Java 2 Enterprise Edition

Construção de Aplicações Web

Helder da Rocha
www.argonavis.com.br

Objetivos

- Este módulo tem como objetivo apenas apresentar os fundamentos básicos de taglibs e abordar superficialmente frameworks MVC
  - Uma abordagem mais profunda foge do escopo deste curso, que não trata apenas de aplicações Web.
  - Para maior aprofundamento, consulte os exemplos de aplicações incluídas no CD (cap12) e as referências no final desta apresentação.
- Este módulo é opcional
  - Para mais detalhes, consulte os slides dos cursos J550 (JSP e servlets), J551 (Struts) e J931 (Design Patterns J2EE)

1. Custom tags

- *JSP com JavaBeans fornecem um meio de diminuir código Java da página, mas não totalmente*
  - *Designers de página ainda têm que usar elementos de script para loops e lógica condicional (getProperty e setProperty não bastam)*
  - *Nem sempre os JavaBeans são suficientes para encapsular toda a lógica da aplicação*
- *A especificação prevê a criação de elementos XML personalizados (custom tags) para resolver essas limitações*
  - *Organizados em bibliotecas (taglibs)*
  - *Cada biblioteca tem seu próprio namespace*
- *Taglibs são declaradas no início de cada página ...*

```
<%@taglib uri="http://abc.com/ex" prefix="exemplo"%>
```

namespace

- *... e usadas em qualquer lugar*

```
<exemplo:dataHoje />
```

produz → **Tuesday, May 5, 2002 13:13:13 GMT-03**

Como usar custom tags

- A URI usada para identificar o prefixo de um custom tag não precisa ser real (e apontar para um local)
  - Serve apenas como **identificador**
  - Ligação entre a especificação da biblioteca (arquivo TLD) e o identificador é feito no arquivo web.xml

```
<web-app>
    ...
    <taglib>
        <taglib-uri>http://abc.com/ex</taglib-uri>
        <taglib-location>
        /WEB-INF/mytaglib.tld
        </taglib-location>
    </taglib>
</web-app>
```

Este é o deployment descriptor do Taglib.

Localização real!

Exemplo de arquivo TLD

```
<?xml version="1.0" ?>
<!DOCTYPE taglib PUBLIC
"-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
"http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>
    <tlib-version>1.0</tlib-version>
    <jsp-version>1.2</jsp-version>
    <short-name>exemplo</short-name>
    <uri>http://abc.com/ex</uri>

    <tag>
        <name>dataHoje</name>
        <tag-class>exemplos.DateTag</tag-class>
        <description>Data de hoje</description>
    </tag>

</taglib>
```

Sugestão de prefixo (autor de página pode escolher outro na hora)

URI identifica o prefixo. (autor de página tem que usar exatamente esta URI)

Implementação

```
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class DateTag extends TagSupport {
    /**
     * Chamado quando o tag terminar.
     */
    public int doEndTag() throws JspException {
        try {
            Writer out = pageContext.getOut();
            java.util.Date = new java.util.Date();
            out.println(hoje.toString());
        } catch (java.io.IOException e) {
            throw new JspException (e);
        }
        return Tag.EVAL_PAGE;
    }
}
```

Para tags que não precisam processar o corpo use a interface Tag ou sua implementação TagSupport

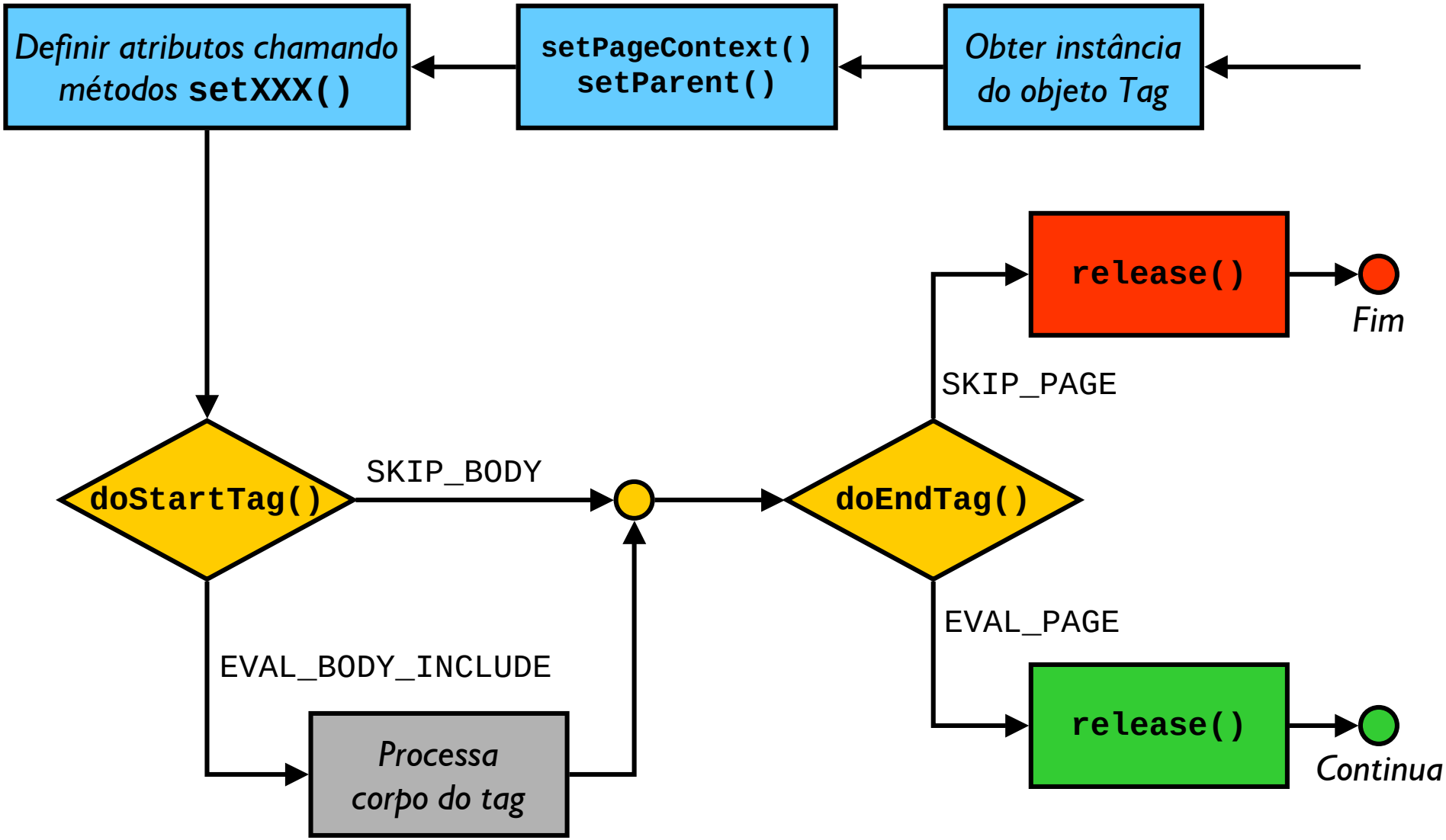
Use doStartTag() para processamento antes do tag, se necessário

