# Spring学习大纲

第一部分： spring框架的概述以及spring中基于XML的IOC（反转控制）配置

第二部分：Spring中基于注解的IOC和IOC案例

第三部分：spring中的AOP和基于XML以及注解的AOP（面向切面编程）配置

第四部分：spring中的JDBC Temlate以及Spring事务控制

### Spring学习第四天——Spring JDBCTemplate&事务控制

在这一部分学习内容中，主要的内容是学会使用Spring的JDBC Template进行数据库操作，以及Spring的事务控制对数据库的操作进行管理和维护。

**1、Spring JDBC Template**

* 1. **JDBC Template是什么**

JdbcTemplate是Spring框架中提供的一个对象,对原始的JDBC API进行简单封装,其用法与DBUtils类似。JdbcTemplate是Spring对JDBC的封装，目的是使JDBC更加易于使用。JdbcTemplate是Spring的一部分。JdbcTemplate处理了资源的建立和释放。他帮助我们避免一些常见的错误，比如忘了总要关闭连接。他运行核心的JDBC工作流，如Statement的建立和执行，而我们只需要提供SQL语句和提取结果。

* 1. **JDBCTemplate创建**

JdbcTemplate对象在执行sql语句时也需要一个数据源，可以选择三种数据源：C3P0或者DBCP,也可以使用Spring的内置数据源DriverManagerDataSource。

在bean.xml中配置三种数据源即可。

接着就可以创建JdbcTemplate对象，向JdbcTemplate中注入数据源创建对象,在bean.xml中配置如下:

<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">

<property name="dataSource" ref="dataSource"></property>

</bean>

# 1.3JdbcTemplate的增删改查操作

# 1.3.1增删改操作

与DBUtils十分类似,JdbcTemplate的增删改操作使用其update("SQL语句", 参数...)方法。

创建JDBC Template

//1.获取Spring容器

ApplicationContext ac = new ClassPathXmlApplicationContext("bean.xml");

//2.根据id获取JdbcTemplate对象

JdbcTemplate jt = (JdbcTemplate) ac.getBean("jdbcTemplate");

增加操作

//3.执行增加操作

jt.update("insert into account(name,money)values(?,?)","名字",5000);

}

删除操作

//4.执行删除操作

jt.update("delete from account where id = ?",6);

更新操作

//5.执行更新操作

jt.update("update account set money = money-? where id = ?",300,6);

## 1.3.2实现查询

与DBUtils十分类似,JdbcTemplate的查询操作使用其query()方法,其参数如下:

* String sql: SQL语句
* RowMapper<T> rowMapper: 指定如何将查询结果ResultSet对象转换为T对象.
* @Nullable Object... args: SQL语句的参数

其中RowMapper类类似于DBUtils的ResultSetHandler类,可以自己写一个实现类,但常用Spring框架内置的实现类BeanPropertyRowMapper<T>(T.class)

查询所有:

List<Account> accounts = jdbcTemplate.query("select \* from account", new BeanPropertyRowMapper<Account>(Account.class));

查询一条记录:

List<Account> accounts = jdbcTemplate.query("select \* from account where id = ?", new BeanPropertyRowMapper<Account>(Account.class), accountId);

聚合查询:

JdbcTemplate中执行聚合查询的方法为queryForObject()方法,其参数如下: String sql: SQL语句

Class<T> requiredType: 返回类型的字节码

@Nullable Object... args: SQL语句的参数

//3.聚合查询

Integer total = jt.queryForObject("select count(\*) from account where money > ?", Integer.class, 500);

**1.4在DAO层使用JdbcTemplate**

**1.4.1直接使用JdbcTemplate**

在DAO层使用JdbcTemplate,需要给DAO注入JdbcTemplate对象.

public class AccountDaoImpl implements IAccountDao {

private JdbcTemplate jdbcTemplate; // JdbcTemplate对象

// JdbcTemplate对象的set方法

public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {

this.jdbcTemplate = jdbcTemplate;

}

// 其它DAO层方法...

}

**产生的问题:**

**当dao 有很多时，每个 dao 都有一些重复性的代码，都需要注入JDBCTemplate并且都要写set方法。**

**用下面一种方法可以解决这个问题。**

**1.4.2 DAO层对象继承JdbcDaoSupport类**

在实际项目中,我们会创建许多DAO对象,若每个DAO对象都注入一个JdbcTemplate对象,会造成代码冗余.

* 实际的项目中我们可以让DAO对象继承Spring内置的JdbcDaoSupport类.在JdbcDaoSupport类中定义了JdbcTemplate和DataSource成员属性,在实际编程中,只需要向其注入DataSource成员即可,DataSource的set方法中会注入JdbcTemplate对象.
* DAO的实现类中调用父类的getJdbcTemplate()方法获得JdbcTemplate()对象.

