04

# Configurando o Redis

### Transcrição

[00:00] Nós configuramos no Eclipse o segundo Tomcat para simular o que nós temos no ambiente de produção da Amazon onde temos os dois servidores com aplicação da Casa do Código. Criamos e utilizamos os recursos dos "perfis" do Spring, e configuramos na nossa classe "JPA configuration" um método do "data source" para realizar a conexão com um banco no nosso computador local da gravação, e o outro perfil de produção com as configurações do banco que nós havíamos feito anteriormente na Amazon.

[00:34] Certificamos que na nossa classe "servletSpringMVC", ainda estamos utilizando o perfil de "dev" de desenvolvimento, que estamos fazendo o teste inicialmente na nossa máquina local. E vamos fazer um teste para recriar aquele cenário que estávamos tendo na Amazon, antes de configurar o sticky session, que era quando colocava um livro no carrinho de compras e o balanceador poderia redirecionar para outra instância, e não teríamos mais a informação do nosso livro no carrinho de compras.

[01:03] Então vamos inicializar os dois Tomcat para tentar recriar essa simulação do que aconteceu na Amazon. Vamos esperar o primeiro Tomcat terminar de inicializar, e inicializamos o segundo Tomcat para fazer o teste. Vamos para o segundo Tomcat, inicializamos, e recriamos o cenário que estava tendo na Amazon antes de configurar o sticky session. Vamos esperar mais um pouco, para o segundo Tomcat terminar de inicializar, e fazemos esse teste.

[01:35] Agora temos os dois Tomcat rodando, um na porta "8080" e o outro na porta "8081". Vamos abrir uma janela no nosso browser, e colocamos "localhost:8080/casadocodigo". Estamos acessando a primeira instância do Tomcat. Temos a informação desses dois livros, porque agora estamos nos conectando no banco do computador da gravação, no banco local, e nos primeiros vídeos tínhamos cadastrado no banco local esses dois livros, da "Amazon aws" e do "Arduino prático".

[02:06] O que acontecia quando estávamos trabalhando na Amazon, antes de configurar o sticky session. Tínhamos, por exemplo, que queríamos comprar esse livro, e colocávamos que queríamos comprar o livro impresso. Quando clicamos no botão comprar, o que podia acontecer?

[02:22] O balanceador poderia vir aqui e redirecionar a nossa requisição para outra instância, por exemplo aqui, a "8081". E acontecia esse caso de não ter o nosso livro no carrinho de compras, porque a sessão está vinculada a instância, e acabava tendo esse problema.

[02:40] Para resolver essa questão e garantir o balanceamento, vamos utilizar o "redis", para armazenar essa sessão do usuário. Vamos fazer essa configuração do"redis". Vamos voltar para o Eclipse, vimos que conseguimos fazer a simulação do problema que estava correndo na Amazon.

[03:00] Vamos parar o Tomcat, para poder fazer essa configuração para que a nossa aplicação da Casa do Código faça o envio da sessão para o "redis". Vamos esperar o segundo Tomcat terminar. Então para realizarmos essa configuração, vamos aqui mesmo no nosso pacote configuration, e vamos criar essa classe de configuração do "redis".

[03:22] Então vou colocar Ctrl+N, colocamos "class", que queremos criar a classe da configuração do "redis", vamos chamar de "RedisConfiguration". Para podermos fazer essa configuração, precisamos adicionar algumas dependências ao nosso projeto. Então já temos as dependências necessárias para poder salvar essa sessão do usuário no "redis" e realizar essa comunicação. Logo abaixo do vídeo nos exercícios, vai ter essa dependência também para adicionar no projeto.

[03:53] Vamos copiar as dependências que precisamos adicionar, vamos lá no "pom.xml". Antes de fechar a aba "dependices", adicionamos essas duas dependências. Vamos colocar Ctrl+Shift+F para identar, Ctrl+S para salvar, e começar a baixar as dependências necessárias para salvar esse sessão do usuário no "redis".

[04:22] Agora já colocamos as dependências no nosso projeto, podemos fazer as configurações necessárias para salvar sessão do usuário lá no "redis". Para poder salvar a sessão do usuário no "redis", o primeiro passo é estabelecer uma comunicação. Por isso precisamos de um objeto que vai ser uma fábrica de conexões com o "redis". Esse objeto é do tipo "JedisconnectionFactory".

[05:51] Vamos ter que colocar o método, para retornar esse objeto "JedisconnectionFactory", e colocamos o nome do nosso método também como "jedisconnectionFactory". E para poder estabelecer essa comunicação, essa fábrica de conexões, precisamos ter um pool de conexões disponíveis. Então temos que instanciar a classe, que é a "JedisPoolConfig", para poder ter essa configuração do pool de conexões disponíveis para essa fábrica de conexões com "redis" que vamos precisar se comunicar.

