阅读笔记4

第五章
征服数据库

Spring帮助处理底层的数据访问
这样就可以专注应用程序的管理
Spring中推荐的是OO原则 也就是面向接口编程
Dao层的是提供数据读取与写入数据库的一种方式 应该用接口的方式发布功能 而其他层就可以通过接口来进行访问
这样子就会使得服务对象容易被测试 因为不与特定数据的访问实现绑定在一起 还可以为这些接口创建mock实现 这样不用连接数据库就可以测试
提升测试效率 Dao层是对持久化无关的方式进行访问的 具体方法只通过接口来进行发布 这样可以将持久化框架对应用的影响变的最小 这样子也不会造成耦合 导致僵化的设计

由于在JDBC中 数据库出错都是由SQLException
这个异常没什么真正的能够提示的东西 所以我们需要在更明显的提示
这个时候我们可以用hibernate的异常提示 但是这样就将hibernate的这个持久层渗透到应用程序的其他部分了
但是这个时候我们有一种选择 这个就是spring为我们设置的异常 **spring是无关持久化的 而且spring的异常不是检查型的异常** 是不需要捕捉的 不用写catch块

数据访问模板化
数据访问的时候有的是可变的有的是不可变的 这个时候我们将不可变的设置为模板在内部集成好 至于可变的我们自己的数据访问逻辑 也就是变成了回调 是可以由我们指定参数或自己写一些spring没有内置的逻辑来实现的

针对不同的持久化平台 spring提供了多个可选的模板

表 5.2 Spring 提供的数据访问模板，分别适用于不同的持久化机制

模板类（org.springframework.*）	用途
jca.cci.core.CciTemplate	JCA CCI 连接
jdbc.core.JdbcTemplate	JDBC 连接
jdbc.core.namedparam.NamedParameterJdbcTemplate	支持命名参数的 JDBC 连接
jdbc.core.simple.SimpleJdbcTemplate	通过 Java 5 简化后的 JDBC 连接
orm.hibernate.HibernateTemplate	Hibernate 2.x 的 Session
orm.hibernate3.HibernateTemplate	Hibernate 3.x 的 Session
orm.ibatis.SqlMapClientTemplate	iBATIS SqlMap 客户端
orm.jdo.JdoTemplate	Java 数据对象（Java Data Object）实现
orm.jpa.JpaTemplate	Java 持久化 API 的实体管理器

用的时候只需配置为Spring上下文中的Bean并将其织入到应用程序的DAO中

我们可以直接继承模板但是 Spring还提供了一种方式 就是DAO支持类
当编写应用程序自己的DAO实现时 可以自继承DAO支持类并调用模板获取方法来直接访问底层的数据访问模板
如果想访问持久化平台 每个DAO支持类都能够访问其余数据库进行通信的类

表 5.3 Spring DAO 支持类提供了便捷的方式来使用数据访问模板

DAO 支持类（org.springframework.*）	为谁提供 DAO 支持
jca.cci.support.CciDaoSupport	JCA CCI 连接
jdbc.core.support.JdbcDaoSupport	JDBC 连接
jdbc.core.namedparam.NamedParameterJdbcDaoSupport	带有命名参数的 JDBC 连接
jdbc.core.simple.SimpleJdbcDaoSupport	用 Java 5 进行了简化的 JDBC 连接
orm.hibernate.support.HibernateDaoSupport	Hibernate 2.x 的 Session
orm.hibernate3.support.HibernateDaoSupport	Hibernate 3.x 的 Session
orm.ibatis.support.SqlMapClientDaoSupport	iBATIS SqlMap 客户端
orm.jdo.support.JdoDaoSupport	Java 数据对象（Java Data Object）实现
orm.jpa.support.JpaDaoSupport	Java 持久化 API 的实体管理器

配置数据源
通过JDBC驱动程序定义的数据源
通过JNDI查找的数据源

连接池的数据源
三种方式 其中的JDBC不用多说 这个是最基础的
对于即将发布到生产环境的应用程序 建议是连接池的数据源
这本书是推荐JNDI来获取池中的数据源

关于JNDI 这是一种部署在web服务器上的数据源 这种好处在于可以脱离应用程序的管理 这样应用程序只需要访问数据库的时候查找数据源就行了 这个在这里我没有进行部署 留到以后进行再进行学习 这里在Spring中主要是使用一个<jee:jndi-lookup>元素来进行检索JNDI中的任何对象（包括数据源）并将其应用到Spring bean中

例如： 如果应用程序的数据源配置在JNDI中 使用如下进行装配到Spring中

```
<jee:jndi-lookup id="datasource" jndi-name="/jdbc/SpitterDS" resource-ref="true"  ></jee:jndi-lookup>
```

jbdi-name属性用于指定JNDI中资源的名字 如果只设置了jndi-name属性 那么就会根据指定的名称查找数据源 但是 如果应用程序运行在java应用程序服务器中 则需要将resource-ref属性设置为true 这样给定的jndi-name将会自动添加java:comp/env/前缀