Para este método, pode ser EVAL_PAGE ou SKIP_PAGE

Tipos de tags

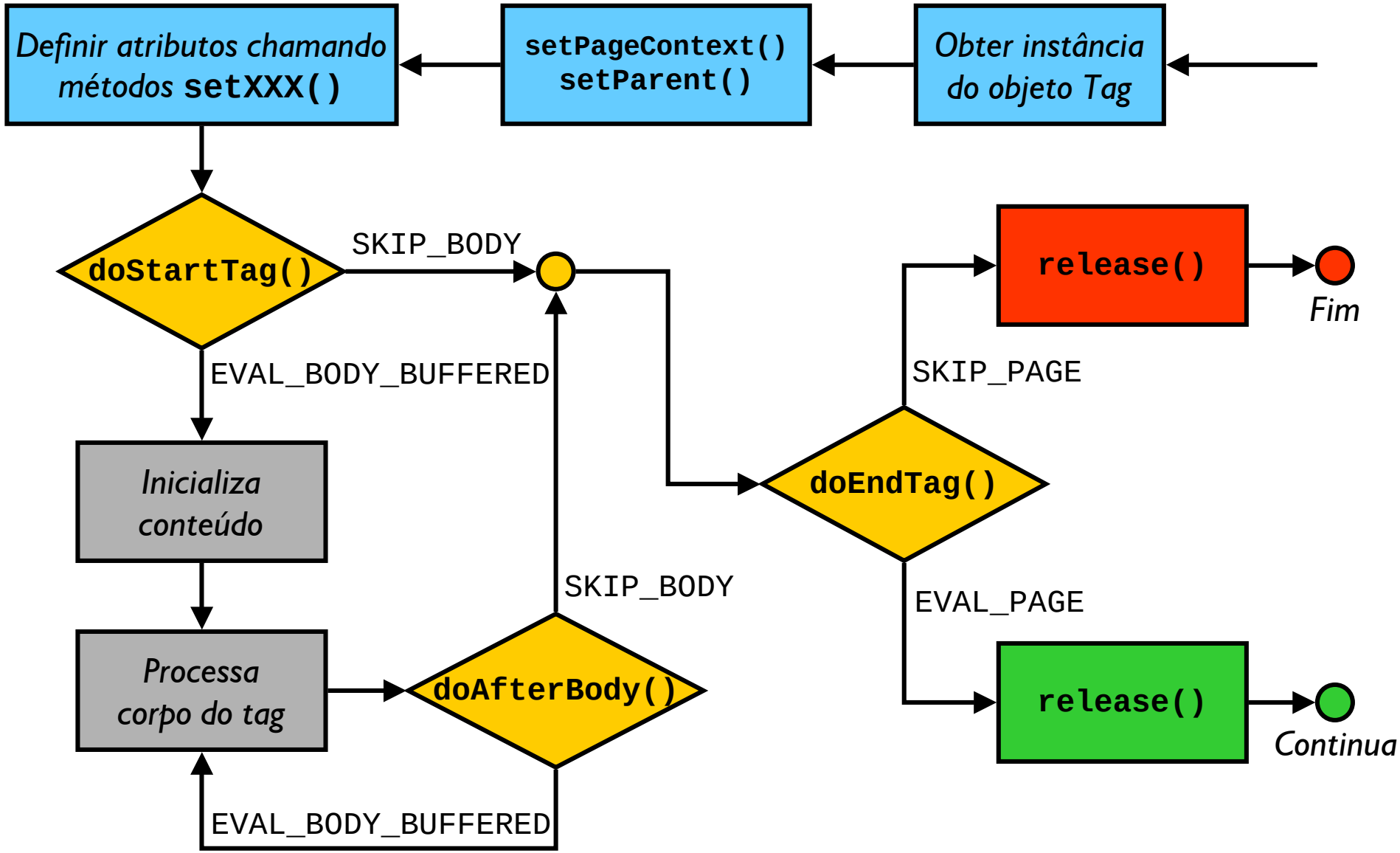
- *Há vários tipos de custom tags. Cada estratégia utiliza diferentes classes base e métodos*
- *1. Diferenciados por herança:*
  - *Tags simples: implementam a interface Tag (TagSupport é uma implementação neutra).*
  - *Tags com corpo que requer processamento: implementam BodyTag (BodyTagSupport é implementação neutra) ou IterationTag*
- *Diferenciados por outras características*
  - *Tags que possuem atributos*
  - *Tags que definem variáveis de scripting fora do seu escopo (requerem classe extra com "Tag Extra Info")*
  - *Tags que interagem com outros tags*

Ciclo de vida de objetos Tag



Fonte: [6]

Ciclo de vida de objetos BodyTag



Fonte: [6]

Atributos

- *Para definir atributos em um tag é preciso*

```
<xyz:upperCase text="abcd" />
```

1. *Definir método setXXX() com o nome do atributo*

```
public class UpperCaseTag {
    public String text;
    public void setText(String text) {
        this.text = text;
    } (...)
```

2. *Declarar atributo no descritor (TLD)*

```
<tag> <name>upperCase</name> (...)
    <attribute>
        <name>text</name>
        <required>true</required>
        <rtexprvalue>false</rtexprvalue>
    </attribute> (...) </tag>
```

- *Os atributos devem setar campos de dados no tag*
  - *Valores são manipulados dentro dos métodos doXXX():*

```
public int doStartTag() throws JspException {
    Writer out = pageContext.getOut();
    out.println(text.toUpperCase());   (...)
```

Obtenção do conteúdo do Body

- *O objeto out, do JSP, referencia a instância BodyContent de um tag enquanto processa o corpo*
  - *BodyContent é subclasse de JspWriter*
  - *Tag decide se objeto BodyContent deve ser jogado fora ou impresso (na forma atual ou modificada)*
- *Exemplo*

```
public int doAfterBody() throws JspException {
    BodyContent body = getBodyContent();
    String conteudo = body.getString();
    body.clearBody();
    (...)
    getPreviousOut().print(novoTexto);
}
```

Guarda conteúdo do tag

Apaga conteúdo

Imprime texto na página (e não no body do Tag)

Exemplos de Custom Tags

- *Veja* **exemplos/cap08/taglibs/**
  - *Vários diferentes exemplos de custom tags (do livro [6])*
  - *Código fonte em taglib/src/taglibdemo/*.java*
  - *Páginas exemplo em src/*Test.jsp (6 exemplos)*

1. Configure build.properties, depois, monte o WAR com:

```
> ant build
```

2. Copie o WAR para o diretório webapps do Tomcat

```
> ant deploy
```

3. Execute os tags, acessando as páginas via browser:

http://localhost:porta/mut/

- *Veja também* **exemplos/cap08/mvc/hellojsp_2/**
  - *Aplicação MVC que usa custom tags (veja como executar na próxima seção)*

JSP Standard Tag Library

- Esforço de padronização do JCP: JSR-152
  - Baseado no Jakarta Taglibs (porém bem menor)
- Oferece dois recursos
  - Conjunto padrão de tags básicos (Core, XML, banco de dados e internacionalização)
  - Linguagem de expressões do JSP 1.3
- Oferece mais controle ao autor de páginas sem necessariamente aumentar a complexidade
  - Controle sobre dados sem precisar escrever scripts
  - Estimula a separação da apresentação e lógica
  - Estimula o investimento em soluções MVC

Como usar JSTL

- *1. Fazer o download da última versão do site da Sun*
- *2. Copiar os JARs das bibliotecas desejadas para o diretório WEB-INF/lib/ da sua aplicação Web e os arquivos TLD para o diretório WEB-INF/*
- *3. Declarar cada taglib e associá-la com seu TLD no deployment descriptor web.xml.*
- *4. Incluir em cada página que usa os tags:*

```
<%@ taglib uri="uri_da_taglib"
           prefix="prefixo" %>
```

- *5. Usar os tags da biblioteca com o prefixo definido no passo anterior*

```
<prefixo:nomeTag atributo="..."> ...
</prefixo:nomeTag>
```

Quatro bibliotecas de tags

- *Core library: tags para condicionais, iterações, urls, ...*

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

  - *Exemplo:* `<c:if test="..." ... >...</c:if>`
- *XML library: tags para processamento XML*

```
<%@ taglib uri="http://java.sun.com/jstl/ea/xml"
           prefix="x" />
```

  - *Exemplo:* `<x:parse>...</x:parse>`
- *Internationalization library*

```
<%@ taglib uri="http://java.sun.com/jstl/ea/fmt"
           prefix="fmt" />
```

  - *Exemplo:* `<fmt:message key="..." />`
- *SQL library*

```
<%@ taglib uri="http://java.sun.com/jstl/ea/sql"
           prefix="sql" />
```

