Java Persistence API

- Sesión 5: Entity manager

© 2008 Depto. Ciencia de la Computación e IA

Índice

- Contextos de persistencia
- Entity managers
- Operaciones del entity manager
- Sincronización con la BD

Repaso de conceptos

- **Unidad de persistencia**: configuración concreta de un conjunto de entidades persistentes en un sistema de base de datos concreto, definida en el fichero `persistence.xml`. Existe de forma estática, definido por la BD.
- **Contexto de persistencia**: conjunto de instancias de entidad gestionadas. Sus clases entidad son las definidas en la unidad de persistencia. Se crea al crear un entity manager y se desconecta al cerrarlo.
- **Entity manager**: responsable de la gestión del contexto de persistencia. Sincroniza con la BD las entidades incluidas en él. Se crea de forma dinámica.

© 2008 Depto. Ciencia de la Computación e IA

Responsabilidades del entity manager

- El entity manager es responsable de mantener la sincronización entre la BD y el contexto de persistencia. El contexto de persistencia define una caché que el entity manager debe volcar (*flush*) en la BD.
- Simplificando, el funcionamiento del entity manager es el siguiente:
  - 1. Si la aplicación solicita una entidad (mediante un find, o accediendo a un atributo de otra entidad en una relación), se comprueba si ya se encuentra en el contexto de persistencia. Si no se ha recuperado previamente, se obtiene la instancia de la entidad de la base de datos.
  - 2. La aplicación utiliza las instancias del contexto de persistencia, accediendo a sus atributos y (posiblemente) modificándolos. Todas las modificaciones se realizan en la memoria, en el contexto de persistencia.
  - 3. En un momento dado (cuando termina la transacción, se hace una consulta o se hace una llamada al método flush) el entity manager comprueba qué entidades han sido modificadas y vuelca los cambios a la base de datos.

© 2008 Depto. Ciencia de la Computación e IA

Problemas manejando el contexto

```
em.getTransaction().begin();
Autor autor = em.find(Autor.class, "kirai");

System.out.println(autor.getNombre() + " ha escrito " +
                   autor.getMensajes().size() +
                   " mensajes");

Mensaje mens = new Mensaje("Nuevo mensaje");
mensaje.setAutor(autor);
em.persist(mens);
em.getTransaction().commit();

System.out.println(autor.getNombre() + " ha escrito " +
                   autor.getMensajes().size() +
                   " mensajes");
```

La entidad ya está en el contexto de persistencia y no se vuelve a consultar la BD

Posibles soluciones

- Dos posibles soluciones
  - 1. La aplicación es la responsable de mantener el contexto de persistencia actualizado, realizando manualmente los cambios
  - 2. Antes de hacer cualquier consulta refrescamos la entidad

```
em.getTransaction().begin();
// ... añado un mensaje al autor
em.getTransaction().commit();

autor.getMensajes().add(mensaje);
System.out.println(autor.getNombre()
                   + " ha escrito " +
      autor.getMensajes().size() +
                     " mensajes");
```

```
em.getTransaction().begin();
// ... añado un mensaje al autor
em.getTransaction().commit();

em.refresh(autor);
System.out.println(autor.getNombre()
                   + " ha escrito " +
      autor.getMensajes().size() +
                     " mensajes");
```


© 2008 Depto. Ciencia de la Computación e IA

Creación de un `EntityManager`

- Dos posibles opciones
- En entornos Java EE con servidores de aplicaciones, se crea de forma declarativa con una anotación. Se denominan entity managers gestionados por el contenedor.
- En entornos Java SE (y aplicaciones web) se crea con una llamada a un `EntityManagerFactory` obtenido de un método estático de la clase `Persistence`. Se denomina entity manager gestionado por la aplicación.

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

Tipos de entity managers

- La creación de entity managers es barata y normalmente se implementa con un pool
- Dos formas de gestionar los entity managers que se corresponden con los tipos de entity managers gestionados por el contenedor
  - Entity manager por operación. Cada operación del DAO abre y cierra su propio entity manager.
  - Entity manager extendido. Un mismo entity manager es compartido por varias operaciones de varios DAO
- Esta decisión la tendremos que tomar también con los EJB de sesión que implementan operaciones de lógica de negocio

