Mục lục

[Mục lục 1](#_Toc24896931)

[Chương 1. Hibernate Framework 2](#_Toc24896932)

[1.1 ORM 2](#_Toc24896933)

[1.2 Persistence layer 2](#_Toc24896934)

[1.3 Hibernate 3](#_Toc24896935)

[1.4 Lợi ích của Hibernate framework 4](#_Toc24896936)

[1.5 Kiến trúc Hibernate 5](#_Toc24896937)

[1.6 Tại sao chọn Hibernate Framework thay vì dùng JDBC 19](#_Toc24896938)

[1.7 Các thao tác cơ bản khi làm việc với hibernate 21](#_Toc24896939)

# Hibernate Framework

## ORM

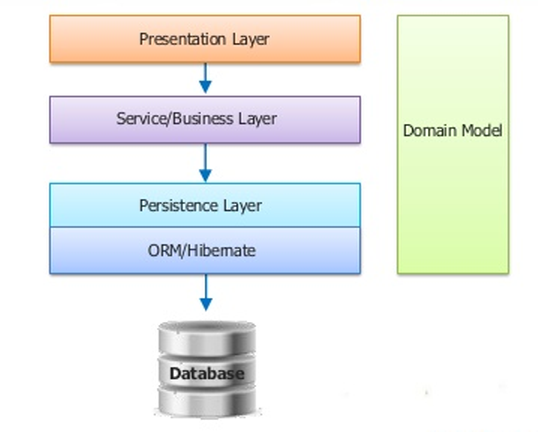
ORM (Object Relational Mapping) framework là một cơ chế cho phép người lập trình thao tác với database một cách hoàn toàn tự nhiên thông qua các đối tượng. Lập trình viên hoàn toàn không quan tâm đến loại database sử dụng SQL Server, MySQL, PostgreSQL, …

ORM giúp đơn giản hoá việc tạo ra dữ liệu, thao tác dữ liệu và truy cập dữ liệu. Đó là một kỹ thuật lập trình để ánh xạ đối tượng vào dữ liệu được lưu trữ trong cơ sở dữ liệu.



Hình ‑ Mô tả ORM

## Persistence layer



Hình ‑ Persistence layer

Một ứng dụng có thể được chia làm 3 phần như sau:

1. Giao diện người dùng (presentation layer).
2. Phần xử lý nghiệp vụ (business layer).
3. Phần chứa dữ liệu (data layer).

Cụ thể ra, business layer có thể được chia nhỏ thành hai layer con là business logic layer và persistence layer.

1. Business logic layer: các tính toán logic nhằm thỏa mãn yêu cầu người dùng.
2. Persistence layer: chịu trách nhiệm giao tiếp với data layer (thường là một hệ quản trị cơ sở dữ liệu quan hệ – Relational DBMS). Persistence sẽ đảm nhiệm các nhiệm vụ mở kết nối, truy xuất và lưu trữ dữ liệu vào các Relational DBMS.

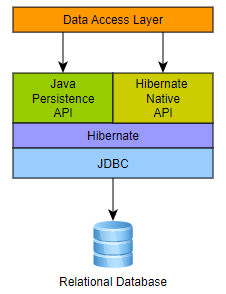
## Hibernate

Hibernate là một trong những ORM Framework. Hibernate framework là một framework cho persistence layer. Như vậy, nhờ có Hibernate framework mà giờ đây khi phát triển ứng dụng ta chỉ còn chú tâm vào những layer khác mà không phải bận tâm nhiều về persistence layer nữa.

Hibernate giúp lập trình viên viết ứng dụng Java có thể map các object (POJO) với hệ quản trị cơ sở dữ liệu quan hệ (database), và hỗ trợ thực hiện các khái niệm lập trình hướng đối tượng với cơ dữ liệu quan hệ.

Hibernate giúp lưu trữ và truy vấn dữ liệu quan hệ mạnh mẽ và nhanh chóng. Hibernate cho phép truy vấn dữ liệu thông qua Java Persistence API (JPA) hoặc bằng ngôn ngữ HQL (Mở rộng từ SQL của Hibernate) hoặc bằng SQL thuần (Native SQL).

Hibernate vốn là một thư viện sinh ra để làm việc với mọi loại database, nó không phụ thuộc vào việc ta chọn loại database nào. Nếu Java là “Viết 1 lần chạy mọi nơi” thì Hibernate là “Viết 1 lần chạy trên mọi loại database”.