  - *Exemplo:* `<sql:update>...</sql:update>`

Linguagem de expressões

- *Permite embutir em atributos expressões dentro de delimitadores ${...}*
  - *Em vez de* `request.getAttribute("nome")`
    `${nome}`
  - *Em vez de* `bean.getPessoa().getNome()`
    `${bean.pessoa.nome}`
- *Suporta operadores aritméticos, relacionais e binários*
- *Converte tipos automaticamente*

```
<tag item="${request.valorNumerico}" />
```

- *Valores default*

```
<tag value="${abc.def}" default="todos" />
```

Principais ações

- *Suporte à impressão da linguagem expressões*
  - `<c:out value="${pessoa.nome}" />`
- *Expressões condicionais*
  - `<c:if test="${pessoa.idade >= 18}">`
    `<a href="adultos.html">Entrar</a>`
    `</c:if>`
  - `<c:choose>`
    `<c:when test="${dia.hora == 13}">`
    `<c:out value="${mensagemEspecial}" />`
    `</c:when>`
    `<c:otherwise>`
    `<c:out value="${mensagemPadrao}" />`
    `</c:otherwise>`
  - `</c:choose>`
- *Iteração*
  - `<c:forEach items="${pessoas}" var="p" varStatus="s">`
    `<c:out value="${s.count}"/>. <c:out value="${p}"/>`
    `</c:forEach>`

Internacionalização, XML e SQL

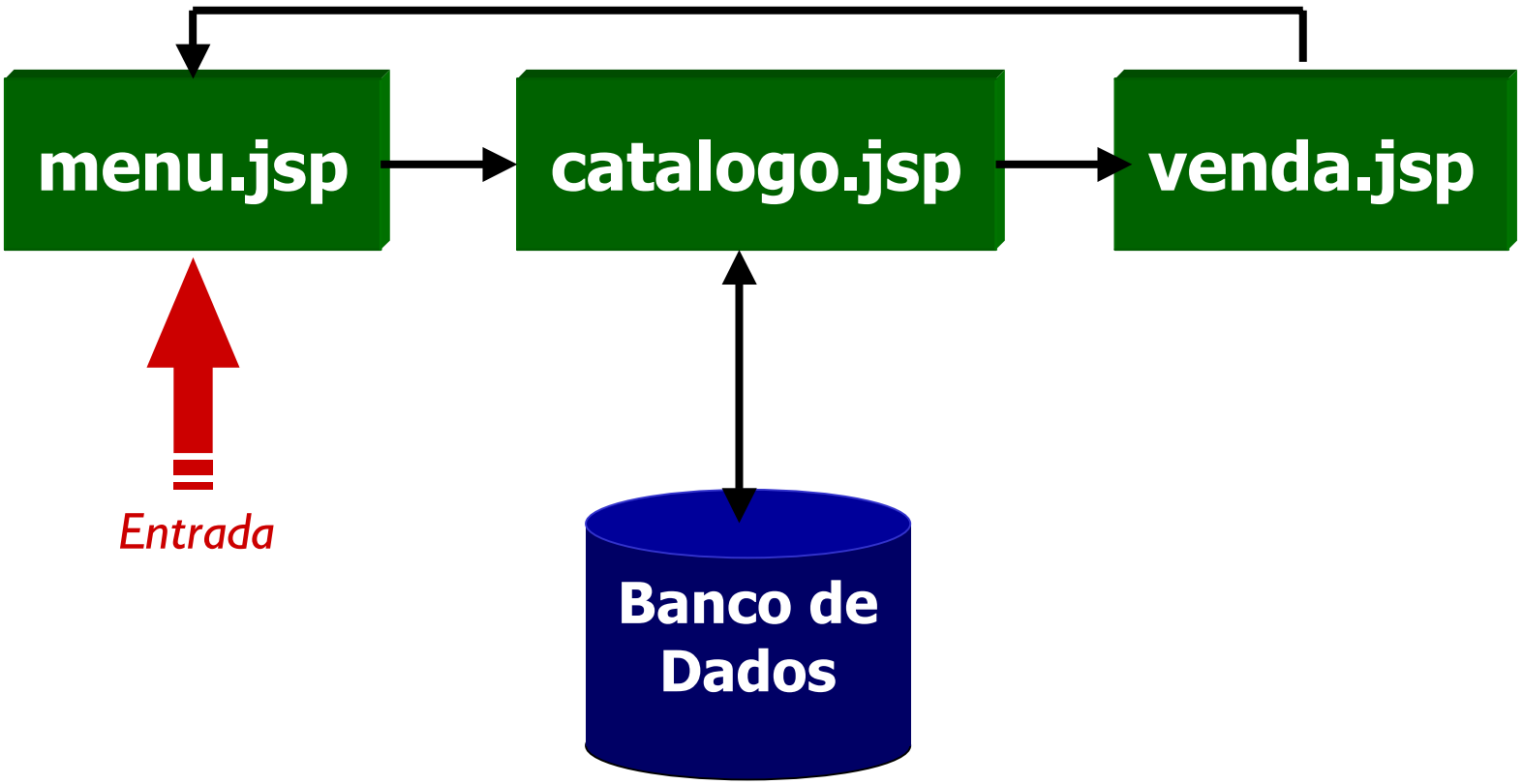
- Ler propriedade de ResourceBundle
  - `<fmt:message key="chave.do.bundle" />`
- Operações diretas em banco de dados
  - `<sql:query dataSource="${dsn}">`
    `SELECT...</sql:query>`
  - `<sql:transaction>`, `<sql:update>`, *etc.*
- Operações com XML
  - *Uso de expressões XPath em tags JSTL para XML*
  - *Ações XML:* `<x:out>`, `<x:set>`, `<x:if>`, `<x:choose>`, `<x:forEach>` *(atributo select contém expr. XPath)*
  - `<x:parse>` *Processa XML usando DOM ou filtro SAX*
  - `<x:transform>` *Realiza transformação XSLT.*

2. Design de aplicações JSP

- *Design centrado em páginas*
  - *Aplicação JSP consiste de seqüência de páginas (com ou sem beans de dados) que contém código ou links para chamar outras páginas*
- *Design centrado em servlet (FrontController* ou MVC)*
  - *Aplicação JSP consiste de páginas, beans e servlets que controlam todo o fluxo de informações e navegação*
  - *Este modelo favorece uma melhor organização em camadas da aplicação, facilitando a manutenção e promovendo o reuso de componentes.*
  - *Um único servlet pode servir de fachada*
  - *Permite ampla utilização de J2EE design patterns*

