
ATENÇÃO

Ao final do vídeo 1 o arquivo .properties estava no singular (message.properties), porém ao início do segundo o mesmo estava no plural (messages.properties). Para evitar erros, estamos mantendo o padrão como messages.properties a partir desta atividade.

Terminamos a aula anterior com as validações já funcionando, mesmo que não exibindo as mensagens. Faremos isto agora de uma forma muito simples. Primeiro vamos abrir nosso arquivo `form.jsp` e adicionar uma nova `taglib`. Veja como está nosso `form.jsp` :

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEnc
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Livros de Java, Android, iPhone, Ruby, PHP e muito mais - Casa d
</head>
<body>
    <form action="/casadocodigo/produtos" method="post">
        <div>
            <label>Título</label>
            <input type="text" name="titulo" />
        </div>
        <div>
            <label>Descrição</descricao>
            <textarea rows="10" cols="20" name="descricao"></textarea>
        </div>
        <div>
            <label>Páginas</label>
            <input type="text" name="paginas" />
        </div>
        <c:forEach items="${tipos}" var="tipoPreco" varStatus="status">
            <div>
                <label>${tipoPreco}</label> <input type="text"
                    name="precos[${status.index}].valor" /> <input type
                    name="precos[${status.index}].tipo" value="${tipoPr
            </div>
        </c:forEach>
        <button type="submit">Cadastrar</button>
    </form>
</body>
</html>
```

A `taglib` que adicionaremos será uma do próprio **Spring**. A `uri` da `taglib` será: `http://www.springframework.org/tags/form` e o `prefix` será `form`. Dessa forma teremos nosso `form.jps` da seguinte forma:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEnc
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

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

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Livros de Java, Android, iPhone, Ruby, PHP e muito mais - Casa d
```

```
</head>
<body>
    <form action="/casadocodigo/produtos" method="post">
        [...]
    </form>
</body>
</html>
```

Com a `taglib` adicionada em nossa página JSP, podemos então usar uma **tag** dessa biblioteca para exibir as mensagens de erro. A tag usada para isso é a tag `form:error` e ela tem um atributo chamado `path` que indica de qual atributo queremos obter a mensagem de erro. Sendo assim, podemos escrever algo do tipo: `<form:errors path="produto.titulo" />`. Simples! Não? Vamos fazer o mesmo para os outros campos, colocando a *tag* acima do campo a que o erro se refere. Vejamos como o código do formulário ficará:

```
<form action="/casadocodigo/produtos" method="post">
    <div>
        <label>Título</label>
        <form:errors path="produto.titulo" />
        <input type="text" name="titulo" />
    </div>
    <div>
        <label>Descrição</label>
        <form:errors path="produto.descricao" />
        <textarea rows="10" cols="20" name="descricao"></textarea>
    </div>
    <div>
        <label>Páginas</label>
        <form:errors path="produto.paginas" />
        <input type="text" name="paginas" />
    </div>
    <c:forEach items="${tipos}" var="tipoPreco" varStatus="status">
        <div>
            <label>${tipoPreco}</label> <input type="text"
                name="precos[${status.index}].valor" /> <input type="hi
                name="precos[${status.index}].tipo" value="${tipoPreco}
        </div>
    </c:forEach>
    <button type="submit">Cadastrar</button>
</form>
```

Agora nossas mensagens devem aparecer no formulário caso cadastremos um produto que não seja válido, ou seja, sem título, descrição ou sem páginas. Vamos testá-lo então. Submeta o formulário sem preencher os campos corretamente!

```
org.springframework.context.NoSuchMessageException: No message found under code 'field.required.produto.titulo' for locale 'en_US'.
        org.springframework.context.support.DelegatingMessageSource.getMessage(DelegatingMessageSource.java:84)
        org.springframework.context.support.AbstractApplicationContext.getMessage(AbstractApplicationContext.java:1127)
        org.springframework.web.servlet.support.RequestContext.getMessage(RequestContext.java:707)
        org.springframework.web.servlet.support.BindStatus.initErrorMessages(BindStatus.java:181)
        org.springframework.web.servlet.support.BindStatus.getErrorMessages(BindStatus.java:277)
        org.springframework.web.servlet.tags.form.ErrorsTag.exposeAttributes(ErrorsTag.java:174)
        org.springframework.web.servlet.tags.form.AbstractHtmlElementBodyTag.writeTagContent(AbstractHtmlElementBodyTag.java:49)
        org.springframework.web.servlet.tags.form.AbstractFormTag.doStartTagInternal(AbstractFormTag.java:84)
        org.springframework.web.servlet.tags.RequestContextAwareTag.doStartTag(RequestContextAwareTag.java:80)
        org.apache.jsp.WEB_002dINF.views.produtos.form_jsp._jspService(form_jsp.java:92)
```

Deu erro? Como assim? Veja a mensagem de erro: **No message found under code field.required.produto.titulo**. O **Spring** não está encontrando nossas mensagens de erro. Em lugar nenhum definimos isso. Se você já conhece, existem os arquivos `properties` que são comuns de serem usados nesses casos. Vamos criar então um `properties` para deixarmos as mensagens para o formulário neste arquivo.

