Tabla de Contenidos

- Spring Starters (Web y Persistencia)
- Spring Boot JPA y entidades
- JPA y Anotaciones
- Spring Boot JPA y properties
- Repositorios de Spring
- Repositorios de Spring Data y simplificación
- Spring Data es grande
  - JpaRepository y su métodos
- Otros artículos relacionados

El uso de Spring Boot JPA es cada día mas importante ya que poco a poco más proyectos van pasando de usar Spring clásico a ser arrancados con Spring Boot incluso proyectos antiguos se intentan renovar hacia esta tecnología. Por lo tanto configurar Spring Boot para trabajar con JPA es casi siempre necesario. Vamos a ver cómo hacerlo. El primer paso es generar un proyecto de Spring Boot que soporte JPA en este caso es sencillo ya que solo necesitamos un starter adicional el de spring-data-jpa . vamos a ver cómo queda el fichero de Maven:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.2.4.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
```

```
</parent>
<groupId>com.arquitecturajava</groupId>
<artifactId>jpa</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>jpa</name>
<description>Demo project for Spring Boot</description>

<properties>
  <java.version>1.8</java.version>
</properties>

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

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
    <exclusions>
      <exclusion>
        <groupId>org.junit.vintage</groupId>
        <artifactId>junit-vintage-engine</artifactId>
      </exclusion>
    </exclusions>
```

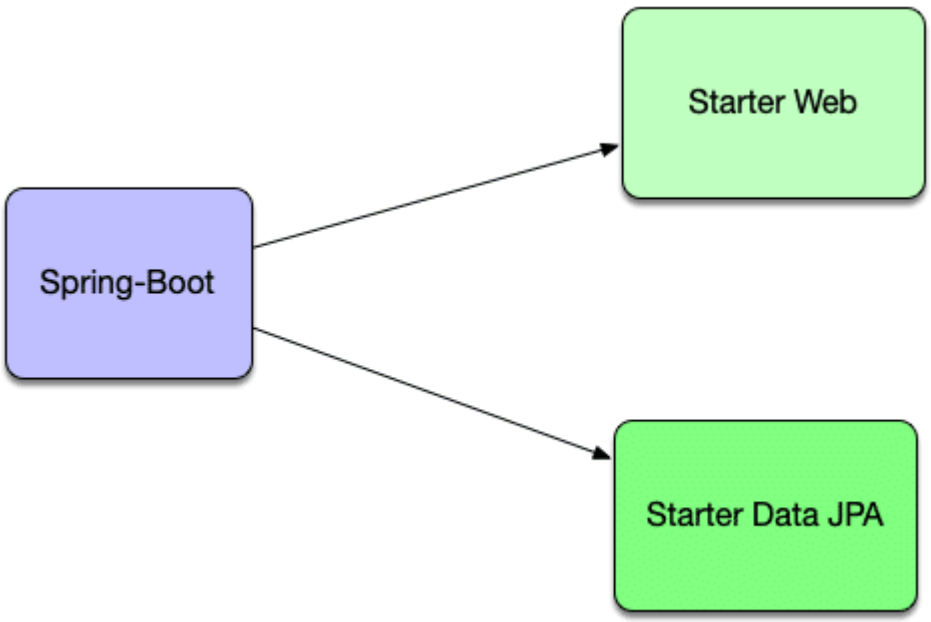
```
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>

</project>
```

Spring Starters (Web y Persistencia)

Acabamos de configurar Spring Boot con los dos starters que necesitamos para este proyecto uno es el starter web y otro es el starter de Spring Data para JPA.



Spring Boot JPA y entidades

Una vez que tenemos configurado el proyecto de arranque el primer paso es generar una entidad que pueda ser persistida.

```
package com.arquitecturajava.jpa.models;

import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class Persona {

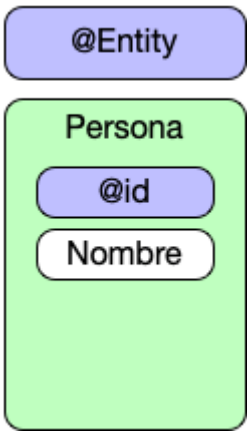
    @Id
    private String nombre;
    private String apellidos;
    private int edad;
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getApellidos() {
        return apellidos;
    }
    public void setApellidos(String apellidos) {
        this.apellidos = apellidos;
    }
    public int getEdad() {
        return edad;
```

```
    }
    public void setEdad(int edad) {
        this.edad = edad;
    }
    public Persona(String nombre, String apellidos, int edad) {
        super();
        this.nombre = nombre;
        this.apellidos = apellidos;
        this.edad = edad;
    }
    public Persona() {
        super();
    }

