第一阶段：

第一周：环境变量，JRE,JVM,JDK(1.8) 数据类型 运算 变量 常量 分支语句 循环语句

方法，数组，二分，冒泡，选择，Arrays工具类

第二周：抽象，封装，

继承:类与类之间的关系 has a 整体与部分的关系 is a 继承

多态：父类对象指向子类的实现 前提：继承和实现

抽象类、抽象方法

接口 (方法/属性)

第三周：四中内部类 枚举 异常Throable

集合：Collection

List :ArrayList LinkedList Vector Stack

Set :HashSet LinkedHashSet TreeSet

Map:

HashMap

TreeMap

Hashtable

Properties 配置文件类

第四周：线程:实现线程的三种方式 线程安全问题 枷锁：同步代码块 同步方法

生产者消费者 线程之间的通讯

IO流:File类 字节流 字符流 缓冲流 转换流 对象流 内存流 打印流 标准输入输出流...

第五周：网络编程 UDP数据包 TCP三次握手

BS/CS

正则表达式 反射

单例设计模式 工厂 模板 装饰者 代理模式、动态代理模式（JDK:Proxy）

第二阶段：

第六周 html css js(用的非常多 对表单数据校验 事件、DOM编程 全选 省市二级联动)

第七周 Mysql:DCL,DDL(create drop alter),DML(insert into 、update、delete),DQL(select)

多表查询：

笛卡尔积

内连接：

显式内连接 inner join on条件

隐式内连接 where

外链接：

左外连接：left outer join on

右外链接：right outer join on

自连接：

JDBC 增删改查

第八周：

xml解析 DOM4J

Tomcat服务器 软件 配置端口号 虚拟网站 虚拟域名 虚拟的站点

Http协议：请求行 请求头 响应头 实体内容

Servlet编程：

HttpServletRequest

HttpServletResponse

ServletContext:1.获取全局的配置参数2.域对象 3.获得真实路径 项目名

ServletConfig

cookie session

jsp:三大指令 九大内置对象 四个域对象 EL表达式 JSTL核心c标签

第九周：

连接池：DBCP/C3P0连接池

DBUtils工具

BeanUtils工具

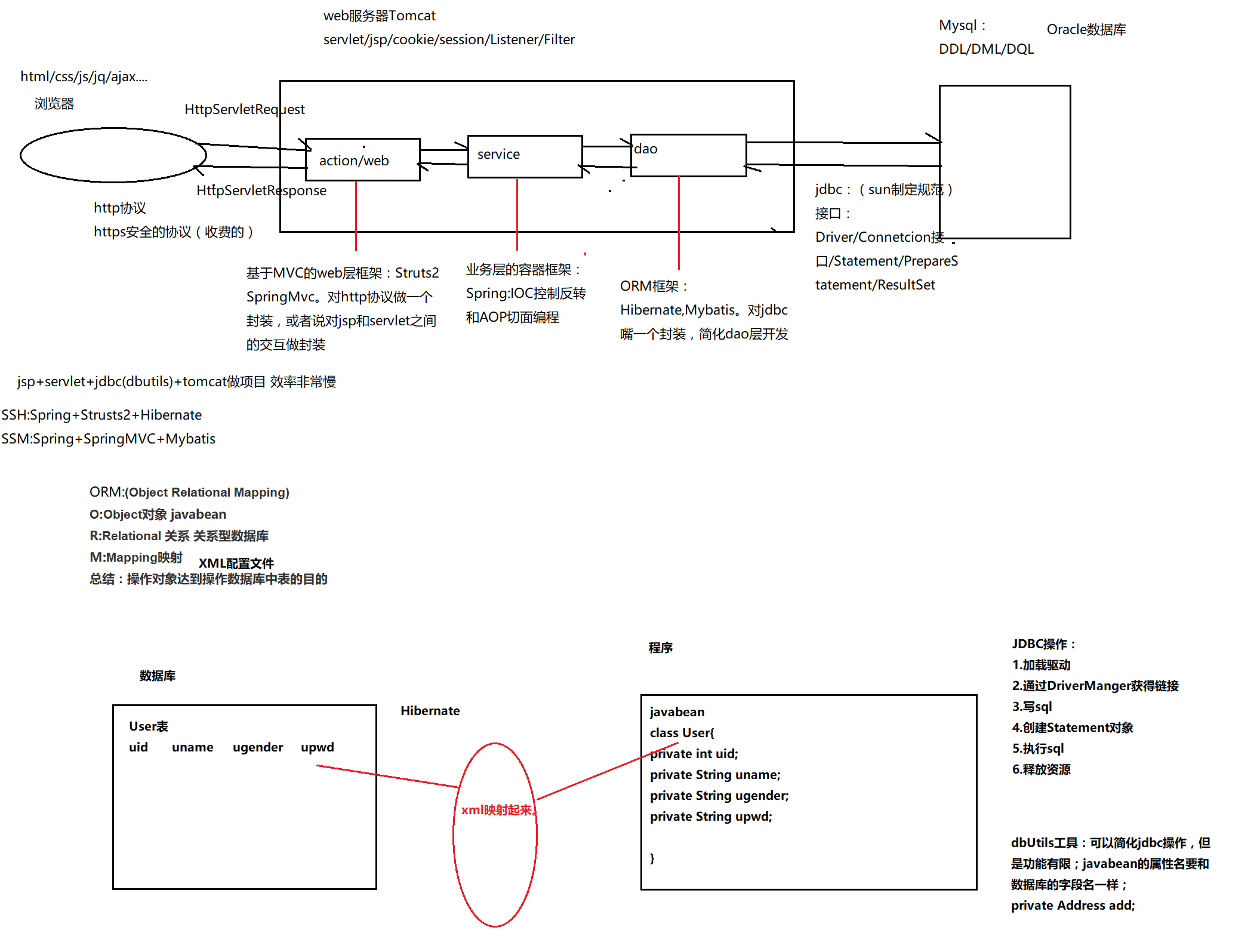
事务

jdbc处理事务 Dbutils处理事务

JQ AJAX $.ajax({})

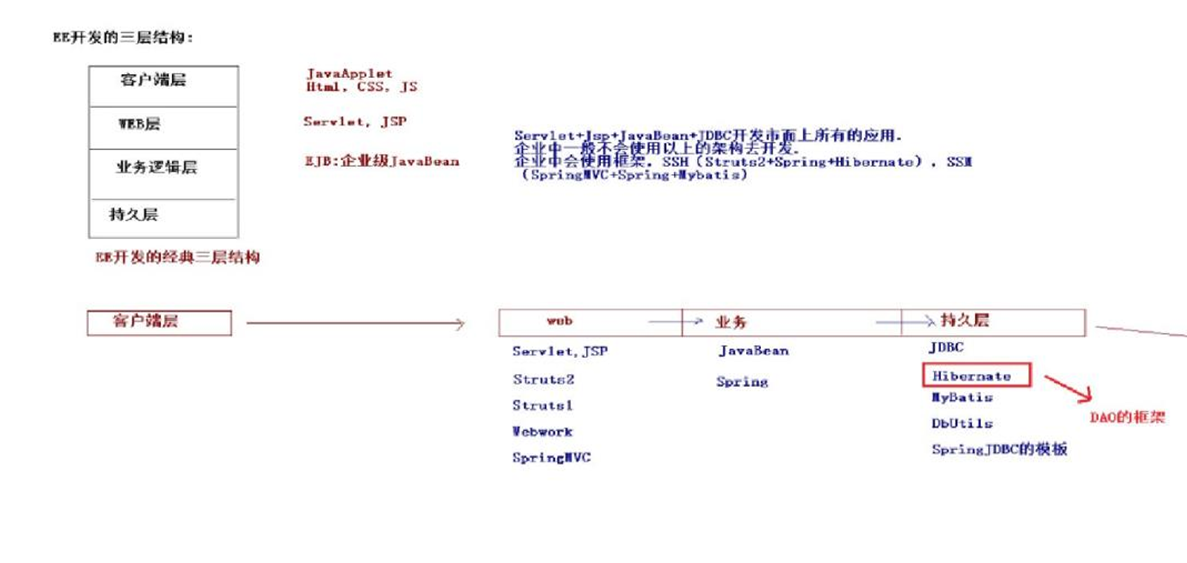
java解析JSON JSONlib Gson（谷歌） FastJson(阿里巴巴开源)

监听器过滤器 分页（普通的分页 Ajax无刷新的分页）



# Hibernate框架

JavaEE三层开发结构：



## ORM 规则

ORM 对象关系映射（英语：**(Object Relational Mapping**，简称**ORM**，或**O/RM**，或**O/R mapping**），是一种程序技术，用于实现面向对象编程语言里不同类型系统的数据之间的转换

\* O：面向对象领域的Object（JavaBean对象）

\* R：关系数据库领域的Relational（表的结构）

\* M：映射Mapping（XML的配置文件）

简单一句话：Hibernate使程序员通过操作对象的方式来操作数据库表记录

目标：

通过orm（对象关系映射）, 可以做到：

a. 不写sql，包对象直接保存到数据库中

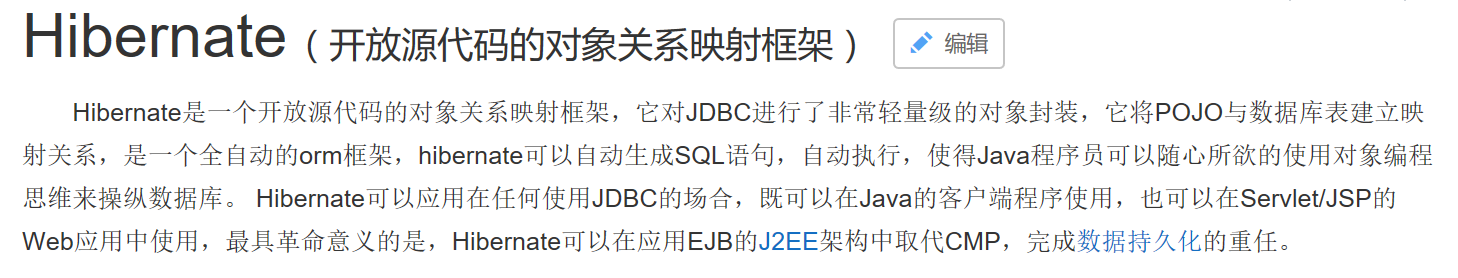
b. 不写sql， 从数据库中读取，直接获取的是封装好的对象！

ORM是一种思想，或者说是一种规则的定义， 基于orm的应用：

1. hibernate 框架

2. mybatis 框架

## Hibernate框架



### 概念

基于ORM的持久层框架，对jdbc操作进行了封装！

Hibernate与具体数据库无关的技术，即可以跨数据库平台！

**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

\* 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 种复杂关系

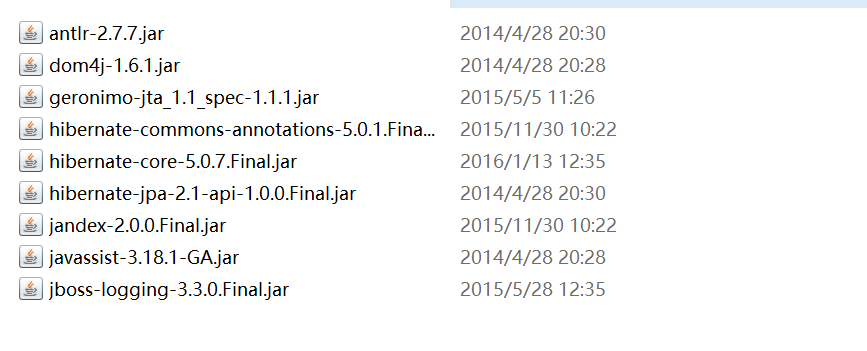
### Hibernate开发环境搭建

Hibernate版本：hibernate-release-5.0.7.Final

整体步骤：

1. 下载源码，引入jar文件





Log4j日志的三个包



数据库驱动包：



2. 写配置

\* hibernate.cfg.xml 主配置

\* Users.hbm.xml 映射配置

3.Api

Configuration

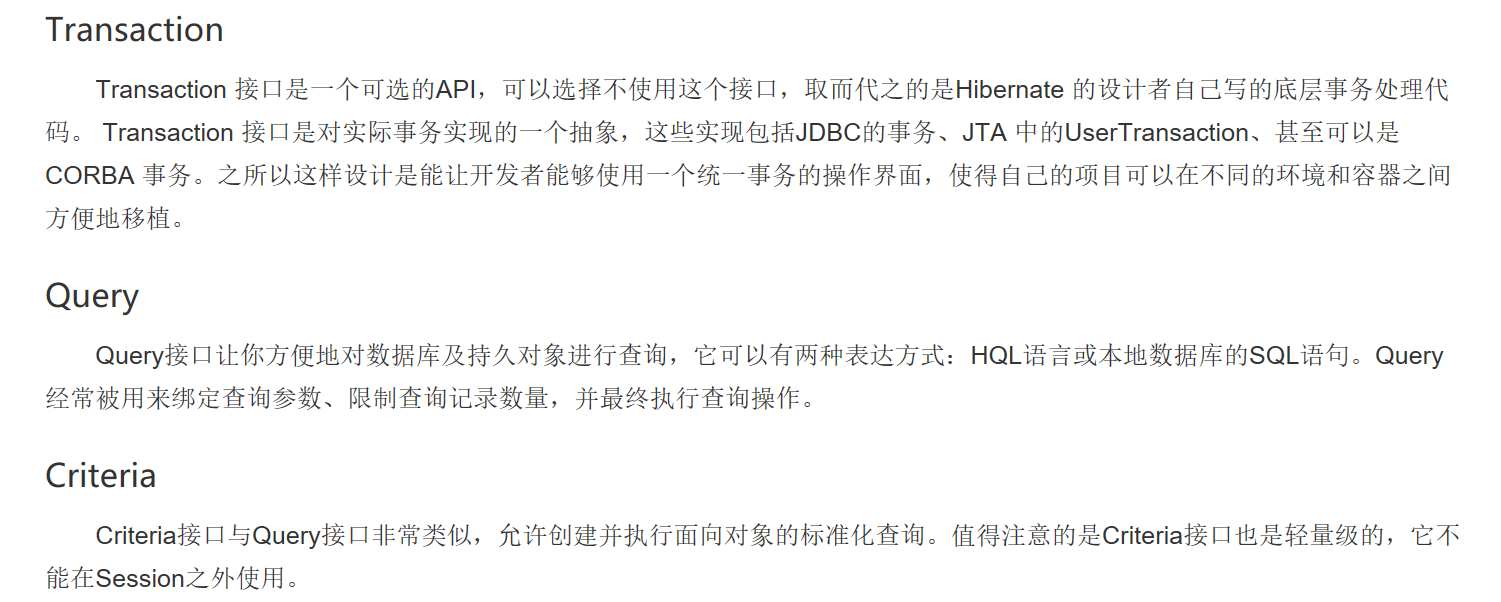
SessionFactory

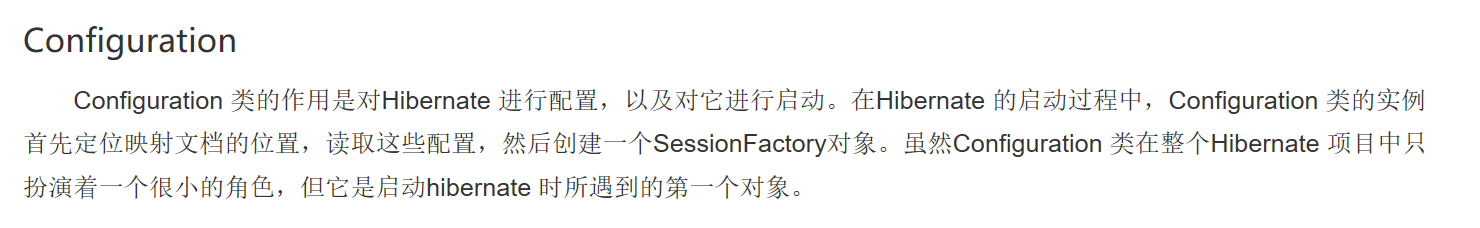
Transaction

Session

Query







## Hibernate快速入门

### 第一步：下载Hibernate5的运行环境

1. 下载相应的jar包等

http://sourceforge.net/projects/hibernate/files/hibernate-orm/5.0.7.Final/hibernate-release-5.0.7.Final.zip/download

