PagingAndSortingRepository

==========

(upto boot 2.x) =>Child Repository of CrudRepository.. having methods to select/retrieve records by applying Sorting and Pagination activities..

=>Sorting takes place in the following order (Ascending)

->special chars (?,_,+ and etc..)

->Numbers

->alphabets

The code is::

In spring boot 2.x

important Repositories in spring data jpa

a) CrudRepository

b) PagingAndSortingRepository c) JPARepository Specific t8ata jpa (SQL)

Common Repositories for both SQL and NoSQL

note:: In spring boot 3.x PagingAndSortingRepository is an independent Repository(1) i.e it is not having any link with CrudRepository(1)

public interface PagingAndSortingRepository<T, ID> extends CrudRepository<T, ID> { Iterable<T> findAll(Sort sort);

}

select operation with sorting

Page<T> findAll(Pageable pageable); ---- select operation with pagination

upto spring boot 2.x

Type hierarchy

upto spring boot 2.x

Repository<T, ID>

CrudRepository<T, ID>

F4 for

✓ PagingAndSortingRepc

JpaRepository<T, ID:

type hierarchy

In spring boot 3.x

public interface PagingAndSortingRepository<T, ID> extends Repository<T, ID> {

Repository interfaces hierarchy in spring boot 2.x Repository(1) ↑ extends CrudRepository(1)

extends

PagingSortingRepository(1)

extends

Iterable<T> findAll(Sort sort);

Page<T> findAll(Pageable pageable);

Iterable<T> findAll(Sort sort);

=>Retrives all the records of Db table either in asc or'desc order specified through Sort object

=>Sort object contains

Example App

==========

Repository Interface

the one or more propertes and the order of Sorting (ASC or DESC) Sort.Direction.ASC

eg1: Sort sort=Sort.by ⇓,"docName"); (asc order on docName property data)

+

eg2:: Sort sort1=Sort.by(,,"docName", "doçId"); --> desc order on docName,docId properties data. Sort. Direction.DESC var args representing the property mames

public interface IDoctorRepository extends PagingAndSortingRepository<Doctor, Integer> {

Sort ---> class

}

Direction ---> Enum

JpaRepository(1)

QueryByExampleExecutor

✓ JpaRepository<T, ID> JpaRepositoryImplementation<T, ID> SimpleJpaRepository<T, ID> Queryds JpaRepository<T, ID>

Repository<T, ID>

✓

✓

CrudRepository<T, ID>

ListCrudRepository<T, ID>

JpaRepository<T, ID>

> JpaRepositoryImplementation<T, ID>

PagingAndSortingRepository<T, ID>

ListPagingAndSortingRepository<T, ID> JpaRepository<T, ID>

> JpaRepositoryImplementation<T, ID>

=>ctrl+shift+1 :: To search and get source code given class

=>ctrl+o :: To get info about

highlighed class/interface/enum

Spring boot 3.x Repository interfaces hierarchy

Repository (1)

extends

PagingAndSortingRepository (1)

servcie Interface

// One method can have only one var arg param that to last param of the method //Var args are internally arrays..

```
public Iterable<Doctor> showDoctorsBySorting(boolean asc, String ...props);
SEvice Impl class
@Autowired
private IDoctorRepository doctorRepo;
@Override
public Iterable<Doctor> showDoctorsBySorting(boolean asc, String... props) { //prepare the Sort object
Sort sort=Sort.by(asc?Direction.ASC:Direction.DESC, props);
// use Repo
Iterable<Doctor> it=doctorRepo.findAll(sort);
return it;
method
In runner class
@Component
public class PandSRepoTestRunner implements CommandLineRunner {
@Autowired
private IDoctorMgmtService service;
@Override
public void run(String... args) throws Exception {
service.showDoctorsBySorting(true, "docName").forEach(System.out::println);
");
service.showDoctorsBySorting(false, "income", "docName").forEach(System.out::println);
System.out.println(".
}
}
pagination
The paging and Sorting activities
are very useful in report generation operations like Sales Report, Progress Report and etc..
>var args are internally arrays
=> In one method decl or definition we can have max of one var arg that to last param/arg
CrudRepository
extends
extends
CassandraRepository
JpaRepository
MongoRepository
note:: In a java method designing we can place only one var arg type parameter that to as the last parameter.
Every var arg param is internally an array type param
```