[05:30] Então vou colocar Ctrl+1 para assinalar a variável local que nós vamos chamar somente de pool. Quando instanciamos essa classe, se for clicar em Ctrl e o botão esquerdo do mouse, ela já configura e coloca um pool de conexões padrão para poder estar utilizando e realizar essa comunicação com o "redis". Então não precisamos mudar nada, podemos deixar esses valores default do construtor da classe "JedisPoolConfig".

[06:03] Agora vamos aqui e instanciamos a nossa classe "JedisconnectionFactory". Colocamos "JedisconnectionFactory", para termos essa fábrica de conexões com o "redis", e no construtor passamos o nosso pool de conexões para poder realizar a comunicação com o "redis".

[06:23] Colocamos Ctrl+1, assinala a variável local, e vamos chamar de "factory", a nossa fábrica de conexões. E temos que especificar com qual "redis" estamos querendo nos comunicar. Vamos trabalhar com o "redis" na nossa máquina local. Então vou falar qual é o "host name" e a porta de comunicação com esse "redis".

[06:44] Então chamamos o método do "setHoneName", que é o localhost, estamos trabalhando na máquina local, e vamos dizer que a conexão vai ser feita com a porta, chamamos o método "setPort". E a porta padrão utilizada pelo "redis" é a "6379". Depois podemos vir aqui e retornar a nossa "Factory", que é do tipo "JedisconnectionFactory". Vamos pedir, assim como fizemos na classe do "amazonConfiguration". Vamos pedir para o Spring fazer o gerenciamento desse objeto para nós. Vamos aqui e anotamos esse método com o "@Bean".

[07:28] Agora temos a configuração relacionada a conexão com o "redis". E agora, temos essa conexão, e precisamos trabalhar na questão de escrever as informações no "redis". Precisamos criar um modelo para poder enviar, escrever essas informações no"redis".

[07:52] O "redis" trabalha com uma estrutura parecida com o "map" aqui no Java, com chave e valor. Temos que criar esse modelo indicando como essas informações serão escritas no "redis" com essa chave e com esse valor.

[08:04] Vamos precisar de um objeto que é o "redis template", o modelo do "redis". Temos que passar a chave e o valor. A chave vai ser do tipo string e o nosso valor vai ser do tipo object. Chamamos o nosso método também de "redisTemplate". O primeiro passo que temos que fazer é instanciar essa nossa classe do "redis template". Falamos que será a chave string e o nosso valor vai ser do tipo object. Colocamos Ctrl+1, e assinala essa variável local que vamos chamar de "template".

[08:42] Agora que temos esse modelo que vai estar estruturado o nosso "redis", temos que falar para esse "template" qual a conexão que será utilizada para poder escrever essas informações no "redis". Já fizemos isso nesse método, já fizemos a nossa "fábrica de conexões". Podemos vir e falar pra quando for escrever essas informações no "redis" ser utilizado essa "fábrica de conexões" que nós fizemos no método anterior.

[09:11] Vamos colocar aqui a "fábrica de conexões" que vamos utilizar será desse método "JedisconnectionFactory", já chamar esse método. Agora precisamos encontrar uma forma de serializar essa chave valor para o "redis". Temos que chamar o método para fazer essa serialização. Queremos fazer a serialização da chave, chamamos o método "setKeySerializer", e temos que instanciar a classe que é "StringRedisSerializer", para poder fazer essa serialização e enviar as informações para o "redis".

[09:51] E temos que fazer a mesma coisa para o valor. Fizemos a chave, e temo que fazer a mesma coisa para o valor. Colocamos "setValueSerializer", e temos que instanciar a classe que é a "GenericToStringSerializer", e falamos que vamos passar o "object.class" que é o nosso valor, e aqui também colocamos "object".

[10:22] Com isso estamos fazendo a serialização tanto na nossa chave, quanto do nosso valor, para poder ser enviado para o "redis". Agora falta retornar esse nosso objeto do tipo "redis template". Vamos aqui e retornamos o nosso "template". E a mesma coisa que fizemos anteriormente, vamos pedir para o Spring fazer o gerenciamento desse objeto para nós e anotamos esse método como "@Bean".

[10:50] Temos aqui a "fábrica de conexões" e especificamos esse modelo para realizar a serialização do envio da chave do valor para o "redis". O que acontece, estamos trabalhando com as dependências do "Spring session". E o "Spring session", por padrão vai tentar escutar modificações que podem ocorrer nas sessões do lado do "redis".

[11:15] Só que em ambientes de produção como a Amazon, esse "listeners" que o "Spring session" vai ter que usar para escutar essas mudanças que podem ocorrer do lado do "redis" não são permitidas. Precisaríamos ter o privilégio de "usuário administrador".

