



# Desenvolvimento de Aplicações *Web* com Struts<sup>2</sup>

Aprenda Struts<sup>2</sup> em 15 passos

# Licença para uso e distribuição

Este material está disponível para uso não-comercial e pode ser derivado e/ou distribuído, desde que utilizando uma licença equivalente.



Atribuição-Usó Não-Comercial-Compatilhamento pela mesma licença, versão 2.5

<http://creativecommons.org/licenses/by-nc-sa/2.5/deed.pt>

Você pode copiar, distribuir, exibir e executar a obra, além de criar obras derivadas, sob as seguintes condições: (a) você deve dar crédito ao autor original, da forma especificada pelo autor ou licenciante; (b) você não pode utilizar esta obra com finalidades comerciais; (c) Se você alterar, transformar, ou criar outra obra com base nesta, você somente poderá distribuir a obra resultante sob uma licença idêntica a esta.

# Sobre o autor – Vítor Souza

- Formação:
  - Graduação em Ciência da Computação, mestrado em Engenharia de Software, pela UFES.
- Java:
  - Desenvolvedor Java desde 1999;
  - Especialista em desenvolvimento *Web*;
  - Coordenador do ESJUG.
- Profissional:
  - Professor substituto no DI / UFES;
  - Engenho de Software Consultoria e Desenvolvimento.
- Contato: [vitorsouza@gmail.com](mailto:vitorsouza@gmail.com)

# Sobre o tutorial

- Apresentação do Struts<sup>2</sup>;
- Passo a passo do uso do *framework*;
- Demonstrações;
- Conteúdo:
  - Instalação
  - Configuração básica
  - Ações
  - Resultados
  - Biblioteca de *tags*
  - Freemarker
  - Interceptadores
  - Validação de dados
  - Inversão de controle
  - Internacionalização
  - Enviando arquivos
  - O resultado **stream**
  - Relatórios JasperReports
  - Ações encadeadas
  - Integração com SiteMesh
  - Integração com Spring

# Servidor de páginas estáticas

```
GET /index.html HTTP/1.0  
Host: www.site.com  
[...]
```

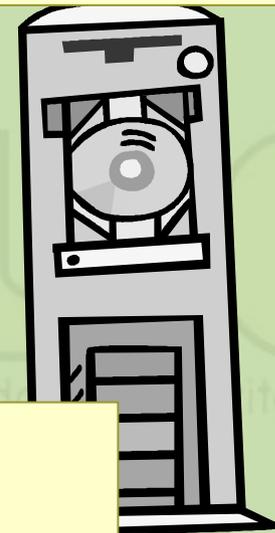
1. Procura arquivo /index.html;
2. Arquivo existe - OK (200);
3. Monta resposta HTTP com conteúdo do arquivo.

Ca



Requisição HTTP

Resposta HTTP - 200 OK



www.site.com

```
HTTP/1.1 200 OK  
Date: Fri, 15 Apr 2005 22:12:30 GMT  
Server: Apache/1.3.26 [...]  
Last-Modified: Wed, 23 Mar 2005 00:43:22 GMT  
Content-Length: 11379  
Content-Type: text/html  
[...]
```

# Servidor de páginas dinâmicas

```
GET /index.asp HTTP/1.0
Host: www.site.com
[...]
```

1. Procura arquivo /index.asp;
2. Interpreta script do arquivo;
3. Arquivo existe e não houve erros - OK (200);
4. Monta resposta HTTP mesclando conteúdo estático e dinâmico (gerado pelo script).



Requis

Resposta HTTP - 200 OK

```
HTTP/1.1 200 OK
Date: Fri, 15 Apr 2005 22:12:30 GMT
Server: Apache/1.3.26 [...]
Last-Modified: Wed, 23 Mar 2005 00:43:22 GMT
Content-Length: 11379
Content-Type: text/html
[...]
```

www.site.com

# Evolução de Java para a Web



# Servlets devem ser registrados

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app >
  <filter> ... </filter>
  <listener> ... </listener>

  <servlet>
    <servlet-name>helloWorld</servlet-name>
    <servlet-class>web.HelloWorldServlet</servlet-
class>
  </servlet>
  <servlet-mapping>
    <servlet-name>helloWorld</servlet-name>
    <url-pattern>/servlets/helloWorld</url-pattern>
  </servlet-mapping>
</web-app>
```

# Não adequado para escrever HTML

```
public class HelloWorldServlet extends HttpServlet {
    public void init() throws ServletException { }
    public void doPost(HttpServletRequest request,
        HttpServletResponse response) throws
        ServletException, IOException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        Date now = new Date(System.currentTimeMillis());

        out.write("<html><head><title>");
        out.write("Hello World</title>");
        out.write("</head><body>" + now);
        out.write("</body></html>");
        out.close();
    }
}
```

# Evolução de Java para a Web



# Menos mapeamento, mais HTML

```
<html>
  <head>
    <title>Hello World</title>
  </head>
  <body>
    <%= new java.util.Date() %>
  </body>
</html>
```

# Inadequado para lógica de negócio

```
<html>[...]
```

```
<%
```

```
Connection conn; PreparedStatement stmt;
```

```
conn = BancoDados.conectar();
```

```
stmt = conn.prepareStatement("SQL");
```

```
ResultSet rs = stmt.executeQuery();
```

```
// [...] Lógica de negócio.
```

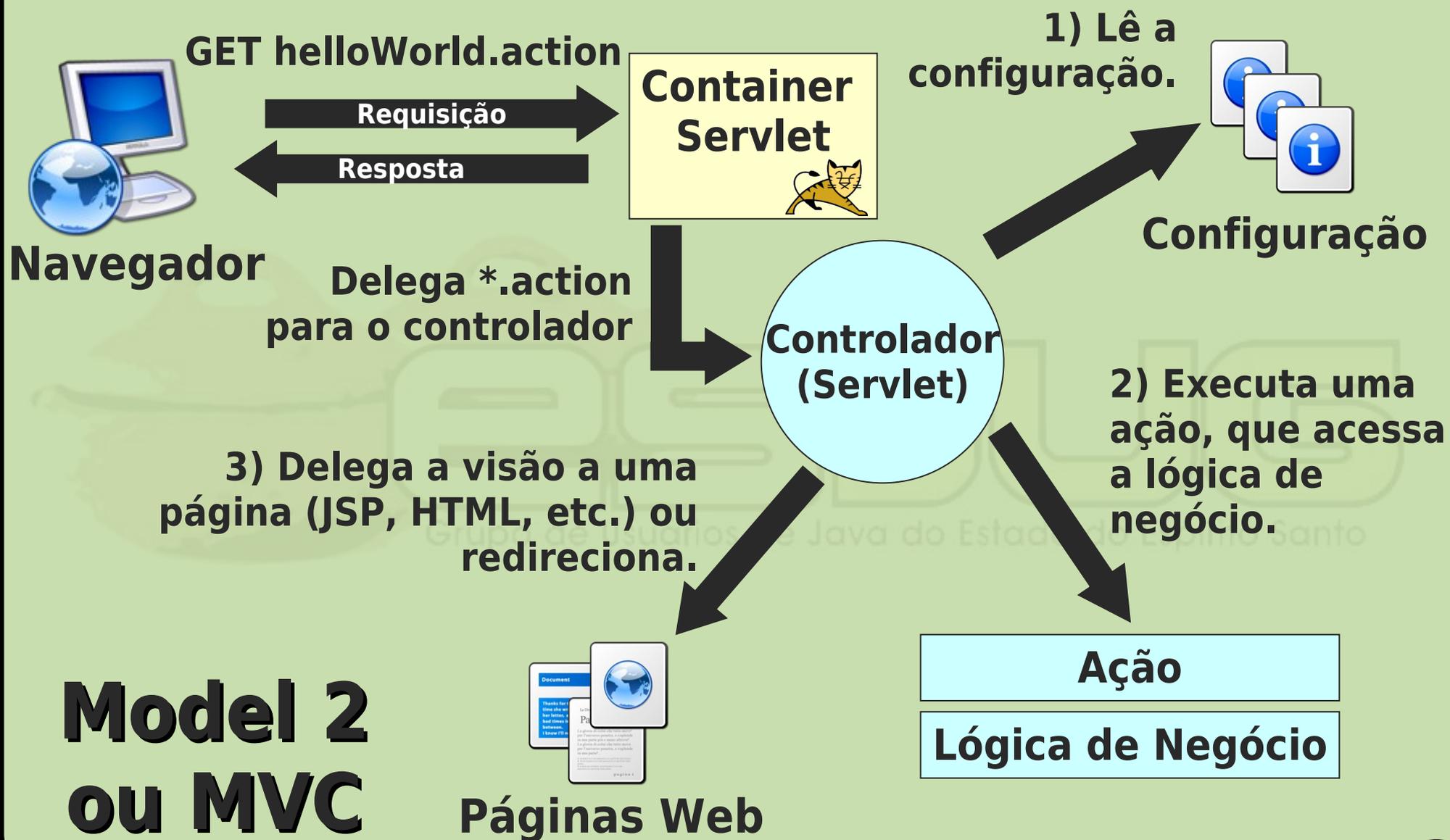
```
stmt = conn.prepareStatement("SQL");
```