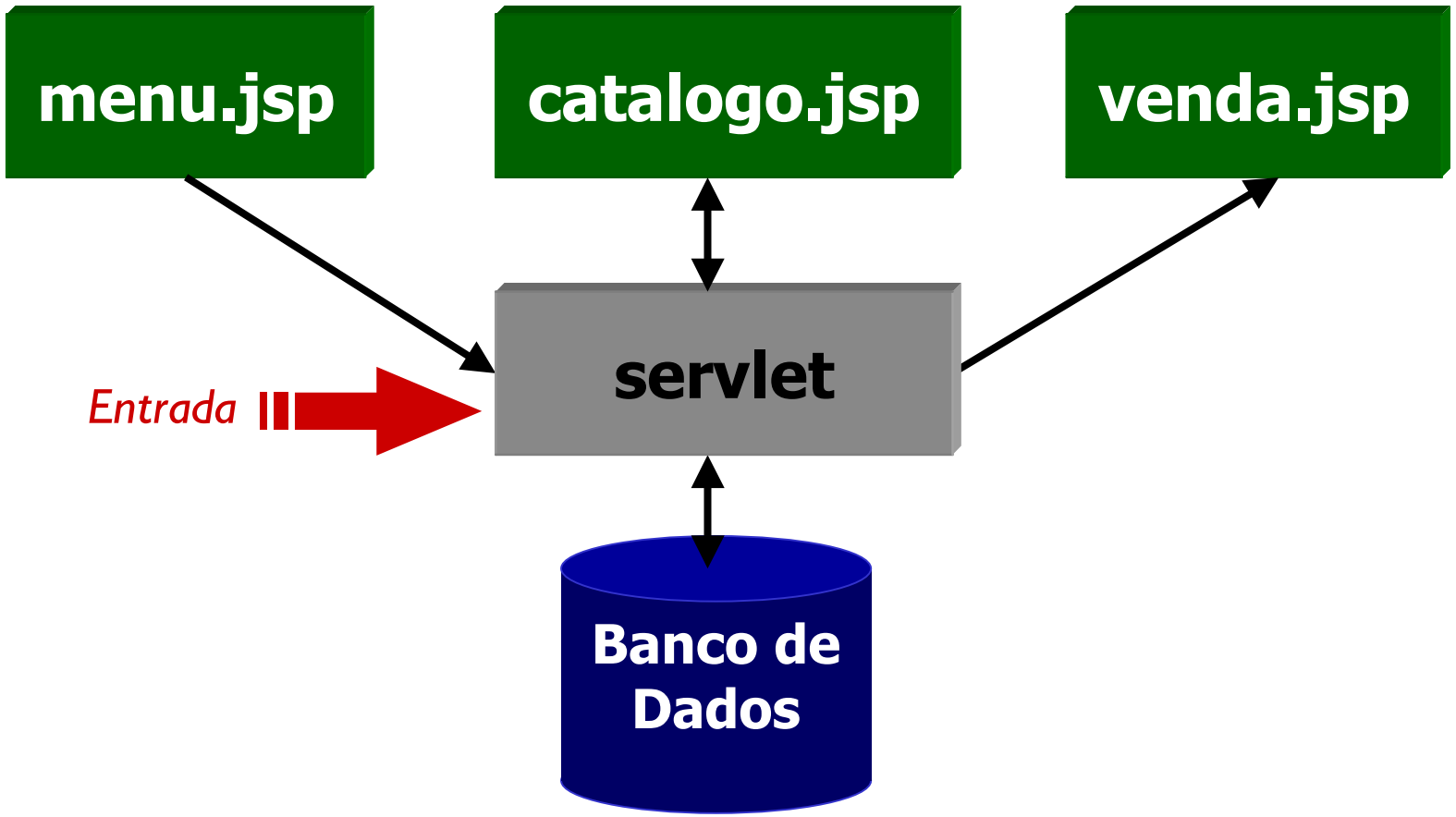
** FrontController é um J2EE design pattern. Vários outros design patterns serão identificados durante esta seção. Para mais informações, veja Sun Bluebrints [7]*

Layout centrado em páginas (JSP Model 1)



Fonte: [6]

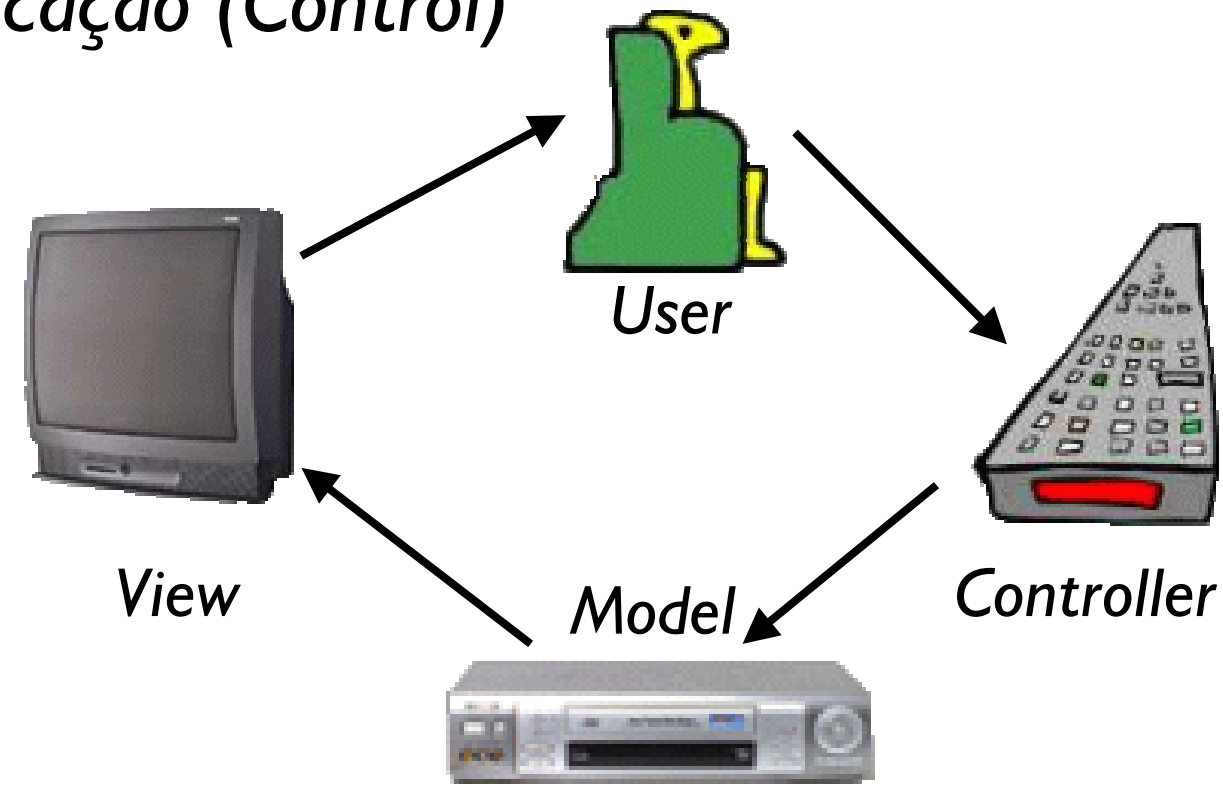
Layout centrado em servlet (JSP Model 2)



Fonte: [6]

O que é MVC

- *Padrão de arquitetura: Model View Controller*
- *Técnica para separar dados ou lógica de negócios (Model) da interface do usuário (View) e do fluxo da aplicação (Control)*



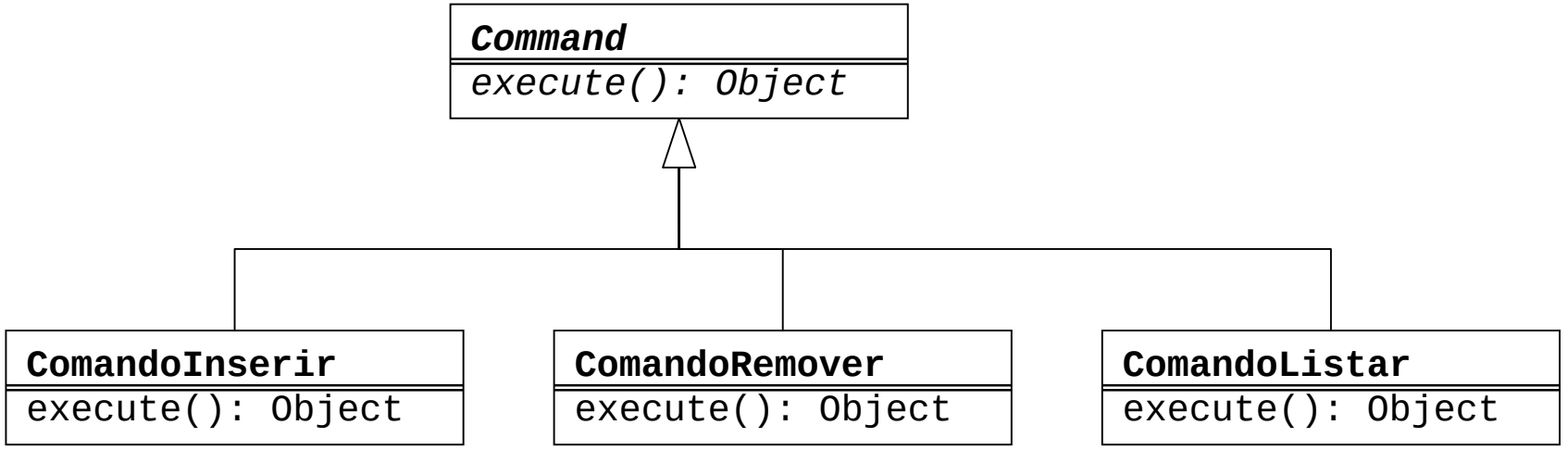
Fonte: *http://www.computer-programmer.org/articles/struts/*

Como implementar?

