El uso de Spring @Transactional readonly es una de las características que en muchas ocasiones nos olvidamos al manejar Spring Framework. Spring nos provee de las capacidades de inyección de gestión transaccional distribuida y transparente para nuestros repositorios y servicios. Ahora bien eso no quiere decir que no delege en JPA para realizar estas tareas en muchas ocasiones estas tareas están orientadas a seleccionar un grupo de registros de la base de datos. Vamos a ver un ejemplo a través de la clase Persona:

```
package com.arquitecturajava;

public class Persona {

	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;
        }

}
```

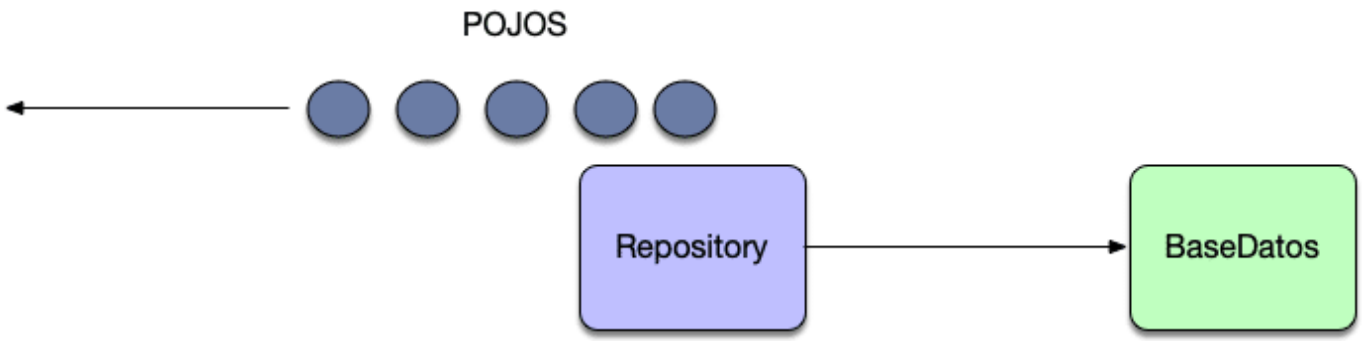
Spring @Transactional readonly

Una vez disponemos de la clase Persona lo más habitual es utilizar Spring Framework para construir una clase de repositorio que se encargue de seleccionar una lista de Personas .Algo del siguiente estilo :

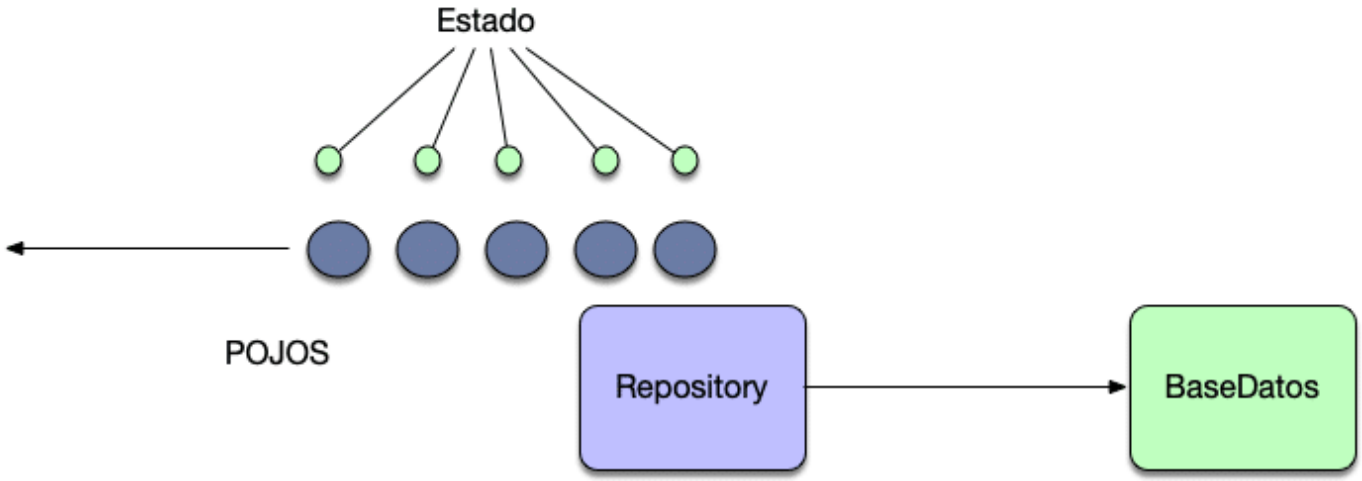
```
package com.arquitecturajava;

import java.util.List;

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

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