JdbcDaoSupport类的源代码如下:

public abstract class JdbcDaoSupport extends DaoSupport {

@Nullable

private JdbcTemplate jdbcTemplate; // 定义JdbcTemplate成员变量

public JdbcDaoSupport() {

}

// DataSource的set方法,注入DataSource时调用createJdbcTemplate方法注入JdbcTemplate

public final void setDataSource(DataSource dataSource) {

if (this.jdbcTemplate == null || dataSource != this.jdbcTemplate.getDataSource()) {

this.jdbcTemplate = this.createJdbcTemplate(dataSource);

this.initTemplateConfig();

}

}

// 创建JdbcTemplate,用来被setDataSource()调用注入JdbcTemplate

protected JdbcTemplate createJdbcTemplate(DataSource dataSource) {

return new JdbcTemplate(dataSource);

}

// JdbcTemplate的get方法,子类通过该方法获得JdbcTemplate对象

@Nullable

public final JdbcTemplate getJdbcTemplate() {

return this.jdbcTemplate;

}

@Nullable

public final DataSource getDataSource() {

return this.jdbcTemplate != null ? this.jdbcTemplate.getDataSource() : null;

}

public final void setJdbcTemplate(@Nullable JdbcTemplate jdbcTemplate) {

this.jdbcTemplate = jdbcTemplate;

this.initTemplateConfig();

}

// ...

}

在bean.xml中,我们只要为DAO对象注入DataSource对象即可,让JdbcDaoSupport自动调用JdbcTemplate的set方法注入JdbcTemplate

<bean id="accountDao" class="com.itheima.dao.impl.AccountDaoImpl">

<!--不用我们手动配置JdbcTemplate成员了-->

<property name="dataSource" ref="dataSource"></property>

</bean>

在实现DAO接口时,我们通过super.getJdbcTemplate()方法获得JdbcTemplate对象.

@Override

public Account findAccountById(Integer id) {

//调用继承自父类的getJdbcTemplate()方法获得JdbcTemplate对象

List<Account> list = getJdbcTemplate().query("select \* from account whereid = ? ", new AccountRowMapper(), id);

return list.isEmpty() ? null : list.get(0);

}

**注意：这种配置方法存在一个问题: 因为我们不能修改JdbcDaoSupport类的源代码,DAO层的配置就只能基于xml配置,而不再可以基于注解配置了**。

**两种方法的区别：**

第一种在 Dao 类中定义 JdbcTemplate 的方式，适用于所有配置方式（xml 和注解都可以）。

第二种让 Dao 继承 JdbcDaoSupport 的方式，只能用于基于 XML 的方式，注解用不了。

**1.5、小结**

在这一部分学习内容中，主要的内容是了解JDBCTemplate，了解JDBCTemplate是什么以及怎样使用JDBCTemplate。并且有两种方法来使用JDBCTemplate：Dao层直接使用和Dao层对象继承JdbcDaoSupport类。两种方法各有优缺点。

**2、Spring的事务控制**

事务是指数据库中的一组逻辑操作，这个操作的特点就是在该组逻辑中，**所有的操作要么全部成功，要么全部失败**。在各个数据具有特别紧密的联系时，最好是使用数据库的事务来完成逻辑处理。

1、JavaEE 体系进行分层开发,事务处理位于业务层,Spring提供了分层设计业务层的事务处理解决方  
案

2、Spring 框架为我们提供了一组事务控制的接口,这组接口在spring-tx-5.0.2.RELEASE.jar中

3、Spring 的事务控制都是基于AOP的,它既可以使用配置的方式实现,也可以使用编程的方式实现.推荐使用配置方式实现.

**2.1 Spring中事务控制的API**

**2.1.1** PlatformTransactionManager

PlatformTransactionManager接口是Spring提供的事务管理器,它提供了操作事务的方法如下:

1.TransactionStatus getTransaction(TransactionDefinition definition): 获得事务状态信息

2.void commit(TransactionStatus status): 提交事务

3.void rollback(TransactionStatus status): 回滚事务

在实际开发中我们使用其实现类:

* org.springframework.jdbc.datasource.DataSourceTransactionManager使用SpringJDBC或iBatis进行持久化数据时使用
* org.springframework.orm.hibernate5.HibernateTransactionManager使用Hibernate版本进行持久化数据时使用

**2.1.2** TransactionDefinition

TransactionDefinition: 事务定义信息对象,提供查询事务定义的方法如下:

* String getName(): 获取事务对象名称
* int getIsolationLevel(): 获取事务隔离级别,设置两个事务之间的数据可见性

事务的隔离级别由弱到强,依次有如下五种:

* 1. ISOLATION\_DEFAULT: Spring事务管理的的默认级别,使用数据库默认的事务隔离级别.
  2. ISOLATION\_READ\_UNCOMMITTED: 读未提交.
  3. ISOLATION\_READ\_COMMITTED: 读已提交.
  4. ISOLATION\_REPEATABLE\_READ: 可重复读.
  5. ISOLATION\_SERIALIZABLE: 串行化.
* getPropagationBehavior(): 获取事务传播行为,设置新事务是否事务以及是否使用当前事务.

