07

Mãos na massa: Cache

Chegou a hora de você pôr em prática o que foi visto na aula. Para isso, execute os passos listados abaixo.

1) Na parte 2 desse curso sobre Spring MVC, você continuará utilizando o mesmo projeto criado na parte 1. Para sua conveniência, você pode fazer o download aqui (https://s3.amazonaws.com/caelum-online-public/springmvc-2-integracao-cache-seguranca-e-templates/springmvc2-projeto-inicial.zip) do ZIP do código fonte do projeto do completo do curso anterior.

Uma vez baixado basta extrair e importar o projeto como ***Maven Project*** no Eclipse. Para rodar o projeto, você também precisa ter configurado o **Apache Tomcat**.

2) Para usar o JSP da página do Casa do Código, é aconselhado usar o arquivo JSP disponibilizado como download nesse zip aqui (https://s3.amazonaws.com/caelum-online-public/springmvc-2-integracao-cache-seguranca-e-templates/springmvc2-arquivos-extras-aula01.zip).

Ao extrair o ZIP, você encontrará:

- O arquivo JSP limpo (**home-limpo.jsp**), mas ainda **sem *expressions language***.
- O arquivo JSP finalizado (**home-pronto.jsp**), com **todas as alterações da aula já aplicadas**.

Caso queira seguir o vídeo, é aconselhado usar o arquivo **home-limpo.jsp**, renomeá-lo para **home.jsp** e adicionar as tags e *expressions language*. O arquivo deve ficar dentro da pasta **src/main/webapp/WEB-INF/views/**. O arquivo **home-pronto.jsp** serve como referência.

3) Garanta que o `HomeController` vai enviar todas as informações que a *home* precisa para ser exibida corretamente. Altere o método `index`, do `HomeController`, para enviar uma lista de produtos para a *view*, usando o método `listar` da classe `ProdutoDAO`:

```
@Controller
public class HomeController {

    @Autowired
    private ProdutoDAO produtoDao;

    @RequestMapping("/")
    public ModelAndView index() {

        ModelAndView modelAndView = new ModelAndView("home");
        List<Produto> produtos = produtoDao.listar();
        modelAndView.addObject("produtos", produtos);

        return modelAndView;
    }

}
```

4) Altere também seu arquivo **home.jsp**, para que ele possa exibir os livros que estão atualmente cadastrados no sistema:

```
<c:forEach items="${produtos}" var="produto ">
    <li>
        <a href="${s:mvcUrl('PC#detalhe').arg(0, produto.id).build()}"
            class="block clearfix">
            <h2 class="product-title">${produto.titulo}</h2>
            <img width="143" height="202"
                src="https://cdn.shopify.com/s/files/1/0155/7645/products/java8-featured_large.
                    alt="Java 8 Prático" title="Java 8 Prático"/>
            <small class="buy-button">Compre</small>
        </a>
    </li>
</c:forEach>
```

Lembre-se de adicionar mais uma *taglib*, que será a do **Spring**, ao JSP:

```
<%@ taglib uri="http://www.springframework.org/tags" prefix="s"%>
```

5) No caso da **Casa do Código**, os produtos da *Home* mudam semanalmente. Sendo assim, não é necessário ficar o tempo todo buscando as informações no banco de dados. Uma forma de manter isso em memória é usando **cache**. Habilite o cache para a *Home*. Então, no método `index`, do `HomeController`, utilize a anotação `@Cachable`:

```
@RequestMapping("/")
@Cacheable(value="produtosHome")
public ModelAndView index() {
    ModelAndView modelAndView = new ModelAndView("home");
    List<Produto> produtos = produtoDao.listar();
    modelAndView.addObject("produtos",produtos);
    return modelAndView;
}
```

6) Adicione o gerente de cache que será utilizado nesse curso, o **Guava**, da Google. Cole o seguinte XML no seu arquivo **pom.xml**:

```
<dependency>
    <groupId>com.google.guava</groupId>
    <artifactId>guava</artifactId>
    <version>18.0</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context-support</artifactId>
    <version>4.1.0.RELEASE</version>
</dependency>
```

7) Além disso, configure qual mecanismo de cache deverá ser utilizado, através da classe `AppWebConfiguration`. Crie o método `cacheManager`, retorne o **Guava** e adicione a anotação `@EnableCaching` na classe:

```
@EnableWebMvc
@EnableCaching
@ComponentScan(basePackageClasses={HomeController.class, ProdutoDAO.class,
    FileSaver.class, CarrinhoCompras.class})
public class AppWebConfiguration {

    @Bean
    public CacheManager cacheManager() {
        CacheBuilder<Object, Object> builder =
            CacheBuilder.newBuilder().maximumSize(100)
                .expireAfterAccess(5, TimeUnit.MINUTES);
        GuavaCacheManager manager = new GuavaCacheManager();
        manager.setCacheBuilder(builder);

        return manager;
    }

    // restante do código omitido
}
```

8) Quando um novo produto é cadastrado, ele não aparece na listagem, pois ela está em cache. Então, quando um produto for cadastrado, o cache deve ser atualizado. No método `gravar`, do `ProdutoController`, utilize a anotação `@CacheEvict` para resolver esse problema:

```
@RequestMapping(method=RequestMethod.POST)
@CacheEvict(value="produtosHome", allEntries=true)
public ModelAndView gravar(MultipartFile sumario, @Valid Produto produto,
        BindingResult result, RedirectAttributes redirectAttributes){

    if(result.hasErrors()) {
        return form(produto);
    }

    String path = fileSaver.write("arquivos-sumario", sumario);
    produto.setSumarioPath(path);

    produtoDao.gravar(produto);

    redirectAttributes.addFlashAttribute("sucesso", "Produto cadastrado com sucesso!");

    return new ModelAndView("redirect:produtos");
}
```