**Giới thiệu**

Trong bài viết trước, tôi đã giới thiệu với các bạn về **Criteria API** của **Hibernate**.

1. [Hibernate là gì?](https://loda.me/hibernate-la-gi-loda1554623701594)
2. [Hướng dẫn sử dụng Criteria API trong Hibernate](https://loda.me/huong-dan-su-dung-criteria-api-trong-hibernate-loda1552815848300)

Trong phần này chúng ta sẽ tìm hiểu một số các phần cần thiết khác trong **Criteria API**, thứ giúp cho bạn xây dựng query một cách đơn giản hơn.

Trong bài có sử dụng:

1. [Lombok](https://loda.me/general-huong-dan-su-dung-lombok-giup-code-java-nhanh-hon-69-loda1552789752787)

**Cài đặt**

Nhớ thêm spring-boot-starter-data-jpa vào dependencies của bạn.

Trong phần này, tôi xài *H2 database* để demo. (H2 là dạng memory database, nó sẽ lưu trong ram và khi tắt chương trình nó sẽ mất sạch)

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

<projectxmlns="http://maven.apache.org/POM/4.0.0"xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"xsi:schemaLocation="http://maven.apache.org/POM/4.0.0https://maven.apache.org/xsd/maven-4.0.0.xsd"><modelVersion>4.0.0</modelVersion><parent><groupId>org.springframework.boot</groupId><artifactId>spring-boot-starter-parent</artifactId><version>2.0.5.RELEASE</version><relativePath/> <!-- lookup parent from repository -->

</parent><groupId>me.loda.spring</groupId><artifactId>example-independent-maven-spring-project</artifactId><version>0.0.1-SNAPSHOT</version><name>example-independent-maven-spring-project</name><description>Demo project for Spring Boot</description><properties><java.version>1.8</java.version></properties><dependencies><dependency><groupId>org.springframework.boot</groupId><artifactId>spring-boot-starter-web</artifactId></dependency><dependency><groupId>org.springframework.boot</groupId><artifactId>spring-boot-devtools</artifactId><scope>runtime</scope><optional>true</optional></dependency><dependency><groupId>org.projectlombok</groupId><artifactId>lombok</artifactId><optional>true</optional></dependency><dependency><groupId>org.springframework.boot</groupId><artifactId>spring-boot-starter-test</artifactId><scope>test</scope><!-- <exclusions>-->

<!-- <exclusion>-->

<!-- <groupId>org.junit.vintage</groupId>-->

<!-- <artifactId>junit-vintage-engine</artifactId>-->

<!-- </exclusion>-->

<!-- </exclusions>-->

</dependency><!--spring jpa-->

<dependency><groupId>org.springframework.boot</groupId><artifactId>spring-boot-starter-data-jpa</artifactId></dependency><!--in memory database-->

<dependency><groupId>com.h2database</groupId><artifactId>h2</artifactId><scope>runtime</scope></dependency><!--https://mvnrepository.com/artifact/org.hibernate/hibernate-jpamodelgen -->

<dependency><groupId>org.hibernate</groupId><artifactId>hibernate-jpamodelgen</artifactId><version>5.4.9.Final</version><scope>provided</scope></dependency></dependencies><build><plugins><plugin><groupId>org.springframework.boot</groupId><artifactId>spring-boot-maven-plugin</artifactId></plugin></plugins></build></project>

Đặc biệt, chú ý hibernate-jpamodelgen, tôi sẽ giải thích tác dụng của nó trước.

**JPA Meta Model**

trong [bài trước](https://loda.me/huong-dan-su-dung-criteria-api-trong-hibernate-loda1552815848300) mọi người cũng biết cách cấu trúc của một câu query trong Criteria API:

SELECT o FROM Office o WHERE o.city = 'hanoi'

Lúc này query của chúng ta sẽ như thế này:

query.select(root).where(builder.equal(root.get("city"), "hanoi"));

Để ý thì có thể thấy, khi muốn lấy column city để kiểm tra, chúng ta đang hardcode bằng String.

Có một số bất lợi khi làm vậy, thứ nhất là bạn phải tự nhớ tên các column mỗi khi gọi, thứ hai là bạn sẽ phải tìm kiếm tất cả các chỗ sử dụng mỗi sửa đổi tên cột.