Na pasta `WEB-INF` vamos criar um arquivo de texto chamado `messages.properties` e associar as chaves dos erros aos valores, ou seja, associar as chaves dos erros às mensagens. Dentro desse arquivo, podemos ter algo do tipo `field.required.produto.titulo = O Campo título é obrigatório`.

Por hora, para testarmos, vamos deixar só a mensagem referente ao título. Depois adicionaremos o restante dos campos. Agora, precisamos configurar o **Spring** para que ele encontre esse nosso arquivos de mensagens. Já temos uma classe de configuração: a `AppWebConfiguration`.

Na classe `AppWebConfiguration` criaremos um novo método que carregará nossos arquivos de mensagens. Este método se chama `messageSource` e retorna um objeto do tipo `MessageSource`. Dentro deste método criaremos um objeto do tipo `ReloadableResourceBundleMessageSource` que chamaremos de `messageSource`. Neste objeto, iremos definir três propriedades: `setBaseName` com o valor `/WEB-INF/message` que terá o nome base dos nossos `resources`. O `setDefaultEncoding` com o valor `UTF-8` para evitar o problema de caracteres estranhos que já vimos outras vezes e o `setCacheSeconds` para que o **Spring** recarregue o arquivo de tempos em tempos com o valor `1`.

Esta última propriedade é muito útil em desenvolvimento pois poderemos ficar sempre modificando as mensagens sem se preocupar em ficar refazendo ***reload*** do arquivo manualmente. Nosso método fica da seguinte forma: **Observação:** Lembre-se de anotar esse método com a anotação `@Bean` para que o **Spring** possa reconhecer essa configuração.

```
@Bean
public MessageSource messageSource(){
    ReloadableResourceBundleMessageSource messageSource = new Reloadabl
    messageSource.setBasename("/WEB-INF/messages");
    messageSource.setDefaultEncoding("UTF-8");
    messageSource.setCacheSeconds(1);
    return messageSource;
}
```

Agora com a configuração correta, vamos tentar mais uma vez? Vamos enviar o formulário mais uma vez sem preencher nenhum dos campos. Deve aparecer alguma mensagem agora, correto?

```
org.springframework.context.NoSuchMessageException: No message found under code 'field.required' for locale 'en_US'.
        org.springframework.context.support.AbstractMessageSource.getMessage(AbstractMessageSource.java:186)
        org.springframework.context.support.AbstractApplicationContext.getMessage(AbstractApplicationContext.java:1127)
        org.springframework.web.servlet.support.RequestContext.getMessage(RequestContext.java:707)
        org.springframework.web.servlet.support.BindStatus.initErrorMessages(BindStatus.java:181)
        org.springframework.web.servlet.support.BindStatus.getErrorMessages(BindStatus.java:277)
        org.springframework.web.servlet.tags.form.ErrorsTag.exposeAttributes(ErrorsTag.java:174)
        org.springframework.web.servlet.tags.form.AbstractHtmlElementBodyTag.writeTagContent(AbstractHtmlElementBodyTag.java:49)
        org.springframework.web.servlet.tags.form.AbstractFormTag.doStartTagInternal(AbstractFormTag.java:84)
```

