**Introduzindo Java Persistence API**

por:

Raphaela Galhardo (raphaela@jeebrasil.com.br)

Este artigo trata de uma introdução à especificação para o gerenciamento de persistência e mapeamento objeto relacional surgida na plataforma Java EE 5.0, a API Java Persistence (JPA). Também será apresentado o gerenciador de persistência (EntityManager), bem como o ciclo de vida de persistência. Por fim, um exemplo simples também será mostrado.

**1. Introdução**

JPA (*Java Persistence API*) é a especificação padrão para o gerenciamento de persistência e mapeamento objeto relacional, surgida na plataforma Java EE 5.0 (Java *Enterprise Edition*) na especificação do EJB 3.0 (*Enterprise Java Beans* 3.0). Introduzida no intuito de substituir os *Entity Beans* (que foram descontinuados) e simplificar o desenvolvimento de aplicações Java EE e Java SE que utilizam persistência de dados, possui uma completa especificação para realizar mapeamento objeto relacional, utilizando anotações da linguagem Java (Java SE 5.0 ou superior). Também dá suporte a uma rica linguagem de consulta, semelhante à SQL, permitindo consultas estáticas ou dinâmicas.

Um dos principais conceitos relacionados à API JPA é o de entidade. Uma entidade corresponde a um objeto que pode ser gravado na base de dados a partir de um mecanismo de persistência. A classe que implementa a entidade persistente é um POJO, que basicamente contém um construtor padrão sem argumentos e uma chave primária e também não precisa derivar de nenhuma interface ou classe, nem implementar qualquer método em especial.

Quando se decide utilizar o JPA, é necessária a escolha de um provedor JPA. Como esta API é uma especificação para *frameworks* de persistência, existe a necessidade de provedores JPA. Por padrão, a implementação de referência é o *Oracle Toplink Essentials*. Também existem outros provedores JPA no mercado, como o *Hibernate Entity Manager*, *Bea Kodo* e o *ApacheJPA*.

**2. Gerenciador de Entidades e Ciclo de Vida de um Objeto Persistente**

Nas diversas aplicações existentes, sempre que for necessário propagar o estado de um objeto que está em memória para o banco de dados ou vice-versa, há a necessidade de que a aplicação interaja com uma camada de persistência. Com a API JPA, isso é feito invocando o gerenciador de persistência, também conhecido como gerenciador de entidades (*EntityManager*), responsável por quase todas as operações de persistência de objetos.

Um outro conceito também relacionado a esta especificação é o de contexto persistente (*PersistenceContext*), que é uma área de memória onde os objetos persistentes se encontram. Quando se interage com o mecanismo de persistência, é necessário para a aplicação ter conhecimento sobre os estados do ciclo de vida da persistência.

Em aplicações orientadas a objetos, a persistência permite que um objeto continue a existir mesmo após a destruição do processo que o criou. Na verdade, o que continua a existir é seu estado, já que pode ser armazenado em disco e então, no futuro, ser recriado em um novo objeto. O clico de vida de uma entidade pode ser constituído por quatro estados: *new*, *managed*, *removed* e *detached*, como mostrado na Figura 1.

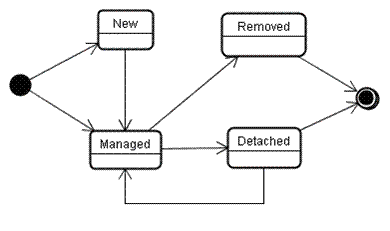


Figura 1- Ciclo de Vida de Objetos Persistentes

Um objeto que se encontra no estado *new* pode ser definido como aquele que foi definido a partir do operador **new** e que não possui um valor para o seu identificador persistente, por exemplo, aquele com um ciclo de vida limitado ao tempo de vida do processo que o instanciou. Objetos neste estado também são conhecidos como objetos transientes. Em outras palavras, o objeto não está associado ao *EntityManager*.

Já um objeto no estado *managed* é aquele que tem um valor definido para o seu identificador persistente e está associado a um contexto de persistência. Assim, o *EntityManager* irá fazer com que a base de dados seja atualizada com novos valores dos atributos persistentes no fim da transação.

Por outro lado, existem objetos também associados a um contexto de persistência, porém que estão “marcados” para serem excluídos da base de dados no final da transação, e assim permanecendo no estado *removed*.

Por fim, também existem aqueles objetos que se encontram em um estado denominado *detached*, que possuem registros correspondentes na base de dados, mas não estão associados a um contexto de persistência.

