Neste artigo veremos como utilizar o EntityManager do JPA para salvar objetos no banco de dados. Trataremos aqui de duas formas de persistência com o EntityManager: Persistência Explícita e Implícita.

O EntityManager é o recurso responsável por gerenciar os objetos/entidades da sua aplicação, ele também realiza as operações de CRUD com o banco de dados e faz consultas ao PersistenceContext para evitar consultar ao banco desnecessárias.

É importante saber que o PersistenceContext funciona como um Cache L1 para o EntityManager, sendo assim não fique surpreso se ao realizar uma chamada ao método “EntityManager.find()” não for disparada nenhuma consulta ao banco de dados, pois isso significa que ele retornou o objeto do Cache L1 (muito mais prático, rápido e eficaz).

Persistência Explícita

Conforme formos explicando sobre a persistência dos objetos usando o EntityManager, também daremos início ao ciclo de vida de um objeto, ou seja, em que estado ele se encontra.

Vamos ver um primeiro exemplo na **Listagem 1** de um objeto sendo persistido no banco de dados

**Listagem 1.**Executando persist no EntityManager

Employee employee = new Employee("Samuel", "Joseph", "Wurzelbacher");

em.getTransaction().begin();

em.persist(employee);

em.getTransaction().commit();

Quando você faz um “new Employee()”, o estado atual do objeto é **New** e ele não está sendo gerenciado pelo EntityManager. Ao iniciar uma nova transação e chamar o método “persist()”, você muda o estado do objeto de **New**para **Managed**, ou seja, agora ele está sendo gerenciado pelo EntityManager corrente, mas até o momento ele não foi persistido na base de dados. Por fim, ao realizar um commit da transação, o objeto gerenciado é inserido no banco de dados fisicamente.

Evitando IllegalArgumentException

Todas as entidades que são passadas como argumento para o método “persist(entity)” devem ser anotadas com @Entity do JPA, caso contrário você terá a exceção IllegalArgumentException. Porém, você pode passar uma entidade que não tem a anotação @Entity se esta estiver dentro de uma outra entidade que possui esta anotação (como uma propriedade), ou seja, funciona como um container, mas esta deve ser “embutida”.

Evitando TransactionRequiredException

Quando o método “persist()” é chamado, há uma obrigatoriedade em existir uma transação corrente, caso contrário, a exceção TransactionRequiredException é lançada, pois qualquer operação que modifique a base de dados (com o EntityManager) exige que esteja dentro de uma transação.

Evitando EntityExistsException

Ocorre quando uma entidade tenta ser persistida no banco de dados e já existe uma outra entidade do mesmo tipo e com o mesmo valor de chave primaria associada, sendo assim uma exceção EntityExistsException é lançada.

Esta exceção pode ocorrer quando for chamado o método persist(), onde neste caso a entidade com mesmo tipo e chave primaria já está sendo gerenciada pelo EntityManager (está dentro do PersistenceContext) ou quando é realizado o commit da transação e é checado que a mesma já encontra-se no banco de dados. Perceba que para este tipo de exceção temos dois momentos distintos: o de checagem direto no PersistenceContext relacionado a esta transação, e o outro de checagem direta na base, o que obviamente demanda mais tempo de processamento.

Persistência Implícita – Objetos Embutidos/Embedded

A partir daqui veremos como realizar a persistência de objetos de forma implícita. O código da **Listagem 2** mostra como fazendo usando essa persistência.

**Listagem 2.**Persistindo objetos embutidos

Employee employee = new Employee("Samuel", "Joseph", "Wurzelbacher");

Address address = new Address("Holland", "Ohio");

employee.setAddress(address);

em.getTransaction().begin();

em.persist(employee);

em.getTransaction().commit();

Temos no código acima um objeto Employee e outro Address, onde o Address está embutido no Employee. A lógica é: um empregado/Funcionário tem um endereço associado a ele. Acontece que para nossa lógica de negócio, nunca precisaremos inserir um Address sem um Employee, pois não faz sentido (para nosso caso). Então anotamos a entidade Address como @Embedded e ela será persistida automaticamente quando o Employee for persistido. É importante salientar que objetos embutidos não podem ser compartilhados entre entidades distintas, isso significa que este Address (seja um 001) não poderá ser usado em outra entidade.

Evitando IllegalStateException

O que acontece se colocarmos a anotação @Entity em vez de @Embedded no objeto Address e tentarmos executar o mesmo código da **Listagem 2**? Teremos então uma exceção IllegalStateException, pois estamos tentando persistir o objeto Employee sendo que ele faz referência a um objeto que ainda não foi persistido, o objeto Address. Assim, evita-se de ter objetos com pendências no banco de dados, o que “feriria” totalmente o principio de “Consistência de Dados”.

É responsabilidade da aplicação checar que ao tentar persistir um objeto, todos os objetos que estão sendo referenciados por este devem ser antes persistidos na base para evitar inconsistências, por isso, como uma segurança é lançada essa exceção. Você teria que usar uma persistência em cascata para evitar tal exceção, mas isso veremos a seguir.

Persistência Implícita – Cascata

Bom, falamos mais acima sobre como realizar a persistência implícita de um objeto embutido em outro, mas o problema de utilizar essa forma é que só poderemos persistir o Address se ele estiver dentro de um Employee e o persist() for chamado para o Employee e não para o Address. No caso da persistência em cascata podemos salvar o Address tanto pelo Employee como por ele mesmo. Veja como na**Listagem 3.**

**Listagem 3.**Persistência em Cascata

@Entity

class Employee {

:

@OneToOne(cascade=CascadeType.PERSIST)

private Address address;

:

}

Colocando a propriedade “cascade = CascadeType.PERSIST” acima, estamos explicitamente dizendo que ao salvar o objeto Employee no banco de dados, o Address também deve ser salvo caso ainda não esteja persistido. Mas além disso, podemos também persistir o objeto Address diretamente através do “persist(address)” sem necessitar do Employee, coisa que não podíamos fazer com entidades embutidas, como explicado nas seções anteriores.

