第 5 章 Spring 的事务管理

学习目的与要求

本章主要介绍了 Spring 框架所支持的事务管理，包括编程式事务管理和声明式事务管理。通过本章的学习，要求读者掌握声明式事务管理，了解编程式事务管理。

本章主要内容

- Spring 的数据库编程
- 编程式事务管理
- 声明式事务管理

在数据库操作中，事务管理是一个重要的概念。例如，银行转账，当从 A 账户向 B 账户转 1000 元后，银行的系统将从 A 账户上扣除 1000 元，而在 B 账户上加 1000 元，这是正常处理的结果。

一旦银行系统出错了怎么办？这里假设发生两种情况：

（1）A 账户少了 1000 元，但 B 账户却没有多 1000 元。

（2）B 账户多了 1000 元钱，但 A 账户却没有被扣钱。

客户和银行都不愿意发生上面两种情况。那么有没有措施保证转账顺利进行？这种措施就是数据库事务管理机制。

Spring 的事务管理简化了传统的数据库事务管理流程，提高了开发效率。在学习事务管理之前，需要了解 Spring 的数据库编程。

扫一扫

视频讲解

5.1 Spring 的数据库编程

数据库编程是互联网编程的基础，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，在工作中更多的是使用 Hibernate 框架和 MyBatis 框架进行数据库编程。

本节简要介绍 Spring jdbcTemplate 的使用方法，对于 MyBatis 框架的相关内容将在第 14 章详细介绍，对于 Hibernate 框架本书不再涉及，需要的读者可以查阅 Hibernate 框架的相关知识。

▶ 5.1.1 Spring JDBC 的配置

本节 Spring 的数据库编程主要使用 Spring JDBC 模块的 core 包和 dataSource 包。core 包是 JDBC 的核心功能包，包括常用的 JdbcTemplate 类；dataSource 包是访问数据源的工具类包。使用 Spring JDBC 操作数据库，需要对其进行配置。配置文件的示例代码如下：

```
<!-- 配置数据源 -->
```

```
<bean id="dataSource" class="org.springframework.jdbc.datasource.
    DriverManagerDataSource">
        <!-- MySQL 数据库驱动 -->
        <property name="driverClassName" value="com.mysql.cj.jdbc.
            Driver"/>
        <!-- 连接数据库的 URL -->
        <property name="url" value="jdbc:mysql://127.0.0.1:3306/
            springtest?useUnicode=
true&amp;characterEncoding=UTF-8&amp;allowMultiQueries=true&amp;
    serverTimezone=GMT%2B8"/>
        <!-- 连接数据库的用户名 -->
        <property name="username" value="root"/>
        <!-- 连接数据库的密码 -->
        <property name="password" value="root"/>
    </bean>
    <!-- 配置 JDBC 模板 -->
    <bean id="jdbcTemplate" class="org.springframework.jdbc.core.
        JdbcTemplate">
            <property name="dataSource" ref="dataSource"/>
    </bean>
```

在上述示例代码中，配置 JDBC 模板时，需要将 dataSource 注入 jdbcTemplate，而在数据访问层（Dao 层）需要使用 jdbcTemplate 时，也需要将 jdbcTemplate 注入对应的 Bean 中。示例代码如下：

```
…
@Repository("testDao")
public class TestDaoImpl implements TestDao{
     @Autowired
    // 使用配置文件中的 JDBC 模板
    private JdbcTemplate jdbcTemplate;
    …
}
```

▶ 5.1.2 Spring jdbcTemplate 的使用方法

在获取 JDBC 模板后如何使用它是本节将要讲述的内容。首先需要了解 jdbcTemplate 类的常用方法，该类的常用方法是 update() 和 query() 方法。

（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：

```
String insertSql = "insert into user values(null,?,?)";
Object param1[] = {"chenheng1", "男"};
jdbcTemplate.update(sql, param1);
```

（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 将结果集映射到用户自定义的类中（前提是自定义类中的属性与数据表的字段对应）。示例代码如下：

```
String selectSql ="select * from user";
RowMapper<MyUser> rowMapper = new BeanPropertyRowMapper<MyUser>(MyUser.
    class);
List<MyUser> list = jdbcTemplate.query(sql, rowMapper, null);
```

