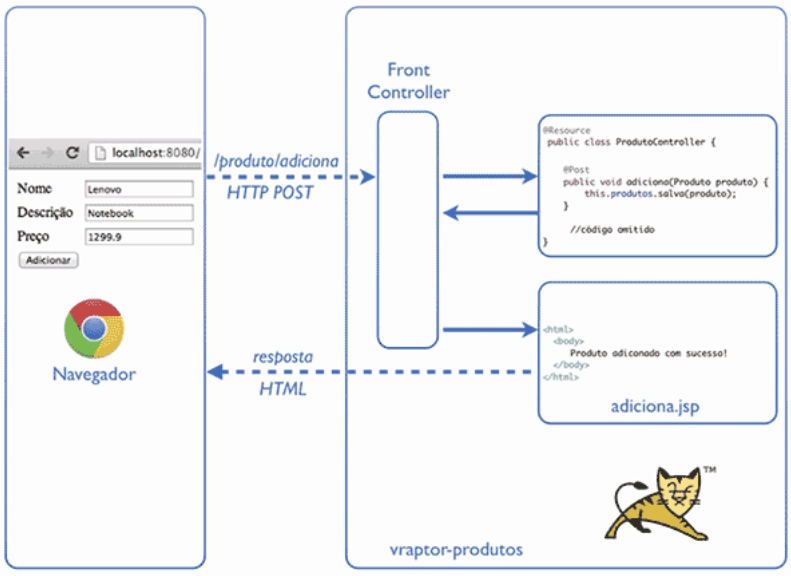
02

Controlando o resultado

Introdução

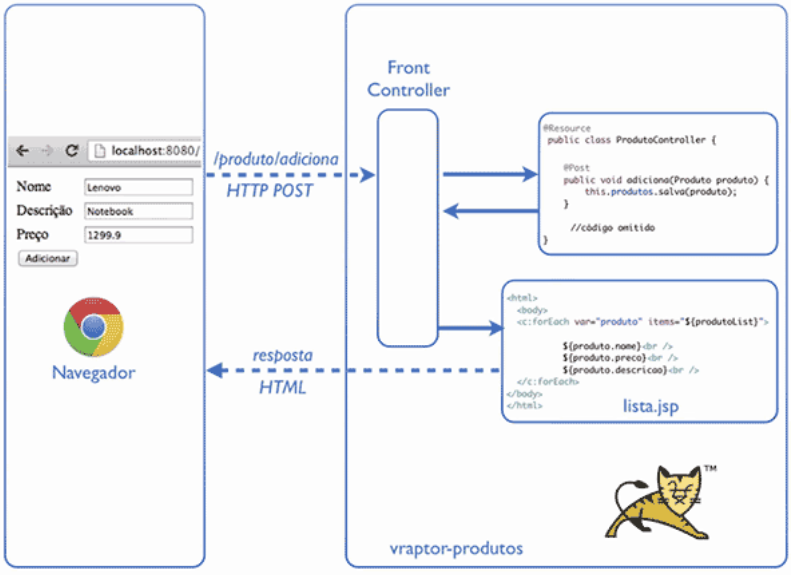
Vimos como funciona o fluxo básico de uma requisição no VRaptor. Ao enviar uma requisição (*request*) para, por exemplo, `/produto/adiciona`, ela é capturada pelo **FrontController** do VRaptor que, por sua vez, procura a classe e o método que atendam a URL acessada. Nesse exemplo, a classe é **ProdutoController** com o método **adiciona()**.

Após o *FrontController* chamar o método do controlador, o fluxo segue para a página JSP que possui o mesmo nome do método, no caso **adiciona.jsp**. Daí o JSP é renderizado e devolve um HTML como resposta (*response*).



Redefinindo o fluxo com o Result

Há situações onde queremos sair do fluxo padrão. Por exemplo, entendemos que faz mais sentido ir para a listagem de produtos cadastrados, após ter salvado um novo produto, do que voltar ao formulário de cadastro.