Persistência Implícita – Global Cascade

No exemplo acima especificamos que um determinado atributo deve ser cascateado, e deveremos fazer isso para todos aqueles que são necessário, um a um. Normalmente a forma de se trabalhar é essa (especificar quais propriedades devem ser cascateadas), porém se você precisa que todas as referências sejam cascateadas você poderá fazer isso através da configuração da **Listagem 4**.

**Listagem 4.**Configurando o Global Cascade

<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm"

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

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

http://java.sun.com/xml/ns/persistence/orm\_1\_0.xsd" version="1.0">

<persistence-unit-metadata>

<persistence-unit-defaults>

<cascade-persist/>

</persistence-unit-defaults>

</persistence-unit-metadata>

</entity-mappings>

A linha cascade-persist está especificando que todas as referências deverão ser cascateados mesmo sem anotação.

Salvando grande quantidade de objetos – Batch

Pelo fato de o JPA trabalhar com o conceito de Cache L1, salvando sempre as entidades no PersistenceContext para evitar consultar ao banco, você deve ficar atento ao uso demasiado do espaço em memória.

Dada as explicações das sessões anteriores você já sabe que ao tentar realizar uma inserção no banco de dados, o EntityManager irá armazenar sua entidade no PersistenceContext para apenas no final da transação (commit) de fato inserir o seu objeto no banco de dados. Uma ótima estratégia para evitar sobrecarga no banco, mas esquecemos de uma coisa: Como fica a memória consumida com tantos objetos sendo gerenciados? E se tivéssemos que fazer uma inserção em lote, de um milhão objetos em uma única transação? Esses objetos seriam colocados no PersistenceContext, uma estratégia nem um pouco recomendada e passível de um “estouro de memória ou stack overflow”.

Então temos que pensar em uma estratégia mas conveniente para esses casos específicos, onde não podemos deixar que o PersistenceContext cuide sozinho de tantos objetos. Para isso, iremos trabalhar com uma técnica conhecida por “Batch Store” que tem por principio o sincronismo dos dados com o banco quando uma determinada quantidade de registros é atingida e consequentemente a “limpeza” do PersistenceContext.

Suponhamos então que desejamos inserir um milhão de objetos no banco de dados, em uma mesma transação. Nesse caso poderíamos usar a técnica da **Listagem 5.**

**Listagem 5.**Usando Batch Store

em.getTransaction().begin();

for (int i = 1; i <= 1000000; i++) {

Point point = new Point(i, i);

em.persist(point);

if ((i % 10000) == 0) {

em.flush();

em.clear();

}

}

em.getTransaction().commit();

Vamos as explicações do código acima.

1. O algoritmo acima divide os objetos em lotes de dez mil. A cada dez mil chamadas ao método “persist()” é executado um flush() e clear().
2. Quando atingimos dez mil objetos, temos dez mil objetos sendo gerenciados pelo PersistenceContext (Cache L1). Fazemos o flush() para sincronizar esses dados com o banco de dados e depois o clear() para limpar todos os objetos da memória, ou seja, do PersistenceContext. Se realizássemos o clear() antes do flush() então perderíamos todas as nossas alterações.
3. Vale lembrar que o flush() apenas grava os updates ou inserts que serão feitos mais tarde, quando executarmos o commit. Isso significa que os objetos sincronizados com o banco são apenas temporários e caso o commit não seja realizado, os dados serão perdidos.

Com isso concluímos como salvar nossos objetos no banco de dados usando as mais diversas estratégias disponíveis com o JPA + EntityManager, um poderoso recurso que nos fornece uma ótima manipulação de nossas entidades mas que pode tornar-se uma dor de cabeça se usado de forma errada.

[http://www.devmedia.com.br/imagens/fotoscolunistas/RLanhellas.jpg](http://www.devmedia.com.br/autor/ronaldo-lanhellas/244531)   
[Ronaldo Lanhellas](http://www.devmedia.com.br/autor/ronaldo-lanhellas/244531)

Bacharel em Ciência da Computação. Atuando como Analista de Sistemas Java.

O que você achou deste post?

[http://www.devmedia.com.br/imagens/portal2010/handUp.png](http://www.devmedia.com.br/java-persistence-api-salvando-objetos-com-entitymanager-do-jpa/30499)

[Gostei (1)](http://www.devmedia.com.br/java-persistence-api-salvando-objetos-com-entitymanager-do-jpa/30499)

[http://www.devmedia.com.br/articles/img/handDown.png](http://www.devmedia.com.br/java-persistence-api-salvando-objetos-com-entitymanager-do-jpa/30499)

[(0)](http://www.devmedia.com.br/java-persistence-api-salvando-objetos-com-entitymanager-do-jpa/30499)

[Seja](http://www.devmedia.com.br/rte3/addcomment.asp?idcomp=30499)

Leia mais em: [Java Persistence API: Salvando Objetos com EntityManager do JPA](http://www.devmedia.com.br/java-persistence-api-salvando-objetos-com-entitymanager-do-jpa/30499#ixzz30BXvk3Wh) <http://www.devmedia.com.br/java-persistence-api-salvando-objetos-com-entitymanager-do-jpa/30499#ixzz30BXvk3Wh>