02

Configurando Multipart

Transcrição

Nossa aplicação já tem uma série de funcionalidades. Ela lista, cadastra e valida os produtos e ainda nos mostra qualquer problema que ocorre durante o processo de cadastramento dos livros.

Neste capítulo, faremos com que a aplicação permita o cadastro do sumário dos livros. Os sumários na maioria das vezes são feitos em PDF. Sendo assim, vamos fazer com que o nosso sistema hospede os arquivos no servidor.

Começaremos a fazer esta mudança a partir do formulário de cadastro de produtos. O `form.jsp` agora terá um novo campo com o label `sumario` e o `input` - que desta vez será do tipo `file` . Este tipo é usado exatamente para os casos nos quais queremos enviar um arquivo para o servidor. Usando este tipo de campo, o navegador já saberá que é preciso abrir uma janela de seleção para selecionarmos o arquivo. O `name` deste campo `file` também será "sumário". Com isto teremos o seguinte código:

```
<div>
    <label>Sumário</label>
    <input name="sumario" type="file" />
</div>
```

O `form.jsp` com esta adição ficará assim:

```
<form:form action="${ s:mvcUrl('PC#gravar').build() }" method="post" commandName="produto">
    <div>
        <label>Título</label>
        <form:input path="titulo" />
        <form:errors path="titulo" />
    </div>
    <div>
        <label>Descrição</label>
        <form:textarea rows="10" cols="20" path="descricao" />
        <form:errors path="descricao" />
    </div>
    <div>
        <label>Páginas</label>
        <form:input path="paginas" />
        <form:errors path="paginas" />
    </div>
    <div>
        <label>Data de Lançamento</label>
        <form:input path="dataLancamento" />
        <form:errors path="dataLancamento" />
    </div>
    <c:forEach items="${tipos}" var="tipoPreco" varStatus="status">
        <div>
            <label>${tipoPreco}</label>
            <form:input path="precos[${status.index}].valor" />
            <form:hidden path="precos[${status.index}].tipo" value="${tipoPreco}" />
        </div>
    </c:forEach>
```

```
    <div>
        <label>Sumário</label>
        <input name="sumario" type="file" />
    </div>
    <button type="submit">Cadastrar</button>
</form:form>
```

Isto é tudo que precisamos fazer no `form.jsp`. Agora precisamos atualizar a classe `Produto`. Vamos adicionar também um novo atributo chamado `sumarioPath` e os seus **Getters and Setters**. Este será do tipo `String`.

A classe `Produto` ficará assim:

```
@Entity
public class Produto {

    [...]
    private String sumarioPath;
    [...]

    public String getSumarioPath() {
        return sumarioPath;
    }
    public void setSumarioPath(String sumarioPath) {
        this.sumarioPath = sumarioPath;
    }

}
```

Existem várias estratégias para guardar arquivos nas aplicações. Uma delas seria guardar o arquivo no banco de dados, mas esta seria muito trabalhosa e precisaríamos converter o arquivo para um formato aceito pelo banco, geralmente `bytes`. Outra opção seria guardar nas pastas do sistema de arquivos do servidor. Optaremos por esta segunda opção, por isso, o atributo `sumarioPath` é do tipo `String`. Nele será guardado apenas o caminho (`path`) do arquivo.

Nossa classe `Produto` já está pronta para armazenar o caminho do arquivo. Podemos então modificar o `ProdutosController` para receber este arquivo e realizar as operações necessárias. O **Spring** enviará nosso arquivo para o `ProdutosController` como um objeto do tipo `MultipartFile`, que chamaremos de `sumario`. Vamos imprimir o nome do arquivo no console do Eclipse usando o método `getOriginalFilename()`. Este será o teste básico para sabermos se o arquivo está sendo enviado corretamente.

Observação: Lembre-se que o formulário envia os dados para o método `gravar`. Estas modificação são realizadas justamente neste método.

Então, receberemos em nosso `controller` este novo objeto da seguinte forma:

```
public ModelAndView gravar(MultipartFile sumario, @Valid Produto produto, BindingResult result,
```

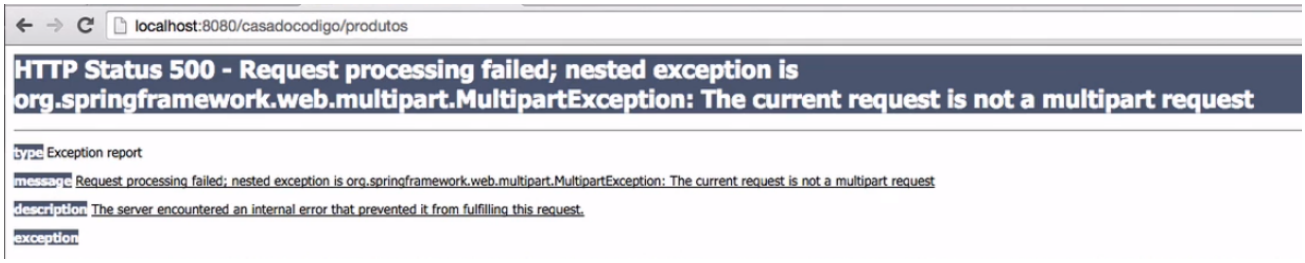
Imprimindo o nome do arquivo, teremos o seguinte código no nosso método `gravar`.

