03

Subindo nossa aplicação web

Transcrição

Com o *Spring Boot* já instalado em nossas dependências, precisamos configurar a aplicação. Sendo esta uma aplicação web, precisaremos de um container que a execute, sendo este o *Tomcat*, *Jetty*, *Wildfly* ou qualquer um.

A vantagem de usar *Spring Boot* é que não precisamos mais nos preocupar com a instalação e configuração do projeto em um destes containers. Precisamos apenas configurar o *Spring Boot* para que inicie um container automaticamente e gerencie todos os nossos *Beans*. Faremos isso por meio das classes Java.

> Até um tempo atrás, as configurações não eram feitas usando as classes Java, mas sim por arquivos XML, o que não é mais necessário hoje. O único XML da nossa aplicação será o de dependências do Maven.

Criaremos então a classe `Configuracao` que irá configurar o *Spring* em nosso projeto. Esta será criada dentro do pacote `br.com.alura.listavip`.

```
package br.com.alura.listavip;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Configuracao {

}
```

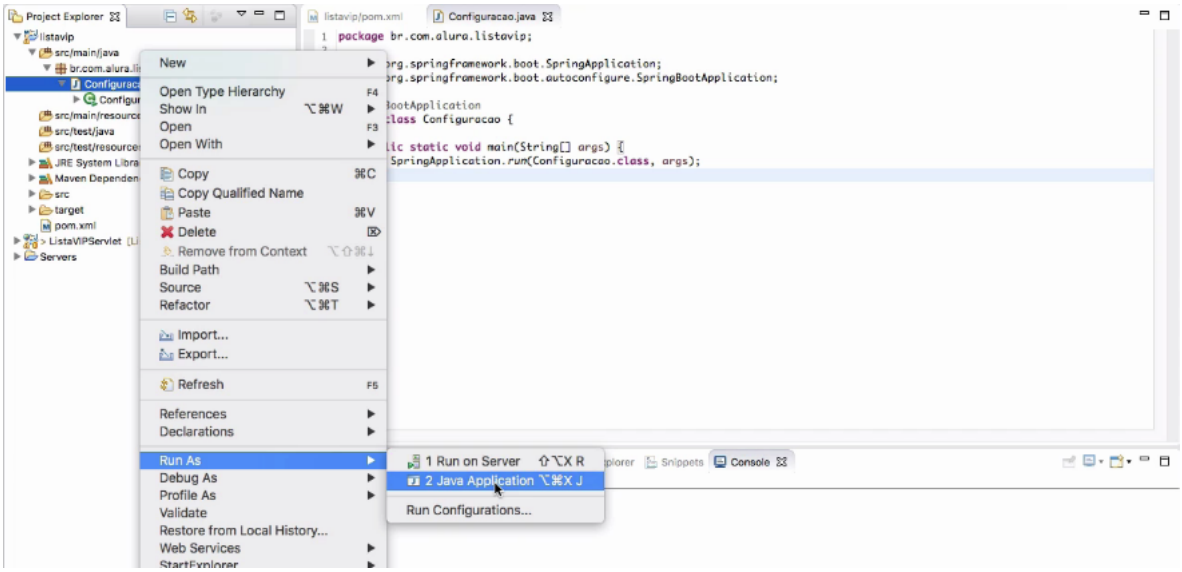
Note a presença da anotação `@SpringBootApplication`, ela é a responsável por configurar nossa aplicação *Spring*. Ela é o resultado de um aglomerado de outras configurações, como por exemplo, o diretório onde o *Spring* deve procurar todos os *Beans* da aplicação, entre outras configurações.

Agora temos nossa aplicação inicial pronta e já podemos executá-la. Mas como podemos executar uma aplicação Java com uma classe praticamente em branco? Não podemos! Toda e qualquer aplicação básica com Java precisa do método `main`. Criaremos um então.

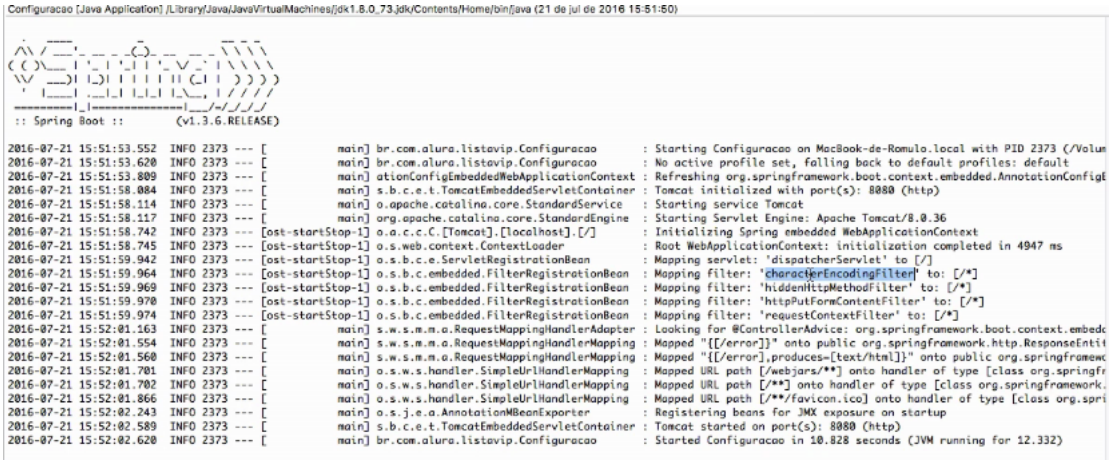
Dentro do método `main`, usaremos uma classe do *Spring Boot* para que este execute todas as configurações da nossa aplicação e a deixe disponível para podermos acessá-la, que receberá o nome de `SpringApplication`. Ela terá o método `run`, responsável por executar a classe de configuração da aplicação. Vejamos:

```
public static void main(String[] args){
    SpringApplication.run(Configuracao.class, args);
}
```

Como estamos na própria classe de configuração, passamos apenas o nome da classe. E como não temos nenhum parâmetro adicional para estas configurações, usamos apenas o `args` recebido pelo método `main`. Assim, já podemos executar nossa aplicação. Execute-a como uma aplicação Java normal.



Logo veremos algumas informações sendo impressas no console.



E então, podemos acessar nossa aplicação no navegador.

Mas convenhamos que mostrar uma página de erro logo no início da aplicação não é uma boa ideia para mostrar para o chefe ou amigo.

Para exibirmos uma mensagem mais interessante, transformaremos nossa classe de configuração em um `Controller`, e mapearemos o endereço `/` para um método que retornará uma *String* com uma mensagem de olá.

```
@SpringBootApplication
@Controller
public class Configuracao {

    @RequestMapping("/")
    @ResponseBody
    public String ola(){
        return "Ola, Bem vindo ao sistema Lista VIPs";
    }
```

```
    public static void main(String[] args){
        SpringApplication.run(Configuracao.class, args);
    }

}
```

As anotações `@Controller` , `@ResponseBody` , `@RequestMapping` , são específicas do *Spring MVC*. Caso não tenha conhecimentos sobre o framework, recomendamos que faça estes cursos:

- Spring MVC I (https://cursos.alura.com.br/course/spring-mvc-1-criando-aplicacoes-web)
- Spring MVC II (https://cursos.alura.com.br/course/springmvc-2-integracao-cache-seguranca-e-templates)

Ao iniciarmos a aplicação novamente, teremos.

Lembre-se de sempre separar seus `controller` s em outras classes. Neste exemplo, apenas usamos a classe de configuração para demonstração da praticidade de trabalhar com o *Spring Boot*.