}
```

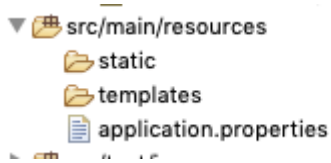
JPA y Anotaciones

En este caso hemos usado las dos anotaciones más sencillas @Entity que define la entidad como persistible y @Id que define la propiedades que es clave primaria.



Spring Boot JPA y properties

Ya tenemos la entidad definida y contiene las anotaciones necesarias. El siguiente paso es configurar los parámetros contra la base de datos usando el fichero de application.properties que Spring Boot nos provee en la estructura de carpetas.

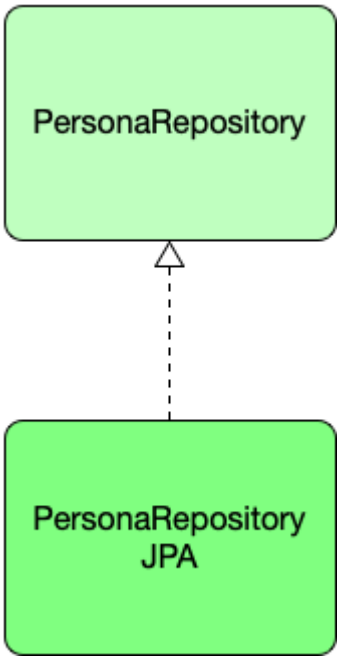


En el deberemos de rellenar las propiedades clásicas con una estructura determinada haciendo referencia a un datasource y al propio hibernate y su dialecto para conectarse a la base de datos:

```
spring.datasource.url=jdbc:mysql://localhost:8889/springjpa
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver.class=com.mysql.jdbc.Driver
spring.jpa.properties.hibernate.dialect =
org.hibernate.dialect.MySQL5Dialect
```

Repositorios de Spring

Una vez configurado Spring para poder gestionar JPA . La forma mas sencilla de trabajar con ello es usar un @Repository con @PersistenceContext de tal manera que podamos crear un repositorio amigable usando un interface y su implementación.



Veamos el código:

```
package com.arquitecturajava.web1.repositories;

import java.util.List;

import com.arquitecturajava.web1.models.Persona;

public interface PersonaRepository {

	List<Persona> buscarTodos();

	void insertar(Persona persona);

	void borrar(Persona persona);
	Persona buscarUno(String nombre);

}
```

Implementamos:

```
package com.arquitecturajava.web1.repositories;

import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.TypedQuery;

import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;

import com.arquitecturajava.web1.models.Persona;

@Repository
public class PersonaRepositoryJPA implements PersonaRepository {

	@PersistenceContext
	EntityManager em;
	@Override
	public List<Persona> buscarTodos() {
		TypedQuery<Persona> consulta= em.createQuery("select p
from Persona p",Persona.class);
		return consulta.getResultList();
	}
	@Transactional
	public void insertar(Persona persona) {
		em.persist(persona);
	}
	@Transactional
```

```
        public void borrar(Persona persona) {
                em.remove(em.merge(persona));
        }
        @Override
        public Persona buscarUno(String nombre) {
                // TODO Auto-generated method stub
                return em.find(Persona.class, nombre);
        }
}
```

Esta sería una solución clásica en donde usamos el EntityManager de forma manual y es el encargado de gestionar la persistencia. Sin embargo hay otras opciones más avanzada pero que también necesitan de algo más de conocimiento.

Repositorios de Spring Data y simplificación

Realizado este paso recordemos que Spring nos permite de forma transparente crear interfaces de repositorios apoyados en su framework Spring Data . Esto nos automatiza las operaciones básicas sin que tengamos que construirlas nosotros mismos. Un ejemplo sencillo sería:

```
package com.arquitecturajava.jpa.repositories;

import org.springframework.data.repository.CrudRepository;

import com.arquitecturajava.jpa.models.Persona;

