# Spring的数据库编程

Spring框架为开发者提供了JDBC模板模式，即jdbcTemplate，它可以简化许多代码，但在实际应用中jdbcTemplate并不常用，在工作 中更多的时候是使用Hibernate框架和MyBatis框架进行数据库编程。（在此讲Spring JDBC主要是为了实战Spring的事务管理功能，因为还未学习MyBatis，所以暂时以Spring JDBC模板来操作数据库，演示Spring事务管理。）

## Spring JDBC的配置

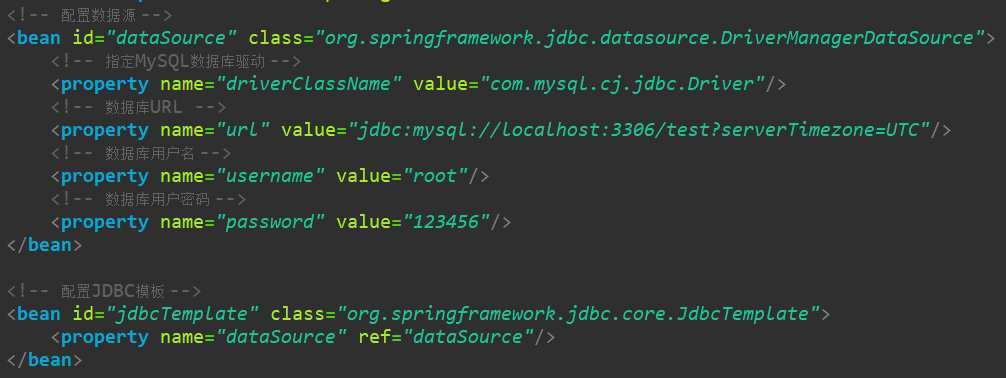
**1、导入Spring JDBC包**

使用Spring JDBC模板将Spring安装包中的spring-jdbc.5.0.2.RELEASE.jar包导入到 WEB-INF/lib目录中。

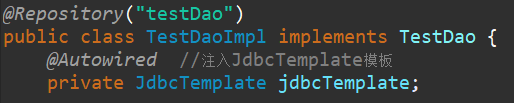


该jar包中有两个核心包，core包和datasource包，core包是JDBC 的核心功能包，包 括常用的JdbcTemplate类；datasource包是访问数据库的工具类包。

**2、配置数据源和Spring JDBC模板**



**3、注入JdbcTemplate**

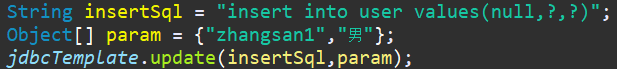


## JdbcTemplate的常用方法

将JdbcTemplate注入到持久层后，通过JdbcTemplate类完成数据库的增删查改操作。 JdbcTemplate类常用的方法是update和query。

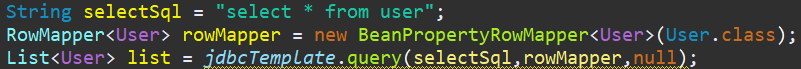
1、**public int update( String sql , Object args [])**

该方法可以对数据表进行增加、修改、删除等操作。使用args[]设置SQL语句中的 参数，并返回更新的行数。示例代码如下：



2、**public List<T> query( String sql , RowMapper<T> rowMapper , Object args[] )**

该方法可以对数据表进行查询操作。rowMapper将结果映射到用户自定义的类中， 但前提是自定义的类中的属性要与数据表的字段对应。示例代码如下：

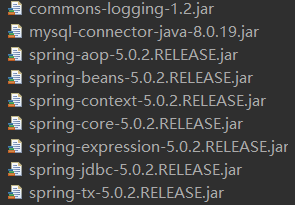


## 使用示例

通过一个示例演示Spring JDBC的使用。

1. **创建应用并导入JAR包**

创建一个名为ch5的Web应用，将Spring框架的4个核心包，1个依赖包、MySQL驱动的JAR包、Spring JDBC的JAR包以及Spring事务处理的JAR包复制到应用的WEB-INF/lib目录下。如下：