2. 解压后对目录结构有一定的了解

### 第二步：创建表结构

1. 建表语句如下

|  |
| --- |
| Create database hibernate\_day01;  Use hibernate\_day01;  CREATE TABLE `cst\_customer` (  `cust\_id` bigint(32) NOT NULL AUTO\_INCREMENT COMMENT '客户编号(主键)',  `cust\_name` varchar(32) NOT NULL COMMENT '客户名称(公司名称)',  `cust\_user\_id` bigint(32) DEFAULT NULL COMMENT '负责人id',  `cust\_create\_id` bigint(32) DEFAULT NULL COMMENT '创建人id',  `cust\_source` varchar(32) DEFAULT NULL COMMENT '客户信息来源',  `cust\_industry` varchar(32) DEFAULT NULL COMMENT '客户所属行业',  `cust\_level` varchar(32) DEFAULT NULL COMMENT '客户级别',  `cust\_linkman` varchar(64) DEFAULT NULL COMMENT '联系人',  `cust\_phone` varchar(64) DEFAULT NULL COMMENT '固定电话',  `cust\_mobile` varchar(16) DEFAULT NULL COMMENT '移动电话',  PRIMARY KEY (`cust\_id`)  ) ENGINE=InnoDB AUTO\_INCREMENT=94 DEFAULT CHARSET=utf8; |

第三步：搭建Hibernate的开发环境

1. 创建WEB工程，引入Hibernate开发所需要的jar包

a. MySQL的驱动jar包

b.Hibernate开发需要的jar包（资料/hibernate-release-5.0.7.Final/lib/required/所有jar包）

c. 日志jar包（资料/jar包/log4j/所有jar包）

### 第三步：编写JavaBean实体类

1. Customer类的代码如下：

public class Customer {

private Long cust\_id;

private String cust\_name;

private Long cust\_user\_id;

private Long cust\_create\_id;

private String cust\_source;

private String cust\_industry;

private String cust\_level;

private String cust\_linkman;

private String cust\_phone;

private String cust\_mobile;

// 省略get和set方法

}

### 第四步：创建类与表结构的映射

**1. 在JavaBean所在的包下创建映射的配置文件**

\* 默认的命名规则为：实体类名.hbm.xml

\* 在xml配置文件中引入约束（引入的是hibernate3.0的dtd约束，不要引入4的约束）

|  |
| --- |
| 映射的dtd文件：  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd"> |

**2. 如果不能上网，编写配置文件是没有提示的，需要自己来配置**

先复制http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd --> window --> preferences --> 搜索xml --> 选择xml catalog --> 点击add --> 现在URI --> 粘贴复制的地址 --> 选择location，选择本地的DTD的路径

**3. 编写映射的配置文件**

|  |
| --- |
| <?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>  <class name="com.qf.model.Customer" table="cst\_customer">  <id name="cust\_id" column="cust\_id">  <generator class="native"/>  </id>  <property name="cust\_name" column="cust\_name"/>  <property name="cust\_user\_id" column="cust\_user\_id"/>  <property name="cust\_create\_id" column="cust\_create\_id"/>  <property name="cust\_source" column="cust\_source"/>  <property name="cust\_industry" column="cust\_industry"/>  <property name="cust\_level" column="cust\_level"/>  <property name="cust\_linkman" column="cust\_linkman"/>  <property name="cust\_phone" column="cust\_phone"/>  <property name="cust\_mobile" column="cust\_mobile"/>  </class>  </hibernate-mapping> |

### 第五步：编写Hibernate核心的配置文件

**1. 在src目录下，创建名称为hibernate.cfg.xml的配置文件**

**2. 在XML中引入DTD约束**

|  |
| --- |
| <!DOCTYPE hibernate-configuration PUBLIC  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd"> |

**3. 打开：资料/hibernate-release-5.0.7.Final/project/etc/hibernate.properties，可以查看具体的配置信息**

a.必须配置的4大参数

#hibernate.connection.driver\_class com.mysql.jdbc.Driver

#hibernate.connection.url jdbc:mysql:///test

#hibernate.connection.username gavin

#hibernate.connection.password

b.数据库的方言（必须配置的）

#hibernate.dialect org.hibernate.dialect.MySQLDialect

c. 可选的配置

#hibernate.show\_sql true

#hibernate.format\_sql true

#hibernate.hbm2ddl.auto update

d.引入映射配置文件（一定要注意，要引入映射文件，框架需要加载映射文件）

\* <mapping resource="com/qf/ model /Customer.hbm.xml"/>

**4. 具体的配置如下**

|  |
| --- |
| <?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>  <property name="hibernate.connection.driver\_class">com.mysql.jdbc.Driver</property>  <property name="hibernate.connection.url">jdbc:mysql:///hibernate\_day01</property>  <property name="hibernate.connection.username">root</property>  <property name="hibernate.connection.password">root</property>  <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>    <mapping resource="com/qf/model/Customer.hbm.xml"/>  </session-factory>  </hibernate-configuration> |

### 第六步：编写Hibernate入门代码

1. 具体的代码如下

|  |
| --- |
| /\*\*  \* 测试保存客户  \*/  @Test  public void testSave(){  // 先加载配置文件  Configuration config = new Configuration();  // 默认加载src目录下的配置文件  config.configure();  // 创建SessionFactory对象  SessionFactory factory = config.buildSessionFactory();  // 创建session对象  Session session = factory.openSession();  // 开启事务  Transaction tr = session.beginTransaction();  // 编写保存代码  Customer c = new Customer();  // c.setCust\_id(cust\_id); 已经自动递增  c.setCust\_name("测试名称");  c.setCust\_mobile("110");  // 保存客户  session.save(c);  // 提交事务  tr.commit();  // 释放资源  session.close();  factory.close();  } |

### 总结快速入门之6步骤

1. 下载Hibernate框架的开发包，创建WEB的项目，导入了开发的jar包

MySQL驱动包、Hibernate开发的必须要有的jar包、日志的jar包

2. 编写数据库和表结构

3. 编写JavaBean，以后不使用基本数据类型，使用包装类

4. 编写映射的配置文件（核心），先导入开发的约束，里面正常配置标签

5. 编写hibernate的核心的配置文件，里面的内容是固定的

6. 编写代码，使用的类和方法

## 技术分析之：Hibernate常用的配置文件

### Hibernate配置文件之映射配置文件

1. 映射文件，即Stu.hbm.xml的配置文件

<hibernate-mapping package="model" >

如果配置 package，name的取值可以是简单类名 name="Person"

\* <class>标签 -- 用来将类与数据库表建立映射关系

\* name -- 类的全路径

\* table -- 表名.(类名与表名一致,那么table属性也可以省略)

\* catalog -- 数据库的名称，基本上都会省略不写

\*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。

\* <id>标签 -- 用来将类中的属性与表中的主键建立映射，id标签就是用来配置主键的。

\* name -- 类中属性名

\* column -- 表中的字段名.(如果类中的属性名与表中的字段名一致,那么column可以省略.)

\* length -- 字段的程度，如果数据库已经创建好了，那么length可以不写。如果没有创建好，生成表结构时，length最好指定。

\* <property> -- 用来将类中的普通属性与表中的字段建立映射.

\* name -- 类中属性名

\* column -- 表中的字段名.(如果类中的属性名与表中的字段名一致,那么column可以省略.)

\* length -- 数据长度

\* 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

\* Hibernate的数据类型 type="string"

\* Java的数据类型 type="java.lang.String"

\* 数据库字段的数据类型 <column name="name" sql-type="varchar"/>

#### 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 |

### Hibernate配置文件之核心配置文件

#### 1. 核心配置文件的两种方式

a. 第一种方式是属性文件的形式，即properties的配置文件

hibernate.properties

hibernate.connection.driver\_class=com.mysql.jdbc.Driver

缺点

不能配置加载映射的配置文件，需要手动编写代码去加载

b．第二种方式是XML文件的形式，开发都会选择这种方式(使用)

hibernate.cfg.xml

<property name="hibernate.connection.driver\_class" >com.mysql.jdbc.Driver</property>

优点

格式比较清晰

编写有提示

可以在该配置文件中加载映射的配置文件（最主要的）

#### 2. 关于hibernate.cfg.xml的配置文件方式

必须有的配置

数据库连接信息:

hibernate.connection.driver\_class -- 连接数据库驱动程序

hibernate.connection.url -- 连接数据库URL

hibernate.connection.username -- 数据库用户名

hibernate.connection.password -- 数据库密码

方言:

hibernate.dialect -- 操作数据库方言

可选的配置

hibernate.show\_sql -- 显示SQL

hibernate.format\_sql -- 格式化SQL

hibernate.hbm2ddl.auto -- 通过映射转成DDL语句

create -- 每次都会创建一个新的表.---测试的时候

create-drop -- 每次都会创建一个新的表,当执行结束之后,将创建的这个表删除.---测试的时候

update -- 如果有表,使用原来的表.没有表,创建一个新的表.同时更新表结构.

validate -- 如果有表,使用原来的表.同时校验映射文件与表中字段是否一致如果不一致就会报错.

加载映射

如果XML方式：<mapping resource="cn/qf/hibernate/model/User.hbm.xml" />

## 技术分析之Hibernate常用的接口和类

### Configuration类和作用

1. Configuration类

Configuration对象用于配置并且启动Hibernate。

Hibernate应用通过该对象来获得对象-关系映射文件中的元数据，以及动态配置Hibernate的属性，然后创建SessionFactory对象。

简单一句话：加载Hibernate的配置文件，可以获取SessionFactory对象。

**2. Configuration类的其他应用（了解）**

加载配置文件的种类，Hibernate支持xml和properties类型的配置文件，在开发中基本都使用XML配置文件的方式。

如果采用的是properties的配置文件，那么通过Configuration configuration = new Configuration();就可以加载配置文件

但是需要自己手动加载映射文件

例如：config.addResource("cn/qf/model/Student.hbm.xml");

如果采用的XML的配置文件，通过Configuration configuration = new Configuration().configure();加载配置文件

### SessionFactory：重要

（二级缓存 被redis缓存数据库给代替了。）

**1. 是工厂类，是生成Session对象的工厂类**

**2. SessionFactory类的特点**

由Configuration通过加载配置文件创建该对象。

SessionFactory对象中保存了当前的数据库配置信息和所有映射关系以及预定义的SQL语句。同时，SessionFactory还负责维护Hibernate的二级缓存。

预定义SQL语句

使用Configuration类创建了SessionFactory对象是，已经在SessionFacotry对象中缓存了一些SQL语句

常见的SQL语句是增删改查（通过主键来查询）

这样做的目的是效率更高

一个SessionFactory实例对应一个数据库，应用从该对象中获得Session实例。

SessionFactory是线程安全的，意味着它的一个实例可以被应用的多个线程共享。

SessionFactory是重量级的，意味着不能随意创建或销毁它的实例。如果只访问一个数据库，只需要创建一个SessionFactory实例，且在应用初始化的时候完成。

SessionFactory需要一个较大的缓存，用来存放预定义的SQL语句及实体的映射信息。另外可以配置一个缓存插件，这个插件被称之为Hibernate的二级缓存，被多线程所共享

**3. 总结**

一般应用使用一个SessionFactory,最好是应用启动时就完成初始化。

**编写HibernateUtil的工具类**

1. 具体代码如下

|  |
| --- |
| public class HibernateUtil {  private static final Configuration cfg;  private static final SessionFactory factory;  static{  // 给常量赋值  // 加载配置文件  cfg = new Configuration().configure();  // 生成factory对象  factory = cfg.buildSessionFactory();  }  // 获取Session对象  public static Session openSession(){  return factory.openSession();  }  } |

