Faça uma revisão do JPA 2.1 em português brasileiro

Thought for 9 seconds

> Estou sendo solicitado a fazer uma revisão do JPA 2.1 em português brasileiro. Preciso fornecer uma visão geral abrangente, mas concisa, dos recursos e conceitos principais do JPA 2.1. O usuário quer uma "revisão", o que pode significar um resumo ou uma recapitulação. Devo abordar os conceitos fundamentais, as principais anotações, a API de Criteria, a linguagem JPQL, as funcionalidades introduzidas no JPA 2.1, e talvez alguns exemplos práticos.
>
> Como a solicitação está em português brasileiro, a resposta deve estar nesse idioma. Devo estruturar a resposta de forma lógica, começando com uma introdução ao JPA, passando pelos conceitos de entidade, mapeamento objeto-relacional, operações básicas, consultas, e finalizando com as novidades do JPA 2.1. Também posso incluir considerações sobre provedores JPA (Hibernate, EclipseLink, etc.) e boas práticas.
>
> Preciso manter um tom informativo e técnico, mas acessível. Vou organizar em seções com títulos claros. Vou evitar me alongar demais, mas cobrir os pontos essenciais. Vou incluir exemplos de código simples para ilustrar anotações e consultas. Vou mencionar recursos como EntityManager, transações, relacionamentos, herança, etc. E destacar as principais mudanças do JPA 2.1: suporte a consultas de atualização/delete em bulk com JPQL, stored procedures, conversores de atributos, etc.
>
> A resposta deve ser em português brasileiro, então preciso escrever de forma natural, usando terminologia técnica em português ou mantendo os termos em inglês quando apropriado (ex: "EntityManager", "JPQL").
>
> Vou começar a esboçar a resposta.

Revisão do JPA 2.1 (Java Persistence API)

Introdução

O **JPA (Java Persistence API)** é a especificação padrão da plataforma Java EE / Jakarta EE para mapeamento objeto-relacional (ORM) e gerenciamento de persistência de dados. O JPA 2.1 foi lançado como parte do Java EE 7 e trouxe diversas melhorias em relação às versões anteriores, mantendo a simplicidade e o poder do modelo de programação baseado em POJOs (Plain Old Java Objects).

O principal objetivo do JPA é reduzir a complexidade do desenvolvimento de camadas de acesso a dados, permitindo que o desenvolvedor trabalhe com objetos

Java e deixe a tarefa de sincronizá-los com o banco de dados relacional para o provedor de persistência (como Hibernate, EclipseLink, OpenJPA etc.).

Conceitos Fundamentais

1. Entidade (Entity)

Uma entidade é uma classe Java persistente que representa uma tabela no banco de dados relacional. Cada instância da entidade corresponde a uma linha da tabela.

```java
@Entity
@Table(name = "usuarios")
public class Usuario {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "nome_completo", nullable = false, length = 100)
    private String nome;

    private String email;

    // getters e setters
}
```

- `@Entity` – indica que a classe é uma entidade JPA.
- `@Table` – especifica detalhes da tabela (opcional se o nome da classe for igual ao da tabela).
- `@Id` – define a chave primária.
- `@GeneratedValue` – configura a estratégia de geração automática de valores para a chave primária.
- `@Column` – customiza o mapeamento da coluna (nome, nulabilidade, tamanho etc.).

2. EntityManager

O `EntityManager` é a interface principal para interagir com o contexto de persistência. Ele é responsável por operações de CRUD (Create, Read, Update,

Delete) e consultas.

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

// Iniciar transação
em.getTransaction().begin();
em.persist(usuario);
em.getTransaction().commit();

em.close();
emf.close();
```

3. Unidade de Persistência (Persistence Unit)

Definida no arquivo `META-INF/persistence.xml`, configura o conjunto de entidades gerenciadas, o provedor JPA, a fonte de dados e outras propriedades.

```xml
<persistence-unit name="meuPU" transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
    <class>com.exemplo.Usuario</class>
    <properties>
        <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/meubanco"/>
        <!-- outras propriedades -->
    </properties>
</persistence-unit>
```

Operações Básicas com EntityManager

- **persist(entity)** – torna a entidade gerenciada e a insere no banco.
- **find(classe, chave)** – recupera uma entidade pelo seu ID.
- **merge(entity)** – sincroniza o estado de uma entidade detached (desanexada) com o banco.
- **remove(entity)** – exclui a entidade do banco.
- **refresh(entity)** – recarrega o estado da entidade a partir do banco.

- **detach(entity)** – desanexa a entidade do contexto de persistência.
- **clear()** – limpa o contexto de persistência.
- **contains(entity)** – verifica se a entidade está gerenciada.

Todas as operações que modificam o banco devem ser executadas dentro de uma transação.

Mapeamento de Relacionamentos

O JPA suporta os principais tipos de associações entre entidades:

Anotação	Descrição	Exemplo de uso
`@OneToOne`	Relacionamento um-para-um	`Usuario` x `Endereco`
`@OneToMany`	Um-para-muitos	`Departamento` x `Empregado`
`@ManyToOne`	Muitos-para-um	`Empregado` x `Departamento`
`@ManyToMany`	Muitos-para-muitos	`Aluno` x `Curso`

```java
@Entity
public class Pedido {
    @ManyToOne
    @JoinColumn(name = "cliente_id")
    private Cliente cliente;

    @OneToMany(mappedBy = "pedido", cascade = CascadeType.ALL, orphanRemoval = tru
e)
    private List<ItemPedido> itens = new ArrayList<>();
}
```

- `mappedBy` – indica o lado inverso do relacionamento.
- `cascade` – propaga operações (persist, merge, remove etc.) para entidades associadas.
- `orphanRemoval` – remove automaticamente entidades filhas que foram removidas da coleção.

Herança no JPA

O JPA oferece três estratégias para mapear hierarquias de classes:

- **SINGLE_TABLE** – uma única tabela para toda a hierarquia, com uma coluna discriminadora.
- **JOINED** – uma tabela para a classe mãe e tabelas separadas para cada subclasse, unidas por chave estrangeira.
- **TABLE_PER_CLASS** – uma tabela por classe concreta (gera esquema menos normalizado).