导入spring-jdbc是因为要配置事务管理器，导入spring-aop和spring-tx包是为了实现Spring事务管理。

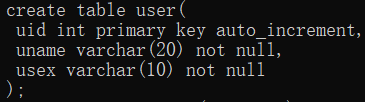
1. **创建并编辑配置文件**

在src目录下创建Spring配置文件applicationContext.xml文件，并在其中配置数据源和JDBC模板。具体代码如下：



**3）创建数据表及对应的实体类**

在src目录下创建com.ch5包，在该包中创建实体类MyUser。注意，该类的属性与 数据表user的字段一致。数据表user的结构如下图：



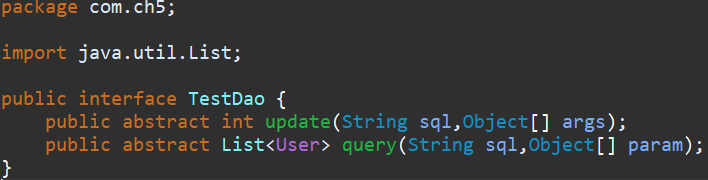
实体类User代码如下：



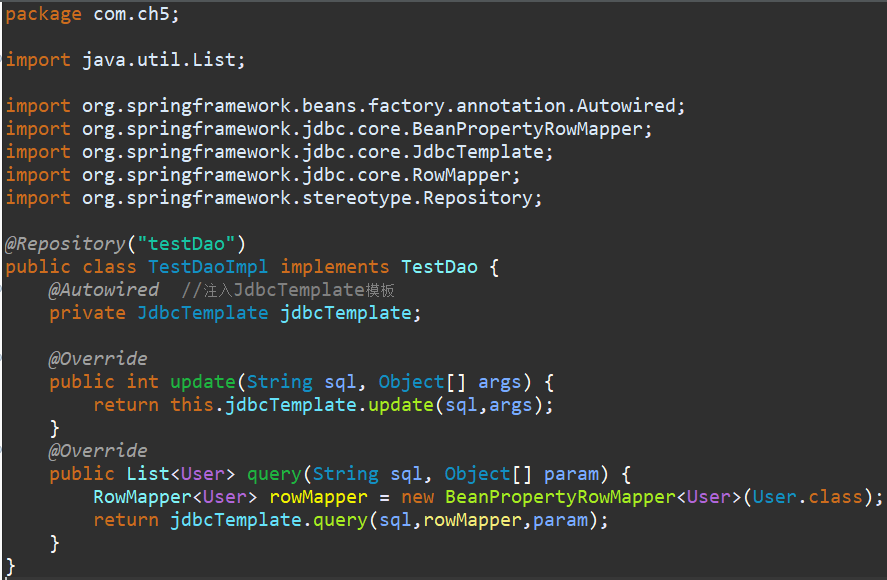
1. **创建数据访问层Dao**

在com.ch5包中创建TestDao接口和TestDaoImpl实现类。在实现类TestDaoImpl中使用JDBC模板JdbcTemplate访问数据库，并将该类标注为@Repository(“testDao”)，所以为了使标注的注解生效还需要在Spring配置文件中扫描包。

