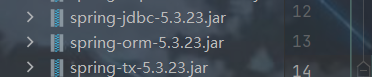
**一、概念和准备**

**1.什么是JdbcTemplate**

（1）Spring框架对JDBC进行封装，使用JDBCTemplate方便实现对数据库的操作

**2.准备工作**

2.1引入相关的jar包



截图.png

2.2在spring中配置数据库连接池

<!-- 数据库连接池 -->

<bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource"

destroy-method="close">

<property name="url" value="jdbc:mysql:///user\_db" />

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

<property name="password" value="123456" />

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

</bean>

2.3配置JDBCTemplate对象，注入DataSource

<!--JDBCTemplate对象-->

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

<!--注入DataSource-->

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

</bean>

2.4创建service类和dao类，在dao注入jdbcTemplate对象

\* 配置文件

<!--开启组件扫描-->

<context:component-scan base-package="Spring5.dao,Spring5.service"></context:component-scan>

Service

@Service

public class BookService {

//注入dao

@Autowired

private BookDao bookDao;

}

dao

@Repository

public class BookDaoImpl {

//注入jdbcTemplate对象

@Autowired

private JdbcTemplate jdbcTemplate;

}

**二、JDBC操作数据库**

**2.1添加功能**

**1.为对应的数据库创建实体类**

package Spring5.entity;

/\*\*

\* @author xh

\* @date 2022/10/15

\* @apiNote

\*/

public class Book {

private String user\_id;

private String username;

private String uStatus;

public String getUser\_id() {

return user\_id;

}

public void setUser\_id(String user\_id) {

this.user\_id = user\_id;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getuStatus() {

return uStatus;

}

public void setuStatus(String uStatus) {

this.uStatus = uStatus;

}

}

**2.编写service和dao**

（1）在dao进行数据库添加操作

（2）调用JdbcTemplate对象里面update方法实现添加

有两个参数

第一个参数：SQL语句

第二个参数：可变参数，设置SQL语句值

package Spring5.dao;

import Spring5.entity.Book;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.stereotype.Repository;

/\*\*

\* @author xh

\* @date 2022/10/13

\* @apiNote

\*/

@Repository

public class BookDaoImpl implements BookDao{

//注入jdbcTemplate对象

@Autowired

private JdbcTemplate jdbcTemplate;

@Override

public void add(Book book) {

String sql = "INSERT INTO t\_book VALUES(?,?,?)";

String[] args = {book.getUser\_id(),book.getUsername(),book.getuStatus()};

int update = jdbcTemplate.update(sql, args);

System.out.println(update);

}

}

package Spring5.service;

import Spring5.dao.BookDao;

import Spring5.entity.Book;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

/\*\*

\* @author xh

\* @date 2022/10/13

\* @apiNote

\*/

@Service

//此类的主要作用是对DAO层的封装，方便使用者直接调用

public class BookService {

//注入dao

@Autowired

private BookDao bookDao;

public void addBook(Book book){

bookDao.add(book);

}

}

（3）编写测试类

public class TestBook {

@Test

public void test1(){

Book book = new Book();

book.setUser\_id("0");

book.setUsername("小明");

book.setuStatus("Online");

ApplicationContext applicationContext = new ClassPathXmlApplicationContext("bean1.xml");

BookService bookservice = applicationContext.getBean("bookService", BookService.class);

bookservice.addBook(book);

System.out.println(bookservice);

}

}

**修改与删除功能与其类似，以下为代码**

//BookDaoImpl类

@Override

public void updateBook(Book book) {

String sql = "update t\_book set username = ?, ustatus = ? where user\_id = ?";

String[] args = {book.getUsername(),book.getuStatus(), book.getUser\_id()};

int update = jdbcTemplate.update(sql, args);

System.out.println(update);

}

@Override

public void deleteBook(String id) {

String sql = "delete from t\_book where user\_id = ?";

int update = jdbcTemplate.update(sql, id);

System.out.println(update);

}

//service类

//修改功能

public void updateBook(Book book){

bookDao.updateBook(book);

}

//删除功能

public void deleteBook(String id){

bookDao.deleteBook(id);

}

//测试类

@Test

public void test2(){

Book book = new Book();

book.setUser\_id("2");

book.setUsername("张三");

book.setuStatus("leave");

ApplicationContext applicationContext = new ClassPathXmlApplicationContext("bean1.xml");

BookService bookService = applicationContext.getBean("bookService", BookService.class);

bookService.updateBook(book);

}

@Test

public void test3(){

String id = "0";

