www.cas-training.com
C/ de la Basílica, 19. 5ª y 8ª planta. 28020 Madrid
Tfno. 91 553 61 62 - Fax 91 598 0346
info@cas-training.com

Cas Training

DESARROLLO DE APLICACIONES JAVA: COMPONENTES WEB Y APLICACIONES DE BBDD

ORGANIZADOR:

JULIO ARRANZ MEDINA

Java Persistence API - JPA

JPA – Introducción

•La API de Java Persistence (JPA) es un framework para el trabajo con bases de datos relacionales, compuesta por tres partes:

•El paquete: javax.persistence

•El lenguaje de consultas de Persistence (Java Persistence Query Language – JPQL)

•Los metadatos para los objetos y sus relaciones

JPA – Introducción

- Una entidad en JPA se refiere generalmente a una tabla de la base de datos
- Las instancias de una entidad corresponden a filas dentro de la tabla
- Generalmente las entidades se relacionan con otras entidades, estas relaciones se expresan mediante los metadatos
- Los metadatos de las relaciones que se dan entre los objetos se pueden definir en un archivo XML o empleando anotaciones en el archivo de cada clase

JPA – Introducción

•El lenguaje de consultas JPQL recuerda en su forma al SQL estándar, pero en lugar de operar sobre tablas de la base de datos lo hace sobre entidades

•POJO: Plain Old Java Object → clases que no extienden a ninguna otra ni implementan interfaces

•Por persistencia se entiende almacenar los datos de una aplicación en un medio físico como una base de datos, un archivo de texto, etc...

JPA – Introducción

•Cualquier entidad cuenta con tres características:

•Persistencia: los datos pueden almacenarse y recuperarse de una base de datos

•Identidad: cada instancia de la entidad es única

•Soporte Transaccional: las operaciones CRUD (Create, Read, Update, Delete) para esta entidad se realizan de forma transaccional

JPA – Anotaciones

•El siguiente cambio en el pseudocódigo convierte a esta clase POJO en una entidad:

```
@Entity
public class Post{
private Integer post_id;  private String post_title;
...
```

JPA – Identidad

- Ahora debemos incluir una nueva anotación que permita definir el campo que identificará cada instancia de la entidad que estamos definiendo:

```
@Id
private Integer post_id;
```

JPA – Convenciones

- Nombre de la tabla: MITABLA
- Nombre de la clase: MiTabla
- Nombre de las columnas: ATTR1, ATTR2, ATTR3
- Nombre de los atributos: attr1, attr2, attr3

JPA – ID's Automáticos

•Si queremos que los id's de nuestra entidad se generen de forma automática podemos incluir otra anotación:

- @Id(generate=GeneratorType.AUTO)

•Las otras estrategias son:

- GeneratorType.SEQUENCE
- GeneratorType.IDENTITY
- GeneratorType.TABLE_NAME

•AUTO es la mejor opción para portabilidad entre diferentes vendedores de bases de datos

JPA – ORM

employee
emp_id INT(10) NOT NULL (PK)
salary DECIMAL(8) NULL
dept_id INT(10) NULL

```
45      @Id
46      @Column(name = "emp_id", unique = true, nullable = false,
47              insertable = true, updatable = true)
48      public Integer getEmpId() {
49          return this.empId;
50      }
```

```
67      @Column(name = "salary", unique = false, nullable = true,
68              insertable = true, updatable = true, precision = 8, scale = 0)
69      public Long getSalary() {
70          return this.salary;
71      }
```

```
56      @ManyToOne(cascade = {}, fetch = FetchType.LAZY)
57      @JoinColumn(name = "dept_id", unique = false, nullable = true,
58              insertable = true, updatable = true)
59      public Department getDepartment() {
60          return this.department;
61      }
```