下面通过一个实例演示 Spring JDBC 的使用过程。

【例 5-1】 使用 Spring JDBC 访问数据库。

❶ 创建模块并导入 JAR 包

创建名为 ch5 的项目，然后在 ch5 项目中创建一个名为 ch5_1 的模块，同时给 ch5_1 模块添加 Web Application，并将 Spring 的 4 个基础包、Spring Commons Logging Bridge 对应的 JAR 包 spring-jcl-6.0.0.jar 和 spring-aop-6.0.0.jar、MySQL 数据库的驱动程序 JAR 包（mysql-connector-java-8.0.29.jar）、Spring JDBC 的 JAR 包（spring-jdbc-6.0.0.jar）、Java 增强库（lombok-1.18.24.jar）以及 Spring 事务管理的 JAR 包（spring-tx-6.0.0.jar）复制到 ch5_1 的 WEB-INF/lib 目录中，添加为模块依赖，如图 5.1 所示。

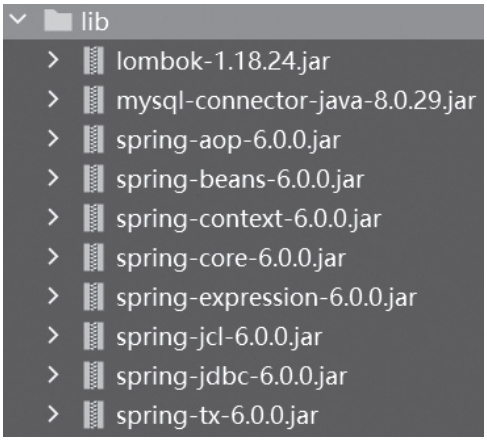

图 5.1　ch5_1 模块所依赖的 JAR 包

❷ 创建并编辑配置文件

在 ch5_1 模块的 src 目录中创建配置文件 applicationContext.xml，并在该文件中配置数据源和 JDBC 模板，具体代码如下：

```
<?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.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">
    <!-- 指定需要扫描的包（包括子包），使注解生效 -->
    <context:component-scan base-package="dao"/>
    <!-- 配置数据源 -->
    <bean id="dataSource" class="org.springframework.jdbc.datasource.
        DriverManagerDataSource">
        <!-- MySQL 数据库驱动 -->
        <property name="driverClassName" value="com.mysql.cj.jdbc.
            Driver"/>
        <!-- 连接数据库的 URL -->
        <property name="url" value="jdbc:mysql://127.0.0.1:3306/springtest?
            useUnicode=true&amp;characterEncoding=UTF-8&amp;
            allowMultiQueries=true&amp;serverTimezone=GMT%2B8"/>
        <!-- 连接数据库的用户名 -->
        <property name="username" value="root"/>
        <!-- 连接数据库的密码 -->
        <property name="password" value="root"/>
    </bean>
    <!-- 配置 JDBC 模板 -->
    <bean id="jdbcTemplate" class="org.springframework.jdbc.core.
```

```
        JdbcTemplate">
        <property name="dataSource" ref="dataSource"/>
    </bean>
</beans>
```

❸ 创建实体类

在 ch5_1 模块的 src 目录中创建名为 entity 的包，并在该包中创建实体类 MyUser。该类的属性与数据表 user 中的字段一致。数据表 user 的结构如图 5.2 所示。

名	类型	长度	小数点	允许空值 (	
▶ uid	int	10	0	☐	🔑1
uname	varchar	20	0	☑	
usex	varchar	10	0	☑	

图 5.2　数据表 user 的结构

实体类 MyUser 的代码如下：

```
package entity;
import lombok.Data;
/**
* 使用 @Data 注解实体属性无须 Setter 和 Getter 方法，但需要给 IDEA 安装 Lombok 插件（安
* 装后重构模块）
*/
@Data
public class MyUser {
    private Integer uid;
    private String uname;
    private String usex;
    public String toString() {
        return "myUser [uid=" + uid +", uname=" + uname + ", usex=" +
            usex + "]";
    }
}
```

