Hibernate01

Hibernate

一. Hibernate 简介

1.1 Hibernate 优缺点

• JDBC 的缺点:

- 代码繁琐, try…catch 使用太多.
- JDBC 没有数据的缓存. [如果要求每5分钟从表中获取10000条数据, 怎么做? 5分钟查询一次数据库. 例如股票那些实时系统,必须要这样做.]
- 内存计算 storm 每秒钟能分析几万条数据.[YY 主播有4000观众].
- JDBC 不是面向对象的, 需要写很多 SQL 语言
- SQL 语句的跨平台性很差 [Mysql 和 Oracle 的一些方言是不通的, 包括分页啊, 数据类型啊. 如果使用了 MySQL 写好了换成了 Oracle]

• JDBC 优点:

- 效率高, 最底层直接

• Hibernate优点:

- 完全面向对象编程
- Hibernate 有缓存, 一级二级查询缓存
- 编程时简洁简单一些.跨平台性好.🤡🤡
- 使用场合就是企业内部的系统

- **Hibernate 缺点:**

    - 效率比较低
    - 表数据在千万以上，不适合
    - 表与表之间的关系特别复杂，不适合

1.2 Hibernate 框架概述

> *• Hibernate 是一个开放源代码的对象关系映射（ORM）框架，它对 JDBC 进行了非常轻量级的对象封装，使得 Java 程序员可以随心所欲的使用对象编程思维来操纵数据库。*
> *• Hibernate 可以应用在任何使用 JDBC 的场合，既可以在 Java 的客户端程序使用，也可以在 Servlet/JSP 的 Web 应用中使用。*
> *• 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.*

1.3 什么是ORM（对象关系映射)

ORM映射：Object Relational Mapping
• 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）
它的作用就是在关系型数据库和对象之间做了一个映射。从对象（Object）映射到关系（Relation），再从关系映射到对象。这样，我们在操作数据库的时候，不需要再去和复杂SQL打交道，只要像操作对象一样操作它就可以了（把关系数据库的字段在内存中映射成对象的属性）.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 .

1.4 Hibernate的项目结构

- **documentation (hibernate的文档)**

    - `quickstart:`快速开发指南

- **lib(jar包)**

  - `required:`hibernate使用必须包
  - `optional:`可选择的包
  - `jpa:`hibernate完成JPA规范的包
  - `envers:`hibernate的一个子项目

- **project(hibernate项目相关的内容)**

  - 包括了代码,测试代码
  - 包括了实例代码

二. Hibernate 入门

1. 创建工程 Hibernate011_CRUD
2. 导入 jar 包 ○ lib/required/jars ▪ javassist包是用来创建代理对象 ○ Sql 数据库驱动 jar 包
3. 在src目录下，创建名称为 hibernate.cfg.xml 的配置文件 hibernate-5.1.5/project/etc/hibernate.cfg.xml
4. 创建持久化类和映射文件
5. 映射文件导入到 hibernate 的配置文件中
6. 生成表
7. 客户端– CRUD 操作

三. Hibernate 详解

3.1 配置文件

3.1.1 核心配置文件 hibernate.cfg.xml

- 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/itcast/hibernate/domain/User.hbm.xml" />

3.1.2 映射配置文件 JavaBean.hbm.xml

- <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
- <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 :字段的长度, 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。
  - 主键的生成策略 - generator
    - 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.
    - assigned : 主键的生成不用Hibernate管理了.必须手动设置主键. [一般不常用]
    - 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. [查看表中的 pid 是否勾选了 Auto_inc]
    - uuid : 适用于 char,archer 类型的作为主键.
        - uuid的字符串是由 hibernate 内部生成的.
        - 使用随机的字符串作为主键.
    - sequence (序列) : 适用于short,int,long作为主键.底层使用的是序列的增长方式. Mysql 没有.
        - Oracle数据库底层没有自动增长,想自动增长需要使用序列. [生成类似于 uuid 的一串数字.在 Oracle 内部做的.]
    - 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.
- <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 :数据类型 (一般都不需要编写, 如果写需要按着规则来编写)