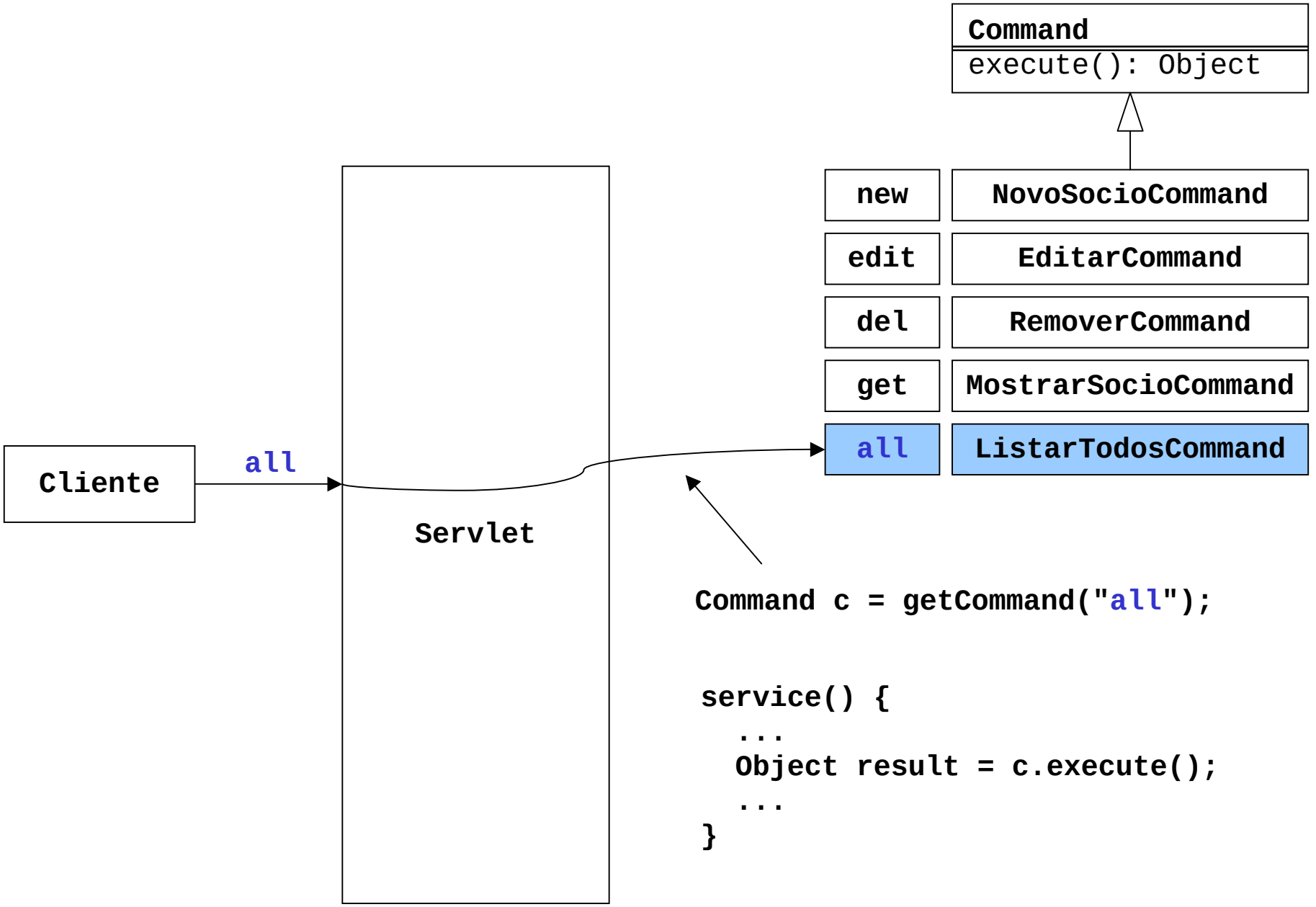
- Há várias estratégias
- Todas procuram isolar
  - As operações de controle de requisições em servlets e classes ajudantes,
  - Operações de geração de páginas em JSP e JavaBeans, e
  - Lógica das aplicações em classes que não usam os pacotes javax.servlet
- Uma estratégia consiste em se ter um único controlador (FrontController pattern) que delega requisições a diferentes objetos que implementam comandos que o sistema executa (Command pattern)

Command Pattern

- *É um padrão de projeto clássico catalogado no livro "Design Patterns" de Gamma et al (GoF = Gang of Four)*
  - *Para que serve: "Encapsular uma requisição como um objeto, permitindo que clientes parametrizem diferentes requisições, filas ou requisições de log, e suportar operações reversíveis." [GoF]*
- *Consiste em usar polimorfismo para construir objetos que encapsulam um comando e oferecer um único método execute() com a implementação do comando a ser executado*



Command Pattern



FrontController com Command Pattern

- Os comandos são instanciados e guardados em uma base de dados na memória (HashMap, por exemplo)
  - Pode-se criar uma classe específica para ser fábrica de comandos
- O cliente que usa o comando (o servlet), recebe na requisição o nome do comando, consulta-o no HashMap, obtém a instância do objeto e chama seu método execute()
  - O cliente desconhece a classe concreta do comando. Sabe apenas a sua interface (que usa para fazer o cast ao obtê-lo do HashMap
- No HashMap