❹ 创建数据访问层 Dao

在 ch5_1 模块的 src 目录中创建名为 dao 的包，并在 dao 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，并将该类注解为 @Repository("testDao")。

TestDao 接口的代码如下：

```
package dao;
import java.util.List;
import entity.MyUser;
public interface TestDao {
    public int update(String sql, Object[] param);
    public List<MyUser> query(String sql, Object[] param);
}
```

TestDaoImpl 实现类的代码如下：

```
package dao;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.BeanPropertyRowMapper;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.stereotype.Repository;
import entity.MyUser;
@Repository("testDao")
```

```
public class TestDaoImpl implements TestDao{
     @Autowired
    // 使用配置文件中的 JDBC 模板
    private JdbcTemplate jdbcTemplate;
    /**
     * 更新方法，包括添加、修改、删除
     * param 为 sql 中的参数，例如通配符 ?
     */
    @Override
    public int update(String sql, Object[] param) {
        return jdbcTemplate.update(sql, param);
    }
    /**
     * 查询方法
     * param 为 sql 中的参数，例如通配符 ?
     */
    @Override
    public List<MyUser> query(String sql, Object[] param) {
        RowMapper<MyUser> rowMapper = new BeanPropertyRowMapper
            <MyUser>(MyUser.class);
        return jdbcTemplate.query(sql, rowMapper, param);
    }
}
```

❺ 创建测试类

在 ch5_1 模块的 src 目录中创建名为 test 的包，并在该包中创建测试类 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：

```
package test;
import java.util.List;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import dao.TestDao;
import entity.MyUser;
public class TestSpringJDBC {
    private static ApplicationContext appCon;
    public static void main(String[] args) {
        appCon = new ClassPathXmlApplicationContext
            ("applicationContext.xml");
        TestDao td = (TestDao)appCon.getBean("testDao");
        String insertSql = "insert into user values(null,?,?)";
        // 数组 param 的值与 insertSql 语句中的 ? 一一对应
        Object param1[] = {"chenheng1", "男"};
        Object param2[] = {"chenheng2", "女"};
        Object param3[] = {"chenheng3", "男"};
        Object param4[] = {"chenheng4", "女"};
        // 添加用户
        td.update(insertSql, param1);
        td.update(insertSql, param2);
        td.update(insertSql, param3);
        td.update(insertSql, param4);
        // 查询用户
        String selectSql ="select * from user";
        List<MyUser> list = td.query(selectSql, null);
        for(MyUser mu: list) {
            System.out.println(mu);
        }
    }
}
```

运行上述测试类，运行结果如图 5.3 所示。

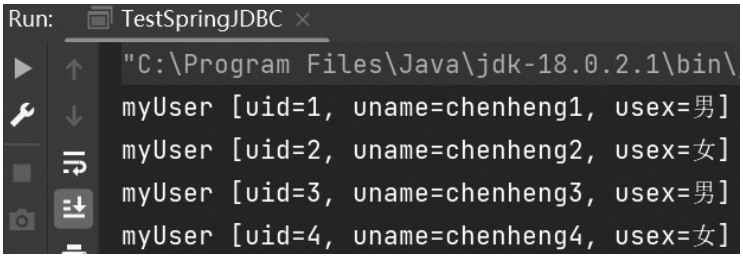

图 5.3　Spring 数据库编程的运行结果

扫一扫

视频讲解

5.2　编程式事务管理

在代码中显式调用 beginTransaction()、commit()、rollback() 等与事务管理相关的方法，这就是编程式事务管理。当只有少数事务操作时，采用编程式事务管理比较合适。

▶ 5.2.1　基于底层 API 的编程式事务管理

基于底层 API 的编程式事务管理就是根据 PlatformTransactionManager、TransactionDefinition 和 TransactionStatus 3 个核心接口通过编程的方式来进行事务管理。