使用数据源连接池
这里介绍的是DBCP
这是配置的例子

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
    <property name="driverClassName" value="com.mysql.jdbc.Driver"></property>
    <property name="url" value="jdbc:mysql://127.0.0.1:3306/imooc?useUnicode=true&amp;charact
erEncoding=utf-8"/>
     <property name="username" value="root"/>
    <property name="password" value="199666"/>
    <property name="initialSize" value="5"/>
    <property name="maxActive" value="10"/>
</bean>
```

表 5.4 BasicDataSource 的池配置属性

池配置属性	所指定的内容
initialSize	池启动时创建的连接数量
maxActive	同一时间可从池中分配的最多连接数，如果设置为 0，表示无限制
maxIdle	池里不会被释放的最多空闲连接数，如果设置为 0，表示无限制
maxOpenPreparedStatements	在同一时间能够从语句池中分配的预处理语句的最大数量，如果设置为 0，表示无限制
maxWait	在抛出异常之前，池等待连接回收的最大时间（当没有可用连接时），如果设置为 -1，表示无限等待
minEvictableIdleTimeMillis	连接在池中保持空闲而不被回收的最大时间
minIdle	在不创建新连接的情况下，池中保持空闲的最小连接数
poolPreparedStatements	是否对预处理语句进行池管理（布尔值）

还有最基本的JDBC驱动的数据源配置

```
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">

    <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
    <property name="url" value="jdbc:mysql://127.0.0.1:3306/imooc?useUnicode=true&amp;charact
erEncoding=utf-8"></property>
    <property name="username" value="root"/>
    <property name="password" value="liuziye"/>
</bean>
```

下面的是Spring中JDBC模板的在xml中的具体写法

这个是在spring xml中具体的配置 以及在具体的Dao类中进行使用模板的具体写法(测试无误)

```
<bean id="jdbcTemplate"
class="org.springframework.jdbc.core.simple.SimpleJdbcTemplate">
   <constructor-arg ref="dataSource" />
</bean>

<!--<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
      <property name="driverClassName" value="com.mysql.jdbc.Driver"></property>
      <property name="url" value="jdbc:mysql://127.0.0.1:3306/imooc?useUnicode=true&amp;chara
```

```
cterEncoding=utf-8"/>
      <property name="username" value="root"/>
      <property name="password" value="199666"/>
      <property name="initialSize" value="5"/>
      <property name="maxActive" value="10"/>
  </bean>-->

<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
      <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
      <property name="url" value="jdbc:mysql://localhost:3306/spitter"></property>
      <property name="username" value="root"/>
      <property name="password" value="liuziye"/>
  </bean>
  <bean id="spitterDap" class="Spitter.Spitter_JdbcTemplete">
      <property name="jdbcTemplate" ref="jdbcTemplate"/>
  </bean>
```

```
@Test
public  void addSpitter(Spitter spitter){

jdbcTemplate.update(SQL_INSERT_SPITTER,spitter.getUsername(),
spitter.getPassword(),
spitter.getFullName(),
spitter.getEmail(),
spitter.isUpdateByEmail());
spitter.setId(queryForIdentity());
}
```

```
public Spitter getSpitterById(long id) {
return jdbcTemplate.queryForObject(//<co id="co_query"/>
SQL_SELECT_SPITTER_BY_ID,
            new ParameterizedRowMapper<Spitter>() {
public Spitter mapRow(ResultSet rs, int rowNum)
throws SQLException {
                    Spitter spitter = new Spitter();//<co id="co_map"/>
spitter.setId(rs.getLong(1));
spitter.setUsername(rs.getString(2));
spitter.setPassword(rs.getString(3));
spitter.setFullName(rs.getString(4));
                    return spitter;
}
            },
id //<co id="co_bind"/>
);
}
```

```
private long queryForIdentity() {
return jdbcTemplate.queryForLong("call identity()");
}
```

这是用使用命名参数来进行更新操作 而不是原来的使用索引方式
这是sql的写法 这样写才能与下面的对应起来

```
private  String SQL_INSERT_SPITTER="INSERT  INTO  spitter(username ,password,fullName)  " +"
VALUE (:username ,:password,:fullname) ";
```

```
public  void addSpitter(Spitter spitter){
    Map<String,Object> params=new HashMap<String, Object>();
params.put("username",spitter.getUsername());
params.put("password",spitter.getPassword());
params.put("fullname",spitter.getFullName());
jdbcTemplate.update(SQL_INSERT_SPITTER,params);
spitter.setId(queryForIdentity());