### Session接口

**1. 概述**

Session是在Hibernate中使用最频繁的接口。也被称之为持久化管理器。它提供了和持久化有关的操作，比如添加、修改、删除、加载和查询实体对象

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 运作的中心

Session是线程不安全的

所有持久化对象必须在 session 的管理下才可以进行持久化操作

Session 对象有一个一级缓存，显式执行 flush 之前，所有的持久化操作的数据都缓存在 session 对象处

持久化类与 Session 关联起来后就具有了持久化的能力

**2. 特点**

不是线程安全的。应避免多个线程使用同一个Session实例

Session是轻量级的，它的创建和销毁不会消耗太多的资源。应为每次客户请求分配独立的Session实例

Session有一个缓存，被称之为Hibernate的一级缓存。每个Session实例都有自己的缓存

**3. 常用的方法**

save(obj)保存一个对象

delete(obj) 删除一个对象

get(Class,id) 根据id查询

update(obj) 更新对象

saveOrUpdate(obj) -- 保存或者修改（如果没有数据，保存数据。如果有，修改数据）

createQuery() -- HQL语句的查询的方式

### Transaction接口

**1. Transaction是事务的接口**

**2. 常用的方法**

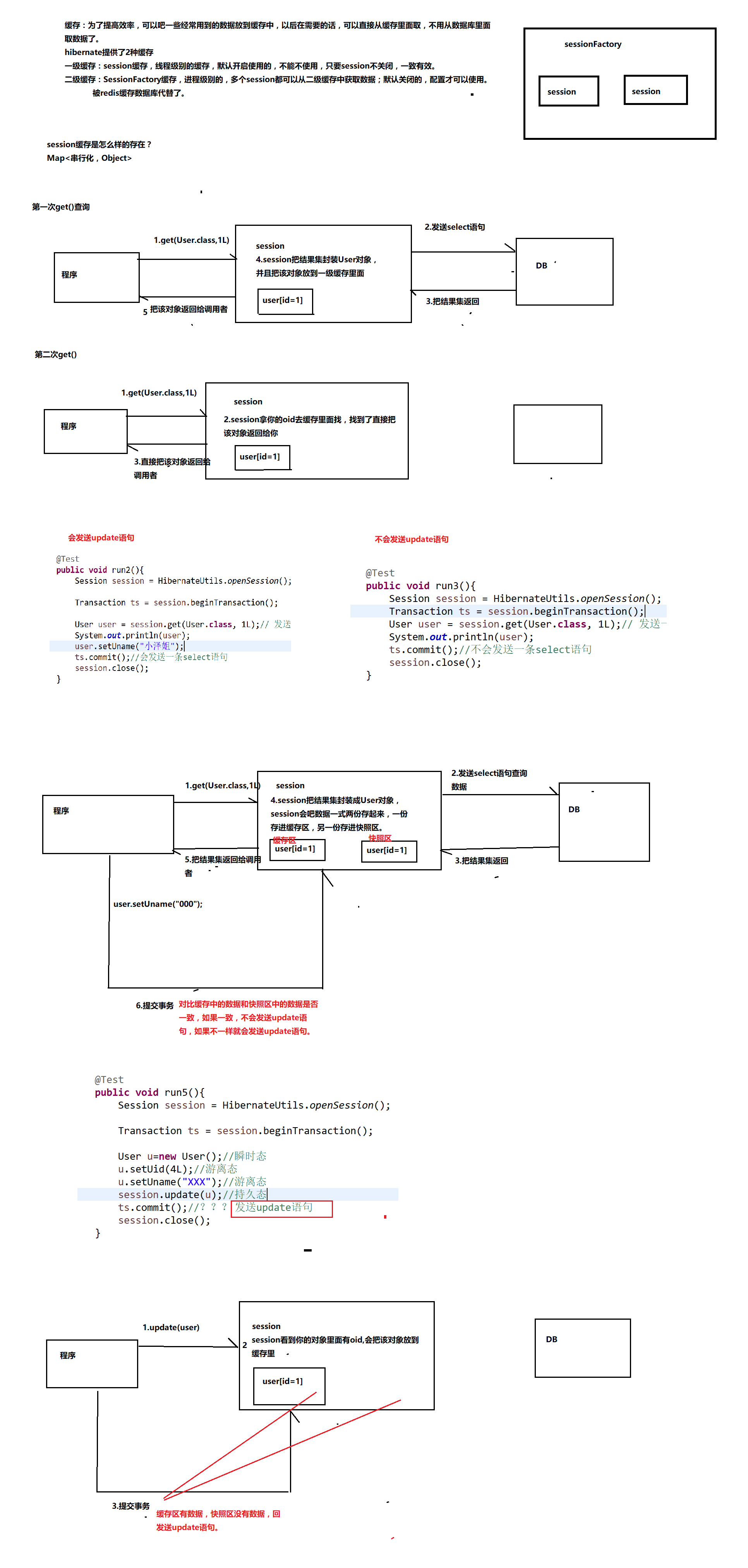
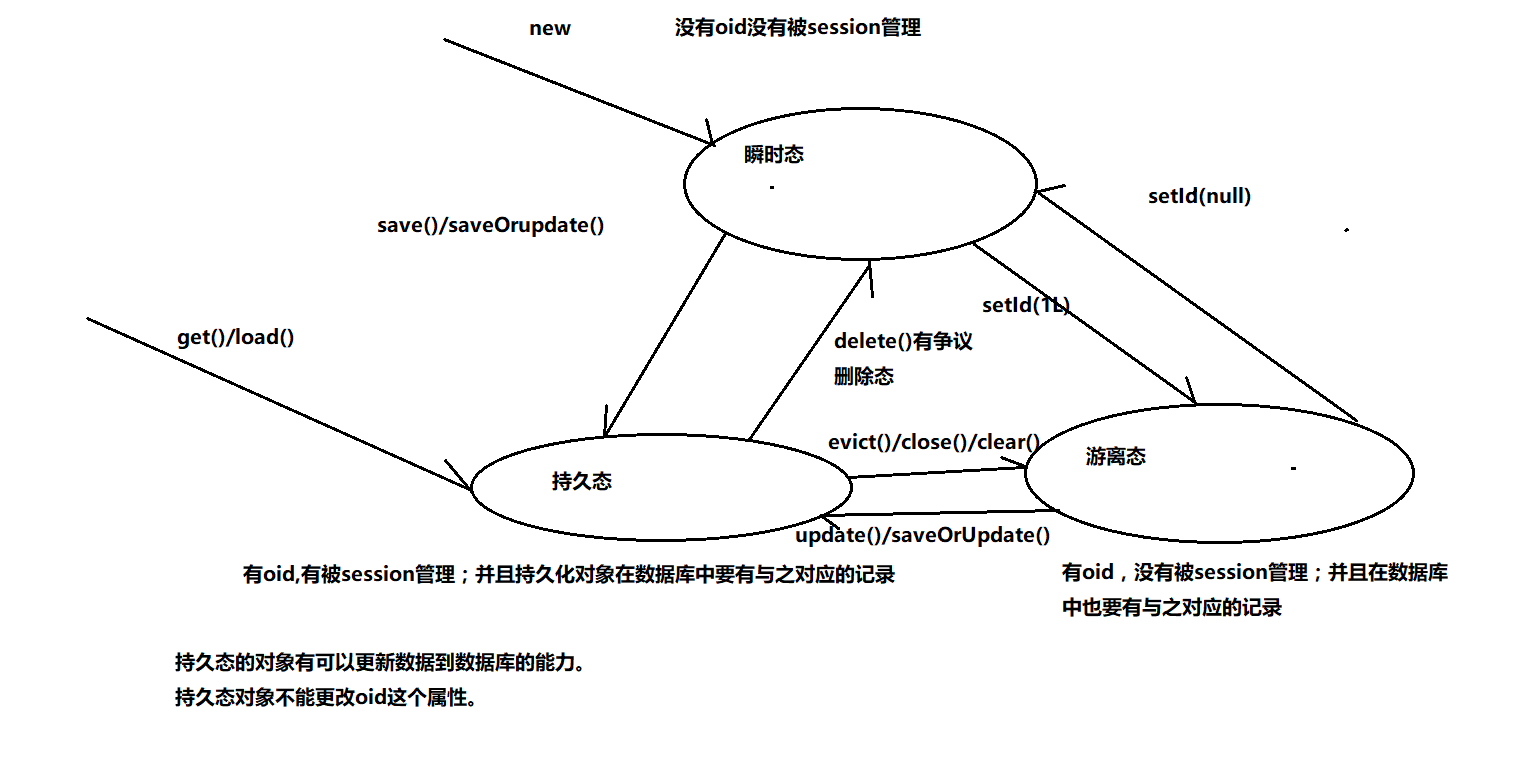
commit() -- 提交事务

rollback() -- 回滚事务

**3. 特点**

Hibernate框架默认情况下事务不自动提交.需要手动提交事务

如果没有开启事务，那么每个Session的操作，都相当于一个独立的事务



## Hibernate的持久化类

### 什么是持久化类

1. 持久化类:就是一个Java类（咱们编写的JavaBean），这个Java类与表建立了映射关系就可以成为是持久化类。

持久化类 = JavaBean + xxx.hbm.xml

### 持久化类的编写规则

1. 提供一个无参数 public访问控制符的构造器 -- 底层需要进行反射.

2. 提供一个标识属性，映射数据表主键字段 -- 唯一标识OID.数据库中通过主键.Java对象通过地址确定对象.持久化类通过唯一标识OID确定记录

3. 所有属性提供public访问控制符的 set或者get 方法

4. 标识属性应尽量使用基本数据类型的包装类型

### 区分自然主键和代理主键

1. 创建表的时候

自然主键:对象本身的一个属性.创建一个人员表,每个人都有一个身份证号.(唯一的)使用身份证号作为表的主键.自然主键.（开发中不会使用这种方式）

代理主键:不是对象本身的一个属性.创建一个人员表,为每个人员单独创建一个字段.用这个字段作为主键.代理主键.（开发中推荐使用这种方式）

2. 创建表的时候尽量使用代理主键创建表

### 主键的生成策略

1. increment:适用于short,int,long作为主键.不是使用的数据库自动增长机制.

Hibernate中提供的一种增长机制.

先进行查询 :select max(id) from user;

再进行插入 :获得最大值+1作为新的记录的主键.

问题:不能在集群环境下或者有并发访问的情况下使用.

2. identity:适用于short,int,long作为主键。但是这个必须使用在有自动增长数据库中.采用的是数据库底层的自动增长机制.

底层使用的是数据库的自动增长(auto\_increment).像Oracle数据库没有自动增长.

3. sequence:适用于short,int,long作为主键.底层使用的是序列的增长方式.

Oracle数据库底层没有自动增长,想自动增长需要使用序列.

4. uuid:适用于char,varchar类型的作为主键.

使用随机的字符串作为主键.

5. native:本地策略.根据底层的数据库不同,自动选择适用于该种数据库的生成策略.(short,int,long)

如果底层使用的MySQL数据库:相当于identity.

如果底层使用Oracle数据库:相当于sequence.

6. assigned:主键的生成不用Hibernate管理了.必须手动设置主键.

7.hilo hibernate自己搞得算法

## Hibernate持久化对象的状态

### 持久化对象的状态

**1. Hibernate的持久化类**

持久化类:Java类与数据库的某个表建立了映射关系.这个类就称为是持久化类.

持久化类 = Java类 + hbm的配置文件

**2. Hibernate的持久化类的状态**

Hibernate为了管理持久化类：将持久化类分成了三个状态

瞬时态:Transient Object

没有持久化标识OID, 没有被纳入到Session对象的管理.

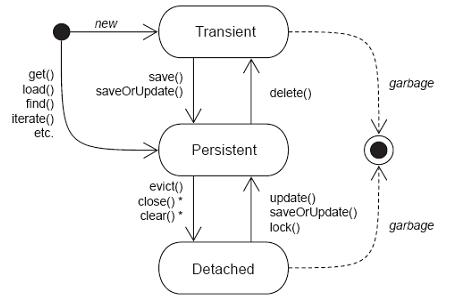
持久态:Persistent Object

有持久化标识OID,已经被纳入到Session对象的管理.

脱管态/游离态:Detached Object

有持久化标识OID,没有被纳入到Session对象的管理.

### Hibernate持久化对象的状态的转换



**1. 瞬时态 -- 没有持久化标识OID, 没有被纳入到Session对象的管理**

获得瞬时态的对象

User user = new User()

瞬时态对象转换持久态

\* save()/saveOrUpdate();

瞬时态对象转换成脱管态

\* user.setId(1)

**2. 持久态 -- 有持久化标识OID,已经被纳入到Session对象的管理**

获得持久态的对象

get()/load();

持久态转换成瞬时态对象

delete(); --- 比较有争议的，进入特殊的状态(删除态:Hibernate中不建议使用的)

持久态对象转成脱管态对象

\* session的close()/evict()/clear();

**3. 脱管态 -- 有持久化标识OID,没有被纳入到Session对象的管理**

获得托管态对象:不建议直接获得脱管态的对象.

User user = new User();

user.setId(1);

脱管态对象转换成持久态对象

update();/saveOrUpdate()/lock();

脱管态对象转换成瞬时态对象

user.setId(null);

**4. 注意：持久态对象有自动更新数据库的能力!!!**

**5.注意：hibernate持久态对象不允许更新oid属性。**

## Hibernate的一级缓存

### Session对象的一级缓存（重点）\*\*

**1. 什么是缓存？**

其实就是一块内存空间,将数据源（数据库或者文件）中的数据存放到缓存中.再次获取的时候 ,直接从缓存中获取.可以提升程序的性能！

**2. Hibernate框架提供了两种缓存**

一级缓存 -- 自带的不可卸载的.一级缓存的生命周期与session一致.一级缓存称为session级别的缓存.

二级缓存 -- 默认没有开启，需要手动配置才可以使用的.二级缓存可以在多个session中共享数据,二级缓存称为是sessionFactory级别的缓存.

**3. Session对象的缓存概述**

Session接口中,有一系列的java的集合,这些java集合构成了Session级别的缓存(一级缓存).将对象存入到一级缓存中,session没有结束生命周期,那么对象在session中存放着

内存中包含Session实例 --> Session的缓存（一些集合） --> 集合中包含的是缓存对象！

**4. 证明一级缓存的存在，编写查询的代码即可证明**

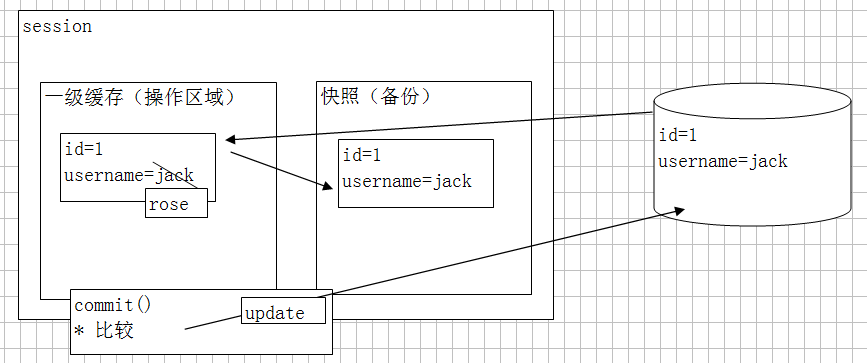
在同一个Session对象中两次查询，可以证明使用了缓存

**5. Hibernate框架是如何做到数据发生变化时进行同步操作的呢？**

使用get方法查询User对象

然后设置User对象的一个属性，注意：没有做update操作。发现，数据库中的记录也改变了。

**利用快照机制来完成的（SnapShot）**



**控制Session的一级缓存（了解）**

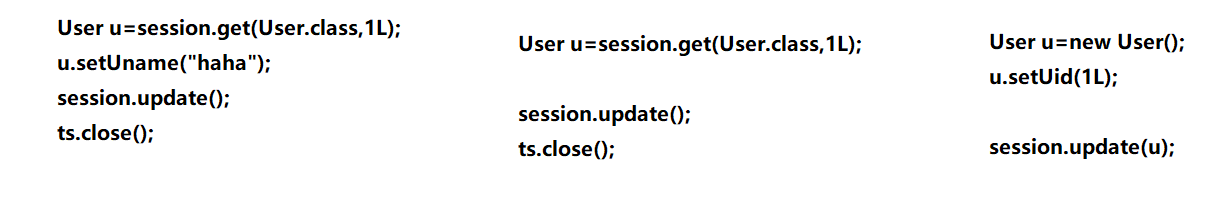
1. 学习Session接口中与一级缓存相关的方法

Session.clear() -- 清空缓存。

Session.evict(Object entity) -- 从一级缓存中清除指定的实体对象。

Session.flush() -- 刷出缓存

案例：演示看效果



## Hibernate框架的查询方式

### Query查询接口

1. 具体的查询代码如下

|  |
| --- |
| // 1.查询所有记录  /\*Query query = session.createQuery("from Customer");  List<Customer> list = query.list();  System.out.println(list);\*/    // 2.条件查询:  /\*Query query = session.createQuery("from Customer where name = ?");  query.setString(0, "李健");  List<Customer> list = query.list();  System.out.println(list);\*/    // 3.条件查询:  /\*Query query = session.createQuery("from Customer where name = :aaa and age = :bbb");  query.setString("aaa", "李健");  query.setInteger("bbb", 38);  List<Customer> list = query.list();  System.out.println(list);\*/ |

