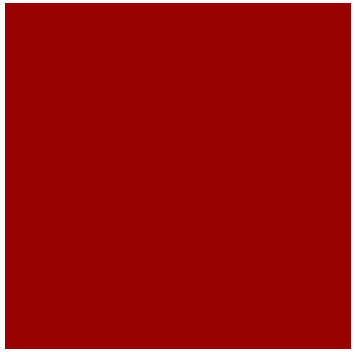
Aplicaciones Open Source

Semana06
28/04/18

Carlos A. Quinto Cáceres
pcsicqui@upc.edu.pe

Agenda

- JPA



Logros del día

- Al final de la clase, los alumnos podrán:
  - Hacer uso del API JPA para aprovechar las ventajas de orientación a objetos.

JPA

Java Persistence API

Antecedentes

- Cuando utilizamos Java, el desarrollo de las aplicaciones se realiza a través de objetos que tienen estado y comportamiento.
- Normalmente, éstas aplicaciones hacen uso de bases de datos relacionales para obtener datos, estos datos se almacenan en tablas.

¿Qué es JPA?

- JPA es un API desarrollada para Java que permite la persistencia de datos, esto quiere decir, que los datos se mantengan entre ejecuciones de la aplicación.
- Colección de clases y métodos para almacenar persistentemente datos.