}
```

使用Spring中的JDBC Dao支持类

首先要这样继承Dao支持类

```
public class Spitter_JdbcTemplete extends SimpleJdbcTemplate{
```

这样就可以将addSpitter方法改成这样了

```
@Test
public  void addSpitter(Spitter spitter){

    getSimpleJdbcTemplate().update(SQL_INSERT_SPITTER,
2,
spitter.getUsername(),
spitter.getPassword(),
spitter.getFullName(),
spitter.getEmail(),
spitter.isUpdateByEmail());
}
```

注意getSImpleJdbcTemplate ( )
这样我们还可以在Spring xml中省时省力了
我们可以将我们的Dao类配置成这样
直接抛开JdbcTemplete的Bean的配置 直接配置dataSource

```
<bean id="spitterDap" class="Spitter.Spitter_JdbcTemplete">
    <property name="dataSource" ref="dataSource"/>
</bean>
```

这样就可以省掉我们在Bean中直接配置JDBCTemlete的Bean然后注入给Dao了 这样子比较方便 但是这样子就导致我们Bean原来设计的理念相反了 这样增加了耦合 但是这样的耦合增加了 碍不碍事就仁者见仁 智者见智了

下面是关于spring中HIbernate的使用
有意思的地方在于这里的hibernate是不需要配置hibernate.cfg.xml文件的 是我们在Spring中手动设置的 这个与Mybatis集成Spring是不一样的

有俩种方式
一种是通过像JDBC一样通过模板来实现的 主要的作用有一个就是管理Hiberna的Session的 涉及打开与关闭必须是一个Session 保证每个相同的事务使用同一个Session
但是HibernateTemplate的容易存在一定程度上的入侵性 也就是耦合

Hibernate3引入了上下文Session 这是HIbernate本身保证每个事务使用同一个Session的方案
所以第一种不是实现Hibernate的最佳方式了

下面我们分别将第一种与第二种的代码列下
只列关键的代码 关于Hibernate的是不列的
这是HibernateTemplate的

```
package com.habuma.spitter.persistence;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.orm.hibernate3.HibernateTemplate;
import org.springframework.stereotype.Component;

import com.habuma.spitter.domain.Spitter;
import com.habuma.spitter.domain.Spittle;

@Component
public class HibernateSpitterDao  {
  @Autowired
private HibernateTemplate template;

  public void addSpitter(Spitter spitter) {
    template.saveOrUpdate(spitter);
}

public Spitter getSpitterById(long id) {
return template.get(Spitter.class, id);
```

```
}

public void saveSpitter(Spitter spitter) {
    template.update(spitter);
}

public List<Spittle> getRecentSpittle() {
return template.loadAll(Spittle.class); // this isn't right...just a placeholder for now
}

public void saveSpittle(Spittle spittle) {
    template.save(spittle);
}

public List<Spittle> getSpittlesForSpitter(
          Spitter spitter) {
// TODO Auto-generated method stub
return null;
}


public Spitter getSpitterByUsername(String username) {
// TODO Auto-generated method stub
return null;
}

public void deleteSpittle(long id) {
    template.delete(getSpittleById(id));
}

public Spittle getSpittleById(long id) {
return template.get(Spittle.class, id);
}

public List<Spitter> findAllSpitters() {
// TODO Auto-generated method stub
return null;
}
}
```

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
      http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd">

<!--<start id="bean_sessionFactory" />-->
<bean id="sessionFactory"
class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
              <property name="dataSource" ref="dataSource" />
              <property name="mappingResources">
                     <list>
                           <value>SpitterEntity.hbm.xml</value>
                     </list>
              </property>
              <property name="hibernateProperties">
                     <props>
                           <prop key="dialect">org.hibernate.dialect.MySQLDialect</prop>
                     </props>
              </property>
       </bean>


       <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
              <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
              <property name="url" value="jdbc:mysql://localhost:3306/spitter"></property>
```

```
            <property name="username" value="root"/>
            <property name="password" value="liuziye"/>
      </bean>

<!--<end id="bean_sessionFactory" />-->

      <!--<start id="bean_hibernateTemplate" />-->
<bean id="hibernateTemplate" class="org.springframework.orm.hibernate3.HibernateTemplate">
            <property name="sessionFactory" ref="sessionFactory" />
      </bean>
<!--<end id="bean_hibernateTemplate" />-->




      <context:component-scan base-package="Spitter" />
</beans>
```

