/\*

\* Licence

\*

\* CopyRight 2008-2009 by The siox software project Lab.

\*

\* @author Lindily 2009-10-14 21:09:05

\*

\*/

package cn.siox.db;

import cn.siox.util.LogManager;

import com.mchange.v2.c3p0.ComboPooledDataSource;

import java.beans.PropertyVetoException;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

import java.sql.Connection;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

import java.util.List;

import java.util.Properties;

import java.util.logging.Level;

import java.util.logging.Logger;

import org.apache.commons.dbutils.QueryRunner;

import org.apache.commons.dbutils.ResultSetHandler;

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

/\*\*

\* @类名 QueryDB

\* @作者 Lindily

\* @日期 2009-10-14 21:09:05

\*/

public class QueryDB {

ComboPooledDataSource pool = new ComboPooledDataSource();

QueryRunner query = new QueryRunner();

Properties prop = new Properties();

Connection conn = null;

Statement stmt = null;

ResultSet rs = null;

public Connection link(String dbIP, String port, String database, String user, String password, Connection conn) {

try {

pool.setDriverClass("com.mysql.jdbc.Driver");

pool.setUser(user);

pool.setPassword(password);

pool.setJdbcUrl("jdbc:mysql://localhost" + ":" + port + "/" + database + "?characterEncoding=utf-8");

pool.setMaxPoolSize(Integer.parseInt(prop.getProperty("defaultPoolSize")));

pool.setMaxStatements(Integer.parseInt(prop.getProperty("defaultStatmentPool")));

conn = pool.getConnection();

return conn;

} catch (SQLException ex) {

LogManager.getLogger(this.getClass()).log(Level.SEVERE, null, ex);

return null;

} catch (PropertyVetoException ex) {

LogManager.getLogger(this.getClass()).log(Level.SEVERE, null, ex);

return null;

}

}

public Connection link(File file) {

try {

prop.load(new BufferedReader(new FileReader(file)));

pool.setDriverClass(prop.getProperty("driver"));

pool.setUser(prop.getProperty("user"));

pool.setPassword(prop.getProperty("password"));

pool.setJdbcUrl("jdbc:mysql://localhost" + ":" + prop.getProperty("port")

+ "/" + prop.getProperty("database") + "?characterEncoding=utf-8");

pool.setMaxPoolSize(Integer.parseInt(prop.getProperty("defaultPoolSize")));

pool.setMaxStatements(Integer.parseInt(prop.getProperty("defaultStatmentPool")));

conn = pool.getConnection();

return conn;

} catch (FileNotFoundException ex) {

LogManager.getLogger(this.getClass()).log(Level.SEVERE, null, ex);

return null;

} catch (SQLException ex) {

LogManager.getLogger(this.getClass()).log(Level.SEVERE, null, ex);

return null;

} catch (PropertyVetoException ex) {

LogManager.getLogger(this.getClass()).log(Level.SEVERE, null, ex);

return null;

} catch (IOException ex) {

LogManager.getLogger(this.getClass()).log(Level.SEVERE, null, ex);

return null;

}

}

public boolean isLinking(){

if(conn==null){

return false;

}else{

return true;

}

}

public int insert(Address addr){

int n=0;

try {

n = query.update(conn, "INSERT INTO address (name,company,department,job,province,city,post,street,"

+ "email,business1,business2,home1,home2,mobile1,mobile2,otherphone,fax,bak) "

+ "VALUES (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)",

addr.getName(), addr.getCompany(), addr.getDepartment(), addr.getJob(), addr.getProvince(),

addr.getCity(), addr.getPost(), addr.getStreet(), addr.getEmail(), addr.getBusiness1(),

addr.getBusiness2(), addr.getHome1(), addr.getHome2(), addr.getMobile1(), addr.getMobile2(),

addr.getOtherphone(), addr.getFax(), addr.getBak());

} catch (SQLException ex) {

LogManager.getLogger(this.getClass()).log(Level.SEVERE, null, ex);

}

return n;

}

public int update(Address addr){

int n=0;

try {

n = query.update(conn, "UPDATE address SET name=?,company=?,department=?,job=?,"

+ "province=?, city=?, street=?, post=?, business1=?, business2=?, home1=?,"

+ " home2=?, mobile1=?, mobile2=?, otherphone=?, fax=?, email=?, bak=?"

+ "WHERE entry=?",

addr.getName(), addr.getCompany(), addr.getDepartment(),

addr.getJob(), addr.getProvince(), addr.getCity(),addr.getStreet(),

addr.getPost(),addr.getBusiness1(),addr.getBusiness2(),

addr.getHome1(), addr.getHome2(),

addr.getMobile1(), addr.getMobile2(),addr.getOtherphone(),

addr.getFax(),addr.getEmail(),addr.getBak(),addr.getEntry());

} catch (SQLException ex) {

LogManager.getLogger(this.getClass()).log(Level.SEVERE, null, ex);

}

return n;

}

public Address query(Address addr){

Address address=null;

ResultSetHandler handler = new BeanListHandler(Address.class);

try {

List results = (List) query.query(conn, "select \* from address where name=? limit 1",handler,addr.getName());

for (int i = 0; i < results.size(); i++) {

address = (Address) results.get(i);

