Curso EBAC
Desenvolvedor Back-End Java
Exercício Módulo 31
Conceitos sobre JPA(Java Persistence API)
Versão 0.2

Aluno **WAGNER ELVIO**

9 de fevereiro de 2024

Sumário

Objetivo: Apresentar uma descrição sobre o JPA.

1 Dificuldades na conexão entre Banco de Dados e Orientação a Objetos

- A maior dificuldade de se usar a abordageem orientada a obejtos é a comuniação com um banco de dados relacional
- Precisava transportar a cada momento de tabela para objeto e vice-versa;
- Havia um custo considerável em termo de programação para fazer essa simples tarefa de acessar os dados em um banco de dados;
- O autor ***Martin Fowler*** em seu livro ***Patterns of Enterprise Application Architecture***(Padrões de arquitetura de aplicativos empresariais)[1] relata neste livro que o esforço da programação da camada de acesso aos dados para fazer a conexão entre o banco de dados relacional e os dados orientados a objetos girava em torno de aproximandamente 30%.
- Antes da criação das ferramentas de mapeamento objeto-relacional, haviam alguns problemas que demandavam um esforço computacional considerável. Abaixo alguns exemplos:

  a. **Contexto de persistência:** *Em programação, especialmente em ambientes de desenvolvimento de aplicativos, o contexto de persistência pode se referir à maneira como o estado de um aplicativo é mantido entre diferentes interações do usuário. Isso pode incluir informações armazenadas localmente no dispositivo do usuário ou em servidores remotos. E antes do uso da JPA era necessário fazer um código para manter esse contexto de persistência.*

  b. ***Mapa de identidade:***Refere-se a fazer um *cache* dos objetos que já foram acessados ou carregados, ou fazer um mapa de gerencamento dos objjetos que já foram carregados, pois uma vez acessados esse objetos, os mesmos já se encontram instanciados na memória e caso seja preciso acessar esse objetos não se faz necesssário **ir novamente no banco de dados**.

  c. **Carregamento tardio (*lazy loading*)**: Permite acessar apeans os dados necessários para a aplicação.

- Todos esses exemplos listados acima, já são feitos de forma automática pelas ferramentas de mapamento objeto relacional.
- A evolução das ferramentas de persistência ao longo do tempo reflete a busca por soluções mais eficientes e padronizadas para lidar com o armazenamento de dados em aplicações de software. A evolução culminou na especificação Java Persistence API (JPA), que faz parte da Java Community Process e é documentada na JSR 338 (Java Specification Request 338).
- A JSR 338 é a especificação mais recente da JPA. Ela introduziu melhorias e novos recursos, aprimorando ainda mais a capacidade da JPA de lidar com a persistência de dados de maneira eficiente e padronizada. Isso inclui suporte a consultas armazenadas, atualizações em lote e outras melhorias de desempenho e funcionalidade.
- A JPA (*Java Persistence API*) é uma especificação baseada na JSR 338 e que se encontra disponível em `https://download.oracle.com/otn-pub/jcp/persistence-2_2-mrel-spec/JavaPersistence.pdf?AuthParam=1707512143_d07382251793fa919a5e51fcc16a2114`

[1] Disponível na loja *online* em `https://www.amazon.com.br/Patterns-Enterprise-Application-Architecture-Martin/dp/0321127420/ref=sr_1_1?crid=88I7Q1ZU69WR&keywords=patterns+of+enterprise+application+architecture+martin+fowler&qid=1707509143&sprefix=martin+fowler%2Caps%2C205&sr=8\protect\discretionary{\char\hyphenchar\font}{}{}1&ufe=app_do%3Aamzn1.fos.4bb5663b-6f7d-4772\protect\discretionary{\char\hyphenchar\font}{}{}84fa-7c7f565ec65b`

2 O que é o paradigma relacional?

- O paradigma relacional é um modelo de organização e manipulação de dados em um sistema de gerenciamento de banco de dados (SGBD). Foi proposto por Edgar F. Codd em 1970 e é amplamente utilizado em sistemas de banco de dados relacionais. No paradigma relacional, os dados são organizados em tabelas (também chamadas de relações), que consistem em linhas (tuplas) e colunas. Cada tabela representa um tipo de entidade ou conceito, e as colunas representam atributos ou características dessa entidade. As relações entre diferentes tabelas são estabelecidas através de chaves primárias e chaves estrangeiras.