Hình ‑ Hibernate

## Lợi ích của Hibernate framework

* Mã nguồn mở và nhẹ: Hibernate Framework là mã nguồn mở có giấy phép LGPL và nhẹ.
* Hiệu suất nhanh: Hiệu suất của Hibernate Framework là nhanh bởi vì bộ nhớ cache được sử dụng trong nội bộ Hibernate Framework. Có hai loại bộ nhớ cache trong Hibernate Framework, gồm bộ nhớ cache cấp một (first level cache) và bộ nhớ cache cấp hai (second level cache). Bộ nhớ cache cấp một được enable mặc định.
* Truy vấn cơ sở dữ liệu độc lập: HQL (Hibernate Query Language) là phiên bản hướng đối tượng của SQL. Nó tạo ra các truy vấn cơ sở dữ liệu độc lập. Vì vậy, ta không cần phải viết các truy vấn cơ sở dữ liệu cụ thể. Trước Hibernate, nếu dự án có cơ sở dữ liệu bị thay đổi, chúng ta cần phải thay đổi truy vấn SQL dẫn đến dễ gây lỗi.
* Tạo bảng tự động: Hibernate framework cung cấp phương tiện để tạo ra các bảng cơ sở dữ liệu một cách tự động. Vì vậy, không cần phải tốn công sức tạo ra các bảng trong cơ sở dữ liệu thủ công.
* Đơn giản lệnh join phức tạp: Có thể lấy dữ liệu từ nhiều bảng một cách dễ dàng với Hibernate framework.
* Cung cấp thống kê truy vấn và trạng thái cơ sở dữ liệu: Hibernate hỗ trợ bộ nhớ cache truy vấn và cung cấp số liệu thống kê về truy vấn và trạng thái cơ sở dữ liệu.
* Hibernate hỗ trợ hầu hết tất cả RDBMS chính, chẳng hạn: Oracle, Microsoft SQL Server, PostgreSQL, MySQL, …

## Kiến trúc Hibernate

Kiến trúc Hibernate bao gồm nhiều đối tượng như đối tượng persistent, session factory, transaction factory, connection factory, session, transaction, …



Hình ‑ Kiến trúc Hibernate

### Persistence object

Chính là các POJO object map với các table tương ứng của cơ sở dữ liệu quan hệ. Nó như là những container chứa dữ liệu từ ứng dụng để lưu xuống database, hay chứa dữ liệu tải lên ứng dụng từ database.

Một class được gọi là POJO khi nó có những thuộc tính sau:

* Tất cả các phương thức setter và getter phải là public.
* Tất cả các biến phải là private.
* Không nên mở rộng từ một lớp chỉ định.
* Không nên thực thi một giao diện chỉ định.
* Có thể không có hàm tạo có đối số.

Ví dụ, ta có 1 table trong cơ sở dữ liệu được lưu trữ như sau:

CREATE TABLE `customer` (

`id` int(11) NOT NULL,

`name` varchar(50) DEFAULT NULL,

`address` varchar(50) DEFAULT NULL

);

Như vậy, một POJO nên là:

**public** **class** Customer {

**private** **int** id;

**private** String name;

**private** String address;

**public** Customer() {

**super**();

}

**public** **int** getXX() {}

**public** **void** setXX() {}

}

### Configuration

Là đối tượng Hibernate đầu tiên ta tạo trong bất kỳ ứng dụng Hibernate nào và chỉ cần tạo một lần trong quá trình khởi tạo ứng dụng. Nó đại diện cho một tập tin cấu hình hoặc thuộc tính yêu cầu của Hibernate. Đối tượng Configuration cung cấp hai thành phần chính:

* Database Connection: Thao tác này được xử lý thông qua một hoặc nhiều tệp cấu hình được Hibernate hỗ trợ. Các tệp này là hibernate.properties và hibernate.cfg.xml.
* Class Mapping Setup: Thành phần này tạo ra kết nối giữa các lớp Java và các bảng cơ sở dữ liệu.

Ví dụ về hibernate.cfg.xml

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

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property name=*"hibernate.connection.driver\_class"*>com.mysql.jdbc.Driver</property>

<property name=*"hibernate.connection.url"*>jdbc:mysql://localhost:3306/dbname</property>

<property name=*"hibernate.connection.username"*>root</property>

<property name=*"hibernate.connection.password"*></property>

<property name=*"hibernate.connection.pool\_size"*>1</property>

<property name=*"hibernate.current\_session\_context\_class"*>thread</property>

<property name=*"hibernate.show\_sql"*>true</property>

<property name=*"hibernate.dialect"*>org.hibernate.dialect.MySQLDialect</property>

<mapping resource=*"name\_file\_mapping.hbm.xml"* />

<mapping class=*"package.class"* />

</session-factory>

</hibernate-configuration>

Trong đó các tag property sẽ chứa những thông tin cấu hình database mà chúng ta sẽ sử dụng, ở ví dụ trên là MySQL cùng với thông tin cấu hình hibernate.

Trong tag mapping sử dụng thuộc tính resource nếu chúng ta tạo file mapping bằng cấu trúc xml hoặc dùng thuộc tính class nếu chúng ta mapping bằng các Annotation của JPA.

Ví dụ về file mapping.

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

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD//EN"

"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

<class name=*"Customer"* table=*"customer"*>

<meta attribute=*"class-description"*>

description User

</meta>

<id name=*"id"* type=*"int"* column=*"id"*>

<generator class=*"native"*/>

</id>

<property name=*"name"* column=*"name"* type=*"string"*/>

<property name=*"address"* column=*"address"* type=*"string"*/>

</class>

</hibernate-mapping>

File nên được lưu là Customer.hbm.xml cùng với hibernate.cfg.xml đặt tại thư mục resources của ứng dụng.

