### JAVA第二阶段—DAY05-JAVA案例

1. JDBC Dao工具类

* 需求

实现Dao基类，用来连接数据库

* 参考答案

import java.sql.\*;  
  
public class BaseDao {  
 private String driver = "com.mysql.cj.jdbc.Driver";// 数据库驱动字符串  
 private String url = "jdbc:mysql://localhost:3306/examdb?characterEncoding=utf8&useSSL=false&serverTimezone=GMT%2B8&rewriteBatchedStatements=true&allowPublicKeyRetrieval=true";// 连接URL字符串  
 private String user = "tl"; // 数据库用户名  
 private String password = "Tlpwd\_1234"; // 用户密码  
 Connection conn = null; // 数据连接对象  
 /\*\*  
 \* 获取数据库连接对象  
 \*/  
 public Connection getConnection() {  
 if(conn==null) {  
 // 获取连接并捕获异常  
 try {  
 Class.forName(driver);  
 conn = DriverManager.getConnection(url, user, password);  
 } catch (Exception e) {  
 e.printStackTrace(); // 异常处理  
 }  
 }  
 return conn; // 返回连接对象  
 }  
 /\*\*  
 \* 关闭数据库连接  
 \* @param conn 数据库连接  
 \* @param stmt Statement对象  
 \* @param rs 结果集  
 \*/  
 public void closeAll(Connection conn, Statement stmt,  
 ResultSet rs) {  
 // 若结果集对象不为空,则关闭  
 if (rs != null) {  
 try {  
 rs.close();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 // 若Statement对象不为空,则关闭  
 if (stmt != null) {  
 try {  
 stmt.close();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 // 若数据库连接对象不为空,则关闭  
 if (conn != null) {  
 try {  
 conn.close();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 /\*\*  
 \* 增、删、改的操作  
 \* @param sql 预编译的 SQL 语句  
 \* @param param 参数的字符串数组  
 \* @return 影响的行数  
 \*/  
 public int exceuteUpdate (String preparedSql, Object[] param) {  
 PreparedStatement pstmt = null;  
 int num = 0;  
 conn = getConnection();  
 try {  
 pstmt = conn.prepareStatement(preparedSql);  
 if (param != null) {  
 for (int i = 0; i < param.length; i++) {  
 //为预编译sql设置参数  
 pstmt.setObject(i + 1, param[i]);  
 }  
 }  
 num = pstmt.executeUpdate();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 } finally{  
 closeAll(conn,pstmt,null);  
 }  
 return num;  
 }  
}

## **实现对象操作接口与实现类**

* 需求

实现对象接口来操作dao

* 参考答案

public interface PetDao {

/\*\*

\* 保存宠物。

\* @param pet 宠物

\*/

int save(Pet pet);

/\*\*

\* 删除宠物。

\* @param pet 宠物

\*/

int del(Pet pet);

/\*\*

\* 更新宠物。

\* @param pet 宠物

\*/

int update(Pet pet);

/\*\*

\* 获取指定昵称的宠物,精确查询。

\* @param name 昵称

\* @return 宠物

\*/

Pet getByName(String name);

/\*\*

\* 获取指定昵称的宠物列表,模糊查询。

\* @param name 昵称

\* @return 宠物列表

\*/

List<Pet> findByName(String name);

/\*\*

\* 获取指定类型的宠物列表。

\* @param type 宠物类型

\* @return 宠物列表

\*/

List<Pet> findByType(String type);

}

public interface PetDao {

/\*\*

\* 保存宠物。

\* @param pet 宠物

\*/

int save(Pet pet);

/\*\*

\* 删除宠物。

\* @param pet 宠物

\*/

int del(Pet pet);

/\*\*

\* 更新宠物。

\* @param pet 宠物

\*/

int update(Pet pet);

/\*\*

\* 获取指定昵称的宠物,精确查询。

\* @param name 昵称

\* @return 宠物

\*/

Pet getByName(String name);

/\*\*

\* 获取指定昵称的宠物列表,模糊查询。

\* @param name 昵称

\* @return 宠物列表

\*/

List<Pet> findByName(String name);

/\*\*

\* 获取指定类型的宠物列表。

\* @param type 宠物类型

\* @return 宠物列表

\*/

List<Pet> findByType(String type);

}

public class PetDaoMySQLImpl extends BaseDao implements PetDao{

public int update(Pet pet) {

String sql="update pet set status=0 where id=?";

Object[] param={pet.getId()};

int result=this.exceuteUpdate(sql, param);

return result;

}

@Override

public List<Pet> findByName(String name) {

// TODO Auto-generated method stub

return null;

}

@Override

public Pet getByName(String name) {

// TODO Auto-generated method stub

return null;

}

@Override

public List<Pet> findByType(String type) {

// TODO Auto-generated method stub

return null;

}

@Override

public int save(Pet pet) {

String sql="insert pet(master\_id,name,type\_id) values(?,?,?)";

Object[]param={pet.getMasterId(),pet.getName(),pet.getTypeId()};

return exceuteUpdate(sql, param);

}

@Override

public int del(Pet pet) {

// TODO Auto-generated method stub

return 0;