下面通过一个实例讲解基于底层 API 的编程式事务管理。

【例 5-2】 基于底层 API 的编程式事务管理。

❶ 创建模块并导入 JAR 包

在 ch5 项目中创建一个名为 ch5_2 的模块，同时给 ch5_2 模块添加 Web Application，并将 Spring 的 4 个基础包、Spring Commons Logging Bridge 对应的 JAR 包 spring-jcl-6.0.0.jar 和 spring-aop-6.0.0.jar、MySQL 数据库的驱动程序 JAR 包（mysql-connector-java-8.0.29.jar）、Spring JDBC 的 JAR 包（spring-jdbc-6.0.0.jar）以及 Spring 事务管理的 JAR 包（spring-tx-6.0.0.jar）复制到 ch5_2 的 WEB-INF/lib 目录中，添加为模块依赖。

❷ 给数据源配置事务管理器

在 ch5_2 模块的 src 目录中创建配置文件 applicationContext.xml，并在配置文件中使用 PlatformTransactionManager 接口的一个间接实现类 DataSourceTransactionManager 为数据源添加事务管理器，具体配置代码如下：

```
<?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.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">
    <!-- 指定需要扫描的包（包括子包），使注解生效 -->
    <context:component-scan base-package="dao"/>
    <!-- 配置数据源 -->
    <bean id="dataSource" class="org.springframework.jdbc.datasource.
        DriverManagerDataSource">
        <!-- MySQL 数据库驱动 -->
        <property name="driverClassName" value="com.mysql.cj.jdbc.
            Driver"/>
        <!-- 连接数据库的 URL -->
```

```
            <property name="url" value="jdbc:mysql://127.0.0.1:3306/
                springtest?useUnicode=true&amp;characterEncoding=UTF-8&amp;
                allowMultiQueries=true&amp;serverTimezone=GMT%2B8"/>
            <!-- 连接数据库的用户名 -->
            <property name="username" value="root"/>
            <!-- 连接数据库的密码 -->
            <property name="password" value="root"/>
        </bean>
        <!-- 配置 JDBC 模板 -->
        <bean id="jdbcTemplate" class="org.springframework.jdbc.core.
            JdbcTemplate">
            <property name="dataSource" ref="dataSource"/>
        </bean>
        <!-- 为数据源添加事务管理器 -->
        <bean id="txManager"
            class="org.springframework.jdbc.datasource.
                DataSourceTransactionManager">
            <property name="dataSource" ref="dataSource"/>
        </bean>
    </beans>
```

❸ 创建数据访问类

在 ch5_2 模块的 src 目录中创建名为 dao 的包，在该包中创建数据访问类 CodeTransaction，并使用 @Repository("codeTransaction") 注解为数据访问层。在 CodeTransaction 类中使用编程的方式进行数据库的事务管理。

CodeTransaction 类的代码如下：

```
package dao;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.datasource.DataSourceTransactionManager;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.TransactionDefinition;
import org.springframework.transaction.TransactionStatus;
import org.springframework.transaction.support.DefaultTransactionDefinition;
@Repository("codeTransaction")
public class CodeTransaction {
    @Autowired
    // 使用配置文件中的 JDBC 模板
    private JdbcTemplate jdbcTemplate;
    // 依赖注入事务管理器 txManager（配置文件中的 Bean）
    @Autowired
    private DataSourceTransactionManager txManager;
    public void test() {
        // 默认事务定义，例如隔离级别、传播行为等
        TransactionDefinition tf = new DefaultTransactionDefinition();
        // 开启事务 ts
        TransactionStatus ts = txManager.getTransaction(tf);
        // 删除表中的数据
        String sql = " delete from user ";
        // 添加数据
        String sql1 = " insert into user values(?,?,?) ";
        Object param[] = { 1, " 陈恒 ", " 男 " };
        try {
            // 删除数据
            jdbcTemplate.update(sql);
            // 添加一条数据
            jdbcTemplate.update(sql1, param);
            // 添加相同的一条数据，使主键重复
            jdbcTemplate.update(sql1, param);
```

```
                // 提交事务
                txManager.commit(ts);
                System.out.println("执行成功，没有事务回滚！");
            } catch (Exception e) {
                System.out.println("主键重复，事务回滚！");
            }
        }
    }
```

