Bài 7
Spring Data JPA Repository

Module: BOOTCAMP WEB-BACKEND DEVELOPMENT

Kiểm tra bài trước

Hỏi và trao đổi về các khó khăn gặp phải trong bài "JPA"

Tóm tắt lại các phần đã học từ bài "JPA"

Mục tiêu

- Trình bày được ý nghĩa Spring Data JPA Respository
- Định nghĩa được interface Repository
- Trình bày được CrudRepository
- Trình bày được PagingAndSortingRepository
- Định nghĩa được các phương thức truy vấn
- Khởi tạo được các thực thể Repository
- Tuỳ biến được các Spring Data Repository
- Triển khai được formatter tuỳ biến
- Phục vụ được các tài nguyên tĩnh

Thảo luận

Ý nghĩa Spring data JPA Repository
Interface Repository
Interface CrudRepository
Interface PagingAndSortingRepository

Spring Data Repository

- Spring Data Repository giúp giảm thiểu lượng code thông thường lặp đi lặp lại ở tầng truy xuất dữ liệu.
- Spring Data Repository định nghĩa một interface chính tên là Repository. Interface này nắm bắt entity cần quản lý và kiểu dữ liệu id của entity đó.
- Interface CrudRepository kế thừa từ interface Repository, chứa các phương thức thông dụng dành cho các thao tác CRUD.
- Interface PagingAndSortingRepository cung cấp các phương thức hỗ trợ cho việc phân trang và sắp xếp các entity.

Interface Repository

- Interface chính của Spring Data JPA là Repository

  public interface **Repository<T, ID>**

  - Trong đó:
    - T là kiểu dữ liệu của entity muốn quản lý
    - ID là kiểu dữ liệu của id của entity muốn quản lý
- Repository là một 'marker interface': chỉ sử dụng để đánh dấu/phân loại mà không khai báo các phương thức

Các interface kế thừa Repository

- CrudRepository<T,ID>
- PagingAndSortingRepository<T,ID>
- ReactiveCrudRepository<T,ID>
- ReactiveSortingRepository<T,ID>
- RevisionRepository<T,ID,N>
- RxJava2CrudRepository<T,ID>
- RxJava2SortingRepository<T,ID>

Interface PagingAndSortingRepository

- Hỗ trợ thực hiện các câu lệnh CRUD cơ bản

Interface PagingAndSortingRepository

```
public interface CrudRepository<T, ID extends Serializable> extends Repository<T, ID> {

  <S extends T> S save(S entity);

  Optional<T> findById(ID primaryKey);

  Iterable<T> findAll();

  long count();

  void delete(T entity);

  boolean existsById(ID primaryKey);

  // … more functionality omitted.
}
```

Lưu một entity

Tìm một entity theo Id

Lấy về tất cả các entity

Lấy về số lượng entity

Xoá một entity

Kiểm tra sự tồn tại của một entity theo Id

Interface PagingAndSortingRepository

- Kế thừa từ CrudRepository
- Bổ sung khả năng phân trang và sắp xếp

```
public interface PagingAndSortingRepository<T, ID extends Serializable> extends
CrudRepository<T, ID> {

   Iterable<T> findAll(Sort sort);

   Page<T> findAll(Pageable pageable);
}
```

Các interface tuỳ biến

- Có thể định nghĩa các interface để thực hiện các thao tác đặc thù
- Ví dụ:

```
interface UserRepository extends CrudRepository<User, Long> {
  long countByLastname(String lastname);
}
```

- Hoặc:

```
interface UserRepository extends CrudRepository<User, Long> {

  long deleteByLastname(String lastname);

  List<User> removeByLastname(String lastname);
}
```

Demo

Interface Repository

Interface CrudRepository

Interface PagingAndSortingRepository

Thảo luận

Phương thức truy vấn (Query method)

Phương thức truy vấn (Query method)

- Query method là những phương thức được khai báo trong repository interface có nhiệm vụ lấy thông tin từ cơ sở dữ liệu.
- Query method giúp cho việc lấy thông tin từ cơ sở dữ liệu mà không cần viết một câu query nào.
- Ví dụ: Lấy thông tin của đối tượng Customer

```
public interface CustomerRepository extends PagingAndSortingRepository<Customer,
Long> {}
public class CustomerServie implements ICustomerService
{
        @Autowired
        private CustomerRepository
        customerRepository; @Override
        public Customer findById(Long id) {
                return customerRepository.findOne(id);
        }
}
```