TestDao接口代码如下：



TestDaoImpl实现类代码如下：

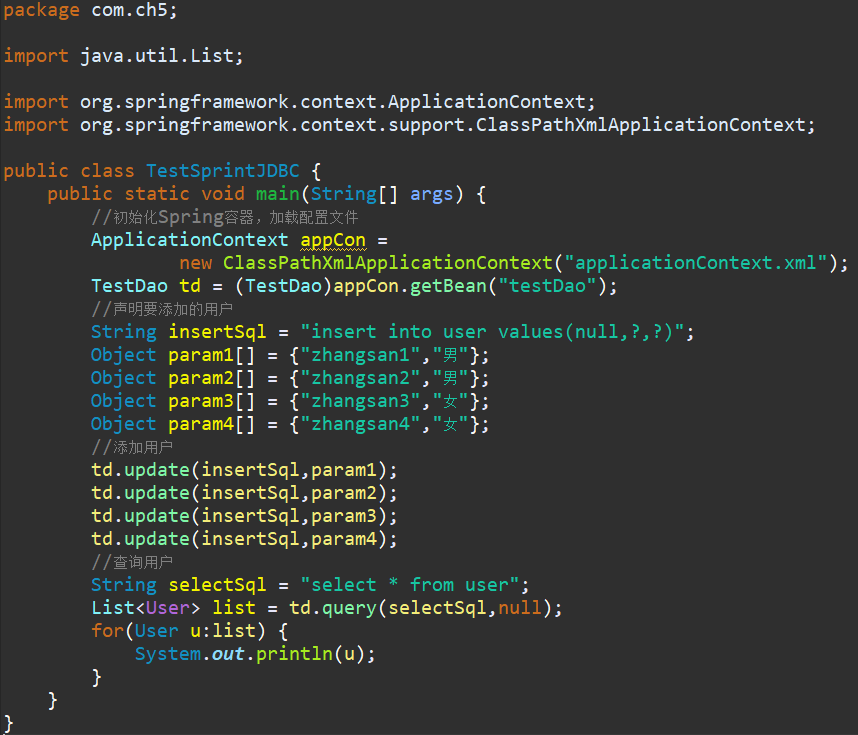


由于TestDaoImpl类使用了@Repository注解标注，所以为了使注解生效，需要在 Spring配置文件中进行扫描包操作。

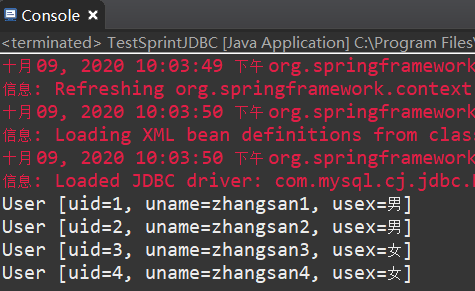


1. **创建测试类**

在com.ch5包中创建测试类TestSpringJDBC。在主方法中调用数据访问层Dao中的方法，对数据表user进行操作。具体代码如下：



运行结果如下，将添加的4位用户查询出来了：



# 编程式事务管理

1. 在代码中显示调用beginTransaction、commiet、rollback等与事务处理相关的方法，

这就是编程式事务管理。当只有少数事务操作时，编程式事务管理才比较合适。

1. 实现编程式事务管理有两种方式，一是基于底层API的编程式事务管理，二是基于TransactionTemplate的编程式事务管理。

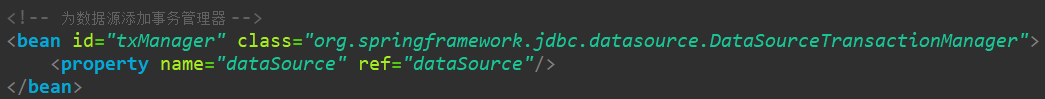
## 基于底层API实现

基于底层API的编程式事务管理就是根据PlatformTransactionManager、 TransactionDefinition和TransactionStatus几个核心接口，通过编程的方式来进行事务处

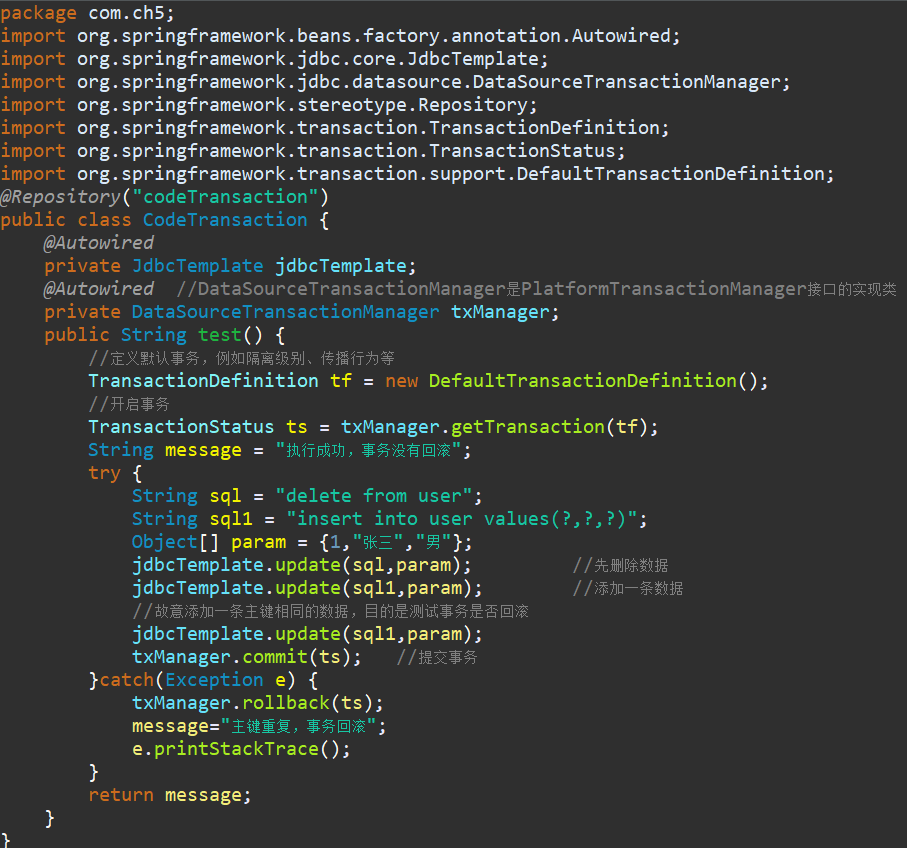
理。示例如下：

1. **给数据源添加事务管理器**

在上一节示例的基础上，在Spring配置文件的基础上使用PlatformTransactionManager接口的实现类org.springframework.jdbc.datasource.DataSourceTransactionManager为数据源添加事务管理器。具体代码如下：