Cách giải quyết hay nhất là tham chiếu tên các column của **Table** vào một **Object** để chúng ta có thể gọi tới mỗi khi sử dụng. Khi có sự thay đổi, chỉ cần thay đổi trong đối tượng này là xong. Đối tượng biểu diễn này được gọi là **Meta Model**.

Và rất may là Hibernate hỗ trợ chúng ta tự động generate ra các Meta Model từ các class @Entity

Ví dụ:

Chúng ta có Class Entity User

*User.java*

@Entity

@Data

@Builder

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private UserType type;

private String name;

public enum UserType {

NORMAL, VIP;

}

}

Thì class Meta Model của User sẽ tên là User\_ và có cấu trúc như sau:

@Generated(value = "org.hibernate.jpamodelgen.JPAMetaModelEntityProcessor")

@StaticMetamodel(User.class)

public abstract class User\_ {

public static volatile SingularAttribute<User, String> name;

public static volatile SingularAttribute<User, Long> id;

public static volatile SingularAttribute<User, UserType> type;

public static final String NAME = "name";

public static final String ID = "id";

public static final String TYPE = "type";

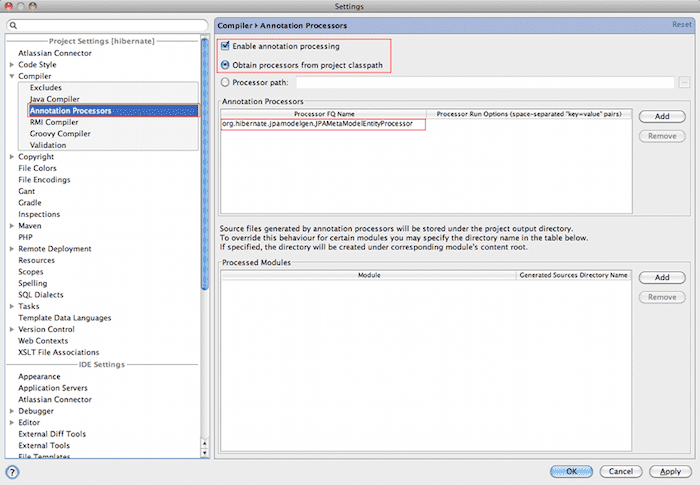
}

Để có thể Generate ra các class Meta Model, bạn sẽ cần thêm dependency hibernate-jpamodelgen vào project của mình.

<dependency><groupId>org.hibernate</groupId><artifactId>hibernate-jpamodelgen</artifactId><version>CURRENT-VERSION</version></dependency>

Khi build jar nó sẽ tự động generate thêm cho bạn

Để có thể sử dụng trong IDE, bạn có thể Config Annotation Processor (Giống với Lombok), để IDE hiểu và tự động generate ra giúp bạn lập trình dễ hơn.



**Predicate**

Để có thể xây dựng câu truy vấn một cách trọn vẹn, bạn cần biết Predicate.

Tạm hiểu một cách đơn giản thì Predicate là một mệnh đề điều kiện trong câu lệnh truy vấn.

Như ví dụ ở dưới đây:

@Repository

public class CustomUserRepository {

@PersistenceContext

private EntityManager em;

public User getUserById(Long id) {

CriteriaBuilder builder = em.getCriteriaBuilder();

CriteriaQuery<User> query = builder.createQuery(User.class);

Root<User> root = query.from(User.class);

Predicate condition = builder.equal(root.get(User\_.ID), id);

query.select(root).where(condition);

return em.createQuery(query).getSingleResult();

}

}

Predicate có thể liên kết với nhau bằng các phép quan hệ and, or, not, v.v..

Ví dụ:

public Collection<User> getUserByComplexConditions(String name, UserType type) {

CriteriaBuilder builder = em.getCriteriaBuilder();

CriteriaQuery<User> query = builder.createQuery(User.class);

Root<User> root = query.from(User.class);

Predicate hasNameLike = builder.like(root.get(User\_.NAME), name);

Predicate hasType = builder.equal(root.get(User\_.TYPE), type);

Predicate condition = builder.and(hasNameLike, hasType);

query.select(root).where(condition);

return em.createQuery(query).getResultList();

}

Tới đây bạn đã nắm được cơ bản **Criteria API** và đã có thể tự sử dụng nó trong đa số các câu lệnh đơn giản.

Trong bài tiếp theo, tôi sẽ hướng dẫn nốt các join bảng nữa thôi là bạn sẽ mas con ông ter.