02

Controlando seus sistemas

Criando controllers e lógicas

Começaremos a implementar nosso sistema com um controle de produtos. No nosso caso, faremos primeiro uma listagem e, para isso, iremos precisar desse controlador.

É uma boa prática ter um controlador para cada *recurso* que sua aplicação disponibiliza na Web, como produtos, categorias, etc. Porém, esta não é uma regra fixa.

Para criar um controlador do VRaptor, basta criar uma Classe cujo nome termine em `Controller`. Por exemplo, `ProdutoController`. Além disso, precisamos anotar essa Classe com `@Resource`:

```
import br.com.caelum.vraptor.Resource;

@Resource
public class ProdutoController {

}
```

Todo método criado dentro desse controller será disponibilizado pelo VRaptor para o mundo de fora. A URL convencionada pelo VRaptor é a seguinte: **`/nome_do_controlador/nome_do_método`**. Por exemplo, se criarmos o método `lista()` dentro do controlador `ProdutoController`, a URL será `/produto/lista`.

Renderizando sua View

Quando a requisição chegar, o VRaptor executará o conteúdo desse método. Dentro dele, você pode fazer qualquer coisa, como, por exemplo, acessar seu banco de dados para pegar ou criar novos produtos. Ao final, o VRaptor exibirá para o usuário a JSP associada a esse método. Por convenção, a JSP deve ser armazenada no seguinte caminho: **`/WEB-INF/jsp/nome do controlador/nome do metodo.jsp`**. No caso do método `lista()`, ele deve ficar em `/WEB-INF/jsp/produto/lista.jsp`.

O VRaptor também permite que você envie valores do seu método para a JSP. Isso é muito útil quando queremos, por exemplo, exibir a listagem de produtos. Para isso, basta fazer seu método retornar o objeto que você quer exibir. No código abaixo, estamos retornando uma lista para a JSP:

```
@Resource
public class ProdutoController {

        public List<Produto> lista() {
            return new ProdutoDao().pegaTodos();
        }
}
```

Para acessar essa lista na JSP, basta você fazer uso de *Expression Language (EL)*. O nome da variável disponibilizada na JSP será `produtoList` (com EL: `${produtoList}`), já que essa é uma lista de produtos. Essa é outra convenção do VRaptor! Se o objeto retornado não for uma lista, então a variável terá o mesmo nome da classe do objeto. Por exemplo, caso o método lista retornasse um `Produto` apenas, o nome da variável seria `${produto}`.

Vamos criar o `lista.jsp` dentro da pasta `WebContent/WEB-INF/jsp/produto`. Nele vamos iterar pelos produtos e renderizar os dados de cada produto:

```
<html>
  <body>
    <table>
       <c:forEach var="produto" items="${produtoList}" >
       <tr>
        <td>${produto.nome}</td>
        <td>${produto.descricao}</td>
        <td>${produto.preco}</td>
      </tr>
      </c:forEach>
    </table>
  </body>
</html>
```

Recebendo parâmetros da requisição

O VRaptor também pode receber parâmetros passados pela URL, como http://meusite.com.br/meucontrolador/minhaacao?idade=18 (http://meusite.com.br/meucontrolador/minhaacao?idade=18). Nesse exemplo, o parâmetro passado é *idade* e possui valor 18. Para receber esse valor na *action*, é preciso colocar um parâmetro no método da *action*. Veja o código abaixo:

```
@Resource
public class ProdutoController {

        //método lista() omitido

        public Produto exibe(long id) {
            return new ProdutoDao().pegaPorId(id);
        }
}
```

O método `exibe()` recebe um parâmetro `id`. Isso significa que se invocarmos a URL http://localhost:8080/vraptor-produtos/produto/exibe?id=2 (http://localhost:8080/vraptor-produtos/produto/exibe?id=2), a *action* `exibe` será invocada, e o parâmetro *id* será populado com o valor 2. O VRaptor mapeia o nome do parâmetro na URL com o nome do parâmetro no método. Se quisermos passar mais de um valor, basta receber mais um atributo no método e passar mais um parâmetro na URL. Por exemplo: http://localhost:8080/vraptor-produtos/produto/exibe?id=2&tipo=abc (http://localhost:8080/vraptor-produtos/produto/exibe?id=2&tipo=abc). Chamamos esse esse conjunto de parâmetros de `querystring`.