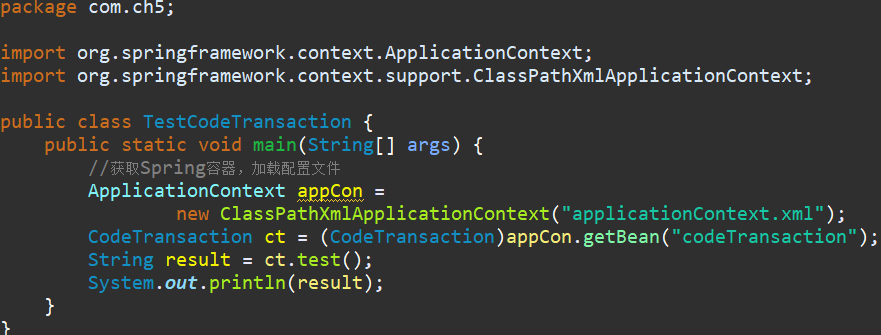


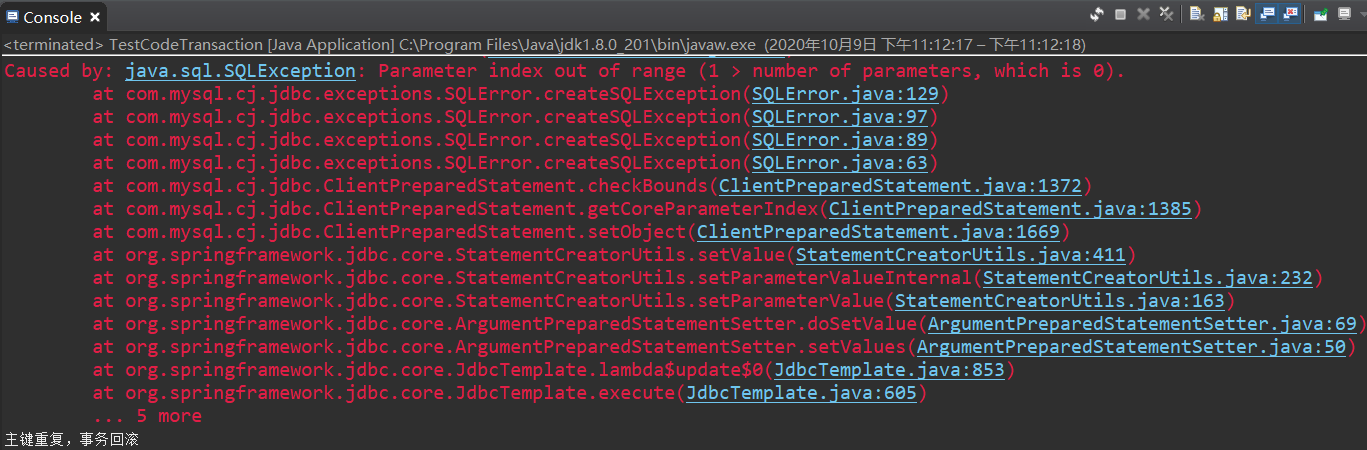
**2）创建数据访问层**



1. **创建测试类**

在com.ch5包中创建测试类TestCodeTransaction，具体代码如下：



 运行程序，结果如下：

可以看出，由于添加了一条相同主键的数据，所以造成事务回滚。

## 基于TransactionTemplate类实现

1. 在TransactionTemplate类中有一个execute()方法，该方法有且只有一个

TransactionCallback接口类型的参数，在TransactionCallback接口中定义了一个 doInTransaction( )方法。通常以匿名内部类的方式实现TransactionCallback接口，并在其 doInTransaction( )方法中书写业务逻辑代码，在这个方法里面可以使用默认的事务提交和回滚，在业务代码中不需要显示地调用任何处理事务的API。

2）在doInTransaction方法 中有一个TransactionStatus类型的参数，可以在方法的任何

位置调用该参数的setRollbackOnly方法将事务标识为回滚，以执行事务回滚。根据

默认规则，如果在执行回调方法的过程中抛出了未检查异常，或者显示调用了 setRollbackOnly方法，则回滚事务；如果事务执行完成或者是抛出了checked类型的

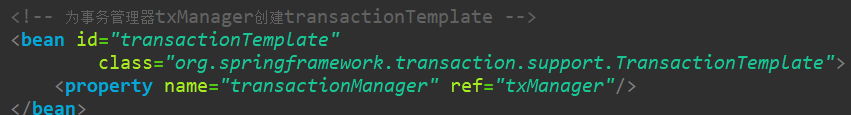
异常，则提交事务。

3）使用基于TransactionTemplate类实现事务管理需要为事务管理器添加事务模板。

通过一个示例演示如何使用基于TransactionTemplate类的编程式事务管理。

1. **为事务管理器添加事务模板**

在上一节中applicationContext.xml配置文件的基础上使用org.springframeworkwork.transaction.support.TransactionTemplate类为事务管理器添加事务模板。具体代码如下：

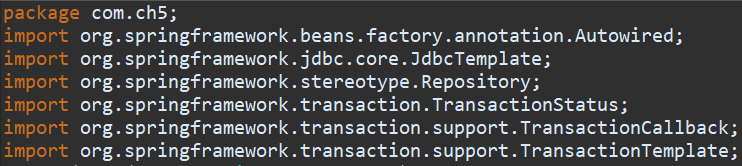


1. **创建数据访问层**

在com.ch5包中创建数据访问类TransactionTemplateDao，并注解为@Repository(“transactionTemplateDao”)。在该类中使用编程的方式，基于TransactionTemplate的方式完成事务管理。

