02

Primeiros passos com Spring MVC

Por que Spring MVC?

Hoje em dia, aplicações web dominam o planeta. Por esse motivo, criá-las não pode ser trabalhoso; precisamos ser produtivos. É fácil perceber que trabalhar com Servlets e JSPs puros não é tão produtivo e organizado. A própria Sun começou a fomentar o uso do padrão MVC e de patterns como Front Controller.

Era muito comum as empresas implementarem esses padrões e criarem soluções baseadas em mini frameworks caseiros. Mas logo se percebeu que o retrabalho era muito grande de projeto para projeto, de empresa para empresa. Usar MVC era bem interessante, mas reimplementar o padrão todo a cada projeto começou a ser inviável. Por esse motivo, "frameworks de caixinha" começaram a aparecer.

Um dos mais famosos no mercado é o Spring MVC. Spring é um framework que inicialmente não foi criado para o desenvolvimento web. Na essência o Spring é um container leve que visa fornecer serviços para sua aplicação como por exemplo o gerenciamento de objetos ou transação. Mas com o tempo a comunidade Spring entendeu que o Struts era ultrapassado e começou criar um framework MVC próprio. O Spring MVC é um framework moderno que usa os recursos atuais da linguagem além de usar todo poder do container Spring. Nesse capítulo veremos as funcionalidades desse framework poderoso.

Configurando o Spring MVC

Neste curso, trabalharemos em um sistema de contas a pagar e a receber. E o primeiro passo que precisamos dar é ter o Spring MVC para adicionarmos em nossa aplicação. Spring MVC vem junto com as bibliotecas do framework Spring que podemos encontrar no site http://www.springsource.org (http://www.springsource.org). Lá, é possível encontrar diversas documentações e tutoriais, além dos JARs do projeto. Uma vez que adicionamos os JARs do Spring MVC em nosso projeto dentro do diretório `WEB-INF/lib` , precisamos declarar um Servlet, que fará o papel de Front Controller da nossa aplicação, recebendo as requisições e as enviando às lógicas corretas.

Para declararmos o filtro do Spring MVC, basta adicionarmos no `web.xml` da nossa aplicação:

```
<servlet>
  <servlet-name>Spring MVC Dispatcher Servlet</servlet-name>
  <servlet-class>
    org.springframework.web.servlet.DispatcherServlet
  </servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
      /WEB-INF/spring-context.xml
    </param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>Spring MVC Dispatcher Servlet</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>
```

Repare que é uma configuração normal de Servlet, com `servlet-class` e `url-pattern`, como as que temos antes. Tem apenas um elemento novo, o `init-param`. Este parâmetro é uma configuração que pode ser passada para o Servlet pelo `web.xml`. Aqui definimos o nome do arquivo de configuração do framework Spring, o `spring-context.xml`. Quando o Servlet é carregado, ele vai procurar esse `spring context.xml` dentro da pasta `WEB-INF`.

O framework Spring possui sua própria configuração XML. O Spring, por ser muito mais do que um controlador MVC, poderia ser utilizado em ambientes não Web, ou seja, nem sempre o Spring pode se basear no `web.xml`. Por este motivo, mas não somente este, o Spring tem o seu próprio XML com várias opções para configurar a aplicação. A primeira coisa que faremos nesse arquivo é habilitar o uso de anotações do Spring MVC e configurar pacote base da aplicação web para o Spring achar as nossas classes:

```
<mvc:annotation-driven />
<context:component-scan base-package="br.com.caelum.contas" />
```

Além disso, é preciso informar ao Spring o local onde colocaremos os arquivos JSP. Para isso Spring MVC oferece uma classe especial que recebe o nome da pasta dos JSPs e a extensão dos arquivos. Vamos criar todos os JSPs na pasta `/WEB-INF/views/`:

```
<bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<property name="prefix" value="/WEB-INF/views/"/>
<property name="suffix" value=".jsp"/>
</bean>
```

Isso já é suficiente para começar com o Spring MVC. O arquivo completo, com todos os cabeçalhos, fica:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:mvc="http://www.springframework.org/schema/mvc"
xsi:schemaLocation="http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd">

  <context:component-scan base-package="br.com.caelum.contas" />
  <mvc:annotation-driven />

  <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/views/"/>
    <property name="suffix" value=".jsp"/>
  </bean>
</beans>
```

Nossa primeira lógica

Para criarmos nossa primeira lógica, vamos criar uma classe chamada `OlaMundoController`. Nela vamos colocar métodos que são as ações (Action). O sufixo Controller não é obrigatório para o Spring MVC, porém é uma convenção do mercado. Como utilizaremos Spring MVC baseado em anotações, é obrigatório que o seu Controller esteja dentro do pacote `br.com.caelum.contas` ou em um sub-pacote. No nosso caso, criaremos a classe dentro do pacote: `br.com.caelum.contas.controller`.

Dentro dessa nossa nova classe, vamos criar um método que imprimirá algo no console e, em seguida, irá redirecionar para um JSP com a mensagem "Olá mundo!". A classe deve ser anotada com `@Controller`, uma anotação do Spring MVC. Ela indica ao Spring que os métodos dessa classe são ações (Action). Podemos criar um método de qualquer nome dentro dessa classe, desde que ele esteja com a anotação `@RequestMapping`. A anotação `@RequestMapping` recebe um atributo chamado value que indica qual será a URL utilizada para invocar o método, como esse atributo já é o padrão não precisamos definir. Portanto, se colocarmos o valor `olaMundoSpring` acessaremos o método dentro do nosso `@Controller` pela URL http://localhost:8080/contas/olaMundoSpring (http://localhost:8080/contas/olaMundoSpring).

Vamos chamar o método `execute`, mas novamente poderia ser qualquer nome. Esse método deve retornar uma String que indica qual JSP deve ser executado após a lógica. Por exemplo, podemos retornar "ok" para enviar o usuário para uma página chamada `ok.jsp`. O método não deve retornar o sufixo da página, já que isso foi configurado no XML do Spring. Também lembrando que o Spring MVC procura as páginas JSP dentro da pasta `WEB-INF/views`.

Dessa forma, teríamos a seguinte classe:

```
@Controller
public class OlaMundoController {
  @RequestMapping("/olaMundoSpring")
  public String execute() {
    System.out.println("Executando a lógica com Spring MVC");
    return "ok";
  }
}
```

Um ponto importante a se notar é que podemos criar outros métodos que respondam por outras URL's, ou seja, várias ações dentro dessa classe (dentro do mesmo `@Controller`). Bastaria que nós utilizássemos novamente a anotação `@RequestMapping` nesses métodos.

Por fim, só precisamos criar a JSP que mostrará a mensagem "Olá mundo!". Basta criar o arquivo ok.jsp dentro da pasta `WEB-INF/views/`, que mapeamos anteriormente no XML do Spring. A JSP terá o seguinte conteúdo:

```
<html>
<body>
<h2>Olá mundo com Spring MVC!</h2>
</body>
</html>
```

Podemos acessar nosso método pela URL http://localhost:8080/contas/olaMundoSpring (http://localhost:8080/contas/olaMundoSpring). O que acontece é que após a execução do método o Spring MVC verifica qual foi o resultado retornado pelo seu método e procura despachar a requisição para a página indicada.

Esse será nosso dia a dia com o Spring MVC. Criaremos controladores e ações, para responder as requisições do usuário.