Các bước khai báo các phương thức truy vấn

1. Khai báo một interface kế thừa Repository hoặc các interface con của nó
2. Khai báo các phương thức truy vấn tuỳ biến
3. Cấu hình Spring để tạo các proxy instance cho các repository tuỳ biến
   - Có thể sử dụng XML hoặc Annotation
4. Sử dụng repository tuỳ biến thông qua cơ chế Injection

Khai báo các interface repository

- Kế thừa interface Repository hoặc các interface con
- Xác định kiểu của entity và kiểu của Id
- Ví dụ:

```
interface PersonRepository extends Repository<Person, Long> {
  …
}
```

- Có thể không kế thừa các interface repository có sẵn bằng cách sử dụng annotation @RepositoryDefinition

Khai báo các phương thức truy vấn

- Cơ chế hoạt động của repository store cho phép 2 hình thức để tạo ra các câu lệnh truy vấn:
  - Dựa vào tên của phương thức. Ví dụ:

```
List<Person> findByEmailAddressAndLastname(EmailAddress
emailAddress, String lastname);
```

  - Dựa vào câu lệnh truy vấn được khai báo cụ thể. Ví dụ:

```
@Query("select u from User u where
u.emailAddress = ?1") User
findByEmailAddress(String emailAddress);
```

Tạo các đối tượng repository

- Có thể sử dụng XML hoặc Java Configuration để cấu hình tạo các đối tượng repository
- Spring Data sẽ dò tìm và tạo ra các proxy tương ứng với từng repository để thao tác với dữ liệu
- Tên của bean được sinh ra dựa trên tên của repository. Chẳng hạn UserRepository sẽ có một bean là userRepository

Tạo các đối tượng repository: XML

- Ví dụ:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans
        xmlns:beans="http://www.springframework.org/schema/beans
        " xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns="http://www.springframework.org/schema/data/jpa"
        xsi:schemaLocation="http://www.springframework.org/schema
        /beans
    http://www.springframework.org/schema/beans/spring-
    beans.xsd
    http://www.springframework.org/schema/data/jpa
    http://www.springframework.org/schema/data/jpa/spring-
    jpa.xsd">
    <repositories base-package="com.acme.repositories" />
</beans:beans>
```

Tạo các đối tượng repository: Java Configration

- Ví dụ:

```
@Configuration
@EnableJpaRepositories("com.acme.repositories")
class ApplicationConfiguration {
  @Bean
  EntityManagerFactory entityManagerFactory() {
    // …
  }
}
```

Demo

Phương thức truy vấn (Query method)

Thảo luận

Khai báo các phương thức truy vấn

Tạo câu truy vấn

- Spring Data repository giúp cho việc tạo các câu truy vấn tự động
- Cơ chế này duyệt qua tên của phương thức với các tiền tố như find…By, read…By, query…By, count…By và get…By để xây dựng câu truy vấn
- Có thể sử dụng thêm một số từ khoá khác, chẳng hạn như Distinct, Asc, Desc, Or, And…

Ví dụ khai báo các phương thức truy vấn

```
interface PersonRepository extends Repository<User, Long> {
  List<Person> findByEmailAddressAndLastname(EmailAddress emailAddress, String lastname);
  // Enables the distinct flag for the query
  List<Person> findDistinctPeopleByLastnameOrFirstname(String lastname, String
  firstname); List<Person> findPeopleDistinctByLastnameOrFirstname(String lastname,
  String firstname);
  // Enabling ignoring case for an individual property
  List<Person> findByLastnameIgnoreCase(String lastname);
  // Enabling ignoring case for all suitable properties
  List<Person> findByLastnameAndFirstnameAllIgnoreCase(String lastname, String firstname);
  // Enabling static ORDER BY for a query
  List<Person> findByLastnameOrderByFirstnameAsc(String lastname);
  List<Person> findByLastnameOrderByFirstnameDesc(String lastname);
}
```

Xử lý các tham số đặc biệt

- Có thể truyền thêm các tham số đặc biệt vào trong câu truy vấn, chẳng hạn như Pageable, Slice và Sort
- Spring Data repository sẽ nhận diện và chuyển thành câu truy vấn phù hợp
- Ví dụ:

```
Page<User>findByLastname(String lastname, Pageable pageable);

Slice<User> findByLastname(String lastname, Pageable pageable);