3.2 常用的接口和类

3.2.1 Configuration类

- Configuration接口:负责配置并启动Hibernate
- 把Configeration 对象中的所有配置信息拷贝到SessionFactory的缓存中.SessionFactory的实例代表一个数据库存储员源, 创建后不再与Configeration 对象关联。

3.2.2 SessionFactory接口:

- 负责初始化Hibernate, hibernate中的配置文件、映射文件、持久化类的信息都在sessionFactory中.
- sessionFactory 中存放的信息是共享的 [每个操作都要用到]
- sessionFactory 本身就是线程安全的,意味着它的一个实例可以被应用的多个线程共享。[因为只加载一次即可, 在Tomcat 启动时加载, 之后不允许被更改]
- 一个 Hibernate 框架 sessionFactory 只有一个
- SessionFactory是重量级别的类, 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。
- 构造SessionFactory 很消耗资源, 一般情况下一个应用只初始化一个SessionFactory

,最好是应用启动时就完成初始化。

3.2.3 Session 接口

- Session接口负责执行被持久化对象的CRUD操作.也被称之为持久化管理器。
- Session 是应用程序与数据库之间交互操作的一个单线程对象
- Session 对象是非线程安全的
- 得到了一个session，相当于打开了一次数据库的连接

3.2.4 Transaction 接口

- Transaction是事务的接口
- 常用的方法
  - commit() :提交事务
  - rollback() :回滚事务
- 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 [JDBC 中是自动提交,不安全. 如果关闭了 Hibernate 中的事务,那么执行将不成功. 内部原理还是 JDBC 的事务, 开始事务之后在内部把 connection 的自动提交事务关闭了]
  - 只有产生了连接，才能进行事务的操作。所以只有有了session以后，才能有transaction

3.3 执行流程

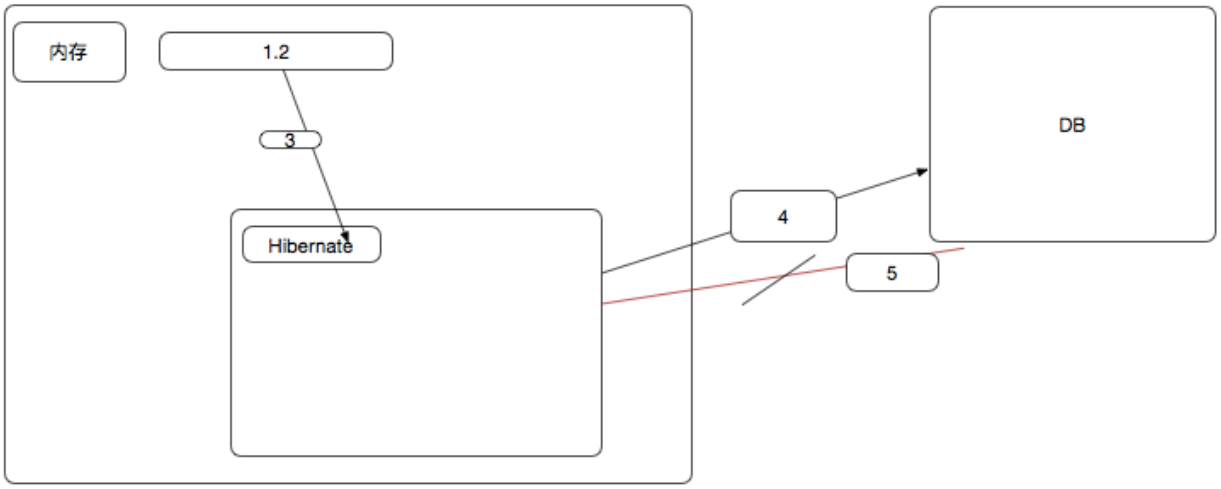
1. 读取 hibernate.cfg.xml 配置文件生成配置对象 Configuration
   - Configuration configuration = new Configuration();
   - configuration.config() / config(String resource);
2. Configuration对象根据当前的配置信息生成 SessionFactory 对象
   - SessionFactory sessionFactory = configuration.buildSessionFactory();
3. SessionFactory 创建 session [session 相当于 jdbc 中的]
   - sessionFactory.openSession();
4. 开启事务 完成 CUD 操作 | 完成查询
5. session 关闭