### Criteria查询接口（做条件查询非常合适）

\* 条件查询使用Criteria接口的add方法，用来传入条件。

\* 使用Restrictions的添加条件的方法，来添加条件，例如：

\* Restrictions.eq -- 相等

\* Restrictions.gt -- 大于号

\* Restrictions.ge -- 大于等于

\* Restrictions.lt -- 小于

\* Restrictions.le -- 小于等于

\* Restrictions.between -- 在之间

\* Restrictions.like -- 模糊查询

\* Restrictions.in -- 范围

\* Restrictions.and -- 并且

\* Restrictions.or -- 或者

\* 测试代码如下

Session session = HibernateUtils.getCurrentSession();

Transaction tr = session.beginTransaction();

Criteria criteria = session.createCriteria(Linkman.class);

// 设置排序

criteria.addOrder(Order.desc("lkm\_id"));

// 设置查询条件

criteria.add(Restrictions.or(Restrictions.eq("lkm\_gender", "男"), Restrictions.gt("lkm\_id", 3L)));

List<Linkman> list = criteria.list();

for (Linkman linkman : list) {

System.out.println(linkman);

}

tr.commit();

1. 具体的查询代码如下

|  |
| --- |
| // 1.查询所有记录  /\*Criteria criteria = session.createCriteria(Customer.class);  List<Customer> list = criteria.list();  System.out.println(list);\*/    // 2.条件查询  /\*Criteria criteria = session.createCriteria(Customer.class);  criteria.add(Restrictions.eq("name", "李健"));  List<Customer> list = criteria.list();  System.out.println(list);\*/    // 3.条件查询  /\*Criteria criteria = session.createCriteria(Customer.class);  criteria.add(Restrictions.eq("name", "李健"));  criteria.add(Restrictions.eq("age", 38));  List<Customer> list = criteria.list();  System.out.println(list);\*/ |

## 绑定本地的Session

1. 之前在讲JavaWEB的事务的时候，需要在业务层使用Connection来开启事务，

一种是通过参数的方式传递下去

另一种是把Connection绑定到ThreadLocal对象中

2. 现在的Hibernate框架中，使用session对象开启事务，所以需要来传递session对象，框架提供了ThreadLocal的方式

需要在hibernate.cfg.xml的配置文件中提供配置

<property name="hibernate.current\_session\_context\_class">thread</property>

重新HibernateUtil的工具类，使用SessionFactory的getCurrentSession()方法，获取当前的Session对象。并且该Session对象不用手动关闭，线程结束了，会自动关闭。

public static Session getCurrentSession(){

return factory.getCurrentSession();

}

**注意：想使用getCurrentSession()方法，必须要先配置才能使用。**

## Hibernate中的事务与并发

### 事务相关的概念

#### 1. 什么是事务

事务就是逻辑上的一组操作，组成事务的各个执行单元，操作要么全都成功，要么全都失败.

转账的例子：赵信给狐狸转钱，扣钱，加钱。两个操作组成了一个事情！

#### 2. 事务的特性

原子性 -- 事务不可分割.

一致性 -- 事务执行的前后数据的完整性保持一致.

隔离性 -- 一个事务执行的过程中,不应该受到其他的事务的干扰.

持久性 -- 事务一旦提交,数据就永久保持到数据库中.

#### 3. 如果不考虑隔离性:引发一些读的问题

脏读 -- 一个事务读到了另一个事务未提交的数据.

不可重复读 -- 一个事务读到了另一个事务已经提交的update数据,导致多次查询结果不一致.

虚读 -- 一个事务读到了另一个事务已经提交的insert数据,导致多次查询结构不一致.

#### 4. 通过设置数据库的隔离级别来解决上述读的问题

未提交读:以上的读的问题都有可能发生.

已提交读:避免脏读,但是不可重复读，虚读都有可能发生.

可重复读:避免脏读，不可重复读.但是虚读是有可能发生.

串行化:以上读的情况都可以避免.

#### 5. 如果想在Hibernate的框架中来设置隔离级别，需要在hibernate.cfg.xml的配置文件中通过标签来配置

通过：hibernate.connection.isolation = 4 来配置

取值

1—Read uncommitted isolation

2—Read committed isolation

4—Repeatable read isolation

8—Serializable isolation

### 丢失更新的问题

1. 如果不考虑隔离性，也会产生写入数据的问题，这一类的问题叫丢失更新的问题。

2. 例如：两个事务同时对某一条记录做修改，就会引发丢失更新的问题。

A事务和B事务同时获取到一条数据，同时再做修改

如果A事务修改完成后，提交了事务

B事务修改完成后，不管是提交还是回滚，如果不做处理，都会对数据产生影响

3. 解决方案有两种

**悲观锁**

采用的是数据库提供的一种锁机制，如果采用做了这种机制，在SQL语句的后面添加 for update 子句

当A事务在操作该条记录时，会把该条记录锁起来，其他事务是不能操作这条记录的。

只有当A事务提交后，锁释放了，其他事务才能操作该条记录

**乐观锁**

采用版本号的机制来解决的。会给表结构添加一个字段version=0，默认值是0

当A事务在操作完该条记录，提交事务时，会先检查版本号，如果发生版本号的值相同时，才可以提交事务。同时会更新版本号version=1.

当B事务操作完该条记录时，提交事务时，会先检查版本号，如果发现版本不同时，程序会出现错误。

4. 使用Hibernate框架解决丢失更新的问题

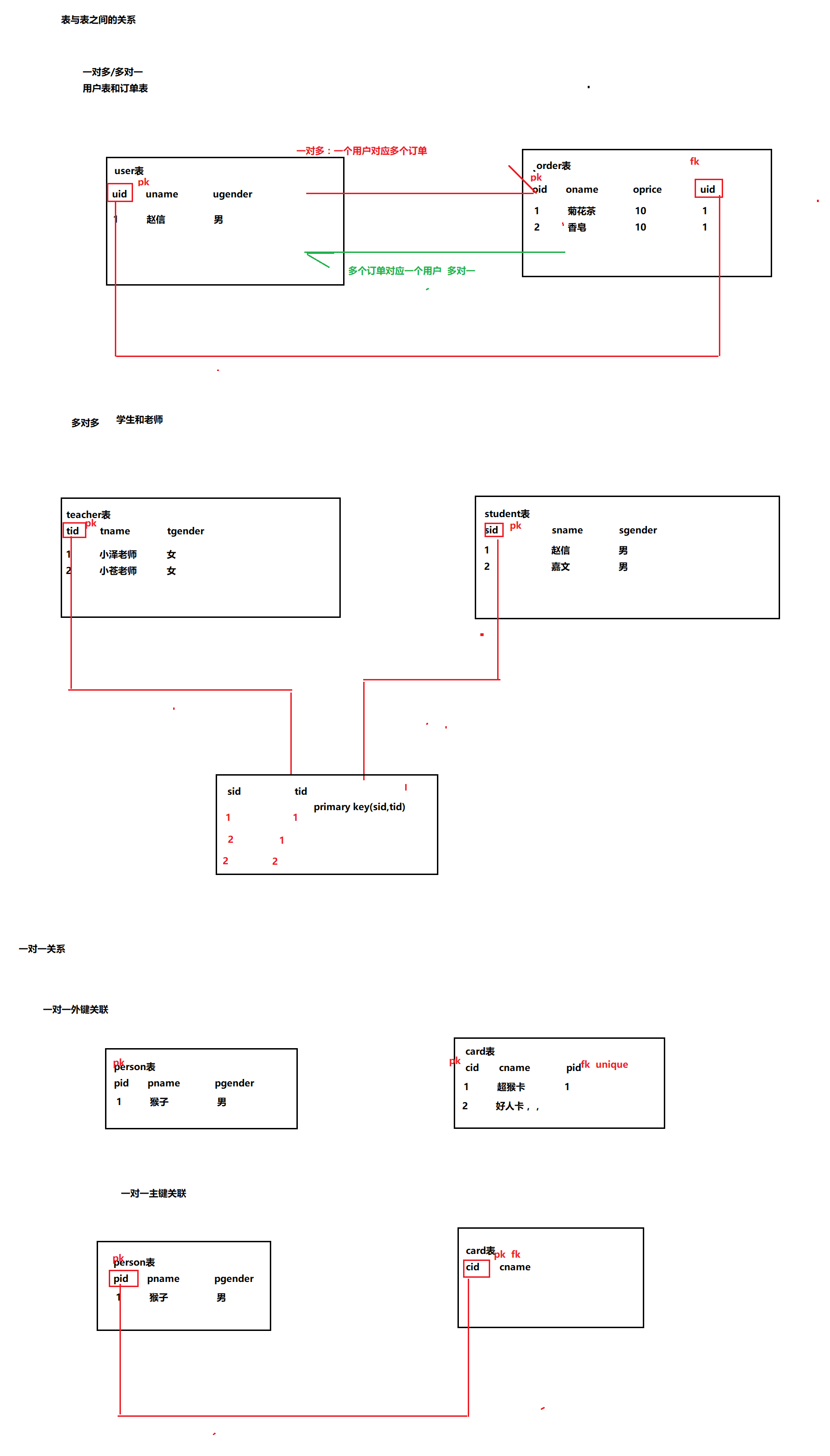
悲观锁

使用session.get(Customer.class, 1,LockMode.UPGRADE); 方法

乐观锁

1.在对应的JavaBean中添加一个属性，名称可以是任意的。例如：private Integer version; 提供get和set方法

2.在映射的配置文件中，提供<version name="version"/>标签即可。



## Hibernate的持久化类

### 什么是持久化类

1. 持久化类:就是一个Java类（咱们编写的JavaBean），这个Java类与表建立了映射关系就可以成为是持久化类。

持久化类 = JavaBean + xxx.hbm.xml

### 持久化类的编写规则

1. 提供一个无参数 public访问控制符的构造器 -- 底层需要进行反射.

2. 提供一个标识属性，映射数据表主键字段 -- 唯一标识OID.数据库中通过主键.Java对象通过地址确定对象.持久化类通过唯一标识OID确定记录

3. 所有属性提供public访问控制符的 set或者get 方法

4. 标识属性应尽量使用基本数据类型的包装类型

### 区分自然主键和代理主键

1. 创建表的时候

自然主键:对象本身的一个属性.创建一个人员表,每个人都有一个身份证号.(唯一的)使用身份证号作为表的主键.自然主键.（开发中不会使用这种方式）

代理主键:不是对象本身的一个属性.创建一个人员表,为每个人员单独创建一个字段.用这个字段作为主键.代理主键.（开发中推荐使用这种方式）

2. 创建表的时候尽量使用代理主键创建表

### 主键的生成策略

1. increment:适用于short,int,long作为主键.不是使用的数据库自动增长机制.

Hibernate中提供的一种增长机制.

先进行查询 :select max(id) from user;

再进行插入 :获得最大值+1作为新的记录的主键.

问题:不能在集群环境下或者有并发访问的情况下使用.

2. identity:适用于short,int,long作为主键。但是这个必须使用在有自动增长数据库中.采用的是数据库底层的自动增长机制.

底层使用的是数据库的自动增长(auto\_increment).像Oracle数据库没有自动增长.

3. sequence:适用于short,int,long作为主键.底层使用的是序列的增长方式.

Oracle数据库底层没有自动增长,想自动增长需要使用序列.

4. uuid:适用于char,varchar类型的作为主键.

使用随机的字符串作为主键.

5. native:本地策略.根据底层的数据库不同,自动选择适用于该种数据库的生成策略.(short,int,long)

如果底层使用的MySQL数据库:相当于identity.

如果底层使用Oracle数据库:相当于sequence.

6. assigned:主键的生成不用Hibernate管理了.必须手动设置主键.

7.hilo hibernate自己搞得算法

## Hibernate持久化对象的状态

### 持久化对象的状态

**1. Hibernate的持久化类**

持久化类:Java类与数据库的某个表建立了映射关系.这个类就称为是持久化类.

持久化类 = Java类 + hbm的配置文件

**2. Hibernate的持久化类的状态**

Hibernate为了管理持久化类：将持久化类分成了三个状态

瞬时态:Transient Object

没有持久化标识OID, 没有被纳入到Session对象的管理.

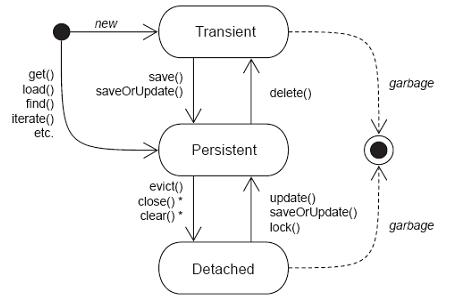
持久态:Persistent Object

有持久化标识OID,已经被纳入到Session对象的管理.

脱管态/游离态:Detached Object

有持久化标识OID,没有被纳入到Session对象的管理.

### Hibernate持久化对象的状态的转换



**1. 瞬时态 -- 没有持久化标识OID, 没有被纳入到Session对象的管理**

获得瞬时态的对象

User user = new User()

瞬时态对象转换持久态

\* save()/saveOrUpdate();

瞬时态对象转换成脱管态

\* user.setId(1)

**2. 持久态 -- 有持久化标识OID,已经被纳入到Session对象的管理**

获得持久态的对象

get()/load();

持久态转换成瞬时态对象

delete(); --- 比较有争议的，进入特殊的状态(删除态:Hibernate中不建议使用的)

持久态对象转成脱管态对象

\* session的close()/evict()/clear();

**3. 脱管态 -- 有持久化标识OID,没有被纳入到Session对象的管理**

获得托管态对象:不建议直接获得脱管态的对象.

User user = new User();

user.setId(1);

脱管态对象转换成持久态对象

update();/saveOrUpdate()/lock();

脱管态对象转换成瞬时态对象

user.setId(null);

**4. 注意：持久态对象有自动更新数据库的能力!!!**

**5.注意：hibernate持久态对象不允许更新oid属性。**

## Hibernate的一级缓存

### Session对象的一级缓存（重点）\*\*

**1. 什么是缓存？**

其实就是一块内存空间,将数据源（数据库或者文件）中的数据存放到缓存中.再次获取的时候 ,直接从缓存中获取.可以提升程序的性能！

**2. Hibernate框架提供了两种缓存**

一级缓存 -- 自带的不可卸载的.一级缓存的生命周期与session一致.一级缓存称为session级别的缓存.

二级缓存 -- 默认没有开启，需要手动配置才可以使用的.二级缓存可以在多个session中共享数据,二级缓存称为是sessionFactory级别的缓存.