```
stmt.executeUpdate();
```

```
%>
```

```
[...]</html>
```

# Evolução de Java para a Web



# Separação de interesses

- Páginas web (JSP, HTML, etc.) cuidam da parte visual;
- Servlet central faz o controle mediante configuração;
- Ações manipulam classes de lógica de negócio (modelo).

Grupo de Usuários de Java do Estado do Espírito Santo

# Nascem os *frameworks* MVC

- O nome correto seria “*Front Controller*”;
- A lógica do MVC é altamente generalizável;
- Podemos listar mais de 50 *frameworks* diferentes:

**Action Framework, Barracuda, Bento, Bishop, Cameleon, Canyamo, Cassandra, Chiba, Cocoon, Dinamica, Dovetail, Echo, Espresso, Folium, Genie, Helma, Jacquard, Jaffa, Japple, JATO, JBanana, Jeenius, JFormular, JPublish, jStatemachine, Jucas, JWAA, JWarp, jZonic, Macaw, Maverick, Melati, Mentawai, Millstone, MyFaces, Nacho, Niggle, OpenEmcee, OXF, RIFE, Scope, Shocks, Smile, SOFIA, Spring MVC, Struts, Tapestry, TeaServlet, Turbine, Verge, VRaptor, Warfare, WebOnSwing, WebWork, wingS, Xoplon**

Fonte: <http://www.manageability.org/blog/stuff/how-many-java-web-frameworks>

# Destques

- Struts 1:
  - Padrão “de facto”, mais antigo e usado no mercado, também mais odiado.
- VRaptor2:
  - Projeto brasileiro, uso de anotações para diminuir a quantidade de configuração XML.
- Spring MVC:
  - Parte do Spring Framework.
- WebWork / Struts 2:
  - Junção dos projetos Struts com WebWork, bastante aceito pela comunidade.

# JavaServer Faces

- JSR 127 – padrão oficial (27/05/2004);
  - Várias implementações;
  - Garantia de continuidade.
- Orientado a componentes e eventos: abordagem do Swing na *Web*;
- “Concorrente indireto” dos *frameworks* MVC, por se tratar de uma outra abordagem;
- JBoss Seam.

# Struts<sup>2</sup>

- *Framework* MVC para *Web*, junção dos projetos Struts (Apache) e WebWork (Opensymphony);
- Baseado no XWork2, implementação do padrão de projeto *Command*, independente da *Web*;
- Projeto da Apache Software Foundation.



# Informações gerais

- Website:
  - <http://struts.apache.org/>
  - Clique em Documentation / Struts 2.x.
- Versão atual (agosto/2007): 2.0.9;
- Licença:
  - Apache Software License.

Grupo de Usuários de Java do Estado do Espírito Santo

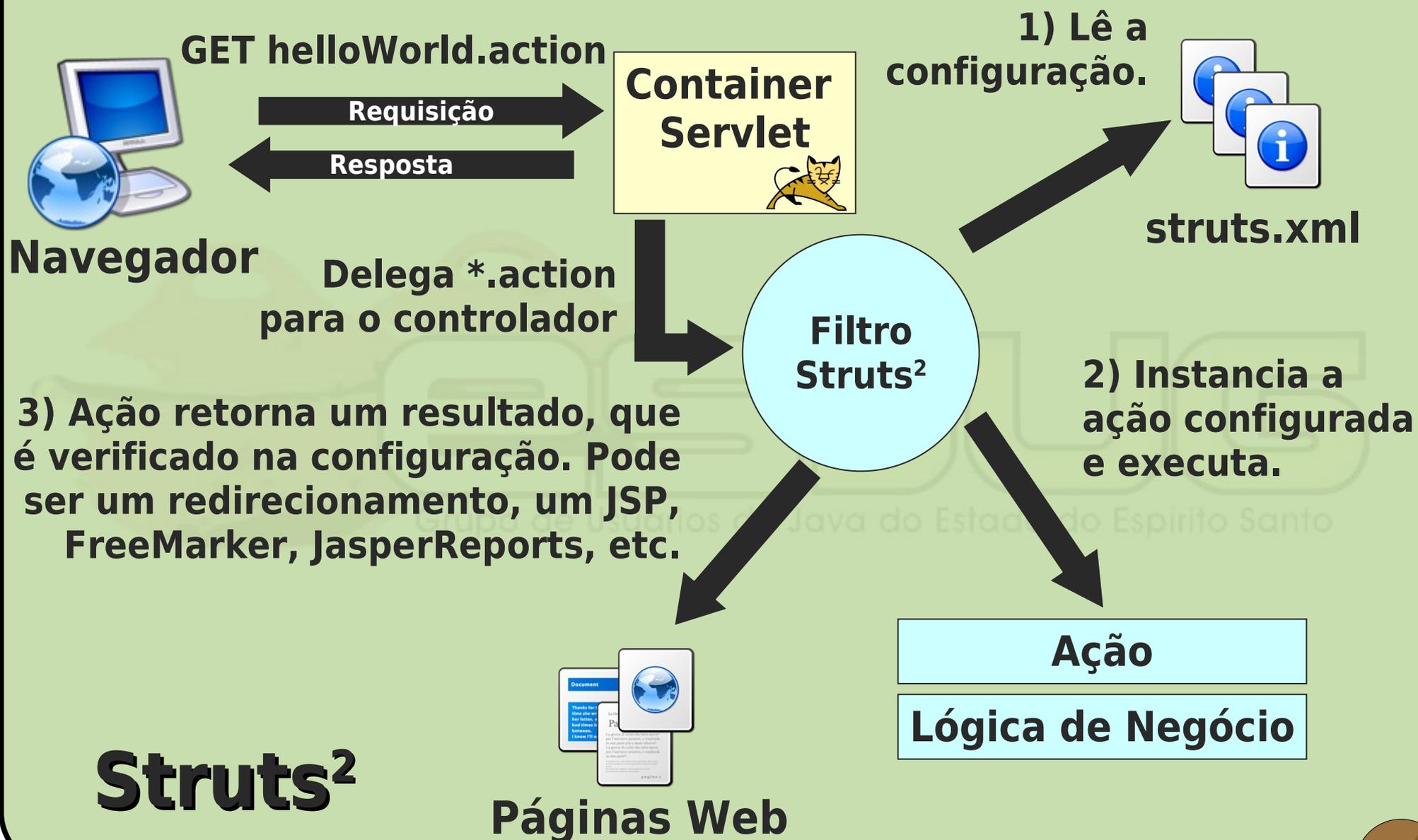
# Funcionalidades

- Dispatcher (controlador) que recebe e delega requisições;
- Tipos de resultado, com suporte a diversas tecnologias de visão:
  - JSP;
  - FreeMarker / Velocity;
  - JasperReports;
  - XML.
- Tag Library (e macros Velocity);
- Configuração flexível;

# Funcionalidades

- Conversão automática de tipos;
- Framework de validação automática;
- Expression Language – OGNL ([www.ognl.org](http://www ognl.org));
- Templates para geração de componentes;
- Extensível por meio de interceptadores;
- Suporte a internacionalização (i18n);
- Suporte a pacotes e espaços de nomes;
- Fácil integração com outros *frameworks*;
- Em especial, integração com o Spring.

# Evolução de Java para a Web



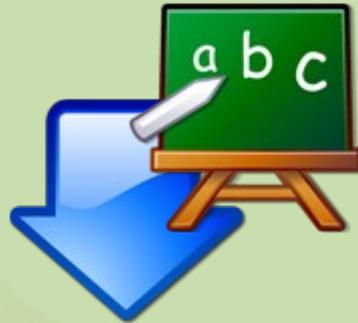
# Vantagens e desvantagens

- Torna o desenvolvimento *Web* muito mais simples, com código mais organizado;
- Grande comunidade de usuários:
  - Projeto da Apache;
  - Excelente suporte voluntário pelo fórum.
- Bem documentado;
- Atualizado em relação às tecnologias *Web*;
- Não é implementação de um padrão do JCP (ex.: JSF).

# Integração com Spring Framework

- Spring é utilizado como *framework* de injeção de dependências;
- Por causa disso, falaremos um pouco sobre Spring;
- No entanto, para conhecer bem o Spring é necessário um curso inteiro sobre ele...

The Spring logo features the word "Spring" in a black serif font. A small green leaf with a brown stem is positioned above the letter 'i'.



## Primeiros passos: instalação

**Struts<sup>2</sup>**

# Download

- <http://struts.apache.org/download.cgi#struts209>
- Full Distribution: struts-2.0.9-all.zip (86mb);
- Conteúdo:
  - apps: aplicações prontas de exemplo (showcase);
  - docs: documentação;
  - lib: o *framework* e suas dependências;
  - src: código-fonte.

# Instalação

- “Instalar” um *framework Web* é saber:
  - Quais as bibliotecas necessárias para escrever o código (inclusão no CLASSPATH);
  - Quais as bibliotecas necessárias para rodar uma aplicação *Web* que utilize o *framework* (inclusão em WEB-INF/lib);
  - Como configurar a aplicação *Web* para usar o *framework* (passo 2).

# Preparação da infra-estrutura

- Servidor *Web*: Apache Tomcat 5.5;
- IDE: Eclipse Europa com ferramentas Java EE;
- Processo:
  - Instalação de ambas as ferramentas;
  - Configuração de um servidor Tomcat dentro do Eclipse (Preferences / Server / Installed Runtimes);
  - Criação de um “Dynamic Web Project”;
  - Adição do projeto no servidor dentro do Eclipse;
  - Execução e depuração dentro da IDE.

# Bibliotecas necessárias

- Copiadas do diretório lib do pacote Struts<sup>2</sup>:
  - Núcleo: struts2-core, xwork, freemarker, commons-logging, ognl;
  - Opcionalmente: dwr, sitemesh, spring, etc.
- Pode-se usar um gerenciador de dependências:
  - Ex.: Ivy (atualmente na Incubadora da Apache);
  - Repositório LabES: <http://labes.inf.ufes.br/ivy-rep/>.

# ivy.xml

```
<ivy-module version="1.0">  
  <info organisation="labes-ufes"  
module="portallabes" />  
  
  <dependencies>  
  
    <dependency org="apache" name="struts2-  
core" rev="2.0" />  
  
  </dependencies>  
</ivy-module>
```

## Alvo no build.xml

```
<target name="deps">  
  <echo message="Obtendo dependencias..." />  
  <ivy:configure  
    url="http://labes.inf.ufes.br/ivy-  
      rep/ivyconf.xml" />  
  <ivy:retrieve />  
  
</target>
```



# Configuração básica e primeira ação

Grupo de Usuários de Java do Estado do Espírito Santo

# Struts<sup>2</sup>

# Aplicação em branco

- Pacote Struts<sup>2</sup> / apps / struts2-blank: contém a configuração mínima;
- Utilizaremos nossas próprias configurações mínimas:
  - applicationContext: configuração do Spring;
  - web.xml: configuração da WebApp, onde indicaremos o uso do Struts<sup>2</sup>;
  - log4j.properties;
  - struts.properties;
  - struts.xml.

# Estrutura da aplicação em branco

- WEB-INF

- classes

- log4j.properties
    - struts.properties
    - struts.xml

} Raiz do src.

- pages

- home.jsp

} Páginas e modelos ficam escondidos.

- applicationContext.xml

- web.xml

} Arquivos de configuração da aplicação *Web*.

- index.jsp

# struts.properties