Giải thích các thẻ trong file mapping:

* Tài liệu mapping là một tài liệu XML có là phần tử gốc chứa tất cả các phần tử <hibernate-mapping>.
* Các phần tử được sử dụng để định nghĩa ánh xạ cụ thể từ các lớp Java sang các bảng cơ sở dữ liệu. Tên lớp Java được chỉ định sử dụng thuộc tính name của phần tử lớp và tên bảng cơ sở dữ liệu được chỉ định sử dụng thuộc tính table.
* Phần tử là thành phần tùy chọn và có thể được sử dụng để tạo ra mô tả lớp.
* Phần tử ánh xạ thuộc tính ID duy nhất trong lớp tới khóa chính của bảng cơ sở dữ liệu. Thuộc tính name của id đề cập đến thuộc tính id trong lớp và thuộc tính column đề cập đến cột trong bảng cơ sở dữ liệu. Thuộc tính type giữ kiểu ánh xạ hibernate, các kiểu mapping này sẽ chuyển đổi từ kiểu dữ liệu Java sang SQL.
* Phần tử bên trong phần tử id được sử dụng để tự động tạo giá trị cho khóa chính. Thiết lập thuộc tính class của phần tử generator được đặt là native để cho phép hibernate chọn identity, sequence hoặc hilo để tạo khoá chính tùy thuộc vào khả năng của cơ sở dữ liệu.
* Phần tử được sử dụng để ánh xạ một thuộc tính của lớp Java vào một cột trong bảng cơ sở dữ liệu. Thuộc tính name của phần tử đề cập đến thuộc tính trong lớp và thuộc tính column đề cập đến cột trong bảng cơ sở dữ liệu. Thuộc tính type giữ kiểu ánh xạ hibernate, các kiểu mapping này sẽ chuyển đổi từ kiểu dữ liệu Java sang SQL.

Từ giải thích trên, chúng ta đã thấy cách hibernate sử dụng file mapping XML để chuyển đổi dữ liệu từ Java POJO sang các bảng cơ sở dữ liệu và ngược lại. Hibernate Annotations là cách mới nhất để định nghĩa các mapping mà không cần sử dụng file XML. Ta có thể sử dụng các annotation bổ sung hoặc để thay thế siêu dữ mapping XML.

Hibernate Annotations là cách mạnh mẽ để cung cấp siêu dữ liệu cho việc mapping đối tượng và bảng quan hệ. Tất cả các siêu dữ liệu được định nghĩa vào file java POJO cùng với mã này giúp người sử dụng đồng thời hiểu được cấu trúc bảng và POJO trong quá trình phát triển.

Tương ứng với file mapping XML trên, chúng ta có Class POJO khi định nghĩa bằng annotation như sau:

@Table(name = "customer")

@Entity

**public** **class** Customer {

@Id @GeneratedValue(strategy = GenerationType.***AUTO***)

@Column(name = "id") **private** **int** id;

@Column(name = "name") **private** String name;

@Column(name = "address") **private** String address;

**public** Customer() {

**super**();

}

// get and set

}

Giải thích về các annotation

* @Entity: Chúng ta đã sử dụng chú thích Entity vào lớp Customer đánh dấu lớp này như một Entity Bean, do đó nó phải có một constructor không có đối số mà có kiểu là public.
* @Table: Chú thích @Table cho phép ta chỉ định các chi tiết của bảng sẽ được sử dụng để lưu trữ thực thể trong cơ sở dữ liệu. Chú thích @Table cung cấp bốn thuộc tính, cho phép ta ghi đè tên của bảng, danh mục, lược đồ, bắt buộc các ràng buộc duy nhất trên các cột trong bảng. Trong lớp Customer ở trên chúng ta đang sử dụng tên bảng là customer.
* @Id và @ GeneratedValue: Mỗi entity bean sẽ có một khóa chính, mà ta chú thích trên lớp với chú thích @Id. Theo mặc định, chú thích @Id sẽ tự động xác định chiến lược tạo primary key, nhưng ta có thể ghi đè bằng cách áp dụng chú thích @GeneratedValue có hai tham số strategy và generator. Chúng ta chỉ sử dụng chiến lược tạo key mặc định. Cho phép Hibernate xác định loại kiểu tạo nào để sử dụng làm cho mã phù hợp với các cơ sở dữ liệu khác nhau.
* @Column: Chú thích @Column được sử dụng để chỉ định chi tiết của cột mà trường hoặc thuộc tính sẽ được ánh xạ.

### Session Factory

Là một interface giúp tạo ra session kết nối đến database bằng cách đọc các cấu hình trong Hibernate configuration.

SessionFactory là đối tượng nặng (heavy weight object) nên thường nó được tạo ra trong quá trình khởi động ứng dụng và lưu giữ để sử dụng sau này.