© 2008 Depto. Ciencia de la Computación e IA

Entity manager por operación

- Cada operación abre y cierra su propio entity manager

```
public class EmpleadoDAO {
   EntityManagerFactory emf;

   public EmpleadoDAO() {
      emf = Persistence.createEntityManagerFactory("SimpleJPA");
   }

   // ... Operaciones del DAO ...

   public Empleado subeSueldoEmpleado(int id, long aumento) {
      Empleado emp = null;
      EntityManager em = emf.createEntityManager();
      em.getTransaction().begin();
      Empleado emp = em.find(Empleado.class, id);
      if (aumento > 0)
         emp.setSueldo(emp.getSueldo + aumento);
      else
        // error
      em.getTransaction().commit();
      em.close();
      return empleado;
   }
}
```

Ejemplo de utilización

- Sencillez: la aplicación no tiene que crear entity managers
- El resultado es una entidad desconectada (*detached entity*) ya que se ha cerrado el contexto de persistencia (es equivalente a un Transfer Object)

```
EmpleadoDAO empDAO = new EmpleadoDAO();
Empleado emp = empDAO.subSueldoEmpleado(122,2000);
System.out.println("El nuevo sueldo de "
                   + emp.getNombre() + " es "
                   + emp.getSueldo());
```

Entity manager extendido

- Varias operaciones comparten el mismo entity manager

```
public class EmpleadoEAO {
   EntityManager em;

   public EmpleadoEAO(EntityManager em) {
      this.em = em;
   }

   // ... Operaciones del EAO ...

   public Empleado findEmpleado(int id) {
      return em.find(Empleado.class, id);
   }

   public Empleado subeSueldoEmpleado(Empleado emp, long aumento) {
      if (aumento > 0)
         emp.setSueldo(emp.getSueldo + aumento);
      else
         // error
      return empleado;
   }
}
```

Ejemplo de funcionamiento

- La creación y cerrado de entity managers es responsabilidad de la aplicación
- Más flexible:
  - se pueden realizar varias operaciones en el mismo entity manager
  - la aplicación es responsable de la desconexión de las entidades

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("SimpleJPA");
EntityManager em = emf.createEntityManager();
EmpleadoDAO empDAO = new EmpleadoDAO(em);

em.getTransaction().begin();
Empleado emp = empDAO.findEmpleado(122);
emp = empDAO.subeSueldoEmpleado(2000);
em.getTransaction().commit();
em.close();

System.out.println("El nuevo sueldo de " + emp.getNombre() + " es "
                    + emp.getSueldo());
```

Interfaz EntityManager

```
public interface EntityManager {
   void clear();
   void close();
   <T> T find(Class<T> entityClass, Object primaryKey)
   FlushModeType getFlushMode();
   <T> T getReference(Class<T> entityClass, Object primaryKey);
   EntityTransaction getTransaction();
   boolean isOpen();
   <T> T merge(T entity);
   void persist(Object entity);
   void refresh(Object entity);
   void remove(Object entity);
   void setFlushMode(FlushModeType flushMode);
}
```

`getReference()`

- Obtiene una referencia a una entidad sin recuperar sus datos de la BD
- No se realiza un `SELECT`, sólo se guarda en el contexto de persistencia la clave primaria de la entidad
- Un problema: no devuelve `null` si la entidad no existe

```
Departamento dept = em.getReference(Departamento.class, 30);
Empleado emp = new Empleado();
emp.setId(53);
emp.setNombre("Pedro");
emp.setDepartamento(dept);
dept.getEmpleados().add(emp);
em.persist(emp);
```

merge()

- Incorpora una entidad desconectada a un contexto de persistencia.
- Similar a persist(), con la diferencia de que devuelve la entidad gestionada.

```
Empleado emp = em.merge(empleado);
```


© 2008 Depto. Ciencia de la Computación e IA

Borrado de entidades

- Es responsabilidad de la aplicación el mantenimiento de la integridad de las relaciones en el contexto de persistencia
- Es conveniente encapsular estas operaciones en los DAO