```
# Localização.  
struts.locale = pt_BR  
  
# Modo de desenvolvimento: recarrega  
# alterações e reporta erros que poderiam ser  
# ignorados.  
struts.devMode = true  
  
# Tamanho máximo para upload (10 MB).  
struts.multipart.maxSize = 10485760  
  
# Configurar o Spring como DI Container.  
struts.objectFactory = spring
```

# struts.xml

```
<!DOCTYPE struts [...]>
<struts>
  <!-- Inclui as configurações default. -->
  <include file="struts-default.xml" />

  <!-- Define um pacote de classes. -->
  <package name="default" extends="struts-default">
    <!-- Ação padrão do pacote. -->
    <default-action-ref name="home" />

    <!-- Definição da ação. -->
    <action name="home">
      <result>/WEB-INF/pages/home.jsp</result>
    </action>

    <!-- Adicione suas ações aqui. -->
  </package>
</struts>
```

```
<!-- Configuração da WebApp. -->
<web-app [...]>
  <!-- Nome da WebApp. -->
  <display-name>TutorialStruts</display-name>

  <!-- Filtros. -->
  <filter>
    <filter-name>springFilter</filter-name>
    <filter-class>
      org.springframework.web.filter.
      RequestContextFilter
    </filter-class>
  </filter>
```

```
<filter>
  <filter-name>strutsFilter</filter-name>
  <filter-class>
    org.apache.struts2.dispatcher
    .FilterDispatcher
  </filter-class>
</filter>

<!-- Mapeamento dos filtros. -->
<filter-mapping>
  <filter-name>springFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

# web.xml

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

<!-- Listeners. -->
<listener>
  <listener-class>
    org.springframework.web.context
    .ContextLoaderListener
  </listener-class>
</listener>

</web-app>
```

# index.jsp

```
<% response.sendRedirect(request  
.getContextPath() + "/home.action"); %>
```

# Primeira ação: Hello Struts<sup>2</sup>!

- Responder à requisição a:
  - `helloStruts2.action`;
- Construir uma mensagem:
  - “Hello, Struts2!”;
- Disponibilizar esta informação para a página (camada de visão) que será exibida ao visitante.

Grupo de Usuários de Java do Estado do Espírito Santo

# Passos para construção de uma ação

- 1º: Criar a classe de ação (se já não existe);
- 2º: Adicionar parâmetros de I/O na classe;
- 3º: Implementar o método que responde pelas evocações da ação;
- 4º: Configura a ação no *framework* (xwork.xml);
- 5º: Chamar a ação por sua URL.

Grupo de Usuários de Java do Estado do Espírito Santo

# Classe de Ação

- Ações devem implementar:
  - `com.opensymphony.xwork2.Action;`
- Herdar da classe de suporte é melhor:
  - `com.opensymphony.xwork2.ActionSupport;`
  - Vários atalhos disponíveis;
- Método executado quando a ação é chamada:

```
public String execute() throws Exception { }
```

# HelloStruts2Action

```
package
net.java.dev.esjug.tutorialstruts2.controller;

import java.util.Date;
import com.opensymphony.xwork2.ActionSupport;

public class HelloStruts2Action
extends ActionSupport {
    private String mensagem = "Hello, Struts2! ("
+ new Date() + ")";

    public String getMensagem() {
        return mensagem;
    }
}
```

# Configuração no struts.xml

```
<package name="default" [...]>
  [...]

  <action name="helloStruts2" class="net.java.
    dev.esjug.tutorialstruts2.
    controller.HelloStruts2Action">

    <result>/WEB-INF/pages/hello.jsp</result>
  </action>

</package>
```

# hello.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Tutorial Struts2</title>
</head>
<body>

<h1>Tutorial Struts2</h1>

<p><s:property value="mensagem" /></p>

</body>
</html>
```



Construir nossa primeira  
ação: HelloStruts2



# Entendendo melhor as ações

**Struts<sup>2</sup>**

# Ação

- Unidade básica de execução (é o C do MVC);
- Responde à uma requisição determinada em `struts.xml`;
- Execução de uma ação:
  - 1º: É feita uma requisição pela URL da ação;
  - 2º: O *framework* verifica qual a classe que implementa a ação daquela URL e cria uma instância dela;
  - 3º: O *framework* verifica qual método implementa a ação e chama-o, aguardando seu resultado;
  - 4º: A partir do resultado, o *framework* verifica o que deve ser feito (geralmente processar uma página).

# Troca de dados

- Controle → Visão:
  - Métodos `getXYZ()` na classe de ação;
  - Tags `<s:property value="xyz" />` no JSP;
  - Hello Struts2!
- Visão → Controle:
  - Métodos `setXYZ()` na classe de ação;
  - Envio de dados por formulários via POST ou GET (usa interceptadores – veremos adiante);
  - Há *tags* para construção do formulário (v. adiante).

## Pilha de Valores (*Value Stack*)

- A *Value Stack* é onde o Struts<sup>2</sup> coloca os dados da ação;
- A instância da classe de ação fica na *Value Stack* e a tag `<s:property />` obtém dados lá;
- Outros objetos, como a requisição e a resposta HTTP, também estão disponíveis;
- Referências para objetos na pilha são escritas em OGNL;
- Outras *tags* manipulam esta pilha. Não entraremos em detalhes.

# Conversão automática

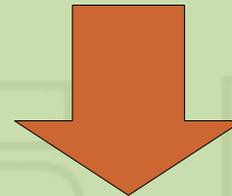
- Struts<sup>2</sup> converte automaticamente de String para o tipo apropriado e vice-versa;
- Basta declarar a propriedade como do tipo que se quer converter: Long, Float, Double, Boolean, Date, ...;
- Obedecem configuração de *Locale* da aplicação (se não for configurado, usa a do SO);
- É possível implementar seu próprio conversor para tipos específicos.

# Troca de dados dirigida a modelo

```
<s:textfield name="func.nome" />
<s:textfield name="func.dataNasc" />
<s:textfield name="func.salario" />
<s:textfield name="func.tempoEmpresa" />
<s:checkbox name="func.gerente" />
```

## Funcionario

- nome : String
- dataNasc : Date
- salario : Float
- tempoEmpresa : Integer
- gerente : Boolean



```
public class AcaoCadastroFuncionario {
    private Funcionario func;

    public Funcionario getFunc() { ... }

    public void setFunc(Funcionario func) { ... }
}
```

# Múltiplos métodos de ação

- Uma classe de ação pode responder por várias URLs, bastando ter diferentes métodos de ação:

```
<action name="acao" class="pacote.ClasseAcao">  
  <result>pagina1.jsp</result>  
</action>
```

```
<action name="acao02" class="pacote.ClasseAcao"  
          method="executePasso02">
```

```
  <result>pagina2.jsp</result>  
</action>
```

Chama o método `executePasso02()`  
ao invés de `execute()`.



# Múltiplos métodos pela URL

- Uma mesma ação pode ser chamada com métodos diferentes;
- Basta colocar !método no meio da URL:

```
<action name="cadastro" class="pacote.AcaoCad">  
  <result name="input">form.jsp</result>  
  <result>resultado.jsp</result>  
</action>
```

```
<a href="cadastro!input.action">Cadastrar</a>
```

- Struts<sup>2</sup> executará o método input ( ) na ação;
- O método deve ser público, sem parâmetros e retornar String (assim como execute ( )).

# Espaços de nome

- Pacotes podem ter *namespaces*;
- Pode haver ações com mesmo nome em espaços de nome diferentes.

```
<package name="pac1" extends="default"
          namespace="/pac1">
  <action name="acao"
          class="pacote.ClasseAcao">
    <result>pagina.jsp</result>
  </action>
</package>
```

Ações do pacote responderão à requisições  
"/pac1/\*.action". Ex.: "/pac1/acao.action".

# Demonstração



Desenvolver uma ação que receba o nome e a data de nascimento de uma pessoa, calcule sua idade e responda: “Caro(a) Fulano, você tem X ano(s).”



## Entendendo melhor os resultados

# Struts<sup>2</sup>

# Resultado

- Uma ação retorna um resultado (String) que determina (no `struts.xml`) o que Struts<sup>2</sup> deve fazer:

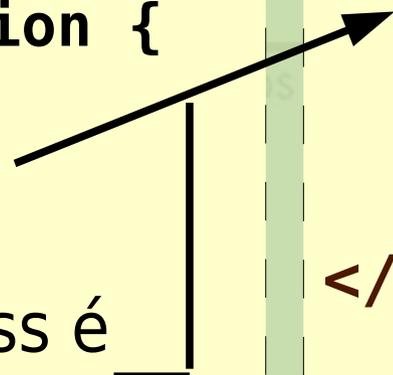
```
class HelloStruts2Action
extends ActionSupport {

    public String execute()
        throws Exception {

        return SUCCESS;

    }
}
```

success é  
o default



```
<xwork>
[... ]
<action [... ]>
    <result>
        helloStruts2.jsp
    </result>
</action>
</xwork>
```

# Tipos de resultado

- São classes que implementam ações que o Struts<sup>2</sup> realizará;
- Para criar seu próprio, implemente com `opensymphony.xwork2.Result`;
- Existem vários tipos prontos;
- A interface `Action` já define 5 constantes de resultado: `SUCCESS`, `NONE`, `ERROR`, `INPUT` e `LOGIN`.

# Tipos de resultado existentes

dispatcher	Despacha a requisição para uma URL.
redirect	Redireciona o navegador para uma URL.
chain	Encadeia uma ação em outra.
velocity	Processa um template Velocity e retorna o resultado.
freemarker	Processa um template FreeMarker e retorna o resultado.
xslt	Transforma um XML usando XSLT e retorna o resultado.
jasper	Retorna um relatório do JasperReports.
httpheader	Adiciona informações vindas da ação ao header HTTP.