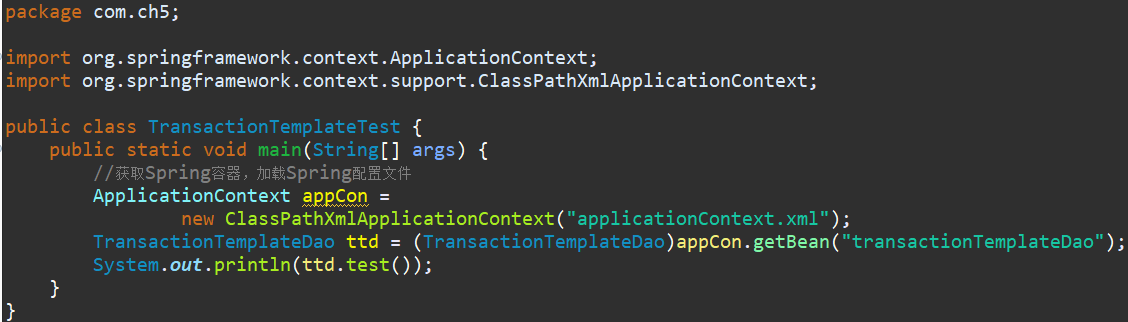


由于截图原因，所以未截出导入包类的语句。其导包语句如下：

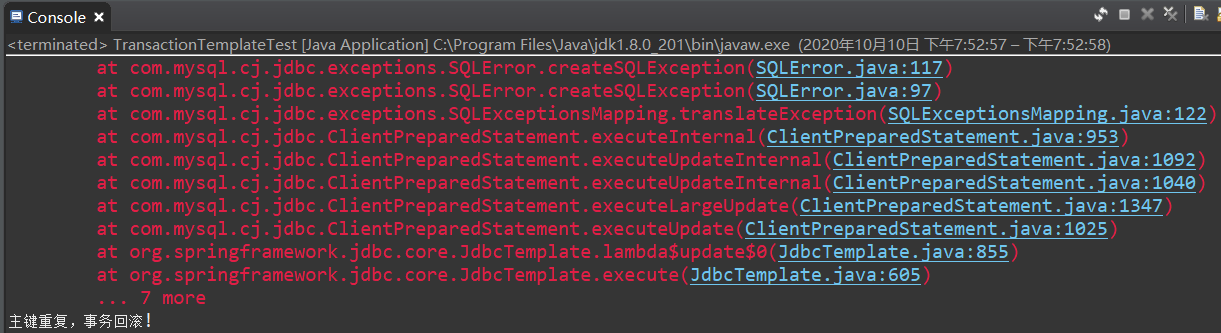


1. **创建测试类**

在com.ch5包中创建测试类TransactionTemplateTest，代码如下：



运行程序，结果如下：



可以发现事务回滚了。

# 声明式事务管理

1. Spring的声明式事务管理是指通过AOP技术实现事务管理，其本质是对方法前后进 行拦截，在目标方法执行开始之前创建或者加入一个事务，在执行完目标方法之后 根据其执行情况提交或者回滚事务。
2. 声明式事务管理最大的优点是不需要通过编程的方法管理事务，因而不需要在业务 逻辑代码中掺杂任何事务处理的代码，只需将相关的事务规则声明便可以将事务规 则应用到业务逻辑中。通常情况下，在开发中使用声明式事务管理。因为这样不仅 简单，还能使得纯业务代码不被污染，方便后期的代码维护。
3. 与编程式事务管理相比，声明式事务管理唯一不足的地方是最细粒度只能作用到方 法级别，无法做到像编程式事务管理那样可以作用到代码块级别。
4. Spring的声明式事务管理可以通过两种方法来实现，一是基于XML的方式，二是基 于@Transactional注解的方式。

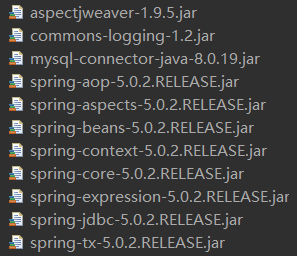
## 基于XML方式实现

基于XML方式的声明式事务管理是通过在配置文件中配置事务规则的相关声明来实现的。Spring框架提供了tx命名空间来配置事务，提供了<tx:advice>元素来配置事务的通知。在配置<tx:advice>元素时一般需要指定id和transaction-manager属性，其中id属性是配置文件中的唯一标识，transaction-manager属性指定事务管理器。另外还需要<tx:attribute>子标签，该子标签可以配置多个<tx:method>子元素指定执行事务的细节。在<tx:advice>元素配置了事务的增强处理后就可以通过编写AOP配置让Spring自动对目标对象生成代理。

通过一个示例来演示如何通过XML方式来实现Spring的声明式事务管理，同时，为了体现事务管理的流程，本实例创建Dao，Service和Controller3层。

1. **导入相关的JAR包**

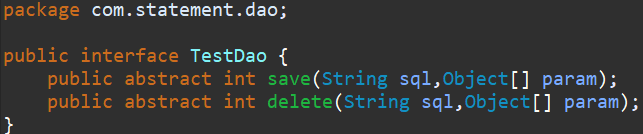
在先前ch5应用的基础上导入AOP所需要的JAR包，导入后lib目录如图所示：



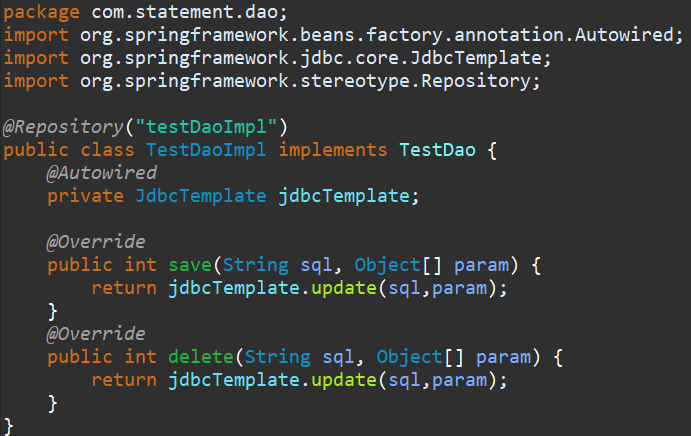
1. **创建Dao层**

在ch5的src目录下创建com.statement.dao包，并在该包中创建TestDao接口和TestDaoImpl实现类。数据访问层有两个数据操作方法，即save和delete方法。