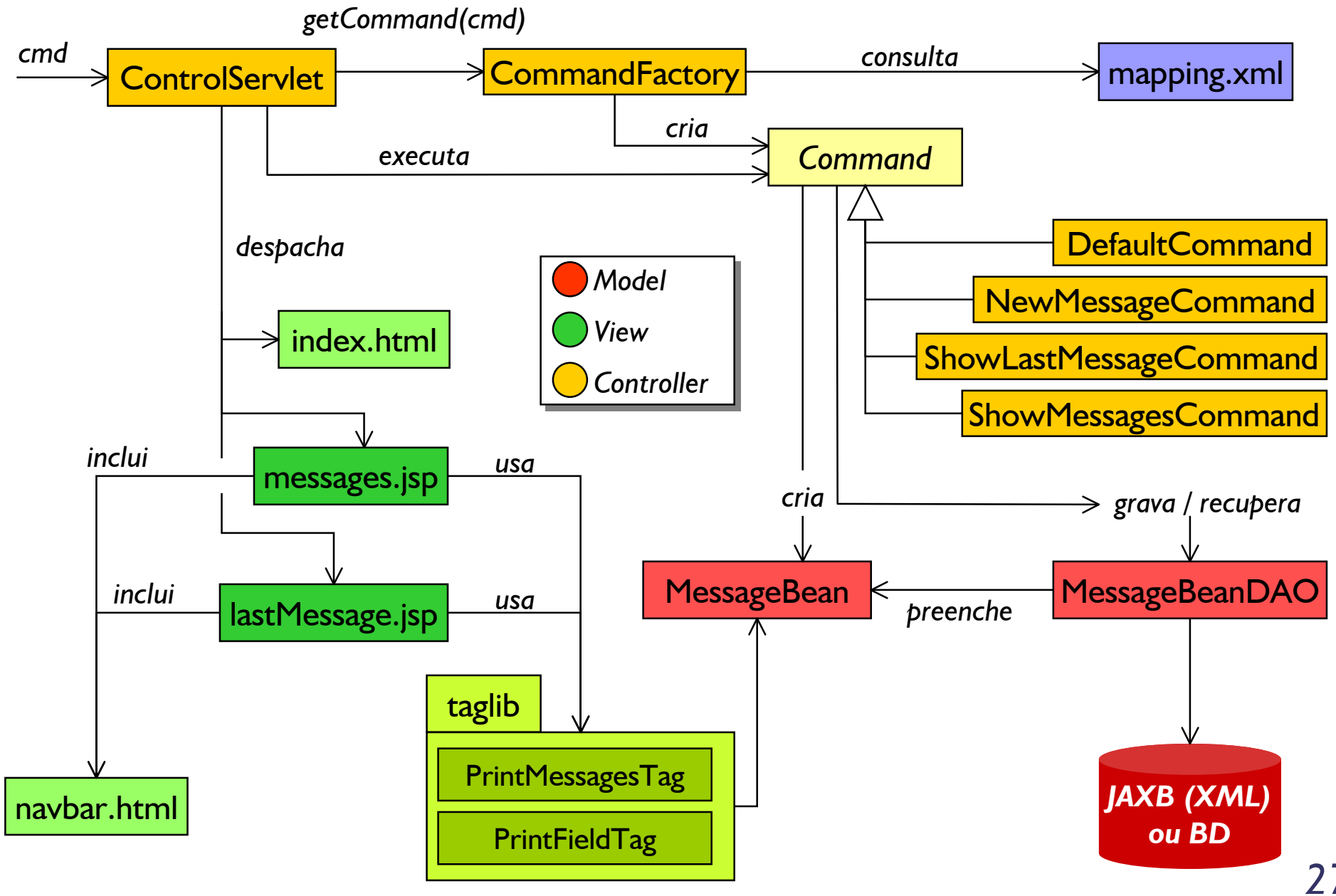
```
Comando c = new ComandoInserir();
comandosMap.put("inserir", c);
```

- No servlet:

```
String cmd = request.getParameter("cmd");
Comando c = (Comando)comandosMap.get(cmd);
c.execute();
```

Exemplo de implementação

exemplos/cap08/mvc/hellojsp_2



Mapeamentos de comandos ou ações

- *No exemplo hellojsp_2, o mapeamento está armazendo em um arquivo XML (webinf/mapping.xml)*

```
<command-mapping> (...)
    <command>
        <name>default</name>
        <class>hello.jsp.DefaultCommand</class>
        <success-url>/index.html</success-url>
        <failure-url>/index.html</failure-url>
    </command>
    <command>
        <name>newMessage</name>
        <class>hello.jsp.NewMessageCommand</class>
        <success-url>/lastMessage.jsp</success-url>
        <failure-url>/index.html</failure-url>
    </command>
    <command>
        <name>showAllMessages</name>
        <class>hello.jsp.ShowMessagesCommand</class>
        <success-url>/messages.jsp</success-url>
        <failure-url>/index.html</failure-url>
    </command>
</command-mapping>
```

Comandos ou ações (Service to Worker)

- *Comandos implementam a interface Command e seu método*

```
Object execute(HttpServletRequest request,
               HttpServletResponse response,
               MessageBeanDAO dao);
```

- *Criados por CommandFactory na inicialização e executados por ControlServlet que os obtém via getCommand(nome)*
- *Retornam página de sucesso ou falha (veja mapping.xml)*
- *Exemplo: ShowMessagesCommand:*

```
public class ShowMessagesCommand implements Command {

   public Object execute(...) throws CommandException {
        try {
            MessageBean[] beanArray = dao.retrieveAll();
            request.setAttribute("messages", beanArray);
            return successUrl;
        } catch (PersistenceException e) {
            throw new CommandException(e);
        }
   } (...)
```

Data Access Objects (DAO)

- *Isolam a camada de persistência*
  - *Implementamos persistência JAXB, mas outra pode ser utilizada (SGBDR) sem precisar mexer nos comandos.*
- *Interface da DAO:*

```
public interface MessageBeanDAO {
   public Object getLocator();

   public void persist(MessageBean messageBean)
                              throws PersistenceException;

   public MessageBean retrieve(int key)
                              throws PersistenceException;

   public MessageBean[] retrieveAll()
                              throws PersistenceException;

   public MessageBean retrieveLast()
                              throws PersistenceException;
}
```

Controlador (FrontController)

- *Na nossa aplicação, o controlador é um servlet que recebe os nomes de comandos, executa os objetos que os implementam e repassam o controle para a página JSP ou HTML retornada.*

```
public void service( ..., ... ) ... {
    Command command = null;
    String commandName = request.getParameter("cmd");

    if (commandName == null) {
        command = commands.getCommand("default");
    } else {
        command = commands.getCommand(commandName);
    }

    Object result = command.execute(request, response, dao);
    if (result instanceof String) {
        RequestDispatcher dispatcher =
              request.getRequestDispatcher((String)result);
        dispatcher.forward(request, response);
    }
   ...
}
```

Método de CommandFactory

Execução do comando retorna uma URI

Repassa a requisição para página retornada

ValueBean ViewHelper (Model)

- *Este bean é gerado em tempo de compilação a partir de um DTD (usando ferramentas do JAXB)*

```
public class MessageBean
    extends MarshallableRootElement
    implements RootElement {

    private String _Time;
    private String _Host;
    private String _Message;

    public String getTime() {...}
    public void setTime(String _Time) {...}

    public String getHost() {...}
    public void setHost(String _Host) {...}

    public String getMessage() {...}
    public void setMessage(String _Message) {...}

    ...
}
```

interfaces JAXB permitem que este bean seja gravado em XML (implementa métodos marshal() e unmarshal() do JAXB)

Página JSP (View) com custom tags

- *Página messages.jsp (mostra várias mensagens)*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<%@ taglib uri="/hellotags" prefix="hello" %>
<html>
<head><title>Show All Messages</title></head>
<body>
<jsp:include page="navbar.html" />
<h1>Messages sent so far</h1>
<table border="1">
<tr><th>Time Sent</th><th>Host</th><th>Message</th></tr>

<hello:printMessages array="messages">
    <tr>
        <td><hello:printField property="time" /></td>
        <td><hello:printField property="host" /></td>
        <td><hello:printField property="message" /></td>
    </tr>
</hello:printMessages>

