Mô tả hệ thống

**KHOA CÔNG NGHỆ THÔNG TIN**

**ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

Nhóm 2 | Mẫu thiết kế hướng đối tượng

*Xây dựng database access management (DAM) framework*

Mục lục

[I. Sơ đồ lớp và ý nghĩa: 2](#_Toc486847311)

[1. Session Factory 2](#_Toc486847312)

[2. SQL Server 2](#_Toc486847313)

[3. MySQL 2](#_Toc486847314)

[4. Configuration 3](#_Toc486847315)

[5. Session 3](#_Toc486847316)

[6. Mapping 3](#_Toc486847317)

[7. Query Template 3](#_Toc486847318)

[8. Insert 3](#_Toc486847319)

[9. Update 3](#_Toc486847320)

[10. Delete 4](#_Toc486847321)

[11. SQL Query 4](#_Toc486847322)

[II. Danh sách các mẫu đã sử dụng: 4](#_Toc486847323)

[1. Mẫu singleton và mẫu abstract factory: 4](#_Toc486847324)

[2. Template Method: 7](#_Toc486847325)

# Sơ đồ lớp và ý nghĩa:

## Session Factory

* Đóng vai trò Abstract của các Factory (MySQL và SQLServer đã được cài đặt sẵn).
* Lớp giữ một đối tượng Connection String (conString) để có thể lấy được nội dung Connection String nhằm tạo kết nối đến cơ sở dữ liệu.

## SQL Server

* Factory để tạo các Session đối với cơ sở dữ liệu được lưu trên Microsoft SQL Server. Kế thừa từ lớp abstract SessionFactory.

## MySQL

* Factory để tạo các Session đối với cơ sở dữ liệu được lưu trên MySQL Server. Kế thừa từ lớp abstract SessionFactory.

## Configuration

* Đọc nội dung của Connection String từ một đường dẫn đến file config (được truyền vào) và lưu ở thuộc tính conString.

## Session

* Một Session có thể được hiểu là một kết nối đến cơ sở dữ liệu. Session sẽ được tạo thành từ SessionFactory cụ thể (SQLServer hoặc MySQL – đã cài đặt).
* Người dùng sẽ phải sử dụng session khi muốn thực hiện query đến cơ sở dữ liệu.

## Mapping

* Đây là lớp hỗ trợ Framework trong việc kiểm tra các thuộc tính khi thực hiện mapping giữa cơ sở dữ liệu và đối tượng.
* Lớp này sẽ kiểm tra tính đúng đắn của các annotation do người dùng định nghĩa so với các bảng trong cơ sỡ dữ liệu trong quá trình gọi tạo Session.

## Query Template

* Đây là lớp tổng quát thể hiện một câu query đến cơ sở dữ liệu. Đối với framework do nhóm thiết kế, mỗi câu query sẽ tương ứng với một object query (có nhiều loại khác nhau) và sẽ được chuyển thành câu SQL tương ứng và gọi thông qua JDBC (bằng connection từ lớp Session).
* Người dùng sẽ gọi lệnh executeQuery(params\_list) để thực hiện câu query.

## Insert

* Đây là lớp hiện thực hóa câu insert query đến cơ sở dữ liệu.
* Lớp sẽ cụ thể hóa 2 method: doTranslate() và MappingData() đối với câu insert.

## Update

* Đây là lớp hiện thực hóa câu update query đến cơ sở dữ liệu.
* Lớp sẽ cụ thể hóa 2 method: doTranslate() và MappingData() đối với câu update.

## Delete

* Đây là lớp hiện thực hóa câu update query đến cơ sở dữ liệu.
* Lớp sẽ cụ thể hóa 2 method: doTranslate() và MappingData() đối với câu update.

## SQL Query

* Đây là lớp hiện thực hóa câu insert query đến cơ sở dữ liệu.
* Lớp sẽ cụ thể hóa 2 method: doTranslate() và MappingData() đối với câu insert.
* Ngoài ra, do tính chất phức tạp của câu select, nhóm có chia nhỏ các thành phần của câu query ra thành các mệnh đề nhỏ.
* Inner Class SQLJoinClause dùng để quản lý các mệnh đề Join trong câu SQL query

# Danh sách các mẫu đã sử dụng:

## Mẫu singleton và mẫu abstract factory:

* Code thực hiên:

**public** **abstract** **class** SessionFactory {

**protected** String connectString;

//protected Connection con;

**protected** **abstract** Connection createConnection() **throws** SQLException;

**public** **abstract** Session createSession() **throws** SQLException ;

}

**public** **class** Session {

**private** Connection con;

**public** Session(Connection con){

**this**.con = con;

}

**public** Connection getConnection(){

**return** **this**.con;

}

**public** **void** closeSession() **throws** SQLException{

con.close();

}

}

**public** **class** SQLServer **extends** SessionFactory {

**private** Configuration config;

**private** **static** SQLServer *curinstance*;

**protected** SQLServer(String path) {

**this**.config = **new** Configuration();

**this**.connectString = config.getconString(path);

}

@Override

**public** Session createSession() **throws** SQLException {

// **TODO** Auto-generated method stub

Connection conn = **this**.createConnection();

Session s = **new** Session(conn);

**return** s;

}

@Override

**protected** Connection createConnection() **throws** SQLException {

// **TODO** Auto-generated method stub

Connection conn = DriverManager.*getConnection*(**this**.connectString);

