JPA

Analista Programador

Francisco Villegas - www.franciscovillegas.net - 2013

Definición

JPA (**J**ava **P**ersistence **A**PI) permite definir e implementar mapeos Objecto-Relacional (**O**bject/**R**elational **M**apping - ORM)

Con ORM podemos manejar datos en BD relacionales mediante objetos.

Beneficios del uso de ORM:

- Simplifica el desarrollo de aplicaciones basadas en BD relacionales (todo son objetos)
- Ofrece persistencia de objetos de forma transparente
- Automatiza y unifica las tareas de mantenimiento de datos relacionales
- Evita que el código sea dependiente de características específicas de un gestor de BD concreto

Elementos definidos por JPA
Mapeo O/R

- Table ~ clase de entidades JPA
- Tupla ~ instancia de una entidad JPA
- Columna ~ atributo de una instancia

JPA - EntityManager

- Define el ciclo de vida
- Gestiona los detalles de la conexión con la DB (url, credenciales, drivers JDBC).

JPA - JPL

Especificación del lenguaje de consulta JPQL (**J**ava **P**ersistence **Q**uery **L**anguage)

- Similar a SQL: usa entidades en lugar de tablas al definir las consultas

Entidades JPA

Las tablas en la base de datos estar representadas por clases en Java

- Una entidad es una clase Java marcada con la anotación @Entity
- Las clases Java deben ser POJOs y deben de tener un constructor vacío y getters/setters para sus atributos.
- Los atributos de la entidad corresponderán con columnas de la tabla.
- Todos los atributos de la clase serán persistentes a menos que se anoten con @Transient.

Tipos de atributos

- **No pueden ser atributos static o de tipo final**
- Tipos primitivos: int, float, etc.
- Objetos para tipos primitivos: java.lang.Integer, java.lang.Float, etc.
  - Tienen una ventaja sobre los tipos primitivos, soportan el uso de null como valor.
- java.util.Date, java.util,Calendar, java.util.Timestamp, etc.
- Otras entidades
- Clases embebibles @Embebedable
- Collections: java.util.Collections, java.util.Sets

@Table

Especifica la tabla relacionada con su entidad

- Atributos relevantes:
  - **name:** nombre de la tabla (por defecto el nombre de la clase)

Anotaciones para atributos

- Los atributos pueden anotarse a nivel de declaración o en su respectivo getter.
- **@Column:** específica una columna de la tabla a malear sobre un atributo de la entidad. Es una anotación opcional, si no se usa, el framework utiliza los valores por defecto para mapear los atributos.
- Atributos relevares:
  - **name:** nombre de la columna
  - **unique:** atributo con valor único
  - **nullable:** permite atributos nulos
- **@Transient:** específica un atributo no persistente.

Anotaciones para atributos

- **@Id:** especifica que se trata de un atributo que identifica la entidad (PK)
- **@GeneratedValue:** especifica como será generado el identificador para la tabla
  - **Strategy**
    - AUTO: (valor por defecto) se define en función a GBD
    - IDENTITY
    - SEQUENCE
    - TABLE (utiliza una tabla de identificadores
  - **Generator**
    - @SequenceGenerator
    - @TableGenerator

Anotaciones para atributos

- **@EmbeddedId:** clave primaria con múltiples campos
- **@IdClass:** se utiliza para el caso de claves primarias compuestas
- **@Temporal:** se usa para indicar campos que almacenan fechas
- **@JoinColumn:** especifica una tributo que actúa como clave foránea de otra entidad

Relaciones entre entidades

- **@OneToOne:** (relación 1:1) indica un atributo que está asociado con una tupla de otra Entidad.
  - Puede requerir una anotación @JoinColumn (marca el atributo como clave foránea)
  - Puede requerir especificar el parámetro mappedBy

Relaciones entre entidades

- **@OneToMany:** (relación 1:N) indica un atributo que está asociado con varias tuplas de otra Entidad (lado N)
    - Aplicada sobre atributos de tipo Set<Entidad> o Collection<Entidad>
    - Requiere especificar el parámetro mappedBy (clave foránea en el extremo N)

