

FIAP GRADUAÇÃO

DIGITAL BUSINESS ENABLEMENT

Prof. Me. Thiago T. I. Yamamoto

#05 – SPRING MVC – VALIDAÇÃO E CONVERSÃO



thiagoyama



thiagoyama@gmail.com

#05 VALIDAÇÕES E CONVERSÕES

- Validações
- Bean Validation e Hibernate Validation
- Validações no Controller
- Exibindo as mensagens de erro
- Conversão de Dados
- Conversão de Datas
- Mensagens de erro customizadas

VALIDAÇÕES

- Validações nos formulários são importantes para o sistema e o usuário;
- Sempre que um valor inserido pelo usuário for inválido (tipo de dados, valores, etc..) é necessário informar o usuário o motivo do erro e como corrigi-lo (Usabilidade);
- Existem várias técnicas para validação de formulários;

- Para realizar as validações nos formulário podemos utilizar os **Bean Validation** (JSR 303) e **Hibernate Validation** que possuem as principais validações básicas.
- Com a **Bean Validation** e **Hibernate Validation** é possível realizar as validações com metadados (**Anotações**) que podem ser utilizadas em todas as camadas de uma aplicação.

Anotação	Descrição
@AssertFalse	O valor deve ser falso
@AssertTrue	O valor deve ser verdadeiro
@DecimalMax	Valor deve ser menor ou igual
@DecimalMin	Valor deve ser maior ou igual
@Future	A data deve estar o futuro
@Past	A data deve estar no passado
@Min	O valor deve ser maior ou igual
@Max	O valor deve ser menor ou igual
@NotNull	O valor não pode ser nulo
@Size	A quantidade de elementos deve estar entre o min e max estabelecidos
@Null	O valor deve ser nulo
@Pattern	O valor deve obedecer a expressão regular

- As anotações possuem a propriedade **message** para definir uma mensagem customizada de erro:

```
@Entity
@SequenceGenerator(name="seqProduto", sequenceName="SEQ_PRODUTO", allocationSize=1)
public class Produto {

    @Id
    @GeneratedValue(strategy=GenerationType.SEQUENCE,generator="seqProduto")
    private int codigo;

    @NotNull
    @Size(min=2)
    private String titulo;

    @DecimalMin(value="0",message="Valor deve ser maior do que zero")
    private BigDecimal preco;

    @NotNull
    @Size(min=10, max=50, message="Número de caractères inválido")
    private String descricao;

    private boolean importado;

    @Temporal(TemporalType.DATE)
    @Past
    private Calendar dataFabricacao;

    //gets e sets..

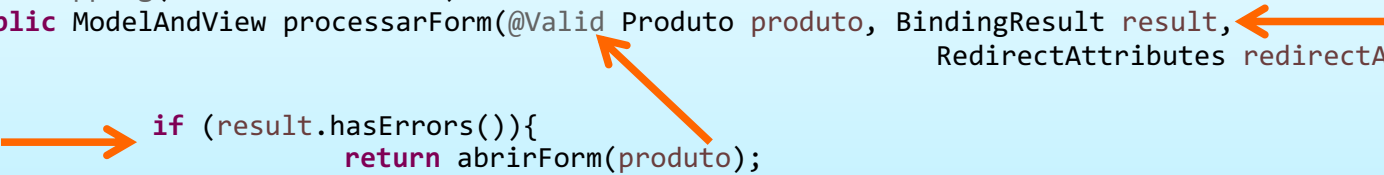
}
```


- Além da implementação da especificação JSR 303, o Hibernate Validation possui algumas outras anotações para validações específicas, como e-mail e cartão de crédito.

Anotação	Descrição
@Email	Validação de e-mail
@CredicardNumber	Validação de cartão de crédito
@NotBlank	Valor não pode ser nulo ou vazio (string)
@NotEmpty	Valor não pode ser nulo e deve possuir pelo menos um elemento

- Após utilizar as anotações para a validação, é necessário realizar alguns ajustes no controller;
- Para o framework validar o nosso modelo, precisamos utilizar a anotação **@Valid** para anotá-lo.
- O segundo parâmetro da ação deve ser o **BindingResult**, esse objeto possui as informações do resultado da validação.
- O BindingResult possui o método **hasErros()** que retorna se o model possui ou não erros de validação.

```
@Transactional
@PostMapping(value="cadastrar")
public ModelAndView processarForm(@Valid Produto produto, BindingResult result,
                                   RedirectAttributes redirectAttributes){
    if (result.hasErrors()){
        return abrirForm(produto);
    }
    try {
        dao.insert(produto);
    } catch (DBCommitException e) {
        e.printStackTrace();
    }
    redirectAttributes.addFlashAttribute("msg","Produto cadastrado!");
    return new ModelAndView("redirect:/produto/listar");
}
```



- Para exibir as mensagens de erro nas páginas podemos utilizar a tag `<form:erros>`, que possui o atributo **path** que identifica o atributo do model que a mensagem será exibida.

```
<form:form action="{action}" method="post" commandName="produto">
  <div class="form-group">
    <form:label path="titulo">Título</form:label>
    <form:input path="titulo" cssClass="form-control"/>
    → <form:erros path="titulo"/>
  </div>
  <div class="form-group">
    <form:label path="preco">Preço</form:label>
    <form:input path="preco" cssClass="form-control"/>
    → <form:erros path="preco"/>
  </div>
  <div class="form-group">
    <form:label path="dataFabricacao">Data de Fabricação</form:label>
    <form:input path="dataFabricacao" cssClass="form-control"/>
    → <form:erros path="dataFabricacao"/>
  </div>
  <div class="form-group">
    <input type="submit" value="Salvar" class="btn btn-primary"/>
  </div>
</form:form>
```

CONVERSÃO DE DADOS

- Como já vimos o spring framework faz as conversões de dados quando recebemos os valores como parâmetros no controller.
- Para alguns tipos de dados é possível configurar um conversor customizado.

