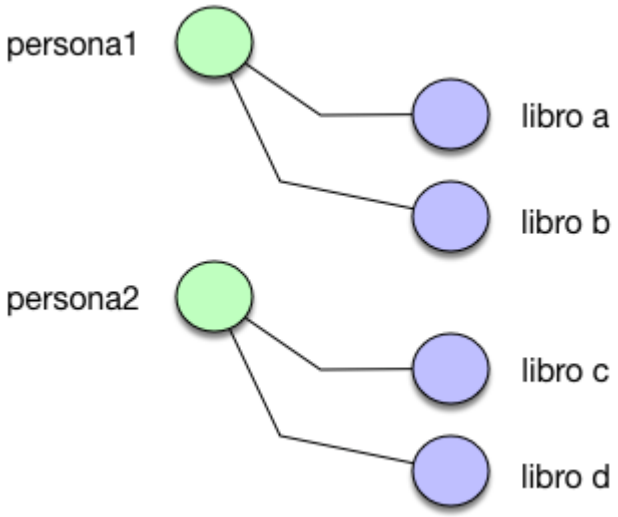
El uso de JPA DTO es algo bastante común cuando trabajamos con JPA. Recordemos que habitualmente cuando realizamos una consulta con Java Persistence API el resultado es un grafo de objetos.



En muchos casos esto nos es suficiente , pero hay algunas ocaciones en las que nos puede ser mucho más útil devolver un DTO (Data Transfer Object) . Este incluye únicamente la información absolutamente necesaria y su rendimiento a nivel de red aumentará. Vamos a construir un ejemplo con las clases Persona y Libro. Para ello lo primero es importar las dependencias de maven.

```
<dependencies>
                <dependency>
<groupId>org.hibernate.javax.persistence</groupId>
                        <artifactId>hibernate-jpa-2.1-api</artifactId>
                        <version>1.0.0.Final</version>
                </dependency>
                <dependency>
                        <groupId>org.hibernate</groupId>
                        <artifactId>hibernate-core</artifactId>
                        <version>5.2.10.Final</version>
                </dependency>
```

```
                <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.42</version>
</dependencies>
```

El siguiente paso es configurar el persistence.xml:

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
version="2.0">
 <persistence-unit name="UnidadPersonas">
 <class>es.curso.bo.Persona</class>
 <properties>
 <property name= "hibernate.show_sql" value="true" />
 <property name="hibernate.dialect"
value="org.hibernate.dialect.MySQLDialect" />
 <property name="javax.persistence.jdbc.driver"
value="com.mysql.jdbc.Driver" />
 <property name="javax.persistence.jdbc.user" value="root" />
 <property name="javax.persistence.jdbc.password" value="mysql" />
 <property name="javax.persistence.jdbc.url"
value="jdbc:mysql://localhost/java2" />

</properties>

</persistence-unit
</persistence>
```

Una vez configuradas las dependencias vamos a construir nuestras clases:

```
package com.arquitecturajava;

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

import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.OneToMany;

import org.hibernate.annotations.Cascade;

@Entity
public class Persona {
        @Id
        private String nombre;
        private String apellidos;
        private int edad;
        @OneToMany(mappedBy="persona",cascade = CascadeType.PERSIST)

        private List&amp;amp;lt;Libro&amp;gt; libros= new
ArrayList&amp;amp;lt;Libro&amp;gt;();

        public List&amp;amp;lt;Libro&amp;gt; getLibros() {
                return libros;
        }
        public void setLibros(List&amp;amp;lt;Libro&amp;gt; libros) {
                this.libros = libros;
        }
```

```
public boolean addLibro(Libro e) {
        return libros.add(e);
}
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();
}
```

```
}
```

```
package com.arquitecturajava;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;

@Entity
public class Libro {
        @Id
        private String titulo;
        private int paginas;

         @ManyToOne
     @JoinColumn(name="persona_nombre")
        private Persona persona;

        public String getTitulo() {
                return titulo;
        }
        public void setTitulo(String titulo) {
                this.titulo = titulo;
        }
        public int getPaginas() {
                return paginas;
        }
```

```
	public void setPaginas(int paginas) {
		this.paginas = paginas;
	}
	public Persona getPersona() {
		return persona;
	}
	public void setPersona(Persona persona) {
		this.persona = persona;
	}
	public Libro(String titulo, int paginas, Persona persona) {
		super();
		this.titulo = titulo;
		this.paginas = paginas;
		this.persona = persona;
	}
	public Libro() {
		super();
	}

}
```

Vamos a ejecutar una consulta normal utilizando JPQL que nos seleccione las personas con sus libros:

```
package com.arquitecturajava;

import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
```

```
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

public class Principal2 {

	public static void main(String[] args) {
		EntityManagerFactory emf =
Persistence.createEntityManagerFactory("UnidadPersonas");
		EntityManager em = emf.createEntityManager();
		List&amp;amp;lt;Persona&amp;amp;gt;
lista=em.createQuery("select p from Persona p",
Persona.class).getResultList();

		for (Persona p:lista) {

			System.out.println(p.getApellidos());
			for(Libro l :p.getLibros()) {

				System.out.println(l.getTitulo());
				System.out.println(l.getPaginas());