Para alterar o resultado do fluxo da execução, o VRaptor fornece um objeto do tipo **Result** que podemos usar para configurar o novo destino. Mas antes, é preciso receber esse objeto no construtor do seu controlador, e também colocá-lo como atributo da classe:

```
import br.com.caelum.vraptor.Result;
// outros imports omitidos

@Resource
public class ProdutoController {
    private final Result result;
    private final ProdutoDao produtos;

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

    // demais métodos omitidos
}
```

Com o objeto em mãos é possível chamar um outro método do controlador na sequência. Vamos usar o **result**, então, para redirecionar o fluxo para a listagem ao invés de exibir uma página de sucesso ou retornar para o formulário de cadastro. Para isso, basta avisar o VRaptor que, ao final da execução do método `adiciona()`, ele deve chamar o método `lista()`, usando o **`result.forwardTo(ProdutoController.class).lista()`**:

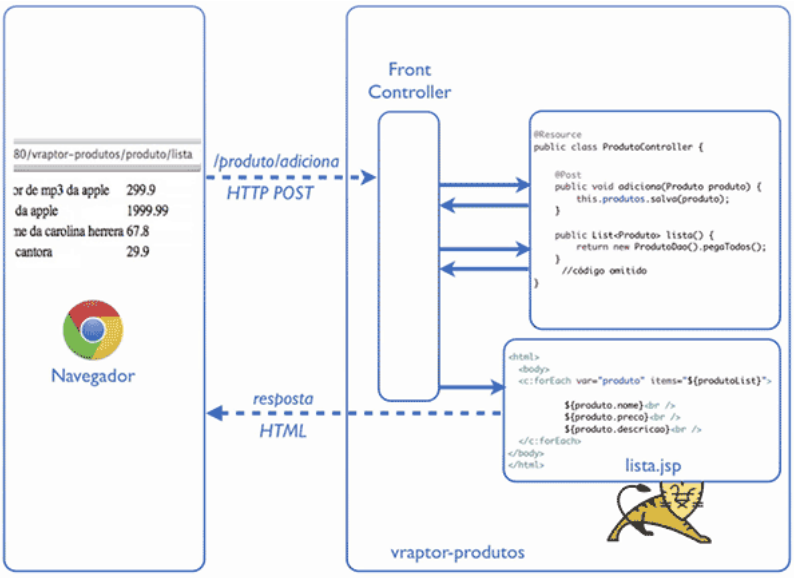
```
@Resource
public class ProdutoController {

    private final Result result;
    private final ProdutoDao produtos;

    //construtores e outros métodos aqui

    @Post
    public void adiciona(Produto produto) {
        produtos.salva(produto);
        result.forwardTo(ProdutoController.class).lista();
    }
}
```

Ou seja, não vamos mais direto para o JSP `adiciona.jsp`, mas sim chamar o método `lista()` do `ProdutoController` que, por sua vez, chamará o JSP para listar os produtos (`lista.jsp`)!



Antes de testar, vamos reiniciar o servidor Tomcat no Eclipse. Depois, abrimos o navegador e acessamos o formulário de cadastro pela URL `/produto/formulario` preenchendo-o com dados válidos. Ao submeter o formulário, repare que agora seremos automaticamente redirecionados para a listagem de produtos.

Incluir mensagens e outros valores no Result

Queremos apresentar uma mensagem ao usuário de que a produto foi salvo com sucesso. Por enquanto, para enviarmos alguma lista ou objeto ao JSP, utilizávamos o retorno dos métodos. Mas, para enviar mais de um item ao JSP, será necessário incluí-los ao mesmo `result` que usamos anteriormente para o redirecionamento. O objeto `result` possui um método `include()` para "pendurar" (atachar) qualquer outro objeto (de qualquer tipo). Repare que estamos usando esse método para incluir mais uma mensagem ao objeto `result`. O método `include()` recebe **2 parâmetros**: o primeiro indica o nome da variável usada no JSP e o segundo aponta para o objeto em questão.

```
@Resource
public class ProdutoController {

    private final Result result;
    private final ProdutoDao produtos;

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

    @Post
    public void adiciona(Produto produto) {
        this.produtos.salva(produto);
        result.include("mensagem", "Novo produto adicionado com sucesso!");
        result.forwardTo(ProdutoController.class).lista();
    }

    //outros métodos aqui
}
```

