Java Persistence API

- Sesión 8: De JPA a Hibernate

Índice

- Introducción
- Configuración de Hibernate
- Mapeo de entidades
- Mapeo de relaciones
- Relaciones de herencia
- Arquitectura Hibernate

Hibernate

- Nace como un proyecto opensource liderado por Gavin King
- Adquirido por JBoss e incorporado en su servidor de aplicaciones, ahora parte de RedHat.
- Amplia comunidad de usuarios
- Origen de JPA: ahora Hibernate es una implementación de JPA
- Proyectos antiguos: Hibernate
- Proyectos nuevos: JPA

JPA e Hibernate

- Los conceptos son idénticos, cambian los nombres de las clases y métodos
  - EntityManagerFactory --> SessionFactory
  - EntityManager --> Session
  - em.persist(autor) --> session.save(autor)
  - em.find() --> session.get()
- Para definir el mapeado se utilizan ficheros de configuración XML
  - Configuración más proclive a errores

Hola mundo

```
SessionFactory sessionFactory =
   new Configuration().configure().buildSessionFactory();
Session session = sessionFactory.getCurrentSession();
...
session.beginTransaction();
autor = (Autor) session.get(Autor.class, autorStr);
if (autor == null) {
   autor = new Autor();
   autor.setNombre(autorStr);
   autor.setCorreo(autorStr + "@ua.es");
   session.save(autor);
}
mensaje = new Mensaje();
mensaje.setAutor(autor);
mensaje.setTexto(mensStr);
session.save(mensaje);
Collection<Mensaje> mensajes = autor.getMensajes();
mensajes.add(mensaje);
session.getTransaction().commit();
```

Fichero de configuración

```
<hibernate-configuration>
   <session-factory>
      <property name="connection.url">jdbc:mysql://localhost/jpa</property>
      <property name="connection.username">root</property>
      <property name="connection.password">root</property>
      <property name="connection.driver_class">com.mysql.jdbc.Driver
         </property>
      <property name="dialect">org.hibernate.dialect.MySQLDialect
         </property>
      <property name="current_session_context_class">thread</property>
      <property name="transaction.factory_class">
         org.hibernate.transaction.JDBCTransactionFactory</property>
      <property name="cache.provider_class">
         org.hibernate.cache.NoCacheProvider</property>
      <property name="hibernate.show_sql">true</property>
      <property name="hbm2ddl.auto">update</property>
      <mapping resource="mappings.hbm.xml"/>
   </session-factory>
</hibernate-configuration>
```

Entidades

```
public class Autor {
   private String nombre;
   private String correo;
   private Integer edad;
   private Set<Mensaje> mensajes = new HashSet<Mensaje>();

   public Autor() {}
   public String getNombre() {return nombre;}
   public void setNombre(String nombre) {this.nombre = nombre;}
   public String getCorreo() {return correo;}
   public void setCorreo(String correo) {this.correo = correo;}
   public Set<Mensaje> getMensajes() {return mensajes;}
   public void setMensajes(Set<Mensaje> mensajes) {
      this.mensajes = mensajes;}
}
```

Mapeo con XML

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
   <class name="entity.Autor" table="Autor">
      <id name="nombre" type="string" column="nombre"/>
         <property name="correo" type="string" column="correo"/>
         <property name="edad" type="integer" column="edad"/>
         <set name="mensajes" inverse="true">
            <key column="autor_nombre"/>
               <one-to-many class="entity.Mensaje"/>
         </set>
   </class>

