02

Utilizando cache

Transcrição

[00:00] Na última aula, aprendemos a como usar os recursos de paginação e ordenação. Agora, vamos aprender a usar o recurso de cache, um recurso muito útil quando queremos melhorar a performance da aplicação. A ideia é que se eu utilizar o cache, consigo dizer para o Spring guardar o retorno de um método em cache. Na primeira vez que eu chamar aquele método, ele vai executar linha por linha do método, vai devolver o resultado, mas nas próximas chamadas ele já devolve direto o que está em memória. Com isso, fica muito mais rápido o retorno para o cliente que fez a chamada para a API.

[00:39] O primeiro passo é: precisamos adicionar esse módulo do Spring Boot, que é um modelo de cache no nosso projeto. Não temos ele adicionado como dependência do Maven. Vamos abrir o .xml, e posso pegar alguma das dependências, por exemplo, da JPA, e só trocar o artifactid, para Spring Boot starter cache. Salvei. Ele vai baixar as dependências do Spring Boot starter cache, colocar no nosso projeto.

[01:08] Outra dependência que deveríamos colocar é de qual a ferramenta, o provedor de cache que quero utilizar para aplicação. Com a aplicação rodando em produção, o ideal é utilizar algum provedor de cache. O Spring suporte alguns provedores, mas para fins de desenvolvimento, não precisamos usar nenhum provedor. Se não declararmos um provedor, o Spring usa um padrão, que é onde ele guarda o cache em memória usando um REST map. Mas esse provider não é recomendado para usar em produção.

[01:54] Coloquei só a dependência do cache. Ele vai usar o provedor padrão, usando um REST map. O próximo passo é: além de declarar dependência de cache no projeto, precisamos habilitar o uso de cache na aplicação, porque ele não vem habilitado por padrão. Nós vimos no outro exemplo da paginação que para habilitar coisas no projeto temos que ir à classe fórum application e colocar uma anotação em cima dela.

[02:24] Já temos o enable Spring data web support. Agora vamos utilizar outra aplicação, que é o @EnableCaching, para habilitar o uso de cache na aplicação. Vou salvar. Com isso, tenho o recurso de cache habilitado no projeto. Agora é só utilizar. Para utilizar cache, tenho que ir em alguma classe, controller, service ou repository, e em cima do método falar para o Spring que eu quero que ele guarde o retorno desse método em cache.

[02:58] No nosso exemplo, vou colocar na nossa classe tópicos Controller um método lista em cache. Em cima do método, temos que colocar a anotação @Cacheable, para falar para o Spring guardar o retorno desse método em cache. Só cuidado na hora de fazer o import, porque existe a mesma anotação no pacote da JPA. O que vamos utilizar no curso é do org.springframework.

[03:26] Essa anotação tem um atributo que precisamos preencher. Um chamado value, em que temos que passar uma string que vai ser o identificador único desse cache. Na nossa aplicação, posso ter vários métodos anotados com @Cacheable, e o Spring precisa saber como ele vai diferenciar um do outro. Ele faz isso utilizando o id único. Vou passar um nome, por exemplo, listaDeTopicos. Essa string vai funcionar como sendo um id desse cache.

[04:12] Agora vou até o Postman e vou fazer uma chamada para esse método lista. É só disparar uma requisição GET para o endpoint/tópicos. Já estou configurado, não preciso passar os parâmetros de paginação e ordenação. Porque são opcionais. Vou clicar no send. Ele disparou a requisição e trouxe os registros do nosso banco de dados.

[04:38] Se eu disparar novamente a requisição, ele traz também os resultados, mas a partir da segunda requisição ele não executou linha por linha do método lista. Ele devolveu o que ele guardou na primeira chamada no cache.

[04:51] Só que como eu sei se realmente ele está trazendo o resultado do cache? Como faço para saber se ele não está entrando, executando linha por linha do método todas as vezes? Eu poderia colocar um system out e ver se na primeira chamada ele imprimiu, passou no método e executou o system out e se nas próximas ele não chamou. Mas tem uma outra maneira melhor de fazer isso. Como esse método está fazendo uma consulta no banco de dados, podemos habilitar o log do hibernate para ele sempre imprimir toda vez que ele for fazer uma chamada no banco de dados. Nós não habilitamos isso no banco de dados, por isso não está imprimindo nada no console quando vai ao banco de dados.