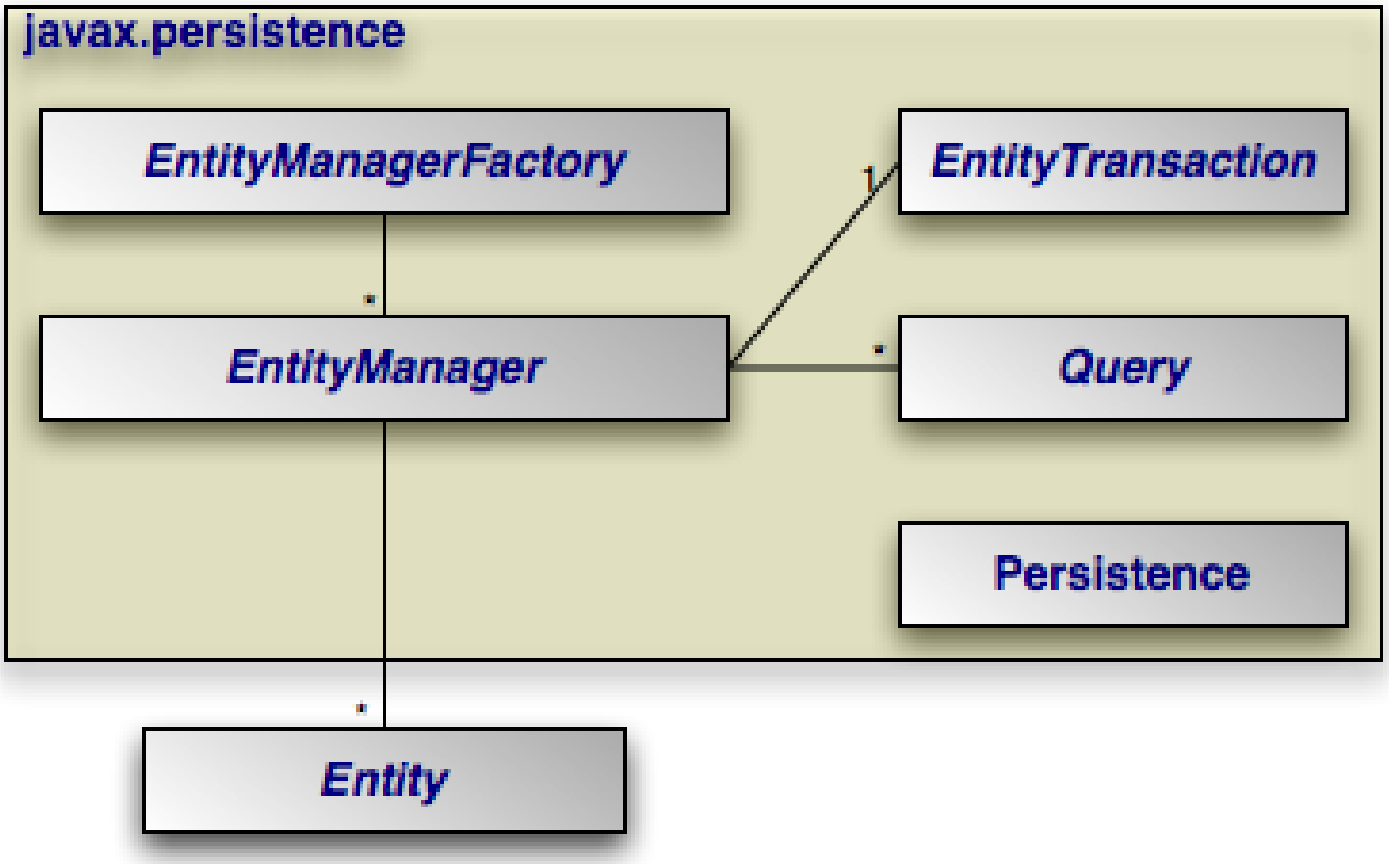
Concepto

- El objetivo del API, también, es no perder las ventajas de la orientación a objetos al interactuar con una base de datos relacional.
- Para lo cual nos sirve de abstracción sobre JDBC para realizar la conversión entre objetos y tablas.
- La conversión entre objetos y tablas se denomina ORM (Object Relational Mapping), la cual se puede realizar a través de xml o anotaciones.

Concepto

- La persistencia cubre 3 áreas:
  - La API definida en javax.persistence
  - El lenguaje de consultas JPQL
  - Metadatos objeto/relacional

Arquitectura JPA



Componentes - JPA

- **Entity**: objeto de dominio de persistencia. Representa una tabla en el modelo de datos relacional.
- **EntityManager**: interfaz de persistencia de aplicaciones. Contiene los métodos CRUD.
- **EntityManagerFactory**: clase que ayuda a crear objetos EntityManager.
- **EntityTransaction**: permite operaciones de datos persistentes de manera que agrupados formen una unidad transaccional. Todo el grupo de operaciones sincroniza su estado de persistencia o todos fallan en el intento.

Componentes - JPA

- **Query**: interfaz implementada por cada proveedor JPA para encontrar objetos persistentes según el criterio de búsqueda del proveedor.
- **Persistence**: clase que contiene métodos estáticos para obtener una instancia EntityManagerFactory de forma independiente del proveedor de la implementación.
- **Persistence.xml,** archivo XML donde se configura todo lo referente a la base de datos (proveedor, dialecto, pool de conexiones, etc.). Así como las clases entidad que serán gestionadas.

Implementaciones

- Dentro de las implementaciones del API, tenemos:
  - EclipseLink
  - Hibernate
  - Spring Data JPA
  - ObjectDB
  - TopLink
  - OpenJPA
  - Etc.

Anotaciones

@Entity	@ManyToOne	@JoinColumn
@Id	@OneToMany	@JoinTable
@GeneratedValue	@ManyToMany	
@Table		
@Column		

Entidad

- Es un objeto de persistencia, normalmente representa una tabla en una base de datos relacional y cada instancia de la entidad corresponde a una fila de la tabla.
- El estado de persistencia de una entidad es representada por campos o propiedades de persistencia.

Entidad

- Debe cumplir con:
  - La clase debe tener la anotación usada de javax.persistence.Entity.
  - Debe tener un constructor sin argumentos.
  - La clase no debe ser declarada como final.
  - De ser llamada remotamente debe implementar la interfaz Serializable.

Clase entidad

```java
@Entity
@Table(name="autores")
public class Autor {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private int id;

    @Column(name="nombres", length=45, nullable=false)
    private String nombre;

    @Column(name="apellidos", length=45, nullable=false)
    private String apellido;

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getApellido() {
        return apellido;
    }

    public void setApellido(String apellido) {
        this.apellido = apellido;
    }

    public void setId(int id) {
        this.id = id;
    }

    public int getId() {
        return id;
    }
}
```

Relaciones

Entidad Libro

```
//bi-directional many-to-many association to Autor
@ManyToMany
@JoinTable(
    name="libro_autor"
    , joinColumns={
        @JoinColumn(name="libro_id")
        }
    , inverseJoinColumns={
        @JoinColumn(name="autor_id")
        }
    )
private List<Autor> autores;
```

Entidad Autor

```
//bi-directional many-to-many association to Libro
@ManyToMany(mappedBy="autores")
private List<Libro> libros;
```

Relaciones

Entidad Libro

```
//bi-directional many-to-one association to Editorial
@ManyToOne
@JoinColumn(name="editorial_id")
private Editorial editoriale;
```

Entidad Editorial

```
//bi-directional many-to-one association to Libro
@OneToMany(mappedBy="editoriale")
private List<Libro> libros;
```

Entidad Libro

```
//bi-directional many-to-one association to Genero
@ManyToOne
private Genero genero;
```

Entidad Género

```
//bi-directional many-to-one association to Libro
@OneToMany(mappedBy="genero")
private List<Libro> libros;
```