这上面俩个就是主要代码了

下面是Hibernate的不使用Templete的方式 来由自己直接注入SessionFactory 来直接实现的第二种方式 与第一种只是改了一部分代码

```
package Spitter;

import java.util.List;
import org.hibernate.SessionFactory;
import org.hibernate.classic.Session;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;



@Repository
public class HibernateSpitterDao  {

public SessionFactory getSessionFactory() {
return sessionFactory;
}

private SessionFactory sessionFactory;

@Autowired
public HibernateSpitterDao(SessionFactory sessionFactory) {//<co id="co_injectSessionFactory"
/>
this.sessionFactory = sessionFactory;
}

private Session openSession(){
return  sessionFactory.openSession();
}

public  void addSpitter(SpitterEntity spitter){
    openSession().save(spitter);
}

public SpitterEntity getSpitterById(long id) {
return (SpitterEntity) openSession().get(SpitterEntity.class, id);//<co id="co_useSession"/>
}



public static void main(String[] args) {
    ApplicationContext context =new ClassPathXmlApplicationContext("hibernate_persitence-cont
ext.xml");
HibernateSpitterDao hibernateSpitterDao= (HibernateSpitterDao) context.getBean("hibernateSpit
terDao");
/*  SessionFactory sessionFactory=  hibernateSpitterDao.getSessionFactory();
    System.out.println(sessionFactory+"    \t" +"");*/
SpitterEntity spitterEntity= hibernateSpitterDao.getSpitterById(1);
```

```
System.out.println(spitterEntity.getId());
System.out.println(spitterEntity.getUsername());
System.out.println(spitterEntity.getPassword());









}
}
```

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
      http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd">

<!--<start id="bean_sessionFactory" />-->
<bean id="sessionFactory"
class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
              <property name="dataSource" ref="dataSource" />
              <property name="mappingResources">
                     <list>
                            <value>SpitterEntity.hbm.xml</value>
                     </list>
              </property>
              <property name="hibernateProperties">
                     <props>
                            <prop key="dialect">org.hibernate.dialect.MySQLDialect</prop>
                     </props>
              </property>
       </bean>


       <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
              <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
              <property name="url" value="jdbc:mysql://localhost:3306/spitter"></property>
              <property name="username" value="root"/>
              <property name="password" value="liuziye"/>
       </bean>

<!--<end id="bean_sessionFactory" />-->

       <!--<start id="bean_hibernateTemplate" />-->
      <!-- <bean id="hibernateTemplate" class="org.springframework.orm.hibernate3.HibernateTemplate">
              <property name="sessionFactory" ref="sessionFactory" />
       </bean>-->
       <!--<end id="bean_hibernateTemplate" />-->



<bean class=
                     "org.springframework.dao.annotation.PersistenceExceptionTranslationPostProcessor"/>


       <context:component-scan base-package="Spitter" />
</beans>
```

基本上就改了一部分代码 但这是不依赖Spring的HIbernate代码 无耦合 比较推荐的方式

下面是关于JPA的使用 关于JPA 这个我也比较懵逼 有时间再看吧 反正是弄出来了
JPA
是提供了俩种类型的实体管理器
一个是应用程序管理器类型 不适合在Java EE容器中独立运行的应用程序
一个是容器管理类型 这个是在JAVA EE容器中创建与管理的 容器可以保留一些自己的配置 最适合JAVA EE容器
这俩种都是实现了一个接口 也就是EntityManager接口
关于Spring集成JPA的博客 http://www.cnblogs.com/jacksun1978/archive/2012/04/11/2443096.html

关于JPA JPA是Java EE5规范之一，是一个orm规范，由厂商来实现该规范。目前有hibernate，OpenJPA，TopLink和EclipseJPA等实现

Spring提供三种方法集成JPA：

1、LocalEntityManagerFactoryBean：适用于那些仅使用JPA进行数据访问的项目。该FactoryBean根据 JPA PersistenceProvider自动检测配置文件进行工作，一般从"META-INF/persistence.xml"读取配置信息。这种方式最简单，但是不能设置Spring中定义的DataSource，且不支持Spring管理的全局事务。不建议使用此方式。这种方法实际上只适用于独立的应用程序和测试环境（这正是JPA规范设计它的原因）。

2 从JNDI中获取：用于从Java EE服务器中获取指定的EntityManagerFactory，这种方式在Spring事务管理时一般要使用JTA事务管理。
3 LocalContainerEntityManagerFactoryBean：适用于所有环境的FactoryBean，能全面控制EntityManagerFactory配置，非常适合那种需要细粒度定制的环境。

关于Spring与JPA
第一种LocalEntityManagerFactoryBean
这个绝大部分的配置文件都是来源一个persistence.xml的配置文件
这个文件必须位于类路径下的META-INF目录下
persostence.xml列出一个或多个持久化类以及其他的配置 如数据源 和基于XML的配置文件
由于配置信息基本都在persistence.xml
所以这个在Spring配置的非常少了
只需要配置一个一个实体工厂