3.4 Hibernate 内部执行流程

四. 根据表生成持久化类和映射文件

五. 对象的状态

```
1. Person person = new Person();
2. person.setName("林教练");
3. session.save(person);
4. transaction.commit();
5. session.close();
```



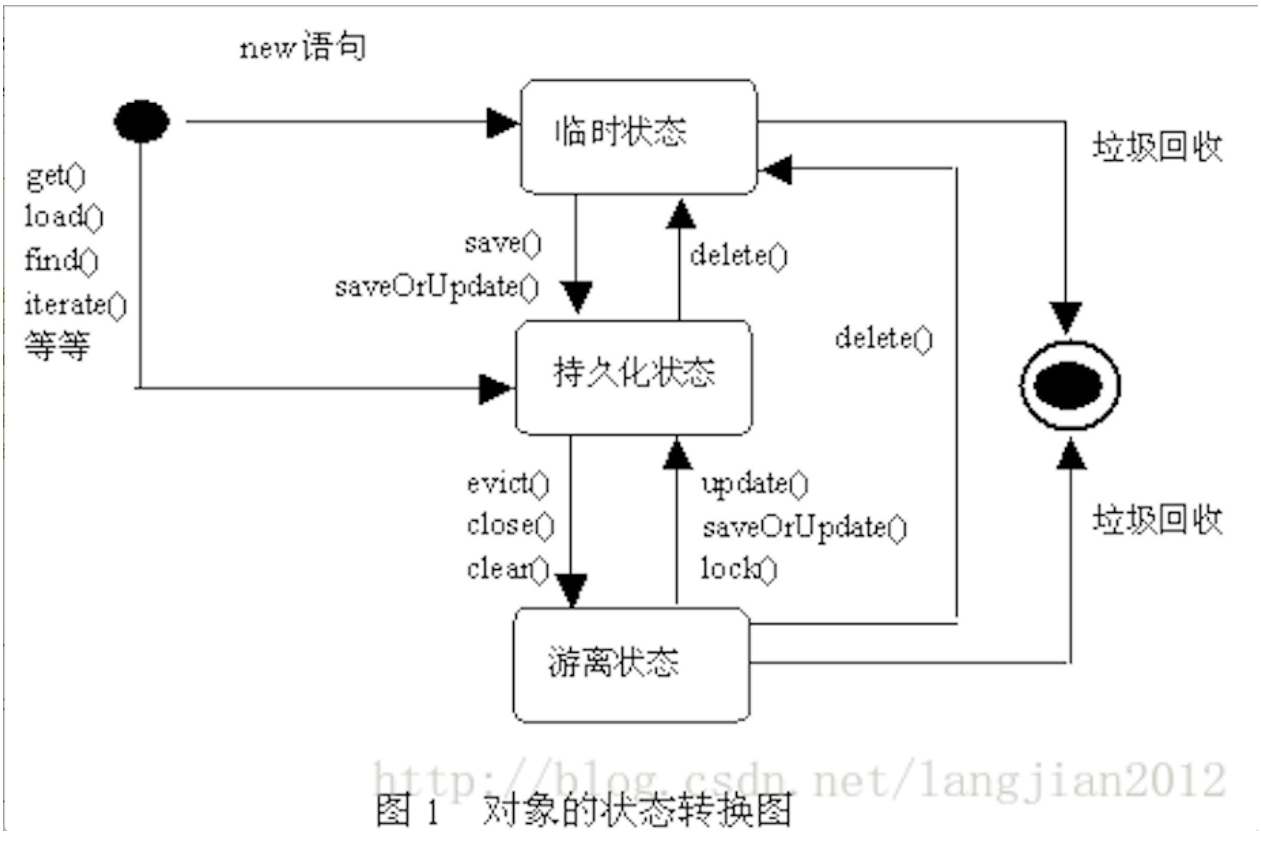


图 1 对象的状态转换图

5.1 save

该方法可以把一个对象从临时状态转换成持久化状态

```
public void saveState(){
    Session session = HibernateUtil.openSession();
    Transaction transaction = session.beginTransaction();
    Person per = new Person();
    per.setName("你");
    session.save(per);
    transaction.commit();
    session.close();
}
```