SessionFactory là một đối tượng luồng an toàn (Thread-safe) và được sử dụng bởi tất cả các luồng của một ứng dụng.

Mỗi một database phải có một session factory. Vì vậy, nếu ta đang sử dụng nhiều cơ sở dữ liệu thì sẽ phải tạo nhiều đối tượng SessionFactory. Giả sử ta sử dụng MySQL và Oracle cho ứng dụng Java của mình thì ta cần có một session factory cho MySQL, và một session factory cho Oracle.

Sau đây là ví dụ để tạo ra 1 đối tượng SessionFactory:

**public** **class** Application {

**private** **static** **final** SessionFactory ***SESSION\_FACTORY***;

**static** {

***SESSION\_FACTORY*** = **new** Configuration()

.configure()

.buildSessionFactory();

}

**public** **static** **void** main(String[] args) {

Session session = ***SESSION\_FACTORY***.openSession();

}

}

Hibernate sẽ tự động tìm file hibernate.cfg.xml trong thư mục resources của ứng dụng, nếu chúng ta đặt ở nơi khác, chúng ta phải khai báo đường dẫn như là một đối số String của phương thức configure(“path:…”).

### Hibernate Session

#### Tổng quan

Một session được sử dụng để có được một kết nối vật lý với một cơ sở dữ liệu. Đối tượng Session là nhẹ và được thiết kế để được tạo ra instance mỗi khi tương tác với cơ sở dữ liệu. Các đối tượng liên tục được lưu và truy xuất thông qua một đối tượng Session.

Các đối tượng Session không nên được mở trong một thời gian dài bởi vì chúng thường không phải là luồng an toàn (thread-unsafe) và chúng cần được tạo ra và được đóng khi cần thiết.

Mỗi một đối tượng session được Session factory tạo ra sẽ tạo một kết nối đến database.

Chức năng chính của session là cung cấp các thao tác create, update, read và delete cho các thể hiện các lớp thực thể được ánh xạ. Các thể hiện có thể tồn tại ở một trong ba trạng thái sau đây tại một thời điểm nhất định:

* transient: Một thể hiện mới của một lớp persistent không liên quan đến một Session, không có đại diện trong cơ sở dữ liệu và không có giá trị định danh được coi là transient (tạm thời) bởi Hibernate.
* persistent: Ta có thể tạo một thể hiện transient persistent bằng cách liên kết nó với một session. Một thể hiện persistent có một dại diện trong cơ sở dữ liệu, một giá trị định danh và được liên kết với một Session.
* detached: Khi chúng ta đóng session trong Hibernate, thể hiện persistent sẽ trở thành một thể hiện detached.

Chúng ta dùng phương thức openSession() của interface SessionFactory để tạo ra một session mới. Hoặc dùng phương thức getCurrentSession() để lấy session đang tồn tại. Một session khi chưa bị hủy bởi phương thức close() sẽ tồn tại trong bộ nhớ Cache cấp 2 của Hibernate.

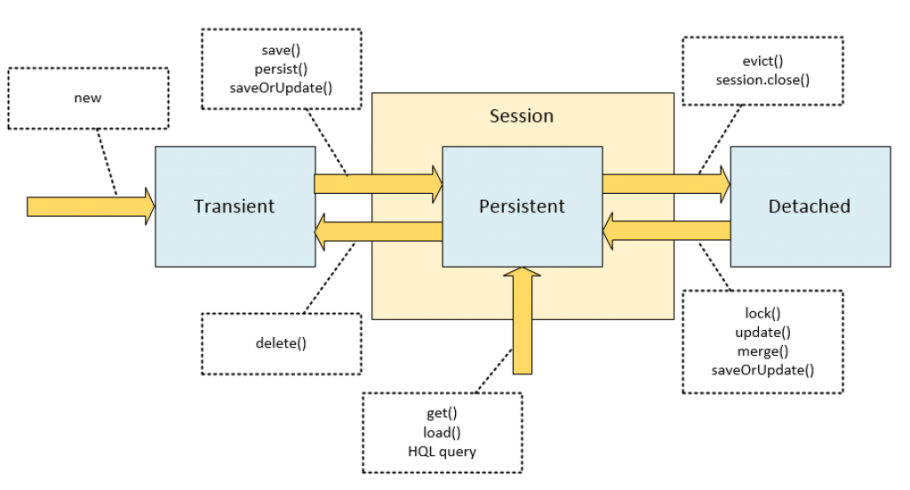
**public** **static** **void** main(String[] args) {

Session curSession = ***SESSION\_FACTORY***.getCurrentSession();

Session newSession = ***SESSION\_FACTORY***.openSession();

}

Dưới đây là sơ đồ mô tả một cách đơn giản về các phương thức bên trong session liên quan đến các trạng thái của thực thể như thế nào.



Hình ‑ Mối liên hệ giữa các trạng thái trong session

Trong mỗi session tồn tại một “Môi trường” chứa tất cả các đối tượng mà ta tạo ra và lưu vào cơ sở dữ liệu trong session đó, “Môi trường” đó được gọi là Persistentce Context là một cache cấp 1 được enable sẵn của Hibernate.