我们通常使用的是前两种: REQUIRED和SUPPORTS.事务传播行为如下:

* 1. REQUIRED: Spring默认事务传播行为. 若当前没有事务,就新建一个事务;若当前已经存在一个事务中,加入到这个事务中.增删改查操作均可用
  2. SUPPORTS: 若当前没有事务,就不使用事务;若当前已经存在一个事务中,加入到这个事务中.查询操作可用
  3. MANDATORY: 使用当前的事务,若当前没有事务,就抛出异常
  4. REQUERS\_NEW: 新建事务,若当前在事务中,把当前事务挂起
  5. NOT\_SUPPORTED: 以非事务方式执行操作,若当前存在事务,就把当前事务挂起
  6. NEVER:以非事务方式运行,若当前存在事务,抛出异常
  7. NESTED:若当前存在事务,则在嵌套事务内执行;若当前没有事务,则执行REQUIRED类似的操作
* int getTimeout(): 获取事务超时时间. Spring默认设置事务的超时时间为-1,表示永不超时.
* boolean isReadOnly(): 获取事务是否只读. Spring默认设置为false,建议查询操作中设置为true

**2.1.3 TransactionStatus**

TransactionStatus: 事务状态信息对象,提供操作事务状态的方法如下:

* void flush(): 刷新事务
* boolean hasSavepoint(): 查询是否存在存储点
* boolean isCompleted(): 查询事务是否完成
* boolean isNewTransaction(): 查询是否是新事务
* boolean isRollbackOnly(): 查询事务是否回滚
* void setRollbackOnly(): 设置事务回滚

**2.2 Spring通过xml实现事务控制**

**2.2.1 准备工作**

首先，导入jar包到项目的lib目录，接着创建bean.xml并导入约束，准备实体类Account和持久层类AccountDaoImpl、IAccountDao、逻辑层类AccountServiceImpl、IAccountService。

同时，在bean.xml中配置数据源以及要扫描的包。

在这里，使用JDBCTemplate作为数据库的连接操作工具。

<!--配置 创建Spring容器时要扫描的包-->

<context:component-scan base-package="com.itheima"></context:component-scan>

<!--配置 数据源-->

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

<property name="driverClassName" value="com.mysql.jdbc.Driver"></property>

<property name="url" value="jdbc:mysql://localhost:3306/spring\_day02"></property>

<property name="username" value="root"></property>

<property name="password" value="1234"></property>

</bean>

<!--配置 JdbcTemplate-->

<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">

<property name="dataSource" ref="dataSource"></property>

</bean>

2.2.2 配置xml

Spring的配置式事务控制本质上是基于AOP的,因此下面我们在bean.xml中配置事务管理器PlatformTransactionManager对象并将其与AOP配置建立联系。

1. **配置事务管理器并注入数据源**

<!--向Spring容器中注入一个事务管理器-->

<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

<!--注入数据源-->

<property name="dataSource" ref="dataSource" />

</bean>

1. **配置事务通知并在通知中配置其属性**

使用<tx:advice>标签声明事务配置,其属性如下:

id: 事务配置的id

transaction-manager: 该配置对应的事务管理器

在<tx:advice>标签内包含<tx:attributes>标签表示配置事务属性

在<tx:attributes>标签内包含<tx:method>标签,为切面上的方法配置事务属性,<tx:method>标签的属性如下:

name: 拦截到的方法,可以使用通配符\*

isolation: 事务的隔离级别,Spring默认使用数据库的事务隔离级别

propagation: 事务的传播行为,默认为REQUIRED.增删改方法应该用REQUIRED,查询方法可以使用SUPPORTS

read-only: 事务是否为只读事务,默认为false.增删改方法应该用false,查询方法可以使用true

timeout: 指定超时时间,默认值为-1,表示永不超时

rollback-for: 指定一个异常,当发生该异常时,事务回滚;发生其他异常时,事务不回滚.无默认值,表示发生任何异常都回滚

no-rollback-for: 指定一个异常,当发生该异常时,事务不回滚;发生其他异常时,事务回滚.无默认值,表示发生任何异常都回滚

<!-- 配置事务的通知-->

<tx:advice id="txAdvice" transaction-manager="transactionManager">

<tx:attributes>

<!--匹配所有方法-->

<tx:method name="\*" propagation="REQUIRED" read-only="false" rollback-for="" no-rollback-for=""/>