Agora, já no JSP, podemos acessar a mensagem atachada no `result` pela *expression language*, basta usar `${mensagem}` no arquivo **lista.jsp**:

```
<head>
    <title>Lista de produtos</title>
</head>
<body>
    ${mensagem}
    <table>
        <c:forEach var="produto" items="${produtoList}" >
        <!-- código omitido -->
```

Vamos testar novamente no navegador. Após ter preenchido o formulário, aparecerá a mensagem junto com a lista de produtos.

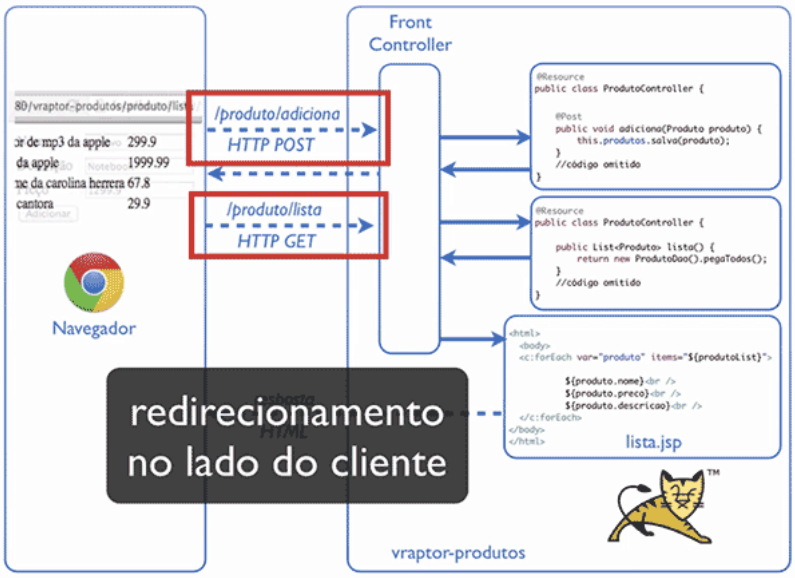
Forward ou Redirect?

Ao observar a URI no navegador, podemos ver ainda na barra de endereços: **/produto/adiciona**, mesmo já com a lista de produtos renderizada no navegador. Isso pode causar um problema! Ao apertarmos **F5**, repetiremos a última requisição, e os dados do formulário serão reenviados inserido, consequentemente, mais um produto.

Não podemos manter as coisas dessa maneira. No VRaptor, temos uma forma bem simples de resolver isso. Basta trocar a chamada do método **forwardTo()** para **redirectTo()** no método `adiciona()` .

```
@Post
public void adiciona(Produto produto) {
    produtos.salva(produto);
    result.redirectTo(ProdutoController.class).lista();
}
```