Trong Hibernate, Persistence Context được đại diện bởi đối tượng org.hibernate.Session. Với JPA, Persistence Context được thể hiện thông qua Class javax.persistence.EntityManager. JPA là bộ đặc tả giành cho việc lưu dữ liệu vào database giành cho ngôn ngữ Java, hibernate đã tuân thủ theo bộ đặc tả đó. Vì thế, Session interface cũng kế thừa từ EntityManager interface, do dó nếu sử dụng session chúng ta sẽ có thể sử dụng cả những phương thức được định nghĩa bởi EntityManager interface và những phương thức được mở rộng bởi Session interface của hibernate.

Nhìn vào dòng khai báo sau trong Session.class chúng ta sẽ thấy rõ điều trên:

**public** **interface** Session **extends** SharedSessionContract, EntityManager, ~~HibernateEntityManager~~, AutoCloseable, Closeable {…}

#### Các phương thức trong Session interface

Sau đây, chúng ta tìm hiểu về một vài phương thức được định nghĩa bởi Session interface mà chúng ta thường dùng để chuyển đổi trạng thái của các POJO:

*Phương thức persist():* Là phương thức của EntityManager mà Session kế thừa. Nó được dùng để chuyển trạng thái của 1 đối tượng từ transisent sang persistent.

Nếu một đối tượng đang ở trạng thái persistent được truyền vào thì sẽ không xảy ra chuyện gì, nhưng nếu một đối tượng đang ở trạng thái detached thì sẽ có exception xảy ra.

*Phương thức save():*Là nguyên bản của Hibernate không theo đặc tả của JPA. Khác với persist(), phương thức này sẽ sinh ra và trả về id của đối tượng. Khi một đối tượng ở trạng thái detached được truyền vào, nó sẽ được sinh ra id mới và không có exception xảy ra như persist().

*Phương thức merge():* Mục đích chính của phương thức này là để cập nhật một đối tượng đang ở trạng thái persistent với các giá trị được lấy từ một đối tượng ở trạng thái detached.

Phương thức này tạo ra và trả về một sự thể hiện được quản lý bởi Persistence Context, vì vậy đối tượng truyền vào chỉ đóng vai trò trung gian để lấy dữ liệu.

*Phương thức update():* Kiểu trả về của phương thức này là void, phương thức này biến đổi đối tượng truyền vào từ trạng thái detached sang trạng thái persistent (Điều mà bộ đặc tả JPA không cho phép).

Phương thức này sẽ bắn ra exception nếu như ta truyền cho nó một đối tượng thuộc trạng thái transisent.

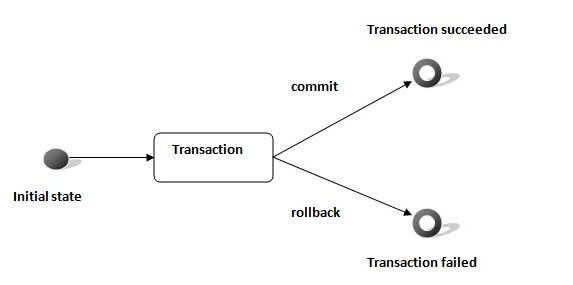
*Phương thức saveOrUpdate():* Phương thức này chỉ xuất hiện trong API của hibernate và không được chuẩn hóa trong bộ đặc tả JPA. Tương tự như update(), phương thức này có thể được dùng để chuyển đổi trạng thái của một đối tượng từ detached sang persistent, điểm khác biệt ở đây là nó không bắn ra exception khi ta truyền vào đối tượng transisent mà sẽ biến đổi đối tượng đó sang trạng thái persistent.

*Phương thức evict():* Phương thức này được dùng để biến đổi một đối tượng từ trạng thái persistent sang trạng thái detached.

### Transaction

Một Transaction đại diện cho một đơn vị làm việc với cơ sở dữ liệu và hầu hết các RDBMS hỗ trợ chức năng transaction. Các transaction trong Hibernate được xử lý bởi trình quản lý transaction và transaction (từ JDBC hoặc JTA).

Transaction đảm bảo tính ACID (Atomicity, Consistency, Isolation, Durability) của phiên làm việc với cơ sở dữ liệu.



Hình ‑ Transaction

Trong Hibernate, một transaction được trả về bởi phương thức getTransaction() hoặc beginTransaction(), sự khác nhau giữa hai phương này là beginTransaction() trả về một Transaction đã được begin còn getTransaction() thì không.

Các phương thức của Transaction interface được mô tả như sau:

* void begin(): Bắt đầu một transaction mới.
* void commit(): Kết thúc một đơn vị công việc.
* void rollback(): Buộc transaction quay lại trạng thái trước khi bắt đầu.
* void setTimeout(int seconds): Thiết lập timeout cho 1 transaction.
* boolean isAlive(): Trả về true nếu transaction đang hoạt động.
* boolean wasCommited(): Kiểm tra transaction đã được commit chưa.
* boolean wasRolledBack(): Kiểm tra nếu transaction đã được rollback chưa.

