**1.QueryRunner的使用**：

QueryRunner简化了jdbc的基本增删改查操作。

**示例：**

增加操作：

**public** **boolean** doCreate(Emp vo) **throws** Exception {

**boolean** flag = **false**;

**QueryRunner runner = new QueryRunner();** //带有事务回滚

String sql = "insert into TblEmp(empid,ename,egendar,deptid) values(?,?,?,?)";

**Object params[] = {vo.getEmpid(),vo.getEname(),vo.getEgendar(),vo.getDeptid()};** //设置sql语句的参数

**if**(**runner.update(conn, sql, params)** > 0) { //执行插入操作

flag = **true**;

}

**return** flag;

}

在以上的操作中，参数里带有conn的是指具有事务回滚的操作。当QueryRunner runner = **new** QueryRunner(); 中没有参数时，代表具有**事务回滚**。

**QueryRunner runner = new QueryRunner(DataSource dataSource); //代表不具有事务回滚**

**Conn:数据库连接对象**

**Sql：sql语句**

**Params:sql语句中需要的传入的参数**

删除操作：

**public** **boolean** doRemove(Integer id) **throws** Exception {

**boolean** flag = **false**;

**QueryRunner runner = new QueryRunner(); //带有事务回滚**

String sql = "delete TblEmp where empid = ?";

**if**(**runner.update(conn, sql,id)** > 0) { //执行删除成功，id为需要传入的查询参数

flag = **true**;

}

**return** flag;

}

修改操作：

**public** **boolean** doUpdate(Emp vo) **throws** Exception {

**boolean** flag = **false**;

**QueryRunner runner = new QueryRunner(); //带有事务回滚操作**

String sql = "update TblEmp set ename = ?,egendar = ?,deptid = ? where empid = ?";

Object params[] = {vo.getEname(),vo.getEgendar(),vo.getDeptid(),vo.getEmpid()};

**if**(r**unner.update(conn, sql, params)** > 0) { //执行更新

flag = **true**;

}

**return** flag;

}

查询操作：

**public** List<Emp> findAll(String keyword) **throws** Exception { //查找全部

//BeanListHandler：将结果集中的每一行数据都封装到一个对应的JavaBean实例中，存放到List里。

**QueryRunner runner = new QueryRunner(JDBCUtils.*getDataSource*()); //创建一个queryRuner不具有事务回滚**

String sql = "select empid,ename,egendar,deptid from TblEmp order by empid";

**List<Emp> list = runner.query(sql, new BeanListHandler<>(Emp.class));** **//执行查询操作，并且将查询结果保存到list集合**

**return** list;

}

**public** Emp findById(Integer id) **throws** Exception {

Emp emp = **null**;

**QueryRunner runner = new QueryRunner(JDBCUtils.*getDataSource*());** //不进行事务处理

String sql = "select empid,ename,egendar,deptid from TblEmp where empid=?";

**emp = runner.query(sql, new BeanHandler<>(Emp.class), id);** //设置查询内容，查询语句，参数 **将查询到的数据封装为一个bean**

**return** emp;

}

在以上程序中，BeanHandler<>(class type)代表将查询到的数据封装为一个Javabean，BeanListHandler<>(Class type)表示将查询到的数据，每一行封装为一个Javabean，并且存放到list集合中。

需要导入的jar包：commons-dbutils-1.6.jar

**2.BeanUtils（自动将页面提交的内容按名字封装为一个bean）**

BeanUtils是一个简化jdbc操作的组件。

需要导入的jar包：commons-dbutils-1.6.jar和commons-logging-1.2.jar

可以使用BeanUtils.populate(要封装的bean, **request.getParameterMap()**);将页面提交过来的内容按照名字封装为一个javabean。

**3.c3p0数据库连接池**

需要导入的jar包：c3p0-0.9.1.2.jar和standard.jar

并且需要在src目录下创建一个名字为：c3p0-config.xml的c3p0配置文件，进行数据库连接池的相关配置。

示例：

package com.wp.dao.dbc;

import java.sql.Connection;

import java.sql.SQLException;

import java.util.List;

import javax.sql.DataSource;

import org.apache.commons.dbutils.QueryRunner;

import org.apache.commons.dbutils.handlers.BeanListHandler;

import org.junit.Test;

import com.mchange.v2.c3p0.ComboPooledDataSource;

import com.wp.vo.Emp;

public class JDBCUtils {

public static DataSource dataSource = null;

static {

dataSource = new ComboPooledDataSource("helloC3P0"); //从配置文件中获取数据源

}

public static DataSource getDataSource() {

return dataSource; //返回数据源

}

public static Connection getConnection() { //返回数据库连接

Connection conn = null;

try {

conn = dataSource.getConnection();

} catch (SQLException e) {

e.printStackTrace();

}

return conn;

}

public static void close(Connection conn) { //关闭数据库连接

if(conn != null) {

try {

conn.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

}

/\* public static void main(String[] args) throws SQLException {

System.out.println(dataSource.getConnection());

}\*/

}

配置文件：

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<c3p0-config>

<named-config name=*"helloC3P0"*>

<!-- 指定连接数据源的基本属性 oracle -->

<property name=*"jdbcUrl"*>jdbc:oracle:thin:@localhost:1522:ORCL</property>

<property name=*"driverClass"*>oracle.jdbc.driver.OracleDriver</property>

<property name=*"user"*>scott</property>

<property name=*"password"*>tiger</property>

　　 <!--当连接池中的连接耗尽的时候c3p0一次同时获取的连接数。即当连接池中可用连接数不够时，向数据库服务器申请连接Default: 3 -->

<property name=*"acquireIncrement"*>3</property>

　　 <!-- 初始化数据库连接池时连接的数量 -->

<property name=*"initialPoolSize"*>10</property>

<!-- 数据库连接池中的最小的数据库连接数 -->

<property name=*"minPoolSize"*>2</property>

<!-- 数据库连接池中的最大的数据库连接数 -->

<property name=*"maxPoolSize"*>10</property>

</named-config>

<default-config>

<!-- 指定连接数据源的基本属性 -->

<property name=*"jdbcUrl"*>jdbc:oracle:thin:@localhost:1522:ORCL</property>

<property name=*"driverClass"*>oracle.jdbc.driver.OracleDriver</property>

<property name=*"user"*>scott</property>

<property name=*"password"*>tiger</property>

</default-config>

</c3p0-config>

4.