

Spring Data JPA

nir en PDF ▶

Contenido

- Objetivos
- 1. Introducción a Spring Data JPA
- 2. Configuración
- 3. Interfaces de Repositorio
- 4. Crear Repositorios
- 5. Query Methods
- 6. Consultas Personalizadas (`@Query`)
- 7. Paginación y Ordenación
- 8. Proyecciones
- 9. Auditoria
- 10. Ejemplo Completo
- 11. Tabla de Referencia

RA3 - Herramientas de Mapeo Objeto-Relacional

Objetivos

- Comprender la arquitectura de Spring Data JPA
- Configurar Spring Data JPA en proyectos Spring Boot
- Crear y utilizar repositorios JPA
- Implementar consultas personalizadas
- Aplicar paginación y ordenación

1. Introducción a Spring Data JPA

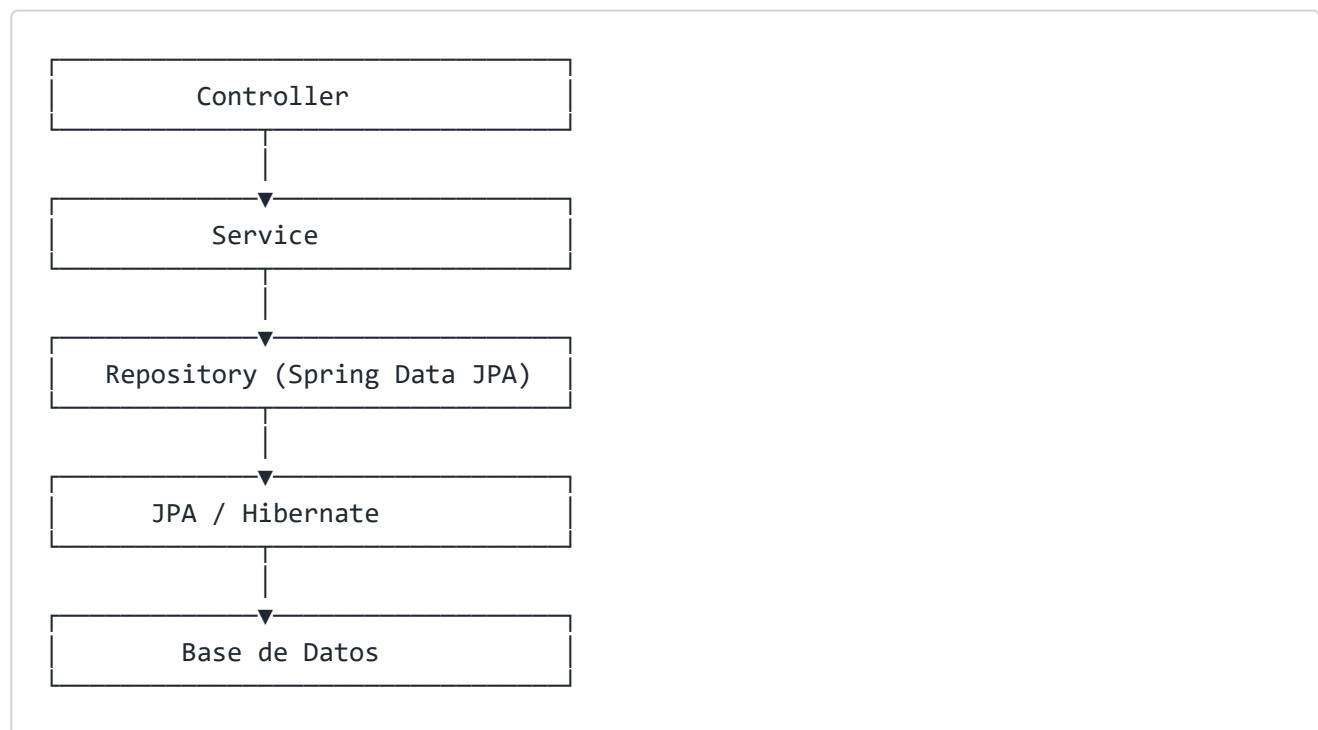
1.1. ¿Qué es Spring Data JPA?

Spring Data JPA es un módulo del proyecto Spring Data que facilita la implementación de repositorios basados en JPA. Proporciona una capa de abstracción sobre JPA que reduce drásticamente el código necesario para acceder a datos.

