

# J2EE 轻量级框架

## 实验 P5

学号：SA16225221

姓名：欧勇

报告撰写时间：2017/1/14

## 1. 实验环境/器材

操作系统：Windows 10

IDE：Eclipse Kepler

SDK：JDK 1.8

Web Server: tomcat

数据库：MySQL 5.1.53

数据库可视化管理软件：Wamp Server

浏览器：Chrome 54.0.2840.87 m (64-bit)

## 2. 实验目的

搭建 SSH 开发环境，理解 SSH 程序开发基本概念和调试方法。

## 3. 实验内容

**1. Dependency Injection by Spring. The following diagrams are helpful to express your idea:**

**A. UML Class diagram**

**2. Integrate Hibernate with Spring**

## 4. 实验过程

本次实验五是在上次实验四的基础上修改而成，主要是在项目中添加了 `spring`，将 `HibernateUtil`、`User`、`UserDaoImpl` 三个类中配置在 `beans.xml` 文件中，将所有 `new HibernateUtil()`、`new User()`、`new UserDaoImpl()` 使用 `ctx.getBean(xxx, xxx.class)` 替代，其中 `ctx` 为 `spring` 上下文对象，最后分别测试三种查询方法，测试悲观锁，乐观锁，测试批处理删除用户（在实验四添加的 1000 个用户）。

悲观锁和乐观锁参考此博客：<http://www.blogjava.net/baoyaer/articles/203445.html>  
 批量添加用户参考了课件。

