

# spring的jdbcTemplate操作和事务

## spring的jdbcTemplate操作

### 1. spring框架一站式框架



- 1) 正对javaEE三层，每一层都有解决技术
- 2) 在dao层，使用JdbcTemplate

### 2. spring对不同的持久层技术都进行了封装


### 3) JdbcTemplate使用和dbutils使用很相似，都对数据库进行crud操作。

## 增加

### 1. 导入相关jar包

- >  spring-jdbc-4.3.6.RELEASE.jar - D:\apache-ma
- >  spring-tx-4.3.6.RELEASE.jar - D:\apache-mave

### 数据库驱动jar包 (注意驱动类的版本号)

- >  mysql-connector-java-6.0.5.jar - D:\apache-maven-3.3.!

## 代码：

//添加操作

@Test

public void add(){

//设置数据库信息

DriverManagerDataSource dataSource = new DriverManagerDataSource();

dataSource.setDriverClassName("com.mysql.jdbc.Driver");

dataSource.setUrl("jdbc:mysql:///test");

dataSource.setUsername("root");

dataSource.setPassword("root");

//创建JdbcTemplate对象，设置数据源

JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);

//调用JdbcTemplate对象里面的方法

//创建sql语句

String sql = "INSERT INTO user values(?,?)";

int i = jdbcTemplate.update(sql, "aaa", "a1");//返回值为修改记录的条数

System.out.println(i);

}

## 修改操作

//修改操作

public void update(){

DriverManagerDataSource dataSource = new DriverManagerDataSource();

dataSource.setDriverClassName("com.mysql.jdbc.Driver");

dataSource.setUrl("jdbc:mysql:///test");

dataSource.setUsername("root");

dataSource.setPassword("root");

//创建JdbcTemplate对象，设置数据源

JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);

//调用JdbcTemplate对象里面的方法

//创建sql语句

String sql = "UPDATE user set password=? where username=?";

int i = jdbcTemplate.update(sql, "a2", "aaa");//返回值为修改记录的条数

System.out.println(i);

}

## 删除操作

```
//删除操作
@Test
public void delete(){
    DriverManagerDataSource dataSource = new DriverManagerDataSource();
    dataSource.setDriverClassName("com.mysql.jdbc.Driver");
    dataSource.setUrl("jdbc:mysql:///test");
    dataSource.setUsername("root");
    dataSource.setPassword("root");

    JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);
    String sql = "delete from user where username=?";
    int i = jdbcTemplate.update(sql, "aaa");
    System.out.println(i);
}
```

## 查询操作

返回某一个值

//查询操作，返回摸一个值

```
@Test
public void select1(){
    DriverManagerDataSource dataSource = new DriverManagerDataSource();
    dataSource.setDriverClassName("com.mysql.jdbc.Driver");
    dataSource.setUrl("jdbc:mysql:///test");
    dataSource.setUsername("root");
    dataSource.setPassword("root");

    JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);
    String sql = "select count(*) from user";
    //调用jdbcTemplate的方法
    int count = jdbcTemplate.queryForObject(sql, Integer.class);
    System.out.println(count);
}
```

返回某一个对象（使用queryForObject方法）

● `queryForObject(String sql, Object[] args, RowMapper<T> rowMapper) : T - Jd`

第一个参数：sql语句

第二个参数：RowMapper接口

第三个参数：可变参数，sql语句中的值。

返回对象集合

代码：

//返回对象

```
@Test
public void selectObject(){
    DriverManagerDataSource dataSource = new DriverManagerDataSource();
    dataSource.setDriverClassName("com.mysql.jdbc.Driver");
    dataSource.setUrl("jdbc:mysql:///test");
    dataSource.setUsername("root");
    dataSource.setPassword("root");
```

```

JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);
String sql = "select * from user where username=?";
//
User user = jdbcTemplate.queryForObject(sql, new MYRowMapper(), "aaa");
System.out.println(user);
}

```

```

class MYRowMapper implements RowMapper<User>{
    @Override
    public User mapRow(ResultSet rs, int num) throws SQLException {
        //从结果集中得到数据
        String username = rs.getString("username");
        String password = rs.getString("password");
        //将得到的数据封装到对象里面
        User user = new User();
        user.setUsername(username);
        user.setPassword(password);
        return user;
    }
}