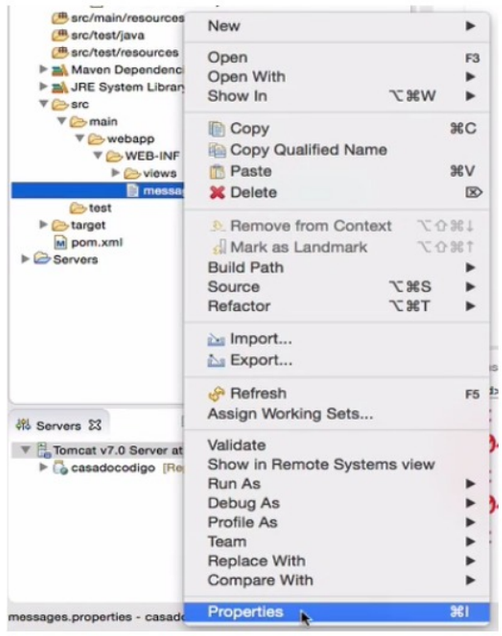
O `erro 500` continua mesmo tendo configurado tudo. Embora o número do erro seja o mesmo, o erro agora é outro. O **Spring** não está encontrando a mensagem referida com a chave `field.required`. Vamos adicionar então a chave em nosso `messages.properties` e aproveitar o momento para pôr as outras mensagens - uma para a descrição, que não pode ser vazia, e a outra para a quantidade de páginas, que deve ser superior a zero. Nosso arquivo `messages.properties` deve ficar dessa forma:

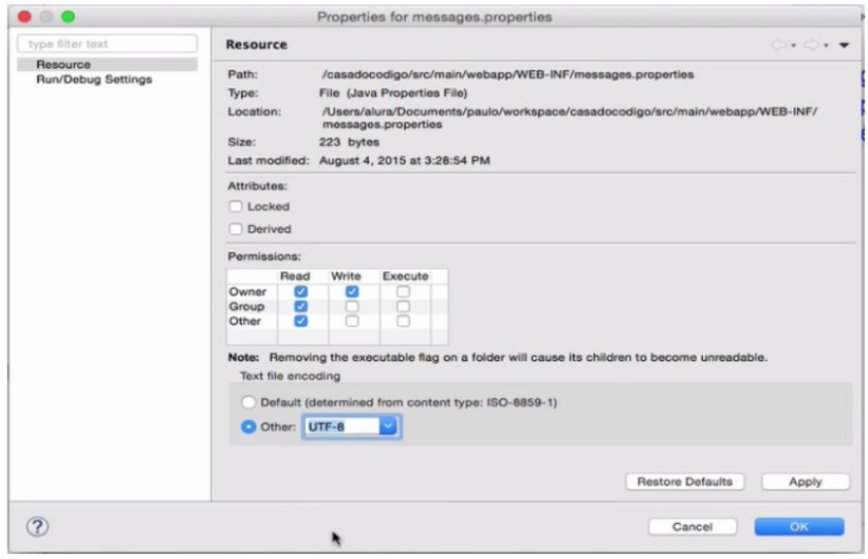
```
field.required = Campo é obrigatório
field.required.produto.titulo = O Campo título é obrigatório
field.required.produto.paginas = Informe o número de páginas
field.required.produto.descricao = O Campo descrição é obrigatório
```

Agora temos todas as mensagens prontas. A `field.required` será a mensagem mais genérica do nosso sistema, as outras serão mais específicas. Vamos tentar de novo então com estas novas modificações.**Observação:** Qualquer modificação nos textos do arquivo `messages.properties` pode ser feita sem precisar reiniciar o servidor. Nossa configuração de *reload* do arquivo já recarrega o arquivo automáticamente.

Nossas mensagens aparecem! Mas que estranho, mesmo configurando os caracteres para ter a codificação `UTF-8` eles aparecem estranhos na página e note uma mensagem de erro aparece no campo de páginas. Vamos resolver primeiro o problema dos caracteres.

O problema dos caracteres acontece por que o **Eclipse** por si só, codifica os arquivos em uma codificação especifica que por padrão não é a `UTF-8`. Devemos mudar isso então para o nosso arquivo `messages.properties`. Siga o seguinte caminho: `clique direito sobre o arquivo: messages.properties > Propriedades`. Na seção `Text file encoding` selecione `UTF-8`. Clique em `aplicar`. Confirma a mensagem que aparece e depois clica em `Ok`.





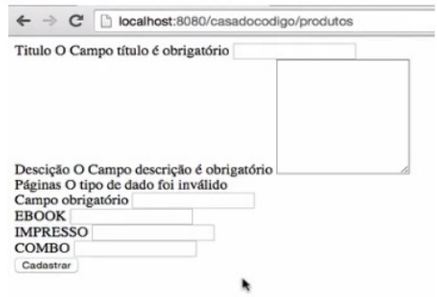
Os caracteres estranhos, agora, aparecem em nosso arquivo. Devemos trocá-los pelo texto que havia antes e salvar o arquivo novamente.

```
field.required = Campo obrigat►rio
field.required.produto.titulo = O Campo t►tulo ► obrigat►rio
field.required.produto.numeros = Informe o n►mero de p►ginas
field.required.produto.descricao = O Campo descri►►o ► obrigat►rio
```