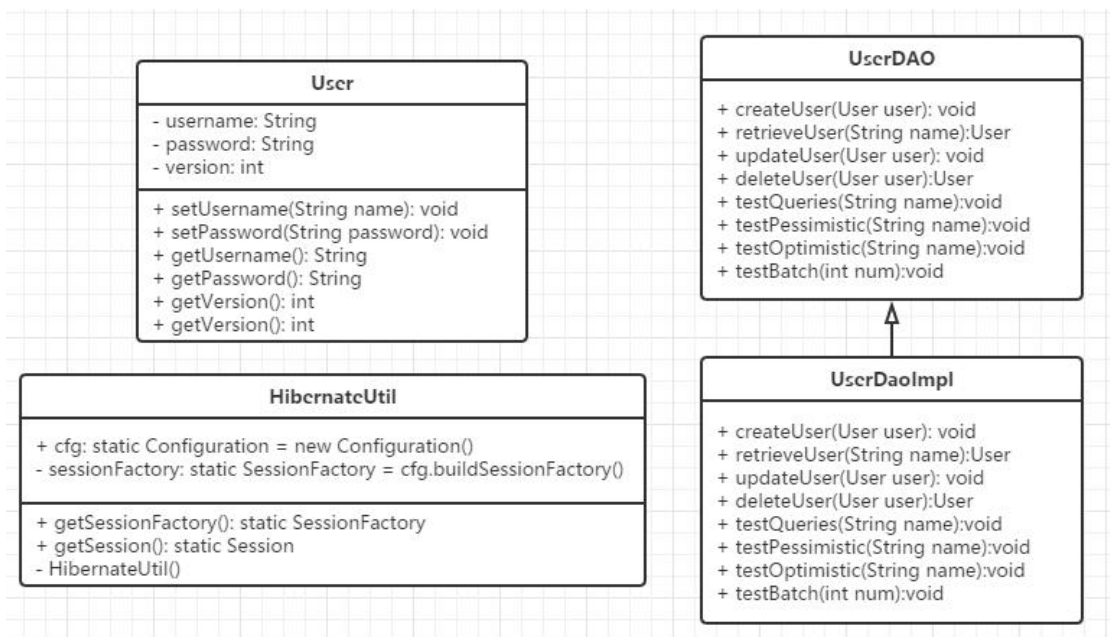


图 1. 类图

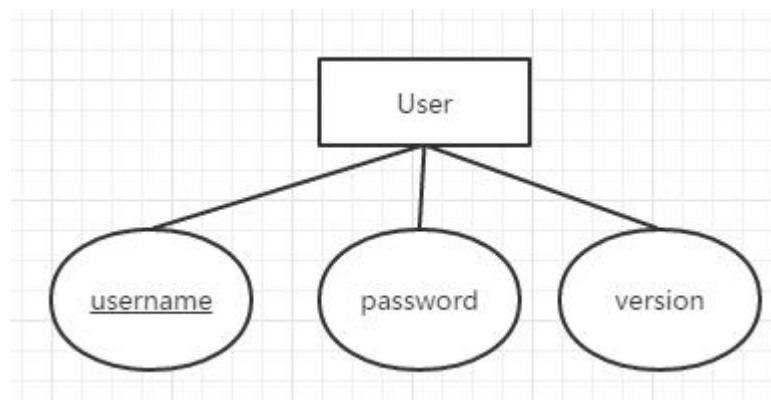


图 2. ER 模型（本次实验仅用到 User 类）

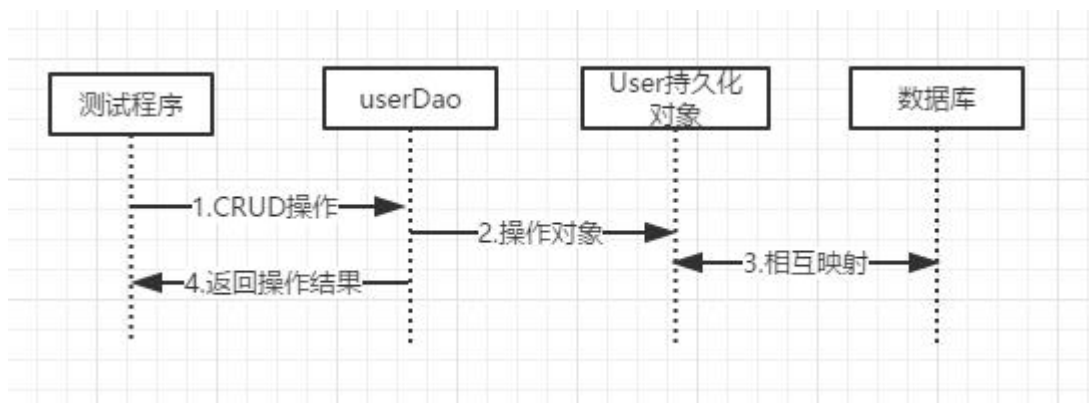


图 3. 顺序图

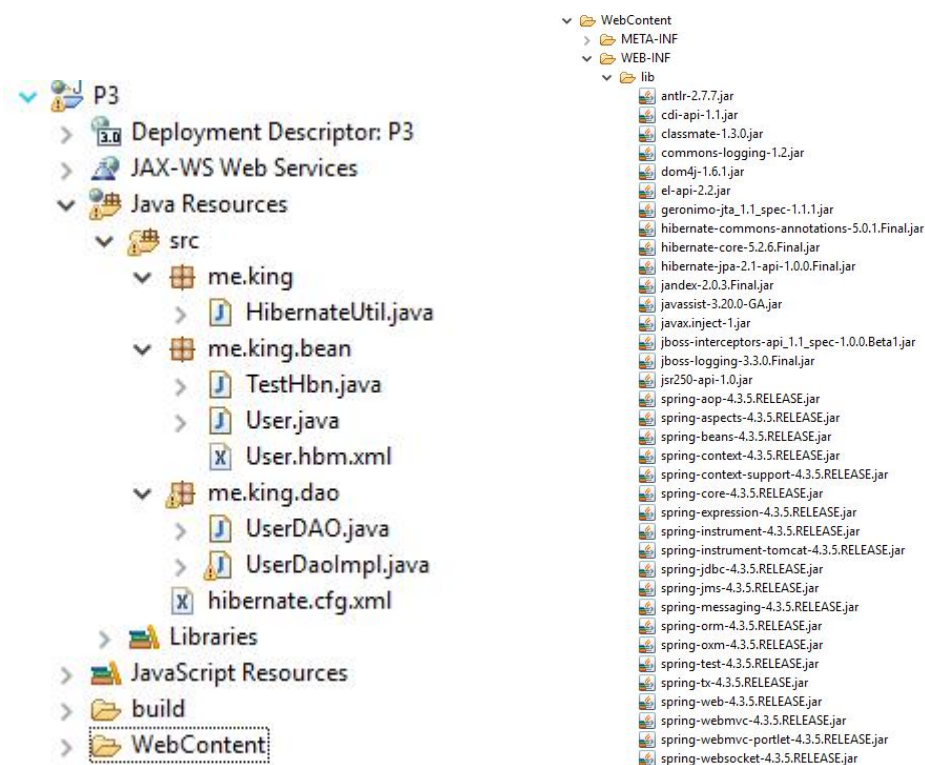


图 4. 项目文件结构图

虽然项目的文件结构为一个 J2EE 的动态 Web 项目，但是为了简单起见，本次实验测试的时候仅仅只用到了一个普通的 java 类 TestHbn 作为测试，而并没有使用 jsp 页面。