```

返回集合

//查询返回集合

```

public void selectList(){
    DriverManagerDataSource dataSource = new DriverManagerDataSource();
    dataSource.setUrl("jdbc:mysql:///test");
    dataSource.setUsername("root");
    dataSource.setPassword("root");
}

```

```

JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);
String sql = "select * from user";
List<User> uList = jdbcTemplate.query(sql, new MYRowMapper());
for(int i = 0 ; i < uList.size(); i++){
    System.out.println(uList.get(i));
}
}

```

```

class MYRowMapper implements RowMapper<User>{

    @Override
    public User mapRow(ResultSet rs, int num) throws SQLException {
        //从结果集中得到数据
        String username = rs.getString("username");
        String password = rs.getString("password");
        //将得到的数据封装到对象里面
        User user = new User();
        user.setUsername(username);
        user.setPassword(password);
        return user;
    }

}

```

## Spring配置连接池和DAO使用JdbcTemplate

### 1.spring 配置连接池

以c3p0连接池

导入jar包

```
> c3p0-0.9.1.2.jar - F:\java\maven\apache-maven-3.3.3\repo\c3p0\c3p0-0.9.1.2.jar
> mchange-commons-java-0.2.12.jar - F:\java\maven\apache-maven-3.3.3\repo\mchange-commons-java-0.2.12.jar
```

创建spring的配置文件

**添加约束，创建c3p0连接池对象，创建service和dao对象，在service中注入dao属性，在dao中注入jdbcTemplate属性，在jdbcTemplate中注入datasource属性。**

```
<!-- 配置c3p0连接池 -->
<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
  <!-- 诸如属性值 -->
  <property name="driverClass" value="com.mysql.jdbc.Driver"/>
  <property name="jdbcUrl" value="jdbc:mysql:///test"/>
  <property name="user" value="root"/>
  <property name="password" value="root"/>
</bean>

<!-- 创建service 和 dao 对象 -->
<bean id="userService" class="com.wmr.springDay03.c3p0.UserService">
  <property name="userDao" ref="userDao"/>
</bean>

<bean id="userDao" class="com.wmr.springDay03.c3p0.UserDao">
  <!-- 注入jdbcTemplate属性 -->
  <property name="jdbcTemplate" ref="jdbcTemplate"/>
</bean>

<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
  <!-- 注入datasource属性 -->
  <property name="dataSource" ref="dataSource"/>
</bean>
```

### 2.dao中使用JdbcTemplate

```
public class UserDao {
    private JdbcTemplate jdbcTemplate;

    public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    //添加操作
    public void add(){
        String sql = "insert into user values(?,?)";
        int rows = jdbcTemplate.update(sql, "ddd","d1");
        System.out.println(rows);
    }
}

public class UserService {
```

```

private UserDao userDao;

public void setUserDao(UserDao userDao) {
    this.userDao = userDao;
}

public void add(){
    System.out.println("service add!");
    userDao.add();
}
}

```

测试代码：

```

public class C3p0Test {
    @Test
    public void testDemo(){
        ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
        UserService service = (UserService) context.getBean("userService");
        service.add();
    }
}

```

## spring事务管理

.事务的概念：

- 1.什么是事务
- 2.事物的特性
- 3.不考虑个理性产生的读问题

4解决方法：

- 1) 设置隔离级别

spring事务管理API

- 1) 编程式事务管理（不用）
- 2) 声明式
  - 基于xml的配置文件方式实现
  - 基于注解方式实现

搭建转账环境

需求：

aaa装张1000给bbb

- 1) 创建数据库表，添加数据

id	username	salary
1	aaa	10000
2	bbb	10000

- 2) 创建service和dao类，完成注入关系
  - service层（业务逻辑层），实现业务操作
  - dao层持久层，操作数据库

准备加入spring事务约束

```
<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"
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/context http://www.springframework.org/schema/context/spring-context.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">
```