TestDao接口代码如下：

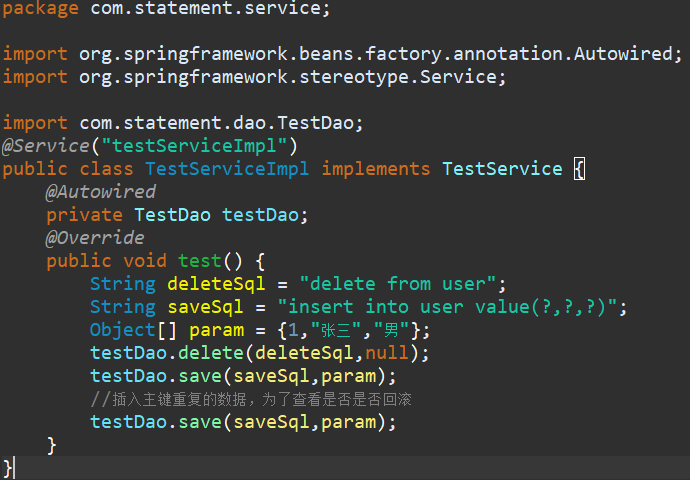


TestDaoImpl实现类代码如下：



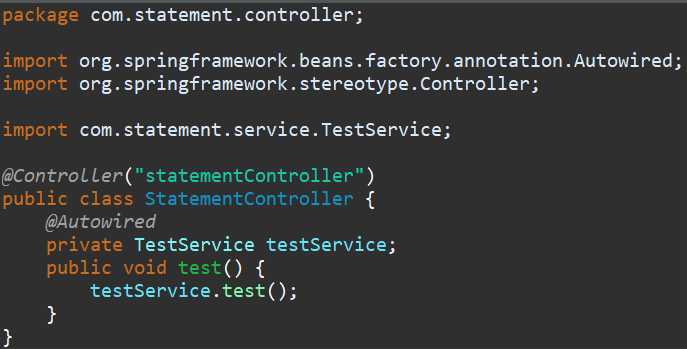
1. **创建Service层**

在ch5的src目录下创建com.statement.service包，并在该包中创建TestService接口和TestServiceImpl实现类。在Service层依赖注入数据访问层。



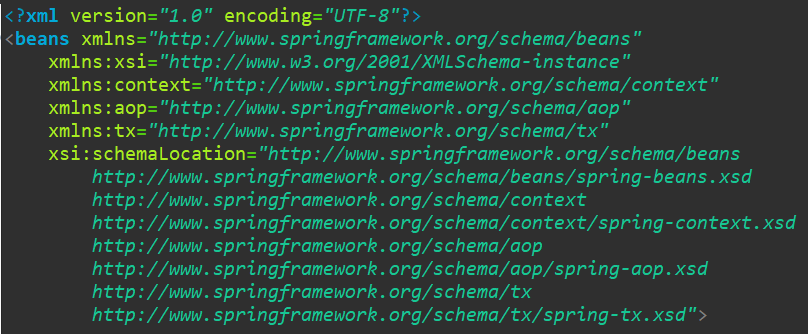
1. **创建Controller层**

在ch5的src目录下创建com.statement.controller包，并在该包中创建StatementController控制器类。在控制层依赖注入Service层。代码如下：