List<User> findByLastname(String lastname, Sort sort);

List<User> findByLastname(String lastname, Pageable pageable);
```

Pageable, Slice

- Pageable là đối tượng hỗ trợ phân trang
- Các phương thức thông dụng của Pageable: first(), getOffset(), getPageNumber(), getPageSize(), getSort(), hasPrevisous(), isPaged(), next(), previousOrFirst()
- Slice là đối tượng hỗ trợ phân trang, nhưng không biết được tổng số lượng trang (Pageable biết tổng số lượng trang)
- Slice có thể tốt hơn Pageable về mặt hiệu năng khi làm việc với nhiều dữ liệu (do không phải đếm tổng số lượng trang)
- Các phương thức thông dụng của Slice: getContent(), getNumber(), getPageable(), getSize(), getSort(), hasNext(), hasPrevious(), isFirst(), isLast()

Sort

- Sort là đối tượng hỗ trợ sắp xếp khi truy vấn
- Các constructor thông dụng:
  - **Sort**(**Sort.Direction** direction, **List**<**String**> properties)
- Các phương thức static thông dụng:
  - **by**(**List**<**Sort.Order**> orders)
  - **by**(**Sort.Direction** direction, **String**... properties)
  - **by**(**Sort.Order**... orders)
- Các phương thức thông dụng của Sort:
  - and(Sort)
  - Ascending()
  - Descending()
  - Interator()

Hạn chế số lượng kết quả

- Sử dụng từ top hoặc first để hạn chế số lượng kết quả
- Có thể thêm một giá trị số để quy định số lượng kết quả
- Nếu không có giá trị số thì 1 kết quả sẽ được trả về
- Ví dụ:

```
User findFirstByOrderByLastnameAsc();

User findTopByOrderByAgeDesc();

Page<User> queryFirst10ByLastname(String lastname, Pageable pageable);

Slice<User> findTop3ByLastname(String lastname, Pageable pageable);

List<User> findFirst10ByLastname(String lastname, Sort sort);

List<User> findTop10ByLastname(String lastname, Pageable pageable);
```

Sử dụng Spring Data

- Spring Data có thể được sử dụng trong container hoặc độc lập
- Trong các container, có thể sử dụng cơ chế CDI (Context and Dependency Injection) để tạo các bean repository
- Trong các ứng dụng độc lập, cần thêm thư viện Spring Data vào trong classpath và tự khởi tạo các đối tượng repository
- Ví dụ:

```
RepositoryFactorySupport factory = … // Instantiate factory here
UserRepository repository =
factory.getRepository(UserRepository.class);
```

Hỗ trợ web

- Spring Data hỗ trợ tích hợp với các tầng web
- Có thể sử dụng annotation @EnableSpringDataWebSupport trong Java Configuration để cho phép tích hợp
- Ví dụ:

```
@Configuration @EnableWebMvc
@EnableSpringDataWebSupport
class WebConfiguration {

}
```

- Annotation @EnableSpringDataWebSupport sẽ thêm các thành phần DomainClassConverter và HandlerMethodArgumentResolvers vào trong ứng dụng web

DomainClassConverter

- DomainClassConverter là thành phần cho phép tự động xác định các tham số trong request của web và chuyển thành các entity tương ứng
- Ví dụ:

```
@Controller
@RequestMapping("/users")
class UserController {
   @RequestMapping("/{id}")
   String showUserForm(@PathVariable("id") User user,
                                    Model model) {
      model.addAttribute("user", user);
      return "userForm";
   }
}
```

HandlerMethodArgumentResolvers

- HandlerMethodArgumentResolvers là thành phần cho phép tự động nhận diện và chuyển đổi các đối tượng Pageable và Sort tương ứng với request của controller
- Ví dụ:

```
@Controller
@RequestMapping("/users")
class UserController {
  @Autowired
  private UserRepository repository;
  @RequestMapping
  String showUsers(Model model, Pageable pageable) {
    model.addAttribute("users",
    repository.findAll(pageable)); return "users";
  }
}
```

Ánh xạ các tham số với Pageable

Tham số	Giải thích
page	Số trang. Mặc định là 0
size	Kích thước trang. Mặc định là 20.
sort	Trật tự sắp xếp (ASC | DESC). Mặc định là ascending. Ví dụ: ?sort=firstname&sort=lastname,asc

Cấu hình sử dụng JPA Repository