</table>
</body>
</html>
```

Para executar o exemplos

- 1. Mude para exemplos/cap08/mvc/hellojsp_2
- 2. Configure build.properties, depois rode

  `> ant DEPLOY`
- 3. Inicie o servidor (Tomcat ou JBoss)
- 4. Rode os testes do Cactus

  `> ant RUN-TESTS`
- 5. Rode a aplicação, acessando a URI

  `http://localhost:porta/hellojsp/`
- 6. Digite mensagens e veja resultados. Arquivos são gerados em /tmp/mensagens (ou c:\tmp\mensagens)

O que é Struts?

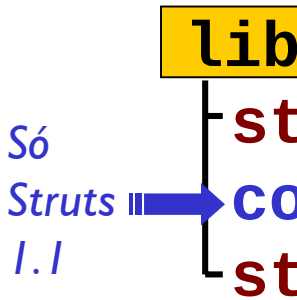
- Framework para facilitar a implementação da arquitetura MVC em aplicações JSP
- Oferece
  - Um servlet controlador configurável através de documentos XML externos, que despacham requisições a classes Action (comandos) criadas pelo desenvolvedor
  - Uma vasta coleção de bibliotecas de tags JSP (taglibs)
  - Classes utilitárias que oferecem suporte a tratamento de XML, preenchimento de JavaBeans e gerenciamento externo do conteúdo de interfaces do usuário
- Onde obter: jakarta.apache.org/struts

Componentes MVC no Struts

- Model (M)
  - Geralmente um objeto Java (JavaBean)
- View (V)
  - Geralmente uma página HTML ou JSP
- Controller (C)
  - org.apache.struts.action.ActionServlet ou subclasse
- Classes ajudantes
  - FormBeans: encapsula dados de forms HTML (M)
  - ActionErrors: encapsulam dados de erros (M)
  - Custom tags: encapsulam lógica para apresentação (V)
  - Actions: implementam lógica dos comandos (C)
  - ActionForward: encapsulam lógica de redirecionamento (C)

Componentes da distribuição

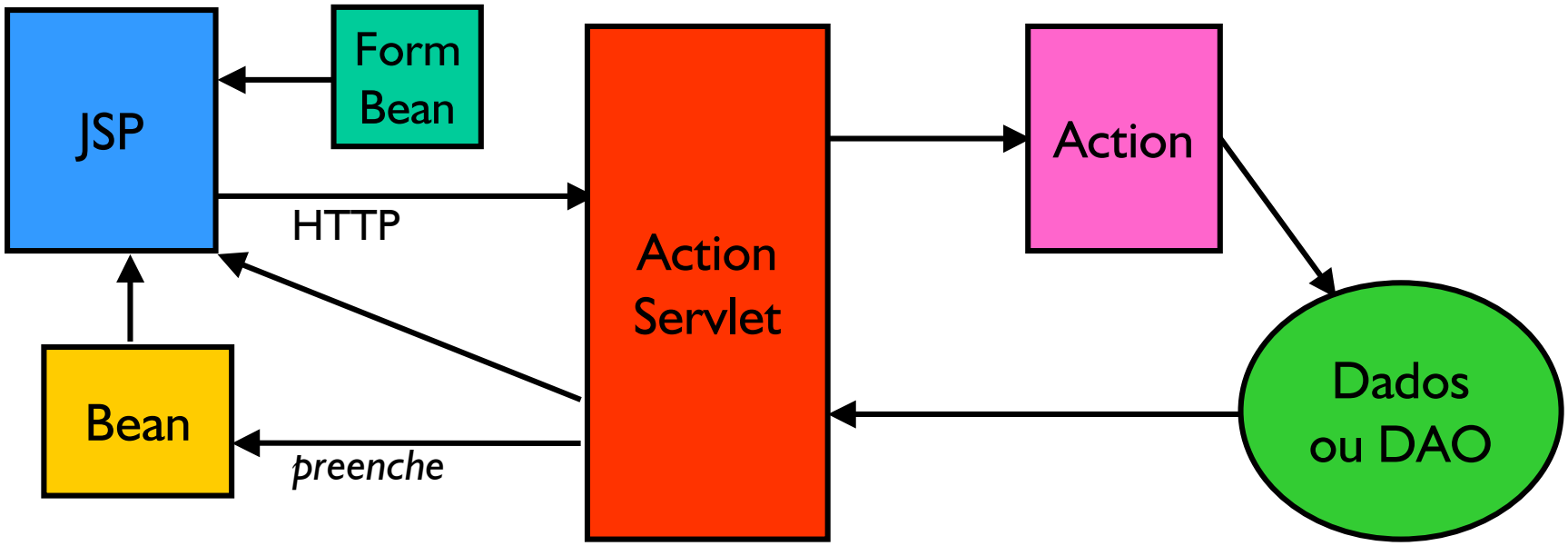
- *Requisitos*
  - *J2SDK 1.4 ou J2SDK1.3 + JAXP*
  - *Servlet container, servlet.jar e Jakarta Commons (Struts 1.1)*
- *Distribuição binária (pré-compilada)*
  - *Abra o ZIP da distribuição. Conteúdo essencial:*



- *Componentes essenciais*
  - *Framework (copie para WEB-INF/lib **de cada aplicação** Web)*
  - *Descritores de taglib (copie para WEB-INF **de cada aplicação**)*
- *Aplicações Web (opcionais)*
  - *(jogue no webapps do Tomcat - instale pelo menos struts-documentation.war)*

Como funciona?

- *Principais componentes*
  - *ActionServlet: despachante de ações*
  - *Action: classe estendida por cada ação (comando) a ser implementada (usa Command design pattern)*
  - *struts-config.xml: arquivo onde se define mapeamentos entre ações, páginas, beans e dados*



Como instalar

- 1. Copiar os arquivos necessários para sua aplicação
  - Copie lib/struts.jar e lib/commons-*.jar para weu WEB-INF/lib (não coloque no common/lib do Tomcat ou no jre/lib/ext do JDK ou o struts não achará suas classes!)
  - Copie os TLDs das bibliotecas de tags que deseja utilizar para o WEB-INF de sua aplicação (copie todos)
- 2. Para usar o servlet controlador (MVC)
  - Defina-o como um <servlet> no seu web.xml
  - Crie um arquivo WEB-INF/struts.config.xml com mapeamentos de ações e outras as configurações
- 3. Para usar cada conjunto de taglibs
  - Defina, no seu web.xml, cada taglib a ser instalada
  - Carregue a taglib em cada página JSP que usá-la

Configuração do controlador no web.xml

- Acrescente no seu web.xml

```
<servlet>
   <servlet-name>action</servlet-name>
   <servlet-class>
      org.apache.struts.action.ActionServlet
   </servlet-class>
   <init-param>
      <param-name>config</param-name>
      <param-value>
         /WEB-INF/struts-config.xml
      </param-value>
   </init-param>
   ... outros init-param ...
</servlet>
```

- Acrescente também os <servlet-mapping> necessários
- Crie e configure as opções de struts-config.xml
- Veja nos docs: /userGuide/building_controller.html
  - use os arquivos de struts-example.war para começar

Configuração das Taglibs

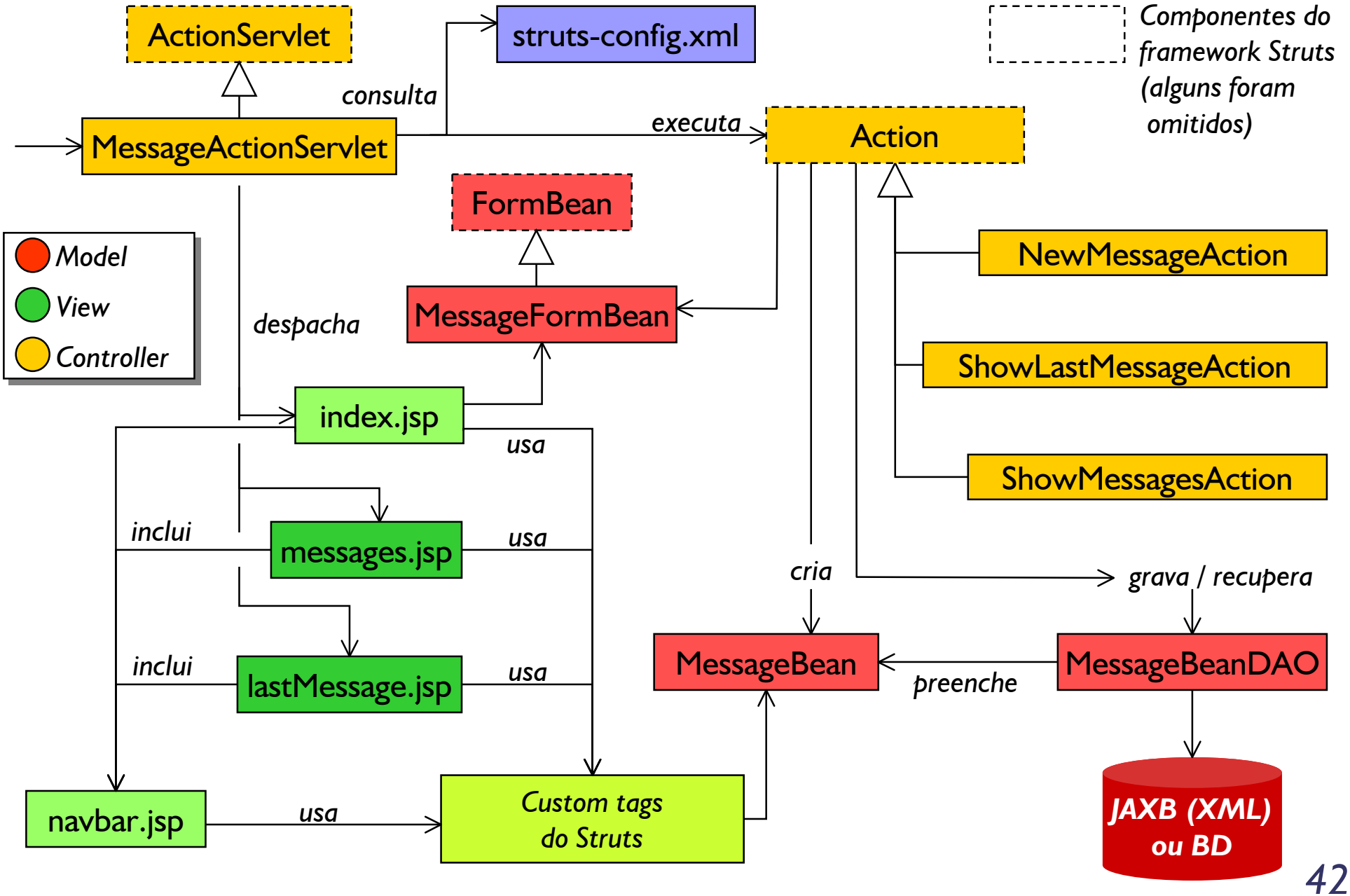
- *Acrescente em **web.xml***
- *Veja detalhes na aplicação struts-example.war ou nos docs:*
  /userGuide/building_controller.html#dd_config_taglib

