本章学习目标

- Spring 整合 Jdbc 进行持久层开发
- Spring 事务管理的 XML 方式
- Spring 事务管理的注解方式
- Spring 事务管理的零配置方式

1. Spring 整合 Jdbc 进行持久层开发

1.1. JdbcTemplate 的基本使用

JdbcTemplate 类是 Spring 框架提供用于整合 Jdcb 技术的工具类。这个工具类提供的 CRUD 操作。

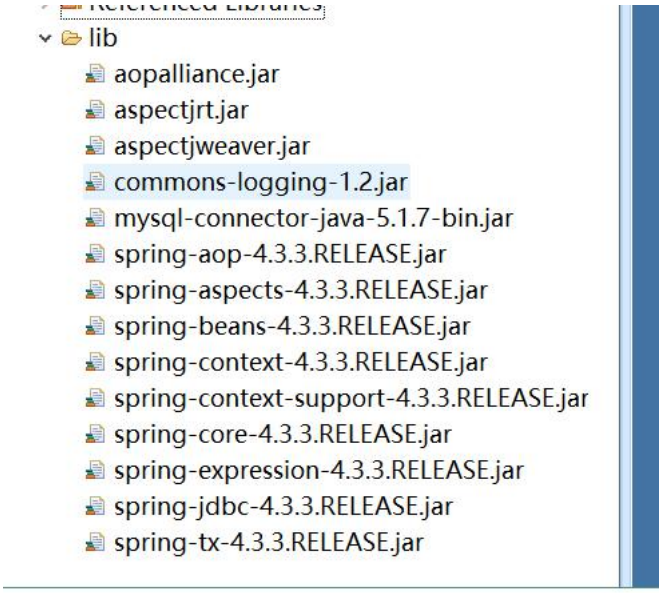
1.1.1. 导入 jar 包

spring-ioc

spring-aop

spring-jdbc

mysql 驱动程序



1.1.2. 编写代码

```
package cn.sm1234.test;


import java.util.List;


import org.junit.Test;
import org.springframework.jdbc.core.BeanPropertyRowMapper;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.datasource.DriverManagerDataSource;


import cn.sm1234.domain.Account;


public class Demo {


    @Test
    public void test1(){
        //1.创建 JdbcTemplate 对象
        JdbcTemplate temp = new JdbcTemplate();


        //2.注入数据源
        //创建 Spring 提供数据源
        DriverManagerDataSource dataSource = new
DriverManagerDataSource();
        dataSource.setUrl("jdbc:mysql:///spring");
        dataSource.setDriverClassName("com.mysql.jdbc.Driver");
        dataSource.setUsername("root");
        dataSource.setPassword("root");
```

```
        temp.setDataSource(dataSource);


        //2.使用 JdbcTemplate


        //增加
        //temp.update("insert into account(name,money)
values(?,?)","lisi",5000D);


        //修改
        //temp.update("update account set money = ? where id=?",6000D,1);


        //删除
        //temp.update("delete from account where id=?",1);


        //查询所有数据
        /*List<Account> list = temp.query("select * from account",new
BeanPropertyRowMapper(Account.class));
        for (Account account : list) {
            System.out.println(account);
        }*/


        //查询一个数据
        Account account =
                temp.queryForObject("select * from account where id = ?", new
BeanPropertyRowMapper(Account.class), 2);
        System.out.println(account);
    }
}
```

1.2. Spring 整合 jdbc 转账案例

1.2.1. Dao

```
public class AccountDaoImpl implements AccountDao {


    //注入 JdbcTemplate
    private JdbcTemplate temp;
    public void setTemp(JdbcTemplate temp) {
        this.temp = temp;
    }


    @Override
    public void outMoney(String name, Double money) {
        temp.update("update account set money = money - ? where name = ?
",money,name);
    }


    @Override
    public void InMoney(String name, Double money) {
        temp.update("update account set money = money + ? where name = ?
",money,name);
    }


}
```

1.2.2. Service

```
public class AccountServiceImpl implements AccountService {


    private AccountDao accountDao;


    public void setAccountDao(AccountDao accountDao) {
        this.accountDao = accountDao;
    }


    @Override
    public void translate(String outName, String inName, Double money) {
        //从账户扣钱
        accountDao.outMoney(outName,money);
        //把钱打入账户
        accountDao.InMoney(inName, money);
    }
}
```