Sau đây là một ví dụ về quản lý Transaction trong Hibernate:

**public** Boolean addUser(Object object) {

Session session = **null**;

Transaction transaction = **null**;

**try** {

session = ***SESSION\_FACTORY***.openSession();

transaction = session.beginTransaction();

session.persist(object);

transaction.commit();

} **catch** (Exception e) {

transaction.rollback();

**return** **false**;

} **finally** {

session.close();

}

**return** **true**;

}

Thói quen tốt nên làm là đặt rollback cho transaction khi có một ngoại lệ xảy ra để đảm bảo tính nguyên tố (Atomicity) và tính nhất quán (Consistency) cho dữ liệu.

### Query

Các đối tượng Query sử dụng chuỗi truy vấn SQL (Native SQL) hoặc Hibernate Query Language (HQL) để lấy dữ liệu từ cơ sở dữ liệu và tạo các đối tượng.

HQL là một ngôn ngữ truy vấn hướng đối tượng, tương tự như SQL nhưng thay vì làm việc trên các bảng và cột, HQL làm việc với các đối tượng persistent và các thuộc tính của chúng. Các truy vấn HQL được dịch bởi Hibernate thành các truy vấn SQL thông thường, lần lượt thực hiện các công việc trên cơ sở dữ liệu.

String hql = "SELECT SUM(C.age), C.name "

+ "FORM Customer C "

+ "WHERE C.address=:address"

+ "GROUP BY C.name";

Query<Customer> query = session.createQuery(hql,Customer.**class**);

query.setParameter("address", "HCM");

List<Customer> result = query.getResultList();

result.stream().forEach(System.***out***::println);

Trong ví dụ trên, Customer không đại diện cho một table trong cơ sở dữ liệu mà đại diện cho lớp persistent.

Ngoài ra, chúng ta cũng có thể sử dụng native SQL trực tiếp qua Hibernate, nhưng cách này thường không khuyến khích được sử dụng. Đối tượng NaviteQuery được trả về qua câu lệnh createSQLQuery từ Session có cú pháp như sau.

@Override

NativeQuery createSQLQuery(String queryString);

### Criteria

Hibenate cung cấp Criteria API chứa các phương thức cho phép xây dựng một đối tượng truy vấn tiêu chuẩn theo yêu cầu của chương trình, nơi ta có thể áp dụng các quy tắc lọc và điều kiện hợp lý.

Session interface cung cấp phương thức createCriteria() để có thể tạo một đối tượng Criteria mà trả về các thể hiện của lớp đối tượng persistence khi ứng dụng của ta thi hành một truy vấn tiêu chuẩn.

Criteria criteria = session.~~createCriteria~~(Customer.**class**);

Từ phiên bản Hibernate 5.2 phương thức createCriteria() không còn được khuyến khích sử dụng nữa mà thay vào đó, để xây dựng một câu query chúng ta cần tới interface CriteriaBuilder, sau đây là một ví dụ:

CriteriaBuilder cBuilder = session.getCriteriaBuilder();

CriteriaQuery<Customer> cQuery = cBuilder

.createQuery(Customer.**class**);

Root<Customer> root = cQuery.from(Customer.**class**);

cQuery.select(root).where(cBuilder.equal(root.get("name"), "hoanghai"));

Query<Customer> query = session.createQuery(cQuery);

List<Customer> result = query.getResultList();

result.stream().forEach(System.***out***::println);

Như chúng ta cũng thấy, đối tượng Query được xây dựng dưới design pattern là Builder Patter nên mọi thao tác có thể viết liên tục thông qua toán tử “.”.

Sau đây để có thể hiểu hơn về Criteria API chúng ta sẽ có ví dụ cụ thể như sau.

Chúng ta có Class Item được định nghĩa như sau:

**public** **class** Item **implements** Serializable {

**private** Integer itemId;

**private** String itemName;

**private** String itemDescription;

**private** Integer itemPrice;

// set and get

}

Một cách đơn giản để lấy tất cả các hàng từ database sử dụng Criteria như sau:

CriteriaBuilder cb = session.getCriteriaBuilder();

CriteriaQuery<Item> cr = cb.createQuery(Item.**class**);

Root<Item> root = cr.from(Item.**class**);

cr.select(root);

Query<Item> query = session.createQuery(cr);

List<Item> results = query.getResultList();

CriteriaBuilder có thể được sử dụng để hạn chế kết quả truy vấn dựa trên các điều kiện cụ thể. Bằng cách sử dụng phương thức CriteriaQuery where() và cung cấp Biểu thức được tạo bởi CriteriaBuilder.