## struts-default.xml

- Em nosso `struts.xml`, importamos o arquivo `struts-default.xml`;
- Um dos diversos motivos é porque nele se encontram as declarações dos tipos de resultado pré-definidos pelo Struts<sup>2</sup>;
- Podemos usá-lo como base para aprender como definir nossos próprios tipos de resultado, se houver necessidade.

# Configuração

- Um resultado pode ser local ou global;
- O resultado *default* é *success*;
- O tipo de resultado *default* é *dispatcher*.

```
<!-- Resultado success, tipo dispatcher. -->  
<result>home.jsp</result>
```

```
<!-- Resultado error, tipo dispatcher. -->  
<result name="error">erro.jsp</result>
```

```
<!-- Resultado logout, tipo redirect. -->  
<result name="logout" type="redirect">  
    index.html  
</result>
```

# Demonstração



Construir um formulário de *login*. Se a senha estiver incorreta, retornar para o formulário. Se estiver OK, verificar o tipo de funcionário (desenvolvedor, gerente ou diretor) e direcionar para a página adequada.



## A biblioteca de *tags* do Struts<sup>2</sup>

Grupo de Usuários de Java do Estado do Espírito Santo

# Struts<sup>2</sup>

# Tags JSP

- Classes que são evocadas por *tags* específicas na página JSP:
  - Manipulam seus parâmetros e conteúdo;
  - Imprimem o resultado na página final.

```
<%@ taglib uri="/WEB-INF/tld/c.tld" prefix="c" %>
<c:if test='${param.p == "someValue"}'>
    'p' é igual a "someValue".
</c:if>
<c:else>
    'p' não é igual a "someValue".
</c:else>
```

# Tags do Struts<sup>2</sup>

- *General tags:*
  - Controle de fluxo;
  - Manipulação de dados na *value stack*;
  - Internacionalização.
- *HTML tags:*
  - Exibição de dados nas páginas;
  - Montagem de formulários HTML;
  - Montagem de outras estruturas HTML.

# Templates e temas

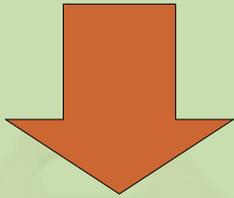
- Componentes HTML são baseados em *templates*;
- Conjunto de *templates* formam um tema;
- Struts<sup>2</sup> vem com quatro temas:
  - simple: o mais simples possível, bom para extensão;
  - xhtml: formulário em tabelas (padrão);
  - css\_xhtml: reimplementação de xhtml com CSS;
  - ajax: baseado no xhtml, com recursos AJAX.
- *Templates* são escritos em FreeMarker (recomendado), Velocity ou JSP.

# Personalizando

- Você pode escolher outro tema:
  - Para todo o formulário ou *tag* por *tag*;
  - Propriedade `theme="meutema"`.
- Você pode criar seus próprios temas:
  - `/template/meutema` na *WebApp* ou no *Classpath*.
- Você pode sobrescrever um componente:
  - Cópia `/template/xhtmll/componente.ftl` para seu projeto e altera o que quiser.
- Você pode criar um componente novo:
  - *Tag* genérica `<s:component />`.

# Transformação

```
<s:textfield label="Login" name="login" />
```



struts2-core.jar :: template/xhtml/text.ftl

```
<tr>
  <td class="tdLabel"><label
for="login.action_login"
class="label">Login:</label></td>
  <td>
<input type="text" name="login" value=""
id="login.action_login"/>
</td>
</tr>
```

# Avaliação de expressões

- Sempre que queremos obter dados da ação e usá-los nas *tags*, colocamos entre `%{ }`;
- No exemplo abaixo, `getLabelLogin()` é chamado e seu resultado é usado como parâmetro `label`:

```
<s:textfield label="%{labelLogin}" name="login" />
```

- O nome dos campos do formulário pode fazer referência a alguma propriedade da ação;
  - Neste caso, o campo é preenchido com seu valor.

## Outras Tags

<code>&lt;s:text /&gt;</code> , <code>&lt;s:i18n /&gt;</code>	Internacionalização (passo 10)
<code>&lt;s:property /&gt;</code>	Mostra dados vindos da ação ( <i>value stack</i> )
<code>&lt;s:push /&gt;</code>	Coloca valores na <i>value stack</i>
<code>&lt;s:set /&gt;</code>	Armazena valores em variáveis
<code>&lt;s:url /&gt;</code>	Constrói URLs nas codificações adequadas
<code>&lt;s:action /&gt;</code>	Chama uma ação de dentro da página.
<code>&lt;s:bean /&gt;</code>	Instancia um objeto qualquer.

# Outras Tags

<code>&lt;s:include /&gt;</code>	Inclui uma outra página ou ação
<code>&lt;s:if /&gt;</code> , <code>&lt;s:elseif&gt;</code> , <code>&lt;s:else&gt;</code>	Controle de fluxo
<code>&lt;s:iterator&gt;</code>	Itera sobre alguma coleção
<code>&lt;s:generator /&gt;</code> , <code>&lt;s:append /&gt;</code> , <code>&lt;s:subset /&gt;</code> , <code>&lt;s:merge /&gt;</code> , <code>&lt;s:sort /&gt;</code>	Outras tags de iteração (criar um iterador, juntar iteradores, ordenar o conjunto, etc.)

# Demonstração



Demonstrar uma página  
com diversas *tags*.

<FreeMarker>

# Trocando JSP por FreeMarker

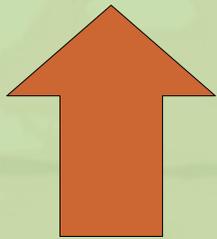
**Struts<sup>2</sup>**

# O que é o FreeMarker?

- *Template Engine*:
  - Separa os dados da forma que são apresentados;
  - Idéia do MVC: programadores focados na lógica, *designers* focados no visual;
  - Não exclusivo para páginas *Web*.
- Site: [www.freemarker.org](http://www.freemarker.org);
- Licença similar à BSD (*open source*).

# Template Engine

**Documento  
Mesclado**



**Modelo de dados**

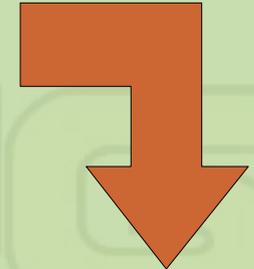
**cliente** →

**Cliente**

nome  
contato  
[...]

**Coleção**  
itens[]

**produtosEmPromocao**



**Olá `{cliente.nome}`!**

**Confira as promoções:**

**`<#list produtosEmPromocao as produto>`**

**`- {produto.nome}: R$ {produto.preco}!!!`**

**`</#list>`**

# FreeMarker Template Language (FTL)

- Texto é impresso *ipsis-litteris*;
- Interpolações representam um valor que é calculado e impresso - `{ }`;
- *Tags* FTL (diretivas) são instruções processadas. Seu conteúdo pode gerar impressão de dados;
- Comentários são ignorados.

Grupo de Usuários de Java do Estado do Espírito Santo

# Exemplo em HTML

```
<html>
<head><title>Produtos</title></head>
<body>
  <!-- Cumprimenta o usuário pelo nome. -->
  <h1>Olá ${usuario}, seja bem-vindo(a)!</h1>

  <!-- Lista os produtos. -->
  <p>Produtos disponíveis:</p>
  <ul>
    <#list produtos as p>
      <li>${p.nome} – R$ ${p.preco}</li>
    </#list>
  </ul>
</body>
</html>
```

# Regras gerais

- *Case-sensitive*;
- Interpolações só ocorrem no meio de texto ou em literais *string*;
- *Tags* FTL não podem ser colocadas dentro da definição de outra *tag*;
- FreeMarker ignora espaço em branco supérfluo (exceto no texto).

# Diretivas

- Tags FTL são iguais a XML:
  - `<#diretiva chave="valor" [...] > ... </#diretiva>;`
  - `<#diretiva chave="valor" [...] />;`
  - Devem ser aninhadas direito.
- Diretivas definidas pelo usuário:  
`<@nomeDiretiva [...] />;`
- Pode ser configurado para não usar o “#”, apesar de não recomendado.

# Diretivas do FreeMarker

- Atribuição de variáveis:
  - `<#assign cliente = "Fulano" />`
- Condicionais:
  - `<#if c1> ... <#elseif c2> ...<#else> ... </#if>`
- Loops:
  - `<#list colecao as item> ${item} </#list>`

Grupo de Usuários de Java do Estado do Espírito Santo

# Diretivas criadas pelo usuário

- Definição de macros:
  - `<#macro ola p>Olá ${p}!</#macro>`
- Uso de macros:
  - `<@ola p="João" />`



# Expressões

- Literais:
  - Strings: "foo", 'bar', "Usa \"apas\"";
  - Números: 100, -123.45, 0.11;
  - Booleanos: true, false;
  - Sequências: ["foo", "bar", 123.45], 1..10;
  - Mapas (*hash*):  
{ "nome": "teclado", "preco": 19.9 }.

# Expressões

- Recuperando variáveis:
  - Simples: `usuario`;
  - Em um mapa: `usuario.nome`, `usuario["nome"]`, `usuario[prop]`;
  - Em uma sequência: `produtos[5]`;
  - Variáveis especiais (definidas pelo *engine*, uso avançado): `.nome`.
- Operações com String:
  - Concatenação: `"Olá ${usuario}"`, `"Free" + "Marker"`;
  - Substrings: `nome[0..2]`.

# Expressões

- Operações com seqüências:
  - Concatenação: `usuarios + ["guest"];`
  - Subseqüência: `produtos [10..15],`  
`produtos [5..].`
- Operações com *hashs*:
  - Concatenação:  
`senhas + {"fulano" : "segredo"}.`
- Cálculos aritméticos:
  - $(x * 1.5 + 10) / 2 - y \% 100.$

# Expressões

- Comparação:
  - `x == y`, `x != y`, `x < y`, `x >= y`, `x gt y`, etc.
- Operações lógicas:
  - `(! registrado) && (efetuandoCompra || enviandoMensagem)`.
- *Built-ins* (funções internas):
  - `nome?upper_case`, `texto?html`, `valor?exists`, `dt?date("dd/MM/yyyy")`.
- Chamada de método:
  - `repetir("FreeMarker", 3)`.