```
Empleado emp = em.find(Empleado.class, empId);
Despacho desp = emp.getDespacho();
emp.setDespacho(null);
em.remove(desp);
```

Operaciones en cascada

- En las relaciones se pueden definir operaciones en cascada, de forma que si realizamos una operación sobre una entidad, se realiza esa misma operación en todas las relacionadas
- Se definen añadiendo el elemento `cascade=CascadeType` a la anotación
- `CascadeType` es un tipo enumerado que puede tener los valores `PERSIST`, `REFRESH`, `REMOVE`, `MERGE` y `ALL`.

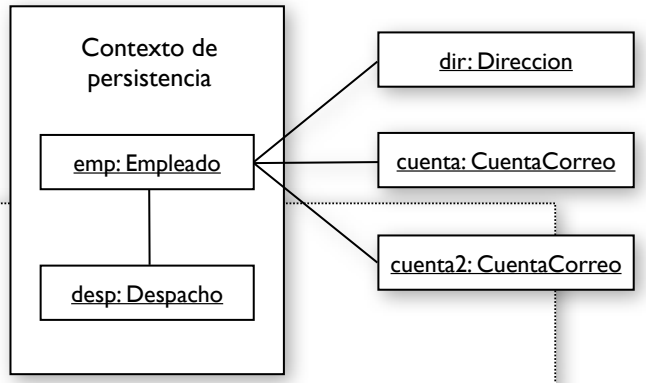
```
@Entity
public class Empleado {
   @OneToOne(cascade=CascadeType.PERSIST)
   Despacho despacho;
   @OneToMany(mappedBy="empleado",
              cascade={CascadeType.PERSIST, CascadeType.REMOVE})
   Collection<CuentaCorreo> cuentasCorreo;
   // ...
}
```

Sincronización de la BD

- Un volcado del contexto de persistencia contiene tres tipos de elementos:
  - Entidades nuevas que hay que hacer persistentes
  - Entidades modificadas que deben ser actualizadas
  - Entidades borradas que deben eliminarse de la BD
- Al hacer persistente o borrar una entidad el entity manager debe detectar las operaciones en cascada asociadas
- Se lanza la excepción IllegalStateException cuando se hace persistente una instancia y alguna otra instancia con la que está relacionada no está gestionada por el entity manager
- La excepción a esta última regla sucede cuando la relación existe una relación one-to-one y many-to-one, en este caso se guarda en la columna la clave primaria de la otra entidad

Ejemplo

- ¿Qué sucede cuando hacemos persistente la instancia emp?

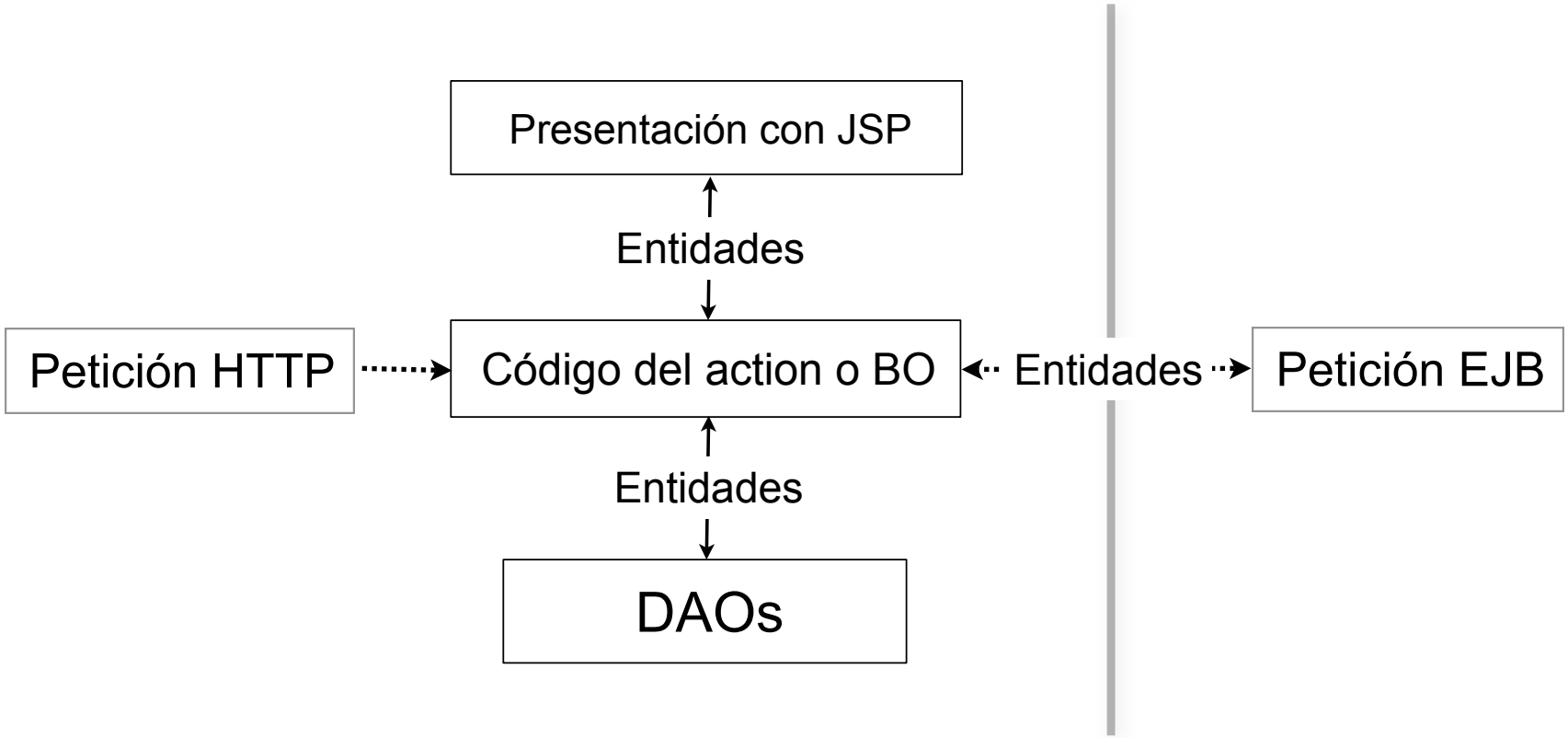