❹ 创建测试类

在 ch5_2 模块的 src 目录中创建名为 test 的包，并在该包中创建测试类 TestCodeTransaction，具体代码如下：

```
package test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import dao.CodeTransaction;
public class TestCodeTransaction {
    private static ApplicationContext appCon;
    public static void main(String[] args) {
        appCon = new ClassPathXmlApplicationContext
            ("applicationContext.xml");
        CodeTransaction ct = (CodeTransaction)appCon.getBean
            ("codeTransaction");
        ct.test();
    }
}
```

上述测试类的运行结果如图 5.4 所示。

图 5.4　基于底层 API 的编程式事务管理的测试结果

从程序运行前后数据表中的数据可以看出，取消了主键重复前执行的删除和插入操作，即事务回滚。

▶ 5.2.2　基于 TransactionTemplate 的编程式事务管理

事务管理的代码散落在业务逻辑代码中，破坏了原有代码的条理性，并且每一个业务方法都包含了类似的启动事务、提交以及回滚事务的样板代码。

TransactionTemplate 的 execute() 方法有一个 TransactionCallback 接口类型的参数，该接口中定义了一个 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。这里可以使用默认的事务提交和回滚规则，在业务代码中不需要显式调用任何事务管理的 API。doInTransaction() 方法有一个 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，或者显式调用了 setRollbackOnly() 方法，则回滚事务；如果事务执行完成或者抛出了 checked 类型的异常，则提交事务。

【例 5-3】 基于 TransactionTemplate 的编程式事务管理。

❶ 创建模块并导入 JAR 包

在 ch5 项目中创建一个名为 ch5_3 的模块，同时给 ch5_3 模块添加 Web Application，并将 ch5_2 模块的 lib 复制到 ch5_3 模块的 WEB-INF 目录中，添加为模块依赖。

❷ 为事务管理器添加事务模板

在 ch5_3 模块的 src 目录中创建配置文件 applicationContext.xml，在配置文件中使用 org.springframework.transaction.support.TransactionTemplate 类为事务管理器添加事务模板，具体配置代码如下：

```
<?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.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">
    <!-- 指定需要扫描的包（包括子包），使注解生效 -->
    <context:component-scan base-package="dao"/>
    <!-- 配置数据源 -->
    <bean id="dataSource" class="org.springframework.jdbc.datasource.
        DriverManagerDataSource">
        <!-- MySQL 数据库驱动 -->
        <property name="driverClassName" value="com.mysql.cj.jdbc.
            Driver"/>
        <!-- 连接数据库的 URL -->
        <property name="url" value="jdbc:mysql://127.0.0.1:3306/
            springtest?useUnicode=true&amp;characterEncoding=UTF-8&amp;
            allowMultiQueries=true&amp;serverTimezone=GMT%2B8"/>
        <!-- 连接数据库的用户名 -->
        <property name="username" value="root"/>
        <!-- 连接数据库的密码 -->
        <property name="password" value="root"/>
    </bean>
    <!-- 配置 JDBC 模板 -->
    <bean id="jdbcTemplate" class="org.springframework.jdbc.core.
        JdbcTemplate">
        <property name="dataSource" ref="dataSource"/>
    </bean>
    <!-- 为数据源添加事务管理器 -->
        <bean id="txManager"
            class="org.springframework.jdbc.datasource.
                DataSourceTransactionManager">
            <property name="dataSource" ref="dataSource"/>
        </bean>
        <!-- 为事务管理器 txManager 创建 transactionTemplate -->
        <bean id="transactionTemplate" class="org.springframework.
            transaction.support.TransactionTemplate">
        <property name="transactionManager" ref="txManager"/>
    </bean>
</beans>
```

❸ 创建数据访问类

在 ch5_3 模块的 src 目录中创建名为 dao 的包，并在该包中创建数据访问类 TransactionTemplateDao，同时注解为 @Repository("transactionTemplateDao")。在 TransactionTemplateDao 类中使用编程的方式进行数据库的事务管理。

