## horizontal line

Spring Data part 5: Paging and Sorting

# Overview

When we perform bulk operations like finding all person from the database or finding all person based on a country often we do the paging so that we can present a small data chunk to End user and in next request, we fetch next data chunk. By doing this we got two main advantages.

**Advantages**:

1. Enhancing readability for end users if we show whole data on a page then the page will be enough long with Scrollbar, the end user has to scroll for finding a particular person which is a bad UI design.
2. Reduce the Query time and enhance the performance as in spite of fetch all data we only fetch a small chunk of data so less query time. Many UI technology support client side paging which at a time fetch all data and based on use request show paginated data but this is not reducing the query time it only provide the advantage number 1.

**Disadvantages:**

To request every data chunk One server trip is required. You can optimize it through caching .

2 . When an end user in the process of fetching data another Person added to the system it may possible, the last entry of a page again shown in next page as a first entry.

**But we are always using paging to fetch a small chunk of data rather than whole data.**

# Spring Data and Pagination

Spring data provides support for pagination , It creates all the logic to implement a paging like count the rows for total pages , create data store specific queries etc.

To implement Paging is very easy in Spring data just we need to follow the below steps,

1. In your custom repository just extends **PagingAndSortingRepository**
2. Create a **PageRequest** Object which is an implementation of **Pageable** interface
3. This **PageRequest** Object takes page number, limit of the page (page size) and sort direction and sort field.
4. By passing requested page number and page limit you can get the data for this page.
5. If you pass a wrong page number Spring data will takes care of that and not return any data.

# Paging Code implementation

1. Create a repository which extends **PagingAndSortingRepository**

package com.example.repo;

import java.util.List;

import org.springframework.data.jpa.repository.Query;

import org.springframework.data.querydsl.QueryDslPredicateExecutor;

import org.springframework.data.repository.CrudRepository;

import org.springframework.data.repository.PagingAndSortingRepository;

import org.springframework.stereotype.Repository;

import com.example.person.Person;

@Repository

public interface PersonRepositary extends PagingAndSortingRepository<Person, Long>,QueryDslPredicateExecutor<Person> {

@Query("select p from Person p where p.country like ?1 order by country")

List<Person> findByCountryContains(String country);

List<Person> findPersonByHobbyName(String name);

@Query("select p from Person p where p.id = ?1 and country='America'")

Person findOne(Long id);

}

**2.** Create Domain Objects

package com.example.person;

import java.util.ArrayList;

import java.util.List;

import javax.persistence.CascadeType;

import javax.persistence.Entity;

import javax.persistence.FetchType;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.OneToMany;

@Entity

public class Person {

@Id

@GeneratedValue(strategy=GenerationType.AUTO)

private Long id;

private String name;

private String country;

private String gender;

@OneToMany(mappedBy="person",targetEntity=Hobby.class,

fetch=FetchType.EAGER,cascade=CascadeType.ALL)

List<Hobby> hobby;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getCountry() {

return country;

}

public void setCountry(String country) {

this.country = country;

}

public String getGender() {

return gender;

}

public void setGender(String gender) {

this.gender = gender;

}

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public List<Hobby> getHobby() {

return hobby;

}

public void setHobby(List<Hobby> hobby) {

this.hobby = hobby;

}

public void addHobby(Hobby ihobby)

{

if(hobby == null)

{

hobby = new ArrayList<Hobby>();

}

hobby.add(ihobby);

}

@Override

public String toString() {

return "Person [id=" + id + ", name=" + name + ", country=" + country + ", gender=" + gender + "]";

}

}

3. Fetch all person, Here I assume per page entry is one so create a Pagerequest Object with limit 1 and requesting for first page.

package com.example.person;

import java.util.List;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.Bean;

import org.springframework.data.domain.PageRequest;

import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

import com.example.repo.HobbyRepository;

import com.example.repo.PersonRepositary;

import com.querydsl.core.types.dsl.BooleanExpression;

@SpringBootApplication

@EnableJpaRepositories("com.example.repo")

public class PersonApplication {

private static final Logger log = LoggerFactory.getLogger(PersonApplication.class);

@Bean

public CommandLineRunner demo(PersonRepositary repository) {

findAll(repository);

return null;

}

private PageRequest gotoPage(int page)

{

PageRequest request = new PageRequest(page,1);

return request;

}

private void findAll(PersonRepositary repository)

{

Iterable<Person> pList = repository.findAll(gotoPage(0));

for(Person p : pList)

log.info("Person " + p);

}

public static void main(String[] args) {

SpringApplication.run(PersonApplication.class, args);

}

}

# Output :

Hibernate:

select

count(person0\_.id) as col\_0\_0\_

from

person person0\_

Hibernate:

select

person0\_.id as id1\_1\_,

person0\_.country as country2\_1\_,

person0\_.gender as gender3\_1\_,

person0\_.name as name4\_1\_

from

person person0\_ limit ?

Person Person [id=13, name=Samir mitra, country=America, gender=male]

# Paging and Sorting Code implementation:

To do sorting we have to pass the sort direction and sorting fields along with page number and limit. Suppose we want to sort by country name in ascending order we modify the goto

Method like following.

private PageRequest gotoPage(int page)

{

PageRequest request = new PageRequest(page,1,Sort.Direction.ASC,"country");

return request;

}

Here we pass the country property and ascending order.

# Output:

Hibernate:

select

count(person0\_.id) as col\_0\_0\_

from

person person0\_

Hibernate:

select

person0\_.id as id1\_1\_,

person0\_.country as country2\_1\_,

person0\_.gender as gender3\_1\_,

person0\_.name as name4\_1\_

from

person person0\_

order by

person0\_.country asc limit ?

Person Person [id=13, name=Samir mitra, country=America, gender=male]