==========

=> The process of dislaying huge no.of records page by page is called pagination. It is very useful

in report generation

eg:: gmail inbox, sales report, product catalogs and etc..

output Page<T>

input (pageNo(0 based), PageSize)

`findAll(Pageable ageable)`

↓

Returns a page of entities meeting the paging restriction provided in the Pageable object.

ed

=>Takes pageNo(0 based),pageSize (no.of recordss in each page) as inputs in the form of Pageable object and returns Page<T> object having multiple details like request page records, total pages count, current page number, total records and etc..

Slice<T> (1)

extends

Page<T> (1)

These two interfaces contain multiple methods to get various details related to pagination like page content, pageno,total pages and etc..

isFirst(), isLast(), getContent(), getNumber(), getSlice(), getTotalPages() and etc.. |--->List<T>

=>Page<T> obj means it is the object of a java class that implements Page<T>(I)

=> To create Pageable object having the inputs like pageNo, PageSize we take the support of PageRequest.of(-,-)method

=> Pageable obj means it is the object of a java class that implements Pageable(1)

O based index 1 based index)

Pageable obj (pageNo, pageSize)

(input)

PagingAndSortingRepository(1)

Impl class obj (IhMemory Proxy class obj)

findAll(Pageable pageable){

Page<T> obi (output)

getContent()+>gives List<T> (requested

o-r mapping logic

page records)

getNumber()

getTotalPages()

isFirst()

isLast()

To create Pageable obj

Example App

==========

1

3

Pageable pageable PageRequest.of(pageNo,pageSize); //only paging info (Static method) (0 based)

Page<Movie> page= movieRepo.findAll(pageable);

output

input

for example if db table is having 20 records.. then we get 7 pages(0-6 pages) like (3,3,3,3,3,3,2 records in pages) and gives f'page records (indirectly 2 page becoz page no is 0 based) like 4,5,6 records will come.. if ask for 6 page records (indirectly 7 page) then we get 19,20 records (only 2 are there) ..if ask for 7 or above page records then we get no records.

note:: we can create Pageable obj even including Sorting and paging information

eg: Pageable pageable=PageRequest.of(pageNo, pageSize, Sort obj); //paging and sorting info

Sort.by(Direction.ASC, "mname")

2

5

Page<Movie> page= movieRepo.findAll(pageable);

of the db table

get

=> First sorting on all records takes place and then pagination takes place (here we 3rd page records after sorting the records)

In service Interface

```
public Page<Doctor> showDoctorsInfoByPageNo(int pageNo, int pageSize, boolean ascOrder, String props);
```

In service Impl class

```
@Override
public Page<Doctor> showDoctorsInfoByPageNo(int pageNo, int pageSize, boolean ascOrder, String props) {
//prepare the Sort object
Sort sort=Sort.by(ascOrder?Direction.ASC:Direction.DESC, props);
//prepre Pageable object
Pageable pageable=PageRequest.of(pageNo, pageSize,sort);
// use the repo
Page<Doctor> page=doctorRepo.findAll(pageable);
return page;
}
```

In runner class

```
System.out.println("
try {
```

--findAll(Pageable pageable)-------");

//service.showDoctorsInfoByPageNo(1, 3, true, "docName").forEach(System.out::println); Page<Doctor> page=service.showDoctorsInfoByPageNo(2, 3, false, "docName");

System.out.println("page number::"+page.getNumber());

System.out.println("pages count::"+page.getTotalPages());

System.out.println(" is it first page::"+page.isFirst());

System.out.println(" is it last page::"+page.isLast());

System.out.println("page Size::"+page.getSize());