同时本次 P5 并没有对项目文件组织做出添加或删除，仅仅是添加了 spring 的 jar 包。



图 4.1 beans.xml 配置文件

分别将 User 类配置名为 user 的 bean，将 HibernateUtil 配置为 hbutil，将 UserDaoImpl 配置为 udimpl。

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE hibernate-configuration PUBLIC
3 "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
4 "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
5 <hibernate-configuration>
6 <!-- 去掉 name="foo" 异常解除 http://blog.csdn.net/dongkai_it/article/details/46352009 -->
7 <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
8 <property name="connection.url">jdbc:mysql://localhost:3306/monitor</property>
9 <property name="connection.username">root</property>
10 <property name="connection.password">hello</property>
11 <property name="dialect">org.hibernate.dialect.MySQLDialect</property> <!-- 指定SQL方言 -->
12 <property name="show_sql">>true</property> <!-- 在控制台输出SQL语句 -->
13 <mapping resource="me/king/bean/User.hbm.xml"/> <!-- 注册实体映射类 -->
14 </session-factory>
15 </hibernate-configuration>

```

图 5. hibernate.cfg.xml 配置文件

从第 7 行~第 10 行配置了数据库 Driver 类，数据库 url 地址（MySQL 为 url 格式），数据库连接用户名，数据库连接密码，

第 11 行指定了数据库方言（每种数据库的方言不太一样，最好指定使用的数据库的方言），

第 12 行配置在控制台输出 hibernate 自动生成的 sql 语句（方便调试），

在 13 行配置文件中注册与数据库表的实体映射类的配置文件位置，此处指定为 [me/king/bean/User.hbm.xml](#)

注意此处若在 session-factory 标签内指定 name 属性的话会报“错误解析 JNDI 名字”的错误。如下图 5.1，解决方法是不指定 name 属性

```

org.hibernate.engine.jndi.JndiException: Error parsing JNDI name [foo]
    at org.hibernate.engine.jndi.internal.JndiServiceImpl.parseName(JndiServiceImpl.java:124)
    at org.hibernate.engine.jndi.internal.JndiServiceImpl.bind(JndiServiceImpl.java:140)
    at org.hibernate.internal.SessionFactoryRegistry.addSessionFactory(SessionFactoryRegistry.java:88)
    at org.hibernate.internal.SessionFactoryImpl.<init>(SessionFactoryImpl.java:368)
    at org.hibernate.boot.internal.SessionFactoryBuilderImpl.build(SessionFactoryBuilderImpl.java:445)
    at org.hibernate.cfg.Configuration.buildSessionFactory(Configuration.java:710)
    at org.hibernate.cfg.Configuration.buildSessionFactory(Configuration.java:726)
    at me.king.HibernateUtil.<clinit>(HibernateUtil.java:15)
    at me.king.dao.UserDaoImpl.retrieveUser(UserDaoImpl.java:36)
    at me.king.bean.TestHbn.main(TestHbn.java:18)

```

图 5.1 JNDI 名字错误截图

由于本次实验中并没有使用到 JNDI 相关的东西，但在 session-factory 中配置 name 属性之后，Hibernate 试图将这个 sessionFacoty 注册到 JNDI 中，但是却无法解析配置的 name，所以报错。

解决方法参考此博客：[http://blog.csdn.net/dongkai\\_it/article/details/46352009](http://blog.csdn.net/dongkai_it/article/details/46352009)



```

1 <?xml version="1.0" encoding="utf-8"?>
2 <!-- 指定Hibernate映射文件的DTD信息 -->
3 <!DOCTYPE hibernate-mapping PUBLIC
4     "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
5     "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
6
7 <!-- hibernate-mapping是映射文件的根元素 -->
8 <hibernate-mapping package="me.king.bean">
9
10     <!-- 每个class元素对应一个持久化对象 -->
11     <!-- <class name="User" table="monitor_user" --> -->
12     <class name="User" table="monitor_user" >
13
14         <!-- id元素定义持久化类的标识属性 -->
15         <id name="username" column="user_name" />
16
17         <!-- version 元素定义版本属性,不能用property,否则会被解析为普通属性,且必须在id后面,否则会报无法解析version标签 -->
18         <version name="version" column="version" type="int" />
19
20         <!-- property元素定义常规属性 -->
21         <property name="password" column="user_password"/>
22     </class>
23
24 </hibernate-mapping>
    
```

图 6. User.hbm.xml 映射配置文件

如图 6 所示，在 User.hbm.xml 文件中配置了 User 类与数据库中表的映射关系，此文件的命名必须遵循 Xxx.hbm.xml 的格式，这样 hibernate 框架才会自动去帮助映射。

将 monitor\_user 表映射到 User 类上，id 标签定义持久化类的标识属性，即表的主键，property 定义其他的常规的属性，此处的定义标签中，若表的字段与持久化类属性名称不一致则需显式使用 column 属性配置对应的表字段，否则可以省略 column 属性。

version 标签定义版本,不能用 property, 否则会被解析为普通属性,而且此标签必须跟在 id 标签后面, 否则会报无法解析 version 标签的错误。