Nosso segundo passo será resolver a mensagem de erro. Ela apareceu devido ao fato do **Spring** ter recebido um valor que não conseguiu converter para inserir no atributo páginas do objeto `produto`. Vamos adicionar uma mensagem dizendo que o tipo do dado fornecido no campo é inválido. A chave usada para este tipo de mensagem é a `typeMismatch`. Nosso arquivo `messages.properties` final ficará dessa forma:

```
field.required = Campo é obrigatório
field.required.produto.titulo = O Campo título é obrigatório
field.required.produto.paginas = Informe o número de páginas
field.required.produto.descricao = O Campo descrição é obrigatório
typeMismatch = O tipo de dado foi inválido
```

Precisamos reiniciar o servidor para que essa alteração se valide porque mudamos uma propriedade do arquivo e não seu conteúdo. Vamos reiniciar o servidor e tentar



novamente então.

Nossas mensagens agora aparecem sem nenhum problema, mas elas se misturam um pouco com o nome dos campos, para melhorar um pouco essa visualização, vamos por a `tag` `form:errors` de cada campo, logo após o campo a que a tag de refere. Nosso `form.jsp` dentro de `views/produtos` esta assim atualmente:

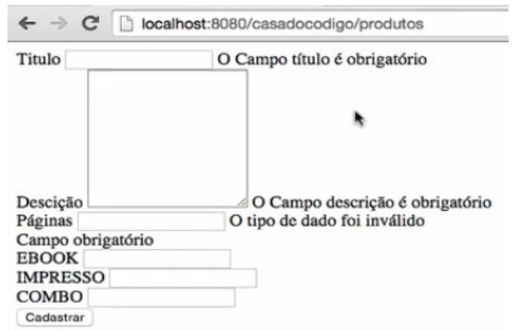
```
<form action="/casadocodigo/produtos" method="post">
    <div>
        <label>Título</label>
        <form:errors path="produto.titulo" />
        <input type="text" name="titulo" />
    </div>
    <div>
        <label>Descrição</label>
        <form:errors path="produto.descricao" />
        <textarea rows="10" cols="20" name="descricao"></textarea>
    </div>
    <div>
        <label>Páginas</label>
        <form:errors path="produto.paginas" />
        <input type="text" name="paginas" />
    </div>
    <c:forEach items="${tipos}" var="tipoPreco" varStatus="status">
        <div>
            <label>${tipoPreco}</label> <input type="text"
                name="precos[${status.index}].valor" /> <input type="hi
                name="precos[${status.index}].tipo" value="${tipoPreco}
        </div>
    </c:forEach>
    <button type="submit">Cadastrar</button>
</form>
```

Com a modificação que foi proposta antes, o mesmo formulário deve ficar desta forma.

```
<form action="/casadocodigo/produtos" method="post">
    <div>
        <label>Título</label>
        <input type="text" name="titulo" />
        <form:errors path="produto.titulo" />
    </div>
    <div>
        <label>Descrição</label>
```

```
        <textarea rows="10" cols="20" name="descricao"></textarea>
        <form:errors path="produto.descricao" />
    </div>
    <div>
        <label>Páginas</label>
        <input type="text" name="paginas" />
        <form:errors path="produto.paginas" />
    </div>
    <c:forEach items="${tipos}" var="tipoPreco" varStatus="status">
        <div>
            <label>${tipoPreco}</label> <input type="text"
                name="precos[${status.index}].valor" /> <input type="hi
                name="precos[${status.index}].tipo" value="${tipoPreco}
        </div>
    </c:forEach>
    <button type="submit">Cadastrar</button>
</form>
```

Desta forma, as mensagens aparecerão um pouco separadas do nome do campo e fica mais fácil de identificar de qual campo é a mensagem que aparece. Faça o teste, o resultado deve ser o mesmo que este:



Tudo funciona perfeitamente, mas podemos melhorar um pouco nosso código do formulário. Note que em todos os campos, onde queremos que apareça a mensagem de erro, temos que usar a tag `form:errors` usando o valor `produto.ALGUMACOISA` no atributo `path`. Nosso formulário só trata de um produto especifico, não precisamos ficar repetindo a informação em todos os campos. Se pudermos fazer: `<form:errors path="titulo" />` seria mais simples.