			}
		}
	}
}
```

El resultado se muestra en la consola:

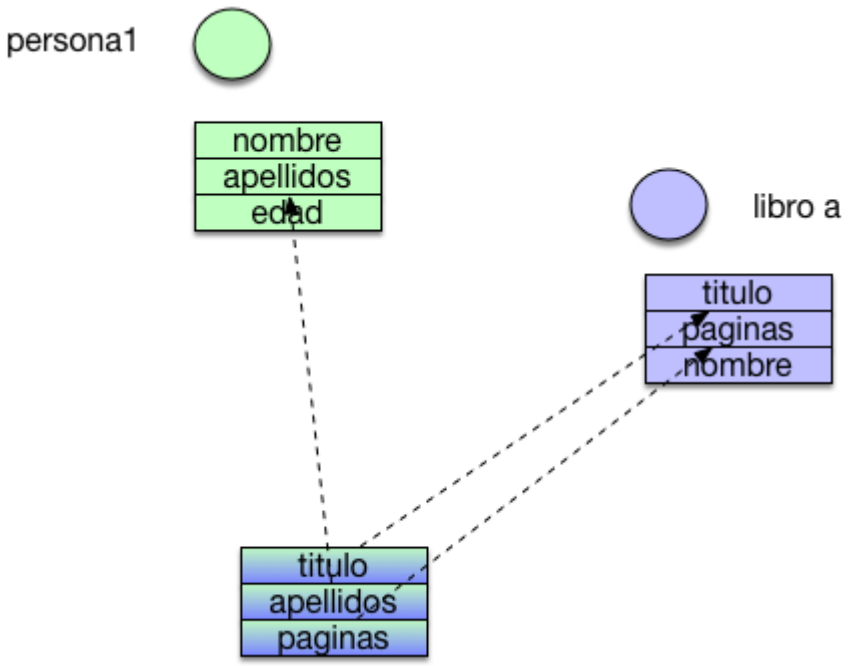
```
Principal2 (3) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_121.jdk/Contents/Home/bin/java (14 jul. 2017 8:51:24)
INFO: HHH000400: Using dialect: org.hibernate.dialect.MySQLDialect
jul 14, 2017 8:51:27 AM org.hibernate.hql.internal.QueryTranslatorFactoryInitiator initiateService
INFO: HHH000397: Using ASTQueryTranslatorFactory
Hibernate: select persona0_.nombre as nombre1_1_, persona0_.apellidos as apellido2_1_, persona0_.edad as edad3_1_ from Persona persona
perez
Hibernate: select libros0_.persona_nombre as persona_3_0_0_, libros0_.titulo as titulo1_0_0_, libros0_.titulo as titulo1_0_1_, libros0
java
200
perez
Hibernate: select libros0_.persona_nombre as persona_3_0_0_, libros0_.titulo as titulo1_0_0_, libros0_.titulo as titulo1_0_1_, libros0
java2
200
```

Se han tenido que realizar varias consultas para obtener tanto las Personas como los Libros

.

JPA DTO un enfoque diferente

En el ejemplo anterior podríamos haber utilizado un join fetch para mejorar el rendimiento y reducir las queries , pero el problema de fondo que en este caso queremos abordar es diferente. En nuestro caso queremos seleccionar únicamente un pequeño subconjunto de los datos del grafo utilizando JPA DTO.



Para ello podemos apoyarnos en un DTO (Data Transfer Object) ,creando una nueva clase:

```
package com.arquitecturajava;

public class LibroPersonaDTO {

	private String apellidos;
	private String titulo;
	private int paginas;
	public int getPaginas() {
		return paginas;
	}
	public void setPaginas(int paginas) {
		this.paginas = paginas;
	}
	public String getTitulo() {
		return titulo;
	}
	public void setTitulo(String titulo) {
		this.titulo = titulo;
	}
	public String getApellidos() {
		return apellidos;
	}
	public void setApellidos(String apellidos) {
		this.apellidos = apellidos;
	}
	public LibroPersonaDTO(String apellidos, String titulo, int
paginas) {
		super();
		this.apellidos = apellidos;
		this.titulo = titulo;
		this.paginas = paginas;
```

```
        }
}
```

En este caso nos queremos quedar con los títulos ,las páginas y el apellido de la persona a la que pertenece el libro.

¿Cómo podemos hacer esto con JPQL? . JPQL soporta un operador new que nos permite enlazar un DTO en la query, vamos a verlo:

```
package com.arquitecturajava;

import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

public class Principal3 {

        public static void main(String[] args) {
                EntityManagerFactory emf =
Persistence.createEntityManagerFactory("UnidadPersonas");
                EntityManager em = emf.createEntityManager();
                List&amp;amp;lt;LibroPersonaDTO&amp;gt;
lista=em.createQuery("select distinct new
com.arquitecturajava.LibroPersonaDTO( p.apellidos,l.titulo,l.paginas )
```

```
from Persona p, Libro l", LibroPersonaDTO.class).getResultList();
                for (LibroPersonaDTO dto: lista) {

                        System.out.println(dto.getTitulo());
                        System.out.println(dto.getPaginas());
                        System.out.println(dto.getApellidos());
                }

        }
}
```

El resultado de la consulta incluirá una lista de DTOS con únicamente los datos que necesitamos, hemos mejorado el rendimiento de nuestra consulta utilizando JPA DTO:

```
Fri Jul 14 09:00:20 CEST 2017 WARN: Establishing SSL connection without server's identity verification is
jul 14, 2017 9:00:20 AM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.MySQLDialect
jul 14, 2017 9:00:21 AM org.hibernate.hql.internal.QueryTranslatorFactoryInitiator initiateService
INFO: HHH000397: Using ASTQueryTranslatorFactory
Hibernate: select distinct persona0_.apellidos as col_0_0_, libro1_.titulo as col_1_0_, libro1_.paginas a
java
200
perez
java2
200
perez
```

Otros artículos relacionados:

1. JPA Database Schema y automatización
2. JPA Proxy y su funcionamiento
3. JPA Single Table Inheritance
4. Un ejemplo de JPA embedded objects
5. DTO concepto