ApplicationContext context = new ClassPathXmlApplicationContext("bean1.xml");

BookService bookService = context.getBean("bookService", BookService.class);

bookService.deleteBook(id);

}

**2.2 查询操作**

1.查询表里面有多少条记录

SQL语句：select count（\*）from t\_book；

2.使用jdbcTemplate实现查询返回某个值代码

//BookDao接口

{

int queryCount();

}

//BookDaoImpl类

@Override

public int queryCount() {

String sql = "select count(\*) from t\_book";

Integer i = jdbcTemplate.queryForObject(sql,Integer.class);

System.out.println(i);

return i;

}

//BookService类

//查询功能

public int queryCount(){

int i = bookDao.queryCount();

return i;

}

//查询

@Test

public void test4(){

ApplicationContext applicationContext = new ClassPathXmlApplicationContext("bean1.xml");

BookService bookService = applicationContext.getBean("bookService", BookService.class);

bookService.queryCount();

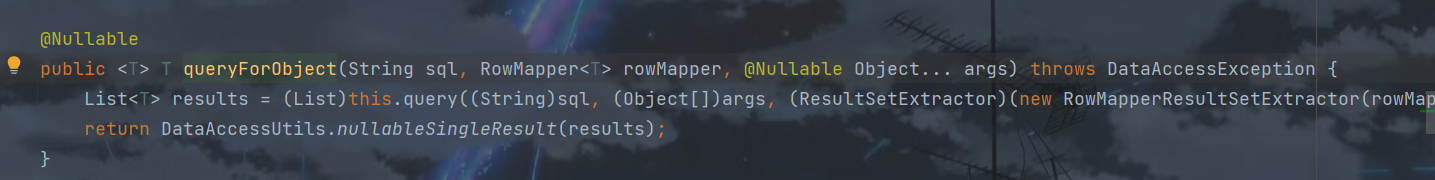
}

**2.3查询返回对象**

**1.应用场景：查询图书、商品，需要返回一个完整的对象,包含各种属性**

**2.JdbcTemplate实现查询返回对象**

调用方法



第一个参数：传入SQL语句

第二个参数：RowMapper接口，此处使用BeanProxyRowMapper类对象，可以根据该传入的类和数据创建该类的对象（将传入该 方法中的数据封装到指定的类对象中去）

第三个参数：可变参数，为SQL语句中需要传入的参数

具体代码如下：

//BookDao 接口

{

public Book selectObject(String id);

}

//BookDaoImpl实现类，调用JdbcTemplate对象实现对数据库的操作

@Override

public Book selectObject(String id) {

String sql = "select \* from t\_book where user\_id = ?";

Book book = jdbcTemplate.queryForObject(sql, new BeanPropertyRowMapper<Book>(Book.class), id);

return book;

}

//BookService类，对dao层的封装

public Book selectObject(String id){

Book book = bookDao.selectObject(id);

return book;

}

//测试方法

@Test

public void test4(){

ApplicationContext applicationContext = new ClassPathXmlApplicationContext("bean1.xml");

BookService bookService = applicationContext.getBean("bookService", BookService.class);

//bookService.queryCount();

Book book = bookService.selectObject("1");

System.out.println(book);

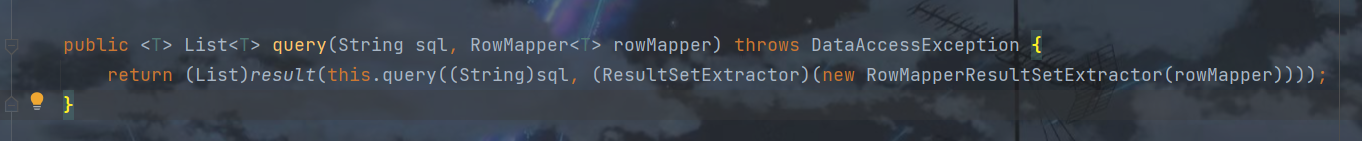
}

**2.4查询返回集合**

**1.应用场景：图书查询分页展示（一次查询多个对象）**

**2.JdbcTemplate实现查询返回集合**

调用query方法



参数一：sql语句

参数二：RowMapper接口的实现类，此处为BeanProxyRowmapper类，将传入的数据封装到T类的实例对象中去

参数三：SQL语句值，可省略

**3.代码**

//BookDao 接口

{

public Book selectList();

}

//BookDaoImpl实现类，调用JdbcTemplate对象实现对数据库的操作

@Override

public List<Book> selectList() {

String sql = "select \* from t\_book ";