```

1 package me.king.bean;
2
3 public class User {
4     private String username; //主键, 用户名
5     private String password; //密码
6     private int version; //版本
7
8     public String getUsername() {
9         return username;
10    }
11
12    public void setUsername(String username) {
13        this.username = username;
14    }
15
16    public String getPassword() {
17        return password;
18    }
19
20    public void setPassword(String password) {
21        this.password = password;
22    }
23
24    public int getVersion() {
25        return version;
26    }
27
28    public void setVersion(int version) {
29        this.version = version;
30    }
31 }
    
```

图 7. 持久化类 User

由于 hibernate 的配置，此持久化类就是一个普通的 java 类，即 pojo，同时由于采用配置文件的方式，所以并没有使用注解。

```

1 package me.king.dao;
2
3 import me.king.bean.User;
4
5 public interface UserDAO {
6     public void createUser(User user); //C操作，创建
7     public User retrieveUser(String name); //R操作，查询
8     public void updateUser(User user); //U操作，更新
9     public void deleteUser(User user); //D操作，删除
10    //P4新加测试内容
11    public void testQueries(String name); //测试三种query的方法
12    public void testPessimistic(String name); //测试悲观锁
13    public void testOptimistic(String name); //测试乐观锁
14    public void testBatch(int num); //测试批处理num为数目
15 }

```

图 8. UserDAO 接口

```

1 package me.king.dao;
2
3 import me.king.HibernateUtil;
4
5 public class UserDaoImpl implements UserDAO{
6     //初始化spring上下文
7     private static ApplicationContext ctx = new ClassPathXmlApplicationContext("beans.xml");
8     public void createUser(User user) {}
9
10    public User retrieveUser(String name) {}
11
12    public void updateUser(User user) {}
13
14    public void deleteUser(User user) {}
15
16    //为了p4而特地实现的三个测试方法
17    public void testQueries(String name) {}
18
19    public void testPessimistic(String name) {}
20
21    public void testOptimistic(String name) {}
22
23    public void testBatch(int num) {}
24 }

```

图 9 UserDaoImpl 实现类概览

```

public void createUser(User user) {
    // TODO Auto-generated method stub
    Session s = null;
    Transaction tx = null;
    try{
        s = (ctx.getBean("hbutil", HibernateUtil.class)).getSession();
        tx = s.beginTransaction();
        s.save(user);
        tx.commit();
    }finally{
        if(s!=null){
            s.close();
        }
    }
}
}

```

```

public User retrieveUser(String name) {
    // TODO Auto-generated method stub
    Session s = null;
    try{
        s = (ctx.getBean("hbutil", HibernateUtil.class)).getSession();
        //采用HQL方式查询
        String sql = "from User as u where u.username='"+ name + "'";
        //此处HQL语句中表名应该是ORM映射的类名 而不是数据库的表名
        Query query = s.createQuery(sql);
        User hUser = (User) query.list().get(0);
        return hUser; //最后返回的是HQL查询的数据
    }finally{
        if(s!=null){
            s.close();
        }
    }
}
}

```

图 9.1 UserDaoImpl 实现类 createUser 和 retrieveUser 方法

图 9.1 中，采用 HQL 的方式查询，同时第 37 行需要注意 sql 语句的写法，表名需要使用 ORM 映射的类名而不是数据库的表名。否则会报 Xxxx is not mapped 的错误。如图 9.2

```

Caused by: org.hibernate.hql.internal.ast.QuerySyntaxException: monitor_User is not mapped [from monitor_User as u where u.username='test']
    at org.hibernate.hql.internal.ast.QuerySyntaxException.generateQueryException(QuerySyntaxException.java:79)
    at org.hibernate.QueryException.wrapWithQueryString(QueryException.java:103)
    at org.hibernate.hql.internal.ast.QueryTranslatorImpl.doCompile(QueryTranslatorImpl.java:217)
    at org.hibernate.hql.internal.ast.QueryTranslatorImpl.compile(QueryTranslatorImpl.java:141)
    at org.hibernate.engine.query.spi.HQLQueryPlan.<init>(HQLQueryPlan.java:115)
    at org.hibernate.engine.query.spi.HQLQueryPlan.<init>(HQLQueryPlan.java:77)
    at org.hibernate.engine.query.spi.QueryPlanCache.getHQLQueryPlan(QueryPlanCache.java:153)
    at org.hibernate.internal.AbstractSharedSessionContract.getQueryPlan(AbstractSharedSessionContract.java:545)
    at org.hibernate.internal.AbstractSharedSessionContract.createQuery(AbstractSharedSessionContract.java:654)
    ... 3 more
Caused by: org.hibernate.hql.internal.ast.QuerySyntaxException: monitor_User is not mapped
    at org.hibernate.hql.internal.ast.util.SessionFactoryHelper.requireClassPersister(SessionFactoryHelper.java:171)
    at org.hibernate.hql.internal.ast.tree.FromElementFactory.addFromElement(FromElementFactory.java:91)
    at org.hibernate.hql.internal.ast.tree.FromClause.addFromElement(FromClause.java:79)

```

图 9.2 Xxxx is not mapped 错误截图

解决方法参考博客：[http://blog.csdn.net/jsj\\_126abc/article/details/6582074](http://blog.csdn.net/jsj_126abc/article/details/6582074)