Podemos fazer isso usando um atributo da tag `form:form` da biblioteca do **Spring**. Ela tem um atributo chamado `commandName`, no qual podemos fazer uma referencia a qual entidade aquele formulário se refere. nesse caso, seria algo como: `commandName="produto"`. Dessa forma, nas tags de erros (`form:errors`) não precisaríamos colocar o prefixo `produto.`. Colocaríamos então só o nome do atributo.

Vamos então melhorar isso. Mudaremos a tag `form` do nosso `form.jsp` para a `form:form` do **Spring**, que tem os mesmos atributos da tag do `HTML` e ainda mais esse extra. O próximo passo é eliminar o prefixo `produto` usado nas tags `form:errors` para usar somente o nome do atributo, como por exemplo: título e descrição. Lembre-se que o fechamento da tag `form` também deve ser modificado para `form:form`. Nosso código então ficará assim:

```
<form:form action="/casadocodigo/produtos" method="post" commandName="p
    <div>
        <label>Título</label>
        <input type="text" name="titulo" />
        <form:errors path="titulo" />
    </div>
    <div>
        <label>Descrição</label>
        <textarea rows="10" cols="20" name="descricao"></textarea>
        <form:errors path="descricao" />
```

```
    </div>
    <div>
        <label>Páginas</label>
        <input type="text" name="paginas" />
        <form:errors path="paginas" />
    </div>
    <c:forEach items="${tipos}" var="tipoPreco" varStatus="status">
        <div>
            <label>${tipoPreco}</label> <input type="text"
                name="precos[${status.index}].valor" /> <input type="hi
                name="precos[${status.index}].tipo" value="${tipoPreco}
        </div>
    </c:forEach>
    <button type="submit">Cadastrar</button>
</form:form>
```

Se testar o formulário novamente, verá que não houve nenhuma mudança de comportamento, mas nosso código com certeza ficou mais claro e mais simples. Mas podemos melhorá-lo mais um ponto em nosso formulário. A `action` do nosso formulário é estática, caso precisemos mudar futuramente, teremos que lembrar em todos os lugares que usamos essa `action`. Conforme o número de controllers também cresce e ficará ainda mais complicado.

Vamos fazer com que o **Spring** gere automaticamente a `action` do nosso formulário. Caso a rota no controller mude, a `action` também muda automaticamente. Para isto precisaremos de uma outra `taglib` do **Spring**. Iremos adicionar a seguinte `taglib` na página `form.jsp`, logo após as outras `taglibs`.

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

Com essa nova biblioteca, podemos fazer uso da tag `mvcUrl` que gera uma `URL` de acordo com um determinado controller. Não precisamos passar o nome do controller por completo. Se passarmos as iniciais `PC` para se referir a `ProdutosController`, o **Spring** já conseguirá fazer a relação entre os dois.

Precisamos passar uma segunda informação para a tag: o método para qual os dados serão enviados. Neste caso, o método será o `gravar`.

Faremos isto da seguinte forma: `mvcUrl('PC#gravar')`. Com isso o **Spring** já consegue montar a rota corretamente, mas para que ele efetivamente faça isto, devemos usar o método `build()`. Sendo assim, teremos na `action` do nosso formulário o seguinte código:

```
action= "${s:mvcUrl('PC#gravar').build()}"
```

Lembre-se que o `s:` é usado por causa do `prefix` presente na declaração da `taglib`. Devemos ainda pôr uma barra na rota de `produtos` definida em nosso `ProdutosController`, para que a `URL` possa ser contruída de forma correta, separando os contextos. Assim, o `@RequestMapping` do nosso `ProdutosController` muda de `@RequestMapping("produtos")` para `@RequestMapping("/produtos")`.

Agora o nosso formulário esta pronto!

Teste cadastrar um produto de forma inválida e depois, de forma válida. Sempre precisamos verificar se alguma das modificações feitas não resultou em que outra parte da aplicação parasse de funcionar. Faça alguns testes, verifique se está tudo certo.

Recapitulando

Vimos bastante coisa até aqui, não é mesmo? Construímos nossa lógica de validação usando a classe `ProdutoValidation` que implementa a `interface validator`. Vimos como relacionar a validação em nosso controller com a classe de validação através do `InitBinder` e usando o `WebDataBinder`.

Fizemos também o nosso formulário ficar mais dinâmico, simples e claro, com o uso das tags do **Spring**. Vimos como podemos fazer o uso da anotação `@Valid` e também observamos que o `BindingResult` precisa ser usado logo após a declaração do `@Valid`.

Agora, vamos fazer alguns exercícios.

TIRAR DÚVIDA PRÓXIMA ATIVIDADE