# Uso de expressões

- Em interpolações (imprime o resultado no documento):
  - `${loc.datas[4]?date("dd/MM/yyyy")}`.
- Em diretivas:
  - `<#if (x > 10) && (! registrado)>`  
    `...`  
    `</#if>`.

Grupo de Usuários de Java do Estado do Espírito Santo

# Outras funcionalidades

- switch-case;
- break;
- include/import;
- function;
- escape;
- etc.



Confira a lista  
de *built-ins*!

<http://www.freemarker.org/docs/index.html>

[http://www.freemarker.org/docs/ref\\_builtins.html](http://www.freemarker.org/docs/ref_builtins.html)

# Integração com Struts<sup>2</sup>

- Escreva o *template* em FTL;
- Use o resultado freemarker;
- Dados disponíveis aos *templates*:
  - `{ req }`: `HttpServletRequest` atual;
  - `{ res }`: `HttpServletResponse` atual;
  - `{ stack }`: a pilha de valores;
  - `{ ognl }`: instância de `OgnlTool`;
  - `{ action }`: ação que acabou de executar;
  - `{ propriedade }`: chama `getPropriedade()`.

# Uso de *tags* JSP

```
<html>
<body>

<@s.form method="post" action="acao">
  <@s.select label="L" name="lst" list="%{l}" />
  <@s.textfield label="Texto" name="texto" />
  <@s.submit value="Enviar" />
</@s.form>

</body>
</html>
```

## Uso de *tags* JSP

- É necessário adicionar um Servlet no `web.xml`:

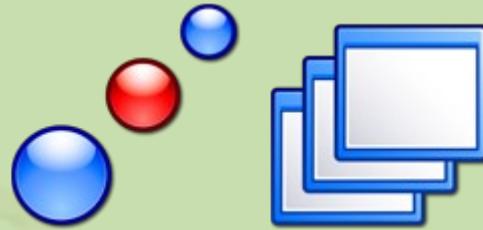
```
<servlet>  
  <servlet-name>JspSupportServlet</servlet-name>  
  <servlet-class>  
    org.apache.struts2.views.JspSupportServlet  
  </servlet-class>  
  <load-on-startup>1</load-on-startup>  
</servlet>
```

# Demonstração



Repetir a demonstração do passo 03 (cálculo da idade), agora utilizando FreeMarker.

Utilizaremos FreeMarker daqui em diante no Tutorial.



## Nos bastidores, interceptadores

**Struts<sup>2</sup>**

# O que são interceptadores?

- Idéia semelhante aos padrões de projeto *Decorator* e *Chain of Responsibility* do GoF;
- Interceptam uma requisição e podem executar código antes e/ou depois.



# AOP com interceptadores

Mesmo trecho de código espalhado por várias classes

**AplicacaoA.metodoNegocio1()**

Iniciar a transação

Acessar banco de dados 1

Commit | Rollback

**AplicacaoA.metodoNegocio2()**

Iniciar a transação

Acessar banco de dados 2

Commit | Rollback

**AplicacaoB.metodoNegocio3()**

Iniciar a transação

Acessar banco de dados 3

Commit | Rollback

# AOP com interceptadores

**AplicacaoA.metodoNegocio1()**

**Acessar banco de dados 1**

**AplicacaoA.metodoNegocio2()**

**Acessar banco de dados 2**

**AplicacaoB.metodoNegocio3()**

**Acessar banco de dados 3**

**Aspectos são separados e implementados uma única vez.**

**Iniciar a transação**

**Commit | Rollback**

# AOP com interceptadores

Interceptador cuida para que aspectos sejam executados.

Iniciar a transação

Commit | Rollback

AOP  
Interceptor

Cliente

AplicacaoA

metodoNegocio1()

# Alguns interceptadores do Struts<sup>2</sup>

alias	Converte nomes de parâmetros de uma ação para outra.
chain	Transporta valores de parâmetros de uma ação para outra, quando são encadeadas.
conversionError	Anexa erros de conversão aos campos dos formulários (validação, passo 8).
createSession	Cria uma sessão HTTP automaticamente.
execAndWait	Executa uma ação em <i>background</i> e envia o usuário para uma página intermediária.

# Alguns interceptadores do Struts<sup>2</sup>

i18n	Altera dinamicamente o <i>locale</i> e lembra dele a cada requisição.
logger	Efetua o <i>log</i> do início e fim da ação.
model-driven	Se a ação implementa <b>ModelDriven</b> , facilita o acesso a elementos do modelo.
params	Chama métodos <b>setXyz()</b> na ação para atribuir valores da requisição.
prepare	Se a ação implementa <b>Preparable</b> , chama seu método <b>prepare()</b> .
scope	Armazena e recupera valores da sessão.
servlet-config	Dá acesso ao <b>HttpServletRequest</b> e ao <b>HttpServletResponse</b> (cuidado!).

# Alguns interceptadores do Struts<sup>2</sup>

static-params	Atribui parâmetros configurados em <b>struts.xml</b> à ação.
timer	Cronometra o tempo de execução da ação.
token	Previne submissão duplicada de formulários.
token-session	O mesmo que acima, porém usando a sessão.
validation	Faz validação de formulários (passo 8).
workflow	Complementa validation, retornando <b>INPUT</b> no caso de erros de validação.

# Configuração

- Podem ser organizados em pilhas (ordem é importante!);
- Cada ação pode definir seus próprios interceptadores;
- Pacotes podem definir o *default*;
- `struts-default.xml`:
  - Interceptores configurados;
  - Várias pilhas prontas.

# Exemplo de configuração

```
<struts>
  <include file="struts-default.xml" />

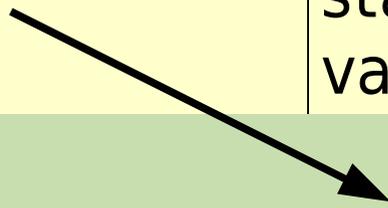
  <package name="default" extends="struts-default">
    <!-- Define a pilha padrão do pacote. -->
    <default-interceptor-ref name="basicStack" />

    <action name="nomeAcao" class="pac.ClasseAcao">
      <!-- Define outra pilha para esta ação. -->
      <interceptor-ref name="defaultStack" />
      <result>pagina.jsp</result>
    </action>

  </package>
</struts>
```

# Pilhas do struts-default

basicStack	exception, servlet-config, prepare, static-params, params, conversionError
validationWorkflowStack	basicStack, validation, workflow
fileUploadStack	fileUpload, basicStack
...	...
defaultStack	exception, alias, servlet-config, prepare, i18n, chain, model-driven, fileUpload, static-params, params, conversionError, validation, workflow



Ao montar pilhas customizadas, observe a ordem definida no defaultStack!

# Criando seu próprio interceptador

- 1º: Escrever uma classe que implemente `com.opensymphony.xwork2.interceptor.Interceptor`
- 2º: Registrar o interceptador no `struts.xml`;
- 3º: Opcionalmente, colocá-lo em alguma pilha;
- 4º: Configurar alguma classe que o utilize ou utilize a pilha que o contém.

Grupo de Usuários de Java do Estado do Espírito Santo

# Demonstração



Criar um interceptador que determine, pela hora do dia, se o visitante deve receber “Bom dia”, “Boa tarde” ou “Boa noite” e colocar este cumprimento na sessão.



# Validando dados automaticamente

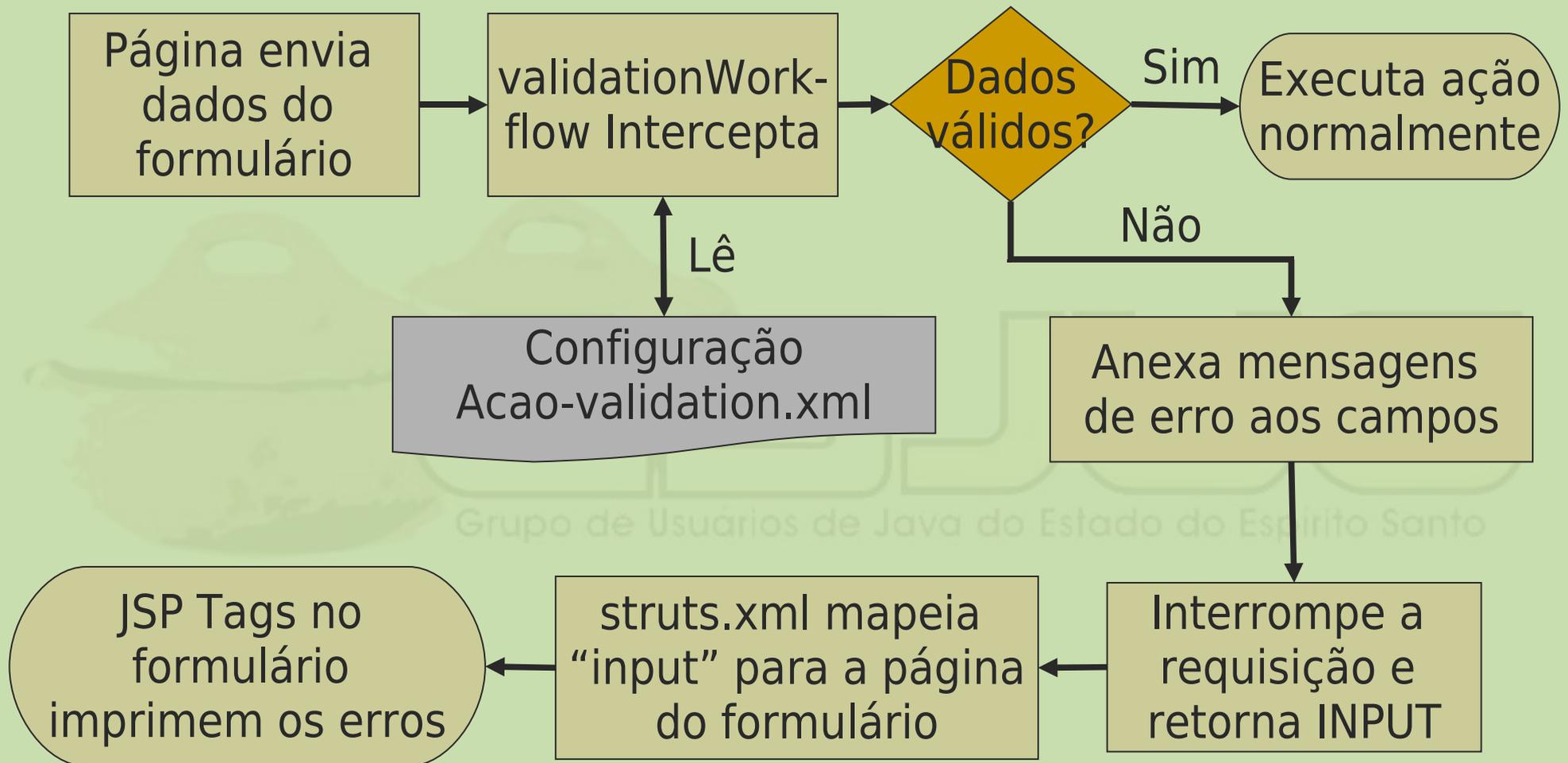
Grupo de Usuários de Java do Estado do Espírito Santo