```

public void updateUser(User user) {
    // TODO Auto-generated method stub
    Session s = null;
    Transaction tx = null;
    try{
        s = (ctx.getBean("hbutil", HibernateUtil.class)).getSession();
        tx = s.beginTransaction();
        s.update(user);
        tx.commit();
    }finally{
        if(s!=null){
            s.close();
        }
    }
}
}

```



```

public void deleteUser(User user) {
    // TODO Auto-generated method stub
    Session s = null;
    Transaction tx = null;
    try{
        s = (ctx.getBean("hbutil", HibernateUtil.class)).getSession();
        tx = s.beginTransaction();
        s.delete(user);
        tx.commit();
    }finally{
        if(s!=null){
            s.close();
        }
    }
}
}

```

图 9.2. UserDaoImpl 实现类 updateUser 和 deleteUser 方法

```

public void testQueries(String name) {
    // TODO Auto-generated method stub
    Session s = null;
    try{
        s = (ctx.getBean("hbutil", HibernateUtil.class)).getSession();
        //采用HQL方式查询
        String sql = "from User as u where u.username='"+ name +"'";
        //此处HQL语句中表名应该是ORM映射的类名而不是数据库的表名
        Query query = s.createQuery(sql);
        User hUser = (User) query.list().get(0);

        //Criteria方法
        Criteria c = s.createCriteria(User.class);
        c.add(Restrictions.eq("username",name)); //找用户名为name的
        User cUser = (User) c.list().get(0);

        //Example查询
        User user = ctx.getBean("user", User.class);//new User();
        user.setUsername(name);
        c.add(Example.create(user));
        User eUser = (User) c.list().get(0); //虽然使用一个c, 但从新查询, 所以还是只有1个, 若填1会报越界错误

        System.out.println("Username/Password/version:");
        System.out.println("HQL,"+ hUser.getUsername() + " / "+ hUser.getPassword() + " / "+ hUser.getVersion());
        System.out.println("QBC,"+ cUser.getUsername() + " / "+ cUser.getPassword() + " / "+ cUser.getVersion());
        System.out.println("QBE,"+ eUser.getUsername() + " / "+ eUser.getPassword() + " / "+ eUser.getVersion());

    }finally{
        if(s!=null){
            s.close();
        }
    }
}
}

```

图 9.3. UserDaoImpl 实现类 testQueries 方法

以此使用 HQL、QBC、QBE 的方式查询用一个用户。

```

public void testPessimistic(String name) {
    // TODO Auto-generated method stub
    Session s = null;
    try{
        s = (ctx.getBean("hbutil", HibernateUtil.class)).getSession();
        //采用HQL方式查询
        String sql = "from User as u where u.username='"+ name +"";
        //此处HQL语句中表名应该是ORM映射的类名而不是数据库的表名
        Query query = s.createQuery(sql);
        //在HQL查询处,对所有别名为u的User记录加悲观锁,使用for update在数据库层实现,
        query.setLockMode("u", LockMode.UPGRADE);
        User hUser = (User) query.list().get(0);
        System.out.println( "Username/Password/version:");
        System.out.println( "HQL,"+ hUser.getUsername() + " / " + hUser.getPassword() + " / " + hUser.getVersion());
    }finally{
        if(s!=null){
            s.close();
        }
    }
}
}

```

图 9.4. UserDaoImpl 实现类 testPessimistic 方法

```

public void testOptimistic(String name) {
    // TODO Auto-generated method stub

    String sql = "from User as u where u.username='"+ name +"";
    //在session s之后新建一个session s2
    Session s = (ctx.getBean("hbutil", HibernateUtil.class)).getSession();
    Query query = s.createQuery(sql); //必须在s2创建之前查询,否则不会报错

    Session s2 = (ctx.getBean("hbutil", HibernateUtil.class)).getSession();
    Query query2 = s2.createQuery(sql);

    User user = (User) query.list().get(0);
    User user2 = (User) query2.list().get(0);

