## 第5章

### **1、P64**

将图5.2修改为：



### **2、P73**

（1）将第4行代码“@Repository("TestDao")”修改为：

@Repository("testDao")

（2）将TestService接口的代码修改为：

package com.statement.service;

public interface TestService {

public void test();

}

（3）将TestServiceImpl实现类的代码修改为：

package com.statement.service;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import com.statement.dao.TestDao;

@Service("testService")

public class TestServiceImpl implements TestService{

@Autowired

private TestDao testDao;

@Override

public void test() {

String deleteSql ="delete from user";

String saveSql = "insert into user values(?,?,?)";

Object param[] = {1,"chenheng","男"};

testDao.delete(deleteSql, null);

testDao.save(saveSql, param);

//插入两条主键重复的数据

testDao.save(saveSql, param);

}

}

### **3、P74**

将StatementController类的代码修改为：

package com.statement.controller;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import com.statement.service.TestService;

@Controller

public class StatementController {

@Autowired

private TestService testService;

public void test() {

testService.test();

}

}

### **4、P76**

将测试类XMLTest的代码修改为：

package com.statement.test;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.statement.controller.StatementController;

public class XMLTest {

public static void main(String[] args) {

ApplicationContext appCon =

new ClassPathXmlApplicationContext("/com/statement/xml/XMLstatementapplicationContext.xml");

StatementController ct = (StatementController)appCon.getBean("statementController");

ct.test();

}

}

### **5、P76**

倒数第8-11行删除

### **6、P78**

将添加@Transactional注解后的TestServiceImpl类的代码修改为：

package com.statement.service;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import com.statement.dao.TestDao;

@Service("testService")

@Transactional

//加上注解@Transactional,就可以指定这个类需要受Spring的事务管理

//注意@Transactional只能针对public属性范围内的方法添加

public class TestServiceImpl implements TestService{

@Autowired

private TestDao testDao;

@Override

public void test() {

String deleteSql ="delete from user";

String saveSql = "insert into user values(?,?,?)";

Object param[] = {1,"chenheng","男"};

testDao.delete(deleteSql, null);

testDao.save(saveSql, param);

//插入两条主键重复的数据

testDao.save(saveSql, param);

}

}

### **7、P78**

新增5.3.3节

### **5.3.3 如何在事务处理中捕获异常**

声明式事务处理的流程是：（1）Spring根据配置完成事务定义，设置事务属性。（2）执行开发者的代码逻辑。（3）如果开发者的代码产生异常（如主键重复）并且满足事务回滚的配置条件，则事务回滚；否则，事务提交。（4）事务资源释放。

现在的问题是，如果开发者在代码逻辑中加入了try...catch...语句，Spring还能不能在声明式事务处理中正常得到事务回滚的异常信息？答案是不能。例如，我们将5.3.1节和5.3.2节中TestServiceImpl实现类的test方法的代码修改如下：

@Override

public void test() {

String deleteSql ="delete from user";

String saveSql = "insert into user values(?,?,?)";

Object param[] = {1,"chenheng","男"};

try {

testDao.delete(deleteSql, null);

testDao.save(saveSql, param);

//插入两条主键重复的数据

testDao.save(saveSql, param);

} catch (Exception e) {

System.out.println("主键重复，事务回滚。");

}

}

这时，我们再运行测试类，发现主键重复但事务并没有回滚。这是因为默认情况下，Spring只在发生未被捕获的RuntimeExcetpion时才回滚事务。现在，如何在事务处理中捕获异常呢？下面从声明式事务管理的两种实现方式来说明。

**1．在基于XML方式的声明式事务管理中捕获异常**

在基于XML方式的声明式事务管理中捕获异常，需要补充两个步骤。

（1）修改声明事务的配置

针对5.3.1节，我们需要将XMLstatementapplicationContext.xml文件中的代码“<tx:method name="\*"/>”修改为：

<tx:method name="\*" rollback-for="java.lang.Exception"/>

<!-- rollback-for属性指定回滚生效的异常类，多个异常类逗号分隔；no-rollback-for属性指定回滚失效的异常类-->

（2）在catch语句中添加“throw new RuntimeException();”语句，如下代码：

@Override

public void test() {

String deleteSql ="delete from user";

String saveSql = "insert into user values(?,?,?)";

Object param[] = {1,"chenheng","男"};

try {

testDao.delete(deleteSql, null);

testDao.save(saveSql, param);

//插入两条主键重复的数据

testDao.save(saveSql, param);

} catch (Exception e) {

System.out.println("主键重复，事务回滚。");

throw new RuntimeException();

}

}

**2．在基于@Transaction注解的声明式事务管理中捕获异常**

在基于@Transaction注解的声明式事务管理中，也同样需要补充两个步骤。

（1）修改@Transactional注解

针对5.3.2节，我们需要将TestServiceImpl类中的@Transactional注解修改为：

@Transactional(rollbackFor= {Exception.class})

//rollbackFor指定回滚生效的异常类，多个异常类逗号分隔;

//noRollbackFor指定回滚失效的异常类

（2）也需要在catch语句中添加“throw new RuntimeException();”语句。

**注意：**在实际工程应用中，经常在catch语句中添加“TransactionAspectSupport.currentTransactionStatus().setRollbackOnly();”语句即可。也就是说，不需要在XML配置文件tx:method元素中添加rollback-for属性或在@Transaction注解中添加rollbackFor属性。

## 第7章

### **1、P107**

倒数第二行的“tinyint(2)”修改为“int(10)”

### **2、P108**

第四行的“tinyint(2)”修改为“int(10)”

### **3、P108**

第七行的“tinyint(2)”修改为“int(10)”

### **4、P113**

第十二行的“tinyint(2)”修改为“int(10)”

### **5、P113**

第十四行的“tinyint(2)”修改为“int(10)”

### **6、P117**

倒数第十行的“tinyint(2)”修改为“int(10)”

### **7、P117**

倒数第2、3、4行的“tinyint(2)”修改为“int(10)”