```
@Repository
public class PersonaRepository {

	@PersistenceContext
	EntityManager em;

	@Transactional
	public List<Persona> BuscarTodas() {
		return em.createQuery("select p from Persona
p",Persona.class).getResultList();
	}
}
```

Esto implica que nos tendremos que conectar a la base de datos y seleccionar un grupo de registros



Sin embargo las cosas no son tan sencillas como las vemos . El hecho de tener un método buscarTodos y seleccionar un conjunto de Personas (POJOS , Plain Old Java Objects) . No es lo último que un framework de persistencia tiene que hacer . El problema es que esta obligado a almacenar también de forma independiente el estado de cada objeto de tal forma que si en algún momento cambiamos los datos de cualquier objeto el sepa la información modificada y sea capaz de lanzar las consultas de actualización correctas.

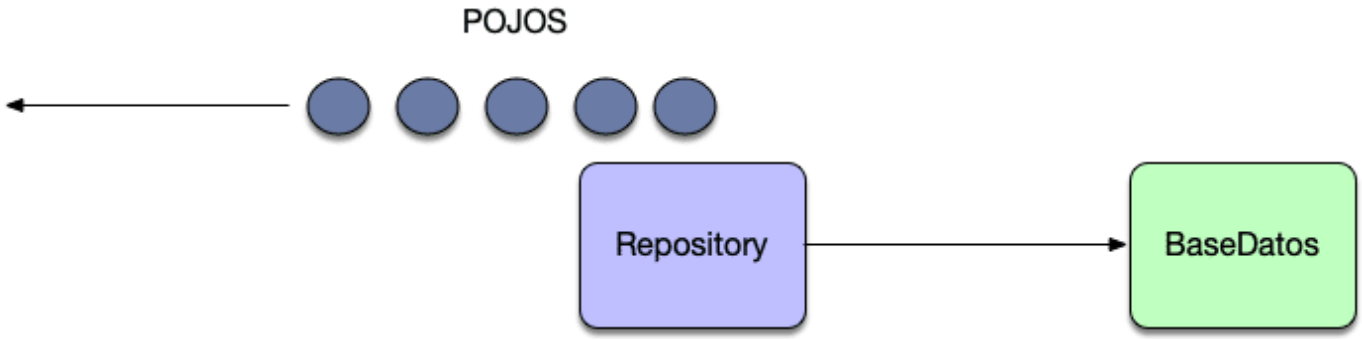


La pregunta que tenemos que hacernos . ¿En algún momento vamos a solicitar una modificación de la lista?. Es bastante evidente que en el 95% de las ocasiones no y por lo tanto podemos marcar con Spring @Transactional readonly el método para que no guarde el estado y tengamos un mejor rendimiento de la consulta.

```
package com.arquitecturajava;

import java.util.List;

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

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

@Repository
public class PersonaRepository {

        @PersistenceContext
        EntityManager em;

        @Transactional(readOnly=true)
```

```
        public List<Persona> BuscarTodas() {
                return em.createQuery("select p from Persona
p",Persona.class).getResultList();
        }
}
```

De esta manera la consulta no almacenará estado adicional a nivel de los objetos seleccionados.



Spring Framework siempre aporta sus mejoras sobre la programación clásica y nos permite afinar el rendimiento de la aplicación.

Otros artículos relacionados

- JPA @OneToOne y relaciones 1 a 1
- JPA Remove y Objetos gestionados
- Spring Boot JPA y su configuración
- https://spring.io/projects/spring-data-jpa