}

return address;

} catch (SQLException ex) {

LogManager.getLogger(this.getClass()).log(Level.SEVERE, null,ex);

return null;

} catch (ClassCastException ex){

LogManager.getLogger(this.getClass()).log(Level.SEVERE, null,ex);

return null;

}

}

public int delete(Address addr){

int n=0;

try {

n=query.update(conn, "DELETE FROM address WHERE entry=?", addr.getEntry());

} catch (SQLException ex) {

LogManager.getLogger(this.getClass()).log(Level.SEVERE, null,ex);

}

return n;

}

public Address[] search(String str){

ResultSetHandler handler = new BeanListHandler(Address.class);

str="%"+str+"%";

try {

//多写一些OR查询

List results = (List) query.query(conn, "select \* from address where name LIKE ? "

+ "OR company LIKE ? "

+ "OR department LIKE ? "

+ "OR job LIKE ? "

+ "OR province LIKE ? "

+ "OR city LIKE ? "

+ "OR street LIKE ? "

+ "OR post LIKE ? "

+ "OR business1 LIKE ? "

+ "OR business2 LIKE ? "

+ "OR home1 LIKE ? "

+ "OR home2 LIKE ? "

+ "OR mobile1 LIKE ? "

+ "OR mobile2 LIKE ? "

+ "OR otherphone LIKE ? "

+ "OR fax LIKE ? "

+ "OR email LIKE ? "

+ "OR bak LIKE ? "

,handler,str,str,str,str,str,str,str,str,str,str,str,str,str,str,str,str,str,str);

Address[] address = new Address[results.size()];

for (int i = 0; i < results.size(); i++) {

address[i] = (Address) results.get(i);

}

return address;

} catch (SQLException ex) {

LogManager.getLogger(this.getClass()).log(Level.SEVERE, null,ex);

return null;

} catch (ClassCastException ex){

LogManager.getLogger(this.getClass()).log(Level.SEVERE, null,ex);

return null;

}

}

}

//Address DB-OR

package cn.siox.db;

public class Address {

private java.lang.Integer entry;

private String name;

private java.lang.String company;

private java.lang.String department;

private String job;

private java.lang.String province;

private java.lang.String city;

private java.lang.String street;

private String post;

private String business1;

private String business2;

private String home1;

private String home2;

private String mobile1;

private String mobile2;

private String otherphone;

private String fax;

private java.lang.String email;

private java.lang.String bak;

public void setEntry(java.lang.Integer entry){

this.entry=entry;

}

public java.lang.Integer getEntry(){

return entry;

}

public void setName(String name){

this.name=name;

}

public String getName(){

return name;

}

public void setCompany(java.lang.String company){

this.company=company;

}

public java.lang.String getCompany(){

return company;

}

public void setDepartment(java.lang.String department){

this.department=department;

}

public java.lang.String getDepartment(){

return department;

}

public void setJob(String job){

this.job=job;

}

public String getJob(){

return job;

}

public void setProvince(java.lang.String province){

this.province=province;

}

public java.lang.String getProvince(){

return province;

}

public void setCity(java.lang.String city){

this.city=city;

}

public java.lang.String getCity(){

return city;

}

public void setStreet(java.lang.String street){

this.street=street;

}

public java.lang.String getStreet(){

return street;

}

public void setPost(String post){

this.post=post;

}

public String getPost(){

return post;

}

public void setBusiness1(String business1){

this.business1=business1;

}

public String getBusiness1(){

return business1;

}

public void setBusiness2(String business2){

this.business2=business2;

}

public String getBusiness2(){

return business2;

}

public void setHome1(String home1){

this.home1=home1;

}

public String getHome1(){

return home1;

}

public void setHome2(String home2){

this.home2=home2;

}

public String getHome2(){

return home2;

}

public void setMobile1(String mobile1){

this.mobile1=mobile1;

}

public String getMobile1(){

return mobile1;

}

public void setMobile2(String mobile2){

this.mobile2=mobile2;

}

public String getMobile2(){

return mobile2;

}

public void setOtherphone(String otherphone){

this.otherphone=otherphone;

}

public String getOtherphone(){

return otherphone;

}

public void setFax(String fax){

this.fax=fax;

}

public String getFax(){

return fax;

}

public void setEmail(java.lang.String email){

this.email=email;

}

public java.lang.String getEmail(){

return email;

}

public void setBak(java.lang.String bak){

this.bak=bak;

}

public java.lang.String getBak(){

return bak;