- Principais conceitos do paradigma relacional:

  a. Tabela (Relação): É uma estrutura que armazena dados de uma entidade específica. Cada linha na tabela é uma tupla (registro) que contém valores correspondentes aos atributos (colunas) da entidade.

  b. Coluna (Atributo): Representa uma característica da entidade que está sendo modelada. Cada coluna tem um nome único e um tipo de dado associado.

  c. Chave Primária: É um ou mais atributos que identificam exclusivamente cada tupla em uma tabela. Garante que não haja duplicatas e é usada para referenciar a tupla em outras tabelas.

  d. Chave Estrangeira: É um atributo em uma tabela que estabelece uma relação com a chave primária de outra tabela. Isso permite a criação de associações entre diferentes entidades.

  e. Integridade Referencial: É a garantia de que as relações entre tabelas são mantidas consistentes, ou seja, os valores das chaves estrangeiras em uma tabela correspondem às chaves primárias correspondentes em outras tabelas.

  f. Operações CRUD: São as operações básicas de manipulação de dados em um SGBD relacional: Create (Inserir), Read (Ler), Update (Atualizar) e Delete (Excluir).

  g. Normalização: É o processo de organizar as tabelas de forma eficiente para reduzir a redundância de dados e garantir a integridade dos dados.

- O paradigma relacional é amplamente utilizado em sistemas de banco de dados comerciais e aplicativos de software em todo o mundo devido à sua capacidade de modelar e gerenciar dados de maneira estruturada e consistente. Grandes sistemas de gerenciamento de banco de dados, como MySQL, PostgreSQL, Oracle Database e Microsoft SQL Server, são baseados no modelo relacional.

3 O que é o paradigma orientado a Objetos ?

- O paradigma orientado a objetos (POO) é um modelo de programação e design de software que se baseia na ideia de que o mundo real é composto por objetos individuais que têm características (atributos) e comportamentos (métodos). No desenvolvimento de software orientado a objetos, esse conceito é aplicado para estruturar e organizar o código de uma maneira que reflete a maneira como pensamos e interagimos com o mundo real.
- Principais conceitos do paradigma orientado a objetos:
  a. **Classe e Objeto:** Uma classe é uma definição que descreve o que um objeto pode ter (atributos) e fazer (métodos). Um objeto é uma instância de uma classe, ou seja, é uma entidade real criada com base na definição da classe.
  b. **Atributos e Métodos:** Atributos são as características ou propriedades de um objeto. Métodos são as ações que um objeto pode executar, ou seja, suas funções.
  c. **Encapsulamento:** É o conceito de esconder detalhes internos de uma classe e expor apenas uma interface para interagir com ela. Isso ajuda a proteger os dados e funcionalidades internas da manipulação indevida.
  d. **Herança:** Permite que uma classe herde atributos e métodos de outra classe. Isso promove a reutilização de código e estabelece uma hierarquia de classes.
  e. **Polimorfismo:** Permite que objetos de diferentes classes sejam tratados de maneira uniforme, utilizando a herança e a sobrescrita de métodos.
  f. **Abstração:** Envolve a criação de classes que representam conceitos ou entidades do mundo real de forma simplificada, capturando apenas os detalhes relevantes.
  g. **Associação:** Representa as relações entre diferentes classes. Pode ser de um-para-um, um-para-muitos, muitos-para-muitos, entre outros.

4 Visão geral sobre Mapeamento objeto-relacional

- O Mapeamento Objeto-Relacional (ORM) é uma técnica que permite a representação de dados de um sistema orientado a objetos em um banco de dados relacional. Ele resolve o problema da disparidade entre o modelo de dados utilizado em um sistema orientado a objetos e o modelo de dados relacional de um banco de dados.
- Ao usar um ORM, o desenvolvedor pode interagir com o banco de dados usando código orientado a objetos, sem a necessidade de escrever consultas SQL manualmente. O ORM cuida da tradução entre a representação orientada a objetos dos dados e a representação relacional no banco de dados. Isso simplifica o desenvolvimento, torna o código mais legível e facilita a manutenção do sistema. No entanto, é importante entender como o ORM está gerenciando as consultas SQL subjacentes para otimizar o desempenho, especialmente em sistemas que lidam com grandes volumes de dados.
- Quando você trabalha com um sistema orientado a objetos, você lida com classes, objetos e relacionamentos entre eles. No entanto, os bancos de dados relacionais operam com tabelas, linhas e colunas. O ORM serve como uma camada intermediária entre o código da aplicação e o banco de dados, facilitando a manipulação dos dados de forma mais orientada a objetos.
- Existem várias bibliotecas e frameworks que implementam o ORM em diversas linguagens de programação. Alguns exemplos incluem:
    a. Hibernate (Java): Muito popular no mundo Java, o Hibernate é uma implementação de ORM que permite mapear classes Java para tabelas de banco de dados e vice-versa.
    b. Entity Framework (.NET): Desenvolvido pela Microsoft, o Entity Framework é um ORM para .NET que facilita a interação com bancos de dados relacionais.
    c. Django ORM (Python): O Django é um framework web para Python que inclui um poderoso ORM para simplificar a interação com bancos de dados.
    d. SQLAlchemy (Python): Uma biblioteca Python que fornece um conjunto abrangente de ferramentas para trabalhar com bancos de dados SQL de forma flexível e eficiente.
    e. Rails ActiveRecord (Ruby on Rails): O Ruby on Rails possui um ORM chamado ActiveRecord que simplifica a manipulação de dados em um banco de dados relacional.