URIs personalizados

O VRaptor ainda permite que você passa parâmetros através das chamadas "pretty URLs". Observe essa URL: http://localhost:8080/vraptor-produtos/produto/2 (http://localhost:8080/vraptor-produtos/produto/2). Queremos passar o valor 2, mas sem utilizar *query-string*. O VRaptor possibilita ao desenvolvedor pegar esses valores na URL de maneira simples. Basta anotar o método com `@Path` e informar como essa URL funciona. No nosso caso, sabemos que nossa URL será sempre assim: `/produto/{id aqui}`. É essa informação que passaremos ao VRaptor:

```
@Resource
public class ProdutoController {

    public List<Produto> lista() {
        return new ProdutoDao().pegaTodos();
    }

    @Path("/produto/{id}")
    public Produto exibe(long id) {
         return new ProdutoDao().pegaPorId(id);
    }
}
```

Veja o `{id}` : ele indica que esse trecho da URL deverá popular a variável de nome "id".

Refatorando o código

Observando os dois métodos `exibe` e `lista` percebemos que ambos instanciam um `ProdutoDao` . Podemos melhorar o código e criar o `ProdutoDao` em um lugar só. Vamos usar o construtor para instanciar o Dao:

```
@Resource
public class ProdutoController {

    private final ProdutoDao produtos;

    public ProdutoController() {
        this.produtos = new ProdutoDao();
    }

    public List<Produto> lista() {
        return produtos.pegaTodos();
    }

    @Path("/produto/{id}")
    public Produto exibe(long id) {
         return produtos.pegaPorId(id);
    }
}
```

Adicionando produtos

O VRaptor consegue popular objetos ainda mais complexos. Vamos criar o arquivo `formulario.jsp` , dentro da pasta `WebContent/WEB-INF/jsp/produto` , cuja função será adicionar um produto:

```
<html>
<body>
    <form action="<c:url value="/produto/adiciona" />" method="post">
        Nome: <input type="text" name="produto.nome" />
        Descrição: <input type="text" name="produto.descricao" />
        Preço: <input type="text" name="produto.preco" />
       <input type="submit" value="Adicionar" />
    </form>
</body>
</html>
```

E dentro do `ProdutoController` um método `formulario()` para chamar esse formulário:

```
@Resource
public class ProdutoController {

    //código omitido

    public void  formulario() {}
}
```

Veja que o formulário contém três campos: nome, descrição e preço. Eles representam um "produto". Então, nada melhor do que fazer o VRaptor receber um produto nessa *action* `adiciona()` :

```
@Resource
public class ProdutoController {

    //código omitido

    public void adiciona(Produto produto) {
        produtos.salva(produto);
    }
}
```

Repare a convenção: o nome do atributo no método é "produto"; o mesmo nos campos de texto no HTML: `produto.algumacoisa` . Vamos pegar o campo `produto.nome` : O VRaptor, ao encontrar esse campo de texto, busca o atributo que se chama `produto` e depois seta o atributo `nome` dentro desse objeto. Basta ir separando por "pontos" e o VRaptor vai entrando no objeto para setar os valores.

Mapeando os verbos HTTP

Além disso, vamos dizer ao VRaptor que o método `adiciona` só atende requisições do tipo *POST*. Para isso basta colocar a anotação `@Post` logo acima do método:

```
@Resource
public class ProdutoController {

    //código omitidos

    @Post
    public void adiciona(Produto produto) {
        produtos.salva(produto);
    }
}
```

O VRaptor também tem notações para os outros verbos HTTP como GET (`@Get`), PUT (`@Put`) e DELETE (`@Delete`).

Por fim, vamos criar um JSP para mostrar uma mensagem depois ter salvado um produto. Novamente, na pasta `WebContent/WEB-INF/jsp/produto` vamos criar um arquivo `adiciona.jsp` :

```
<html>
  <body>
    Produto adiconado com sucesso!
  </body>
</html>
```