* *Lấy các Item có giá hơn 1000:*

cr.select(root).where(cb.gt(root.get("itemPrice"), 1000));

* *Tiếp theo, lấy các Item có giá nhỏ hơn 1000:*

cr.select(root).where(cb.lt(root.get("itemPrice"), 1000));

* *Lấy những Item có chứa chuỗi Chair trong tên:*

cr.select(root).where(cb.like(root.get("itemName"), "%chair%"));

* *Những Item có giá nằm trong khoảng 100 đến 200:*

cr.select(root).where(

cb.between(root.get("itemPrice"), 100, 200));

* *Những Item có thuộc tính Description là null:*

cr.select(root).where(cb.isNull(root.get("itemDescription")));

Các biểu thức được tạo ra như trên là 1 sự thể hiện của interface Predicate, chúng ta có thể dễ dàng quan sát điều đó qua khai báo của phương thức like trong CriteriaBuilder.class như sau:

/\*\*

\* Create a predicate for testing whether the expression

\* satisfies the given pattern.

\* **@param** x string expression

\* **@param** pattern string

\* **@return** like predicate

\*/

Predicate like(Expression<String> x, String pattern);

Vì vậy, ta có thể lưu trữ các biểu thức điều kiện trên như sau:

Predicate greaterThanPrice = cb.gt(root.get("itemPrice"), 1000);

Predicate chairItems = cb.like(root.get("itemName"), "Chair%");

Và sử dụng lại như sau:

cr.select(root).where(cb.or(greaterThanPrice, chairItems));

Qua những ví dụ trên, chúng ta đã biết cách sử dụng Criteria đơn giản, sau đây là các ví dụ về sắp xếp, các hàm tổng hợp (count, avg, sum, max, …), CriteriaUpdate cũng như CriteriaDelete.

* *Sắp xếp kết quả tăng dần theo tên và giảm dần theo giá:*

cr.orderBy(cb.asc(root.get("itemName")),

cb.desc(root.get("itemPrice")));

* *Tính trung bình giá của Item:*

CriteriaQuery<Double> cr = cb.createQuery(Double.**class**);

Root<Item> root = cr.from(Item.**class**);

cr.select(cb.avg(root.get("itemPrice")));

Query<Double> query = session.createQuery(cr);

List avgItemPriceList = query.getResultList();

* *CriteriaUpdate:*

CriteriaUpdate<Item> criteriaUpdate = cb

.createCriteriaUpdate(Item.**class**);

Root<Item> root = criteriaUpdate.from(Item.**class**);

criteriaUpdate.set("itemPrice", newPrice);

criteriaUpdate.where(cb.equal(root.get("itemPrice"), oldPrice));

Transaction transaction = session.beginTransaction();

session.createQuery(criteriaUpdate).executeUpdate();

transaction.commit();

* CriteriaDelete

CriteriaDelete<Item> criteriaDelete = cb

.createCriteriaDelete(Item.**class**);

Root<Item> root = criteriaDelete.from(Item.**class**);

criteriaDelete.where(

cb.greaterThan(root.get("itemPrice"), targetPrice));

Transaction transaction = session.beginTransaction();

session.createQuery(criteriaDelete).executeUpdate();

transaction.commit();

## Tại sao chọn Hibernate Framework thay vì dùng JDBC

### Minimize Code Changes

Khi ta thay đổi (thêm) cột vào bảng, với JDBC ta phải thay đổi:

* Thêm thuộc tính vào POJO class.
* Thay đổi method chứa câu truy vấn “select”, “insert”, “update” để bổ sung cột mới.
* Có thể có rất nhiều method, nhiều class chứa các câu truy vấn như trên.

Với Hibernate ta chỉ cần:

* Thêm thuộc tính vào Entity class.
* Cập nhật Hibernate Annotation để map column – property…

### Database Independent

Code sử dụng Hibernate là độc lập với hệ quản trị cơ sở dữ liệu, nghĩa là ta không cần thay đổi câu lệnh HQL khi ta chuyển từ hệ quản trị CSDL MySQL sang Oracle, hay các hệ quản trị CSDL khác… Do đó rất dễ để ta thay đổi CSDL quan hệ, đơn giản bằng cách thay đổi thông tin cấu hình hệ quản trị CSDL trong file cấu hình.

### Object Mapping

Với JDBC ta phải map các trường trong bảng với các thuộc tính của Java object một cách “thủ công”. Với Hibernate sẽ hỗ trợ ta map một cách “tự động” thông qua các file cấu hình map XML hay sử dụng các anotation.

### HQL

Hibernate cung cấp các câu lệnh truy vấn tương tự SQL, HQL của Hibernate hỗ trợ đầy đủ các truy vấn đa hình, HQL “hiểu” các khái niệm như kế thừa (inheritance), đa hình (polymorphysm), và liên kết (association).

### Lazy Loading

Với những ứng dụng Java làm việc với cơ sở dữ liệu lớn hàng trăm triệu bản ghi, việc có sử dụng Lazy loading trong truy xuất dữ liệu từ database mang lại lợi ích rất lớn.

### Loại bỏ Try-Catch Blocks

Sử dụng JDBC nếu lỗi xảy khi tao tác với database thì sẽ có exception SQLexception ném ra. Bởi vậy ta phải sử dụng try-catch block để xử lý ngoại lệ.