5 JPA

- O JPA (*Java Persistence API*) é um *framework* leve, baseado em POJOS (*Plain Old Java Objects*) para persistir objetos Java.
- A Java Persistence API, diferente do que muitos imaginam, não é apenas um framework para Mapeamento Objeto-Relacional (ORM - *Object-Relational Mapping*), ela também oferece diversas funcionalidades essenciais em qualquer aplicação corporativa.
- Atualmente temos que praticamente todas as aplicações de grande porte utilizam JPA para persistir objetos Java. JPA provê diversas funcionalidades para os programadores, como será mais detalhadamente visto nas próximas seções.
- Inicialmente será visto a história por trás da JPA, a qual passou por algumas versões até chegar na sua versão atual.

5.1 Breve histórico

- Após diversos anos de reclamações sobre a complexidade na construção de aplicações com Java, a especificação Java EE 5 teve como principal objetivo a facilidade para desenvolver aplicações JEE 5. O EJB 3 foi o grande percursor para essa mudança fazendo os Enterprise JavaBeans mais fáceis e mais produtivos de usar. No caso dos Session Beans e Message-Driven Beans, a solução para questões de usabilidade foram alcançadas simplesmente removendo alguns dos mais onerosos requisitos de implementação e permitindo que os componentes sejam como Plain Java Objects ou POJOS.
- Já os Entity Beans eram um problema muito mais sério. A solução foi começar do zero. Deixou-se os Entity Beans sozinhos e introduziu-se um novo modelo de persistência. A versão atual da JPA nasceu através das necessidades dos profissionais da área e das soluções proprietárias que já existiam para resolver os problemas com persistência. Com a ajuda dos desenvolvedores e de profissionais experientes que criaram outras ferramentas de persistência, chegou a uma versão muito melhor que é a que os desenvolvedores Java conhecem atualmente.
- Dessa forma os líderes das soluções de mapeamento objetos-relacionais deram um passo adiante e padronizaram também os seus produtos. Hibernate e TopLink foram os primeiros a firmar com os fornecedores EJB.
- O resultado final da especificação EJB finalizou com três documentos separados, sendo que o terceiro era o Java Persistence API. Essa especificação descrevia o modelo de persistência em ambos os ambientes Java SE e Java EE.
- Cronologia:
  a. **Início dos Anos 2000:** Antes do surgimento da JPA, os desenvolvedores Java geralmente usavam tecnologias específicas de fornecedores, como o Hibernate, para realizar o mapeamento objeto-relacional. Isso significava que o código estava fortemente acoplado à implementação do ORM escolhido.
  b. **2003:** O Hibernate, uma implementação ORM popular em Java, foi lançado. Ele desfrutou de grande aceitação devido à sua eficácia e recursos avançados. No entanto, como era uma solução independente, a portabilidade do código entre diferentes fornecedores de ORM era um desafio.
  c. **2006:** A Java Community Process (JCP) iniciou o desenvolvimento da JPA como parte da especificação EJB 3.0 (Enterprise JavaBeans). O principal objetivo era padronizar o mapeamento objeto-relacional e fornecer uma API unificada para persistência em Java. 2007:A JPA 1.0 foi lançada como parte da Java EE 5.0. Isso marcou um grande avanço na simplificação da persistência de dados em Java, fornecendo uma abordagem padronizada e independente de fornecedor.
  d. **2011:** A JPA 2.0 foi lançada como parte da Java EE 6. Esta versão trouxe melhorias significativas, como o suporte a consultas criteriosas, atualizações e exclusões em lote, além de aprimoramentos no suporte a herança e mapeamento de relacionamentos. **2013:** A JPA 2.1 foi lançada como parte da Java EE 7. Introduziu recursos como suporte a armazenamento de procedimentos e funções, conversores de atributos, e suporte a enumerações.

e. Atualmente, a JPA continua sendo uma tecnologia essencial no ecossistema Java para persistência de dados em aplicações empresariais. As versões mais recentes, fora do escopo deste histórico, continuam aprimorando a API com base no feedback da comunidade e nas necessidades emergentes do desenvolvimento de software.