Relaciones entre entidades

- **@ManyToOne:** (relaciónN:1) indica un atributo (ladoN) asociado con una tupla de otra Entidad (lado 1)
  - Requiere una anotación @JoinColumn para marcar el atributo como clave foránea

Relaciones entre entidades

- **@ManyToOne:** (relación N:M) indica un atributo que tiene una relación N:M con tuplas de otra Entidad
  - Aplicada sobre atributos de tipo Set<Entidad> o Collection<Entidad>
  - Requiere especificar la tabla que implementa la relación con sus claves foráneas con la anotación @JoinTable

Relaciones entre entidades

- **@ManyToMany:** (relación N:M) indica un atributo que tiene una relación N:M con tuplas de otra Entidad
  - Aplicada sobre atributos de tipo Set<Entidad> o Collection<Entidad>
  - Requiere especificar la tabla que implementa la relación con sus claves foráneas con la anotación @JoinTable
- **@JoinTable:** se usa en relaciones ManyToOne y ManyToMany

Herencia

- Se usa la anotación @Inheritance a nivel de declaración de la clase
  - **Stategy:**
    - SINGLE_TABLE
    - JOINED
    - TABLE_PER_CLASS

Example

```
@Entity
@Table(name = "Persona")
public class LineaPedido implements Serializable {
  @Id
  @GeneratedValue(strategy=GenerationType.AUTO)
  private Long id;
  @Column(name = "nombre")
  private String nombre;
  private String apellido;
  @JoinColumn(name = "id")
  @OneToOne
  private Direccion direccion;
 ...
```

EntityManager (EM)

- Ofrece una interfaz para interactuar con el contexto de persistencia
- Responsabilidades
  - Control de transacciones
  - Gestión del ciclo de vida de la Entidades
  - Creación de consultas
  - Otros: gestión de cachés, gestión de identidad de las entidades.

EntityManager (EM)

- Cada EM esta asociado a un PersistenceUnit (PU)
- Especifica el proveedor de persistencia a usar. (persistence provider) y el datasource a usar.
- La configuraciones en almacenada en el archivo persistence.xml (en el directorio META-INF/ para los JAR o en WEB-INF/ para los WAR)

EntityManager (EM)

```
<persistence>
        <persistence-unit name="EjemploPedidosPU"
                transaction-type="JTA">
          <jta-data-source>
                jdbc/EjemploPedidos
          </jta-data-source>
          <properties>
              <property name=“toplink.ddl-generation"
                        value="create-tables"/>
          </properties>
        </persistence-unit>
</persistence>
```

EntityManager (EM)
Obtención del contexto de persistencia

```
EntityManagerFactory emf =
Persistence.createEntityManagerFactory("EjemploPU");

EntityManager em = emf.createEntityManager();
```

EntityManager (EM)
Ciclo de vida

new

- Entidad recién creada. El EM no tiene conocimiento de esta entidad.

managed

- Es una entidad que forma parte del contexto de persistencia y es manejada por el EM.
- Los cambios realizados a la entidad serán manejados por el EM y serán persistidos al terminar la transacción o al llamar al método flush del EM.