# Struts<sup>2</sup>

# Framework de validação

- Validação automática de campos de formulário mediante configuração;
- Participantes:
  - Interceptors “validation” e “workflow” (pilha “validationWorkflowStack”);
  - Classe de ação qualquer: `Acao.java`;
  - Configuração: `Acao-validation.xml`;
  - Validadores;
  - *Tags* JSP na página de formulário.

# Fluxo de validação



# Configuração

- Validadores disponíveis:
  - Arquivo `validators.xml`;
  - Struts<sup>2</sup> provê uma configuração padrão.
- Validadores utilizados em uma ação:
  - `ClasseAcao-validation.xml`;
  - `ClasseAcao-NomeAcao-validation.xml`;
  - Configurações de validação são herdadas por subclasses de ações;
  - Para cada campo, especifique as validações a serem feitas e as mensagens de erro.

## Alguns validadores do Struts<sup>2</sup>

required	Não pode ser nulo.
requiredstring	String não nula nem vazia.
int	Número inteiro em uma determinada faixa de valores.
date	Data em um determinado período.
expression	Uma expressão qualquer é verdadeira.
fieldexpression	Idem, porém relacionado a um campo.
email	String é um endereço de e-mail.
url	String é um endereço URL.

## Alguns validadores do Struts<sup>2</sup>

visitor	Delega a validação a um objeto de domínio, que possui suas próprias configurações de validação.
conversion	Verifica erros de conversão.
stringlength	String possui tamanho entre uma faixa de valores.
regex	String encaixa-se numa expressão regular.

Além disso, erros de conversão são automaticamente anexados aos campos.

# Adicionando erros manualmente

- A classe `ActionSupport` (da qual normalmente herdamos) possui métodos para isso:
  - `addActionError(String mensagem);`
  - `addActionMessage(String mensagem);`
  - `addFieldError(String campo, String msg).`
- O *framework* de validação usa estes métodos.

Grupo de Usuários de Java do Estado do Espírito Santo

# Validação *client-side*

- A validação dos campos pode também ser feita no *browser*, via JavaScript;
- Os *templates* `xhtml` e `css_xhtml` provêem estes *scripts* automaticamente;
- Somente alguns validadores funcionam do lado do cliente.

```
<html><head><s:head /></head><body>  
  
<s:form action="acao" validate="true" method="post">  
  <s:textfield label="Nome" name="nome" />  
  <s:textfield label="Idade" name="idade" />  
  <s:submit/>  
</s:form>
```

# Validação *client-side* AJAX

- Outra opção de validação no cliente usa AJAX (*Asynchronous JavaScript and XML*);
- A medida que os campos são preenchidos eles são validados no servidor;
- Validação AJAX no servidor:
  - Todos os validadores funcionam;
  - Validação é codificada apenas uma vez.

```
<s:head theme="ajax" />
```

```
<s:form action="acao" validate="true" theme="ajax">
```

# Validação *client-side* AJAX

- Precisa de alterações no `web.xml`:

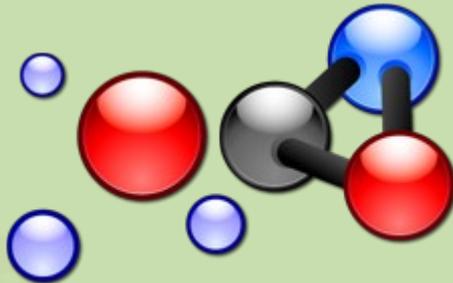
```
<servlet>
  <servlet-name>dwr</servlet-name>
  <servlet-class>
    uk.ltd.getahead.dwr.DWRServlet
  </servlet-class>
  <init-param>
    <param-name>debug</param-name>
    <param-value>>true</param-value>
  </init-param>
</servlet>
<servlet-mapping>
  <servlet-name>dwr</servlet-name>
  <url-pattern>/dwr/*</url-pattern>
</servlet-mapping>
```

- Precisa do arquivo de configuração `dwr.xml`.

# Demonstração



Criar um formulário com: nome (não-vazio), tipo ("A", "B" ou "C"), percentual de comissão (0 a 100), data de nascimento (data passada), e-mail (validar e-mail) e página pessoal (opcional). Mostrar os três tipos de validação (básica, cliente e AJAX).



## Invertendo o controle: IoC/DI

**Struts<sup>2</sup>**

# Injeção de dependências

- *Dependency Injection*, conhecida também como Inversão de Controle (Inversion of Control – IoC);
- Maneira de administrar dependências entre objetos:
  - “Don’t call us, we’ll call you”;
  - Se o objeto A depende de um objeto B, antes que A execute, B é injetado em A;
  - Alguém deve realizar as injeções (*DI Container* – Spring).

# Spring

- A partir do WebWork 2.2: recomendação do Spring como IoC (portanto, Struts<sup>2</sup> idem);
  - Há um *container* IoC interno, *deprecated*;
- Já configuramos o Spring para funcionar no primeiro passo do tutorial:
  - Filtro e *listener* no `web.xml`;
  - `WEB-INF/applicationContext.xml`.

```
<!-- Para determinar os arqs. de config., adicione (web.xml): -->
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/applicationContext-*.xml</param-value>
</context-param>
```

# Componentes injetáveis

- Objetos que são automaticamente injetados nas classes que dependem deles;
- Escopo (visibilidade e tempo de vida):
  - De aplicação (*singleton*);
  - De sessão;
  - De requisição.
- Formas de ligação (*wiring*):
  - Automática por nome (*default*);
  - Automática por tipo;
  - Manual.

# Componentes injetáveis

- Passos para criação:

1º: Escreva a classe do componente;

2º: Coloque um atributo desta classe (ou interface) na classe que depende dele, juntamente com um método *setter*;

3º: Registre o componente na configuração do Spring, com o mesmo nome do atributo (não esqueça do escopo!);

4º: Use seu componente, confiando que o *framework* injetará as dependências pra você.

# Exemplos de configuração

```
<beans [...]>
```

```
  <bean id="componenteDeAplicacao"  
        class="pacote.Classe1" />
```

```
  <bean id="componenteDeSessao"  
        class="pacote.Classe2" scope="session" />
```

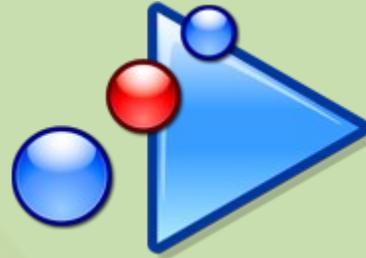
```
  <bean id="componenteDeRequisicao"  
        class="pacote.Classe3"  
        singleton="false" />
```

```
</beans>
```

# Demonstração



Criar um carrinho de compras (sessão) e uma classe de aplicação que implemente um caso de uso de adicionar um item ao carrinho. Criar uma ação que dê acesso a este caso de uso, usando DI para injetar os dois componentes nela.