```
<bean id="emf" class=
        "org.springframework.orm.jpa.LocalEntityManagerFactoryBean">
<!-- <property name="dataSource" ref="dataSource" />-->
<property name="jpaVendorAdapter" ref="jpaVendorAdapter" />
</bean>
```

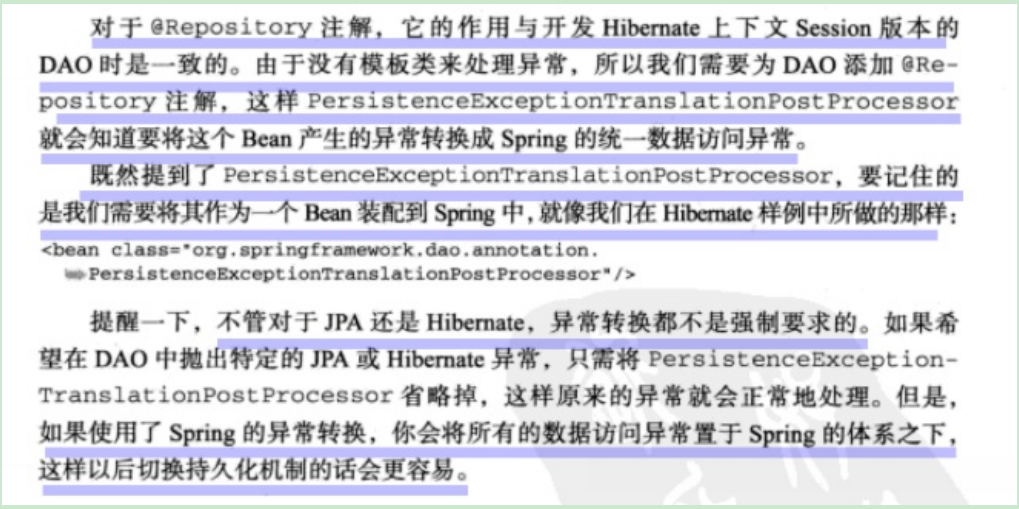
创建应用管理类型的EntityManagerFactory都是在persistence.xml中进行的 而这正是应用程序的本意
在应用程序管理的场景下 完全由应用程序本身来负责获取EntityManagerFactory 这是通过JPA实现的PersistenceProvider做到的
这样比较好 不用我们自己来定义持久化单元 由JPA自动在这个特定的位置查找持久化单元定义
这种方式其实Spring做不了什么

下面是使用容器管理类型的JPA
容器管理的JPA采取了一种不同的方式 当在容器中运行时 可以使用容器（我们这里使用的是Spring）提供的信息来生成EntityManagerFactory
我们可以将数据源信息配置在Spring上下文中 而不是persistence.xml中
jpaVendorAdapter属性用于指明所使用的是哪一个厂商的JPA实现 在下面的例子中 我们使用的hibernate的JPA实现
最重要的是database属性 可以由我们自己设置了

表 5.5 Hibernate 的 JPA 适配器支持多种数据库，可以通过其属性配置使用哪个数据库

数据库平台	属性 database 的值
IBM DB2	DB2
Apache Derby	DERBY
H2	H2
Hypersonic	HSQL
Informix	INFORMIX
MySQL	MYSQL
Oracle	ORACLE
PostgresQL	POSTGRESQL
Microsoft SQL Server	SQLSERVER
Sybase	SYBASE

一些特定的动态持久化功能需要对持久化类按照指令进行修改才能支持 在延迟加载（只在它们被实际访问到时才从数据库中进行获取）的对象中 必须要有知道如何查询未加载数据的代码
一些框架使用动态代理来实现延迟加载 而一些框架像JDO 则是在编译时执行类指令



对于 @Repository 注解，它的作用与开发 Hibernate 上下文 Session 版本的 DAO 时是一致的。由于没有模板类来处理异常，所以我们需要为 DAO 添加 @Repository 注解，这样 PersistenceExceptionTranslationPostProcessor 就会知道要将这个 Bean 产生的异常转换成 Spring 的统一数据访问异常。

既然提到了 PersistenceExceptionTranslationPostProcessor，要记住的是我们需要将其作为一个 Bean 装配到 Spring 中，就像我们在 Hibernate 样例中所做的那样：

```
<bean class="org.springframework.dao.annotation.
  ➥PersistenceExceptionTranslationPostProcessor"/>
```

提醒一下，不管对于 JPA 还是 Hibernate，异常转换都不是强制要求的。如果希望在 DAO 中抛出特定的 JPA 或 Hibernate 异常，只需将 PersistenceExceptionTranslationPostProcessor 省略掉，这样原来的异常就会正常地处理。但是，如果使用了 Spring 的异常转换，你会将所有的数据访问异常置于 Spring 的体系之下，这样以后切换持久化机制的话会更容易。

其实使用哪种实体管理器工厂主要取决如何使用它 对于简单的应用程序来讲
上面的LocalEntityManagerFactoryBean 就足够了
但是LocalContainerEntityManagerFactoryBean 这个能够更多在Spring中配置JPA 所以对于生产级的使用场景是很有吸引力的

下面还有单独的一个JNDI获取实体管理器工厂

