

FIAP GRADUAÇÃO

DIGITAL BUSINESS ENABLEMENT

Prof. Me. Thiago T. I. Yamamoto

#15 – SPRING MVC - VIEWS



thiagoyama



thiagoyama@gmail.com

#15 – SPRING MVC - VIEWS

- JSTL
- Spring Form Tag Library
- Forward x Redirect
- Pathvariable

Conjunto de tags que podemos utilizar nas páginas JSPs.

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

- **<c:forEach>** - itera um conjunto de valores;
- **<c:url>** - formata uma url e armazena em uma variável;
- **<c:if>** - instrução condicional;

SPRING-FORM JSP TAG LIBRARY

- O Spring framework possui um conjunto de tags que ajuda o desenvolvedor na construção de páginas web.
- A biblioteca possui tags para construção de formulários e seus campos, mensagens de erro, internacionalização..

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

<FORM:FORM>

- Podemos utilizar a tag **<form:form>** para construir um formulário:
- **action** – url que irá processar as informações do formulário;
- **method** – método HTTP que o formulário vai utilizar;
- **commandName** – nome da classe de modelo, que possui os atributos que serão referenciados no formulário;

```
<c:url value="/produto/cadastrar" var="action"/>
<form:form action="${action}" method="post" commandName="produto">
    //...
</form:form>
```

Vamos utilizar a tag `<c:url>` para criar a url correta para enviar os dados do formulário. Essa tag não pode estar dentro da tag `<form:form>`.

```
@GetMapping("cadastrar")
public String abrirForm(Produto produto){
    return "produto/cadastro";
}
```

No controller, precisamos alterar o método que abre a tela do formulário para que ele recebe um objeto do tipo do modelo do formulário (*commandName*).

- A biblioteca de tags do spring possui tags para construção dos campos de formulário como input text, textarea, password, checkbox, select, hidden e etc..
- Os dois principais atributos dessas tags são:
- **path** – nome da propriedade do model para o campo;
- **cssClass** – classe para css.

```
<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"/>
  </div>
  <div class="form-group">
    <form:label path="preco">Preço</form:label>
    <form:input path="preco" cssClass="form-control"/>
  </div>
  <div class="form-group">
    <form:label path="descricao">Descrição</form:label>
    <form:textarea path="descricao" rows="10" cols="20" cssClass="form-control"/>
  </div>
  <div class="form-group">
    <form:checkbox path="importado"/>
    <form:label path="importado">Importado</form:label>
  </div>
  <div class="form-group">
    <input type="submit" value="Salvar" class="btn btn-primary"/>
  </div>
</form:form>
```


- **<form:label>** - Tag de label;
- **<form:input>** - Campo de texto;
- **<form:textarea>** - Campo de textarea;
- **<form:checkbox>** - Campo de checkbox;
- **<form:radiobutton>** - Campo de radio button;
- **<form:password>** - Campo de senha;
- **<form:hidden>** - Campo oculto;
- **<form:select>** - Campo de seleção;
- **<form:option>** - Opção para o campo de seleção;

<http://docs.spring.io/spring/docs/current/spring-framework-reference/html/view.html>

FORWARD E REDIRECT

Forward:

- É executado internamente pelo servidor;
- O browser não sabe o que está ocorrendo durante o processamento no servidor;
- No final do processamento da requisição a url no browser não se altera;
- O reload da página resultante irá executar a requisição original;

Redirect:

- É um processo de dois passos, ao receber uma requisição a aplicação web “pede” para o browser acessar a segunda url, por isso a url muda;
- O reload da página não irá processar a requisição original;
- Os atributos colocados no primeiro request **são perdidos** durante o segundo request;

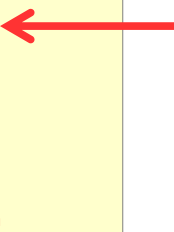
É uma boa prática realizar um **redirect** após um POST, pois caso o usuário atualize a página não será realizado a mesma operação novamente (cadastro, atualização ou remoção).

```
@Transactional
@PostMapping(value="cadastrar")
public ModelAndView processarForm(Produto produto){
    try {
        dao.insert(produto);
    } catch (DBCommitException e) {
        e.printStackTrace();
    }
    return new ModelAndView("redirect:/produto/listar");
}
```




- No redirect não é possível passar atributos do primeiro request para o segundo request como de costume. Para isso precisamos do escopo flash.
- Precisamos receber como parametro do método um objeto do tipo **RedirectAttributes**, esse objeto possui o método **addFlashAttribute**, que permite adicionar atributos para o segundo request.

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



```
<c:if test="${not empty msg}">
    <div class="alert alert-success">
        ${msg}
    </div>
</c:if>
```



Exibe a mensagem enviada pelo controller

EXERCÍCIO

Faça o cadastro e a listagem de um livro:

- Código;
- Título;
- Número de páginas;
- Sumário;

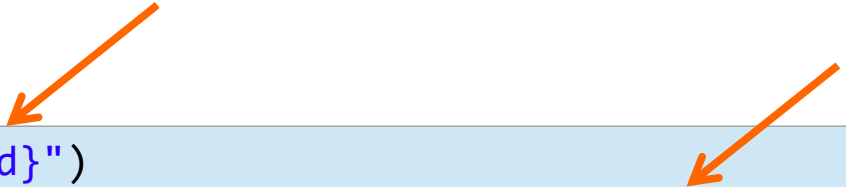
PATHVARIABLE

Na listagem abaixo, observe que temos um link para editar um produto. Esse link envia o **id** do produto que será atualizado:

```
<table class="table table-striped">
  <thead>
    <tr>
      <th>Título</th>
      <th>Preço</th>
      <th>Descrição</th>
      <th>Categoria</th>
      <th>Importado</th>
      <th></th>
    </tr>
  </thead>
  <tbody>
    <c:forEach items="{produtos}" var="p">
      <tr>
        <td>${p.titulo}</td>
        <td>${p.preco}</td>
        <td>${p.descricao}</td>
        <td>${p.categoria.descricao}</td>
        <td>${p.importado}</td>
        <td>
          <a href="<c:url value="/produto/editar/${p.codigo }"/>">Editar</a>
        </td>
      </tr>
    </c:forEach>
  </tbody>
</table>
```



Para receber o id, precisamos adicionar o parametro no **path** da url e recebe-lo no parâmetro do método, utilizando a anotação **@PathVariable** para informar que o valor recebido na url deve ser passado no parâmetro do método:



```
@GetMapping("/editar/{id}")
public ModelAndView abrirFormEdicao(@PathVariable("id") int id){
    ModelAndView retorno = new ModelAndView("produto/edicao");
    retorno.addObject("produto", dao.findById(id));
    return retorno;
}
```

Complete o CRUD para o livro!

Para não ter problemas com encoding de caracteres no spring vamos configurar no arquivo **web.xml** um filtro para que ele utilize o charset UTF-8:

```
<filter>
  <filter-name>encoding-filter</filter-name>
  <filter-class>
    org.springframework.web.filter.CharacterEncodingFilter
  </filter-class>
  <init-param>
    <param-name>encoding</param-name>
    <param-value>UTF-8</param-value>
  </init-param>
  <init-param>
    <param-name>forceEncoding</param-name>
    <param-value>true</param-value>
  </init-param>
</filter>

<filter-mapping>
  <filter-name>encoding-filter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

Copyright © 2017 - 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).

“Não importa o quão devagar você vá, desde que você não pare”