03

Medindo o tempo de processamento

Transcrição

[00:00] Nosso objetivo aqui é fazer um Interceptor com o qual consigamos guardar as informações das requisições do usuário, tempo de processamento e quais as páginas que são mais acessadas. Na verdade, quais as páginas mais acessadas, média de tempo são informações que geramos em cima do conteúdo que vamos guardar com o Interceptor.

[00:20] Para criarmos um Interceptor nós estendemos uma classe chamada de `HandlerInterceptorAdapter`. Então vou criar aqui uma nova classe, não no pacote "Controller", mas vou colocar no pacote "Interceptor". Essa classe vai se chamar `InterceptadorDeAcessos`. E nós vamos estender `HandlerInterceptorAdapter`.

[00:57] Aqui nós podemos implementar alguns métodos. O que vamos implementar aqui é o `preHandler` e o `afterCompletion`, que se referem à antes de começar o processo, ou seja, antes de bater no `Controller`. Antes de chegar no `RestController`, e depois que a resposta foi processada, que é o `afterCompletion`. Vou reimplementar o `preHandler` e vou reimplementar o `afterCompletion`. Ótimo!

[01:29] O que vamos fazer logo no início? Precisamos das informações de acesso do usuário, que é o momento em que ele está acessando uma determinada página; e qual é a requisição que ele está fazendo, isso é o dado de acesso. Depois também vamos guardar a duração disso, mas precisamos fazer isso nesse momento aqui, no `afterCompletion`.

[01:54] O que vou fazer aqui é criar uma classe, eu vou deixar por enquanto só assim: `class Acesso {` . Nós vamos criar três atributos, que é o *path* que o usuário está tentando acessar, `private String path;` . No momento que ele está tentando acessar, com `LocalDateTime` , `private LocalDateTime data;` e a duração, com `Duration` , `private Duration duracao;` , do `java.time` também.

[02:26] Pronto, essas três informações já nos são suficientes. Eu só preciso criar aqui um acesso, representando o acesso do usuário nesse momento. Eu preciso passar duas informações: primeiro, qual é o *path* que o usuário está tentando acessar - e conseguimos isso com os dados da requisição.

[02:51] Você vê que no `preHandler` nós recebemos o `HttpServletRequest` , `HttpServletResponse` , o objeto `handler` aqui que ele está processando e no `afterCompletion` a mesma coisa. Só que caso tenha dado um erro, ele recebe uma *exception* também.

[03:06] E aqui é só pegar do `request.` o `getRequestURI();` , que é exatamente qual endereço que ele está tentando acessar. Por exemplo: "/usuario/pedido/aguardando", essa é a página que está tentando acessar. Então vamos guardar exatamente essa chamada que ele está fazendo. Depois vamos guardar a data, então: `acesso.data =` recebe `LocalDateTime.now()` .

[03:43] Só que, o que acontece? Isso é antes do processamento em si, que é um booleano. Então vamos ter que retornar `true` , porque se você retornar `false` , o processamento não segue em frente. Nós sempre retornamos `true` .

[03:59] Só que temos que acessar esse mesmo acesso aqui, no `afterCompletion` . Aí você pensa: "É só colocar como atributo do interceptador", não vamos colocar como atributo, podemos adicionar aqui no `request` e recuperar do `request` aqui. Nós não colocamos esse `Acesso acesso = new Acesso();` como atributo da classe. Então nós fazemos o quê? `request.setAttribute("acesso", acesso);` .

[04:31] E aqui, `Acesso acesso`. No `Completion` nós colocamos `Acesso acesso = (Acesso)`, aí convertemos do `request.getAttribute("acesso");`, o objeto que está salvo com o ID "acesso". Pronto!

[04:47] Agora, o que precisamos preencher? Precisamos preencher qual foi a duração do processamento, e nós utilizamos isso: `acesso.duracao = Duration.between`. Esse método `between` nos permite pegar. Então o início do processamento é `(acesso.data,` e o fim do processamento é `LocalDateTime .now());`. Agora já temos a informação do acesso.

[05:18] E o que nós fazemos? Uma coisa que você fazer é salvar esses dados de acesso em banco de dados; mas para simplificar, nesse caso, eu vou colocar em uma variável estática, `private static List<Acesso> acessos =` recebendo `new ArrayList`. Aqui no final eu coloco `acessos.add(acesso);`, essa informação de acesso aqui.

[06:01] Já temos o Interceptor configurado, só que esse Interceptor não vai funcionar automaticamente. Precisamos de alguma coisa para habilitar esse interceptador, para ele passar a funcionar.

[06:13] Por isso fazemos uma classe de configuração. Eu vou chamá-la de `WebConfig` e vou estender aqui uma classe que nos ajuda a fazer a configuração de Interceptor, que é a `WebMvcConfigurationSupport`. É só anotar aqui como `@Configuration`, para ser reconhecido aqui pelo Spring.

[06:39] E nós implementamos o método `addInterceptors`, que já recebe o `InterceptorRegistry` e a única coisa que precisamos fazer é digitarmos `registry.addInterceptor(new InterceptadorDeAcessos())`. Passamos também qual é o *path* que ele vai interceptar, ou seja, que requisições ele vai interceptar. Podemos colocar aqui todas as requisições.