    System.out.println( "user: "+ user.getUsername() + " / " + user.getPassword() + " / " + user.getVersion());
    System.out.println( "user2: "+ user2.getUsername() + " / " + user2.getPassword() + " / " + user2.getVersion());

    try{
        Transaction tx = s.beginTransaction();
        Transaction tx2 = s2.beginTransaction();
        //使用user2更新user
        user2.setPassword("name2");
        s2.update(user2);
        tx2.commit(); //tx2提交后会更新,在tx.commit()处会报版本错误
        //此时查询最新的user
        User user3 = retrieveUser(name); //查看此时的user信息
        System.out.println( "user3: "+ user3.getUsername() + " / " + user3.getPassword() + " / " + user3.getVersion());
        //使用user更新user
        user.setPassword("name1");
        s.update(user);
        tx.commit();
    }finally{
        if(s!=null){
            s.close();
        }
        if(s2!=null){
            s2.close();
        }
    }
}
}

```

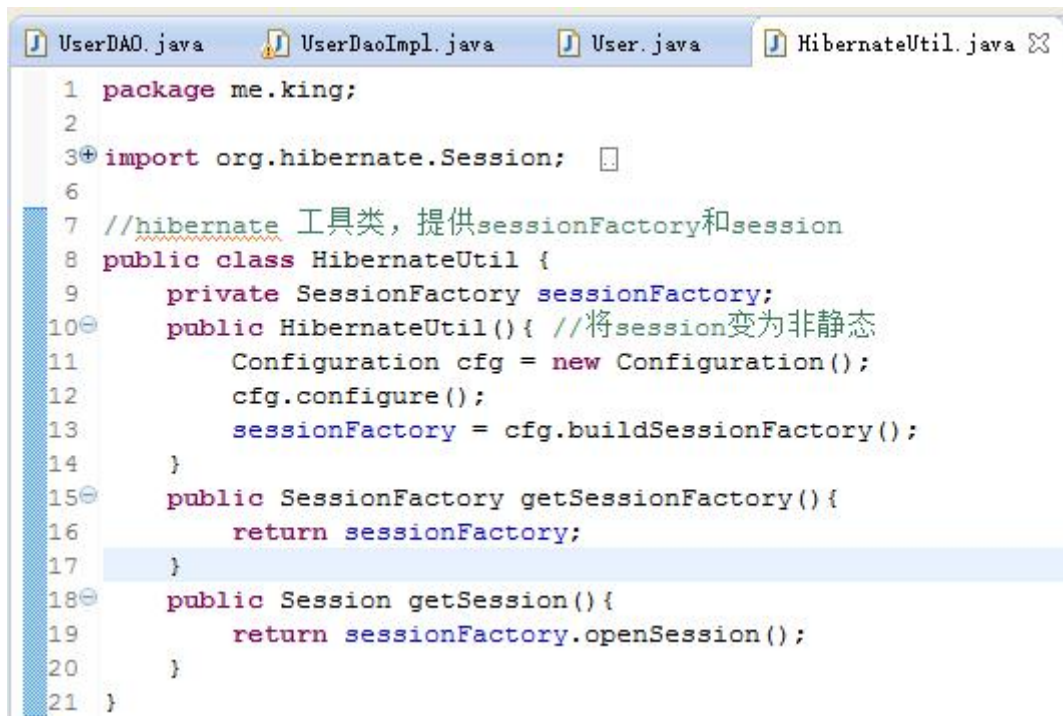
图 9.5. UserDaoImpl 实现类 testOptimistic 方法

```

public void testBatch(int num) {
    // TODO Auto-generated method stub
    Session s = null;
    try{
        s = (ctx.getBean("hbutil", HibernateUtil.class)).getSession();
        Transaction tx =s.beginTransaction();
        for ( int i=0; i<num; i++ ) {
            //User u = ctx.getBean("user", User.class);
            //u.setUsername("username"+i);
            //u.setPassword("username"+i);
            //s.save(u); //添加记录
            String sql = "from User as u where u.username='username'+ i +'";
            Query query = s.createQuery(sql);
            if(!query.list().isEmpty()){ //若查询结果为非空
                User u = (User) query.list().get(0);
                s.delete(u); //删除记录
            }
            if ( i % 20 == 0 ) { //20, 是因为与JDBC默认的批处理大小一样
                s.flush(); //使用flush方法将所有插入的用户写入数据库
                s.clear(); //将内存释放, 为下一次插入准备
            }
        }
        tx.commit();
    }finally{
        if(s!=null){
            s.close();
        }
    }
}

```

图 9.6. UserDaoImpl 实现类 testBatch 方法



```

UserDAO.java  UserDaoImpl.java  User.java  HibernateUtil.java
1 package me.king;
2
3+ import org.hibernate.Session;
6
7 //hibernate 工具类, 提供sessionFactory和session
8 public class HibernateUtil {
9     private SessionFactory sessionFactory;
10 public HibernateUtil(){ //将session变为非静态
11     Configuration cfg = new Configuration();
12     cfg.configure();
13     sessionFactory = cfg.buildSessionFactory();
14 }
15 public SessionFactory getSessionFactory(){
16     return sessionFactory;
17 }
18 public Session getSession(){
19     return sessionFactory.openSession();
20 }
21 }

```

图 10. HibernateUtil 工具类

为了测了乐观锁, 获取多个不同的 session, 所以将 P3 中 HibernateUtil 类的方法和属性

修改为非 static。