1.2. Ventajas

Característica	Descripción
Menos código	Elimina código repetitivo de DAOs
Query Methods	Genera consultas desde nombres de métodos
Paginación	Soporte integrado para paginación
Auditoría	Campos automáticos de creación/modificación
Transacciones	Gestión automática de transacciones

1.3. Arquitectura



2. Configuracion

2.1. Dependencias Maven

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

    <!-- Driver MySQL -->
    <dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <scope>runtime</scope>
    </dependency>

    <!-- H2 para desarrollo/test -->
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <scope>runtime</scope>
    </dependency>
</dependencies>
```

2.2. Configuracion application.properties

```
# Conexion a la base de datos
spring.datasource.url=jdbc:mysql://localhost:3306/tienda
spring.datasource.username=root
spring.datasource.password=secret
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

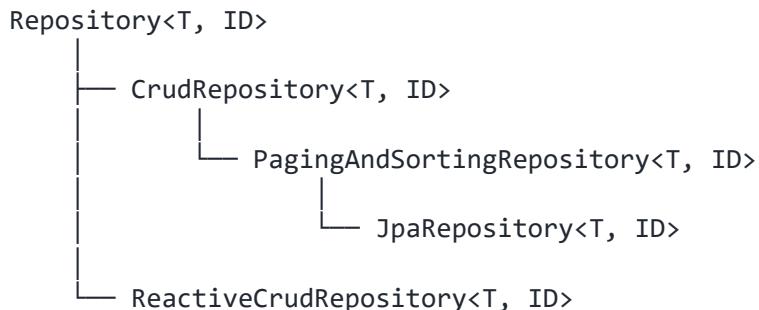
# JPA / Hibernate
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
```

2.3. Valores de ddl-auto

Valor	Descripcion	Uso
none	No hace nada	Produccion
validate	Valida esquema	Produccion
update	Actualiza esquema	Desarrollo
create	Crea esquema cada arranque	Testing
create-drop	Crea y borra al cerrar	Testing

3. Interfaces de Repositorio

3.1. Jerarquia de Interfaces



3.2. CrudRepository

Operaciones CRUD basicas:

```
public interface CrudRepository<T, ID> extends Repository<T, ID> {
    <S extends T> S save(S entity);
    <S extends T> Iterable<S> saveAll(Iterable<S> entities);
    Optional<T> findById(ID id);
    boolean existsById(ID id);
    Iterable<T> findAll();
    Iterable<T> findAllById(Iterable<ID> ids);
    long count();
    void deleteById(ID id);
    void delete(T entity);
    void deleteAll(Iterable<? extends T> entities);
    void deleteAll();
}
```

3.3. JpaRepository

Extiende con operaciones adicionales:

```
public interface JpaRepository<T, ID> extends PagingAndSortingRepository<T, ID> {
    List<T> findAll();
    List<T> findAll(Sort sort);
    List<T> findAllById(Iterable<ID> ids);
    <S extends T> List<S> saveAll(Iterable<S> entities);
    void flush();
    <S extends T> S saveAndFlush(S entity);
    void deleteInBatch(Iterable<T> entities);
    void deleteAllInBatch();
    T getOne(ID id); // Deprecated, usar getReferenceById
    T getReferenceById(ID id);
}
```

4. Crear Repositorios

4.1. Repositorio Basico

```
public interface ProductoRepository extends JpaRepository<Producto, Long> {
    // Metodos heredados: save, findById, findAll, delete, count...
}
```

4.2. Uso en Servicio

```
@Service
@RequiredArgsConstructor
public class ProductoService {

    private final ProductoRepository productoRepository;

    public List<Producto> listarTodos() {
        return productoRepository.findAll();
    }

    public Optional<Producto> buscarPorId(Long id) {
        return productoRepository.findById(id);
    }

    public Producto guardar(Producto producto) {
        return productoRepository.save(producto);
    }

    public void eliminar(Long id) {
        productoRepository.deleteById(id);
    }
}
```

5. Query Methods

5.1. Derivacion de Consultas

Spring Data JPA genera consultas automaticamente a partir del nombre del metodo:

```
public interface ProductoRepository extends JpaRepository<Producto, Long> {

    // SELECT p FROM Producto p WHERE p.nombre = ?1
    List<Producto> findByNombre(String nombre);

    // SELECT p FROM Producto p WHERE p.precio > ?1
    List<Producto> findByPrecioGreaterThanOrEqual(Double precio);

