03

Salvando upload

Transcrição

Apesar de funcionar, a intenção não é simplesmente imprimir o nome do arquivo no console. Mas sim enviar o arquivo e deixa-lo hospedado no servidor. Este código é um código de infra (abreviação de infraestrutura). Ele carregará os arquivos enviados e assim irá salvar os arquivos em algum diretório/pasta especifico.

Vamos criar uma nova classe para conter esse código. Vamos chama-la de `FileSaver` e deixa-la no pacote `br.com.casadocodigo.loja.infra` . Nós precisamos que essa classe seja reconhecida pelo **Spring** para que ele consiga fazer os `injects` corretamente. Está classe é importante e ela representa um componente em nosso sistema. Teremos então que usar a anotação `@Component` .

Nesta classe criaremos um método chamado `write` que fará a transferência do arquivo e retornará o caminho onde o arquivo foi salvo. Este método então precisara de duas informações, o local onde o arquivo será salvo e o arquivo em si. O local será recebido como `String` e o arquivo como um objeto `MultipartFile` . Os quais chamaremos de `baseFolder` e `file` respectivamente.

```
@Component
public class FileSaver {
    public String write(String baseFolder, MultipartFile file){

    }
}
```

Com o `baseFolder` e o `file` em mãos, conseguiremos facilmente montar uma `String` que indique o caminho do arquivo a ser salvo. Com esta `String` construída, criaremos um novo objeto do tipo `File` que irá representar o arquivo a ser gravado no servidor. Este último objeto será passado para o método `transferTo` que será o método responsável por transferir o arquivo para o servidor. O código parece ser mais fácil de entender.

```
@Component
public class FileSaver {
    public String write(String baseFolder, MultipartFile file) {
        try {
            String path = baseFolder + "/" + file.getOriginalFilename();
            file.transferTo(new File(path));
            return path;

        } catch (IllegalStateException | IOException e) {
            throw new RuntimeException(e);
        }
    }
}
```

Note que a `String path` monta o caminho do arquivo. O `file.transferTO()` faz a transferência do arquivo e o objeto `File` representa um o arquivo no servidor. O bloco `try/catch` foi adicionado por causa que operações `I/O` , ou seja, de entrada e saída, que podem gerar erros. Perceba também que estamos retornando a `String path` dentro do bloco `try` .

Apesar deste código parecer claro, não podemos definir com certeza o caminho final do arquivo, o caminho absoluto que ele vai ter ao ser enviado. Podemos mudar isto detectando o caminho atual que o usuário está em nosso sistema e fazer o upload do arquivo baseado neste caminho. Para isso precisamos dos dados da requisição, pois com ela sabemos onde o usuário está em nosso sistema.

Pensando nisso, criaremos um atributo do tipo `HttpServletRequest` na classe `FileSaver`, chamaremos este de `request` e o marcaremos com a anotação `@Autowired` para que o **Spring** faça o `inject` desse atributo. A partir deste objeto, conseguimos extrair o contexto atual em que o usuário se encontra e então conseguir o caminho absoluto desse diretório em nosso servidor.

Vamos começar criando este novo atributo.

```
@Component
public class FileSaver {
    @Autowired
    private HttpServletRequest request;
    [...]
}
```

E então, dentro do bloco `try/catch` usaremos o método `getServletContext` para extrair o contexto atual do usuário e logo em seguida, do retorno deste método, usaremos o `getRealPath` que irá nos retornar o caminho completo de onde está determinada pasta dentro do servidor. Passaremos para o `getRealPath` o nome da pasta base que estamos recebendo em nosso método para que este método encontre a pasta correta. O bloco `try/catch` então fica dessa forma:

```
public String write(String baseFolder, MultipartFile file) {
    try {
        request.getServletContext().getRealPath("/"+baseFolder);
        [...]
    }catch([...]){
        [...]
    }
}
```