## Ações encadeadas

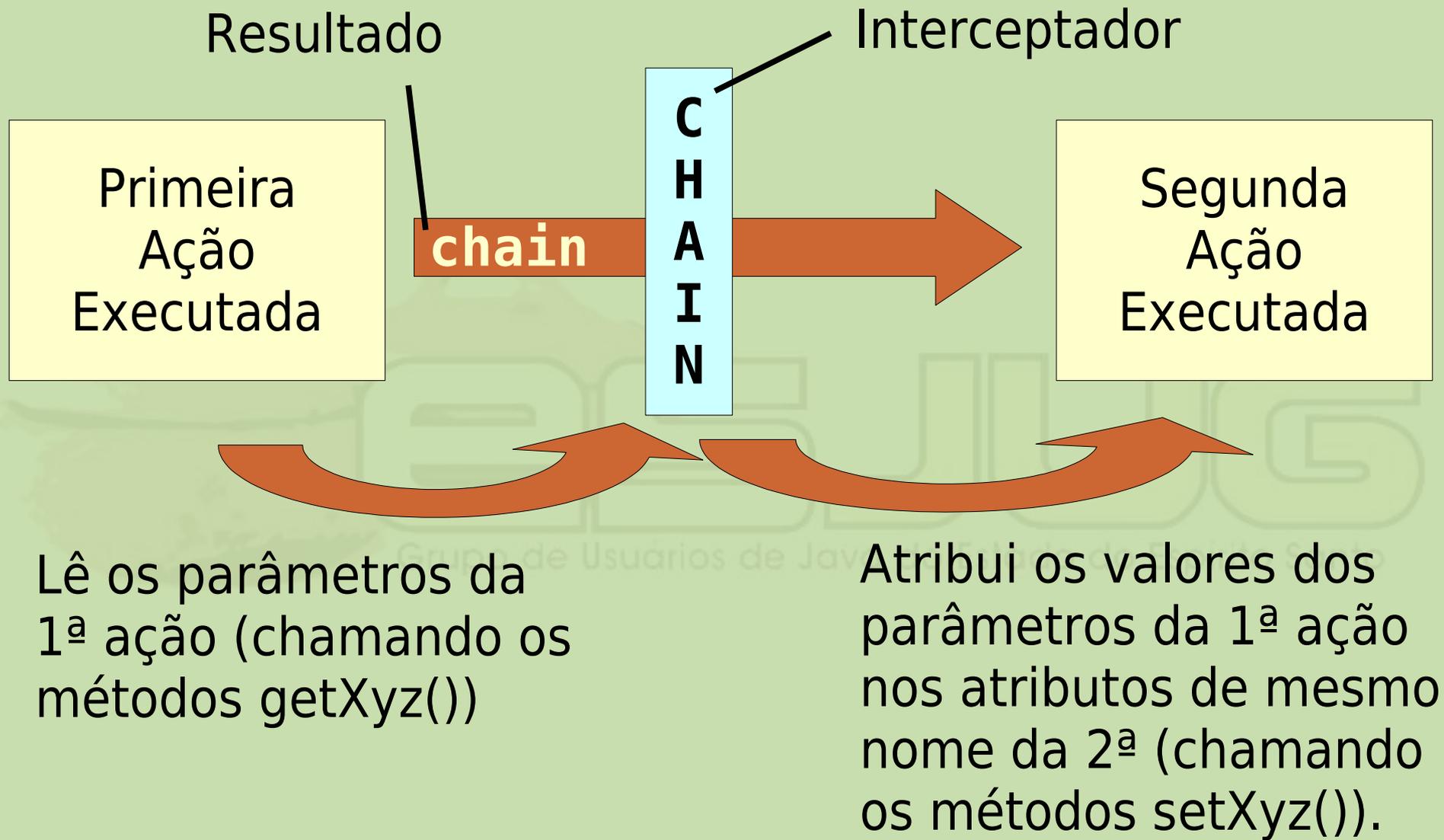
**Struts<sup>2</sup>**

## O resultado chain

- Permite que você defina uma ação como resultado de outra:
  - Promove o reuso de ações ao invés da repetição de código;
  - Necessita do interceptador chain;
  - As propriedades de todas as ações encadeadas estão disponíveis no final da cadeia.

Grupo de Usuarios de Java do Estado do Espírito Santo

# Funcionamento



# Demonstração



Construir uma ação de cadastro de funcionário encadeada com uma ação de exibição de dados de funcionário.



## Enviando arquivos

**Struts<sup>2</sup>**

# Integração com ferramentas externas

- Struts<sup>2</sup> integra-se com ferramentas externas de upload:
  - Jakarta Commons FileUpload (*default*);
  - O'Reilly COS Multipart;
  - PELL Multipart.
- É necessário incluir a biblioteca adequada:
  - `lib/fileupload/*.jar` e `commons-io` (*download separado*);
  - `lib/fileupload-cos/*.jar`;
  - `lib/fileupload-pell/*.jar`.

## Formulário com *upload*

- Inclusão do parâmetro `enctype="multipart/form-data"`;
- JSP tag `<s:file />`.

```
<@s.form action="upload" method="post"
          enctype="multipart/form-data">

  <@s.file label="Arquivo" name="arquivo" />
  <@s.submit value="Enviar" />

</@s.form>
```

# Ação com upload

- Se o campo do formulário chama-se arquivo:
  - `setArquivo(File f)` atribui o arquivo;
  - `setArquivoFileName(String s)` atribui o nome do arquivo;
  - `setArquivoContentType(String s)` atribui o tipo de arquivo.

```
public class UploadAction extends ActionSupport {  
    private File arquivo;  
    private String arquivoFileName;  
    private String arquivoContentType;  
  
    /* Getters & setters. */  
}
```

# Demonstração



Criar uma ação que receba um arquivo e imprima o tipo do arquivo, o nome e o tamanho em Bytes, KiloBytes e MegaBytes.



## Interfaces internacionais: i18n

**Struts<sup>2</sup>**

# Internacionalização (i18n)

- Capacidade de escolher a linguagem do sistema;
- Mensagens são extenalizadas em “*resource bundles*”;
  - Arquivos `.properties`;
  - Formato chave = valor;
  - Padrão de nome (v. API `ResourceBundle`):  
`nome_língua_país.properties`.
- O *framework* escolhe o *bundle* apropriado de acordo com informações de localização (*locale*).

## Suporte do Struts<sup>2</sup> à i18n

- *Tags* de formulário podem obter seus *labels* de *resource bundles*;
- Há *tags* específicas para imprimir mensagens internacionalizadas;
- *Framework* de validação também suporta i18n;
- Troca rápida de *locale*:

**<p>Idioma:**

```
<a href="?request_locale=en_US">Inglês</a> |  
<a href="?request_locale=pt_BR">Português</a>  
</p>
```

# Resource bundles

## # AcaoCadastrar\_pt\_BR.properties

form.nome = Nome

form.dataNasc = Data de Nascimento

form.submit = Enviar

texto.resposta = Caro {0} você tem {1} ano(s).

erro.nome.vazio = Por favor, preencha seu nome.

## # AcaoCadastrar\_en.properties

form.nome = Name

form.dataNasc = Birthdate

form.submit = Submit

texto.resposta = {0}, you are {1} year(s) old.

erro.nome.vazio = Please, fill in your name.

## Tags de formulário com i18n

```
<@s.form action="calcular.action" method="post">  
  <@s.textfield label="%{getText('form.nome')}}"  
    name="nome" required="true" />  
  
  <@s.textfield label="%{getText('form.data')}}"  
    name="login" required="true" />  
  
  <@s.submit value="%{getText('form.submit')}}" />  
  
</@s.form>
```

## Tags específicas de i18n

```
<@s.i18n name="meupacote.AcaoCadastrar" />
```

Carrega o *resource bundle* de uma classe de ação

```
<@s.text name="texto.resposta">  
  <@s.param value="nome" />  
  <@s.param value="idade" />  
</@s.text>
```

Imprime na página uma mensagem vinda de um *resource bundle* carregado.

# I18n no *framework* de validação

```
<validators>
  <field name="nome">
    <field-validator type="requiredstring">
      <message key="erro.nome.vazio" />
    </field-validator>
  </field>
</validators>
```

Grupo de Usuários de Java do Estado do Espírito Santo

## **Bundles carregados pelo Struts<sup>2</sup>**

- 1º: ClasseAcao.properties;
- 2º: SuperClasse.properties (toda a hierarquia);
- 3º: Interface.properties (todas implementadas);
- 4º: Arquivo .properties da classe de modelo (repetindo passos 1, 2 e 3), para ModelDriven;
- 5º: package.properties (toda a hierarquia);
- 6º: Arquivo .properties global (configurado em struts.properties).

# Demonstração



Repetir a demonstração do passo 06 (cálculo da idade com FreeMarker), adicionando validação e com toda a aplicação internacionalizada. Permitir exibição em inglês ou português.



## Usando o resultado Stream

**Struts<sup>2</sup>**

# Resultado stream

- Usado para retornar qualquer formato não-texto:
  - imagens, vídeos, PDF, etc.
- Útil quando:
  - O arquivo é gerado dinamicamente ou recuperado de um banco de dados;
  - Quer proteger determinados arquivos atrás de uma lógica de negócio (ex.: verificação de acesso).

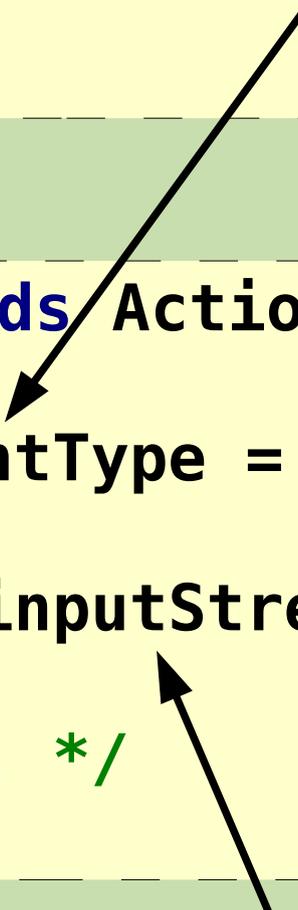
# Parâmetros

- Configurações em struts.xml:
  - contentType: o tipo MIME que será informado ao navegador (default: `text/plain`);
  - contentTypeLength: tamanho do arquivo, para que o navegador mostre indicador de progresso;
  - contentTypeDisposition: `inline` ou `filename="nome.ext"` (default: *inline*);
  - inputName: nome do InputStream de onde vêm os dados (default: `inputStream`);
  - bufferSize: tamanho do buffer de cópia (default: `1024`)

# Exemplo

```
<result type="stream">  
<param name="contentType">{%contentType}</param>  
</result>
```

```
public class Acao extends ActionSupport {  
    private String contentType = "image/jpeg";  
    private InputStream inputStream;  
    /* Getters & setters. */  
}
```



Nome *default* da fonte de dados.

# Resultado chart

- Existe um plugin para JFreeChart:
  - Adicionar struts2-jfreechart-plugin-2.0.9.jar.
- Mais informações sobre seu uso na documentação do Struts<sup>2</sup>:
  - *Getting Started > Guides > Bundled Plugins.*

Grupo de Usuários de Java do Estado do Espírito Santo

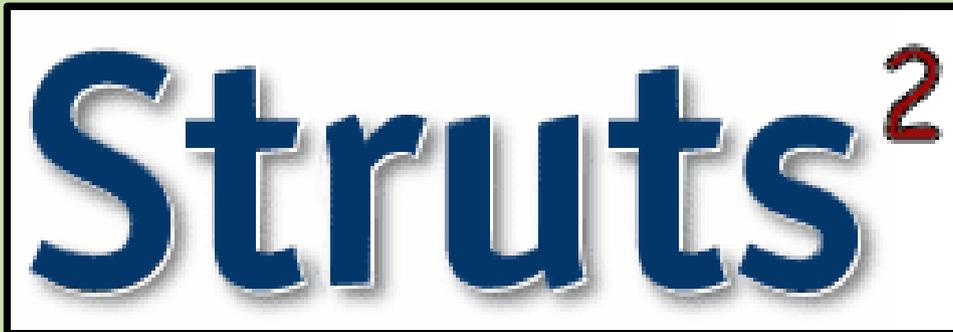
# Demonstração



Usando a biblioteca JFreeChart, obter dados sobre utilização de *frameworks Web* em projetos e exibir um gráfico do tipo pizza.