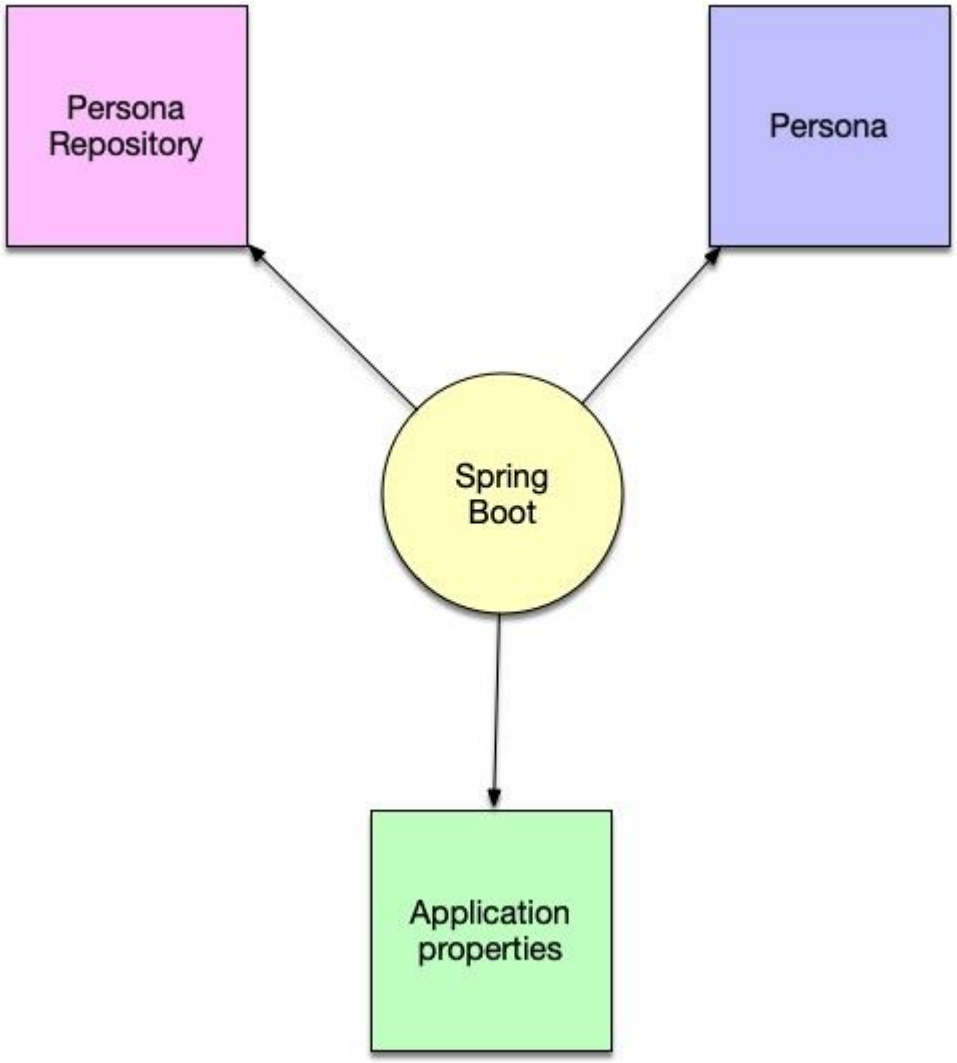
public interface PersonaRepository extends CrudRepository < Persona,
String > {

}
```

podríamos por lo tanto eliminar el código anterior y basarnos en este interface que contiene métodos como findAll ,save etc

Spring Data es grande

Spring Data es un framework grande y aunque parezca muy sencillo de manejar implementa conceptos complejos . Ahora y a disponemos de las piezas necesarias para acceder a la base de datos vía JPA (Entidad ,Fichero de Configuración,Repositorio) con Spring Data como framework de apoyo que nos ayuda a simplificar todo.



Una vez realizados los cambios a nivel Spring Boot JPA lo último que queda es ver cuál es el

contenido del programa principal que invoca a este repositorio y nos selecciona los datos de la base de datos.

```
package com.arquitecturajava.jpa;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

import com.arquitecturajava.jpa.repositories.PersonaRepository;

@SpringBootApplication
public class JpaApplication implements CommandLineRunner {

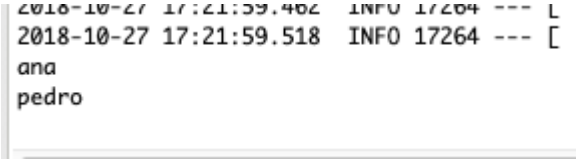
    @Autowired
    PersonaRepository repositorio;

    public static void main(String[] args) {
        SpringApplication.run(JpaApplication.class, args);
    }

    @Override
    public void run(String...args) throws Exception {

        repositorio.findAll().forEach((p) - >
System.out.println(p.getNombre()));
    }
}
```

Simplemente inyectamos el repositorio y ejecutamos Spring Boot desde linea de consola:



Los registros de la base de datos se imprimen en la consola . Finalmente hemos visto como se configura JPA con Spring y como hacer uso de Spring Data . Accediendo a los datos de forma rápida usando el interface CommandLineRunner.

JpaRepository y su métodos

Los métodos más comunes en JpaRepository incluyen:

1. save(S entity): Guarda o actualiza una entidad.
2. findById(ID id): Busca una entidad por su ID.
3. findAll(): Retorna una lista con todas las entidades.
4. deleteById(ID id): Elimina una entidad por su ID.
5. count(): Cuenta el total de entidades.
6. existsById(ID id): Verifica si una entidad existe por su ID.
   Estos métodos permiten CRUD básico y operaciones adicionales en una base de datos de manera sencilla.

Otros artículos relacionados

1. Uso de Spring Properties y encriptación
2. Spring GetMapping ,PostMapping etc
3. Spring Initializer
4. JPA Persist y buenas prácticas
5. ¿JPA vs Hibernate?
6. Spring Data Projections y Rendimiento
7. Spring Data Custom Query