O caminho do arquivo agora é diferente do que fizemos antes, ele não é mais uma simples junção do `baseFolder` com o nome do arquivo. Este caminho agora precisa ser concatenado com o caminho absoluto que acabamos de implementar através do request. Sendo assim, guardaremos o retorno do `request.getServletContext().getRealPath("/"+baseFolder);` em uma nova `String` que chamaremos de `realPath` e usaremos esta `String` para concatenar ao `path` do arquivo que geramos anteriormente. Observe o código:

```
@Component
public class FileSaver {
    @Autowired
    private HttpServletRequest request;

    public String write(String baseFolder, MultipartFile file) {
        try {
            String realPath = request.getServletContext().getRealPath("/"+baseFolder);
            String path = realPath + "/" + file.getOriginalFilename();
            file.transferTo(new File(path));
            return path;
```

```
            } catch (IllegalStateException | IOException e) {
                throw new RuntimeException(e);
            }
        }
    }
```

Quase nada mudou, apenas a `String path` deixou de concatenar o `basePath` e passou a concatenar o `realPath`. A classe `FileSaver` está pronta. Ela recebe um arquivo e o nome de uma pasta, transfere o arquivo enviado pelo formulário para a pasta e retorna o caminho onde o arquivo foi salvo.

Agora só precisamos alterar o `ProdutosController` para usar a classe `FileSaver`. Como queremos que o **Spring** fique responsável por instanciar estes objetos. Usaremos a mesma estratégia do `request` na classe `FileSaver`, mas agora em nosso `ProdutosController` com o `FileSaver`. Criaremos um atribudo da classe e assinaremos este atributo com `@Autowired`.

```
@Controller
@RequestMapping("/produtos")
public class ProdutosController {

    @Autowired
    private FileSaver fileSaver;

    [...]
}
```

O próximo passo é usar este objeto no método `gravar`. Usaremos o método `write` deste objeto e passaremos o objeto `MultipartFile` que recebemos no método `gravar` como arquivo a ser salvo e como nome da pasta passaremos a `String arquivos-sumario`. Vamos pôr este código após a verificação de erros, desta forma o arquivo só será efetivamente gravado caso não haja erros de validação no formulário. O código do método gravar fica assim:

```
@RequestMapping(method=RequestMethod.POST)
public ModelAndView gravar(MultipartFile sumario, @Valid Produto produto, BindingResult result,

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

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

    produtoDao.gravar(produto);
    redirectAttributes.addFlashAttribute("message","Produto cadastrado com sucesso");
    return new ModelAndView("redirect:produtos");
}
```

Lembre-se que a classe que salva o arquivo no servidor retorna o caminho do arquivo. Este caminho deve ser salvo no banco de dados, por isso estamos usando a `String path` e passando esta `String` para o método `setSumarioPath` do `produto`.

Podemos reiniciar o servidor e fazer alguns testes agora. Mas quando reiniciamos, recebemos um erro:

```
No qualifying bean of type [br.com.casadocodigo.loja.infra.FileSaver] found for dependency: expected at least 1 bean which qualifies as autowire
:ory.raiseNoSuchBeanDefinitionException(DefaultListableBeanFactory.java:1261)
:ory.doResolveDependency(DefaultListableBeanFactory.java:1009)
:ory.resolveDependency(DefaultListableBeanFactory.java:904)
eanPostProcessor$AutowiredFieldElement.inject(AutowiredAnnotationBeanPostProcessor.java:514)
```

O Spring indica que não foi encontrado nenhum `bean` qualificado. Isso acontece porque ele não consegue encontrar nossa classe `FileSaver` pois esta em um pacote não gerenciado pelo Spring. Solucionamos isso atualizando o `componentScan` na classe de configuração da aplicação, a classe `AppWebConfiguration` :