**3. Session对象的缓存概述**

Session接口中,有一系列的java的集合,这些java集合构成了Session级别的缓存(一级缓存).将对象存入到一级缓存中,session没有结束生命周期,那么对象在session中存放着

内存中包含Session实例 --> Session的缓存（一些集合） --> 集合中包含的是缓存对象！

**4. 证明一级缓存的存在，编写查询的代码即可证明**

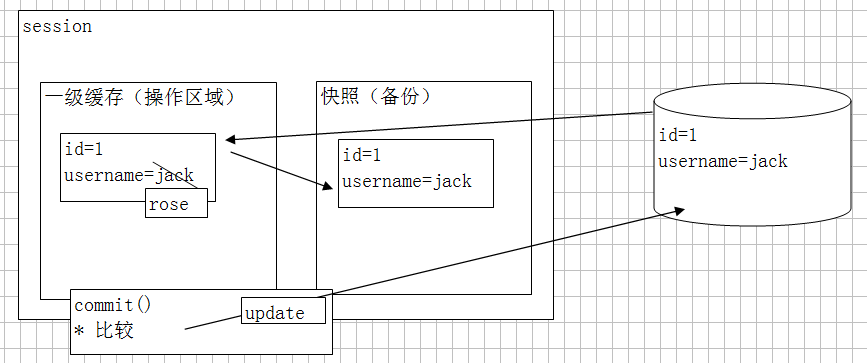
在同一个Session对象中两次查询，可以证明使用了缓存

**5. Hibernate框架是如何做到数据发生变化时进行同步操作的呢？**

使用get方法查询User对象

然后设置User对象的一个属性，注意：没有做update操作。发现，数据库中的记录也改变了。

**利用快照机制来完成的（SnapShot）**



**控制Session的一级缓存（了解）**

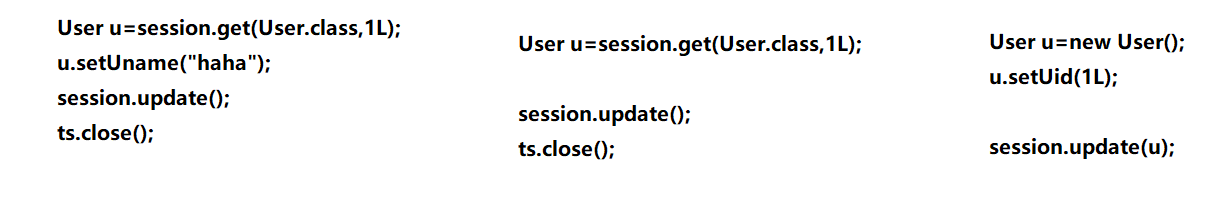
1. 学习Session接口中与一级缓存相关的方法

Session.clear() -- 清空缓存。

Session.evict(Object entity) -- 从一级缓存中清除指定的实体对象。

Session.flush() -- 刷出缓存

案例：演示看效果



## Hibernate框架的查询方式

### Query查询接口

1. 具体的查询代码如下

|  |
| --- |
| // 1.查询所有记录  /\*Query query = session.createQuery("from Customer");  List<Customer> list = query.list();  System.out.println(list);\*/    // 2.条件查询:  /\*Query query = session.createQuery("from Customer where name = ?");  query.setString(0, "李健");  List<Customer> list = query.list();  System.out.println(list);\*/    // 3.条件查询:  /\*Query query = session.createQuery("from Customer where name = :aaa and age = :bbb");  query.setString("aaa", "李健");  query.setInteger("bbb", 38);  List<Customer> list = query.list();  System.out.println(list);\*/ |

### Criteria查询接口（做条件查询非常合适）

\* 条件查询使用Criteria接口的add方法，用来传入条件。

\* 使用Restrictions的添加条件的方法，来添加条件，例如：

\* Restrictions.eq -- 相等

\* Restrictions.gt -- 大于号

\* Restrictions.ge -- 大于等于

\* Restrictions.lt -- 小于

\* Restrictions.le -- 小于等于

\* Restrictions.between -- 在之间

\* Restrictions.like -- 模糊查询

\* Restrictions.in -- 范围

\* Restrictions.and -- 并且

\* Restrictions.or -- 或者

\* 测试代码如下

Session session = HibernateUtils.getCurrentSession();

Transaction tr = session.beginTransaction();

Criteria criteria = session.createCriteria(Linkman.class);

// 设置排序

criteria.addOrder(Order.desc("lkm\_id"));

// 设置查询条件

criteria.add(Restrictions.or(Restrictions.eq("lkm\_gender", "男"), Restrictions.gt("lkm\_id", 3L)));

List<Linkman> list = criteria.list();

for (Linkman linkman : list) {

System.out.println(linkman);

}

tr.commit();

1. 具体的查询代码如下

|  |
| --- |
| // 1.查询所有记录  /\*Criteria criteria = session.createCriteria(Customer.class);  List<Customer> list = criteria.list();  System.out.println(list);\*/    // 2.条件查询  /\*Criteria criteria = session.createCriteria(Customer.class);  criteria.add(Restrictions.eq("name", "李健"));  List<Customer> list = criteria.list();  System.out.println(list);\*/    // 3.条件查询  /\*Criteria criteria = session.createCriteria(Customer.class);  criteria.add(Restrictions.eq("name", "李健"));  criteria.add(Restrictions.eq("age", 38));  List<Customer> list = criteria.list();  System.out.println(list);\*/ |

## 绑定本地的Session

1. 之前在讲JavaWEB的事务的时候，需要在业务层使用Connection来开启事务，

一种是通过参数的方式传递下去

另一种是把Connection绑定到ThreadLocal对象中

2. 现在的Hibernate框架中，使用session对象开启事务，所以需要来传递session对象，框架提供了ThreadLocal的方式

需要在hibernate.cfg.xml的配置文件中提供配置

<property name="hibernate.current\_session\_context\_class">thread</property>

重新HibernateUtil的工具类，使用SessionFactory的getCurrentSession()方法，获取当前的Session对象。并且该Session对象不用手动关闭，线程结束了，会自动关闭。

public static Session getCurrentSession(){

return factory.getCurrentSession();

}

**注意：想使用getCurrentSession()方法，必须要先配置才能使用。**

## Hibernate中的事务与并发

### 事务相关的概念

#### 1. 什么是事务

事务就是逻辑上的一组操作，组成事务的各个执行单元，操作要么全都成功，要么全都失败.

转账的例子：赵信给狐狸转钱，扣钱，加钱。两个操作组成了一个事情！

#### 2. 事务的特性

原子性 -- 事务不可分割.

一致性 -- 事务执行的前后数据的完整性保持一致.

隔离性 -- 一个事务执行的过程中,不应该受到其他的事务的干扰.

持久性 -- 事务一旦提交,数据就永久保持到数据库中.

#### 3. 如果不考虑隔离性:引发一些读的问题

脏读 -- 一个事务读到了另一个事务未提交的数据.

不可重复读 -- 一个事务读到了另一个事务已经提交的update数据,导致多次查询结果不一致.

虚读 -- 一个事务读到了另一个事务已经提交的insert数据,导致多次查询结构不一致.

#### 4. 通过设置数据库的隔离级别来解决上述读的问题

未提交读:以上的读的问题都有可能发生.

已提交读:避免脏读,但是不可重复读，虚读都有可能发生.

可重复读:避免脏读，不可重复读.但是虚读是有可能发生.

串行化:以上读的情况都可以避免.

#### 5. 如果想在Hibernate的框架中来设置隔离级别，需要在hibernate.cfg.xml的配置文件中通过标签来配置

通过：hibernate.connection.isolation = 4 来配置

取值

1—Read uncommitted isolation

2—Read committed isolation

4—Repeatable read isolation

8—Serializable isolation

### 丢失更新的问题

1. 如果不考虑隔离性，也会产生写入数据的问题，这一类的问题叫丢失更新的问题。

2. 例如：两个事务同时对某一条记录做修改，就会引发丢失更新的问题。

A事务和B事务同时获取到一条数据，同时再做修改

如果A事务修改完成后，提交了事务

B事务修改完成后，不管是提交还是回滚，如果不做处理，都会对数据产生影响

3. 解决方案有两种

**悲观锁**

采用的是数据库提供的一种锁机制，如果采用做了这种机制，在SQL语句的后面添加 for update 子句

当A事务在操作该条记录时，会把该条记录锁起来，其他事务是不能操作这条记录的。

只有当A事务提交后，锁释放了，其他事务才能操作该条记录

**乐观锁**

采用版本号的机制来解决的。会给表结构添加一个字段version=0，默认值是0

当A事务在操作完该条记录，提交事务时，会先检查版本号，如果发生版本号的值相同时，才可以提交事务。同时会更新版本号version=1.

当B事务操作完该条记录时，提交事务时，会先检查版本号，如果发现版本不同时，程序会出现错误。

4. 使用Hibernate框架解决丢失更新的问题

悲观锁

使用session.get(Customer.class, 1,LockMode.UPGRADE); 方法

乐观锁

1.在对应的JavaBean中添加一个属性，名称可以是任意的。例如：private Integer version; 提供get和set方法

2.在映射的配置文件中，提供<version name="version"/>标签即可。

## 加载策略(优化查询)

### 策略的种类

* 立即检索：立即查询，在执行查询语句时，立即查询所有的数据。
* 延迟检索：延迟查询，在执行查询语句之后，在需要时在查询。（懒加载）

### 策略的应用:

类级别的加载策略.

关联级别的加载策略.

#### 类级别加载策略:

**1. get/load**

get: 立即查询数据库,将数据初始化

load: hbm文件中,class元素的lazy属性绝定类级别load方法的加载策略

true:先返回一个代理对象.使用代理对象的属性时,才去查询数据库.

false: 与get一样,会立即加载数据

|  |
| --- |
| @Test  **public** **void** run1() {  //类级别  Session session = HibernateUtil.getCurrentSession();    Transaction ts = session.beginTransaction();  //1立即  // Customer customer = (Customer) session.get(Customer.class, 1);  //2延迟  Customer customer = (Customer) session.load(Customer.**class**, 1);        //打印  System.*out*.println(customer.getCid());  System.*out*.println(customer.getCname());  ts.commit(); |

#### 关联级别检索

##### 一对多或多对多

#### 介绍

* 容器<set> 提供两个属性：fetch、lazy

lazy：关联对象是否延迟。

fetch：确定使用sql格式

* fetch：join、select、subselect

join：底层使用迫切左外连接

select：使用多个select语句（默认值）

subselect：使用子查询

* lazy：false、true、extra

false：立即

true：延迟（默认值）

extra：极其懒惰

**集合(一对多):**

set

lazy: 是否对set数据使用懒加载

true:(默认值) 对集合使用才加载

false: 集合将会被立即加载

extra: 极其懒惰,如果使用集合时,之调用size方法查询数量, Hibernate会发送count语句,只查询数量.不加载集合内数据.

fetch : 决定加载集合使用的sql语句种类

select: (默认值) 普通select查询

join: 表链接语句查询集合数据

subselect: 使用子查询 一次加载多个Customer的订单数据

select true 默认值, 会在使用集合时加载,普通select语句

select false 立刻使用select语句加载集合数据

select extra 会在使用集合时加载,普通select语句,如果只是获得集合的长度,会发送Count语句查询长度.

join true 查询集合时使用表链接查询,会立刻加载集合数据

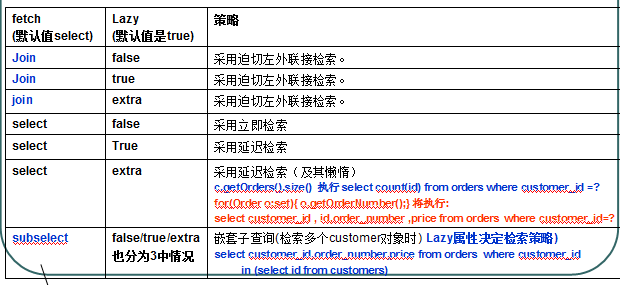
join false 查询集合时使用表链接查询,会立刻加载集合数据

join extra 查询集合时使用表链接查询,会立刻加载集合数据

subselect true 会在使用集合时加载,子查询语句

subselect false 会在查询用户时,立即使用子查询加载客户的订单数据

subselect extra 会在使用集合时加载,子查询语句,如果只是获得集合的长度,会发送Count语句查询长度.



### 多对一

#### 介绍

* <many-to-one fetch="" lazy=""> (<one-to-one>)
* fetch取值：join、select

join：底层使用迫切左外连接

select：多条select语句

* lazy取值：false、proxy、no-proxy

false：立即

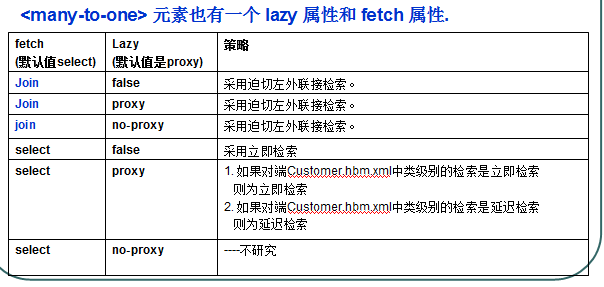
proxy：采用关联对象 类级别检索的策略。lazy="proxy"是否延迟，取决关联对象 类级别检索策略。

订单 关联 客户 （多对一）

订单 立即获得 客户，需要在客户Customer.hbm.xml <class lazy="false">

订单 延迟获得 客户，需要在客户Customer.hbm.xml <class lazy="true">

no-proxy 不研究



## 检索总结

|  |  |  |  |
| --- | --- | --- | --- |
| 检索策略 | 优点 | 缺点 | 优先考虑使用的场合 |
| 立即检索 | 对应用程序完全透明，不管对象处于持久化状态还是游离状态，应用程序都可以从一个对象导航到关联的对象 | (1)select语句多  (2)可能会加载应用程序不需要访问的对象，浪费许多内存空间。 | (1)类级别  (2)应用程序需要立即访问的对象  (3)使用了二级缓存 |
| 延迟检索 | 由应用程序决定需要加载哪些对象，可以避免执行多余的select语句，以及避免加载应用程序不需要访问的对象。因此能提高检索性能，并节省内存空间。 | 应用程序如果希望访问游离状态的代理类实例，必须保证她在持久化状态时已经被初始化。 | (1)一对多或者多对多关联  (2)应用程序不需要立即访问或者根本不会访问的对象 |
| 表连接检索 | (1)对应用程序完全透明，不管对象处于持久化状态还是游离状态，都可从一个对象导航到另一个对象。  (2)使用了外连接，select语句少 | (1)可能会加载应用程序不需要访问的对象，浪费内存。  (2)复杂的数据库表连接也会影响检索性能。 | (1)多对一或一对一关联  (2)需要立即访问的对象  (3)数据库有良好的表连接性能。 |

Web 三层开发 页面==》action🡺service.=>dao层

Lazy=”true”

Dao: User u=session.load(User.class,1L);//user orders <set lazy=”extra”>

Service:dao.findById(1L);

Action: request.setAttribute(“user”,user); 转发到页面 session已经关闭了

页面jsp:${user.uname} ${user.orders}

1. 勤快点查找
2. 业务层，业务层知道页面要展示哪些数据，获得以下你要战士的数据。

User

## 技术分析之Hibernate框架的查询方式

### 1. 唯一标识OID的检索方式

session.get(对象.class,OID)

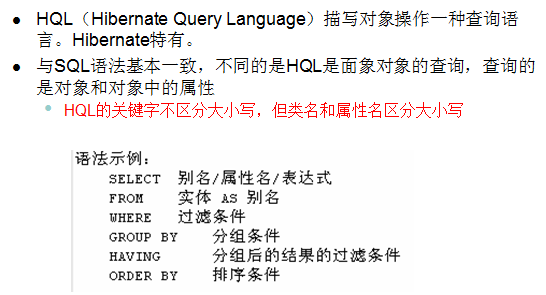
session.load(User.class,oid);

### 2. 对象的导航的方式

User u=session.get(User.class,1L);

u.getOrders();

### 3. HQL的检索方式



Query query = session.createQuery("hql语句") --> 对象，对象属性

Hibernate Query Language -- Hibernate的查询语言

### 4. QBC的检索方式

Query By Criteria -- 条件查询

### 5. SQL检索方式（了解）

SQLQuery sqlQuery = session.createSQLQuery("sql 语句") --->表，表字段（列）

sqlQuery.list() 查询所有

sqlQuery.uniqueResult() 查询一个

本地的SQL检索

技术分析之HQL的查询方式概述

1. HQL的介绍

HQL(Hibernate Query Language) 是面向对象的查询语言, 它和 SQL 查询语言有些相似

在 Hibernate 提供的各种检索方式中, HQL 是使用最广的一种检索方式

2. HQL与SQL的关系

HQL 查询语句是面向对象的,Hibernate负责解析HQL查询语句, 然后根据对象-关系映射文件中的映射信息, 把 HQL 查询语句翻译成相应的 SQL 语句.