5.2 O que é a JPA

– O que é o JPA ((*Java Persistence API*)): A JPA (*Java Persistence API*) é uma API do Java que fornece um *framework* de mapeamento objeto-relacional (ORM) para Java EE (Enterprise Edition) e Java SE (Standard Edition). Ela simplifica o desenvolvimento de aplicativos Java que precisam interagir com bancos de dados relacionais. A principal finalidade da JPA é fornecer uma maneira padronizada de mapear objetos Java para tabelas de banco de dados e vice-versa. Isso elimina a necessidade de escrever manualmente código SQL para interagir com o banco de dados, facilitando o desenvolvimento e manutenção de aplicações. Principais conceitos e funcionalidades da JPA:

a) **Entidades:** São classes Java que são mapeadas para tabelas no banco de dados. Cada instância de uma entidade representa uma linha na tabela.

b) **Mapeamento Objeto-Relacional (ORM):** A JPA permite que você mapeie objetos Java para tabelas de banco de dados e vice-versa, simplificando a persistência de dados.

c) **EntityManager:** É uma interface da JPA que gerencia as operações de persistência, como salvar, atualizar, recuperar e excluir entidades.

d) **Consultas JPQL (Java Persistence Query Language):** Uma linguagem de consulta orientada a objetos semelhante ao SQL, mas opera em entidades mapeadas.

e) **Relacionamentos entre Entidades:** A JPA suporta o mapeamento de relacionamentos entre entidades, como associações um-para-um, um-para-muitos e muitos-para-muitos.

f) **Transações:** A JPA integra-se ao sistema de transações do Java, permitindo que você gerencie transações de forma eficiente.

g) **Annotations:** Utilização de anotações Java para configurar o mapeamento objeto-relacional, como @Entity, @Table, @Id, etc.

5.3 Exemplos de anotações em uma JPA

– Na JPA, as anotações são usadas para mapear classes Java para tabelas de banco de dados e configurar o comportamento do ORM. Aqui estão alguns exemplos de anotações comuns utilizadas na JPA:

a. **@Entity:** Marca uma classe como uma entidade que será mapeada para uma tabela no banco de dados. Um exemplo dessa anotação está mostrada no código listado em 1.

```
1 import javax.persistence.Entity;
2
3 @Entity
4 public class Produto {
5     // atributos, construtores, metodos getters/setters, etc.
6 }
```

Listing 1: Anotacao @Entity

b. **@Table:** Esta é especifica o nome da tabela no banco de dados associada à entidade. Se não for fornecido, o nome da tabela será o mesmo que o nome da classe. Um exemplo dessa anotação está mostrada no código listado em 2.

```
1 import javax.persistence.Entity;
2 import javax.persistence.Table;
3
4 @Entity
5 @Table(name = "tbl_produto")
6 public class Produto {
7     // ...
```

```
}
```

Listing 2: Anotacao @Table

c. **@Id:** Essa anotação marca um campo como a chave primária da entidade. Um exemplo dessa anotação está mostrada no código listado em 3.

```
import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class Produto {
    @Id
    private Long id;
    // ...
}
```

Listing 3: Anotacao @Id

d. **@GeneratedValue:** Especifica a estratégia de geração de valores para a chave primária. Um exemplo dessa anotação está mostrada no código listado em 4.

```
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Produto {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    // ...
}
```

Listing 4: Anotacao @GeneratedValue

e. **@Column:** Essa anotação permite configurar propriedades específicas de colunas no banco de dados, como nome, comprimento, *nullable*, etc. Um exemplo dessa anotação está mostrada no código listado em 5.

```
import javax.persistence.Entity;
import javax.persistence.Column;

@Entity
public class Produto {
    @Column(name = "nome_produto", length = 50, nullable = false)
    private String nome;
    // ...
}
```

Listing 5: Anotacao @Column

Referências

http://www.cprogressivo.net/2012/09/hello-world-comentado-c.html Como criar seu primeiro programa em C

https://aulasdec.wordpress.com/2010/11/05/o-primeiro-programa-helloworld-c/ O primeiro programa: HelloWorld.c

https://www.devmedia.com.br/introducao-a-jpa-java-persistence-api/28173 Introdução à JPA - Java Persistence API

https://openjpa.apache.org/ Welcome to the Apache OpenJPA project

https://access.redhat.com/documentation/pt-br/red_hat_jboss_enterprise_application_platform/6.4/html/development_guide/sect-java_persistence_api_jpa Java Persistence API (JPA)