04

Produzindo o FacesContext

Transcrição

Pensando um pouco em como melhorar nossa estrutura, veremos que estamos utilizando o `FacesContext` para no contexto externo, manter as mensagens de *flash* configurando o `setKeepMessages` como `true`. Depois, estamos adicionando a mensagem de `Livro cadastrado com sucesso`.

Perceba que em ambos casos, estamos recorrendo ao mesmo contexto. Podemos melhorar isso fazendo com que o contexto seja armazenado em uma variável, em vez de recuperá-lo sempre. Nosso código que antes estava dessa maneira:

```
@Transactional
public String salvar() {
        for(Integer autorId : autoresId){
                livro.getAutores().add(new Autor(autorId));
        }

        dao.salva(livro);

        FacesContext.getContext().getCurrentInstace()
                .getExternalContext().getFlash().setKeepMessages(true);
        FacesContext.getContext().getCurrentInstace()
                .addMessage(null, new FacesMessage("Livro Cadastrado com sucesso"));

        return "/livros/lista?faces-redirect=true";
}
```

Ficará dessa forma:

```
@Transactional
public String salvar() {
        for(Integer autorId : autoresId){
                livro.getAutores().add(new Autor(autorId));
        }

        dao.salva(livro);

        context = FacesContext.getContext().getCurrentInstace();
        context.getExternalContext().getFlash().setKeepMessages(true);
        context.addMessage(null, new FacesMessage("Livro Cadastrado com sucesso"));

        return "/livros/lista?faces-redirect=true";
}
```

Podemos fazer também com que o objeto `context` seja reaproveitável em outros métodos transformando-o em um atributo da classe. Desta forma nosso código que antes criava um novo objeto, apenas faria sua atribuição.

```
context = FacesContext.getContext().getCurrentInstace();
```

Para isso precisaremos criar apenas este atributo.

```
public class AdminLivrosBean {
    private FacesContext context;
    // restante de código
}
```

O problema de deixar o código exatamente como está agora é precisar utilizar o atributo contexto fora do método `salvar()`. No método `getAutores()` por exemplo, caso utilizássemos o objeto, teríamos problemas porque ele não foi inicializado. Para resolver o problema, moveremos a atribuição do contexto para dentro do construtor da classe da seguinte forma:

```
public class AdminLivrosBean {
    // código anterior

    public AdminLivrosBean(){
        context = FacesContext.getContext().getCurrentInstace();
    }

    // restante de código
}
```

Desta forma, não precisaremos de nenhuma atribuição de contexto dentro dos métodos, visto que o construtor já fará isso automaticamente. Essa solução funciona porém podemos utilizar uma outra, que é pedir para o CDI injetar o objeto para nós. Foi isso que fizemos com os outros objeto. Por que não fazer com este também? Desta forma, descartaremos o construtor e depois, simplesmente anotar o objeto com o `@Inject`.

```
public class AdminLivrosBean {
    @Inject
    private FacesContext context;
    // restante de código
}
```

O problema é que o CDI e o JSF ainda não estão totalmente integrados e, por isso, a solução não funciona diretamente. Para que funcione, indicaremos para o CDI como criar o objeto que será injetado em nosso código. Por hora, ele não sabe criar o `context` sozinho, mas podemos ensiná-lo.

Vamos criar uma nova classe chamada `FacesContextProducer` no pacote `conf`. Nesta classe, teremos o método `getFacesContext` que retornará uma instância de `FacesContext` para cada *request*, ou seja, no escopo da requisição.

```
br.com.casadocodigo.loja.conf

public class FacesContextProducer{

        @RequestScoped
        public FacesContext getFacesContext(){
                return FacesContext.getCurrentInstance();
        }

}
```

Isto é tudo que a nossa classe que cria objetos de contexto precisa fazer, porém precisamos indicar para o CDI que este método faz exatamente isso: produz um objeto. Para isso utilizamos a anotação `@Produces` .

```
br.com.casadocodigo.loja.conf

public class FacesContextProducer{

        @RequestScoped
        @Produces
        public FacesContext getFacesContext(){
                return FacesContext.getCurrentInstance();
        }

}
```

Se testarmos agora, veremos que tudo continua funcionando e de forma simples, ganhamos a possibilidade de obter o `FacesContext` em qualquer outra classe da nossa aplicação. Precisaremos apenas declarar o atributo como sendo deste tipo e utilizando a anotação `@Inject` . Isso é injeção de dependência. Não precisamos mais fazer atribuições e também não precisaremos de construtores.