[07:14] Lembrando que você pode passar vários *paths* diferentes, que ele recebe uma lista de *paths* aqui. Então pronto, é isso! Você pode passar uma o mais configurações dessa daqui, de *pattern*.

[07:27] Isso aqui já é o suficiente, a aplicação vai reiniciar, já para habilitar o nosso interceptador. E podemos navegar aqui, ele já vai salvando as informações lá, não tem problema.

[07:40] Beleza, mas cadê as informações? Onde eu pego essas informações? Onde o usuário está acessando? Precisamos expor essas informações aqui. Quais informações? Essa lista, `private static List<Acesso> acessos = new ArrayList<Acesso>();` , para que na interface gráfica possamos implementar alguma coisa, jogamos um JSON com os dados de acesso, criamos uma interface gráfica de um *dashboard* da aplicação. Podemos fazer várias coisas aqui.

[08:09] E para simplificarmos, para conseguirmos visualizar esses dados, eu vou criar um `RestController` aqui, vou chamar de `AcessosRest` , `@RequestsMapping` , vou chamar de `("acessos")` . Esse é um `@RestController` . O único método que vai ter aqui vai ser o `public List<Acesso>` , que vai dar o `getAcessos` , que vai pegar todos os acessos que estamos salvando e retornar.

[08:45] Qual é a questão aqui? Ele não consegue visualizar os acessos, nem se eu der um *import* e apertar "Ctrl + Shift + O" para importar automaticamente. Por quê? Porque a classe `Acesso` foi implementada dentro do interceptador de acessos e ela é uma *inner class*.

[09:03] O que podemos fazer é deixá-la - `public static class Acesso {` , que aí conseguimos usar as teclas de atalho "Ctrl + Shift + O" aqui e ele importa o acesso corretamente. Então, `getAcessos()` .

[09:17] E agora o que fazemos? Como isso aqui é uma variável estática, `private list List<Acesso> acessos = new ArrayList <Acesso>();` , eu pego aqui o interceptador de acesso e faço um `return InterceptadorDeAcessos.acessos;` .

[09:30] Só que o interceptador não foi importado e o `acessos` não é acessível, porque ele está privado. Nós vamos colocá-lo para `public` . Salvei. Agora se e

fizer um `/acessos` ele vai funcionar. Será que vai mesmo? Vamos ver.

[10:11] Vamos derrubar e subir de novo, para ver se o problema é de carregamento da configuração. Vou entrar de novo no "Meus Pedidos", vou me logar, ok, "Meus Pedidos", "Home". Vamos ver se eu consigo acessar o `acessos`. Ainda não, o que faltou? Está direto, eu não esqueci de colocar o `/api`. É isso, tem que colocar o método `@GetMapping` e associar o método à resposta.

[11:12] Pronto, agora vamos conseguir fazer a chamada! Ele não tem nada salvo ainda, vamos acessando aqui as páginas e quando atualizar... Ele deu erro, ele não está conseguindo converter o acesso. Por quê? Porque `acesso` não tem os *getters* para ele conseguir gerar o JSON.

[11:35] Então nós fazemos aqui um "Ctrl + 3 + GGAS" no Eclipse, que ele gera os *getters and setters*. Será que agora vai? Vamos ver. Acessar mais páginas aqui... Agora foi. Então `/acessos`, 0.004 segundos, `/usuario/pedido`, `/oferta`, `/api/pedidos/aguardando`.

[12:05] Vamos entrar em "Home" tres vezes. É para `home` três vezes lá embaixo. Está aqui, então ele está aguardando. Esses números são a data convertida para JSON, ano, mês, dia, hora, minuto, segundo, nanossegundo... 0.041. Depois diminui para 0.013.

[12:24] É isso! Conseguimos então guardar informações com o Interceptor e expor essas informações depois com um `RestController`. Lembrando que você pode salvar e o ideal é você salvar esse banco de dados, porque a aplicação uma hora vai cair por causa disso daqui, `public static List<Acesso> acessos = new ArrayList<Acesso>();`.

[12:43] Se você colocar isso em produção e os usuários começarem a acessar teu site, ele vai salvando isso em memória, vai salvando, essa lista vai crescendo e em hora nenhuma você limpa essa lista. A máquina virtual não vai limpar, exatamente porque está ligada a um objeto vivo.

[13:00] Então o ideal é salvar isso em banco de dados, até para fazer consultas no banco, por exemplo: "Eu quero saber quantas requisições para *home*, qual a média de tempo de acesso" etc.

[13:12] Para saber mais com relação ao monitoramento, pesquisem por "spring boot actuator". O Spring Boot Actuator traz várias *features* para conseguirmos monitorar a nossa aplicação, ele é bem legal para fazer métricas e tudo mais.

[13:34] Aqui abrimos já a documentação dele e você só precisa habilitar e você já consegue monitoramento de várias coisas da tua aplicação. Aqui na Alura tem curso sobre ele, por isso que não vamos entrar em detalhes aqui. Até mais!