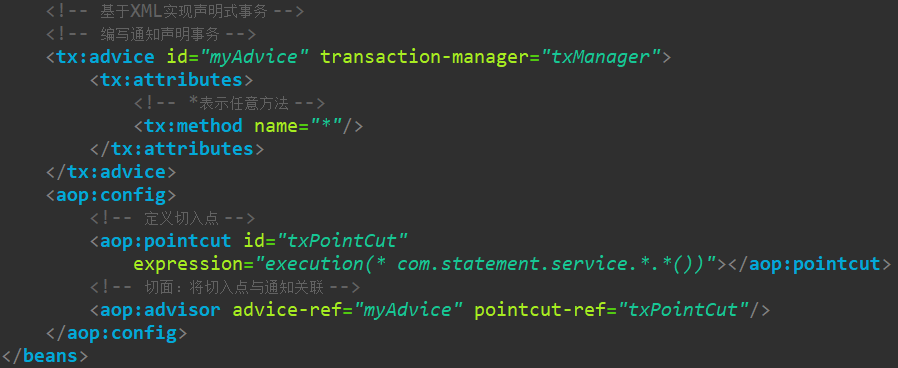


1. **创建配置文件**

在src目录下创建Spring配置文件，代码如下：

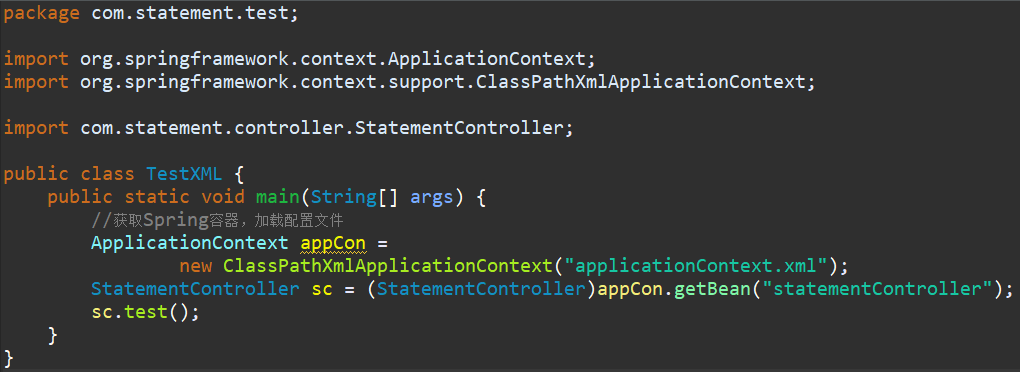






1. **创建测试类**

在src目录下创建com.statement.test包，并在该包中创建测试类XMLTest，在测试类中通过访问Controller测试基于XML方式的声明式事务管理。代码如下：



运行程序，查看运行结果或核对数据表中的数据，验证事务是否回滚。

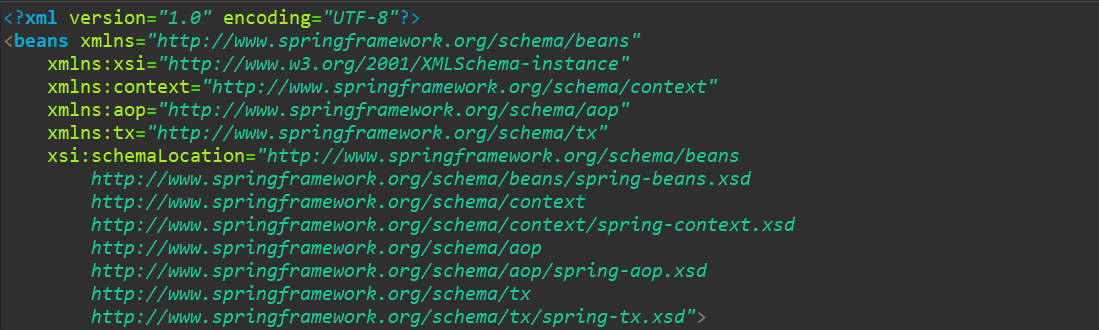
## 基于@Transactional注解实现

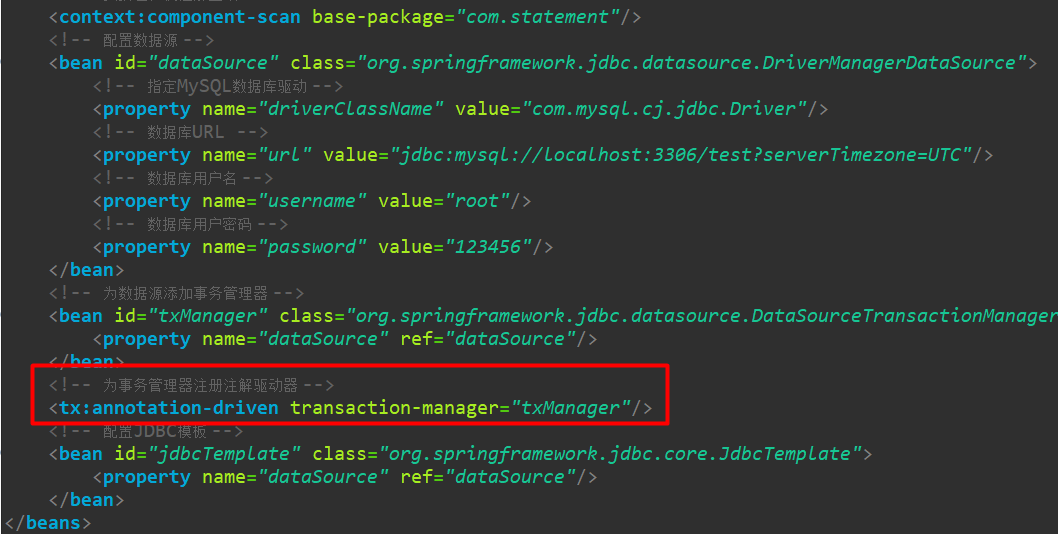
@Transactional注解可以可用于接口、接口方法、类以及类的方法上。当作用于类上时，该类的所有public方法都将具有该类型的事务属性，同时也可以在方法级别使 用该注解来覆盖类级别的定义。虽然@Transactional注解可以标注在接口、接口方法、类以及类的方法上，但是Spring小组建议不要在接口或者接口方法上使用该注解，因为它只有在使用基于接口的代理时才会生效。