```
@Entity
public class Empleado {
   // ...
   @OneToOne
   Despacho despacho;
   @OneToMany(mappedBy="empleado", cascade=CascadeType.PERSIST)
   Collection<CuentaCorreo> cuentasCorreo;
   @ManyToOne
   Direccion direccion;
   // ...
}
```

Arquitectura JPA

- ¿Cómo incluir JPA en una arquitectura por capas?
- ¿Cómo incluir JPA en una aplicación web basada en peticiones?
- Las preguntas se podrían concretar básicamente en un único problema: ¿cuándo abrir y cerrar el entity manager?
- El problema principal es cómo pasar las entidades obtenidas en la capa de negocio a la capa de presentación ¿desconectadas o conectadas?

Planteamiento del problema



© 2008 Depto. Ciencia de la Computación e IA

Servlet

```
public class EmployeeServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
            throws ServletException, IOException {

        EntityManagerFactory emf = Persistence
                                        .createEntityManagerFactory("SimpleJPA");
        EntityManager em = emf.createEntityManager();
        EmpleadoDAO empService = new EmpleadoDAO(em);

        List emps = empService.findAll();
        request.setAttribute("empleados", emps);
        getServletContext().getRequestDispatcher("/listaEmpleados.jsp")
                           .forward(request, response);

    }
}
```

¿Dónde cerramos `em`?

Presentación: página JSP

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
  <head>
    <title>Lista empleados</title>
  </head>
              <body>
    <table>
      <thead>
        <tr>
          <th>Nombre</th>
          <th>Departamento</th>
        </tr>
     </thead>
      <tbody>
        <c:forEach items="${empleados}" var="emp">
          <tr>
            <td><c:out value="${emp.nombre}"/></td>
            <td><c:out value="${emp.departmento.nombre}"/></td>
          </tr>
        </c:forEach>
      </tbody>
    </table>
  </body>
</html>
```

Problemas con el lazy loading

Solución 1

- Pasamos entidades gestiondas

```
public class EmployeeServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
            throws ServletException, IOException {

        EntityManagerFactory emf = Persistence
                                        .createEntityManagerFactory("SimpleJPA");
        EntityManager em = emf.createEntityManager();
        EmpleadoDAO empService = new EmpleadoDAO(em);

        List emps = empService.findAll();
        request.setAttribute("empleados", emps);
        getServletContext().getRequestDispatcher("/listaEmpleados.jsp")
                           .forward(request, response);
        em.close();
    }
}
```

Solución 2

- Pasamos entidades desconectadas y recuperamos previamente los datos que vamos a mostrar

```
public class EmployeeServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
            throws ServletException, IOException {
        //...
        List emps = empService.findAll();
        for (Employee emp : emps) {
          Departmento dept = emp.getDepartmento();
          if (dept != null) {
              dept.getNombre();
          }
        }
        em.close();
        request.setAttribute("empleados", emps);
        getServletContext().getRequestDispatcher("/listaEmpleados.jsp")
                           .forward(request, response);
    }
}
```

Ventajas e inconvenientes

- Solución 1 es más flexible ya que podemos cambiar la presentación fácilmente
- Se puede mejorar incluso haciendo que el entity manager se abra y se cierra al recibir y terminar la petición de forma que no tenemos que escribir el código en la propia petición (Spring)
- Solución 2 es más segura ya que respetamos la estructura de capas y no permitimos que la presentación cambie los datos de la BD

¿Preguntas?