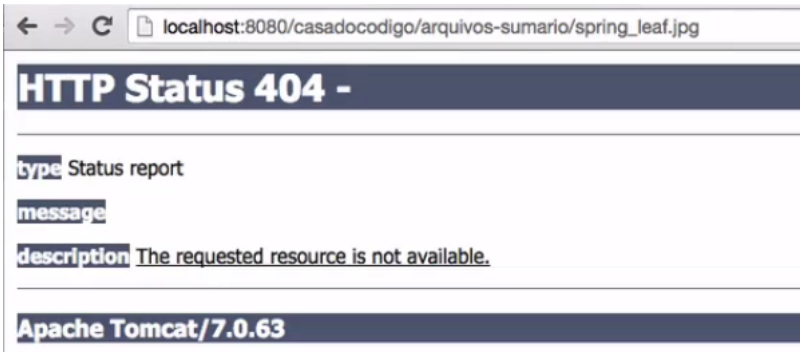
```
@EnableWebMvc
@ComponentScan(basePackageClasses={HomeController.class, ProdutoDAO.class, FileSaver.class})
public class AppWebConfiguration {
    [...]
}
```

Com esta configuração feita, o servidor reiniciará normalmente. Então podemos testar cadastrar um livro com sumário. Por questões de ser um teste simples, escolha um arquivo qualquer, pode ser até mesmo uma pequena imagem. Preencha e submeta o formulário para ver o resultado.

HTTP Status 500 - Request processing failed; nested exception is java.lang.RuntimeException: java.io.IOException: java.io.FileNotFoundException: /Users/Alura2/Documents/paulo/apache-tomcat-7.0.63/wtpwebapps/casadocodigo/arquivos-sumario/spring_leaf.jpg (No such file or directory)

O erro acontece porque a pasta `arquivos-sumario` não existe. Vamos criar então esta pasta dentro de `src/main/webapp/` . Com a pasta criada, refaça o teste e veja tudo funcionar perfeitamente.

Note que apesar de salvarmos o caminho completo para o arquivo, não precisamos realmente do caminho completo. Perceba que se acessarmos `localhost:8080/casadocodigo/arquivos-sumario/NOME_DO_ARQUIVO` já poderiamos acessar o arquivo diretamente. Mas se tentarmos teremos um erro 404.



O motivo deste erro descobriremos mais a frente neste curso, mas por hora, faremos um pequeno ajuste no caminho retornado pelo médoto `write` na classe `FileSaver` que retorna o caminho absoluto do nosso arquivo para retornar o caminho relativo ao nosso sistema.

O caminho relativo é composto pelo `baseFolder` + nomeDoArquivo. Nosso método que estava assim:

```
public String write(String baseFolder, MultipartFile file) {
    try {
        String realPath = request.getServletContext().getRealPath("/"+baseFolder);
        String path = realPath + "/" + file.getOriginalFilename();
        file.transferTo(new File(path));
        return path;

    } catch (IllegalStateException | IOException e) {
```

```
            throw new RuntimeException(e);
        }
    }
```

Agora ficará assim:

```
public String write(String baseFolder, MultipartFile file) {
    try {
        String realPath = request.getServletContext().getRealPath("/"+baseFolder);
        String path = realPath + "/" + file.getOriginalFilename();
        file.transferTo(new File(path));
        return baseFolder + "/" + file.getOriginalFilename();

    } catch (IllegalStateException | IOException e) {
        throw new RuntimeException(e);
    }
}
```

Está mudança aparentemente não afetou em nada nosso sistema, mas agora em vez de guardamos o caminho completo até o arquivo, armazenamos apenas uma parte. Isso fará com que fique mais simples a exibição das imagens posteriormente.

Recapitulando

Fizemos uma série de adições em nosso sistema nesta aula. Adicionamos um `input` de arquivos para o envio dos sumários dos livros que serão cadastrados. Agora, os produtos guardam o caminho dos sumários. O *upload* dos arquivos também funciona graças às configurações de `Resolver` e de `Multipart` que fizemos e por último - mas não menos importante - fizemos a classe `FileSaver` que efetivamente realiza a transferência dos arquivos para o servidor.

Em seguida, faremos alguns exercícios para fixar o que aprendemos até aqui. Não se esqueça de que qualquer dúvida pode ser postada no forúm.