```
<jem:jndi-lookup id="emf" jndi-name="persistence/spitterPU />
```

可以这样通过JNDI来获取引用
这是persistence.xml的所有内容

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persist
ence/persistence_1_0.xsd"
version="1.0">
   <persistence-unit name="spitterPU" transaction-type="RESOURCE_LOCAL">
      <class>Spitter.Spitter</class>
<!--      <class>com.habuma.spitter.domain.Spittle</class>-->
<properties>
<!-- <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect"/>
&lt;!&ndash;         <property name="hibernate.hbm2ddl.auto" value="create-drop"/>&ndash;&gt;
         <property name="hibernate.connection.driver_class" value="com.mysql.jdbc.Driver" />
-->
                      <!--配置数据库用户名 -->
                     <!--  <property name="hibernate.connection.username" value="root" />
                      &lt;!&ndash;配置数据库密码 &ndash;&gt;
                       <property name="hibernate.connection.password" value="liuziye" />
                     &lt;!&ndash;配置数据库url &ndash;&gt;
                      <property name="hibernate.connection.url" value="jdbc:mysql://localho
st:3306/spitte-->r" />
<!--设置外连接抓取树的最大深度 -->
<property name="hibernate.max_fetch_depth" value="3" />
<!--自动输出schema创建DDL语句 -->
<property name="hibernate.hbm2ddl.auto" value="update" />
      </properties>
   </persistence-unit>
</persistence>
```

好吧 设置的非常少 在spring我们来设置JPA配置的具体内容 管理JPA的事务 数据连接 数据显示方式 是否输出SQL语句 这些选项

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:jem="http://www.springframework.org/schema/jee"
xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.or
g/schema/beans/spring-beans-2.0.xsd
      http://www.springframework.org/schema/context http://www.springframework.org/schema/con
text/spring-context.xsd http://www.springframework.org/schema/jee http://www.springframework.
```

```
org/schema/jee/spring-jee.xsd">

       <bean id="emf" class=
              "org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
             <property name="dataSource" ref="dataSource" />
             <property name="jpaVendorAdapter" ref="jpaVendorAdapter" />
       </bean>

       <bean id="jpaVendorAdapter"
class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
             <property name="database" value="MYSQL" />
             <property name="showSql" value="true"/>
             <property name="generateDdl" value="false"/>
             <property name="databasePlatform"
value="org.hibernate.dialect.MySQLDialect" />
       </bean>

<!--<end id="bean_hibernateAdapter"/>-->
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
        <property name="url" value="jdbc:mysql://localhost:3306/spitter"></property>
        <property name="username" value="root"/>
        <property name="password" value="liuziye"/>
       </bean>

       <bean class=
                     "org.springframework.dao.annotation.PersistenceExceptionTranslationPostP
rocessor"
/>
    <bean id="transactionManager"
class="org.springframework.orm.jpa.JpaTransactionManager">
        <property name="entityManagerFactory" ref="emf" />
    </bean>

       <context:component-scan base-package="Spitter" />
<!-- <jem:jndi-lookup id="emf" jndi-name="persistence/spitterPU />-->
```

下面就是JPA的具体Dao类了

```
//<start id="java_contextualJpaDao"/>
package Spitter;
import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.dao.DataAccessException;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;



@Repository("spitterDao")
@Transactional
public class JpaSpitterDao     {
private static final String RECENT_SPITTLES =
"SELECT s FROM Spittle s";
  private static final String ALL_SPITTERS =
"SELECT s FROM Spitter s";
  private static final String SPITTER_FOR_USERNAME =
"SELECT s FROM Spitter s WHERE s.username = :username";
  private static final String SPITTLES_BY_USERNAME =
"SELECT s FROM Spittle s WHERE s.spitter.username = :username";