[05:35] Para habilitar isso, precisamos ir ao nosso arquivo application properties, que é uma configuração que fazemos para o projeto. Nas propriedades da JPA, precisamos colocar mais uma propriedade, que é o Spring.jpa.properties.hibernate, e a propriedade é show_sql=true. Isso fala para o hibernate imprimir os comandos SQL que ele dispara toda vez que acessa o banco de dados. Mas por padrão ele imprime todos os selects em uma linha só. Se quisermos um SQL formatado, precisamos colocar mais uma propriedade, a format_sql, que quebra a linha e deixa o SQL formatado.

[06:31] Vou salvar. No Postman, vou disparar a primeira requisição. No Eclipse, ele montou o select, fez vários selects, porque ele tem que fazer os joins com os relacionamentos da entidade tópico. Vou limpar o console. Se eu disparar na segunda vez, é para disparar select, porque não é para entrar e executar o método. É para devolver o que ele guardou em memória.

[07:12] Percebe que a partir da segunda vez, ele não executa linha por linha do método, ele não vai no banco de dados, ele devolve o que está em memória, que é muito mais rápido do que fazer o acesso ao banco de dados.

[07:22] Com isso, conseguimos implementar o uso de cache. No meu método lista, a primeira vez que eu chamar esse método, ele vai pegar o resultado, guardar em memória, e nas próximas vezes ele devolve o que está em memória. Com isso, eu tenho um ganho de performance.

[07:37] Só um detalhe importante. O nosso método tem um parâmetro nome do curso. No Postman, não passei esse parâmetro, e fiz a chamada. Se eu passar um parâmetro, será que isso vai mudar alguma coisa? Porque ele guardou no cache sem parâmetro, sem o nome curso. Vamos fazer o teste. Ele trouxe o resultado, só trouxe o registro um e dois. O terceiro está no curso html. Só que se olharmos o console do Eclipse, vamos ver que agora ele fez um select.

[08:33] O Spring é esperto. Ele detecta que mudou o parâmetro, só que o cache que ele tinha era sem parâmetro. Então, dessa vez ele vai precisar fazer o select. Se eu disparar de novo essa requisição, ele não fez o select. Se eu tirar o parâmetro e disparar a requisição, ele também não fez o select. Ou seja, ele guarda as diferenças, não guarda só um único valor no cache. Ele guarda o valor sem parâmetro, guarda o valor com Spring Boot no parâmetro. Inclusive, se eu passar o parâmetro nome curso, mas agora vou passar html5, ele sabe que é o mesmo parâmetro nome curso, mas como passei outro valor, ele não tem isso guardado no cache. Então, ele vai no banco de dados.

[03:18] Para cada variação de parâmetro ele guarda o resultado no cache. Nas próximas vezes que você chamar, ele avalia. Está vindo parâmetro? Qual o valor? Já tenho esse resultado com esse parâmetro com esse valor em cache? Devolve o que está em memória. Eu não tenho nada com esse parâmetro ou com esse valor de parâmetro? Executa um método, vai no banco de dados. Ele é esperto, ele sabe quando tem que executar o método, quando não tem que executar.

[09:43] Com isso, conseguimos utilizar o recurso de cache para ter uma melhora de performance, já que evito uma ida ao banco de dados. Devolvo o que está em memória, que é muito mais rápido.

[09:53] Só que aqui tem outro problema, que é o seguinte: e se alguém cadastrar um novo tópico no projeto? Quando eu fizer essa consulta, ele vai devolver só os três registros que estão em cache, na teoria. Ele não vai pegar o quarto registro que foi inserido. Isso seria um problema. No próximo vídeo vamos aprender como posso ensinar para o Spring quando ele tem que limpar o cache, quando tem que invalidar o meu cache, porque um novo registro foi atualizado, cadastrado ou excluído.