

Arquitecturas de Software para Aplicaciones Empresariales

Query Methods

PROGRAMA DE INGENIERIA DE SISTEMAS

Ing. Daniel Eduardo Paz Perafán (danielp@unicauca.edu.co)

Ing. Pablo A. Magé (pmage@unicauca.edu.co)



¿Qué son los Query Methods?

- ❖ Son los métodos que permiten encontrar información en la BD y son declarados en la interfaz del repositorio.
- ❖ Los Query Methods se declaran en la interfaz del repositorio, la implementación es responsabilidad de **Spring Data JPA**.
- ❖ Para definir métodos de acceso específicos, **Spring Data JPA** soporta varias opciones:
 - Definir un nuevo método en la interface.
 - Proporcionar una JPQ query usando la anotación **@Query**
 - Usar el soporte avanzado de especificación y **querydsl** en Spring Data.
 - Definir consultas personalizadas via **JPA Named Queries**.

Consultas personalizadas automáticas

- ❖ Al crear una nuevo repositorio, **Spring Data** analiza los métodos definidos en la interface del repositorio y trata de generar automáticamente las consultas a partir de lo nombres de los métodos.
- ❖ Para la definición de los métodos nuevos se utilizan un conjunto de **keywords**.

```
public interface UsuarioRepository extends CrudRepository<Cliente,Integer>{  
  
    //Select * from Clientes where apellido=?  
    List<Cliente> findByApellido(String apellido);  
}
```

Keyword

Atributo de clase

Parámetro

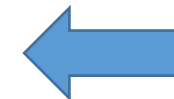
Consultar archivo Keywords-query
Methods.pdf

Keyword findBy:

```
public interface UsuarioRepository extends CrudRepository<Cliente,Integer>{  
  
    //Select * from Clientes where apellido=?  
    List<Cliente> findByApellido(String apellido);  
}
```

Para usar este método se modifican las clases de services:

```
public interface IClienteServices {  
    public List<Cliente> findAll();  
    public Cliente findById(Integer id);  
    public Cliente save(Cliente cliente);  
    public Cliente update(Integer id, Cliente cliente);  
    public boolean delete(Integer id);  
    public List<Cliente> findByApellido(String apellido);  
}
```



Keyword findBy:

Para usar este método se modifican las clases de services:

```
@Override
    public List<Cliente> findByApellido(String apellido) {
        List<Cliente> lista=this.servicioAccesoBaseDatos.findByApellido(apellido);

        System.out.println("Registros encontrados:"+lista.size());

        for(Cliente c:lista){
            System.out.println(c.getId()+":"+c.getNombre()+":"+c.getApellido());
        }
        return lista;
    }
```

Keyword And:

Permite realizar consultas más complejas.

Por ejemplo: Buscar clientes por apellido e email ordenado descendente por id.

Modificar la interface UsuarioRepository:

@Repository

```
public interface UsuarioRepository extends CrudRepository<Cliente,Integer>{
```

```
    //Select * from Clientes where apellido=?
```

```
    List<Cliente> findByApellido(String apellido);
```

```
    List<Cliente> findByApellidoAndEstado(String apellido, Integer estado);
```

```
}
```

Keyword And:

Modificar las clases de services:

```
public interface IClienteServices {  
    public List<Cliente> findAll();  
    public Cliente findById(Integer id);  
    public Cliente save(Cliente cliente);  
    public Cliente update(Integer id, Cliente cliente);  
    public boolean delete(Integer id);  
    public List<Cliente> findByApellido(String apellido);  
    public List<Cliente> findByApellidoAndEstado(String apellido, Integer  
estado);  
}
```

Keyword And:

Modificar las clases de services:

```
@Override
    public List<Cliente> findByApellidoAndEstado(String apellido, Integer estado) {
        List<Cliente>
        lista=this.servicioAccesoBaseDatos.findByApellidoAndEstadoOrderByIdDesc(apellido, estado);
        System.out.println("Registros encontrados:"+lista.size());
        for(Cliente c:lista){
            System.out.println(c.getId()+":"+c.getEstado()+":"+c.getApellido());
        }
        return lista;
    }
```


Keyword Between:

Permite realizar consultas entre rangos.

Por ejemplo: Buscar clientes entre un rango de ids.