@PersistenceContext
```

```
private EntityManager em; //<co id="co_injectEM"/>

public void addSpitter(SpitterEntity spitter) {
em.persist(spitter); //<co id="co_useEM"/>
}

public SpitterEntity getSpitterById(long id) {
return em.find(SpitterEntity.class, id); //<co id="co_useEM"/>
}

public void saveSpitter(SpitterEntity spitter) {
em.merge(spitter); //<co id="co_useEM"/>
}
//<end id="java_contextualJpaDao"/>
@SuppressWarnings("unchecked")
public List<Spittle> getRecentSpittle() {
return (List<Spittle>) em.createQuery(RECENT_SPITTLES).
        getResultList();
}

public void saveSpittle(Spittle spittle) {
em.persist(spittle);
}

@SuppressWarnings("unchecked")
public List<Spittle> getSpittlesForSpitter(
          SpitterEntity spitter) {
return (List<Spittle>) em.createQuery(SPITTLES_BY_USERNAME).
        setParameter("username", spitter.getUsername()).
        getResultList();
}


public SpitterEntity getSpitterByUsername(String username) {
return (SpitterEntity) em.createQuery(SPITTER_FOR_USERNAME).
        setParameter("username", username).
        getSingleResult();
}

public void deleteSpittle(long id) {
try {
em.remove(getSpittleById(id));
} catch(DataAccessException e) {}
  }

public Spittle getSpittleById(long id) {
return em.find(Spittle.class, id);
}

@SuppressWarnings("unchecked")
public List<SpitterEntity> findAllSpitters() {
return em.createQuery(ALL_SPITTERS).getResultList();
}

public static void main(String[] args) {
    ApplicationContext context=new ClassPathXmlApplicationContext("JPA-Spring.xml");
JpaSpitterDao jpaSpitterDao= (JpaSpitterDao) context.getBean("spitterDao");
SpitterEntity spitterEntity= jpaSpitterDao.getSpitterById(1);
System.out.println(spitterEntity.getId());
System.out.println(spitterEntity.getUsername());
System.out.println(spitterEntity.getPassword());
System.out.println(spitterEntity.getEmail());

}


}
```

注意@PerisitenceContext这个注解 这个注解是自动装配Spring xml中的Bean id为emf的
注入的是实体管理器，执行持久化操作的，需要配置文件persistence.xml。

@Transactional注解 这个是表示这个DAO中的持久化方法是在事务上下文执行的
至于@Repository注解 是与在hibernate中我们使用的效果是一样 代替原来的异常处理 用Spring提供的异常处理
这个不是强制要求的东西
这三者缺一不可（测试了几下） 另外jar与jar直接的不兼容也非常的重要
需要明白 spring-jpa是与spring-orm是不兼容的

上面这种是LocalContainerEntityManagerFactoryBean
下面我又弄了一种基本的应用管理
也就是LocalEntityManagerFactoryBean

注意我们用hibernate实现的时候 用的是hibernate帮我们集成好的东西 hibernate独有的查询 HQL 或者直接是save get load这些方法
而且还必须得有对应的类映射xml
在Spring中只需要配置如下就好了

```
<bean id="emf" class=
        "org.springframework.orm.jpa.LocalEntityManagerFactoryBean">
<!--<property name="dataSource" ref="datasourcePool" />-->
    <!--<property name="jpaVendorAdapter" ref="jpaVendorAdapter" />-->
<property name="persistenceUnitName" value="userPU"></property>
</bean>



<bean class=
             "org.springframework.dao.annotation.PersistenceExceptionTranslationPostProcessor"
/>

<bean id="tx"
class="org.springframework.orm.jpa.JpaTransactionManager">
    <property name="entityManagerFactory" ref="emf" />
</bean>

<tx:annotation-driven transaction-manager="tx" />



<context:component-scan base-package="com.Spring.Spring_in_action.chapter5_6"/>
```

在properties中配置如下

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-i
nstance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persist
ence_2_0.xsd"
version="2.0">
    <persistence-unit name="userPU" transaction-type="RESOURCE_LOCAL">
        <properties>
            <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect"/>
            <property name="hibernate.connection.driver_class" value="com.mysql.jdbc.Driver"></property>
            <property name="hibernate.connection.username" value="root"></property>
            <property name="hibernate.connection.password" value="liuziye"></property>
            <property name="hibernate.connection.url" value="jdbc:mysql://localhost:3306/spitter?useUnico
de=true&amp;characterEncoding=UTF-8"></property>
            <property name="hibernate.hbm2ddl.auto" value="update"/>
        </properties>
    </persistence-unit>
</persistence>
```

具体的测试类

```
@Repository
@Transactional
public class TestJpaDao {

@PersistenceContext
private EntityManager em;
```

```
private static final   String USER_FOR_ID="Select id,username,userage,userAddress  from user u  where u.id= 1";
private  static  final  String  listgetUser ="Select id,username,userage,userAddre  ss  from user  ";



public User getUser(int id) {

return (User) em.find(User.class, id);

    }



public static void main(String[] args) {
ApplicationContext context = new ClassPathXmlApplicationContext("Spring/Spring in action/Spring_in_action3.xml");
TestJpaDao testJpaDao = (TestJpaDao) context.getBean("testJpaDao");
System.out.println(testJpaDao.getUser(1));
    }
}
```