基于@Transaction注解实现事务管理的步骤使用简单，只需两步：

1. **注册事务注解驱动器**

在Spring配置文件中使用<tx:annotation-driven>元素为事务管理器注册注解驱动器。代码如下：





红色区括出的代码表示为事务管理器注册注解驱动器。

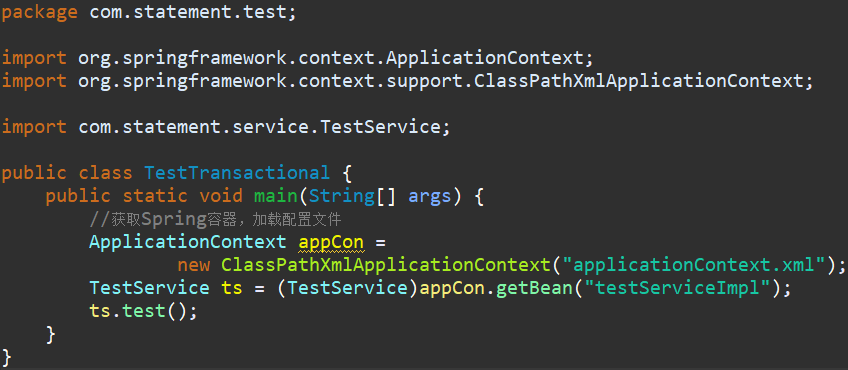
**2）为Service层标注@Transaction注解**

在Spring MVC中通常通过Service层进行事务管理，因此需要为Service层添加@Transactional注解。添加@@Transactional注解后的TestServiceImpl类的代码如下：



1. **编写测试类**

编写测试类，测试基于@Transaction注解实现事务管理是否有效。在com.statement.test包中创建TestTransactional类，代码如下：



运行程序，查看数据库或观察运行结果验证事务是否回滚。

## 在声明式事务中捕获异常

在开发中，一般都是使用声明式事务。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。

但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？答案是不能的。例如，上一小节的TestServiceImpl 实现类的test方法的代码修改如下：

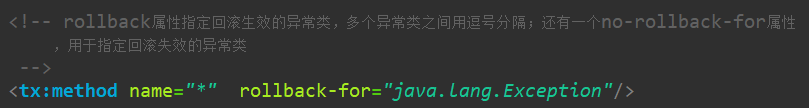


这时，在运行测试类，发现注解重复但事务并没有回滚（可以查看数据库的数据来验证）。 这是因为默认情况下，Spring只在发送未被捕获的RuntimeException时才回滚事务。如 果在事务处理中要获取异常，方法如下：

**1、基于XML方式的声明式事务管理中捕获异常**

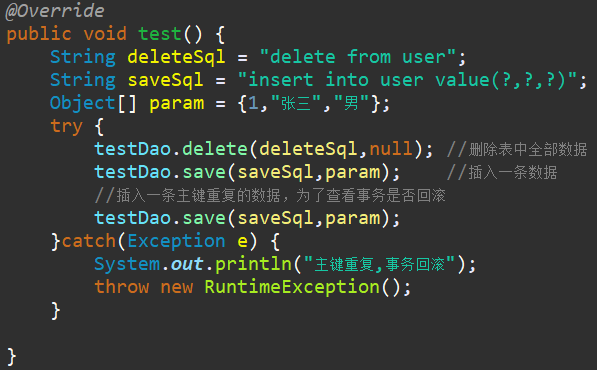
在基于XML方式的声明式事务管理中捕获异常，需要补充两个步骤。

1）在Spring配置文件中为受事务管理的方法指定回滚生效的异常类。如下：



rollback属性指定回滚生效的异常类，多个异常类之间用逗号分隔；还有一个 no-rollback-for属性，用于指定回滚失效的异常类。

2）在catch语句中抛出一个RuntimeException类型的异常。代码如下：



**2、基于@Transactional注解的声明式事务管理中捕获异常**

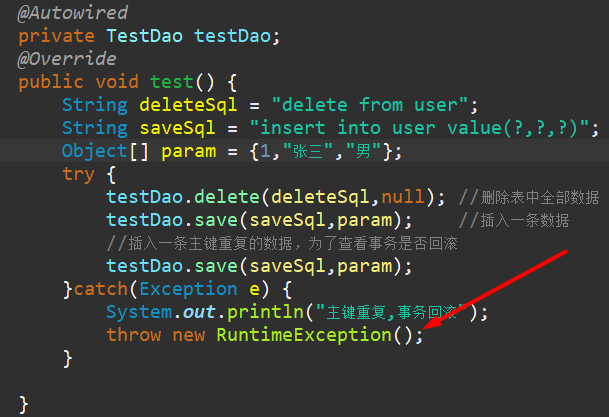
在基于@Transactional注解的声明式事务管理中捕获异常，也需要补充两个步骤。

1. 修改@Transactional注解

将标注在业务层上的@Transactional注解修改为：



1. 在catch语句中抛出一个new RuntimeException类型的异常。如下：



**3、在声明式事务中捕获异常（通用方式）**

在声明式事务中获取异常，不管是基于XML还是基于@Transactional注解，都是通

过rollbackFor属性指定回滚生效的的异常类，然后在catch语句中手动抛出一个 RuntimeException类型的异常。但在实际开发中，不管你是通过XML方式还是基于 @Transactional注解的方式实现声明式事务管理。可以直接在catch语句中调用事务 回滚方法，达到发送异常后将事务进行回滚的目的。如下：