List<Book> bookList = jdbcTemplate.query(sql, new BeanPropertyRowMapper<Book>(Book.class));

return bookList;

}

//BookService类，对dao层的封装

public List<Book> selectList(){

List<Book> book = bookDao.selectList();

return book;

}

//测试方法

@Test

public void test4(){

ApplicationContext applicationContext = new ClassPathXmlApplicationContext("bean1.xml");

BookService bookService = applicationContext.getBean("bookService", BookService.class);

//bookService.queryCount();

BookList<Book> bookList = bookService.selectList();

System.out.println(bookList);

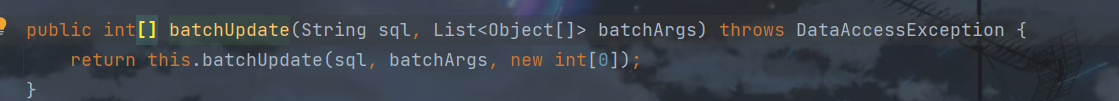
}

**2.5批量添加功能**

1.批量操作：操作表里面有多条数据记录

2.jdbcTemplate实现批量添加操作

调用batchUpdate方法



参数一：SQL语句

参数二：传入要添加的数据集合。底层原理：List集合会依次遍历集合中的每个数组，然后将每个数组的值传递给SQL语句实现添加操作

3.实现代码

//Dao接口

{public void addBatch(List<Object[]> batchArgs);}

//Dao实现类

@Override

public void addBatch(List<Object[]> batchArgs) {

String sql = "insert into t\_book values(?,?,?)";

int[] ints = jdbcTemplate.batchUpdate(sql, batchArgs);

System.out.println(Arrays.toString(ints)); //数组对象的输出，调用Arrays类的toString方法

}

//Service类

public void addBatch(List<Object[]> batchArgs){

bookDao.addBatch(batchArgs);

}

//测试

@Test

public void test5(){

ApplicationContext applicationContext = new ClassPathXmlApplicationContext("bean1.xml");

BookService bookService = applicationContext.getBean("bookService", BookService.class);

//List集合会遍历集合中的每个数组，并依次传入到SQL语句中

List<Object[]> batchArgs = new ArrayList<>();

String[] args1 = {"4","Java","a"};

String[] args2 = {"6","C++","b"};

String[] args3 = {"5","Python","c"};

batchArgs.add(args1);

batchArgs.add(args2);

batchArgs.add(args3);

bookService.addBatch(batchArgs);

}

**2.6批量修改功能**

//Dao接口

{public void updateBatch(List<Object[]> batchArgs);}

//Dao实现类

@Override

public void updateBatch(List<Object[]> batchArgs) {

String sql = "update t\_book set username = ?,ustatus = ? where user\_id = ?";

int[] ints = jdbcTemplate.batchUpdate(sql, batchArgs);

System.out.println(Arrays.toString(ints)); //数组对象的输出，调用Arrays类的toString方法

}

//Service类

public void updateBatch(List<Object[]> batchArgs){

bookDao.updateBatch(batchArgs);

}

//测试

@Test

public void test5(){

ApplicationContext applicationContext = new ClassPathXmlApplicationContext("bean1.xml");

BookService bookService = applicationContext.getBean("bookService", BookService.class);

//List集合会遍历集合中的每个数组，并依次传入到SQL语句中

List<Object[]> batchArgs = new ArrayList<>();

String[] args1 = {"Java1","a","4"};

String[] args2 = {"C++1","b","6"};

String[] args3 = {"Python1","c","5"};

batchArgs.add(args1);

batchArgs.add(args2);

batchArgs.add(args3);

bookService.addBatch(batchArgs);

}

**2.6批量删除功能**

//1.dao接口

{

void deleteBatch(List<Object[]> batchArgs);

}

//dao实现类

@Override

public void deleteBatch(List<Object[]> batchArgs) {

String sql = "delete from t\_book where user\_id = ?";

int[] ints = jdbcTemplate.batchUpdate(sql, batchArgs);

System.out.println(Arrays.toString(ints));

}

//service类

public void deleteBatch(List<Object[]> batchArgs){

bookDao.deleteBatch(batchArgs);

}

//批量删除

@Test

public void test7(){

ApplicationContext context = new ClassPathXmlApplicationContext("bean1.xml");

BookService bookService = context.getBean("bookService", BookService.class);

List<Object[]> stringList = new ArrayList<>();

String[] a = {"3"};

String[] b = {"4"};

stringList.add(a);

stringList.add(b);

bookService.deleteBatch(stringList);

}