HQL 查询语句中的主体是域模型中的类及类的属性

SQL 查询语句是与关系数据库绑定在一起的. SQL查询语句中的主体是数据库表及表的字段

## 技术分析之HQL的查询演示

### 1. HQL基本的查询格式

支持方法链的编程，即直接调用list()方法

简单的代码如下

session.createQuery("from Customer").list();

### 2. 使用别名的方式

可以使用别名的方式

session.createQuery("from Customer c").list();

session.createQuery("select c from Customer c").list();

### 3. 排序查询

排序查询和SQL语句中的排序的语法是一样的

升序

session.createQuery("from Customer order by cust\_id").list();

降序

session.createQuery("from Customer order by cust\_id desc").list();

### 4. 分页查询

Hibernate框架提供了分页的方法，咱们可以调用方法来完成分页

两个方法如下

setFirstResult(a) -- 从哪条记录开始，如果查询是从第一条开启，值是0

setMaxResults(b) -- 每页查询的记录条数

演示代码如下

List<LinkMan> list = session.createQuery("from LinkMan").setFirstResult(0).setMaxResults().list();

### 5. 带条件的查询

setParameter("?号的位置，默认从0开始","参数的值"); 不用考虑参数的具体类型

按位置绑定参数的条件查询（指定下标值，默认从0开始）

按名称绑定参数的条件查询（HQL语句中的 ? 号换成 :名称 的方式）

例如代码如下

|  |
| --- |
| Query query = session.createQuery("from Linkman where lkm\_name like ? order by lkm\_id desc");  query.setFirstResult(0).setMaxResults(3);  query.setParameter(0, "%熊%");  List<Linkman> list = query.list();  for (Linkman linkman : list) {  System.out.println(linkman);  } |

### 6. HQL的投影查询

1. 投影查询就是想查询某一字段的值或者某几个字段的值

2. 投影查询的案例

如果查询多个字段，例如下面这种方式

|  |
| --- |
| List<Object[]> list = session.createQuery("select c.cust\_name,c.cust\_level from Customer c").list();  for (Object[] objects : list) {  System.out.println(Arrays.toString(objects));  } |

如果查询两个字段，也可以把这两个字段封装到对象中

先在持久化类中提供对应字段的构造方法

使用下面这种HQL语句的方式

|  |
| --- |
| List<Customer> list = session.createQuery("select new Customer(c.cust\_name,c.cust\_level) from Customer c").list();  for (Customer customer : list) {  System.out.println(customer);  } |

### 7.技术分析之聚合函数查询

**1. 获取总的记录数**

|  |
| --- |
| Session session = HibernateUtils.getCurrentSession();  Transaction tr = session.beginTransaction();  List<Number> list = session.createQuery("select count(c) from Customer c").list();  Long count = list.get(0).longValue();  System.out.println(count);  tr.commit(); |

**2. 获取某一列数据的和**

Session session = HibernateUtils.getCurrentSession();

Transaction tr = session.beginTransaction();

List<Number> list = session.createQuery("select sum(c.cust\_id) from Customer c").list();

Long count = list.get(0).longValue();

System.out.println(count);

tr.commit();

### 8 多表的查询进来使用HQL语句进行查询，HQL语句和SQL语句的查询语法比较类似。

笛卡尔积：在开发中，尽量避免出现笛卡尔积 ，这种结果集没有任何意义。

\* 内连接查询 按照条件查找数句

\* 显示内连接

\* select \* from customers c inner join orders o on c.cid = o.cno;

\* 隐式内连接

\* select \* from customers c,orders o where c.cid = o.cno;

\* 外连接查询

\*

\* select \* from customers c left join orders o on c.cid = o.cno;

\* 右外连接

\* select \* from customers c right join orders o on c.cid = o.cno;

#### HQL的多表查询



1.交叉连接 ，等效 sql 笛卡尔积

2.隐式内连接，等效 sql 隐式内连接

3.内连接，等效sql内连接

4.迫切内连接，hibernate底层使用 内连接。

5.左外连接，等效sql左外连接

6.迫切左外连接，hibernate底层使用 左外连接

7.右外连接，等效sql右外连接

**迫切和非迫切：**

\* 非迫切返回结果是Object[]

\* 迫切连接返回的结果是对象，把客户的信息封装到客户的对象中，把订单的信息封装到客户的Set集合中。

#### 内连接查询

\* 内连接使用 inner join ，默认返回的是Object数组

Session session = HibernateUtils.getCurrentSession();

Transaction tr = session.beginTransaction();

List<Object[]> list = session.createQuery("from Customer c inner join c.linkmans").list();

for (Object[] objects : list) {

System.out.println(Arrays.toString(objects));

}

tr.commit();

\* 迫切内连接:inner join fetch ，返回的是实体对象

Session session = HibernateUtils.getCurrentSession();

Transaction tr = session.beginTransaction();

List<Customer> list = session.createQuery("from Customer c inner join fetch c.linkmans").list();

Set<Customer> set = new HashSet<Customer>(list);

for (Customer customer : set) {

System.out.println(customer);

}

tr.commit();

#### 左外连接查询

\* 左外连接: 封装成List<Object[]>

\* 迫切左外连接

Session session = HibernateUtils.getCurrentSession();

Transaction tr = session.beginTransaction();

List<Customer> list = session.createQuery("from Customer c left join fetch c.linkmans").list();

Set<Customer> set = new HashSet<Customer>(list);

for (Customer customer : set) {

System.out.println(customer);

}

tr.commit();

## 技术分析之QBC检索方式

QBC：Query By Criteria 按条件进行查询

### 1. 简单查询，使用的是Criteria接口

List<Customer> list = session.createCriteria(Customer.class).list();

for (Customer customer : list) {

System.out.println(customer);

}

### 2. 排序查询

\* 需要使用addOrder()的方法来设置参数，参数使用org.hibernate.criterion.Order对象

\* 具体代码如下：

Session session = HibernateUtils.getCurrentSession();

Transaction tr = session.beginTransaction();

Criteria criteria = session.createCriteria(Linkman.class);

// 设置排序

criteria.addOrder(Order.desc("lkm\_id"));

List<Linkman> list = criteria.list();

for (Linkman linkman : list) {

System.out.println(linkman);

}

tr.commit();

### 3. 分页查询

\* QBC的分页查询也是使用两个方法

\* setFirstResult();

\* setMaxResults();

\* 代码如下;

Session session = HibernateUtils.getCurrentSession();

Transaction tr = session.beginTransaction();

Criteria criteria = session.createCriteria(Linkman.class);

// 设置排序

criteria.addOrder(Order.desc("lkm\_id"));

criteria.setFirstResult(0);

criteria.setMaxResults(3);

List<Linkman> list = criteria.list();

for (Linkman linkman : list) {

System.out.println(linkman);

}

tr.commit();

### 4. 条件查询（Criterion是查询条件的接口，Restrictions类是Hibernate框架提供的工具类，使用该工具类来设置查询条件）

\* 条件查询使用Criteria接口的add方法，用来传入条件。

\* 使用Restrictions的添加条件的方法，来添加条件，例如：

\* Restrictions.eq -- 相等

\* Restrictions.gt -- 大于号

\* Restrictions.ge -- 大于等于

\* Restrictions.lt -- 小于

\* Restrictions.le -- 小于等于

\* Restrictions.between -- 在之间

\* Restrictions.like -- 模糊查询

\* Restrictions.in -- 范围

\* Restrictions.and -- 并且

\* Restrictions.or -- 或者

\* 测试代码如下

Session session = HibernateUtils.getCurrentSession();

Transaction tr = session.beginTransaction();

Criteria criteria = session.createCriteria(Linkman.class);

// 设置排序

criteria.addOrder(Order.desc("lkm\_id"));

// 设置查询条件

criteria.add(Restrictions.or(Restrictions.eq("lkm\_gender", "男"), Restrictions.gt("lkm\_id", 3L)));

List<Linkman> list = criteria.list();

for (Linkman linkman : list) {

System.out.println(linkman);

}

tr.commit();

### 5. 聚合函数查询（Projection的聚合函数的接口，而Projections是Hibernate提供的工具类，使用该工具类设置聚合函数查询）

\* 使用QBC的聚合函数查询，需要使用criteria.setProjection()方法

\* 具体的代码如下

Session session = HibernateUtils.getCurrentSession();

Transaction tr = session.beginTransaction();

Criteria criteria = session.createCriteria(Linkman.class);

criteria.setProjection(Projections.rowCount());

List<Number> list = criteria.list();

Long count = list.get(0).longValue();

System.out.println(count);

tr.commit();

## 技术分析之SQL查询方式（了解）

1. 基本语法

Session session = HibernateUtils.getCurrentSession();

Transaction tr = session.beginTransaction();

SQLQuery sqlQuery = session.createSQLQuery("select \* from cst\_linkman where lkm\_gender = ?");

sqlQuery.setParameter(0,"男");

sqlQuery.addEntity(Linkman.class);

List<Linkman> list = sqlQuery.list();

System.out.println(list);

tr.commit();

## 命名查询

# 二级缓存

## 什么是二级缓存

* hibernate 提供缓存机制：一级缓存、二级缓存

一级缓存：session级别缓存，在一次请求中共享数据。

二级缓存：sessionFactory级别缓存，整个应用程序共享一个会话工厂，共享一个二级缓存。

## 应用场景

* 适合放入二级缓存中的数据:

很少被修改

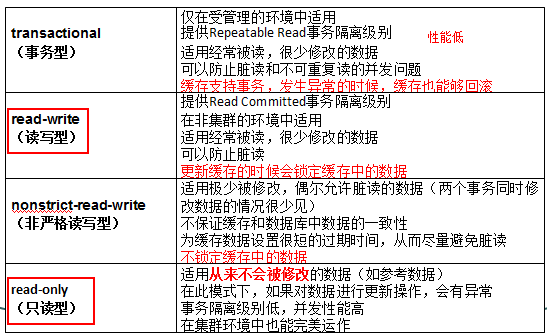
不是很重要的数据, 允许出现偶尔的并发问题

* 不适合放入二级缓存中的数据:

经常被修改

财务数据, 绝对不允许出现并发问题

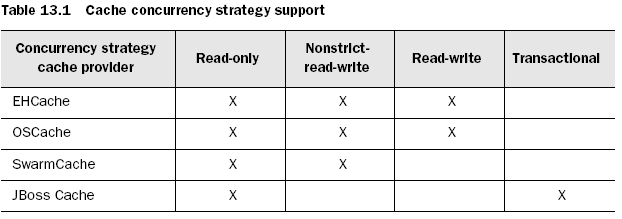
## 并发访问策略



* 访问策略：读写型（read-write）、只读型（read-only）

## 二级缓存提供商

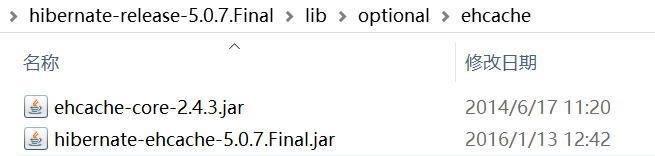
* **EHCache**: 可作为进程（单机）范围内的缓存, 存放数据的物理介质可以是内存或硬盘, 对 Hibernate 的查询缓存提供了支持。--支持集群。
* OpenSymphony `:可作为进程范围内的缓存, 存放数据的物理介质可以是内存或硬盘, 提供了丰富的缓存数据过期策略, 对 Hibernate 的查询缓存提供了支持
* SwarmCache: 可作为集群范围内的缓存, 但不支持 Hibernate 的查询缓存
* JBossCache:可作为集群范围内的缓存, 支持 Hibernate 的查询缓存



X表示支持

## 配置二级缓存（操作）

### 1. 导入jar包：



### 2.开启二级缓存(我要使用二级缓存)

|  |
| --- |
| <!--开启二级缓存-->  <property name=*"hibernate.cache.use\_second\_level\_cache"*>true</property> |

### 3.确定二级缓存提供商(我要使用哪个二级缓存)

|  |
| --- |
| <!--配置使用的二级缓存 -->  <property name=*"cache.region.factory\_class"*>  org.hibernate.cache.ehcache.EhCacheRegionFactory</property> |

### 4.确定需要缓存内容

#### 1>配置需要缓存的类

|  |
| --- |
| <!-- 配置二级缓存的类 -->  <class-cache usage=*"read-only"* class=*"pojo.User"*/>  <class-cache usage=*"read-only"* class=*"pojo.Order"*/> |

#### 2>配置需要缓存的集合

|  |
| --- |
| <!-- 配置二级缓存的集合 缓存集合要把该类先缓存了 不然不生效-->  <collection-cache usage=*"read-only"* collection=*"pojo.User.orders"* /> |

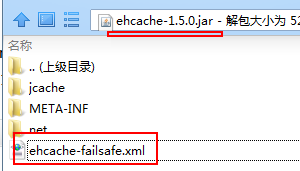
### 5.测试

|  |
| --- |
|  |



### 6.配置ehcache自定义配置文件

步骤1：从jar包复制xml文件



步骤2：将xml重命名“ehcache.xml”



