Tabla de Contenidos

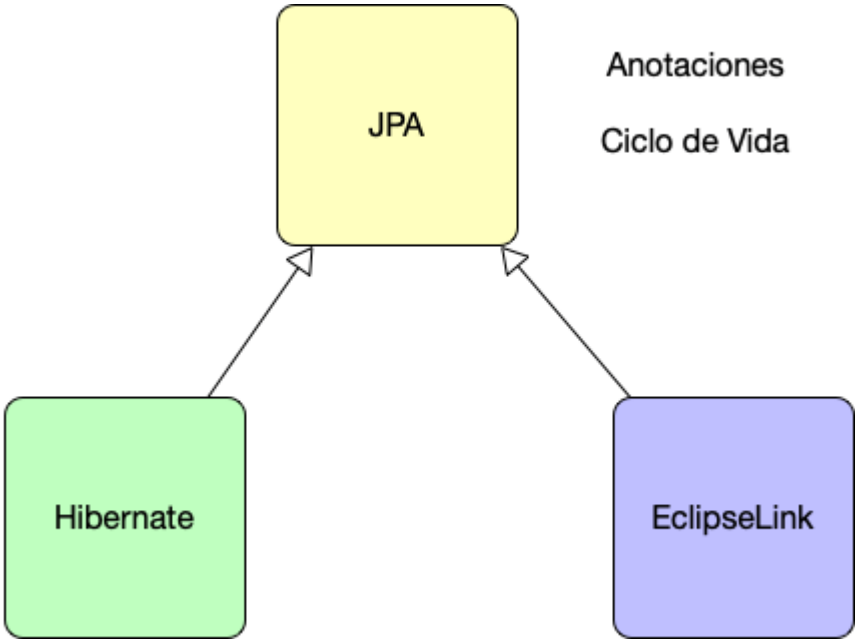
JPA vs Spring Data es una de las preguntas más clásicas en una entrevista. ¿Has trabajado con Spring Data , has trabajado con JPA? . Muchas personas no son capaces de diferenciar con claridad estas dos tecnologías. ¿Qué es JPA?. JPA o Java Persistence API es el standard de Java EE orientado a persistencia de datos . Como standard no es un framework sino que es un conjunto de reglas y anotaciones con las cuales los diversos frameworks de persistencia deben gestionar el ciclo de vida de los objetos a nivel de base de datos. Por ejemplo si mostramos una Entidad con sus anotaciones:

```
package com.arquitecturajava.web1.models;

import java.util.ArrayList;
import java.util.List;
import java.util.Objects;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import javax.persistence.Table;
@Entity
@Table(name="personas")
public class Persona {
        @Id
        private String nombre;
        private String apellidos;
```

```
private int edad;
@OneToMany (mappedBy = "persona")
private List<Examen> examenes= new ArrayList<Examen>();
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();
}
@Override
```

```
	public int hashCode() {
		return Objects.hash(nombre);
	}
	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		Persona other = (Persona) obj;
		return Objects.equals(nombre, other.nombre);
	}
	public Persona(String nombre) {
		super();
		this.nombre = nombre;
	}
}
```

Nos daremos cuenta de que usa anotaciones standard como @Entity o @Id o @Table. Todas estas anotaciones están ligada los Standares de JPA.



Eso lo podemos ver cuando revisamos los diversos imports ya que pertenecen a javax que es parte del standard y existen varias implementaciones como Hibernate o EclipseLink.

```
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import javax.persistence.Table;
```

Algunas personas me comentan que ellas usan org.hibernate , en ese caso no te estás basando en los standares sino que estás usando el propio framework Hibernate eso hay que tenerlo en cuenta.

Repositories Spring Framework

Ahora bien el standard define las anotaciones y el ciclo de vida de las entidades. Pero no define gran cosa a nivel de patronaje . Por ejemplo ¿Cómo podemos gestionar los repositorios con Spring Framework?. Es aquí donde el uso de JPA vs Spring empieza a clarificar las cosas:

```
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);
        }
}
```

En este caso hemos construido un repositorio nosotros con la anotación @Repository y usado un EntityManager para gestionar el ciclo de vida de las entidades a través inyectando la dependencia con @PersistenceContext. Este código mezcla JPA con Spring Framework . Es decir mezclamos las capacidades de inyección de dependencia de Spring con las capacidades de persistencia de JPA . Ahora bien tanto @Repository como @PersistenceContext pertenecen a Spring Framework , no pertenecen a JPA. En este código las anotaciones son de Spring.

JPA vs Spring Data

Este no es el último paso, podemos avanzar más y añadir Spring Data como framework para gestionar JPA . Spring Data Genera una capa de abstracción a nivel de patrones de diseño sobre nuestro sistema de persistencia. No solo genera una capa de abstracción sobre JPA sino sobre otras tecnologías como NoSQL con MongoDB. Por ejemplo un repositorio de Spring Data puede ser tan Sencillo como:

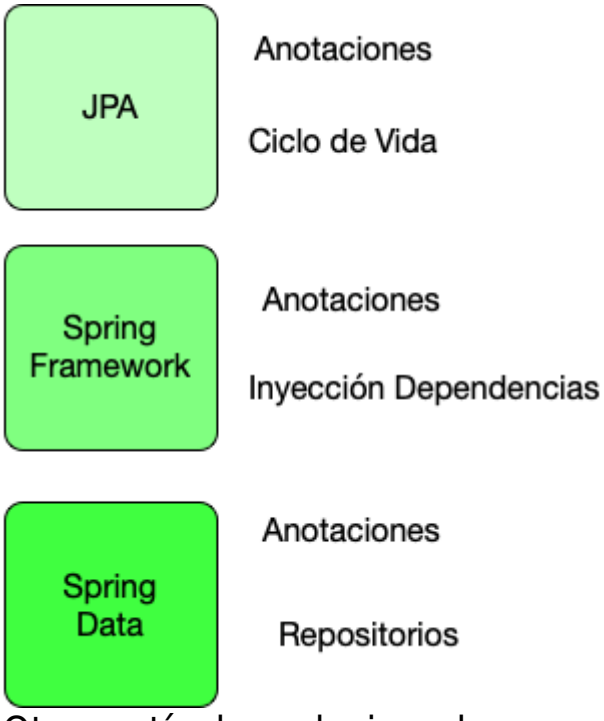
```
package com.arquitecturajava.web1.repositories;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

import com.arquitecturajava.web1.models.Persona;

@Repository
public interface PersonaRepository extends JpaRepository<Persona,
```

```
String> {


}
```

Este repositorio extiende JpaRepository que es el nombre de uno de los repositorios genéricos de Spring Data. ¿ Es esto parte de JPA y del Standard? . No no es parte de JPA sino que es parte de Spring Data que aporta repositorios Genéricos.

Conclusiones

Por lo tanto resumiendo mucho cuando hablamos de JPA vs Spring Data estamos hablando de dos cosas distintas que estan relacionadas pero que son distintas.



Otros artículos relacionados

- JPA Persist y buenas prácticas
- JPA Query Language Objetos vs Tablas
- ¿JPA vs Hibernate?

- Un ejemplo de JPA Entity Graph
- Spring Data
- Spring Boot JPA y su configuración