```
@Configuration @EnableJpaRepositories
@EnableTransactionManagement
class ApplicationConfig  {
   @Bean
   public DataSource dataSource() {
      EmbeddedDatabaseBuilder builder = new EmbeddedDatabaseBuilder();
      return builder.setType(EmbeddedDatabaseType.HSQL).build();
   }
   .......
}
```

```
@Bean
public LocalContainerEntityManagerFactoryBean entityManagerFactory() {
  HibernateJpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();
  vendorAdapter.setGenerateDdl(true);
  LocalContainerEntityManagerFactoryBean factory = new
  LocalContainerEntityManagerFactoryBean(); factory.setJpaVendorAdapter(vendorAdapter);
  factory.setPackagesToScan("com.acme.domain");
  factory.setDataSource(dataSource());
  return factory;
}
@Bean
public PlatformTransactionManager transactionManager(EntityManagerFactory entityManagerFactory) {
  JpaTransactionManager txManager = new
  JpaTransactionManager();
  txManager.setEntityManagerFactory(entityManagerFactory);
  return txManager;
}
```

Thảo luận

Converter và Formatter

Converter và Formatter

- Converter và Formatter hỗ trợ chuyển đổi dữ liệu nhập vào sang kiểu dữ liệu thích hợp
- Ví dụ, Spring sẽ tự động thử chuyển đổi dữ liệu từ một trường <input type="date"> sang một đối tượng java.util.Date
- Converter là các thành phần sử dụng chung cho toàn bộ hệ thống, có thể sử dụng converter ở bất cứ tầng nào của ứng dụng.
- Formatter thì chỉ được thiết kế để sử dụng ở tầng web (web tier)

Định nghĩa Converter

- Triển khai interface Converter

```
public interface Converter<S, T>
```

  - Trong đó:
    - S là kiểu dữ liệu nguồn
    - T là kiểu dữ liệu đích

- Ví dụ, định nghĩa converter để chuyển từ kiểu String sang LocalDate:

```
public class StringToLocalDateConverter implements Converter<String, LocalDate> {

  @Override
   public LocalDate convert(String source) {
      //Conversion
   }
}
```

Đăng ký converter

```
@Configuration
@ComponentScan("com.codegym.converter")
@EnableWebMvc
public class ApplicationConfig extends WebMvcConfigurerAdapter {
  @Override
   public void addFormatters(FormatterRegistry registry) {
     StringToLocalDateConverter convert = new
StringToLocalDateConverter("MM-dd-yyyy");
     registry.addConverter(stringToLocalDateConverter);
   }
}
```

Định nghĩa Formatter

- Triển khai interface Formatter

```
public interface Formatter<T>
```

  - Trong đó T là kiểu dữ liệu đích

- Ví dụ, chuyển đổi sang kiểu dữ liệu LocalDate:

```
public class LocalDateFormatter implements
  Formatter<LocalDate> { @Override
  public LocalDate parse(String text, Locale locale) throws ParseException {
    //String to LocalDate
  }
  @Override
  public String print(LocalDate date, Locale locale) {
    //LocalDate to String
  }
}
```

Đăng ký formatter