}

}

## **使用DBCP来获取数据连接**

* 需求

使用dbcp连接池来获取数据连接

* 参考答案

#<!-- 连接设置 -->

driverClassName=com.mysql.jdbc.Driver

url=jdbc:mysql://localhost:3306/mydb

username=root

password=root

#<!-- 初始化连接 -->

initialSize=10

#<!-- 最大连接数量 -->

maxActive=50

#<!-- 最大空闲连接 -->

maxIdle=20

#<!-- 最小空闲连接 -->

minIdle=5

#<!-- 超时等待时间(单位毫秒) -->

maxWait=50000

#<!-- 编码方式 -->

connectionProperties=useUnicode=true;characterEncoding=utf8

##<!-- 指定由连接池所创建的连接自动提交 -->

defaultAutoCommit=true

#<!-- 指定由连接池所创建的连接的事务级别 -->

defaultTransactionIsolation=REPEATABLE\_READ

import java.sql.Connection;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

import java.util.Properties;

import javax.sql.DataSource;

import org.apache.commons.dbcp.BasicDataSourceFactory;

public class DBCPUtil {

private static DataSource dataSource = null;

//创建数据库连接池

static{

Properties properties = new Properties();

try {

ClassLoader classLoader=DBCPUtil.class.getClassLoader();

properties.load(classLoader.getResourceAsStream("dbcpconfig.properties"));

dataSource = BasicDataSourceFactory.createDataSource(properties);

} catch (Exception e) {

throw new ExceptionInInitializerError("DBCP始化错误，请检查配置文件");

}

}

//创建连接

public static Connection getConnection(){

try {

return dataSource.getConnection();

} catch (SQLException e) {

throw new RuntimeException("数据库连接错误");

}

}

//释放连接

public static void releaseConnection(Connection connection, Statement statement, ResultSet resultSet) {

if (resultSet != null) {

try {

resultSet.close();

} catch (Exception e) {

e.printStackTrace();

}

resultSet = null;

}

if (statement != null) {

try {

statement.close();

} catch (Exception e) {

e.printStackTrace();

}

statement = null;

}

if (connection != null) {

try {

connection.close();

} catch (Exception e) {

e.printStackTrace();

}

connection = null;

}

}

}

## **使用Druid来连接数据库**

* 需求

通过Druid连接池来连接数据库查询

* 参考答案

#MySQL驱动

druid.driverClassName=com.mysql.jdbc.Driver

#MySQL连接地址

druid.url=jdbc\:mysql\://192.168.0.8\:3306/test

#数据库用户

druid.username=moster

#该用户密码

druid.password=shzygjrmdwg

#自动提交

druid.defaultAutoCommit=false

#只读设置

druid.defaultReadOnly=false

druiddruid.defaultTransactionIsolation=

druid.defaultCatalog=

#连接池最大使用连接数量

druid.maxActive=25

#连接池最大空闲

druid.maxIdle=25

#连接池最小空闲

druid.minIdle=1

#初始化连接池大小

druid.initialSize=1

#连接等待时间，默认：-1

druid.maxWait=3000

druiddruid.testOnBorrow=

druid.testOnReturn=

#检测间隔，检测需要关闭的空闲连接，单位：毫秒

druiddruid.timeBetweenEvictionRunsMillis=

druid.numTestsPerEvictionRun=

#一个连接在池中最小生存的时间，单位：毫秒

druiddruid.minEvictableIdleTimeMillis=

druid.testWhileIdle=

#测试SQL

druid.validationQuery=

#测试超时时间

druid.validationQueryTimeout=

#初始化连接SQL

druiddruid.initConnectionSqls=

druid.accessToUnderlyingConnectionAllowed=

#移除被废弃连接

druid.removeAbandoned=

#超时时间

druid.removeAbandonedTimeout=

#日志记录

druid.logAbandoned=

#PreparedStatements

druid.poolPreparedStatements=

#PreparedStatement最大数量

druid.maxOpenPreparedStatements=

#属性配置文件

druid.connectionProperties=

#filters配置

druid.filters=

#Exception处理

druid.exceptionSorter=

#Exception处理类名

druid.exception-sorter-class-name=

#初始化

druid.init=

import javax.sql.DataSource;

import java.io.IOException;

import java.sql.Connection;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

import java.util.Properties;

/\*\*

\* Druid连接池的工具类

\*/

public class DruidUtil {

//1、定义成员变量DataSource

private static DataSource ds;

static {

try {

//1、加载配置文件

Properties pro =new Properties();

pro.load(DruidUtil.class.getClassLoader().getResourceAsStream("druid.properties"));

//2、获取DataSource

ds= DruidDataSourceFactory.createDataSource(pro);

} catch (IOException e) {

e.printStackTrace();

} catch (Exception e) {

e.printStackTrace();

}

}

/\*\*

\* 获取连接

\*/

public static Connection getConnection() throws SQLException {

return ds.getConnection();

}

/\*\*

\* 释放资源

\*/

public static void close(Statement stmt,Connection conn){

close(null, stmt, conn);

}

public static void close(ResultSet rs,Statement stmt,Connection conn){

if (rs !=null){

try {

rs.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

if (stmt !=null){

try {

stmt.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

if (conn !=null){

try {

conn.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

}

/\*\*

\* 获取连接池方法

\*/

public static DataSource getDataSource(){

return ds;

}

}

## 