**3. Introduzindo Anotações (*Annotations*)**

Como já citado, o mapeamento objeto relacional utilizando JPA pode ser feito a partir de anotações. As anotações podem ser definidas como metadados que aparecem no código fonte e são ignorados pelo compilador. Qualquer símbolo em um código Java que comece com uma @ (arroba) é uma anotação. Este recurso foi introduzido na linguagem Java a partir da versão Java SE 5.0. Em outras palavras, as anotações marcam partes de objetos de forma que tenham algum significado especial.

A Listagem 1 e a Listagem 2 apresentam exemplo de um tipo de anotação denominado de **TesteAnotacao** e do seu uso em um método qualquer (**metodoTeste**), respectivamente. O exemplo é meramente ilustrativo, de forma que o tipo de anotação definido não agrega nenhum significado especial ao código fonte.

public @interface TesteAnotacao{

...

}

Listagem 1 – Exemplo ilustrativo para a criação de um tipo de anotação

@TesteAnotacao

public void metodoTeste{

...

}

Listagem 2 – Exemplo ilustrativo para o uso de um tipo de anotação

As anotações podem possuir nenhum, um ou mais de um elementos em sua definição. Um exemplo de tipo de anotação sem nenhum atributo já foi visto nas Listagem 1 e na Listagem 2. Se um tipo de anotação possui um único elemento, como definido e ilustrado na Listagem 3, no momento de se utilizar a anotação, um valor para este atributo deve ser passo entre parênteses, como mostrado na Listagem 4

public @interface TesteAnotacao{

String elemento1();

}

Listagem 3 – Exemplo ilustrativo para a criação de um tipo de anotação

@TesteAnotacao(“Definindo valor”)

public void metodoTeste{

...

}

Listagem 4 – Exemplo ilustrativo para o uso de um tipo de Anotação

No caso caso em que o tipo da anotação ser definido com mais de um elemento, como mostrado na Listagem 5, na hora em que a anotação for usada, para atribuir valores para seus elementos é preciso passá-los também entre parênteses, porém definindo o nome do elemento e o seu valor, como exemplo ilustrado na Listagem 6.

public @interface TesteAnotacao{

String elemento1();

int elemento2();

}

Listagem 5 – Exemplo ilustrativo para a criação de um tipo de anotação

@TesteAnotacao(elemento1 = “Definindo valor 1”, elemento2 = 25)

public void metodoTeste{

...

}

Listagem 6 – Exemplo ilustrativo para o uso de um tipo de. Anotação

**4. Exemplo Simples com JPA**

Neste primeiro exemplo, será apresentado como transformar uma classe Java em uma classe persistente utilizando anotações. A Listagem 7 apresenta a classe **Aluno**, porém marcada com a anotação **@Entity**, que faz com que ela se torne uma classe persistente. A anotação **@Table** informa o nome da tabela, através do atributo **name**, para a qual a classe **Aluno** será mapeada, no caso, a tabela **ALUNO**. Quando o nome da tabela é igual ao nome da classe, está anotação não precisa ser informada, basta a anotação **@Entity**.

Já a anotação **@Id** faz com que o atributo a que ele se refere (no caso o atributo **id** da classe **Aluno**) se torne o identificador único para os objetos desta classe. Em relação à tabela **ALUNO**, seria o atributo correspondente à chave primária da tabela. Já a anotação **@GeneretedValue** faz com que o *framework* de persistência gere valores para a chave primária na tabela de forma automática e valores únicos.

Os demais atributos da classe não possuem anotações associadas. Desta forma, eles são mapeados com anotações padrões que mapeiam os atributos para colunas na tabela com tipos padrões, por exemplo, o atributo **nome** do tipo **String** é mapeado para uma coluna da tabela do tipo **VARCHAR**.

É importante ressaltar a presença da linha **import javax.persistence.\*;** , que possibilita o uso das anotações presentes no pacote **javax.persistence**.

package br.com.jeebrasil.dominio;

import javax.persistence.\*;

@Entity

@Table (name=”aluno”)

public class Aluno {

@Id @GeneratedValue

private int id;

private int matricula;

private String nome;

}

Listagem 7 – Primeiro exemplo: classe Aluno persistente com anotações.

Para persistir objetos mapeados, por exemplo, objetos da classe **Aluno**, é preciso utilizar o módulo gerenciador de persistência, o *EntityManager*. Este módulo tem a função de se conectar à base de dados e gerenciar os objetos, permitindo salvá-los, atualizá-los, excluí-los e consultá-los na base de dados.