   <class name="entity.Mensaje" table="Mensaje">
      ...
   </class>
</hibernate-mapping>
```

Tipos de valores en el mapeo

- `integer`, `long`, `short`, `float`, `double`, `character`, `byte`, `boolean`, `yes_no`, `true_false`: estos mapeos transforman los tipos básicos de Java (y sus clases wrappers) en los tipos SQL apropiados (dependientes del vendedor). Los mapeos `boolean`, `yes_no` y `true_false` son distintos nombres para booleanos
- `string`: mapeo de `java.lang.String` a `VARCHAR` (o `VARCHAR2` de Oracle).
- `date`, `time`, `timestamp`: mapeo de la clase `java.util.Date` y sus subclases a los tipos SQL `DATE`,`TIME` y `TIMESTAMP` (o equivalentes).
- `calendar`, `calendar_date`: mapeo de `java.util.Calendar` a los tipos SQL `TIMESTAMP` y `DATE` (o equivalentes).
- `locale`, `timezone`, `currency`: mapeo de `java.util.Locale`,`java.util.Timezone` y `java.util.Currency` a `VARCHAR` (o `VARCHAR2` de Oracle). Las instancias de `Locale` y `Currency` se mapean a sus códigos ISO. Las instancias de `TimeZone` se mapean a su ID.
- `binary`: mapeo de array de bytes al tipo binario SQL apropiado.
- `text`: mapeo de cadenas largas de Java al tipo SQL `TEXT` o `CLOB`.
- `serializable`: mapeo de un tipo Java serializable a un tipo binario SQL.

Igualdad

- En el contexto de persistencia funciona correctamente la igualdad de referencia, pero en entidades desconectadas no:

```
Autor autor1 = (Autor) session.get(Autor.class, autorStr);
Autor autor2 = (Autor) session.get(Autor.class, autorStr);
if (autor1 == autor2) {
   System.out.println("Iguales en referencia");
}
if (autor1.equals(autor2) {
   System.out.println("Iguales en contenido");
}
```

Entidades gestionadas

```
Collection<Mensaje> mensajes = Collection<Mensaje>
   autorEAO.allMensajesAutor(autor);
Mensaje mejorMensaje = mensajeEAO.findMejorMensaje();
if (mensajes.contains(mejorMensaje)) {
    autor.setMejor(true);
    ...
}
```

Entidades desconectadas

Equals y hashMap

```
public class Cat {
    ...
    public boolean equals(Object other) {
       if (this == other) return true;
       if ( !(other instanceof Cat) ) return false;
       final Cat cat = (Cat) other;
       if ( !cat.getId().equals( getId() ) ) return false;
       if ( !cat.getMother().equals( getMother() ) ) return false;
           return true;
    }

    public int hashCode() {
       int result;
       result = getMother().hashCode();
       result = 29 * result + getLitterId();
       return result;
    }
}
```

Componentes

- Equivalentes a los @Embedded de JPA

```
<hibernate-mapping>
   <class name="entity.Persona" table="Persona">
   <id name="Key" column="pid" type="string">
      <generator class="uuid"/>
   </id>
   <property name="cumpleanyos" type="date"/>
   <component name="Nombre" class="entity.Nombre">
      <property name="nombre"/>
      <property name="primerApellido"/>
      <property name="segundoApellido"/>
   </component>
   </class>
</hibernate-mapping>
```

```
public class Persona {
   private String key;
   private java.util.Date cump
   private Nombre nombre;
   ...
}

public class Nombre {
   String nombre;
   String primerApellido;
   String segundoApellido;
   ...
}
```

Mapeo de relaciones

- Similares a JPA:
  - uno-a-muchos: con claves ajenas en un lado de la relación (el 'propietario' de la relación)
  - muchos-a-muchos: tabla join adicional
- Muchas otras configuraciones
  - Documentación de Hibernate

Uno-a-muchos Autor-Mensajes

```
public class Autor {
   private String nombre;
   private String correo;
   private Integer edad;
   private Set<Mensaje> mensajes = new HashSet<Mensaje>();
   ...
}
```

```
public class Mensaje {
   private long id;
   private String texto;
   private Date creado;
   Autor autor;
   ...
}
```

Fichero de mapeo

```
<hibernate-mapping>
   <class name="entity.Autor" table="Autor">
      <id name="nombre" type="string" column="nombre"/>
         <property name="correo" type="string" column="correo"/>
         <property name="edad" type="integer" column="edad"/>
         <set name="mensajes" inverse="true">
            <key column="autor_nombre"/>
               <one-to-many class="entity.Mensaje"/>
         </set>
   </class>