```
@Configuration
@EnableWebMvc
@ComponentScan("com.codegym.formatter")
public class ApplicationConfig extends WebMvcConfigurerAdapter {
  @Override
  public void addFormatters(FormatterRegistry registry) {
    LocalDateFormatter localDateFormatter = new
    LocalDateFormatter("MM-dd-yyyy");
    registry.addFormatter(localDateFormatter);
  }
}
```

Demo

Converter và Formatter

Thảo luận

Phục vụ tài nguyên tĩnh

Phương thức addResourceHandlers()

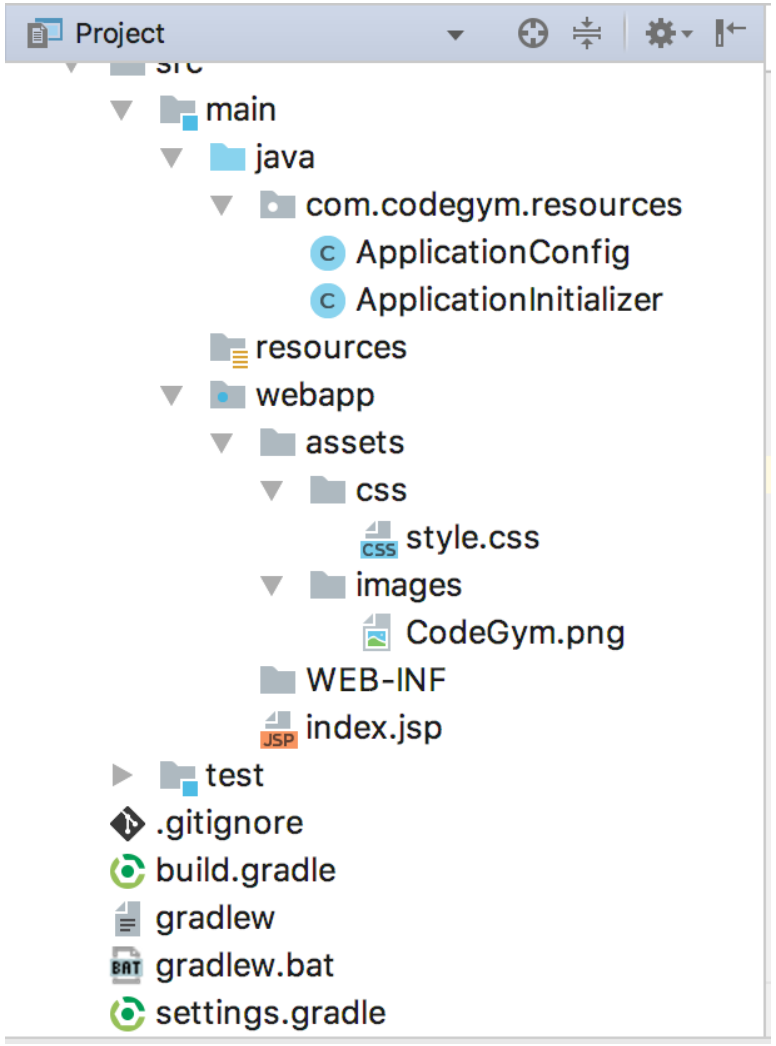
- Phương thức addResourceHandlers() của interface WebMvcConfigurer giúp đăng ký các thư mục chứa các tài nguyên tĩnh
- Các tài nguyên tĩnh bao gồm các file như ảnh, css, javascript…
- Các tài nguyên tĩnh có thể được đặt trong thư mục của ứng dụng, trong classpath hoặc các vị trí khác

Đăng ký các resource location

```
@Configuration
@EnableWebMvc
@ComponentScan("com.codegym.resources")
public class ApplicationConfig extends WebMvcConfigurerAdapter {
  @Override
  public void addResourceHandlers(ResourceHandlerRegistry registry) {
     registry
          .addResourceHandler("/assets/**")
          .addResourceLocations("/assets/");
     registry
          .addResourceHandler("/uploads/**")
          .addResourceLocations("file:/Users/nhat/Desktop/upload/");
  }
}
```

Sử dụng các file tĩnh

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
 <head>
  <title>Home page</title>
  <link rel="stylesheet" type="text/css"
  href="/assets/css/style.css"/>
 </head>
 <body>
  <img src="/assets/images/CodeGym.png">
  <h1 class="title">Home Page</h1>
  <p class="content">This paragraph is styled.</p>
 </body>
</html>
```



Demo

Phục vụ tài nguyên tĩnh

Tổng kết

- Spring Data cung cấp các interface repository để tự động hoá các thao tác với CSDL
- Interface CrudRepository hỗ trợ các thao tác CRUD cơ bản
- Interface PagingAndSortingRepository hỗ trợ phân trang và sắp xếp
- Có thể khai báo các interface tuỳ biến kế thừa từ interface Repository hoặc các interface con của nó
- Có thể định nghĩa các phương thức truy vấn tuỳ biến
- Câu lệnh truy vấn có thể được sinh ra trực tiếp dựa trên tên của phương thức truy vấn

Tổng kết

- Converter và Formatter là hai cơ chế để chuyển đổi từ một kiểu dữ liệu sang kiểu dữ liệu khác
- Converter được sử dụng chung cho toàn bộ ứng dụng
- Formatter thích hợp để sử dụng cho tầng web
- Phương thức addResourceHandlers() của interface WebMvcConfigurer giúp đăng ký các thư mục chứa các tài nguyên tĩnh

Hướng dẫn

Hướng dẫn làm bài thực hành và bài tập

Chuẩn bị bài tiếp theo: Validation