JPA – ORM

department
dept_id INT(10) NOT NULL (PK)
dept_desc VARCHAR(100) NULL

```
47      @Id
48      @Column(name = "dept_id", unique = true, nullable = false,
49              insertable = true, updatable = true)
50      public Integer getDeptId() {
51          return this.deptId;
52      }
```

```
58      @Column(name = "dept_desc", unique = false, nullable = true,
59              insertable = true, updatable = true, length = 100)
60      public String getDeptDesc() {
61          return this.deptDesc;
62      }
```

```
68      @OneToMany(cascade = { CascadeType.ALL }, fetch = FetchType.LAZY,
69              mappedBy = "department")
70      public Set<Employee> getEmployees() {
71          return this.employees;
72      }
```

JPA – Anotaciones

- @Column, permite definir una variable de clase enlazada con una columna

- Tiene diferentes propiedades

  - Updatable (boolean)
  - Nullable (updatable)
  - Length (int)

```
58      @Column(name = "dept_desc", unique = false, nullable = true,
59              insertable = true, updatable = true, length = 100)
60      public String getDeptDesc() {
61          return this.deptDesc;
62      }
```

JPA – Relaciones

- Hay cuatro de tipos de relaciones:
  - @OneToOne
  - @OneToMany
  - @ManyToOne
  - @ManyToMany

- En la mayoría de los casos, ponemos la anotación en el método getter de una propiedad podría ser suficiente

- En otras ocasiones necesitamos configurar algunos parámetros en

JPA – Relaciones

- Las dos entidades comparten el mismo valor de la clave primaria

```
16 @Entity
17 @Table(name = "employee", catalog = "test", uniqueConstraints = {})
18 public class Employee implements java.io.Serializable {
56     @ManyToOne(cascade = {}, fetch = FetchType.LAZY)
57     @JoinColumn(name = "dept_id", unique = false, nullable = true,
58             insertable = true, updatable = true)
59     public Department getDepartment() {
60         return this.department;
61     }
```

```
18 @Entity
19 @Table(name = "department", catalog = "test", uniqueConstraints = {})
20 public class Department implements java.io.Serializable {
68     @OneToMany(cascade = { CascadeType.ALL }, fetch = FetchType.LAZY,
69             mappedBy = "department")
70     public Set<Employee> getEmployees() {
71         return this.employees;
72     }
```

JPA – EntityManager

•La clase en pseudo-código es casi una entidad. Para que sea completamente una entidad es necesario asociarla con un EntityManager

•EntityManager: se refiere a una interfaz de la API que ofrece los servicios requeridos para trabajar con una entidad

JPA – EntityManager

•En J2SE se define un EntityManager de la siguiente manera (explícita):

```
EntityManagerFactory entityManagerFactory =
Persistence.createEntityManagerFactory(“PersistentUnitName”);

EntityManager eManager =  entityManagerFactory.createEntityManager();
```

JPA – EntityManager

•En J2EE se emplea una anotación en la clase de la entidad que se desea manejar y el contenedor (Servidor de Aplicaciones) se encarga del resto:

```
@Resource
private EntityManager entityManager;
```

JPA – Persistence Context

•Un contexto de persistencia administra un conjunto de entidades que a su vez es manejado por un EntityManager.

•El contexto de persistencia lleva el registro del estado y/o los cambios que puedan ocurrirle a una entidad.

•El EntityManager por su parte hace uso de los servicios que provee el contexto de persistencia para enviar (commit) o deshacer estos cambios.

JPA – Persistence Context

- Tan pronto como se crea un objeto EntityManager éste es asociado implícitamente con el contexto de persistencia correspondiente.

