01

Injeção de dependências na estrutura de login

Transcrição

Ainda falta para alterar a estrutura de login da nossa aplicação, ou seja, as classes `UsuarioDao` e `LoginBean`. Vamos começar pelo `UsuarioDao`, nessa classe ainda estamos instanciando e fechando o `EntityManager`, mas essas duas responsabilidades agora são do CDI. Vamos remover o `EntityManager` atual e injetá-lo. Mas não podemos esquecer de remover a linha que fecha o `EntityManager` (`em.close()`) e de implementar `Serializable`:

```
public class UsuarioDao implements Serializable {

    @Inject
    EntityManager em;

    // restante do código
    // remova em.close();
}
```

E no `LoginBean`, vamos injetar o `UsuarioDao`:

```
@Named
@ViewScoped
public class LoginBean implements Serializable {

    @Inject
    UsuarioDao dao;

    public String efetuaLogin() {
        System.out.println("fazendo login do usuario "
                + this.usuario.getEmail());

        FacesContext context = FacesContext.getCurrentInstance();
        boolean existe = dao.existe(this.usuario);
        // restante do código comentado
    }
    // restante do código comentado
}
```

Produzindo e injetando o FacesContext

Reparem mais uma coisa na classe `LoginBean`, nos métodos `efetuaLogin` e `deslogar`, ambos criam o `FacesContext`, e como nós já sabemos isso o torna uma dependência da classe. E o que fazemos com as dependências? Injetamos! Então injete o `FacesContext` e remova suas declarações (`FacesContext context = FacesContext.getCurrentInstance();`):

```
@Named
@ViewScoped
public class LoginBean implements Serializable {
```

```
    @Inject
    FacesContext context;

    // restante do código
    // remova as linhas FacesContext context = FacesContext.getCurrentInstance();
}
```

Mas quem precisa criar o `FacesContext` é o JSF, então o CDI não pode criar esse `FacesContext`. Então o que iremos fazer? Algo parecido com o que fizemos com o `EntityManager`, vamos criar um *Producer*.

Vamos criar a classe `JsfUtil`, com um método que retorna um `FacesContext`. Lembrando que precisamos mostrar para o CDI que esse método é um **produtor** através da anotação `@Produces` e que queremos criar o `FacesContext` uma vez por requisição, através da anotação `@RequestScoped`:

```
public class JsfUtil {

    @Produces
    @RequestScoped // javax.enterprise.context.RequestScoped
    public FacesContext getFacesContext() {
        return FacesContext.getCurrentInstance();
    }

}
```

Na classe `LivroBean`, podemos também injetar o `FacesContext`. Injete-o e substitua `FacesContext.getCurrentInstance()` pelo nome do atributo injetado.

Podemos testar e ver que a injeção e produção do `FacesContext` está funcionando corretamente.