    // SELECT p FROM Producto p WHERE p.nombre LIKE %?1%
    List<Producto> findByNombreContaining(String texto);

    // SELECT p FROM Producto p WHERE p.activo = true
    List<Producto> findByActivoTrue();
}
```

5.2. Palabras Clave

Palabra Clave	Ejemplo	JPQL Generado
And	findByNombreAndPrecio	WHERE nombre=?1 AND precio=?2
Or	findByNombreOrPrecio	WHERE nombre=?1 OR precio=?2
Is, Equals	findByNombre	WHERE nombre=?1
Between	findByPrecioBetween	WHERE precio BETWEEN ?1 AND ?2
LessThan	findByPrecioLessThan	WHERE precio < ?1
GreaterThan	findByPrecioGreaterThan	WHERE precio > ?1
IsNull	findByDescripcionIsNull	WHERE descripcion IS NULL
Like	findByNombreLike	WHERE nombre LIKE ?1
Containing	findByNombreContaining	WHERE nombre LIKE %?1%
StartingWith	findByNombreStartingWith	WHERE nombre LIKE ?1%
EndingWith	findByNombreEndingWith	WHERE nombre LIKE %?1
OrderBy	findByNombreOrderByPrecio	ORDER BY precio ASC
Not	findByNombreNot	WHERE nombre <> ?1
In	findByNombreIn	WHERE nombre IN (?1)
IgnoreCase	findByNombreIgnoreCase	WHERE UPPER(nombre)=UPPER(?1)

5.3. Navegacion por Relaciones

```
public interface ProductoRepository extends JpaRepository<Producto, Long> {

    // Navegacion por ManyToOne
    List<Producto> findByCategoriaNombre(String nombreCategoria);

    // Navegacion profunda
    List<Producto> findByCategoriaProveedorNombre(String nombreProveedor);
}
```

5.4. Limitar Resultados

```
public interface ProductoRepository extends JpaRepository<Producto, Long> {

    // Primer resultado
    Optional<Producto> findFirstByOrderByPrecioAsc();

    // Top N resultados
    List<Producto> findTop5ByOrderByPrecioDesc();

    // Conteo
    Long countByCategoriaNombre(String categoria);

    // Existencia
    boolean existsByNombre(String nombre);
}
```

6. Consultas Personalizadas (@Query)

6.1. JPQL con @Query

```
public interface ProductoRepository extends JpaRepository<Producto, Long> {

    // Parametros nombrados
    @Query("SELECT p FROM Producto p WHERE p.precio > :precio")
    List<Producto> buscarPorPrecioMayorA(@Param("precio") Double precio);

    // Parametros posicionales
    @Query("SELECT p FROM Producto p WHERE p.categoría.id = ?1")
    List<Producto> buscarPorCategoria(Long categoryId);

    // JOIN FETCH
    @Query("SELECT p FROM Producto p JOIN FETCH p.categoría")
    List<Producto> buscarTodosConCategoria();
}
```

6.2. SQL Nativo

```
@Query(value = "SELECT * FROM productos WHERE precio > ?1", nativeQuery = true)
List<Producto> buscarPorPrecioNativo(Double precio);

@Query(value = "SELECT COUNT(*) FROM productos WHERE categoría_id = :id",
       nativeQuery = true)
Long contarPorCategoria(@Param("id") Long categoryId);
```

6.3. Consultas de Modificacion

```

@Modifying
@Transactional
@Query("UPDATE Producto p SET p.precio = p.precio * :factor")
int actualizarPrecios(@Param("factor") Double factor);

@Modifying
@Transactional
@Query("DELETE FROM Producto p WHERE p.activo = false")
int eliminarInactivos();

```

7. Paginacion y Ordenacion

7.1. Sort

```

// En el servicio
Sort sort = Sort.by("nombre").ascending();
List<Producto> productos = productoRepository.findAll(sort);

// Ordenacion multiple
Sort sort = Sort.by("categoria.nombre").ascending()
    .and(Sort.by("precio").descending());

```

7.2. Pageable y Page

```

// Crear peticion de pagina
Pageable pageable = PageRequest.of(0, 10); // Pagina 0, 10 elementos
Page<Producto> pagina = productoRepository.findAll(pageable);

// Con ordenacion
Pageable pageable = PageRequest.of(0, 10, Sort.by("nombre"));