JPA – Entidades y Transacciones

•Todas las entidades cuentan con la propiedad de ser transaccionales. Todas sus operaciones CRUD se realizan en un contexto transaccional. Existen dos clases principales de transacciones: JTA y Resource-local

•Con las transacciones JTA el programador no debe preocuparse por nada, pero en las transacciones tipo Resource-local las validaciones corren por su cuenta y basado en los resultados de las mismas debe determinar si envía una transacción (commit) o por el contrario la anula (roll-back)

JPA – Operaciones con Entidades

```
26      public void create(){
27          // 1. get entity manager
28          EntityManagerFactory factory = Persistence
29                  .createEntityManagerFactory("JPAPU");
30          EntityManager entityMgr = factory.createEntityManager();
31          // 2. prepare entity
32          Department dept = new Department();
33          dept.setDeptId(1);
34          dept.setDeptDesc("test");
35          // 3. start transaction
36          entityMgr.getTransaction().begin();
37          // 4. save entity
38          entityMgr.persist(dept);
39          // 5. commit transaction
40          entityMgr.getTransaction().commit();
41          // 6. close connection
42          entityMgr.close();
43          factory.close();
44      }
```

JPA – Operaciones con Entidades

```
46      public void findById(){
47          // 1. get entity manager
48          EntityManagerFactory factory = Persistence
49                  .createEntityManagerFactory("JPAPU");
50          EntityManager entityMgr = factory.createEntityManager();
51          // 2. start transaction
52          entityMgr.getTransaction().begin();
53          // 3. find entity by id
54          Department result = entityMgr.find(Department.class, 1);
55          System.out.println(result.getDeptId()+", "+result.getDeptDesc());
56          // 4. commit transaction
57          entityMgr.getTransaction().commit();
58          // 5. close connection
59          entityMgr.close();
60          factory.close();
61      }
```

JPA – Operaciones con Entidades

```
81      public void delete(){
82          // 1. get entity manager
83          EntityManagerFactory factory = Persistence
84                  .createEntityManagerFactory("JPAPU");
85          EntityManager entityMgr = factory.createEntityManager();
86          // 2. start transaction
87          entityMgr.getTransaction().begin();
88          // 3. find entity by id
89          Department result = entityMgr.find(Department.class, 1);
90          // 4. delete entity
91          entityMgr.remove(result);
92          // 5. commit transaction
93          entityMgr.getTransaction().commit();
94          // 6. close connection
95          entityMgr.close();
96          factory.close();
97      }
```

JPA – Operaciones con Entidades

```
63      public void update(){
64          // 1. get entity manager
65          EntityManagerFactory factory = Persistence
66                  .createEntityManagerFactory("JPAPU");
67          EntityManager entityMgr = factory.createEntityManager();
68          // 2. start transaction
69          entityMgr.getTransaction().begin();
70          // 3. find entity by id
71          Department result = entityMgr.find(Department.class, 1);
72          // 4. give new value
73          result.setDeptDesc("RD Center");
74          // 5. commit transaction
75          entityMgr.getTransaction().commit();
76          // 6. close connection
77          entityMgr.close();
78          factory.close();
79      }
```

JPA – Operaciones con Entidades

•flush() es el método que sincroniza los cambios realizados sobre una entidad con la base de datos.

•refresh() es el método que devuelve los atributos de un objeto al último estado recuperado desde la base de datos.

JPA – Operaciones con Entidades

•Es claro que contar con la posibilidad de recuperar instancias de una entidad, exclusivamente utilizando su llave primaria, no es adecuado en prácticamente ningún ambiente. Para superar este obstáculo es necesario emplear JPQL, un lenguaje de consultados basado en SQL y orientado a objetos

•Existen 2 tipos de consultas diferentes: estáticas y dinámicas

JPA - JPQL

•Una consulta estática se define empleando XML justo antes de la definición de la clase de la entidad. Este tipo de consultas llevan un nombre gracias al cual otros componentes de la misma unidad de persistencia pueden usarla:

```
@NamedQuery(name=”PostEntity.findAll” query= “SELECT M FROM POSTENTITY”)
@Entity
class PostEntity{
……
}
```

JPA - JPQL

•Luego una vez definida, una consulta estática puede utilizarse de la siguiente manera:

```
Query findAllQuery =  entityManager.createNamedQuery(“PostEntity.findAll”);
```