<!--匹配所有查询方法-->

<tx:method name="find\*" propagation="SUPPORTS" read-only="true"/>

</tx:attributes>

</tx:advice>

**注意：第二个<tx:method>匹配得更精确,所以对所有查询方法,匹配第二个事务管理配置;对其他查询方法,匹配第一个事务管理配置**

3、配置AOP并为事务管理器事务管理器指定切入点

<!--配置AOP-->

<aop:config>

<!-- 配置切入点表达式-->

<aop:pointcut id="pt1" expression="execution(\* cn,maoritian.service.impl.\*.\*(..))"></aop:pointcut>

<!--为事务通知指定切入点表达式-->

<aop:advisor advice-ref="txAdvice" pointcut-ref="pt1"/>

</aop:config>

**2.3 Spring通过注解实现事务控制**

spring中基于注解 的声明式事务控制配置步骤：

1. 配置事务管理器
2. 开启spring对注解事务的支持
3. 在需要事务支持的地方使用@Transactional注解

首先需要在bean.xml中配置事务管理器。

<!--向Spring容器中注入一个事务管理器-->

<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

<!--注入数据源-->

<property name="dataSource" ref="dataSource"></property>

</bean>

接着需要开启spring对注解事务的支持

<!-- 开启spring对注解事务的支持-->

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

2.3.1半注解方式

在业务层使用@Transactional注解,其参数与<tx:method>的属性一致.

该注解可以加在接口,类或方法上

* 对接口加上@Transactional注解,表示对该接口的所有实现类进行事务控制
* 对类加上@Transactional注解,表示对类中的所有方法进行事务控制
* 对具体某一方法加以@Transactional注解,表示对具体方法进行事务控制

**三个位置上的注解优先级依次升高。**

// 业务层实现类,事务控制应在此层

@Service("accountService")

@Transactional(propagation = Propagation.REQUIRED, readOnly = false) // 读写型事务配置

public class AccountServiceImpl implements IAccountService {

@Autowired

private IAccountDao accountDao;

@Override

@Transactional(propagation = Propagation.SUPPORTS, readOnly = true) // 只读型事务配置,会覆盖上面对类的读写型事务配置

public Account findAccountById(Integer accountId) {

return accountDao.findAccountById(accountId);

}

//…其他操作

}

这里，配置在方法上的注解会覆盖前面的其他注解。

2.3.2纯注解方式

如果不使用xml配置事务,就要在cn.itheima.config包下新建一个事务管理配置类TransactionConfig,对其加上@EnableTransactionManagement注解以开启事务控制。

事务控制配置类TransactionConfig类：

@Configuration

@EnableTransactionManagement // 开启事务控制

public class TransactionConfig {

// 创建事务管理器对象

@Bean(name="transactionManager")

public PlatformTransactionManager createTransactionManager(@Autowired DataSource dataSource){

return new DataSourceTransactionManager(dataSource);

}

}

JDBC配置类JdbcConfig类:

@Configuration

@PropertySource("classpath:jdbcConfig.properties")

public class JdbcConfig {

@Value("${jdbc.driver}")

private String driver;

@Value("${jdbc.url}")

private String url;

@Value("${jdbc.username}")

private String username;

@Value("${jdbc.password}")

private String password;

// 创建JdbcTemplate对象

@Bean(name="jdbcTemplate")

@Scope("prototype")

public JdbcTemplate createJdbcTemplate(@Autowired DataSource dataSource){

return new JdbcTemplate(dataSource);

}

// 创建数据源对象

@Bean(name="dataSource")

public DataSource createDataSource(){

DriverManagerDataSource ds = new DriverManagerDataSource();

ds.setDriverClassName(driver);

ds.setUrl(url);

ds.setUsername(username);

ds.setPassword(password);

return ds;

}

}

Spring主配置类SpringConfig:

@Configuration

@ComponentScan("cn.itheima")

@Import({JdbcConfig.class, TransactionConfig.class})

public class SpringConfig {

}

2.4 小结

本部分了解了Spring中事务控制的API，同时还学习了使用Spring事务控制的步骤和方法。在Spring事务控制的实现中，也可以使用xml配置文件或者注解方式实现，可以使用半注解方式与纯注解方式实现。

**3、总结**

在这一部分学习内容中，主要的内容是学会使用Spring的JDBC Template进行数据库操作，以及Spring的事务控制对数据库的操作进行管理和维护。有两种方法来使用JDBCTemplate：Dao层直接使用和Dao层对象继承JdbcDaoSupport类。两种方法各有优缺点。在Spring事务控制的实现中，也可以使用xml配置文件或者注解方式实现，可以使用半注解方式与纯注解方式实现。