[11:29] Temos que falar para o "Spring session" desabilitar esses "listeners", para não ficar escutando essas mudanças que podem ocorrer na sessão no lado do "redis". Então temos que vir e fazer essa configuração para que o "Spring session" não fique escutando essas notificações que podem ser recebidas.

[11:48] Temos que chamar o retorno do método "ConfigureRedisAction", para que não fique escutando essas notificações de alteração de sessão que podem ocorrer do lado do "redis". Não queremos ficar escutando isso, porque podemos ter um problema na Amazon, já que a Amazon não permite esse tipo de configuração do "Spring session" de configurar esses "listeners", para poder escutar essas alterações que podem ocorrer na sessão do lado do "redis".

[12:17] Simplesmente retorno o "ConfigureRedisAction" e falamos para não realizar nenhuma operação, para não ficar configurando esses "listeners" para escutar as alterações que podem ocorrer nas seções do lado do "redis". Mesma coisa, vamos pedir para o Spring fazer o gerenciamento desse objeto para nós. Colocamos aqui a notação "@Bean".

[12:40] Agora já está quase completo. Para poder de fato falar para o "redis" armazenar essa sessão do usuário, temos que vir aqui e colocar a notação na nossa classe do "EnableRedisHttpSession", e temos que estender essa classe, e falar que o "RedisConfiguration", vai estender a classe do "AbstractHttpSessionApplicationInitializer". Com isso quando a nossa aplicação for inicializada, ela já vai saber que o "redis" vai ser responsável por estar armazenar a sessão do usuário.

[13:15] Agora para finalizar só falta dizer que essa classe "RedisConfiguration" vai ter que expor esses beans que configuramos. Vamos colocar a notação na nossa classe "@Configuration". Com isso o que estamos fazendo? Estamos estabelecendo a comunicação da nossa aplicação, dizendo que a sessão do usuário deverá ser armazenada no "redis". Vamos agora fazer um teste e verificar se resolvemos o nosso problema.

[13:49] Temos o "redis" instalado nesse meu computador da gravação, e logo abaixo dos vídeos nos exercícios, eu também coloco como você trabalha com "redis" no seu sistema operacional e como você consegue fazer a instalação. Eu já tenho ele instalado, vamos só inicializar. Vamos inicializar o "redis-server". Vou inicializar, e como falei, nos exercícios eu deixo os passos para você realizar a instalação.

[14:22] Então eu já tenho o "redis" rodando, e agora basta fazer o teste para ver se a nossa aplicação vai salvar a sessão do usuário no "redis", e se conseguimos corrigir aquele problema do balanceador poder enviar uma requisição para outra instância, e manter no livro no carrinho de compras.

[14:42] Vamos só inicializar os dois Tomcat□s, e fazemos esse teste. Vamos esperar o primeiro Tomcat terminar de inicializar, e vamos inicializar o segundo Tomcat também. E vamos fazer esse teste, vamos ver se o nosso livro que vamos comprar vai ser mantido no carrinho de compras independentemente do balanceador redirecionar para a primeira ou a segunda instância da Amazon.

[15:12] Os dois Tomcat estão rodando. Vou abrir uma nova janela no browser para fazer este teste. Vou colocar "localhost:8080/casadocodigo". Agora vamos voltar para esse cenário e vamos tentar comprar o livro "Amazon AWS", vou clicar aqui para comprar, e vou no livro impresso.

[15:42] Nesse momento como falamos, pode acontecer do balanceador estar redirecionando esta requisição para outra instância, o que acontecia se trocarmos para o Tomcat "8081"? Nós perdíamos este livro, pois essa informação estava na sessão do usuário. Mas se fizermos o teste agora, percebemos que o nosso livro foi mantido, porque a sessão do usuário está sendo mantida no "redis". Vamos confirmar se a sessão do usuário está no "redis"?

[16:11] Vou voltar para nossa pasta do "redis" e vou clicar no "redis-cli" para confirmar que a sessão do usuário foi armazenada no "redis". Vou clicar aqui no "redis-cli", e para podermos ver as chaves das sessões salvas aqui no redis basta colocar "Keys *". Temos a informação do nosso usuário. Vamos confirmar que está é a sessão do usuário?

[16:54] Voltamos aqui, vou clicar com o botão direito do mouse, inspecionar, vou vir em application e na opção dos cookies temos exatamente o valor da sessão do nosso usuário no "redis". Vamos comparar as informações, exatamente igual ao que temos aqui da informação da sessão do nosso usuário.

[17:26] Agora estamos armazenando essa informação no "redis", por esse motivo não importa se o balanceador na Amazon redirecionar nossa requisição pra primeira ou segunda instância, porque a análise da sessão do usuário está sendo feita no "redis". Agora nosso livro foi mantido no carrinho de compras independentemente de estar na primeira ou segunda instância do Tomcat.

[17:51] Agora só falta levar essa configuração para o nosso ambiente de produção da Amazon. Vamos fazer isso na próxima etapa.