再+1策略  不建议使用，存在线程并发问题  2.identity hibernate底层采用数据库本身自动增长列  例如：mysql auto\_increment  3.sequence hibernate底层采用数据库序列  例如：oracle 提供序列  4.hilo    </generator>  5.native 根据底层数据库的能力选择 identity、sequence 或者 hilo 中的一个。【】  ##以上策略使用整形，long, short 或者 int 类型  6.uuid 采用字符串唯一值【】  ##以上策略 代理主键，有hibernate维护。  7.assigned 自然主键，由程序自己维护。【】 |