package com.gonsi.dao;  
import com.gonsi.util.MySqlUtil;  
  
import java.lang.reflect.Field;  
import java.lang.reflect.InvocationTargetException;  
import java.sql.\*;  
import java.time.LocalDate;  
import java.time.LocalDateTime;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Objects;  
  
public class BaseDao {  
 public <T> List<T> baseQuery(Class clazz, String sql, Object ... args){  
 //1初始化空的Array List  
 //2:连接数据库  
 //3：定义PreparedStatement和ResultSet  
 //4：进行预编译sql：preparedStatement = connection.prepareStatement(sql);  
 //5：for循环设置语句参数preparedStatement.setObject(i+1,args[i]);  
 //6：查询：resultSet = preparedStatement.executeQuery();  
 //7:获取与 resultSet 关联的 ResultSetMetaData 对象，并将其引用赋值给 metaData 变量。ResultSetMetaData metaData = resultSet.getMetaData();  
 */\*\*  
 \* 以下是一些常用的 ResultSetMetaData 方法：  
 \*  
 \* getColumnCount()：返回结果集中的列数。  
 \* getColumnName(int column)：返回指定列的列名。  
 \* getColumnType(int column)：返回指定列的SQL类型。  
 \* getColumnTypeName(int column)：返回指定列的数据库特定的类型名。  
 \* isNullable(int column)：指示指定列中的值是否可以为null。  
 \*/* //8:遍历结果集：while(re.next)  
 //Object obj = clazz.getDeclaredConstructor().newInstance();  
 */\*\*  
 \* clazz.getDeclaredConstructor().newInstance(); 这行代码用于通过反射API创建clazz类的一个新实例。  
 \* 这里，clazz是一个Class对象，代表某个类。  
 \* clazz是一个Class对象，代表某个类。  
 \* getDeclaredConstructor()方法用于获取这个类的一个Constructor对象，  
 \* 这个Constructor对象表示该类的一个无参数的构造方法。  
 \* 注意，getDeclaredConstructor()只能获取到声明在类中的构造方法，  
 \* 无论它是私有的、受保护的、包私有的还是公共的。如果该类没有无参数的构造方法，  
 \* 这个方法会抛出NoSuchMethodException。  
 \* .newInstance()  
 \* 这是Constructor类的一个方法，用于创建该构造方法所对应类的一个新实例。  
 \* 当你调用newInstance()方法时，它实际上会调用该构造方法来创建一个新的对象实例。  
 \* 注意，newInstance()方法在Java 9中已经被标记为过时（deprecated），取而代之的是使用Constructor的newInstance(Object... initargs)方法，即使你不传递任何参数，也应该使用这个方法。在Java 9及以后的版本中，你可以使用以下代码来创建实例：  
 \*/* //9:遍历列数：  
 // a:获取结果集中第i列的标签，并将其存储在columnName字符串变量中。  
 // b:使用 ResultSet 对象的 getObject(String columnName) 方法用于获取指定列名的列的值。  
 //  
 //  
 //  
 //  
 //  
 */\*\*  
 \** ***@renturn：返回对象的集和*** *\** ***@parpm*** *: Class clazz: 调用者会传入他们想要映射的实体类的 Class 对象作为 clazz 参数  
 \* Object ... args :可变参数  
 \*  
 \*/* //  
  
 ArrayList List = new ArrayList<>();  
  
 Connection connection = MySqlUtil.*getConnection*();  
  
 */\*\*  
 PreparedStatement 是 java.sql 包中的一个接口，  
 它代表一个预编译的SQL语句。使用 PreparedStatement 可以提高执行相同或相似SQL语句的性能，  
 因为它允许数据库预先编译SQL语句并重用编译后的计划。  
 此外，PreparedStatement 还可以用来防止SQL注入攻击，  
 因为它使用占位符（通常是 ?）来代表参数，而不是直接将参数拼接到SQL语句中。\*\*/* PreparedStatement preparedStatement = null;  
 ResultSet resultSet = null;  
 int rows = 0;  
 try{  
 //准备语句对象  
 preparedStatement = connection.prepareStatement(sql);  
 //设置语句参数  
 for (int i = 0; i < args.length; i++) {  
 preparedStatement.setObject(i+1,args[i]);  
 }  
 //查询  
 resultSet = preparedStatement.executeQuery();  
 //获取结果及的元数据，并获取列数量  
 */\*\*  
 \* ResultSetMetaData 是一个接口，它提供了关于 ResultSet 对象中列的信息  
 \*/* ResultSetMetaData metaData = resultSet.getMetaData();  
 int columnCount = metaData.getColumnCount();  
 //将结果集通过反射封装成实体类对象  
 while (resultSet.next()){  
 //getDeclaredConstructor()方法用于获取这个类的一个Constructor对象，这个Constructor对象表示该类的一个无参数的构造方法。  
 Object obj = clazz.getDeclaredConstructor().newInstance();  
 for (int i = 0; i < columnCount; i++) {  
 */\*\*  
 \* ResultSetMetaData接口的getColumnLabel(int column)方法  
 \* 用于获取结果集中指定列的标签（label）。通常，列的标签就是列的别名（如果设置了的话），  
 \* 或者在没有设置别名的情况下，它就是列的名字。  
 \* 获取结果集中第i列的标签，并将其存储在columnName字符串变量中。  
 \*/* String columnName = metaData.getColumnLabel(i);  
 */\*\*  
 \* ResultSet 对象的 getObject(String columnName) 方法用于获取指定列名的列的值。  
 \* 这个方法返回一个 Object 类型的值，表示该列的值。  
 \*/* Object value = resultSet.getObject(columnName);  
 if (value.getClass().equals(LocalDate.class)){  
 //检查一个对象的类型是否与你期望的特定类相匹配  
 //value对象的类是否严格等于LocalDate.clas  
 value = Timestamp.*valueOf*((LocalDateTime) value);  
 }  
 //getDeclaredField(String name)是Class类的一个方法，用于获取当前类声明的指定名称的字段（Field对象）。  
 Field field = clazz.getDeclaredField(columnName);  
 //// 使字段可访问  
 field.setAccessible(true);  
// 在Java反射API中，Field类的set(Object obj, Object value)方法用于设置指定对象上  
// 此Field对象表示的字段的值。  
// obj参数是你要设置字段值的对象实例，而value参数是你想要设置的新值。  
 field.set(obj,value);  
 }  
 */\*\*  
 \* 中，List.add((T)obj); 这行代码是在向一个List集合中添加一个元素。  
 \* 这里的(T)obj是一个类型转换操作，将obj对象转换为List集合声明的泛型类型T。  
 \* 这里的List应该是某个具体类型的List的实例，比如List<String>或List<Integer>，  
 \* T就是List集合中元素的类型。obj是你想要添加到List中的对象。  
 \* 类型转换(T)obj在编译时并不会执行任何转换，  
 \* 它仅仅是一个告诉编译器“我确信obj可以安全地转换为T类型”的声明。如果obj实际上不能转换为T类型，  
 \* 那么在运行时将会抛出一个ClassCastException。  
 \*/* List.add((T)obj);  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }finally {  
 if (null != resultSet){  
 try {  
 resultSet.close();  
 } catch (SQLException e) {  
 throw new RuntimeException(e);  
 }  
 }  
 if(null != preparedStatement){  
 try {  
 preparedStatement.close();  
 } catch (SQLException e) {  
 throw new RuntimeException(e);  
 }  
  
 }  
 //归还数据库连接  
 }  
 return List;  
 }  
  
  
  
  
  
  
}