Modificar la interface UsuarioRepository:

@Repository

```
public interface UsuarioRepository extends CrudRepository<Cliente,Integer>{
```

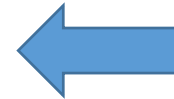
```
    //Select * from Clientes where apellido=?
```

```
    List<Cliente> findByApellido(String apellido);
```

```
    List<Cliente> findByApellidoAndEstadoOrderByIdDesc(String apellido, Integer estado);
```

```
    List<Cliente> findByIdBetween(int id1, int id2);
```


```
}
```



Keyword Between:

Modificar las clases de services/:

```
public interface IClienteServices {  
    public List<Cliente> findAll();  
    public Cliente findById(Integer id);  
    public Cliente save(Cliente cliente);  
    public Cliente update(Integer id, Cliente cliente);  
    public boolean delete(Integer id);  
    public List<Cliente> findByApellido(String apellido);  
    public List<Cliente> findByApellidoAndEstado(String apellido, Integer estado);  
    public List<Cliente> findByIdPorRango(int id1, int id2);  
}
```



Keyword Between:

Modificar las clases de services/:

```
@Override
    public List<Cliente> findByIdPorRango(int id1, int id2) {

        List<Cliente> lista=this.servicioAccesoBaseDatos.findByIdBetween(id1, id2);
        System.out.println("Registros encontrados:"+lista.size());
        for(Cliente c:lista){
            System.out.println(c.getId()+":"+c.getNombre()+":"+c.getApellido());
        }
        return lista;
    }
```

Anotación `@PathVariable` vs Anotación `@RequestParam`:

- ❖ Estas anotaciones se usan para extraer valores de una petición URI, con algunas diferencias.
- ❖ `@RequestParam` extrae valores de una petición, `@PathVariable` extrae desde una ruta URI.
- ❖ `@RequestParam` extrae valores codificados, `@PathVariable` extrae valores no codificados.
- ❖ Los valores de estas anotaciones pueden ser opcionales:
`@RequestParam(required = false)`
`@PathVariable(requiere = false)`

Invocación de las consultas desde el controlador:

```
@GetMapping("/clientes/{variable}")
public List<Cliente> show(@PathVariable Integer variable, @RequestParam Integer id, @RequestParam String apellido,
    @RequestParam Integer estado) {
    List<Cliente> lista = new LinkedList<>();
    System.out.println("variable=" + variable + " id=" + id + " apellido=" + apellido + " estado=" + estado);
    if (variable == 1) {
        Cliente objCliente = clienteService.findById(id);
        lista.add(objCliente);
    } else if (variable == 2) {
        lista = clienteService.findByApellido(apellido);
    } else if (variable == 3) {
        lista = clienteService.findByApellidoAndEstado(apellido, estado);
    }
    return lista;
}
```

¿Cuáles serían las Keywords-query que se deben utilizar para construir las siguientes consultas?

- Buscar un conjunto de docentes que coincidan con un patrón de búsqueda ignorando las mayúsculas y minúsculas. El patrón puede corresponder a la identificación, nombres, apellidos o correo electrónico.
- Buscar un conjunto de docentes que coincidan con un nombre ignorando las mayúsculas y minúsculas y ordenarlos en orden descendente por su identificación.
- Buscar un conjunto de docentes que pertenezcan a un grupo de investigación y a un tipo de vinculación (planta, ocasional, catedra).

Uso de la salida estándar:

Para imprimir las peticiones en la salida estándar se fija en el archivo

`application.properties` la propiedad:

```
spring.jpa.show-sql=true
```

Para embellecer la petición usar:

```
spring.jpa.properties.hibernate.format_sql=true
```

Uso de loggers:

```
logging.level.org.hibernate.SQL=DEBUG
```

```
logging.level.org.hibernate.type.descriptor.sql.BasicBinder=TRACE
```

- ❖ Persistence with Spring, Baeldung. Sitio: <https://s3.amazonaws.com/baeldung.com/Persistence+with+Spring.pdf>
- ❖ JPA Query API. Sitio: <https://www.objectdb.com/java/jpa/query/api>
- ❖ JA-JPQL. Sitio: https://www.tutorialspoint.com/jpa/jpa_jpql.htm

Muchas gracias
Preguntas