步骤3：将修改后的xml，拷贝到src下

|  |
| --- |
| <ehcache xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xsi:noNamespaceSchemaLocation=*"../config/ehcache.xsd"*>  <diskStore path=*"D:/cache"*/>  <!-- 配置属性          timeToIdleSeconds 当缓存闲置n秒后销毁          timeToLiveSeconds 当缓存存活n秒后销毁          maxElementsInMemory：缓存最大个数。          eternal:对象是否永久有效，一但设置了，timeout将不起作用。          timeToIdleSeconds：设置对象在失效前的允许闲置时间（单位：秒）。仅当eternal=false对象不是永久有效时使用，可选属性，默认值是0，也就是可闲置时间无穷大。          timeToLiveSeconds：设置对象在失效前允许存活时间（单位：秒）。最大时间介于创建时间和失效时间之间。仅当eternal=false对象不是永久有效时使用，默认是0.，也就是对象存活时间无穷大。          overflowToDisk：当内存中对象数量达到maxElementsInMemory时，Ehcache将会对象写到磁盘中。          diskSpoolBufferSizeMB：这个参数设置DiskStore（磁盘缓存）的缓存区大小。默认是30MB。每个Cache都应该有自己的一个缓冲区。          maxElementsOnDisk：硬盘最大缓存个数。 diskPersistent：是否缓存虚拟机重启期数据 Whether the          disk store persists between restarts of the Virtual Machine. The          default value is false.          diskExpiryThreadIntervalSeconds：磁盘失效线程运行时间间隔，默认是120秒。          memoryStoreEvictionPolicy：当达到maxElementsInMemory限制时，Ehcache将会根据指定的策略去清理内存。默认策略是LRU（最近最少使用）。你可以设置为FIFO（先进先出）或是LFU（较少使用）。          clearOnFlush：内存数量最大时是否清除。          -->  <defaultCache  maxElementsInMemory=*"10000"*  eternal=*"false"*  timeToIdleSeconds=*"120"*  timeToLiveSeconds=*"120"*  overflowToDisk=*"true"*  maxElementsOnDisk=*"10000000"*  diskPersistent=*"false"*  diskExpiryThreadIntervalSeconds=*"120"*  memoryStoreEvictionPolicy=*"LRU"*    />  </ehcache> |

### 7.查询缓存

* 查询缓存默认不使用。需要手动开启
* 查询缓存：将HQL语句与 查询结果进行绑定。通过HQL相同语句可以缓存内容。

默认情况Query对象只将查询结果存放在一级和二级缓存，不从一级或二级缓存获取。

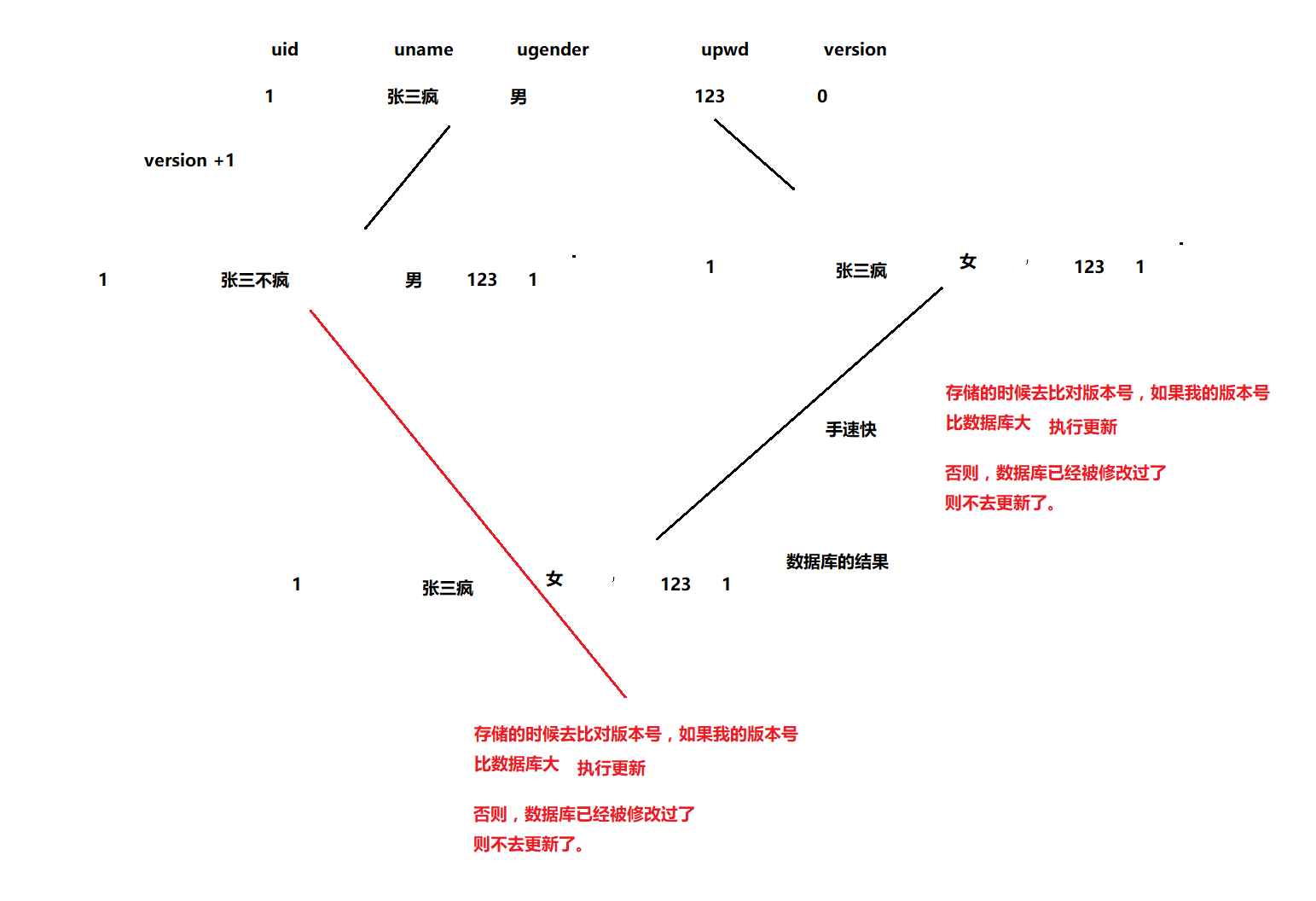
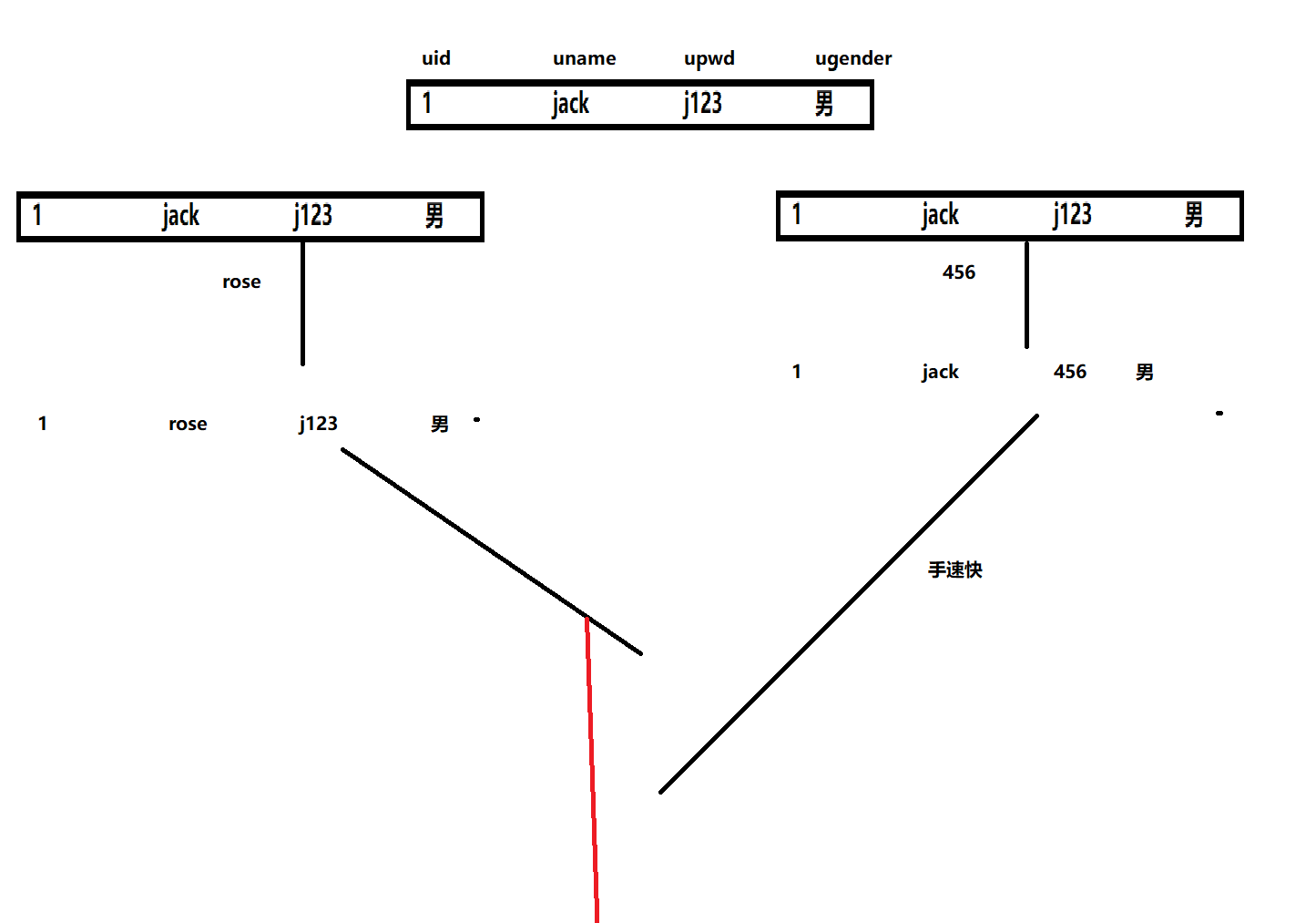
查询缓存就是让Query可以从二级缓存获得内容。

步骤一：开启查询缓存

|  |
| --- |
| <!-- 开启查询缓存 -->  <property name=*"hibernate.cache.use\_query\_cache"*>true</property> |

步骤二：在查询query对象，设置缓存内容（注意：存放和查询 都需要设置）





## Hibernate中的事务与并发

### 事务相关的概念

#### 1. 什么是事务

事务就是逻辑上的一组操作，组成事务的各个执行单元，操作要么全都成功，要么全都失败.

**转账**的例子：赵信给狐狸转钱，扣钱，加钱。两个操作组成了一个事情！

#### 2. 事务的特性 ACID

原子性（Atomicity） -- 事务不可分割. 整体

在事务中的多条sql语句是一个整体，要么就全部执行成功，有一条执行失败，都不执行。

一致性 （Consistency）-- 事务执行的前后数据的完整性保持一致. 数据的完整等性。赵信给狐狸转账2000块。能量守恒

隔离性 （Isolation）-- 一个事务执行的过程中,不应该受到其他的事务的干扰. 关于事务的隔离性数据库提供了多种隔离级别

持久性 （Durability）-- 事务一旦提交,数据就永久保持到数据库中.

#### 3. 如果不考虑隔离性:引发一些读的问题

脏读 -- 一个事务读到了另一个事务未提交的数据.

**张三和李四做交易，张三给李四转账，但是没有提交事务，李四这个时候去查询数据，这个时候读到了张三还有提交的数据，张三回滚事务，导致李四在查询钱少了。**

不可重复读 -- 一个事务读到了另一个事务已经提交的update数据,导致多次查询结果不一致.

张三在取钱的时候，第一查询余额是1万，这个事务李四取走了1万块，导致张三在取钱的时候，取不出来。张三在一个事务中，两次查询的结果不一样，因为中间被另外一个事务做了update操作。

幻读/虚读 -- 一个事务读到了另一个事务已经提交的insert数据,导致多次查询结构不一致.

#### 4. 通过设置数据库的隔离级别来解决上述读的问题

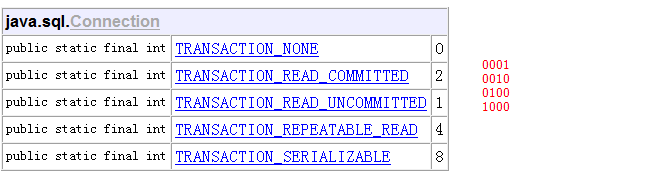
数据库为我们提供的四种隔离级别：

　　 Serializable (串行化)：可避免脏读、不可重复读、幻读的发生。 8

　　 Repeatable read (可重复读)：可避免脏读、不可重复读的发生。 4

　　 Read committed (读已提交)：可避免脏读的发生。 2

　　 Read uncommitted (读未提交)：最低级别，任何情况都无法保证。 1



#### 5. 如果想在Hibernate的框架中来设置隔离级别，需要在hibernate.cfg.xml的配置文件中通过标签来配置

通过：hibernate.connection.isolation = 4 来配置

取值

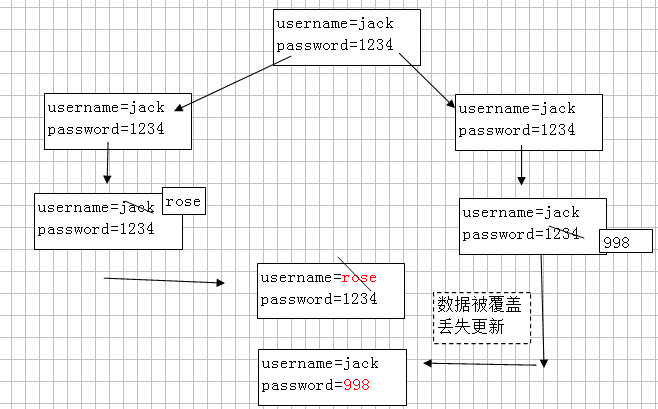
1—Read uncommitted isolation

2—Read committed isolation

4—Repeatable read isolation

8—Serializable isolation

### lost update 丢失更新



* 悲观锁：

顾名思义，就是很悲观，每次去拿数据的时候都认为别人会修改，所以每次在拿数据的时候都会上锁，这样别人想拿这个数据就会block直到它拿到锁。传统的关系型数据库里边就用到了很多这种锁机制，比如 读锁，写锁等，都是在做操作之前先上锁。

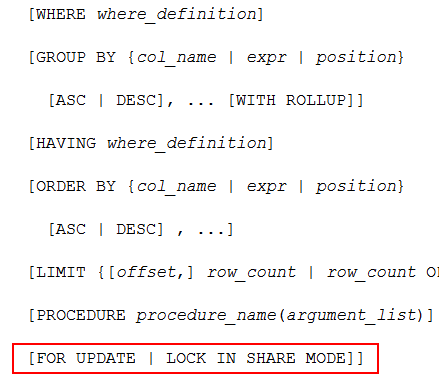
采用数据库锁机制。

读锁：共享锁。该锁可以共享

select .... from ... lock in share mode;