}

}

<c3p0-config>

<default-config>

<!--当连接池中的连接耗尽的时候c3p0一次同时获取的连接数。Default: 3 -->

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

<!--定义在从数据库获取新连接失败后重复尝试的次数。Default: 30 -->

<property name="acquireRetryAttempts">30</property>

<!--两次连接中间隔时间，单位毫秒。Default: 1000 -->

<property name="acquireRetryDelay">1000</property>

<!--连接关闭时默认将所有未提交的操作回滚。Default: false -->

<property name="autoCommitOnClose">false</property>

<!--c3p0将建一张名为Test的空表，并使用其自带的查询语句进行测试。如果定义了这个参数那么

属性preferredTestQuery将被忽略。你不能在这张Test表上进行任何操作，它将只供c3p0测试

使用。Default: null-->

<property name="automaticTestTable">Test</property>

<!--获取连接失败将会引起所有等待连接池来获取连接的线程抛出异常。但是数据源仍有效

保留，并在下次调用getConnection()的时候继续尝试获取连接。如果设为true，那么在尝试

获取连接失败后该数据源将申明已断开并永久关闭。Default: false-->

<property name="breakAfterAcquireFailure">false</property>

<!--当连接池用完时客户端调用getConnection()后等待获取新连接的时间，超时后将抛出

SQLException,如设为0则无限期等待。单位毫秒。Default: 0 -->

<property name="checkoutTimeout">100</property>

<!--通过实现ConnectionTester或QueryConnectionTester的类来测试连接。类名需制定全路径。

Default: com.mchange.v2.c3p0.impl.DefaultConnectionTester-->

<property name="connectionTesterClassName"></property>

<!--指定c3p0 libraries的路径，如果（通常都是这样）在本地即可获得那么无需设置，默认null即可

Default: null-->

<property name="factoryClassLocation">null</property>

<!--Strongly disrecommended. Setting this to true may lead to subtle and bizarre bugs.

（文档原文）作者强烈建议不使用的一个属性-->

<property name="forceIgnoreUnresolvedTransactions">false</property>

<!--每60秒检查所有连接池中的空闲连接。Default: 0 -->

<property name="idleConnectionTestPeriod">60</property>

<!--初始化时获取三个连接，取值应在minPoolSize与maxPoolSize之间。Default: 3 -->

<property name="initialPoolSize">3</property>

<!--最大空闲时间,60秒内未使用则连接被丢弃。若为0则永不丢弃。Default: 0 -->

<property name="maxIdleTime">60</property>

<!--连接池中保留的最大连接数。Default: 15 -->

<property name="maxPoolSize">15</property>

<!--JDBC的标准参数，用以控制数据源内加载的PreparedStatements数量。但由于预缓存的statements

属于单个connection而不是整个连接池。所以设置这个参数需要考虑到多方面的因素。

如果maxStatements与maxStatementsPerConnection均为0，则缓存被关闭。Default: 0-->

<property name="maxStatements">100</property>

<!--maxStatementsPerConnection定义了连接池内单个连接所拥有的最大缓存statements数。Default: 0 -->

<property name="maxStatementsPerConnection"></property>

<!--c3p0是异步操作的，缓慢的JDBC操作通过帮助进程完成。扩展这些操作可以有效的提升性能

通过多线程实现多个操作同时被执行。Default: 3-->

<property name="numHelperThreads">3</property>

<!--当用户调用getConnection()时使root用户成为去获取连接的用户。主要用于连接池连接非c3p0

的数据源时。Default: null-->

<property name="overrideDefaultUser">root</property>

<!--与overrideDefaultUser参数对应使用的一个参数。Default: null-->

<property name="overrideDefaultPassword">password</property>

<!--密码。Default: null-->

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

<!--定义所有连接测试都执行的测试语句。在使用连接测试的情况下这个一显著提高测试速度。注意：

测试的表必须在初始数据源的时候就存在。Default: null-->

<property name="preferredTestQuery">select id from test where id=1</property>

<!--用户修改系统配置参数执行前最多等待300秒。Default: 300 -->

<property name="propertyCycle">300</property>

<!--因性能消耗大请只在需要的时候使用它。如果设为true那么在每个connection提交的

时候都将校验其有效性。建议使用idleConnectionTestPeriod或automaticTestTable

等方法来提升连接测试的性能。Default: false -->

<property name="testConnectionOnCheckout">false</property>

<!--如果设为true那么在取得连接的同时将校验连接的有效性。Default: false -->

<property name="testConnectionOnCheckin">true</property>

<!--用户名。Default: null-->

<property name="user">root</property>

<!--早期的c3p0版本对JDBC接口采用动态反射代理。在早期版本用途广泛的情况下这个参数

允许用户恢复到动态反射代理以解决不稳定的故障。最新的非反射代理更快并且已经开始

广泛的被使用，所以这个参数未必有用。现在原先的动态反射与新的非反射代理同时受到

支持，但今后可能的版本可能不支持动态反射代理。Default: false-->

<property name="usesTraditionalReflectiveProxies">false</property>

<property name="automaticTestTable">con\_test</property>

<property name="checkoutTimeout">30000</property>

<property name="idleConnectionTestPeriod">30</property>

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

<property name="maxIdleTime">30</property>

<property name="maxPoolSize">25</property>

<property name="minPoolSize">10</property>

<property name="maxStatements">0</property>

<user-overrides user="swaldman">

</user-overrides>

</default-config>

<named-config name="dumbTestConfig">

<property name="maxStatements">200</property>

<user-overrides user="poop">

<property name="maxStatements">300</property>

</user-overrides>

</named-config>

</c3p0-config>