EntityManager (EM)
Ciclo de vida

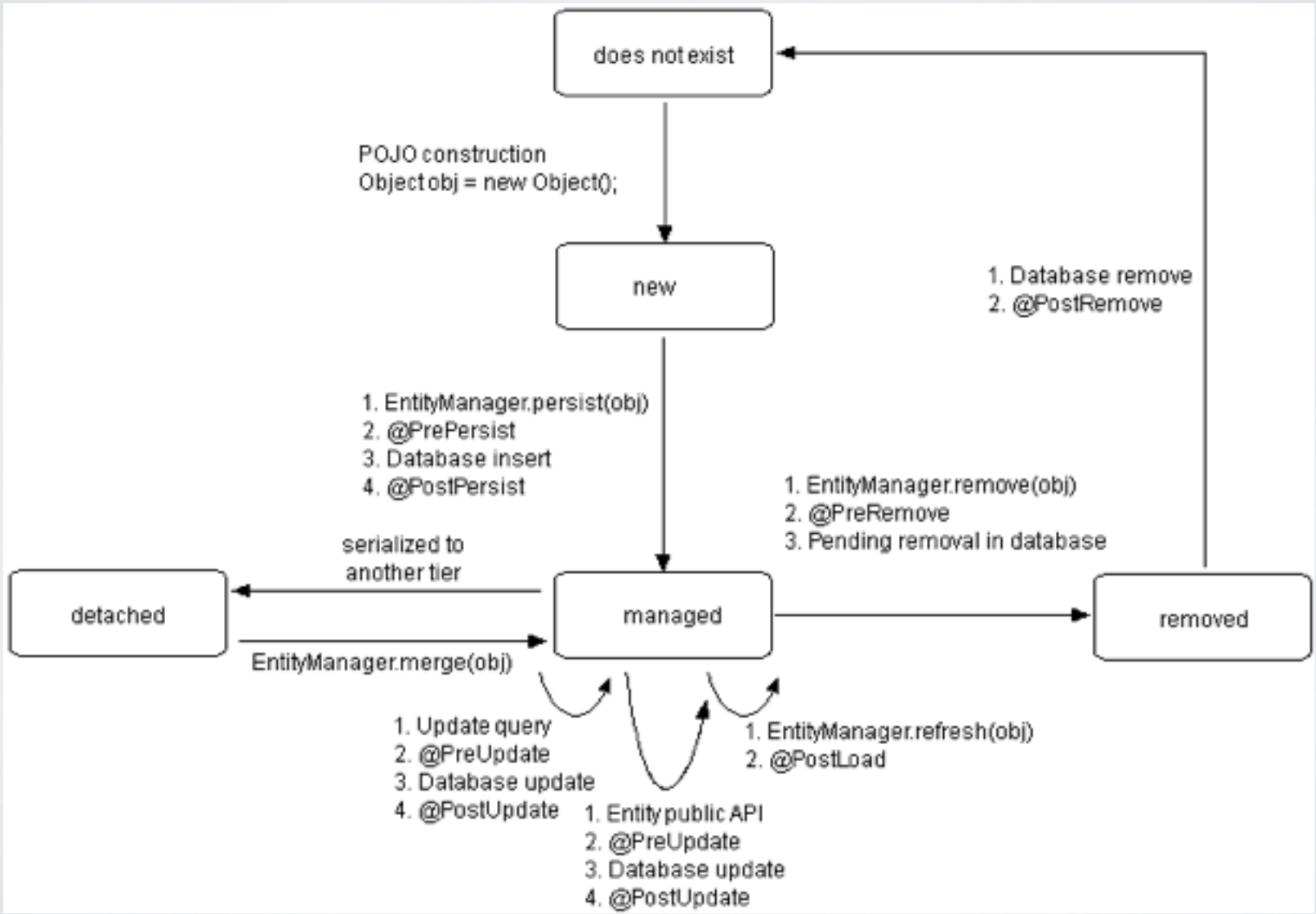
detached

- Entidad que formaba parte del contexto de persistencia y ya no.
- Es necesario llamar al método merge del EM para volver a hacerla manejada.

removed

- instancia de entidad sobre la que a llamado al metodo **remove** del EM sobre ella.

EntityManager (EM)
Ciclo de vida



EntityManager (EM)
Métodos

- **T find(Class<T> e, Object id)**
  busca una entidad en el contexto de persistencia usando su clave primaria.

- **persist(Object e)**
  almacena un entidad en la base de datos

EntityManager (EM)
Métodos

T merge(T e)

- recibe una entidad no manejada y retorna una entidad manejada.

remove

- Elimina una entidad.

EntityManager (EM)
Métodos

refresh(Object e)

- Actualiza los datos de una instancia con los valores de la BD.

flush()

- Sincroniza las instancias gestionadas por el EM actualizando la DB.

Referencias
Imágenes en orden de aparición

- http://fc00.deviantart.net/fs71/i/2013/016/f/7/layered_database_source_documents_by_barrymieny-d5rnycs.jpg
- http://docs.oracle.com/cd/E16439_01/doc.1013/e13981/img/lifeent30.gif