Material descargable

Java Persistence API

egg

Persistencia en JPA con EntityManager

Ahora que comprendes cómo las anotaciones del ORM permiten mapear las tablas de la base de datos a objetos Java, llamados entidades, es importante saber cómo gestionar estas entidades en la base de datos.

Para realizar operaciones como guardar, editar, eliminar y consultar entidades, JPA utiliza la interfaz `EntityManager`. Esta interfaz es clave para manejar el ciclo de vida de las entidades definidas en una unidad de persistencia.

Operaciones del EntityManager:

- **Persistir**: Instancia nuevas entidades y las prepara para su almacenamiento en la base de datos.
- **Eliminar**: Borra entidades existentes de la base de datos.
- **Actualizar**: Modifica entidades ya almacenadas.
- **Consultar**: Recupera entidades desde la base de datos.
- **Controlar Transacciones**: Administra el ciclo de vida de las transacciones.

Configuración del EntityManager:

El EntityManager se configura utilizando las unidades de persistencia definidas en el archivo persistence.xml. La configuración básica se realiza con el siguiente código:

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("ViveroPU");
EntityManager em = emf.createEntityManager();
```

Explicacion del codigo

- **Persistence.createEntityManagerFactory("ViveroPU"):** Este método estático de la clase Persistence se utiliza para crear una instancia de EntityManagerFactory. El parámetro "ViveroPU" es el nombre de la unidad de persistencia definido en el archivo persistence.xml.
- **EntityManagerFactory:** Es una interfaz proporcionada por JPA que se encarga de crear instancias de EntityManager. Se configura a partir de la unidad de persistencia especificada en el archivo persistence.xml, que define los detalles de la conexión a la base de datos y otras configuraciones.
- **emf.createEntityManager():** Este método de la instancia EntityManagerFactory se utiliza para crear una instancia de EntityManager.
- **EntityManager:** Es la interfaz principal en JPA para interactuar con la base de datos. Permite realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en las entidades, así como gestionar las transacciones.

Operaciones con EntityManager

El EntityManager en JPA es crucial para realizar operaciones básicas de persistencia sobre las entidades en la base de datos. A través de él, puedes cargar, crear, actualizar y eliminar entidades. A continuación, exploramos algunos de los métodos más importantes del EntityManager y cómo se utilizan:

Persist()

El método `persist()` se utiliza para guardar una entidad en la base de datos. Persistir significa hacer que una instancia de una entidad se convierta en un registro almacenado de manera permanente en la base de datos. Esto no solo implica que los datos se almacenan, sino que también se pueden recuperar y utilizar posteriormente.

Importante: Antes de persistir una entidad, es fundamental comprender el concepto de transacciones, ya que la operación de persistencia debe ejecutarse dentro de una transacción. Una transacción es una serie de operaciones realizadas en la base de datos que se deben ejecutar de manera atómica, es decir, todas juntas o ninguna. No toda consulta a la base de datos se considera una transacción.

Ejemplo de Uso del Método `persist()`

```
// Crear una instancia de EntityManagerFactory con el nombre de la unidad de
persistencia
EntityManagerFactory emf = Persistence.createEntityManagerFactory("ViveroPU");

// Crear una instancia de EntityManager a partir de EntityManagerFactory
EntityManager em = emf.createEntityManager();

// Crear una nueva instancia de la entidad Alumno
Alumno alumno = new Alumno();
alumno.setNombre("Nahuel"); // Configurar el nombre del alumno

// Iniciar una transacción
em.getTransaction().begin();

// Persistir la entidad en la base de datos
em.persist(alumno);

// Confirmar la transacción para hacer permanentes los cambios
em.getTransaction().commit();

// Cerrar el EntityManager cuando ya no se necesite
em.close();
```

```
// Cerrar el EntityManagerFactory cuando ya no se necesite
emf.close();
```

Find()

El método `find()` se encarga de buscar y devolver una entidad en la base de datos a través de su clave primaria (`Id`). Para ello, necesita que le pases la clave y el tipo de entidad a buscar.

Ejemplo de Uso del Método `find()`

```
// Crear una instancia de EntityManagerFactory con el nombre de la unidad de persistencia
EntityManagerFactory emf = Persistence.createEntityManagerFactory("ViveroPU");

// Crear una instancia de EntityManager a partir de EntityManagerFactory
EntityManager em = emf.createEntityManager();

// Usar el método find para buscar una Persona con el id 123 en la base de datos
Persona persona = em.find(Persona.class, 123);

// De esta manera, obtendremos una instancia de Persona desde la base de datos para usar como deseemos

// Cerrar el EntityManager cuando ya no se necesite
em.close();

// Cerrar el EntityManagerFactory cuando ya no se necesite
emf.close();
```

Merge()

El método `merge()` se utiliza para actualizar una entidad en la base de datos. A diferencia del método `persist()`, que se usa para guardar nuevas entidades, `merge()` es útil para modificar entidades ya existentes. Si la entidad que estás

intentando actualizar no existe en la base de datos, `merge()` la insertará; si ya existe, actualizará sus valores.

Ejemplo de Uso del Método `merge()`:

```
// Crear una instancia de EntityManagerFactory con el nombre de la unidad de
persistencia
EntityManagerFactory emf = Persistence.createEntityManagerFactory("ViveroPU");

// Crear una instancia de EntityManager a partir de EntityManagerFactory
EntityManager em = emf.createEntityManager();

// Buscar el alumno con el id 1234 en la base de datos
Alumno alumno = em.find(Alumno.class, 1234);