```
java

@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "tipo_pessoa")
public abstract class Pessoa { ... }

@Entity
@DiscriminatorValue("F")
public class PessoaFisica extends Pessoa { ... }
```

Consultas: JPQL, Criteria API e SQL nativo

1. JPQL (Java Persistence Query Language)

Linguagem orientada a objetos, semelhante ao SQL, mas que opera sobre as entidades e seus atributos.

```
java

TypedQuery<Usuario> query = em.createQuery(
    "SELECT u FROM Usuario u WHERE u.email = :email", Usuario.class);
query.setParameter("email", "joao@email.com");
Usuario resultado = query.getSingleResult();
```

2. Criteria API

API baseada em objetos para construir consultas de forma dinâmica e type-safe. Útil quando a consulta depende de condições em tempo de execução.

```java
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Usuario> cq = cb.createQuery(Usuario.class);
Root<Usuario> root = cq.from(Usuario.class);
cq.select(root).where(cb.equal(root.get("email"), "joao@email.com"));

TypedQuery<Usuario> query = em.createQuery(cq);
Usuario resultado = query.getSingleResult();
```

3. SQL nativo

Permite executar SQL diretamente no banco, retornando entidades ou arrays de objetos.

```java
Query query = em.createNativeQuery(
    "SELECT * FROM usuarios WHERE email = ?1", Usuario.class);
query.setParameter(1, "joao@email.com");
Usuario resultado = (Usuario) query.getSingleResult();
```

Funcionalidades introduzidas no JPA 2.1

O JPA 2.1 adicionou diversos recursos importantes:

1. Consultas de atualização e exclusão em massa (Bulk Update/Delete) com JPQL

Possibilidade de executar operações em lote diretamente com JPQL, utilizando `UPDATE` e `DELETE`.

```java
em.createQuery("DELETE FROM Usuario u WHERE u.ultimoAcesso < :data")
  .setParameter("data", dataLimite)
  .executeUpdate();
```

2. Suporte a Stored Procedures

Facilidade para chamar procedimentos armazenados no banco de dados.

```java
StoredProcedureQuery query = em.createStoredProcedureQuery("nome_da_procedure");
query.registerStoredProcedureParameter("param1", String.class, ParameterMode.IN);
query.setParameter("param1", "valor");
query.execute();
```

3. Conversores de Atributos (Attribute Converters)

Permitem converter automaticamente um atributo Java para uma coluna do banco e vice-versa, útil para tipos personalizados ou enums complexos.

```java
@Converter
public class BooleanSimNaoConverter implements AttributeConverter<Boolean, String>
{
    @Override
    public String convertToDatabaseColumn(Boolean attribute) {
        return attribute ? "S" : "N";
    }

    @Override
    public Boolean convertToEntityAttribute(String dbData) {
        return "S".equals(dbData);
    }
}

// Uso na entidade
@Convert(converter = BooleanSimNaoConverter.class)
private Boolean ativo;
```

4. Entity Graph (Consultas com gráficos de objetos)

Permite definir dinamicamente quais atributos devem ser carregados (eager) em uma consulta, otimizando o desempenho e evitando o problema N+1.

```java
EntityGraph<Usuario> graph = em.createEntityGraph(Usuario.class);
graph.addAttributeNodes("perfis");

TypedQuery<Usuario> query = em.createQuery("SELECT u FROM Usuario u", Usuario.clas
```

```
s);
query.setHint("javax.persistence.fetchgraph", graph);
List<Usuario> usuarios = query.getResultList();
```

5. Melhorias nas consultas JPQL/Criteria

- Suporte a `JOIN ON` para condições adicionais no `JOIN`.
- Funções agregadas como `SUM`, `AVG`, `MAX`, `MIN` com possibilidade de usar expressões.
- Suporte a `INDEX()` em coleções ordenadas.
- Suporte a `CASE` expressions.

6. Schema Generation

Configuração para gerar ou atualizar automaticamente o esquema do banco a partir das entidades, através de propriedades como:

- `javax.persistence.schema-generation.database.action`
- `javax.persistence.schema-generation.create-source`
- `javax.persistence.schema-generation.drop-source`

7. CDI em Listeners de Entidade

Os listeners (classes anotadas com `@EntityListeners`) podem agora utilizar injeção de dependência CDI (se o ambiente suportar).

Considerações Finais

O JPA 2.1 consolidou-se como uma especificação madura e poderosa para persistência em Java, oferecendo um equilíbrio entre produtividade, flexibilidade e controle sobre o acesso a dados. Sua adoção é ampla tanto em aplicações Java EE completas quanto em ambientes Java SE com frameworks como Spring.

As principais implementações do JPA 2.1 incluem:

- **Hibernate 4.3+**
- **EclipseLink 2.5+** (implementação de referência)
- **Apache OpenJPA 2.3+**

Ao utilizar JPA, é importante compreender bem os conceitos de contexto de persistência, ciclo de vida das entidades e a diferença entre carregamento **eager** e **lazy**, bem como estratégias de otimização como *fetch joins* e *entity graphs*.

Esta revisão abordou os aspectos centrais da especificação. Para um aprofundamento, recomenda-se a leitura da especificação oficial ou de livros especializados como "Pro JPA 2" (Mike Keith, Merrick Schincariol).