// Acceder a datos de la pagina
List<Producto> contenido = pagina.getContent();
int totalPaginas = pagina.getTotalPages();
long totalElementos = pagina.getTotalElements();
int numeroPagina = pagina.getNumber();
boolean hayMas = pagina.hasNext();

```

7.3. Query Methods con Paginacion

```
public interface ProductoRepository extends JpaRepository<Producto, Long> {
    Page<Producto> findByCategoriaNombre(String categoria, Pageable pageable);
    @Query("SELECT p FROM Producto p WHERE p.precio > :precio")
    Page<Producto> buscarPorPrecio(@Param("precio") Double precio, Pageable pageable);
}
```

8. Proyecciones

8.1. Proyeccion con Interfaz

```
// Interfaz de proyeccion
public interface ProductoResumen {
    String getNombre();
    Double getPrecio();
    String getCategoríaNombre(); // Navegacion
}

// En el repositorio
public interface ProductoRepository extends JpaRepository<Producto, Long> {
    List<ProductoResumen> findByActivoTrue();
}
```

8.2. Proyeccion con DTO

```
@Value
public class ProductoDTO {
    String nombre;
    Double precio;
}

@Query("SELECT new com.ejemplo.dto.ProductoDTO(p.nombre, p.precio) FROM Producto p")
List<ProductoDTO> obtenerResumen();
```

9. Auditoria

9.1. Configuracion

```
@Configuration  
@EnableJpaAuditing  
public class JpaConfig {  
  
    @Bean  
    public AuditorAware<String> auditorProvider() {  
        return () -> Optional.of("sistema");  
    }  
}
```

9.2. Entidad Auditable

```
@Entity  
@EntityListeners(AuditingEntityListener.class)  
public class Producto {  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    private String nombre;  
  
    @CreatedDate  
    @Column(updatable = false)  
    private LocalDateTime fechaCreacion;  
  
    @LastModifiedDate  
    private LocalDateTime fechaModificacion;  
  
    @CreatedBy  
    @Column(updatable = false)  
    private String creadoPor;  
  
    @LastModifiedBy  
    private String modificadoPor;  
}
```

10. Ejemplo Completo

10.1. Entidad

```

@Entity
@Table(name = "productos")
@getter @Setter
@NoArgsConstructor @AllArgsConstructor
@Builder
public class Producto {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String nombre;

    private Double precio;

    private Boolean activo = true;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "categoria_id")
    private Categoria categoria;
}

```

10.2. Repositorio

```

public interface ProductoRepository extends JpaRepository<Producto, Long> {

    List<Producto> findByActivoTrue();

    List<Producto> findByCategoriaNombre(String categoria);

    Page<Producto> findByPrecioBetween(Double min, Double max, Pageable pageable);

    @Query("SELECT p FROM Producto p JOIN FETCH p.categoria WHERE p.activo = true")
    List<Producto> findActivosConCategoria();

    @Modifying
    @Transactional
    @Query("UPDATE Producto p SET p.activo = :activo WHERE p.categoria.id = :catId")
    int actualizarActivoPorCategoria(@Param("catId") Long categoriaId,
                                    @Param("activo") Boolean activo);
}

```

10.3. Servicio

```

@Slf4j
@Service
@RequiredArgsConstructor
@Transactional(readOnly = true)
public class ProductoService {

    private final ProductoRepository productoRepository;

    public Page<Producto> listarPaginado(int pagina, int tamano) {
        Pageable pageable = PageRequest.of(pagina, tamano, Sort.by("nombre"));
        return productoRepository.findAll(pageable);
    }

    public Optional<Producto> buscarPorId(Long id) {
        return productoRepository.findById(id);
    }

    @Transactional
    public Producto guardar(Producto producto) {
        log.info("Guardando producto: {}", producto.getNombre());
        return productoRepository.save(producto);
    }

    @Transactional
    public void eliminar(Long id) {
        log.info("Eliminando producto con id: {}", id);
        productoRepository.deleteById(id);
    }
}

```

11. Tabla de Referencia

Interfaz	Metodos Principales
CrudRepository	save, findById, findAll, delete, count
JpaRepository	saveAll, flush, deleteInBatch, getReferenceById
PagingAndSortingRepository	findAll(Sort), findAll(Pageable)

Anotacion	Uso
@Query	Definir consulta JPQL/SQL
@Param	Nombrar parametro de consulta
@Modifying	Marcar consulta de modificacion
@Transactional	Gestion de transacciones
@EntityGraph	Carga eager de relaciones