Hibernate xử lý việc này giúp chúng ta bằng cách nó override toàn bộ JDBC exception thành Unchecked Exception và ta không cần viết try-catch trong code của mình nữa.

### Hibernate Caching

Hibernate cung cấp một cơ chế bộ nhớ đệm, giúp giảm số lần truy cập vào database của ứng dụng càng nhiều càng tốt. Điều này sẽ có tác dụng tăng performance đáng kể cho ứng dụng.

### Associations

Việc tạo liên kết giữa các bảng bằng Hibernate như quan hệ một-một, một-nhiều, nhiều-một và nhiều-nhiều tỏ ra rất dễ dàng với cả cấu hình bằng xml và càng thuận tiện hơn với annotation.

### JPA Annotation Support

Hibernate implement đặc tả JPA, do đó chúng ta có thể sử dụng các Annotation của JPA như @Entity, @Table, @Column, … Nhờ đặc điểm này, chúng ta có thể dễ dàng chuyển đổi giữa các ORM Framework mà không cần phải sử đổi code.

### Connection Pooling

Một số Connection Pooling được hỗ trợ bởi Hibernate: C3p0, Apache DBCP.

Chúng ta có thể dễ dàng tích hợp với các thư viện Connection Pooling có sẵn chỉ với vài dòng cấu hình.

## Các thao tác cơ bản khi làm việc với hibernate

Phần này không có cấu trúc rõ ràng, ở một góc nhìn nào đó, nó giúp ta ôn lại những kiến thức đã được trình bày phía trên cũng như giới thiệu thêm những thao tác khác mà có thể trong quá trình làm việc với hibernate chúng ta sẽ cần tới.

Chúng ta cũng sẽ sử dụng ví dụ và giải thích để trình bày cho phần này.

### Các thư viện

Thông thường, khi làm việc với hibernate framework chúng ta sẽ cần tối thiểu 2 thư viện cho các thao tác cơ bản. Ngoài ra còn cần có 1 số thư viện khác như hỗ trợ cho kỹ thuật database connection pool như HikariCP, DBCP2, C3P0, BoneCP, Vibur… Mà trong các ví dụ về Spring Hibernate ở mục [3.5.6 (phần này ở tài liệu tổng hợp)] sẽ trình bày.

<!-- Chức năng ORM cốt lõi của hibernate -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>5.4.8.Final</version>

</dependency>

<dependency>

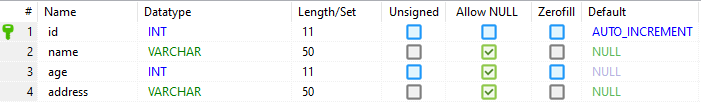
<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

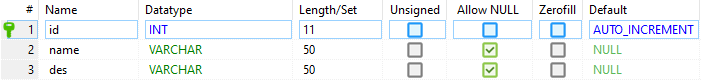
<version>8.0.18</version>

</dependency>

### Quan hệ nhiều-nhiều



Hình ‑ Table customer

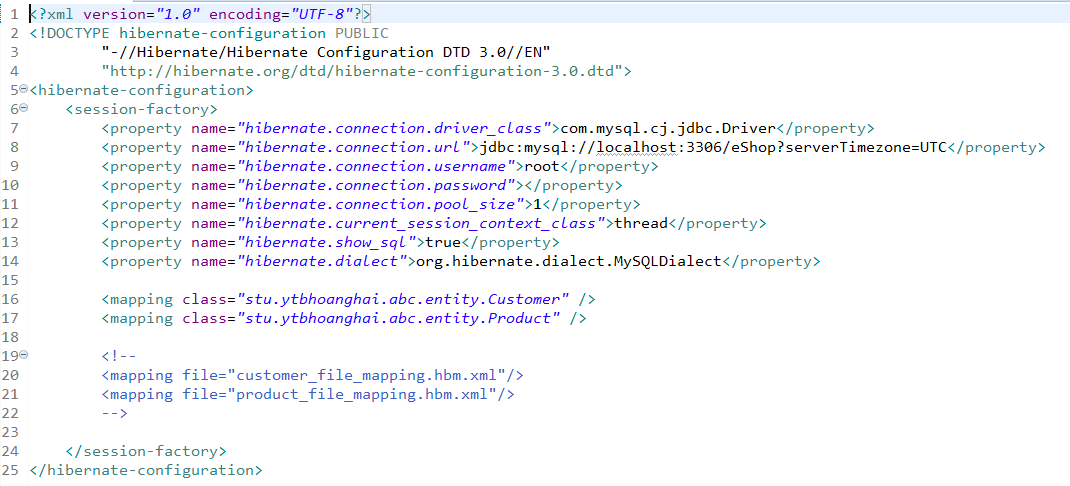


Hình ‑ Table product



Hình ‑ Table customer\_product

### File hibernate.cfg.xml



Hình ‑ Ví dụ hibernate.cfg.xml

<--Trong giai đoạn tạm dừng để phát triển phần khác 16/11/2019-->