FREE eBook

LEARNING spring-data-jpa

Free unaffiliated eBook created from
Stack Overflow contributors.

#spring-data-jpa

Table of Contents

About 1

Chapter 1: Getting started with spring-data-jpa 2

Remarks 2

Versions 2

Examples 2

Installation or Setup 2

search by Entity property and search by Entity property in 3

Chapter 2: Repositories 4

Remarks 4

Examples 4

Creating a repository for a JPA-managed entity 4

Finding all instances of an entity class 5

Finding a particular instance of an entity class by the identifier 5

Finding all instances of an entity class with an attribute matching a specified value 6

Credits 7

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: spring-data-jpa

It is an unofficial and free spring-data-jpa ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official spring-data-jpa.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with spring-data-jpa

Remarks

This section provides an overview of what spring-data-jpa is, and why a developer might want to use it.

It should also mention any large subjects within spring-data-jpa, and link out to the related topics. Since the Documentation for spring-data-jpa is new, you may need to create initial versions of those related topics.

Versions

Version	Release Date
1.9.0.RELEASE	2015-09-01
1.8.0.RELEASE	2015-03-23
1.7.0.RELEASE	2014-09-05
1.6.0.RELEASE	2014-05-20
1.5.0.RELEASE	2014-02-24
1.4.0.RELEASE	2013-09-09
1.3.0.RELEASE	2012-02-07
1.2.0.RELEASE	2012-10-10
1.1.0.RELEASE	2012-05-16
1.0.0.RELEASE	2011-07-21

Examples

Installation or Setup

To start using Spring data JPA, you must include the dependency in your project with the one of Spring core, all together. If you're using Maven as dependency management system (replace *version-number* for the version you want to use):

```
<dependencies>
    <dependency>
        <groupId>org.springframework.data</groupId>
        <artifactId>spring-data-jpa</artifactId>
        <version>version-number</version>
    </dependency>
</dependencies>
```

And if you're using Gradle:

```
dependencies {
    compile 'org.springframework.data:spring-data-jpa:version-number'
}
```

You can also set it up when using Spring Boot, just include the starter dependency and get rid of the version number:

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
</dependencies>
```

search by Entity property and search by Entity property in

```
WarehosueEntity findWarehouseById(@Param("id") Long id);

List<WarehouseEntity> findWarehouseByIdIn(@Param("idList") List<Long> warehouseIdList);
```

Read Getting started with spring-data-jpa online: https://riptutorial.com/spring-data-jpa/topic/4318/getting-started-with-spring-data-jpa

Chapter 2: Repositories

Remarks

The Spring Data project allows application programmers to work with data stores using a consistent interface that makes use of an abstraction called `Repository`. A Spring Data `Repository` is modeled after the Repository pattern made popular by domain-driven design. Spring Data provides a central Java interface named `Repository` that subprojects can extend to provide features specific to data stores.

In addition to the `Repository` interface, Spring Data also provides two more core interfaces - `CrudRepository` that defines the contract for basic *CRUD* (*create*, *read*, *update* and *delete*) functionality; and `PagingAndSortingRepository` that extends `CrudRepository` by defining a contract for pagination and sorting.

These three core interfaces (`Repository`, `CrudRepository` and `PagingAndSortingRepository`) ensure that:

1. Application programmers can access data stores (such as relational databases, document based NoSQL databases, graph databases, etc.) in a consistent way.
2. It is easy to switch the underlying storage for a *domain entity* (see domain-driven design) without having to also change the way in which the application interacts with the data store.
3. Specific implementations can provide features specific to data stores.

Examples

Creating a repository for a JPA-managed entity

Entity class

```
@Entity
@Table(name = "USER")
public class User {
    @Id
    @Column(name = "ID")
    private Long id;

    @Column(name = "USERNAME")
    private String username;

    @ManyToOne
    @JoinColumn("ORGANIZATION_ID")
    private Organization organization;
}
```

Repository interface

```
@Repository
```

```
public interface UserRepository extends CrudRepository<User, Long> {
    public User findByUsername(String username);
}
```

The method declaration in the interface will generate the following jpql query:

```
select u from User u where u.username = :username
```

alternatively we can define a custom query:

```
@Query("select u from User u where u.username = :username")
public User findByUsername(@Param("username") String username)
```

we can easily add sorting to the method declaration:

```
public interface UserRepository extends PagingAndSortingRepository<User, Long> {
    public User findByUsernameOrderByUsernameAsc(String username);
}
```

we can also use in-built pagination support:

```
public Page<User> findByOrganizationPaged(Organization organization, Pageable pageable);
```

the service layer (or whoever calls this method) will then pass a PageRequest to the method:

```
public Page<User> getByOrganizationPagedOrderByUsername(Organization organization, int page,
int size, String direction){
    return userRepository.findByOrganizationPaged(organization, new PageRequest(page, size,
Direction.valueOf(direction),
                "username")
}
```

Finding all instances of an entity class

All instances (objects) of an entity class can be loaded from the underlying database table as follows (akin to retrieving all rows from the table):

```
Iterable<Foo> foos = fooRepository.findAll();
```

The `findAll` method is provided by the `CrudRepository` interface. It returns an `Iterable` instead of a more concrete type like `List` or `Set` because some implementations of the interface may be unable to return a `Collection` type and therefore using a `Collection` type for the returned value will result in loss of functionality for them.

Invoking the `findAll` method results in the JPA query `select foo from Foo foo` being executed on the underlying database.

Finding a particular instance of an entity class by the identifier

A particular instance of an entity class can be loaded as follows:

```
Foo foo = fooRepository.findOne(id);
```

The `findOne` method is provided by the `CrudRepository` interface. It expects an identifier that uniquely identifies an entity instance (for instance, a primary key in a database table). The Java type for the `id` parameter must match the type assigned to the entity attribute annotated with the JPA `@Id` annotation.

Invoking the `findOne` method results in the JPA query `select foo from Foo foo where foo.[primary-key-column] = :id` being executed on the underlying database.

Finding all instances of an entity class with an attribute matching a specified value

All instances of an entity class with one of the class attributes matching a specified value can be retrieved as follows:

```
public interface FooRepository extends CrudRepository<Foo, Long> {
  List<Foo> findAllByName(String name);
}
```

Invoking the `findAllByName` method results in the JPA query `select foo from Foo foo where foo.name = :name` will be executed on the underlying database.

Points to note

1. `name` must be an attribute on the `Foo` entity class.
2. The method name must begin with `find`, `get` or `read`. Other keywords like `select` will not work.
3. There is no guarantee on the order in which the results will be returned.

Read Repositories online: https://riptutorial.com/spring-data-jpa/topic/5688/repositories

Credits

S. No	Chapters	Contributors
1	Getting started with spring-data-jpa	Community, Sheldon Papa, Xtreme Biker
2	Repositories	amicoderozer, Gautam Jose, ido flax, manish, sanjaykumar81, SirKometa