```

1 package me.king.bean;
2
3 import org.springframework.context.ApplicationContext;
4
5
6
7
8
9 public class TestHbn
10 {
11     public static void main(String[] args)
12         throws Exception
13     {
14         ApplicationContext ctx = new ClassPathXmlApplicationContext("beans.xml");//初始化spring上下文
15         UserDAO dao = ctx.getBean("udimpl",UserDaoImpl.class);
16         dao.testQueries("name");//测试三种query的方法
17         dao.testBatch(1000);//测试批处理num为数目
18         dao.testPessimistic("name");//测试悲观锁
19         dao.testOptimistic("name");//测试乐观锁
20     }
21 }
    
```

图 11. TestHbn 测试类

图 11.TestHbn 测试类，在类主函数 main 中，直接对本次实验需要的 hibernate CRUD 操作进行测试，使用 spring 将 udimpl “注射”到 main 函数中，测试方式如下：

测试三种查询方法，测试批处理添加用户，测试悲观锁，测试乐观锁。具体实现代码请看图 9.\* 系列截图。

```

INFO: HHH000397: Using ASTQueryTranslatorFactory
Hibernate: select user0_.user_Name as user_Nam1_0_, user0_.version as version2_0_, user0_.user_Password as user_Pas3_0_ from
Jan 14, 2017 3:34:36 PM org.hibernate.internal.SessionImpl createCriteria
WARN: HHH9000022: Hibernate's legacy org.hibernate.Criteria API is deprecated; use the JPA javax.persistence.criteria.Crit
Hibernate: select this_.user_Name as user_Nam1_0_0_, this_.version as version2_0_0_, this_.user_Password as user_Pas3_0_0_
Hibernate: select this_.user_Name as user_Nam1_0_0_, this_.version as version2_0_0_, this_.user_Password as user_Pas3_0_0_
Username/Password/version:
HQL,name / name2 / 3
QBC,name / name2 / 3
QBE,name / name2 / 3
    