- Para a conversão de data é possível utilizar a anotação **@DateTimeFormat**, que possui o atributo **pattern** para definir o formato da data.
- Com essa anotação o framework consegue converter automaticamente a string para uma data:

```
@DateTimeFormat(pattern="dd/MM/yyyy")  
private Calendar dataFabricacao;
```

- É possível configurar um **conversor de data global** para a aplicação web. Dessa forma, não será necessário adicionar a anotação `@DateTimeFormat` em todos os atributos de data.
- No arquivo **spring-context.xml** vamos configurar um bean responsável pela conversão de data:

```
<!-- Conversão de data -->
<bean id="conversionService" class="org.springframework.format.support.FormattingConversionServiceFactoryBean">
    <property name="registerDefaultFormatters" value="false" />
    <property name="formatters">
        <set>
            <bean class="org.springframework.format.number.NumberFormatAnnotationFormatterFactory" />
        </set>
    </property>
    <property name="formatterRegistrars">
        <set>
            <bean class="org.springframework.format.datetime.DateFormatterRegistrar">
                <property name="formatter">
                    <bean class="org.springframework.format.datetime.DateFormatter">
                        <property name="pattern" value="dd/MM/yyyy"/>
                    </bean>
                </property>
            </bean>
        </set>
    </property>
</bean>
<mvc:annotation-driven conversion-service="conversionService"/>
```



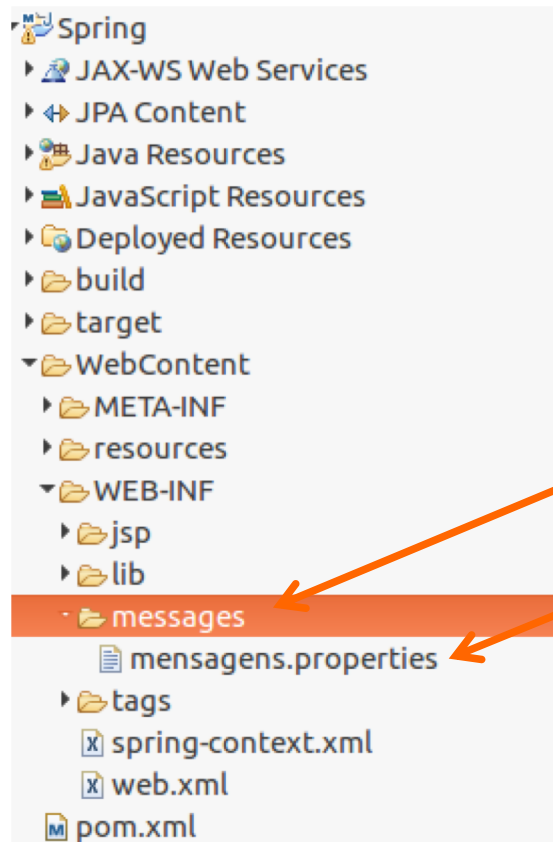
- Para exibir as datas formatadas, podemos utilizar a biblioteca de tags de formatação do JSTL:

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>  
  
<fmt:formatDate value="${data.time }" pattern="dd/MM/yyyy"/>
```


MENSAGENS CUSTOMIZADAS

- O framework possui as **mensagens padrões** de **erros** e de **conversão de dados**.
- É possível customiza-las adicionando um arquivo de **propriedades** e configurando o ResourceBundle no **spring-context.xml**.

- Crie uma pasta chamada messages dentro do diretório **WebContext/WEB-INF**;
- Crie um arquivo chamado **mensagens.properties**;



Crie o diretório

Crie o arquivo de propriedades

- Agora é preciso configurar o spring framework para utilizar o arquivo de propriedades para encontrar as mensagens customizadas.
- Para isso, faça a configuração no arquivo **spring-context.xml**:

```
<!-- Mensagens -->
<bean id="messageSource"
class="org.springframework.context.support.ReloadableResourceBundleMessageSource">
    <property name="basename">
        <value>/WEB-INF/messages/mensagens</value>
    </property>
    <property name="defaultEncoding">
        <value>UTF-8</value>
    </property>
</bean>
```



Diretórios e nome do arquivo

- O arquivo de propriedades é constituído por pares de **chave** e **valor**;
- **Para customizar as mensagens de conversão de dados:** adicione a palavra **typeMismatch** e a classe de conversão como chave da mensagem;
- **Para customizar as mensagens de erro:** utilize como chave o tipo de validação, se quiser, pode utilizar também o nome da classe de modelo e o atributo, para deixar a mensagem mais específica.

```
typeMismatch.java.lang.Integer = Digite somente números  
typeMismatch.java.math.BigDecimal = Digite somente números  
typeMismatch.java.util.Calendar = Data inválida  
  
NotBlank.produto.titulo = Título não pode estar vazio  
Size = Tamanho inválido
```

Copyright © 2017 - 2018 - Prof. Me. Thiago T. I. Yamamoto

Todos direitos reservados. Reprodução ou divulgação total ou parcial deste documento é expressamente proibido sem o consentimento formal, por escrito, do Professor (autor).

“Ter sucesso é falhar repetidamente, mas sem perder o entusiasmo”