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

    System.out.println(sumario.getOriginalFilename());

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

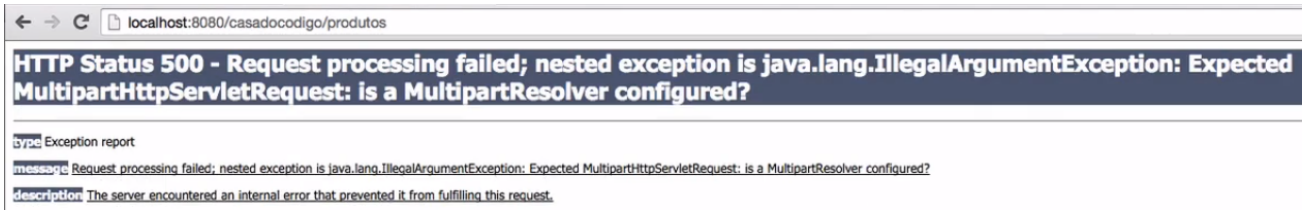
Teste agora, cadastrar um produto preenchendo todos os campos, inclusive escolhendo um arquivo qualquer para o sumário. Teremos um erro!



A mensagem do erro diz que a requisição atual não é `multipart`. Requisições deste tipo podem fazer envios de arquivos, sendo estes de qualquer tipo. Corrigir o erro é simples, basta usar o atributo `enctype` com o valor `multipart/form-data` na tag `form` do nosso `form.jsp`.

```
<form:form action="${ s:mvcUrl('PC#gravar').build() }" method="post" commandName="produto" encty
    [...]
</form:form>
```

Atualize a página de cadastro de produtos. Novamente, tente cadastrar um produto preenchendo todos os campos. Teremos um novo erro.



A mensagem de erro nos diz que era esperado um `MultipartHttpServletRequest` e nos pergunta `is a MultipartResolver configured?`. Esta mensagem parece bem ser clara. Ela nos pergunta se temos um `MultipartResolver` configurado. Não configuramos nada disso em nossa aplicação. Vamos fazer essa configuração, então.

Nossas configurações ficam na classe `AppWebConfiguration`. Vamos até esta classe e adicionaremos a nova configuração. Vamos criar um método chamado `multipartResolver` que retorna um objeto do tipo `MultipartResolver`. Este objeto será instanciado da classe `StandardServletMultipartResolver` e retornado. Sendo assim, teremos o seguinte código em nossa classe `AppWebConfiguration`:

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

    [...]
    @Bean
    public MultipartResolver multipartResolver(){
        return new StandardServletMultipartResolver();
    }

}
```

Agora que temos um `multipartResolver` configurado em nossa aplicação, podemos tentar cadastrar um produto novamente. Lembre-se de reiniciar o servidor para as alterações funcionarem.

Observação: `MultipartResolver` se refere a um resolvedor de dados multimídia. Quando temos texto e arquivos por exemplo. Os arquivos podem ser: imagem, PDF e outros. Este objeto é que identifica cada um dos recursos enviados e nos fornece uma forma mais simples de manipulá-los.

Quando tentarmos cadastrar um produto agora, iremos receber um novo erro.

Este erro acontece porque o nosso método `gravar`, em `ProdutosController`, tenta imprimir o nome do arquivo enviado. Aparentemente não resolvemos nosso problema de `multipartResolver` por completo. Mesmo tendo feito a configuração do `multipartResolver`, o **Spring** ainda não consegue fazer a conversão dos dados. Teremos que configurar mais algumas coisas.

As novas configurações devem ser feitas na classe `ServletSpringMVC`, que é a classe de inicialização da nossa aplicação. Nesta classe, iremos sobrescrever um método chamado `customizeRegistration` que recebe um objeto do tipo `Dynamic` que chamaremos de `registration`. Neste objeto, usaremos o método `setMultipartConfig` que requer um objeto do tipo `MultipartConfigElement`. O `MultipartConfigElement` espera receber uma `String` que configure o arquivo. Não usaremos nenhuma configuração para o arquivo, queremos receber este do jeito que vier. Passamos então uma `String` vazia.

O código destas mudanças ficará assim:

```
public class ServletSpringMVC extends AbstractAnnotationConfigDispatcherServletInitializer{
    [...]
    @Override
    protected void customizeRegistration(Dynamic registration) {
        registration.setMultipartConfig(new MultipartConfigElement(""));
    }

}
```

Reiniciando o servidor e testando novamente, veremos que o produto foi cadastrado com sucesso e no console do **Eclipse** o nome do arquivo deve estar impresso. Após o teste, verificaremos em nosso sistema o arquivo `spring_leaf.jpg` , que estávamos enviando de fato. No entanto, não queremos simplesmente ter o seu nome, e sim enviá-lo ao servidor. Assim, quando recebemos `sumario` seria interessante podermos gravá-lo no servidor, transferindo-o, pegando o nome completo de onde ele foi colocado.