// Asignar un nuevo nombre al alumno
alumno.setNombre("Francisco");

// Iniciar una transacción
em.getTransaction().begin();

// Actualizar la entidad en la base de datos
em.merge(alumno);

// Confirmar la transacción para hacer permanentes los cambios
em.getTransaction().commit();

// Cerrar el EntityManager cuando ya no se necesite
em.close();

// Cerrar el EntityManagerFactory cuando ya no se necesite
emf.close();
```

Remove()

El método `remove()` se utiliza para eliminar una entidad de la base de datos. Debes tener en cuenta que la entidad que deseas eliminar debe estar gestionada por el `EntityManager`, lo que significa que debe estar en el estado de persistencia.

Ejemplo de Uso del Método `remove()`:

```
// Crear una instancia de EntityManagerFactory con el nombre de la unidad de
persistencia
EntityManagerFactory emf = Persistence.createEntityManagerFactory("ViveroPU");

// Crear una instancia de EntityManager a partir de EntityManagerFactory
EntityManager em = emf.createEntityManager();
```

```java
// Buscar el alumno con el id 1234 en la base de datos
Alumno alumno = em.find(Alumno.class, 1234);

// Iniciar una transacción
em.getTransaction().begin();

// Eliminar la entidad de la base de datos
em.remove(alumno);

// Confirmar la transacción para hacer permanentes los cambios
em.getTransaction().commit();

// Cerrar el EntityManager cuando ya no se necesite
em.close();

// Cerrar el EntityManagerFactory cuando ya no se necesite
emf.close();
```

Cascadas en JPA

En JPA, el manejo de cascadas permite controlar cómo se propagan las operaciones de persistencia a las entidades asociadas. Esta capacidad es esencial para mantener la integridad referencial y simplificar la gestión de las relaciones entre entidades.

Repaso sobre Relaciones entre Entidades

Las relaciones entre entidades en JPA se configuran utilizando anotaciones como `@OneToMany`, `@ManyToOne`, `@OneToOne`, y `@ManyToMany`. Estas anotaciones definen cómo las entidades se conectan entre sí y establecen las reglas para la interacción entre ellas. Para más detalles sobre cómo configurar estas relaciones, consulta la documentación específica sobre relaciones entre entidades.

Propiedad Cascade

La propiedad `cascade` se usa para definir cómo se deben propagar las operaciones de persistencia (como `persist`, `remove`, `merge`, y `refresh`) desde una entidad a sus entidades asociadas. Puedes especificar estos comportamientos utilizando la anotación `@Cascade` en las relaciones entre entidades.

Tipos de Cascada

- **CascadeType.PERSIST**: Propaga la operación de persistencia a las entidades asociadas.
- **CascadeType.MERGE**: Propaga la operación de actualización a las entidades asociadas.

- **CascadeType.REMOVE**: Propaga la operación de eliminación a las entidades asociadas.
- **CascadeType.REFRESH**: Propaga la operación de refresco a las entidades asociadas.
- **CascadeType.ALL**: Propaga todas las operaciones (persist, merge, remove, refresh) a las entidades asociadas.

¿Qué hacer si NO quieres que ciertos cambios afecten a las entidades asociadas?

Si **no** deseas que los cambios en una entidad se propaguen a las entidades relacionadas (por ejemplo, que al eliminar un empleado no se elimine la oficina a la que pertenece), simplemente **no especifiques ningún tipo de cascada**.

Ejemplo de Uso de Cascada

Considera una entidad `Oficina` que tiene una relación uno a muchos con la entidad `Empleado`. Puedes definir el comportamiento de cascada en la entidad `Oficina` para que las operaciones realizadas sobre una oficina se propaguen automáticamente a sus empleados asociados:

```
@Entity
public class Oficina {

    @OneToMany(cascade = CascadeType.ALL, mappedBy = "oficina")
    private List<Empleado> empleados;

    // Otros atributos y métodos
}

@Entity
public class Empleado {

    @ManyToOne
    @JoinColumn(name = "oficina_id")
    private Oficina oficina;

    // Otros atributos y métodos
}
```

En este ejemplo:

- **Persistencia**: Al persistir una instancia de `Oficina`, todos los `Empleado` asociados también se persistirán automáticamente debido a `CascadeType.ALL`.
- **Eliminación**: Al eliminar una `Oficina`, todos los `Empleado` asociados también serán eliminados si se usa `CascadeType.REMOVE`.

Clases DAO

Para gestionar las operaciones CRUD de las entidades, es recomendable usar clases DAO (Data Access Object). Cada DAO se encarga de las operaciones de persistencia para una entidad específica y utiliza el `EntityManager` para interactuar con la base de datos.

```
public class OficinaDAO {

    private final EntityManagerFactory emf =
Persistence.createEntityManagerFactory("ViveroPU");
    private final EntityManager em = emf.createEntityManager();

    public void guardarOficina(Oficina oficina) {
        em.getTransaction().begin();
        em.persist(oficina);
        em.getTransaction().commit();
    }

    public Oficina buscarOficina(int id) {
        return em.find(Oficina.class, id);
    }

    public void actualizarOficina(Oficina oficina) {
        em.getTransaction().begin();
        em.merge(oficina);
        em.getTransaction().commit();
    }

    public void eliminarOficina(int id) {
        Oficina oficina = em.find(Oficina.class, id);
        if (oficina != null) {
            em.getTransaction().begin();
            em.remove(oficina);
            em.getTransaction().commit();
        }
    }


}
```