数据访问类 TransactionTemplateDao 的代码具体如下：

```
package dao;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.TransactionStatus;
import org.springframework.transaction.support.TransactionCallback;
import org.springframework.transaction.support.TransactionTemplate;
@Repository("transactionTemplateDao")
public class TransactionTemplateDao {
    // 依赖注入 JDBC 模板
    @Autowired
    private JdbcTemplate jdbcTemplate;
    // 依赖注入 transactionTemplate
    @Autowired
    private TransactionTemplate transactionTemplate;
    public void test() {
        // 以匿名内部类的方式实现 TransactionCallback 接口，使用默认的事务提交和回滚
        // 规则，在业务代码中不需要显式调用任何事务处理的 API
        transactionTemplate.execute(new TransactionCallback<Object>(){
            @Override
            public Object doInTransaction(TransactionStatus arg0) {
                // 删除表中的数据
                String sql = " delete from user  ";
                // 添加数据
                String sql1 = " insert into user values(?,?,?) ";
                Object param[] = {1," 陈恒 "," 男 "};
                try{
                    // 删除数据
                    jdbcTemplate.update(sql);
                    // 添加一条数据
                    jdbcTemplate.update(sql1, param);
                    // 添加相同的一条数据，使主键重复
                    jdbcTemplate.update(sql1, param);
                    System.out.println(" 执行成功，没有事务回滚！ ");
                }catch(Exception e){
                    System.out.println(" 主键重复，事务回滚！ ");
                }
                return null;
            }
        });
    }
}
```

❹ 创建测试类

在 ch5_3 模块的 src 目录中创建名为 test 的包，并在该包中创建测试类 TransactionTemplateTest，测试类的代码具体如下：

```
package test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import dao.TransactionTemplateDao;
public class TransactionTemplateTest {
    private static ApplicationContext appCon;
    public static void main(String[] args) {
        appCon = new ClassPathXmlApplicationContext
            ("applicationContext.xml");
        TransactionTemplateDao ct = (TransactionTemplateDao)appCon.
            getBean("transactionTemplateDao");
        ct.test();
    }
}
```

扫一扫

视频讲解

5.3　声明式事务管理

Spring 的声明式事务管理是通过 AOP 技术实现的事务管理，其本质是对方法执行前后进行拦截，在目标方法执行前，创建或者加入一个事务；在目标方法执行后，根据执行情况提交事务或回滚事务。

声明式事务管理最大的优点是不需要通过编程的方式管理事务，因此不需要在业务逻辑代码中掺杂事务管理的代码，只需要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，不仅因为其简单，更因为这样使纯业务代码不被污染，极大地方便了后期的代码维护。

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，无法像编程式事务管理那样可以作用到代码块级别。如果的确有需求，也可以通过变通的方法进行解决，比如将需要进行事务管理的代码块独立为方法。

Spring 的声明式事务管理可以通过两种方式来实现：一种是基于 XML 的方式；另一种是基于 @Transactional 注解的方式。

▶ 5.3.1　基于 XML 方式的声明式事务管理

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，用 <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 属性用于指定事务管理器。另外还需要 <tx:attributes> 子元素，该子元素可配置多个 <tx:method> 子元素指定执行事务的细节。

当 <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。下面通过一个实例演示如何通过 XML 方式来实现 Spring 的声明式事务管理。

【例 5-4】 基于 XML 方式的声明式事务管理。

为了体现事务管理的流程，本实例创建了 Dao、Service 和 Controller 3 层，具体实现步骤如下。

❶ 创建模块并导入 JAR 包

在 ch5 项目中创建一个名为 ch5_4 的模块，同时给 ch5_4 模块添加 Web Application，并将如图 5.5 所示的 JAR 包复制到 ch5_4 的 WEB-INF/lib 目录中，添加为模块依赖。

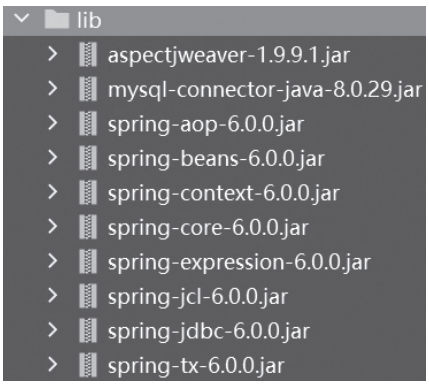

图 5.5　ch5_4 模块所依赖的 JAR 包