   <class name="entity.Mensaje" table="Mensaje">
      <id name="id" column="id">
         <generator class="native"/>
      </id>
      <property name="texto" type="string" column="texto" not-null="true"/>
      <property name="creado" type="date" column="creado"/>
      <many-to-one name="autor"
         column="autor_nombre"
         class="entity.Autor"
         not-null="true"/>
    </class>
</hibernate-mapping>
```

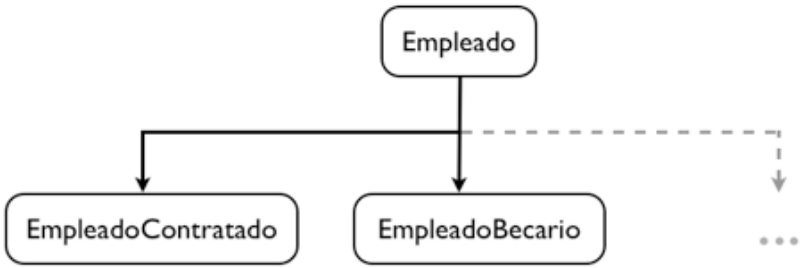
Relación muchos-a-muchos unidireccional

```
<class name="entity.Autor" table="Autor">
   <id name="nombre" type="string" column="nombre"/>
   <property name="correo" type="string" column="correo"/>
   <property name="edad" type="integer" column="edad"/>

      <set name="mensajes" table="autor-mensaje">
      <key column="autorId"/>
         <many-to-many class="mensaje" column="mensajeId"/>
      </set>
   </class>