System.out.println("page Elements count::"+page.getNumberOfElements()); if(!page.isEmpty()) {

List<Doctor> list =page.getContent();

list.forEach(System.out::println);

}

else

System.out.println("no page found");

}//try catch(Exception e) {

Boot-DataJPA-Proj2-PagingAndSortingRepsitory [boot]

Spring Elements

src/main/java

e.printStackTrace();

}

com.nt

It internally uses psuedo column "rownum" and sub queries concept a lot to get records through pagination

› BootDataJpaProj2PagingAndSortingRepsitoryApplication.java

com.nt.entity

> Doctor.java

boot

note:: spring 3.x needs minimum oracle 12c version to execute generated SQL Query related pagination i.e 3.x generated SQL Query fails in the prior versions oracle 12c.. This problem is not there in MySQL

>

com.nt.repository

>IDoctorRepository.java

com.nt.runner

› PandSRepo TestRunner.java

com.nt.service

> DoctorMgmentServiceImpl.java

> IDoctorMgmtService.java

#src/main/resources

application.properties

src/test/java

> JRE System Library [JavaSE-17]

>

Maven Dependencies

>src

› target

WHELP.md

mvnw

mvnw.cmd

M pom.xml

upto spring boot 2.x, the PagingAndSortingRepository(I) is given as the Sub interface of

of CrudRepository... From spring boot 3.x the PagingAndSortingRepository(1) is given

as the direct sub interface of Repository(1). The JpaRepository(1) is extending from CrudRepository (1) indirectly

example on Pagination

Repository interface

====

```
public interface IDoctorRepository extends
```

with respect

```
PagingAndSortingRepository<Doctor, Integer>,Crud Repository<Doctor, Integer> {
```

to spring boot 3.x

```
}
```

service interface

```
public void showDataThrough Pagination(int pageSize);
```

service Impl class

```
@Override
public void showDataThrough Pagination(int pageSize) { //decide the no.of pages
long count=doctorRepo.count();
long pagesCount=count/pageSize;
//pagesCount=count%pageSize==0?pagesCount:++pagesCount;
if(count%pagesCount!=0)
pagesCount++;
for(int i=0;i<pagesCount;++i) {
```

Runner class

============

```
service.showDataThrough Pagination(3);
//create Pageable object
```

Pageable pageable=PageRequest.of(i, pageSize);

//get each page records

Page<Doctor> page=doctorRepo.findAll(pageable);

System.out.println("page :"+(page.getNumber()+1)+" records of "+page.getTotalPages());

page.getContent().forEach(System.out::println);

The Repository interfaces hierarchy in spring boot 3.x

is different from spring boot 2.x

System.out.println(".

}

JpaRepository

==========

_");

(in spring boot 2.x)

=> CrudRepository, PagingSortingRepository are Commons Repository interfaces for

both SQL and NO SQL DB s/ws.. if use methods these repositories we need not to change code in service class though our app moves from SQL DB s/w to NO SQL DB s/w. Repository(1) (0 methods)

extends

CrudRepository(1) (12 methods)

This hiearchy is given

w. r. to spring boot 2.x

spring data commons

extends

Repository

PagingSortingRepository(1)

(2 methods)

JpaRepository(1)

MangoRepository(1)

(15 methods)

(n methods)

For spring data jpa

w.r.t SQL DB s/w

of

The 15 methods JPARepository(1) are

All Methods

Modifier and Type

void

void

void

default void

List<T>

<S extends T>

List<S>

<S extends T>

List<S>

List<T>

List<T>

void

T

T

<S extends T>

List<S>

<S extends T>

List<S>

<S extends T>

S

==========

Neo4JRepository(1) CassendraRepository(1)

(n methods)

For spring data No SQL DB s/w

w.r.t No-SQL DB s/ws

Instance Methods Abstract Methods

Default Methods

Deprecated Methods

Method and Description

deleteAllByIdInBatch (Iterable<ID> ids)

Deletes the entities identified by the given ids using a single query.

deleteAllInBatch()

Deletes all entities in a batch call.

deleteAllInBatch (Iterable<T> entities)

Deletes the given entities in a batch which means it will create a single query.

deleteInBatch(Iterable<T> entities)

Deprecated.

