02

Produzindo e descartando objetos

Transcrição

Agora vamos analisar alguns métodos da nossa classe `DAO`:

```
public void remove(T t) {
    EntityManager em = new JPAUtil().getEntityManager();
    em.getTransaction().begin();

    em.remove(em.merge(t));

    em.getTransaction().commit();
    em.close();
}

public void atualiza(T t) {
    EntityManager em = new JPAUtil().getEntityManager();
    em.getTransaction().begin();

    em.merge(t);

    em.getTransaction().commit();
    em.close();
}
```

Em todos os métodos a classe `DAO` cria uma instância da classe `JPAUtil`. Já havíamos discutido que a classe `DAO` conhece demais a classe `JPAUtil`. Vimos que se invertermos o controle, resolveremos o problema de ficar instanciando a `JPAUtil`. Então vamos receber essa classe no construtor e atribuir o valor recebido para um atributo:

```
private EntityManager em;

public DAO(Class<T> classe, EntityManager em) {
    this.classe = classe;
    this.em = em;
}
```

Em todos os métodos, vamos remover a instância da JPAUtil, pois agora `em` é um atributo da classe:

```
public void remove(T t) {
    em.getTransaction().begin();

    em.remove(em.merge(t));

    em.getTransaction().commit();
    em.close();
}

public void atualiza(T t) {
    em.getTransaction().begin();
```

```
        em.merge(t);

        em.getTransaction().commit();
        em.close();
    }
```

Agora que a classe `DAO` recebe um `EntityManager` no construtor, a classe `DAOFactory` para de funcionar:

```
@SuppressWarnings("unchecked")
public class DAOFactory {

    @Produces
    public <T> DAO<T> factory(InjectionPoint point) {

        ParameterizedType type = (ParameterizedType) point.getType();

        Class<T> classe = (Class<T>) type.getActualTypeArguments()[0];

        return new DAO<T>(classe);  // faltou o EntityManager
    }
}
```

Podemos passar o `EntityManager` aqui, em um só lugar:

```
return new DAO<T>(classe, new JPAUtil().getEntityManager);
```

Se reiniciarmos o servidor agora, essa abordagem irá funcionar. Porém agora acoplamos o `DAOFactory` ao `JPAUtil`. O ideal seria receber o `EntityManager` como dependência da classe `DaoFactory`. Como o `DaoFactory` é uma classe gerenciada pelo CDI, conseguimos injetar dependências dentro dela também.

```
@SuppressWarnings("unchecked")
public class DAOFactory {

    @Inject
    private EntityManager manager;

    @Produces
    public <T> DAO<T> factory(InjectionPoint point) {

        ParameterizedType type = (ParameterizedType) point.getType();

        Class<T> classe = (Class<T>) type.getActualTypeArguments()[0];

        return new DAO<T>(classe, manager);
    }
}
```

Vamos tentar reiniciar o servidor e ver o que acontece. Recebemos um erro:

```
WELD-001408: Unsatisfied dependencies for type EntityManager with qualifiers @Default
```

Já vimos esse erro antes, quando o CDI não sabia produzir um `DAO`. O CDI não está sabendo produzir um `EntityManager`. Nós já temos uma classe que sabe criar um `EntityManager`: a JPAUtil. Vamos torná-la um produtor de `EntityManager`. Basta anotá-la com a anotação `@Produces`:

```
public class JPAUtil {

    private static EntityManagerFactory emf = Persistence
            .createEntityManagerFactory("livraria");

    @Produces // produtor
    public EntityManager getEntityManager() {
        return emf.createEntityManager();
    }

    public void close(EntityManager em) {
        em.close();
    }
}
```

Uma coisa que temos que tomar cuidado quando produzimos um objeto e não definimos o tempo de vida do mesmo, a definição será feita por alguém, por padrão será feita pelo `@Dependent`. Então se tivermos um *bean* de sessão, teríamos um `EntityManager` de sessão. O problema é que o `EntityManager` representa uma conexão com o banco de dados. Se temos uma conexão durando uma sessão, vamos ter uma conexão que dura muito tempo.

Vamos restringir o escopo. Um escopo de request é suficiente. Para isso vamos utilizar a anotação `@RequestScoped`.

```
public class JPAUtil {

    private static EntityManagerFactory emf = Persistence
            .createEntityManagerFactory("livraria");

    @Produces
    @RequestScoped
    public EntityManager getEntityManager() {
        return emf.createEntityManager();
    }

    public void close(EntityManager em) {
        em.close();
    }
}
```

Ao reiniciar o servidor e tentar acessar a página de livros, recebemos um erro informando que o `EntityManager` está fechado.

```
HTTP Status 500 - EntityManager is closed
```

Por que será que isso está ocorrendo? Vamos dar uma olhada dentro de um trecho do nosso `DAO`:

```
public void adiciona(T t) {

    // abre transacao
    em.getTransaction().begin();

    // persiste o objeto
    em.persist(t);

    // commita a transacao
    em.getTransaction().commit();

    // fecha a entity manager
    em.close();
}
```

No método `adiciona()` e em todos os métodos da classe estão fechando o `EntityManager` chamando o método `close()`. O que ocorre é o seguinte: o CDI está produzindo o `EntityManager` e nós manualmente estamos fechando o objeto. A intenção é que apenas uma pessoa gerencie o objeto: ou nós tomamos conta do objeto dentro da classe `DAO`, ou somente o CDI, já que é ele que está produzindo o `EntityManager`.

Portanto vamos passar essa responsabilidade para o CDI, vamos ensiná-lo a fechar o `EntityManager` e parar de fechar nos métodos do `DAO`.

Na classe JPAUtil, já temos um método `close()` que recebe um `EntityManager`, ele verifica se está aberto e fecha com a chamada do método `close()`. Vamos anotar o parâmetro com a anotação `@Disposes`. Isso indica ao CDI que quando ele precisar descartar o objeto, poderá executar aquele método.

```
public void close(@Disposes EntityManager em) {
    if(em.isOpen()) {
        em.close();
    }
}
```

Na classe `DAO`, vamos remover todas as chamadas ao método `close()`, em todos os métodos da classe. Exemplo:

```
public void adiciona(T t) {

    // abre transacao
    em.getTransaction().begin();

    // persiste o objeto
    em.persist(t);

    // commita a transacao
    em.getTransaction().commit();

}
```

Ao reiniciar o Tomcat, tudo volta a funcionar. Agora o CDI está gerenciando todo o ciclo de vida o `EntityManager`: desde a criação ao descarte.