**return** conn;

}

**public** **static** SQLServer Instance(String path) **throws** SQLException {

**if** (*curinstance* == **null**) {

*curinstance* = **new** SQLServer(path);

}

**return** *curinstance*;

}

}

**public** **class** MySQL **extends** SessionFactory {

**private** Configuration config;

**protected** **static** MySQL *curinstance*;

**public** MySQL(String path) {

**this**.config = **new** Configuration();

**this**.connectString = config.getconString(path);

}

@Override

**public** Session createSession() **throws** SQLException {

// **TODO** Auto-generated method stub

Connection conn = **this**.createConnection();

Session s = **new** Session(conn);

**return** s;

}

@Override

**protected** Connection createConnection() **throws** SQLException {

// **TODO** Auto-generated method stub

Connection conn = DriverManager.*getConnection*(**this**.connectString);

**return** conn;

}

**public** **static** MySQL Instance(String path) **throws** SQLException {

**if**(*curinstance* == **null**){

*curinstance* = **new** MySQL(path);

}

**return** *curinstance*;

}

}

SessionFactory f = SQLServer.*Instance*("D:/workspace2/testDAM/src/config.xml");

Session s = f.createSession();

* Ý nghĩa sử dụng:

Sử dụng mẫu abstract factory khởi tạo session và kiểm soát số lượng session tạo ra các kết nối với các hệ quản trị cơ sở dữ liệu. Xác định các loại session do người dùng lựa chọn trong quá trình runtime. Mẫu singleton giúp kiểm soát số lượng factory được tạo ra.

## Template Method:

* Code thực hiện:

**public** **abstract** **class** QueryTemplate {

**public** **boolean** executeQuery(Session s, Class<?> clazz, ArrayList<Class<?>> resList)

**throws** SQLException, IllegalAccessException, InvocationTargetException, NoSuchMethodException {

Connection conn = s.getConnection();

String queryString = **this**.doTranslate(conn);

System.***out***.println(queryString);

Statement stmt = conn.createStatement();

ResultSet rs = **null**;

// rs = stmt.executeQuery(queryString);

**boolean** res = stmt.execute(queryString);

rs = stmt.getResultSet();

**if** (rs != **null**) {

resList = mappingData(rs, clazz);

}

**if** (res == **true**) {

**return** **true**;

} **else** {

**return** **false**;

}

}

**public** **abstract** String doTranslate(Connection conn)

**throws** IllegalAccessException, InvocationTargetException, NoSuchMethodException, SQLException;

**public** **abstract** ArrayList<Class<?>> mappingData(ResultSet rs, Class<?> clazz);

}

**public** **class** Insert **extends** QueryTemplate {

**private** Object o;

**public** Insert(Object o) {

**this**.o = o;

}

@Override

**public** String doTranslate(Connection conn)

**throws** IllegalAccessException, InvocationTargetException, NoSuchMethodException, SQLException {

// **TODO** Auto-generated method stub

String resString = "insert into ";

Class<?> clazz = **this**.o.getClass();

Field[] fields = clazz.getDeclaredFields();

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery("SELECT TOP 2 \* FROM DEPARTMENT");

ResultSetMetaData rsmd = rs.getMetaData();

**int** numberOfColumns = rsmd.getColumnCount();

ArrayList<String> columnNamesDB = **new** ArrayList<String>();

**for** (**int** i = 1; i <= numberOfColumns; i++) {

columnNamesDB.add(rsmd.getColumnName(i));

}

Annotation tableAnnotation = clazz.getAnnotation(Table.**class**);

String tablename = ((Table) tableAnnotation).name();

resString += tablename + " values (";

PropertyDescriptor[] objDescriptors = PropertyUtils.*getPropertyDescriptors*(clazz);

**int** i = 1;

**for** (String columnName : columnNamesDB) {

**for** (PropertyDescriptor objDes : objDescriptors) {

String propertyName = objDes.getName();

Object propType = PropertyUtils.*getPropertyType*(**this**.o, propertyName);

Object propValue = PropertyUtils.*getProperty*(**this**.o, propertyName);

**if** (columnName.equals(propertyName) == **true**) {

**if** (i != (objDescriptors.length - 1)) {

**if** (propertyName.equals("class") == **false**) {

**if** (propType.toString().equals(String.**class**.toString()) == **true**) {

resString += "'" + propValue + "', ";

} **else** {

**if** (propType.toString().equals(Date.**class**.toString()) == **true**

|| propType.toString().equals(Time.**class**.toString()) == **true**) {

resString += "'" + propValue + "', ";

} **else** {

resString += propValue + ", ";

}

}

}

} **else** {

**if** (propertyName.equals("class") == **false**) {

**if** (propType.toString().equals(String.**class**.toString()) == **true**) {

resString += "'" + propValue + "')";

} **else** {

**if** (propType.toString().equals(Date.**class**.toString()) == **true**

|| propType.toString().equals(Time.**class**.toString()) == **true**) {

resString += "'" + propValue + "')";

} **else** {

resString += propValue + ")";

}

}

}

}

}

}

i++;

}

resString += ";";

**return** resString;

}

@Override

**public** ArrayList<Class<?>> mappingData(ResultSet rs, Class<?> clazz) {

// **TODO** Auto-generated method stub

System.***out***.println("Insert successfully");

**return** **null**;

}

}

* Ý nghĩa sử dụng:

Tận dụng được hàm execute query ở lớp cha QueryTemplate và tùy vào mỗi lớp con là insert hay delete hoặc update có thể cài đặt riêng và tái sử dụng được phần chung.