1.2.3. applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:p="http://www.springframework.org/schema/p"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
```

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

    <bean id="jdbcTemplate"
class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource" ref="dataSource"/>
    </bean>

    <bean id="accountDao" class="cn.sm1234.dao.impl.AccountDaoImpl">
        <property name="temp" ref="jdbcTemplate"/>
    </bean>

    <bean id="accountService"
class="cn.sm1234.service.impl.AccountServiceImpl">
        <property name="accountDao" ref="accountDao"/>
    </bean>



</beans>
```

2. Spring 的事务管理-XML 方式

底层：采用 SpringAOP 编程

事务管理器

事务通知

事务切面

```
<!-- 事务管理 -->
    <bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager
">
        <property name="dataSource" ref="dataSource"/>
    </bean>

    <!-- 事务通知 -->
    <tx:advice id="txAdvice" transaction-manager="transactionManager">
        <tx:attributes>
            <tx:method name="*" isolation="DEFAULT"/>
        </tx:attributes>
    </tx:advice>

    <!-- 事务切面 -->
    <aop:config>
        <aop:pointcut expression="execution(public *
cn.sm1234.service.impl.AccountServiceImpl.*(..))" id="pt"/>
        <aop:advisor advice-ref="txAdvice" pointcut-ref="pt"/>
    </aop:config>
```

3. Spring 的事务管理-注解方式

3.1. applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:p="http://www.springframework.org/schema/p"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
         http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx.xsd
         http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop.xsd">



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

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

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

    <bean id="accountDao" class="cn.sm1234.dao.impl.AccountDaoImpl">
        <property name="temp" ref="jdbcTemplate"/>
    </bean>

    <bean id="accountService"
class="cn.sm1234.service.impl.AccountServiceImpl">
        <property name="accountDao" ref="accountDao"/>
    </bean>

    <!-- 事务管理 -->
    <bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager
">
        <property name="dataSource" ref="dataSource"/>
    </bean>

    <!-- 开启 Spring 事务注解 -->
    <tx:annotation-driven transaction-manager="transactionManager"/>

</beans>
```

3.2. 在需要切入事务的类加上注解

```
@Transactional(isolation=Isolation.DEFAULT)
public class AccountServiceImpl implements AccountService {
```

```
    private AccountDao accountDao;


    public void setAccountDao(AccountDao accountDao) {
        this.accountDao = accountDao;
    }


    @Override
    public void translate(String outName, String inName, Double money) {
        //从账户扣钱
        accountDao.outMoney(outName,money);


        int i = 100/0;


        //把钱打入账户
        accountDao.InMoney(inName, money);
    }
}
```

4. Spring 的事务管理-零注解方式

```
@Configurable
@ComponentScan(basePackages="cn.sm1234")
@Import(JdbcConfig.class)
public class SpringConfig {
}
```

```
/**
 * 获取数据源
```

```
 * @author lenovo
 *
 */
@PropertySource(value="classpath:jdbc.properties")
@EnableTransactionManagement
public class JdbcConfig {


    @Value("${jdbc.url}")
    private String url;
    @Value("${jdbc.driverClass}")
    private String driverClass;
    @Value("${jdbc.username}")
    private String username;
    @Value("${jdbc.password}")
    private String password;


    @Bean(name="dataSource")
    public DataSource getDataSource(){
        DriverManagerDataSource dataSource = new
DriverManagerDataSource();
        dataSource.setUrl(url);
        dataSource.setDriverClassName(driverClass);
        dataSource.setUsername(username);
        dataSource.setPassword(password);
        return dataSource;
    }


    @Bean(name="jdbcTemplate")
    public JdbcTemplate
```

```
getJdbcTemplate(@Qualifier("dataSource")DataSource dataSource){
        JdbcTemplate temp = new JdbcTemplate();
        temp.setDataSource(dataSource);
        return temp;
    }

    @Bean(name="transactionManager")
    public DataSourceTransactionManager
getTransactionManager(@Qualifier("dataSource")DataSource dataSource){
        DataSourceTransactionManager tm = new
DataSourceTransactionManager();
        tm.setDataSource(dataSource);
        return tm;
    }


}
```