```

图 12. 测试三种查询方法时控制台输出的 SQL 语句及辅助信息截图

可以看到 HQL、QBC、QBE 三种方式查询到的用户数据都是一样的。



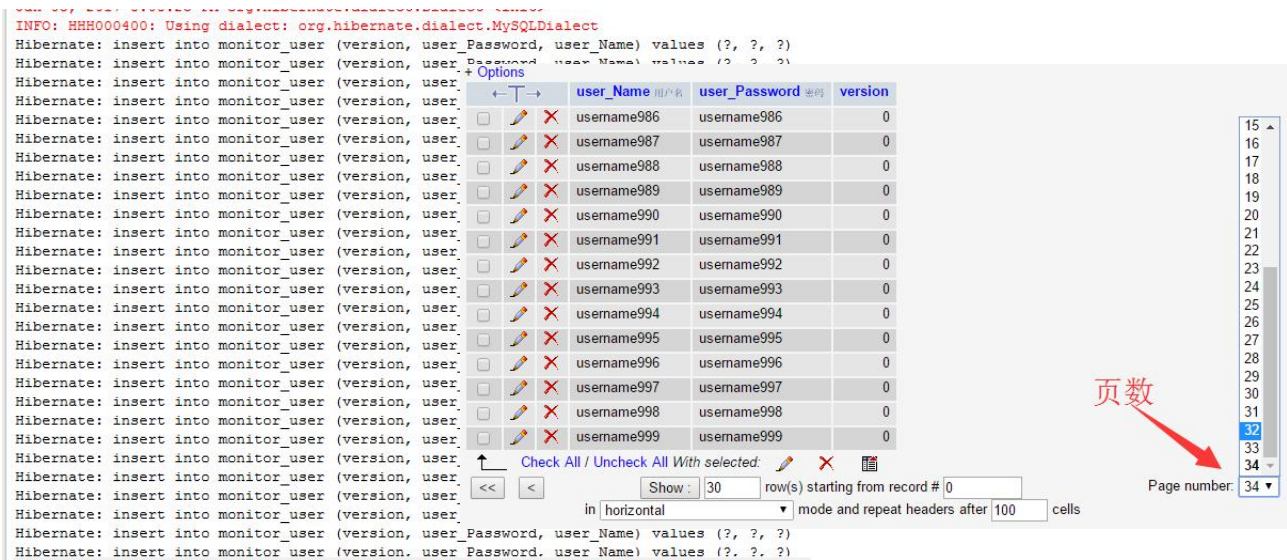


图 13. 测试批处理添加用户时控制台输出的 SQL 语句及数据库结果截图 (P4 内容)

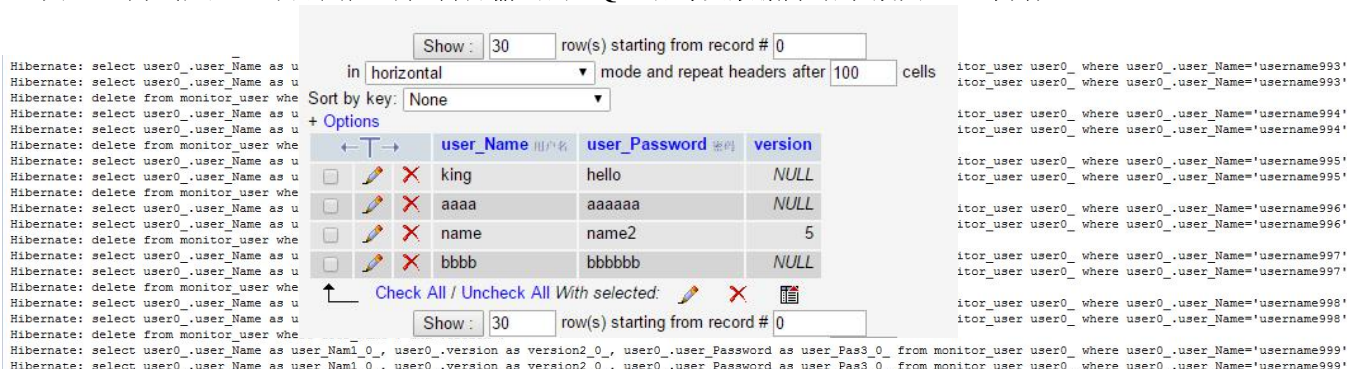


图 13.1 测试批处理删除用户时控制台输出的 SQL 语句及数据库结果截图 (P5 内容)

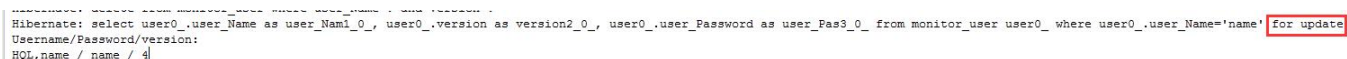


图 14. 测试悲观锁时控制台输出的 SQL 语句及辅助信息截图

注意看图 14 最后的 for update, 表示使用了数据库加锁的方式实现悲观锁。

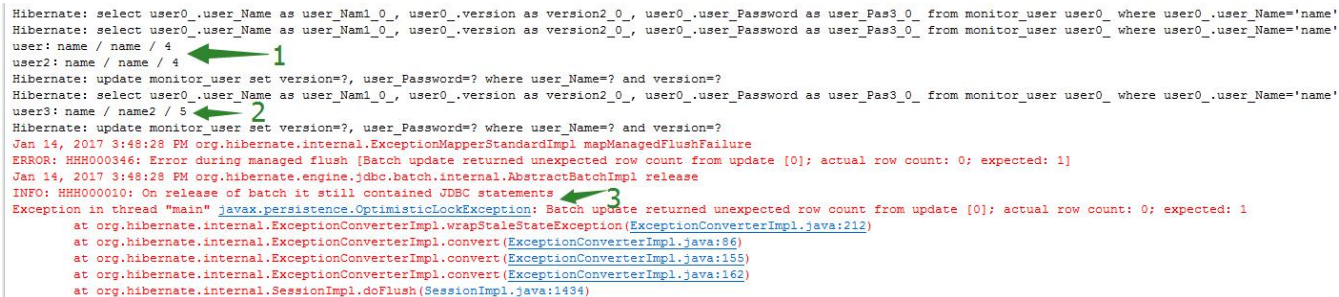


图 15. 测试乐观锁时控制台输出的 SQL 语句及辅助信息截图

图 15 中，在测试乐观锁的输出中：

绿 1 表示两个 session 都查询到相同的用户数据，

绿 2 表示此时 session2 已经执行完成，将密码更新为 name2，同时 version 为 5。

绿 3 表示此时报错的信息为，乐观锁异常，在程序中可以通过捕捉此异常以处理它。

## 5. 实验总结

对概念/方法的理解与总结，实验碰到的问题及解决方法.....

由于仅仅用到了最基本的 spring 的方法和功能，本次 spring+hibernate 的集成实验基本没有遇到任何问题。

本次实验五并没有在 web.xml 中配置 listener 标签，即：

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/applicationContext*.xml,classpath*:applicationContext*.xml</param-value>
</context-param>
<listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
```

因为没有采用 struts 且使用了手动加载 beans.xml 文件，所以不需要将 spring 的 ContextLoaderListener 当作 struts 的监听器来自动加载 beans.xml 文件。