```
<taglib>
   <taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
   <taglib-location>/WEB-INF/struts-bean.tld
   </taglib-location>
</taglib>
<taglib>
   <taglib-uri>/WEB-INF/struts-form.tld</taglib-uri>
   <taglib-location>/WEB-INF/struts-form.tld
   </taglib-location>
... outros taglibs ...
</taglib>
```

- *Acrescente em cada página JSP*

```
 <@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
 <@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
   ...
```

Implementação de hellojsp com Struts



Mapeamentos (ActionMappings)

- *Veja webinf/struts-config.xml*

```
<struts-config>
  <form-beans>
    <form-bean name="newMessageForm" type="hello.jsp.NewMessageForm" />
  </form-beans>
  <global-forwards>
    <forward name="default" path="/index.jsp" />
  </global-forwards>

  <action-mappings>
    <action path="/newMessage" type="hello.jsp.NewMessageAction"
            validate="true"
            input="/index.jsp" name="newMessageForm" scope="request">
      <forward name="success" path="/showLastMessage.do" />
    </action>
    <action path="/showLastMessage"
            type="hello.jsp.ShowLastMessageAction" scope="request">
      <forward name="success" path="/lastMessage.jsp" />
    </action>
    <action path="/showAllMessages"
            type="hello.jsp.ShowMessagesAction" scope="request">
      <forward name="success" path="/messages.jsp" />
    </action>
  </action-mappings>

  <message-resources parameter="hello.jsp.ApplicationResources" />
</struts-config>
```

FormBeans

- *Form beans permitem simplificar a leitura e validação de dados de formulários*
  - *Devem ser usados em conjunto com custom tags da biblioteca <html:* />*

```
<html:form action="/newMessage" name="newMessageForm"
           type="hello.jsp.NewMessageForm">
    <p>Message: <html:text property="message" />
       <html:submit>Submit</html:submit>
    </p>
</html:form>
```

Configuração em struts-config.xml

```
public class NewMessageForm extends ActionForm {
    private String message = null;
    public String getMessage() { return message; }
    public void setMessage(String message) {
        this.message = message;
    }
    public void reset(...) {
        message = null;
    }
    public ActionErrors validate(...) {...}
}
```

ActionErrors

- *ActionErrors encapsulam erros de operação, validação, exceções, etc.*
  - *Facilitam a formatação e reuso de mensagens de erro.*
- *Exemplo: Método validate() do form bean:*

```
public ActionErrors validate(ActionMapping mapping,
                             HttpServletRequest request) {
  ActionErrors errors = new ActionErrors();
  if ( (message == null) || (message.trim().length() == 0) ) {
    errors.add("message",
               new ActionError("empty.message.error"));
  }
  return errors;
}
```

Nome de campo <input> ao qual o erro se aplica.

Este valor corresponde a uma chave no ResourceBundle

- *Como imprimir:*

```
<html:errors />
```

i18n

- *Informações localizadas podem ser facilmente extraídas de Resource Bundles através de*

```
<bean:message key="chave" />
```

  - *Locale default é usado automaticamente (pode ser reconfigurado)*
- *Exemplo de ResourceBundle*

```
empty.message.error=<tr><td>Mensagem não pode ser
vazia ou conter apenas espaços em branco.</td></tr>
new.message.input.text=Digite a sua mensagem
message.submit.button=Enviar Mensagem
```

hello/jsp/ApplicationResources_pt.properties

- *Configuração em struts-config.xml*

```
<message-resources
      parameter="hello.jsp.ApplicationResources" />
```

- *Exemplo de uso:*

```
<p><bean:message key="new.message.input.text" />
```

Action (Controller / Service To Worker)

- *Controlador processa comandos chamando o método execute de um objeto Action*

```
public class ShowMessagesAction extends Action {

  private String successTarget = "success";
  private String failureTarget = "default";

  public ActionForward execute(ActionMapping mapping,
                               ActionForm form,
                               HttpServletRequest request,
                               HttpServletResponse response)
                        throws IOException, ServletException {
    try {
       MessageBeanDAO dao =
          (MessageBeanDAO)request.getAttribute("dao");
       MessageBean[] beanArray = dao.retrieveAll();
       request.setAttribute("messages", beanArray);
       return (mapping.findForward(successTarget));
     } catch (PersistenceException e) {
       throw new ServletException(e);
     }
  } ...
```

Como rodar o exemplo

- *1. Mude para exemplos/cap08/mvc/hellojsp_3*
- *2. Configure build.properties, depois rode*

  > ant DEPLOY
- *3. Inicie o servidor (Tomcat ou JBoss)*
- *4. Rode os testes do Cactus*

  > ant RUN-TESTS
- *5. Rode a aplicação, acessando a URI*

  http://localhost:porta/hellojsp-struts/
- *6. Digite mensagens e veja resultados. Arquivos são gerados em /tmp/mensagens (ou c:\tmp\mensagens)*

Exercícios

- 1. Escreva um custom tag que
  - Receba como atributos: o nome de uma cor HTML (red, blue, etc.) e um número
  - Receba um texto como corpo

  Sua execução deve imprimir o texto recebido na cor selecionada e no tamanho em pontos recebido.
  - Use CSS:

    `<span style="color: red; font-size: 10pt">texto</span>`
- 2. Instale as aplicações Web mostradas como exemplo
  - MVC
  - Struts
- 3. Aplicação J2EE: veja exercicio_3/README.txt

Fontes

[1] *http://www.computer-programmer.org/articles/struts/*

[2] *Manual do Struts Copie o arquivo struts-documentation.war (se quiser, altere o nome do WAR antes) para o diretório webapps/ to Tomcat 4.0 ou deploy do JBoss.*

[3] *Jim Farley, Java Enterprise in a Nutshell. O'Reilly, 2002. Contém tutorial curso e objetivo sobre JSP e taglibs.*

[4] *Fields/Kolb. Web Development with JavaServer Pages. Manning, 2000. Contém dois capítulos dedicados a taglibs básicos e avançados.*

[5] *Sun J2EE Blueprints.*

helder@argonavis.com.br

argonavis.com.br

J500 - Aplicações Distribuídas com J2EE e JBoss
Revisão 1.4 (março de 2003)