4、声明式事务-JdbcTemplate

笔记本： spring
创建时间： 2022/4/3 17:03
作者： 雷丰阳

事务：
操作数据库；
Spring提供了JdbcTemplate能快捷的操作数据库；
JdbcTemplate和QueryRunner；
JdbcTemplate使用步骤：
1）、导包；

```
spring-jdbc-4.0.0.RELEASE.jar
spring-orm-4.0.0.RELEASE.jar
spring-tx-4.0.0.RELEASE.jar
```

2）、写配置

```
    <!--引入外部配置文件  -->
    <context:property-placeholder location="classpath:dbconfig.properties"/>

    <!-- 实验1：测试数据源
    ${}取出配置文件中的值
    #{}Spring的表达式语言
     -->
    <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
        <property name="user" value="${jdbc.user}"></property>
        <property name="password" value="${jdbc.password}"></property>
        <property name="jdbcUrl" value="${jdbc.jdbcUrl}"></property>
        <property name="driverClass" value="${jdbc.driverClass}"></property>
    </bean>

    <!-- Spring提供了一个类JdbcTemplate，我们用它操作数据库；
        导入Spring的数据库模块
spring-jdbc-4.0.0.RELEASE.jar
spring-orm-4.0.0.RELEASE.jar
spring-tx-4.0.0.RELEASE.jar
         -->
    <bean id="jdbcTemplate"
class="org.springframework.jdbc.core.JdbcTemplate">
        <constructor-arg name="dataSource" ref="dataSource"></constructor-
arg>
    </bean>
```

3）、测试

```
@Test
    public void test02(){
        String sql = "UPDATE employee SET salary=? WHERE emp_id=?";
        int update = jdbcTemplate.update(sql, 1300.00,5);
        System.out.println("更新员工："+update);
    }
```

```
package com.atguigu.test;

import static org.junit.Assert.*;
```

```
import java.sql.Connection;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import javax.sql.DataSource;

import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.dao.DataAccessException;
import org.springframework.jdbc.core.BeanPropertyRowMapper;
import org.springframework.jdbc.core.JdbcTemplate;
import
org.springframework.jdbc.core.namedparam.BeanPropertySqlParameterSource;
import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;

import com.atguigu.bean.Employee;
import com.atguigu.dao.EmployeeDao;

public class TxTest {

    ApplicationContext ioc = new
ClassPathXmlApplicationContext("ApplicationContext.xml");
    JdbcTemplate jdbcTemplate = ioc.getBean(JdbcTemplate.class);
    NamedParameterJdbcTemplate namedJdbcTemplate =
ioc.getBean(NamedParameterJdbcTemplate.class);

    /**
     * 实验9：创建BookDao，自动装配JdbcTemplate对象
     */
    @Test
    public void test09(){
        EmployeeDao bean = ioc.getBean(EmployeeDao.class);
        Employee employee = new Employee();
        employee.setEmpName("哈哈2");
        employee.setSalary(998.98);
        bean.saveEmployee(employee);
    }

    /**
     * 实验8：重复实验7，以SqlParameterSource形式传入参数值
     */
    @Test
    public void test08(){
        String sql = "INSERT INTO employee(emp_name,salary)
VALUES(:empName,:salary)";
        Employee employee = new Employee();
        employee.setEmpName("哈哈");
        employee.setSalary(998.98);

        //
        int i = namedJdbcTemplate.update(sql, new
BeanPropertySqlParameterSource(employee));
        System.out.println(i);

    }

    /**
     * 实验7：使用带有具名参数的SQL语句插入一条员工记录，并以Map形式传入参数值
     *
     * 具名参数：（具有名字的参数，参数不是占位符了，而是一个变量名）
     *          语法格式：    :参数名
     * Spring有一个支持具名参数功能的JdbcTemplate
     *
     * 占位符参数：?的顺序千万不能乱。传参的时候一定注意；
     */
    @Test
    public void test07(){
        String sql = "INSERT INTO employee(emp_name,salary)
VALUES(:empName,:salary)";
```

```
        //Map
        Map<String, Object> paramMap = new HashMap<>();
        //将所有具名参数的值都放在map中；
        paramMap.put("empName", "田七");
        paramMap.put("salary", 9887.98);
        int update = namedJdbcTemplate.update(sql, paramMap);
        System.out.println(update);
    }

    /**
     * 实验6：查询最大salary
     */
    @Test
    public void test06(){
        String sql = "select max(salary) from employee";
        //无论是返回单个数据还是单个对象，都是调用queryForObject
        Double object = jdbcTemplate.queryForObject(sql, Double.class);
        System.out.println(object);
    }

    /**
     * 实验5：查询salary>4000的数据库记录，封装为List集合返回
     */
    @Test
    public void test05(){
        String sql = "SELECT emp_id empId,emp_name empName,salary FROM
employee WHERE salary>?";
        //封装List；集合里面元素的类型
        List<Employee> list = jdbcTemplate.query(sql, new
BeanPropertyRowMapper<>(Employee.class), 4000);

        for (Employee employee : list) {
            System.out.println(employee);
        }
    }

    /**
     * 实验4：查询emp_id=5的数据库记录，封装为一个Java对象返回；
     * javaBean需要和数据库中字段名一致，否则无法完成封装；
     *
     * jdbcTemplate在方法级别进行了区分
     * 查询集合：jdbcTemplate.query()
     * 查询单个对象：jdbcTemplate.queryForObject()
     *          如果查询没结果就报错；
     *
     */
    @Test
    public void test04(){
        String sql = "SELECT emp_id empId,emp_name empName,salary FROM
employee WHERE emp_id=?";
        //RowMapper：每一行记录和javaBean的属性如何映射
        Employee employee = null;
        try {
            employee = jdbcTemplate.queryForObject(sql, new
BeanPropertyRowMapper<>(Employee.class), 50);
        } catch (DataAccessException e) {

        }
        System.out.println(employee);
    }

    /**
     * 实验3：批量插入；
     */
    @Test
    public void test03(){
        String  sql ="INSERT INTO employee(emp_name,salary) VALUES(?,?)";
        //List<Object[]>
        //List的长度就是sql语句要执行的次数
        //Object[]：每次执行要用的参数
        List<Object[]> batchArgs = new ArrayList<Object[]>();
        batchArgs.add(new Object[]{"张三",1998.98});
        batchArgs.add(new Object[]{"李四",2998.98});
```

```
        batchArgs.add(new Object[]{"王五",3998.98});
        batchArgs.add(new Object[]{"赵六",4998.98});

        int[] is = jdbcTemplate.batchUpdate(sql, batchArgs);
        for (int i : is) {
            System.out.println(i);
        }
    }

    /**
     * 实验2：将emp_id=5的记录的salary字段更新为1300.00
     */
    @Test
    public void test02(){
        String sql = "UPDATE employee SET salary=? WHERE emp_id=?";
        int update = jdbcTemplate.update(sql, 1300.00,5);
        System.out.println("更新员工："+update);
    }

    @Test
    public void test() throws SQLException {
        DataSource bean = ioc.getBean(DataSource.class);
        Connection connection = bean.getConnection();
        System.out.println(connection);
        connection.close();

    }

    @Test
    public void test01() throws SQLException {
        System.out.println(jdbcTemplate);
    }

}
```