JPA - JPQL

•También es posible definir arreglos de consultas estáticas para usarlos luego:

```
@NamedQueries( {
@NamedQuery(name = “Post.selectAllQuery” query = “SELECT  M FROM POSTENTITY”),
@NamedQuery(name = “Post.deleteAllQuery” query = “DELETE  M FROM POSTENTITY”)
} )
```

JPA - JPQL

•Una consulta dinámica se define dentro del código de la entidad, empleando código SQL corriente. Aunque a primera vista puede resultar más simple que definir consultas estáticas, su uso puede disminuir en gran medida el desempeño de la aplicación ya que todo el mapeo de objetos / base de datos ocurre en tiempo de ejecución:

```
Query singleSelectQuery = entityManager.createQuery( “SELECT M FROM
MOBILEENTITY WHERE M.IMEI = ‘ABC-123’”);
```

JPA - JPQL

•Una consulta puede traer un resultado, en cuyo caso se accede a el de la siguiente manera:

```
PostEntity postObj = singleSelectQuery.getSingleResult();
```

•O puede traer múltiples resultados en cuyo caso es necesario convertir (cast) el resultado a un objeto tipo Lista:

```
List posts = (List)multipleSelect.getResultList();
```

JPA - JPQL

•Tanto las consultas dinámicas como las consultas estáticas pueden emplear parámetros dinámicos, los cuales son enviados en tiempo de ejecución:

```
@NamedQuery(
name = “Post.findByTitle”
query = “SELECT P FROM POSTENTITY WHERE P.TITLE LIKE
'%:keyword%‘”
)
```

JPA - JPQL

•Y luego en el código podría llamarse pasando el parámetro deseado:

```
Query namedQuery =  entityManager.createNamedQuery(“Post.findByTitle”);

namedQuery.setParameter(“keyword”, entradaUsuario);
```

JPA - JPQL

•Si una consulta devuelve, por ejemplo, 1000 resultados. No es una buena idea devolver todo el conjunto de resultados al usuario de una sola vez, en lugar de esto deben dividirse los resultados en páginas de menor tamaño e ir retornando una página a la vez de acuerdo con las solicitudes del usuario

JPA - JPQL

```
int maxRecords = 10;  int startPosition = 0;
String queryString = “SELECT M FROM MOBILEENTITY”;  while(true){
Query selectQuery = entityManager.createQuery(queryString);
selectQuery.setMaxResults(maxRecords);  selectQuery.setFirstResult(startPosition);
……..
}
```

JPA - JPQL

```
List mobiles = entityManager.getResultList(queryString);  if (mobiles.isEmpty()){
break;
}

//Process the mobile entities.  process(mobiles);  entityManager.clear();
startPosition = startPosition + mobiles.size();
……..
```

JPA – Unidad de Persistencia

•Unidad de Persistencia: una manera de categorizar un conjunto de entidades que trabajaran de forma conjunta y compartirán una misma configuración

•Las unidades de persistencia se definen empleando archivos XML y deben asociarse a los objetos EntityManager:

```
@PersistentUnit(unitName = “MyPostPersistentUnit”) private EntityManager entityManager;
```

JPA – Unidad de Persistencia

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <persistence xmlns="http://java.sun.com/xml/ns/persistence"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
5      http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd" version="1.0">
6
7      <persistence-unit name="JPAPU" transaction-type="RESOURCE_LOCAL">
8          <provider>org.hibernate.ejb.HibernatePersistence</provider>
9          <class>ext.entity.Employee</class>
10         <class>ext.entity.Department</class>
11         <properties>
12             <property name="hibernate.connection.driver_class"
13                 value="com.mysql.jdbc.Driver" />
14             <property name="hibernate.connection.url"
15                 value="jdbc:mysql://localhost:3306/test" />
16             <property name="hibernate.connection.username" value="root" />
17             <property name="hibernate.connection.password"
18                 value="albert" />
19         </properties>
20     </persistence-unit>
21
22 </persistence>
```

EntityManagerFactory Name

Entity classes

JDBC Driver

JDBC URL

User name

password