## Relatórios com JasperReports



# Resultado jasper

- Necessita struts2-jasperreports-plugin-2.0.9.jar;
- Usado para gerar e exibir relatórios com JasperReports:
  - Suporta todos os formatos que o JasperReports suporta;
  - Ação deve disponibilizar o *datasource* (fonte de dados para o relatório);
  - Localização do relatório (\*.jasper) e formato são especificados no struts.xml.

# Parâmetros

- Configurações em struts.xml:
  - location: localização do relatório compilado;
  - dataSource: expressão OGNL que indica onde a fonte de dados se encontra;
  - format: formato de relatório a ser gerado (CSV, HTML, PDF, XLS ou XML – *default*: PDF);
  - contentDisposition: idem stream;
  - documentName: nome do documento gerado (o navegador já salva com o nome dado);
  - delimiter: delimitador usado no formato CSV (*default*: “,”).

# Exemplo

```
<result name="success" type="jasper">
  <param name="location">
    WEB-INF/reports/meurelatorio.jasper
  </param>
  <param name="dataSource">dataSource</param>
  <param name="documentName">Relatorio</param>
</result>
```

```
public class Acao extends ActionSupport {
    private List dataSource = new ArrayList();
    private String parametro;

    /* Getters & setters. */
}
```



# Construção do relatório

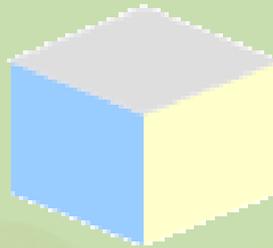
- Propriedades dos objetos do dataSource são acessíveis via `$F{}`:
  - Ex.: se a lista contém objetos Funcionario, podemos acessar seus dados com `$F{nome}`, `$F{login}`, etc.
- Atributos da classe de ação são lidos via `$P{}`:
  - Ex.: `$P{parametro}` está disponível no relatório.
- É necessário ter o método `getXyz()` apropriado.

Grupo de Usuários de Java do Estado do Espírito Santo

# Demonstração



Em cima do exemplo anterior, exibir as informações passadas agora em forma tabular num relatório em PDF. Exibir, também, o gráfico do JFreeChart logo abaixo da tabela.



OPENSYPHONY  
QUALITY COMPONENTS

## Integração com SiteMesh

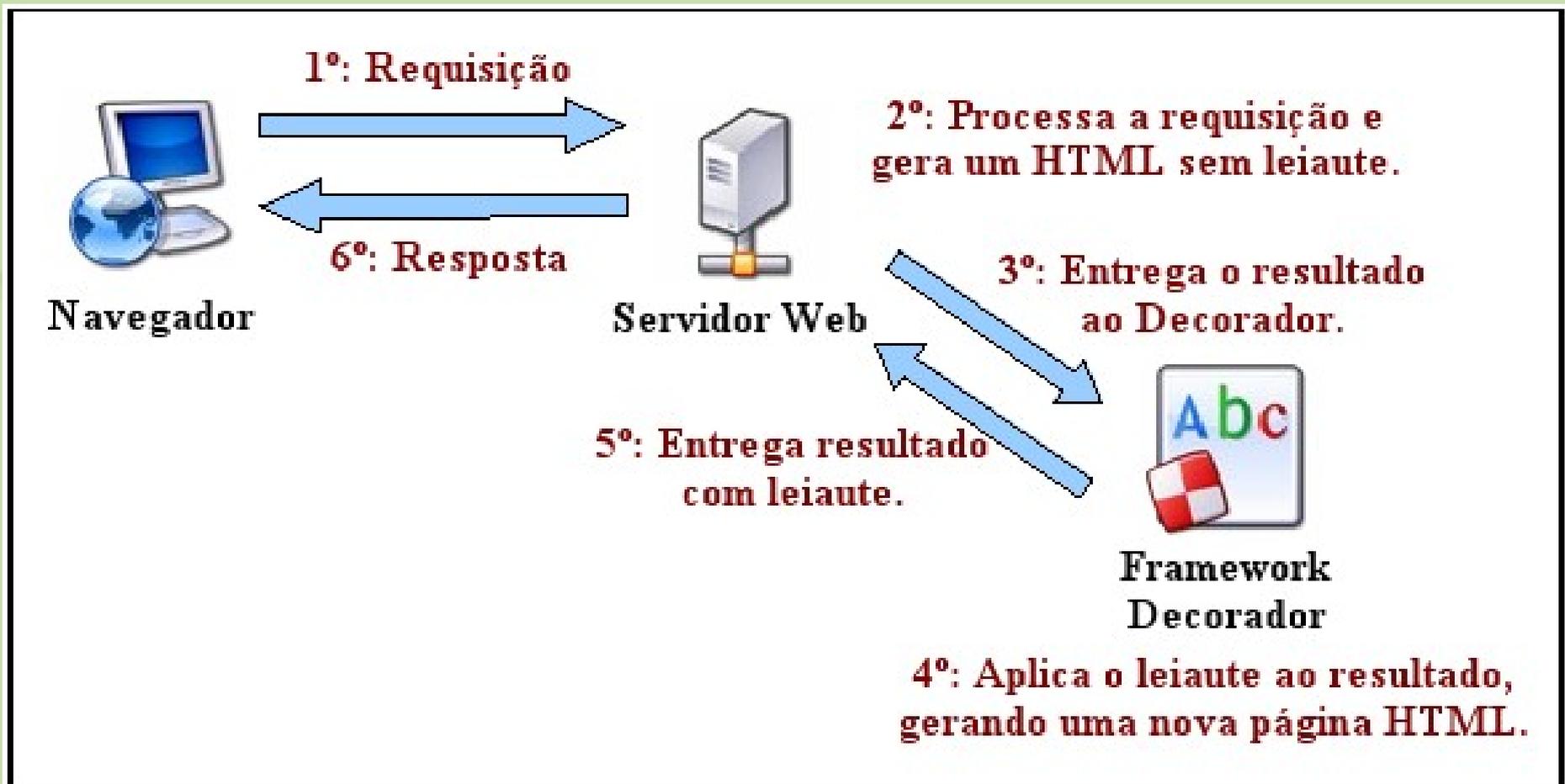
**Struts<sup>2</sup>**

# SiteMesh

- *Framework open-source* para decoração de páginas *Web* (similar ao *Tiles*, da *Apache*):
  - Ajuda a manter a consistência de *layout* em sites com muitas páginas;
  - Intercepta a requisição e aplica a decoração antes da resposta ao cliente (padrão *Decorator* do *GoF*);
  - Também pode montar uma página a partir de vários painéis (semelhante a portais, padrão *Composite* do *GoF*).

# Integração com o Struts<sup>2</sup>

- Qualquer aplicação *Web* integra-se facilmente com SiteMesh:



# Características

- Necessita struts2-sitemesh-plugin-2.0.9.jar e sitemesh-2.2.jar;
- Podemos usar as JSP Tags do Struts<sup>2</sup> em nossas páginas de layout (decoradores);
- Podemos escrever decoradores em HTML, JSP ou FreeMarker;
- Precisamos configurar o SiteMesh:
  - WEB-INF/web.xml;
  - WEB-INF/decorators.xml.

```
<!-- Entre o filtro do Spring e do Struts2. -->  
<filter>  
  <filter-name>strutsCleanup</filter-name>  
  <filter-class>  
    org.apache.struts2.  
    dispatcher.ActionContextCleanUp  
  </filter-class>  
</filter>  
<filter>  
  <filter-name>sitemesh</filter-name>  
  <filter-class>  
    org.apache.struts2.  
    sitemesh.FreeMarkerPageFilter  
  </filter-class>  
</filter>
```

```
<!-- Entre o filtro do Spring e do Struts2. -->  
<filter-mapping>  
  <filter-name>strutsCleanup</filter-name>  
  <url-pattern>/*</url-pattern>  
</filter-mapping>  
<filter-mapping>  
  <filter-name>sitemesh</filter-name>  
  <url-pattern>/*</url-pattern>  
</filter-mapping>
```

Grupo de Usuários de Java do Estado do Espírito Santo

# decorators.xml

```
<decorators defaultdir="/WEB-INF/decorators">  
  <decorator name="main" page="main.ftl">  
    <pattern>/*</pattern>  
  </decorator>  
</decorators>
```



# meutemplate.ftl

```
<html>
<head>
  <title>${title}</title>
  <link rel="stylesheet" type="text/css"
    href="${base}/main.css" />
</head>
<body>

<!-- Alguns cabeçalho. -->
${body}
<!-- Alguns rodapé. -->
</body>
</html>
```

Título definido na página

Diretório raiz da WebApp

Corpo da página

## Isso é só o básico...

- SiteMesh tem muitas outras funcionalidades:
  - A página pode escolher um decorador;
  - A página pode definir valores de parâmetros que são acessíveis no decorador (ex.: indicar se deve mostrar ou não um determinado menu);
  - SiteMesh tem suas próprias *tags*;
  - Etc.

Grupo de Usuários de Java do Estado do Espírito Santo

# Demonstração



Integrar todo o tutorial  
com o SiteMesh,  
automaticamente  
aplicando um mesmo  
*layout* a todas as  
páginas.

O que este tutorial não cobre...



**Tópicos Avançados**

**Struts<sup>2</sup>**

# Tópicos avançados

- Criação de tipos de resultado, conversores de dados e validadores customizados;
- *Continuations*;
- Criação do seu próprio `ActionMapper`;
- Uso avançado de *tags* JSP;
- Criação de componentes visuais e temas;
- Uso avançado de OGNL;
- Suporte à anotações (Java 5);
- Portlets.

# Ferramentas

- Plugins para IDEs:
  - EclipseWork;
  - WebWork IDEA plugin.
- SiteGraph:
  - Monta um grafo com as ações e resultados.

Grupo de Usuários de Java do Estado do Espírito Santo