注意我这里没有粘贴关于hibernate集成映射文件的那陪部分 在用的时候小心

这里将jar包粘贴下来 这个是没有不兼容的部分的jar

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>goOverCeshi01</groupId>
    <artifactId>goOverCeshi_codesource</artifactId>
    <version>1.0-SNAPSHOT</version>
    <properties>

    </properties>
    <dependencies>


        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-test</artifactId>
            <version>4.1.1.RELEASE</version>
            <scope>test</scope>
        </dependency>


        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>4.1.1.Final</version>
        </dependency>


        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-c3p0</artifactId>
            <version>4.1.1.Final</version>
        </dependency>
```

```
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-entitymanager</artifactId>
            <version>4.1.1.Final</version>
        </dependency>




        <dependency>
            <groupId>commons-collections</groupId>
            <artifactId>commons-collections</artifactId>
            <version>3.2.1</version>
        </dependency>







        <dependency>
            <groupId>dom4j</groupId>
            <artifactId>dom4j</artifactId>
            <version>1.6.1</version>
        </dependency>

        <dependency>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
            <version>1.2.16</version>
        </dependency>

        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>1.6.1</version>
        </dependency>

        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-nop</artifactId>
            <version>1.6.4</version>
        </dependency>

        <dependency>
            <groupId>javassist</groupId>
            <artifactId>javassist</artifactId>
            <version>3.12.1.GA</version>
        </dependency>

        <dependency>
            <groupId>org.springframework.data</groupId>
            <artifactId>spring-data-jpa</artifactId>
            <version>1.5.0.RELEASE</version>
        </dependency>


<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.38</version>
        </dependency>


        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>3.3.1</version>
        </dependency>

<!--Spring-->
<dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
```

```
            <version>3.1.1.RELEASE</version>
        </dependency>



        <dependency>
            <groupId>commons-collections</groupId>
            <artifactId>commons-collections</artifactId>
            <version>3.2.1</version>
        </dependency>

        <dependency>
            <groupId>commons-dbcp</groupId>
            <artifactId>commons-dbcp</artifactId>
            <version>1.2.2</version>
        </dependency>

        <dependency>
            <groupId>commons-logging</groupId>
            <artifactId>commons-logging</artifactId>
            <version>1.1</version>
        </dependency>


        <dependency>
            <groupId>commons-pool</groupId>
            <artifactId>commons-pool</artifactId>
            <version>1.6</version>
        </dependency>

        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis-spring</artifactId>
            <version>1.2.2</version>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-aop</artifactId>
            <version>3.1.1.RELEASE</version>
        </dependency>

<!-- https://mvnrepository.com/artifact/org.aspectj/aspectjweaver -->
<dependency>
            <groupId>org.aspectj</groupId>
            <artifactId>aspectjweaver</artifactId>
            <version>1.8.6</version>
        </dependency>


        <dependency>
            <groupId>aspectj</groupId>
            <artifactId>aspectjrt</artifactId>
            <version>1.5.3</version>
        </dependency>

        <dependency>
            <groupId>aopalliance</groupId>
            <artifactId>aopalliance</artifactId>
            <version>1.0</version>
        </dependency>


        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-asm</artifactId>
            <version>3.1.1.RELEASE</version>
        </dependency>


        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-aspects</artifactId>
            <version>3.1.1.RELEASE</version>
        </dependency>
```

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-beans</artifactId>
    <version>3.1.1.RELEASE</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context-support</artifactId>
    <version>3.1.1.RELEASE</version>
</dependency>



<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>3.1.1.RELEASE</version>
</dependency>


<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-expression</artifactId>
    <version>3.1.1.RELEASE</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-instrument-tomcat</artifactId>
    <version>3.1.1.RELEASE</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-instrument</artifactId>
    <version>3.1.1.RELEASE</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>3.1.1.RELEASE</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jms</artifactId>
    <version>3.1.1.RELEASE</version>
</dependency>


<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-oxm</artifactId>
    <version>3.1.1.RELEASE</version>
</dependency>


<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>3.1.1.RELEASE</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
    <version>3.1.1.RELEASE</version>
</dependency>


<dependency>
```

```
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc-portlet</artifactId>
            <version>3.1.1.RELEASE</version>
        </dependency>


        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>3.1.1.RELEASE</version>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
            <version>3.1.1.RELEASE</version>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-struts</artifactId>
            <version>3.1.1.RELEASE</version>
        </dependency>




        <dependency>
            <groupId>cglib</groupId>
            <artifactId>cglib</artifactId>
            <version>2.2</version>
        </dependency>



        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-orm</artifactId>
            <version>3.1.1.RELEASE</version>
        </dependency>


    </dependencies>



<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.5.1</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
                <encoding>UTF-8</encoding>
            </configuration>
        </plugin>
    </plugins>
</build>
</project>
```