A Listagem 8 mostra o primeiro exemplo de persistência de objetos da classe **Aluno**. A criação de um objeto do tipo **EntityManager** é feita utilizando o padrão de projeto *factory*. Dessa forma, cria-se um objeto do tipo **EntityManagerFactory** através do método estático **Persistence.createEntityManagerFactory(“exemplo”)** . A **String** “exemplo”, passada como argumento do método, serve para definir o nome da unidade de persistência utilizada (conceito a ser definido posteriormente). A partir deste objeto criado, é possível criar o gerenciador de objetos, ou seja, um objeto **EntityManager**, utilizando o método **createEntityManager()**. Em geral, a aplicação deve conter apenas uma instância da classe **EntityManagerFactory** e cada usuário uma instância da classe **EntityManager**, já que o primeiro tem a característica de *thread-safe* e o segundo não.

Dentro do bloco **try{}** são feitas as persistências de dois objetos da classe **Aluno**. Primeiro, uma transação é criada, a partir do método **getTransaction()** do gerenciador de persistência. Em seguida, dois objetos são criados, recebem valores para seus atributos e são persistidos na base de dados a partir do método **persist()**. Por fim, a transação é confirmada e os recursos do gerenciador de persistência e de sua fábrica são liberados. Caso ocorra algum erro dentro do bloco da transação, a exceção é capturada e o método **rollback** é executado, desfazendo o que foi feito até o momento em que o erro ocorreu.

package br.com.jeebrasil.exemplo;

import javax.persistence.\*;

public class Principal {

public static void main(String[] args){

//Cria objeto que gerenciará persistência

EntityManagerFactory fabrica =

Persistence.createEntityManagerFactory("exemplo");

EntityManager gerenciador = fabrica.createEntityManager();

try {

EntityTransaction tx = gerenciador.getTransaction();

tx.begin(); //Inicia transação

Aluno a1 = new Aluno();

a1.setNome(“Pedro”);

a1.setMatricula(20009387);

gerenciador.persist(a1); //Persiste primeiro aluno

Aluno a2 = new Aluno();

a2.setNome(“Maria Eduarda”);

a2.setMatricula(20052347);

gerenciador.persist(a2); //Persiste segundo aluno

tx.commit(); //Finaliza transação

} catch (Exception ex) {

tx.rollback(); //Desfaz operações em caso de erro

} finally {

//Libera recursos

gerenciador.close();

fabrica.close();

}

}

}

Listagem 8 – Primeiro exemplo: uso do EntityManager

**4.1 Configurando a Base de Dados**

Para tudo funcionar corretamente, é necessário realizar algumas configurações da base de dados. Essas configurações são feitas a partir de um arquivo XML, onde é montada a unidade de persistência, em que são listadas as classes a serem persistidas, informações para conexão com a base de dados, entre outras informações para a persistência (por exemplo, o dialeto utilizado). A Listagem 9 apresenta um exemplo deste arquivo de configuração. Observa-se que a primeira configuração, da tag é informado o nome da unidade de trabalho “exemplo”, usada no momento da criação da fábrica de gerenciadores de persistência, através da invocação do método **Persistence.createEntityManagerFactory**.

<?xml version="1.0" encoding="UTF-8"?>

<persistence xmlns="http://java.sun.com/xml/ns/persistence"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/persistence

http://java.sun.com/xml/ns/persistence/persistence\_1\_0.xsd"

version="1.0">

<persistence-unit name="exemplo" transaction-type="RESOURCE\_LOCAL">

<class>br.com.jeebrasil.dominio.Aluno</class>

<properties>

<property name="hibernate.connection.driver\_class"

value="org.hsqldb.jdbcDriver"/>

<property name="hibernate.connection.username"

value="sa"/>

<property name="hibernate.connection.password"

value=""/>

<property name="hibernate.connection.url"

value="jdbc:hsqldb:hsql://localhost"/>

<property name="hibernate.dialect"

value="org.hibernate.dialect.HSQLDialect"/>

</properties>

</persistence-unit>

</persistence>

Listagem 9 - Exemplo de configuração de uma unidade de persistência

**5. Conclusões**

Com este artigo, procurou-se passar uma breve introdução sobre a nova API JPA, surgida no Java EE 5.0 para facilitar o gerenciamento de persistência e o mapeamento objeto relacional. Sendo ela uma especificação para um *framework* de persistência, permite que surjam vários provedores JPA de forma que o programador possa escolher dentre aquele melhor a se trabalhar. A partir do surgimento da API JPA, acredita-se que a plataforma Java EE adquira credibilidade com o seu mecanismo de persistência, antes não existente com os *Entity Beans*.