5.2 get

从数据库中根据主键提取出一个对象，该对象就是一个持久化状态的对象.

```
public void getState(){     Session session = HibernateUtil.openSession
();    Person per = session.get(Person.class, 1); // per 是一个持久化状态
                  session.close();}
```

5.3 update

把一个对象变成持久化状态
```
@Test
public void updateTest(){    Session session = HibernateUtil.openSession();    Transaction transaction = session.beginTransaction();      Person per = session.get(Person.class, 1); // per 是一个持久化状态      per.setName("2345sdfg");    // 需要执行 session.update 吗?    // * update 就是把一个对象变成持久化状态, 而 person 已经是持久化对象了. 所以不需要写.自动 update    transaction.commit();    session.close();}
```

```
###5.4 evict
把某一个对象从持久化状态转化为脱管状态
```

```
@Test
public void evictTest(){    Session session = HibernateUtil.openSession();    Transaction transaction = session.beginTransaction();    Person per = session.get(Person.class, 1); // per 是一个持久化状态      per.setName("2345sdfg");    session.evict(per);    // 没有执行 update 的语句, 因为执行了evict就变成托管状态      // 从脱管变成持久化状态
```

session.update(per); transaction.commit(); session.close();}
```

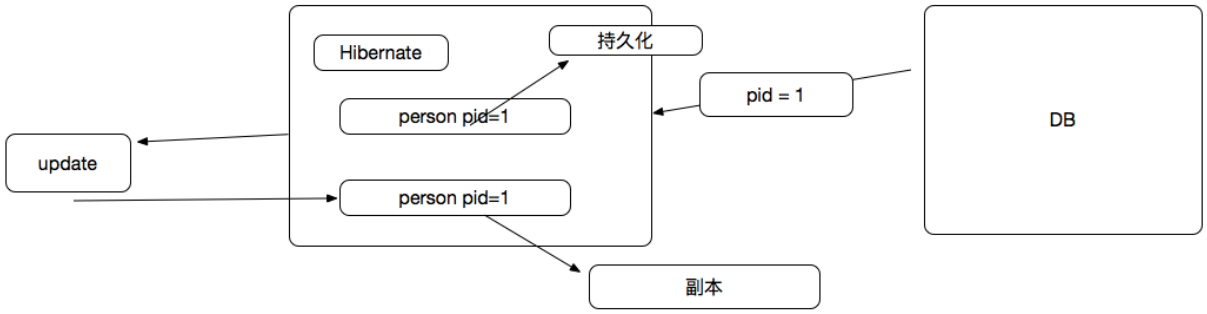
### 5.5 clear

把所有的 hibernate 中的持久化对象都转换成脱管状态的对像

```
@Testpublic void clearTest(){ Session session = HibernateUtil.openSession(); Transaction transaction = session.beginTransaction(); Person per = session.get(Person.class, 1); // per 是一个持久化状态 per.setName("2345sdfg"); Person per2 = new Person(); per2.setName("ddd"); session.save(per2); // per2 是一个持久化状态 session.clear(); // 把 hibernate 里面所有对象都变成了脱管状态对象 transaction.commit(); session.close();}
```

六. 对象的副本

> *克隆一份, 不是引用*
> *副本中的数据需要和数据库中保持一致*
> *一般情况下就是查询之后克隆一份*



flush

```
/*
  flush: 向数据库发送一列 sql 语句, 并且执行 sql. commit 才是提交. commit 中已经调用了 发送了 flush 方法.
*/
/*
   检查 hibernate 中的所有的持久化状态的对象,
   如果持久化状态的对象是由临时状态装换过来的,就发出 insert语句
   如果是由 get 方法获得的持久化状态的对象,
   查看副本[快照], 如果查看结果一致, 啥也不干
```
```

```
    如果不一样, 发出 update 语句.
*/
public void flush(){
        Session session = HibernateUtil.openSession();
        Transaction transaction = session.beginTransaction();
        // per1 啥也不干
        Person per1 = session.get(Person.class, 1L);
        // per2 进行修改
        Person per2 = session.get(Person.class, 2L);
        per2.setDes("姐姐妹妹们");
        // 新建 per3
        Person per3 = new Person("首博", "熊大2");
        session.save(per3);

        session.flush();

        transaction.commit();
        session.close();

    }
```