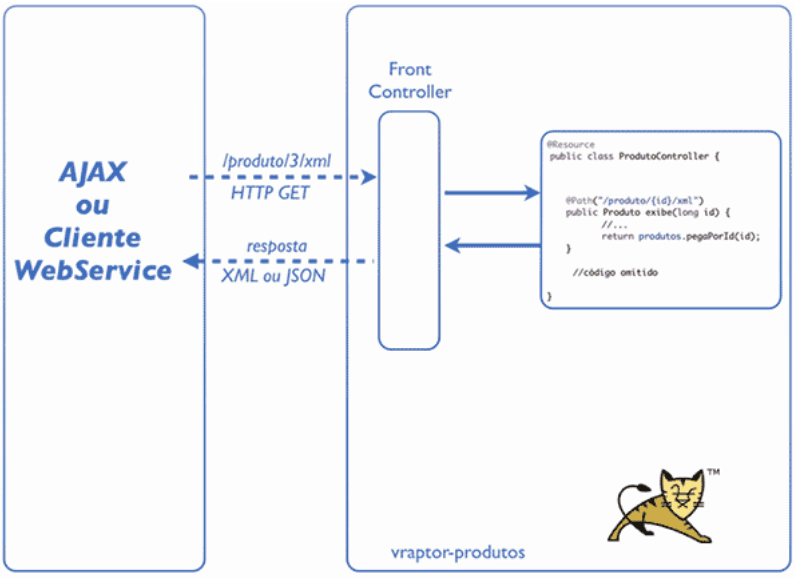
Desta forma, avisaremos ao controlador que queremos chamar o método `lista()` , mas através de uma nova requisição. Assim, após ter executado o método `adiciona()` , o VRaptor devolve uma resposta ao navegador avisando que se deve enviar um segundo *request* imediatamente. Depois continua tudo normalmente; o VRaptor chama o método `lista()` e o mesmo chama o JSP na sequência. Isso é chamado de "redirecionamento no lado do cliente", pois duas requisições foram feitas.



Vamos testar novamente no navegador: fazemos sua reinicialização e preenchemos o formulário, a lista de produtos aparece mas agora com a URL correta.

Serialização para XML ou JSON para AJAX

Além de tudo que vimos, o `result` também é usado para facilitar o envio de dados de requisições do tipo AJAX, ou mesmo para disponibilizar *WebServices* de maneira bem simples. Facilmente o VRaptor consegue transformar qualquer objeto ou lista de objetos em formatos **JSON** ou **XML**.



Para isso vamos usar novamente o `result` , mas primeiro criaremos um novo método na classe chamado **exibeComoXML**. O `@Path` será **/produto/{id}/xml**. O método não retornará nada diretamente, mas guardaremos o produto na variável local. Depois usaremos o **result** para definir o formato XML:

```
result.use(Results.xml())
```

Falta dizer qual objeto que queremos serializar (transformar) em XML. Para isso, existe o método `from()` que recebe o produto que carregamos anteriormente. Por fim chamamos o método `serialize()` . A linha completa ficará assim:

```
result.use(Results.xml()).from(produto).serialize();
```

Com isso o VRaptor transformará o produto em um XML e o devolverá na resposta. Vamos testar acessando a URL **/produto/3/xml**. No navegador aparecem os dados do produto como XML:

```
<produto>
  <id>3</id>
  <nome>212 for women</nome>
  <descricao>perfume da carolina herrera</descricao>
  <preco>67.8</preco>
</produto>
```

De forma análoga, podemos serializar o produto para JSON, um outro formato muito comum em requisições AJAX. Basta definir **Results.json()** e o VRaptor faz o resto. Testando também, mas agora com o URL **/produto/3/json**, o produto aparece em JSON:

```
{"produto":
  {"id": 3,
   "nome": "212 for women",
```

```
    "descricao": "perfume da carolina herrera",
    "preco": 67.8}
}
```

O código abaixo exemplifica a serialização de um produto em XML e em JSON no nosso **ProdutoController**:

```
import static br.com.caelum.vraptor.view.Results.*;

@Resource
public class ProdutoController {

    private final ProdutoDao produtos;
    private final Result result;

    //construtor e outros métodos omitidos

    @Path("/produto/{id}/xml")
    public void exibeComoXML(long id) {
        Produto produto = produtos.pegaPorId(id);
        result.use(Results.xml()).from(produto).serialize();
    }

    @Path("/produto/{id}/json")
    public void exibeComoJson(long id) {
        Produto produto = produtos.pegaPorId(id);
        result.use(Results.json()).from(produto).serialize();
    }
}
```