Use deleteAllInBatch (Iterable) instead.

findAll()

`findAll(Example<S> example)`

`findAll(Example <S> example, Sort sort)`

`findAll(Sort sort)`

`findAllById(Iterable<ID> ids)`

`flush()`

(n methods)

Flushes all pending changes to the database. getById(ID id)

Returns a reference to the entity with the given identifier. getone (ID id)

Deprecated.

use JpaRepository#getById(ID) instead.

`saveAll(Iterable<S> entities)`

`saveAllAndFlush (Iterable<S> entities)`

Saves all entities and flushes changes instantly.

`saveAndFlush(S entity)`

Saves an entity and flushes changes instantly.

note:: In spring boot 3.x getById() is also deprecated as alternate they have given getReferenceById(-) method

(performs lazy loading)

Methods inherited from interface orn onrinafromauark data ranociton, Dosing And CortinaDonacitanı

Spring boot 3.x Repositories hierarchy

✓ Repository<T, ID>

• CrudRepository<T, ID>

✓ ListCrudRepository<T, ID> JpaRepository<T, ID>

> > JpaRepositoryImplementation<T, ID>

✓ PagingAndSorting Repository<T, ID> ListPagingAndSortingRepository<T, ID> JpaRepository<T, ID>

Repository (1)

extends

CrudRepository(1)

extends

ListCrudRepository(1)

↑

boot

spring 3.x

Repositories hierarchy

PagingAndSortingRepository(1)

extends

ListPagingSortingRepository(1)

All Methods

Modifier and Type

`void`

`void`

`void`

`default void`

extends

JpaRepository(1)

In Spring boot 3.x

Instance Methods Abstract Methods Default Methods

Method

`deleteAllByIdInBatch (Iterable <ID> ids)`

`deleteAllInBatch()`

`deleteAllInBatch (Iterable <T> entities)`

`deleteInBatch (Iterable <T> entities)`

Deprecated Methods

Description Deletes the entities identified by the given ids using a single query.

Deletes all entities in a batch call.

Deletes the given entities in a batch which means it will create a sin Deprecated.

`Use deleteAllInBatch (Iterable) instead.`

=>Most of the the methods avaiable in JpaRepository are also there in Crud Repository,

PagingAndSortingRepository interfaces but they work in undelying JPA Impl (hibernate) style..

Use this JpaRepository methods only when the same methods are not there in CrudRepository,

PagingSortingRepository Interfaces..

some differences here

note::: CrudRepository,PagingAndSortingRepository methods are are implemented in Spring data jpa linked with hibernate where as the JpaRepository methods are directly implemented in Hibernate

`flush()`

`getById(ID id)`

`<S extends T>`

`findAll(Example <S> example)`

`List <S>`

`<S extends T>`

`findAll(Example <S> example, Sort sort)`

`List <S>`

`void`

T

Flushes all pending changes to the database. Deprecated.

=======

CrudRepository(1) Methods

=>saveAll(), findAll() methods return type Iterable<T> Collection

=>Does not take Example obj, Sort

obj as the arguments in findXxx() methods

=> findById(-) return type is Optional<T>

=> deletexxx(-) methods perform

bulk deletion by generating multiple delete queries

working with

=> while findById(-) method no need of

extra cfg in application.properties enabling

=>These methods ¿mmon for

=>saveAll(), findAll() methods return type is List<T> Collection

=> findXxx() methods are avaiable taking Example,Sort objs

=>getById(-) return type is_<T> (Replaced with getReference Bvld(-) method) (deprecated)

Entity class

=> deletexxx(-) methods perform:

bulk deletion by generating

single delete SQL query through BatchProcessing

,we

(new method)

=> while working with getById(-) method nee an extra cfg in application.properties enabling hibernate lazy loading

lazy loading

to

both SQL and NO SQL DB s/ws

=> findById(-) performs eager Loading of record/object

=> These methods are specific SQL DB s/ws... =>getById(-),getOne(-), getReference Byld(-) methods perform Lazy Loa ng

same is applicable for

getOne(-), getReference Byld(-) methods

The methods of pre-defined Repositories

are designed to perform operations by taking

T

getOne (ID id)

JpaRepository (1) methods

the id property value as the criteria value

if u want to do same operations by using other

property values as the criteria values then

T

`getReferenceById(ID id)`

use getReferenceById(ID) instead.