配置文件方式：

配置文件：

第一步：配置事务管理器

第二步：配置事务处理规则

第三步：配置切面

```
<!-- 转账环境 创建对象，并注入相关对象的相关属性 -->
<bean id="account" class="com.wmr.springDay03.tx.Account"></bean>
<bean id="accountService" class="com.wmr.springDay03.tx.AccountService">
    <property name="accountDao" ref="accountDao"/>
</bean>
<bean id="accountDao" class="com.wmr.springDay03.tx.AccountDao">
    <property name="jdbcTemplate" ref="jdbcTemplate"/>
</bean>

<!-- 声明式事务处理 配置文件方式 使用AOP思想 -->
<!-- 配置事务管理器 -->
<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"/>
</bean>
<!-- 第二步 -->
<!-- 配置事务增强 -->
<tx:advice id="txAdvice" transaction-manager="transactionManager">
    <!-- 做事务操作 -->
    <tx:attributes>
        <!-- 设置进行事务操作匹配方法 -->
        <tx:method name="account*" propagation="REQUIRED"/><!-- 以account开头的方法都可以做事务操作，propagation事务隔离级别 -->
    </tx:attributes>
</tx:advice>

<!-- 第三步 -->
<!-- 配置切面 -->
<aop:config>
    <!-- 切入点 -->
    <aop:pointcut expression="execution(* com.wmr.springDay03.tx.AccountService.*(..))" id="pointcut1"/>
    <!-- 切面 -->
    <aop:advisor advice-ref="txAdvice" pointcut-ref="pointcut1"/>
</aop:config>
```

注解事务处理方式

第一步：配置事务管理器

第二步：配置事务注解

第三步：要使用事务方法，所在类上面添加注解

配置文件：

```

<!-- 配置c3p0连接池 -->
<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
    <!-- 添加属性值 -->
    <property name="driverClass" value="com.mysql.jdbc.Driver" />
    <property name="jdbcUrl" value="jdbc:mysql:///test" />
    <property name="user" value="root" />
    <property name="password" value="root" />
</bean>

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

<!-- 解除环境 创建对象，并注入相关对象的相关属性 -->
<bean id="account" class="com.wmr.springDay03.tx.Account"></bean>
<bean id="accountService" class="com.wmr.springDay03.tx.AccountService">
    <property name="accountDao" ref="accountDao" />
</bean>
<bean id="accountDao" class="com.wmr.springDay03.tx.AccountDao">
    <property name="jdbcTemplate" ref="jdbcTemplate" />
</bean>

<!-- 声明式事务处理 配置文件方式 使用aop思想 -->
<!-- 第一步 -->
<!-- 配置事务管理器 -->
<bean id="transactionManager"
    class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />
</bean>

<!-- 第二步开启事务注解 -->
<tx:annotation-driven transaction-manager="transactionManager"/>

```

代码：

```

public class Account {
    private Integer id;
    private String username;
    private Integer salary;

    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public Integer getSalary() {
        return salary;
    }
    public void setSalary(Integer salary) {
        this.salary = salary;
    }
}

//对数据库操作
public class AccountDao {
    private JdbcTemplate jdbcTemplate;

    public JdbcTemplate getJdbcTemplate() {
        return jdbcTemplate;
    }
}

```

```
}
```

```
public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {  
    this.jdbcTemplate = jdbcTemplate;  
}
```

```
//少钱的方法
```

```
public void lessMoney(){  
    String sql = "update account set salary=salary+? where username=?";  
    jdbcTemplate.update(sql, -1000, "aaa");  
}
```

```
//加钱的方法
```

```
public void moreMoney(){  
    String sql = "update account set salary=salary+? where username=?";  
    jdbcTemplate.update(sql, 1000, "bbb");  
}  
}
```

```
@Transactional
```

```
public class AccountService {  
    private AccountDao accountDao;
```

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

```
//转账操作
```

```
public void accountMoney(){  
    //aaa少1000  
    accountDao.lessMoney();  
    //bbb多1000  
    accountDao.moreMoney();  
}
```

```
}
```

测试：

```
@Test
```

```
public void txAnoDemo(){  
    ApplicationContext context = new ClassPathXmlApplicationContext("txann.xml");  
    AccountService accountService = (AccountService) context.getBean("accountService");  
    accountService.accountMoney();  
}
```