CETEJ35 - Java Web - JAVA_XXX (2024_01)

Integração

✓ **Feito:** Ver | **A fazer:** Gastar pelo menos 20 minutos na atividade | **A fazer:** Passar pela atividade até o fim

Fecha: segunda-feira, 2 dez. 2024, 00:00

Normalmente, um aplicativo real vai precisar mais do que salvar dados em memória - ele vai precisar de persistência em banco! É aí que entra o conteúdo dessa aula.

Data Transfer Object

DTO significa Data Transfer Object, e é um dos padrões frequentemente usados para transportar dados entre objetos. Mesmo sem perceber, nos estamos usando esse padrão desde o início da nossa aplicação.

A classe `Cidade`, que carrega os atributos `nome` e `estado` é um exemplo típico de DTO. Essa classe só existe para carregar os dados entre controlador e a interface gráfica (página Web). Talvez você esteja se perguntando: "Perae, mas essa classe não representava o 'modelo' do MVC"?

Sim, claro, dentro do contexto do MVC, essa classe tem a função de 'modelo'. Mas, de uma perspectiva maior, ela é só um DTO porque o propósito dela é só carregar dados. Além disso, o padrão MVC é bem mais antigo do que o conceito de DTO – popularizado no livro de padrões do J2EE.

Usar DTOs é mais comum do que se possa imaginar. Frequentemente, DTOs são usados na interface gráfica ou em métodos que precisam processar ou retornar vários valores de uma só vez.

No caso da nossa aplicação, a classe de entidade (persistente) e a classe DTO/modelo são quase idênticas. Isso também costuma ser comum em vários sistemas. Por isso, frequentemente, os desenvolvedores usam as classes de entidade como DTOs.

Apesar de comum, isso não é uma boa prática e deve ser evitado. É claro que parece trabalho dobrado, mas à medida que o aplicativo cresce, cresce também a diferença entre o que está na interface gráfica e o que está no banco. Normalmente, a interface gráfica agrupa dados de várias entidades.

Um dos problemas que temos nosso aplicativo agora é que os dados que vêm da interface gráfica estão em um DTO. Por isso, eles precisam ser transformados em uma entidade persistente. Fizemos isso de forma manual na aba anterior. Porém, esse não é o melhor caminho.

Uma boa prática é criar um método usando algo similar ao padrão prototype. Para isso, adicione dois métodos na classe `Cidade`. Ambos se chamam `clonar`, mas o primeiro usa os dados da instância `Cidade` atual e retorna um objeto do tipo `CidadeEntidade`, enquanto o outro recebe um objeto do tipo `CidadeEntidade` como parâmetro e retorna um objeto do tipo `Cidade`.

```java

    public CidadeEntidade clonar() {
        var cidadeEntidade = new CidadeEntidade();

        cidadeEntidade.setNome(this.getNome());
        cidadeEntidade.setEstado(this.getEstado());

        return cidadeEntidade;
    }

    public Cidade clonar(CidadeEntidade cidade) {

        return new Cidade(cidade.getNome(), cidade.getEstado());
    }
```

Esses métodos substituem os métodos que colocamos no controlador. Isso reduz o código no controlador e facilita a manutenabilidade. Veja como ficou o método `criar()` no controlador.

```java
@PostMapping("/criar")
public String criar(@Valid Cidade cidade, BindingResult validacao, Model memoria) {

    if (validacao.hasErrors()) {
        validacao
            .getFieldErrors()
            .forEach(error ->
                    memoria.addAttribute(
                        error.getField(),
                        error.getDefaultMessage())
                    );

            memoria.addAttribute("nomeInformado", cidade.getNome());
            memoria.addAttribute("estadoInformado", cidade.getEstado());
            memoria.addAttribute("listaCidades", repository.findAll());

            return ("/crud");
    } else {
        repository.save(cidade.clonar());
    }

    return "redirect:/";
}
```

```java
    } else {
        var novaCidade = new CidadeEntidade();
        novaCidade.setNome(cidade.getNome());
        novaCidade.setEstado(cidade.getEstado());

        repository.save(novaCidade);
    }

    return "redirect:/";
}
```

Você deve ter percebido que mesmo antes de inserir o método `clonar()`, o método `alterar()` funcionava com a `CidadeEntidade`. Por que isso? Porque o nome e tipo dos atributos são exatamente os mesmos! O Spring Boot usa isso como referencia para o mapeamento. Porém, considere que em cenários reais os nomes podem ser diferentes.

O código desenvolvido nesta Seção está disponível no **Github**, na branch `semana05-30-integracao-conversao-dto`.

Retroceder Avançar

◄ Verificação de aprendizado - Validação de Dados

Seguir para...

API Reativa ►