Deprecated.

use getReferenceById(ID) instead.

Returns a reference to the entity with the given identifier.

we need to place custom methods in the Repository interfaces.

`<S extends T> List <S>`

saveAllAndFlush (Iterable <S> entities)

Saves all entities and flushes changes instantly.

`<S extends T> S`

saveAndFlush (S entity)

Saves an entity and flushes changes instantly.

note:: JpaRepository is bit known for performing delete operations through batch processing.

deleteAllByIdinBatch

`void deleteAllByIdInBatch (Iterable<ID> ids)`

Deletes the entities identified by the given ids using a single query. This kind of operation leaves JPAs first level cache and the database out of sync. Consider flushing the EntityManager before calling this

method.

Parameters:

`ids the ids of the entities to be deleted. Must not be null.`

Since:

=>allows to pass null values as the element values.

`2.5`

example app

==========

Repository Interfaces

public interface IDoctorRepository extends JpaRepository<Doctor, Integer> {

}

In service Inteface

public String deleteDoctorsByIdsInBatch(List<Integer> ids);

Service Impl class

@Override

public String delete DoctorsByIdsInBatch(List<Integer> ids) {

//load the entities

```
//delete the entities
List<Doctor> list=doctorRepo.findAllById(ids);
doctorRepo.deleteAllByIdinBatch(ids);
return list.size()+" records are deleted";
}
```

In runner class

```
System.out.println(service.deleteDoctorsByIdsInBatch(List.of(678,901)));
System.out.println(service.deleteDoctorsByIdsInBatch(List.of(678,null)));
```

throws exception becoz List.of(-,-) does

```
System.out.println(service.deleteDoctorsByIdsInBatch(Arrays.asList(16,null)));
```

not allow null elements

Does not throw any exception

=>List.of(-,-,-,-), Set.of(-), Map.of(---) are static factory methods given from Java9 to create Immutable Collection obj given datame this process they do not allow to have null values as the element values (Immutable collections can not have null values as the element values)

=> other pratices of creating collection like "using new operator" and using Arrays.asList(-) method and etc..

gives mutable collection where null values can be adâ as the element values.

b/w

What is difference deleteAllById(-) method of CrurdRepository and deleteAllByIdInBatch() method of JpaRepository

deleteAllById(-) in CrudRepository

(a) delete bulk records by generating multiple delete queries (no batch processing)

deleteAllByIdinBatch(-) in JpaRepository

(a) delete bulk records by generating single delete query with IN clause condition (batch processing takes place)

(b) Given ids can be the null values

same

b) Given ids for deletion can be null

(c) if few of the given ids based records are not available then those id values will be ignored

(d) The generated queries indirectly applies or clause condition (multiple delete queries)

(f) Works in both SQL And NoSQL DB s/ws

same

c) Not mandatory to have all records availability

in Db table matching with the given ids

(e) IN clause based condition nothing but

"or" condtion. (single delete query)

(f) only for SQL Db s/ws.

Assingnment :: find out diff b/w findAllbyId(-) of CrudRepository and

==========

findAllById(-) of JpaRepository

if main class name is having more than 30 letters then we get this exception Caused by: java.sql.SQLException: ORA-17190: Connection property format error: Property is 'v $session.program' and value is 'BootDataJpaProj02 PagingAndSortingRepositoryApplication' https://docs.oracle.com/error-help/db/ora-17190/

no.of

Solution is reduce the characters in the main class name

Chat

Ra