写锁：排他锁。（独占） 只能有一个线程拥有该锁。

select ... from .... for update

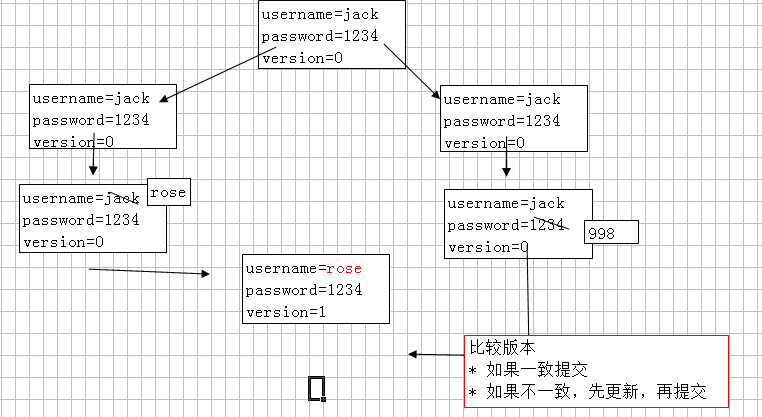


* 乐观锁：

顾名思义，就是很乐观，每次去拿数据的时候都认为别人不会修改，所以不会上锁，但是在更新的时候会判断一下在此期间别人有没有去更新这个数据，可以使用版本号等机制。

在表中提供一个字段（版本字段），用于标识记录。如果版本不一致，不允许操作。

在数据库变里面整个version字段，0，



### 绑定本地的Session

1. 之前在讲JavaWEB的事务的时候，需要在业务层使用Connection来开启事务，

一种是通过参数的方式传递下去

另一种是把Connection绑定到ThreadLocal对象中

2. 现在的Hibernate框架中，使用session对象开启事务，所以需要来传递session对象，框架提供了ThreadLocal的方式

需要在hibernate.cfg.xml的配置文件中提供配置

<property name="hibernate.current\_session\_context\_class">thread</property>

重新HibernateUtil的工具类，使用SessionFactory的getCurrentSession()方法，获取当前的Session对象。并且该Session对象不用手动关闭，线程结束了，会自动关闭。

public static Session getCurrentSession(){

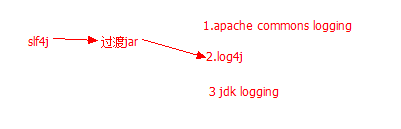
return factory.getCurrentSession();

}

**注意：想使用getCurrentSession()方法，必须要先配置才能使用。**

# 整合log4j(了解)

* slf4j 核心jar ： slf4j-api-1.6.1.jar 。slf4j是日志框架，将其他优秀的日志第三方进行整合。



* 整合导入jar包

log4j 核心包：log4j-1.2.17.jar

过渡jar（整合jar）：slf4j-log4j12-1.7.5.jar

* 导入配置文件

log4j.properties ，此配置文件通知log4j 如何输出日志

* 配置文件内容：

1.记录器

2.输出源

3.布局

* 记录器

例如： log4j.rootLogger=info, stdout,file

格式：log4j.rootLogger=日志级别, 输出源1，输出源2，。。。。

log4j 日志级别 ： fatal 致命错误 error 错误 warn 警告 info 信息 debug 调试信息 trace 堆栈信息 （由高到底顺序）

* 输出源：

例如：log4j.appender.file=org.apache.log4j.FileAppender

格式：log4j.appender.输出源的名称=输出源的实现类

名称：自定义

实现类：log4j提供

输出源属性例如：log4j.appender.file.File=d\\:mylog.log

输出源属性格式：log4j.appender.名称.属性=值

每一个输出源对应一个实现类，实现类都属性（setter），底层执行setter方法进行赋值

* 常见的输出源实现类

org.apache.log4j.FileAppender 输出文件中

file ,表示文件输出位置

org.apache.log4j.ConsoleAppender 输出到控制台

Target ，表示使用哪种输出方式，在控制台打印内容，取值：System.out / System.err

* 布局 -- 确定输出格式

例如：log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

格式：log4j.appender.数据源.layout=org.apache.log4j.PatternLayout

布局属性：log4j.appender. 数据源.layout.ConversionPattern=值

12:56:30,123 info

* 扩展：对指定的目录设置日志级别

例如：log4j.logger.org.hibernate.transaction=debug

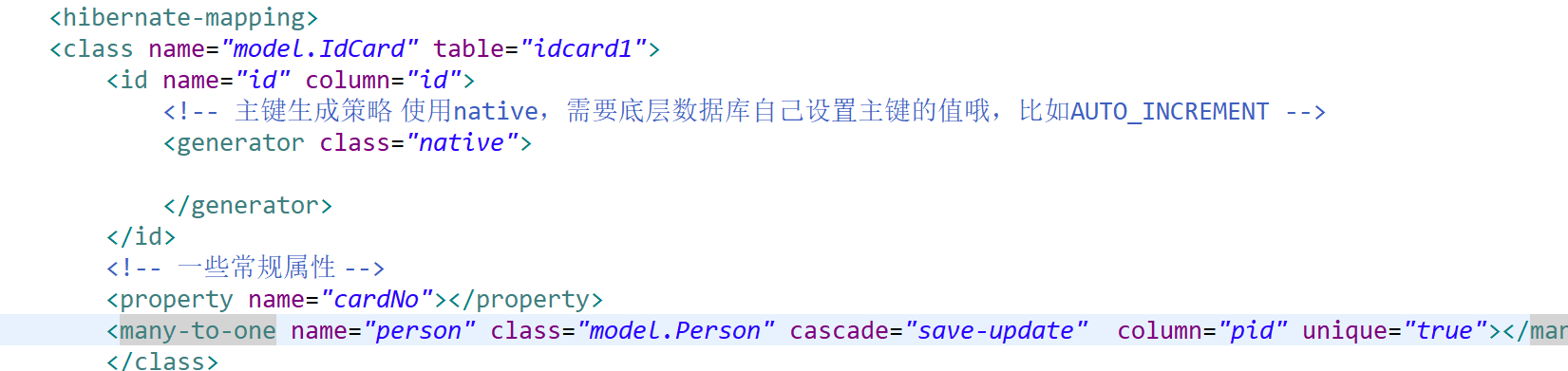
格式：log4j.logger.包结构=级别

# 一对一（了解）

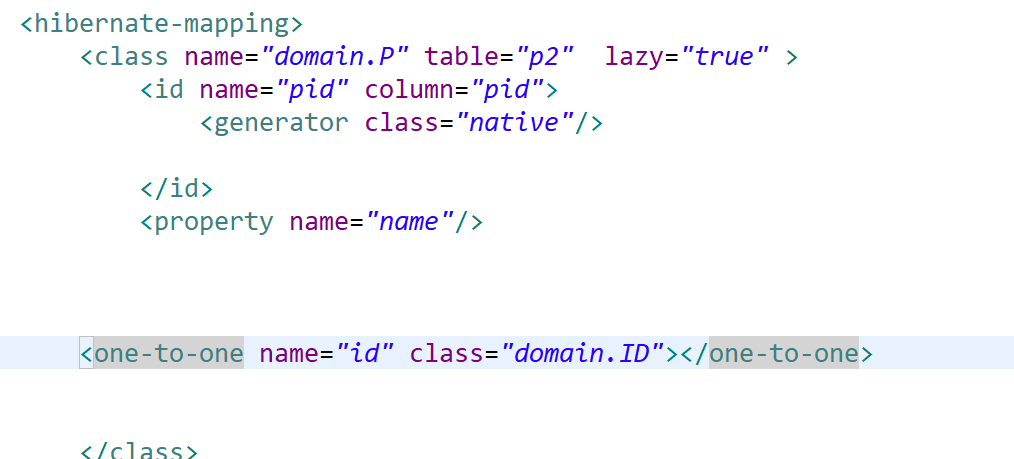
* 情况1：主表的主键，与从表的外键（唯一），形成主外键关系



外键一方



* 情况2：主表的主键，与从表的主键，形成主外键关系 （从表的主键又是外键）





# Hibernate常用的注解

注意：JPA的注解可以写在getter方法上和属性上

## 1、@Entity

标记当前的类是映射的表

## 2、@Table

标记当前类对应的表信息

常用属性：

name:表名

## 3、@Id

标记当前属性为主键

## 4、@GeneratedValue

主键的生成策略

常用属性：

1、strategy：整型的生成策略

取值：

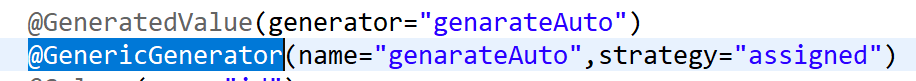
GenerationType.IDENTITY：MySQL的数据库的自增策略

GenerationType.SEQUENCE：Oracle的序列

GenerationType.AUTO:根据数据库自动选择，会创建表来维护数据的增长

GenerationType.TABLE:创建一个表，自己来维护

2、@GenericGenerator 是hibernate对jpa主键生成策略注解的扩展



## 5、@Column

设置当前属性对应的字段信息

常用属性：

name:设置属性对应的字段名称

length:设置字段的长度

nullable：是否允许为null

unique:是否为唯一约束

columnDefinition:手动标记数据库中字段类型，默认也会转换

|  |
| --- |
| columnDefinition:表示该字段在数据库中的实际类型 通常ORM框架可以根据属性类型自动判断数据库中字段的类型, 但是对于Date类型仍无法确定数据库中字段类型究竟是DATE,TIME,TIMESTAMP还是DATETIME, 此外,String的默认映射类型为VARCHAR,如果要将String类型映射到特定数据库的BLOB或TEXT字段类型,该属性非常有用 如: @Column(name="BIRTH",nullable="false",columnDefinition="DATE")  insertable:默认情况下,JPA持续性提供程序假设所有列始终包含在 SQL INSERT 语句中。 如果该列不应包含在这些语句中，请将 insertable 设置为 false  updatable：列始终包含在 SQL UPDATE 语句中。如果该列不应包含在这些语句中，请将 updatable 设置为 false |

注：如果属性没有写@Column该注解，那么默认拥有@Basic注解

@Basic  
@javax.persistence.Basic(fetch=FetchType.LAZY,optional=true)  
fetch:抓取策略,延时加载与立即加载  
optional:指定在生成数据库结构时字段是否允许为 null

## 6、@Transient

标记该属性不是数据库表中字段

## 7、@JoinColumn

多表关系中的外键字段

常用属性：

name:外键名称

多表关系中的主键的通用属性：

1、fetch：抓取策略，可以标记勤加载或懒加载

取值：

FetchType.EAGER:勤加载

FetchType.LAZY:懒加载，如果是集合那么就是懒加载

2、cascade：级联设置，在有外键的一方设置，新增、修改、删除

取值说明：

CascadeType.ALL:全部

CascadeType.DETACH:挂载

CascadeType.MERGE:修改

CascadeType.PERSIST:保存

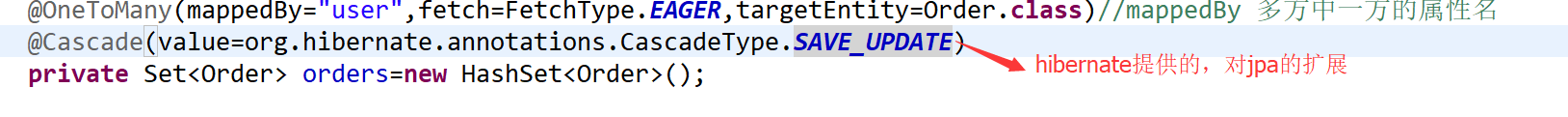
CascadeType.REFRESH:刷新

CascadeType.REMOVE:删除

|  |
| --- |
|  CascadeType.REMOVE Cascade remove operation，级联删除操作。 删除当前实体时，与它有映射关系的实体也会跟着被删除。   CascadeType.MERGE Cascade merge operation，级联更新（合并）操作   CascadeType.DETACH Cascade detach operation，级联脱管/游离操作。 如果你要删除一个实体，但是它有外键无法删除，你就需要这个级联权限了。它会撤销所有相关的外键关联。   CascadeType.REFRESH Cascade refresh operation，级联刷新操作 不用。   CascadeType.ALL Cascade all operations，清晰明确，拥有以上所有级联操作权限。 |

3、targetEntity:目标类的Class类型

注意：



|  |
| --- |
| **public** **enum** CascadeType {  /\*\*  \* Includes all types listed here.  \*/  ***ALL***,  /\*\*  \* Corresponds to {@link javax.persistence.CascadeType#PERSIST}.  \*/  ***PERSIST***,  /\*\*  \* Corresponds to {@link javax.persistence.CascadeType#MERGE}.  \*/  ***MERGE***,  /\*\*  \* Corresponds to {@link javax.persistence.CascadeType#REMOVE}.  \*/  ***REMOVE***,  /\*\*  \* Corresponds to {@link javax.persistence.CascadeType#REFRESH}.  \*/  ***REFRESH***,  /\*\*  \* Corresponds to the Hibernate native DELETE action.  \*/  ***DELETE***,  /\*\*  \* Corresponds to the Hibernate native SAVE\_UPDATE (direct reattachment) action.  \*/  ***SAVE\_UPDATE***,  /\*\*  \* Corresponds to the Hibernate native REPLICATE action.  \*/  ***REPLICATE***,  /\*\*  \* Hibernate originally handled orphan removal as a specialized cascade.  \*  \* **@deprecated** use **@OneToOne**(orphanRemoval=true) or **@OneToMany**(orphanRemoval=true)  \*/  @Deprecated  ~~DELETE\_ORPHAN~~,  /\*\*  \* Corresponds to the Hibernate native LOCK action.  \*/  ***LOCK***,  /\*\*  \* JPA originally planned on calling DETACH EVICT.  \*  \* **@deprecated** use javax.persistence.CascadeType.DETACH  \*/  @Deprecated  ~~EVICT~~,  /\*\*  \* Corresponds to {@link javax.persistence.CascadeType#REFRESH}.  \*/  ***DETACH***  } |

## 8、@OneToOne

体现一对一关系映射

一般要结合@JoinColumn标记外键名称

## 9、@ManyToOne

体现的是多对一

一般要结合@JoinColumn标记外键名称

## 10、@OneToMany

体现一对多的映射，不会出现外键

常用属性：

mappedBy：关联的属性名称，就是存在多的一方中的本类的属性名称

targetEntity:集合中泛型类的Class类对象，其实就是多的类的Class

## 11、@ManyToMany

体现的多对多，隐式实现

需要配合的@JoinTable:关系表

## 12、@JoinTable

多对多的时候的关系表

常用属性：

name:关系表名

joinColumns:外键字段名称，当前类在关系表中的外键名称，得使用注解@JoinColumns

inverseJoinColumns:存在多对多关系的另一方在关系表中的外键名称