   <class name="Mensaje">
      <id name="mensajeId">
         <generator class="sequence"/>
      </id>
      <property name="texto"/>
   </class>
</hibernate-mapping>
```

Relaciones de herencia



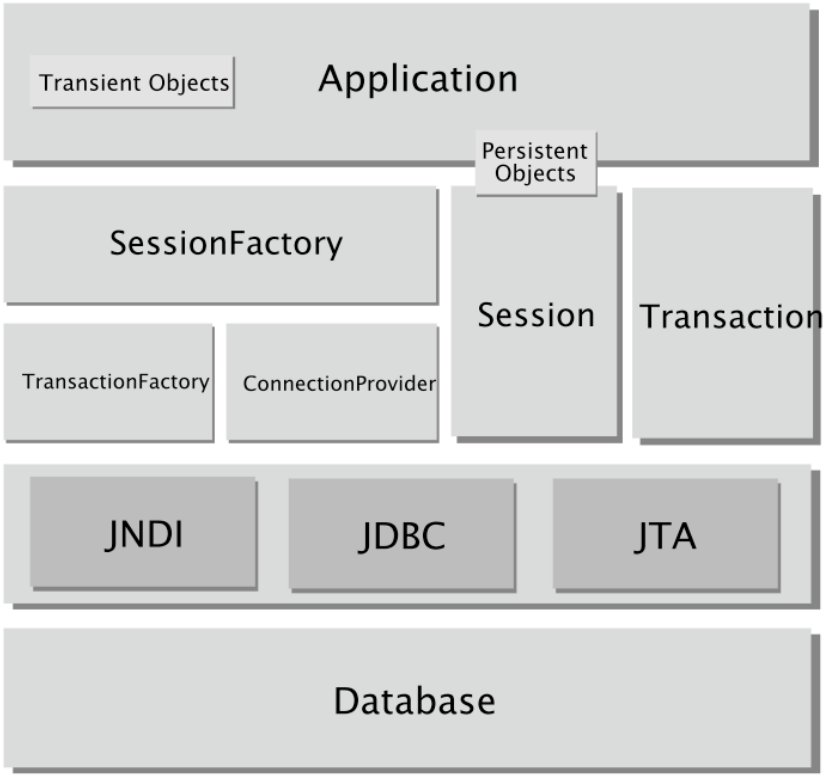
ID	Nombre	Salario	Tipo	PlanPensiones	SeguroMedico
1	Antonio	2.300	Contrato	230	NULL
2	Juan	1.200	Beca	NULL	150
3	María	2.400	Contrato	240	NULL

Mapeo de la relación

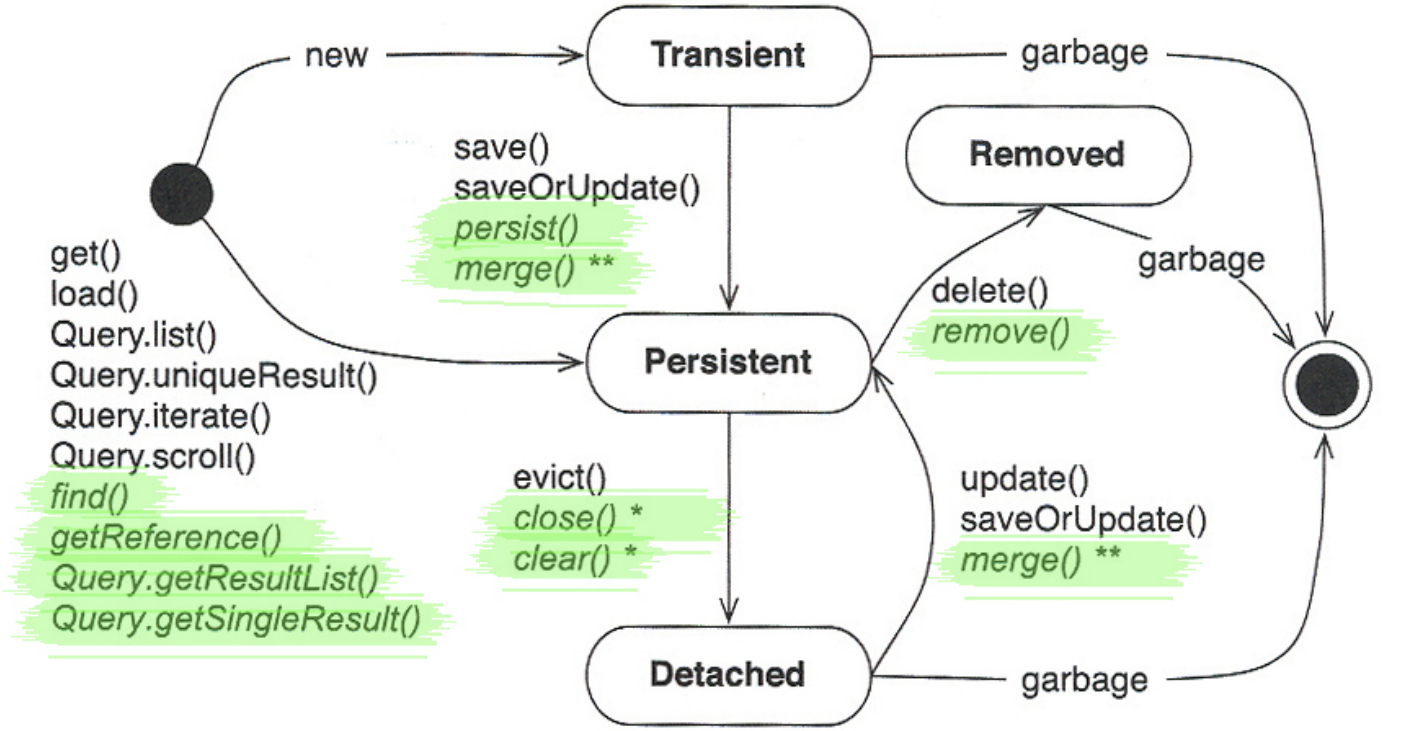
```
<class name="Empleado" table="EMPLEADO">
    <id name="empleadoId" type="long" column="EMPLEADO_ID">
        <generator class="native"/>
    </id>
    <discriminator column="TIPO" type="string"/>
    <property name="nombre" type="string" column="NOMBRE"/>
    ...
    <subclass name="EmpleadoContratado" discriminator-value="contrato">
        <property name="planPensiones" type="long" column="PLAN_PENSIONES"/>
    </subclass>
    <subclass name="EmpleadoBecario" discriminator-value="beca">
        <property name="seguroMedico" type="long" column="SEGURO_MEDICO"/>
    </subclass>
</class>
```

Arquitectura Hibernate

- Idéntica a la de JPA



Ciclo de vida de una entidad



¿Preguntas?