
# PROJETO FRAMEWORK - CELEPAR

# GERENCIAMENTO DE LAYOUT COM TILES


**Janeiro – 2005**

## Sumário de Informações do Documento

| | | |
|---|---|---|
| **Tipo do Documento:** Definição | | |
| **Título do Documento:** Gerenciamento de Layout com Tiles | | |
| **Estado do Documento:** EB (Elaboração) | | |
| **Responsáveis**: Grupo Framework | | |
| **Palavras-Chaves:** layout, tiles, struts, view | | |
| **Resumo:** Padronização da forma de gerenciar a estrutura da apresentação das aplicação | | |
| **Número de páginas**: 23 | | |
| **Software utilizados:** | | |
| **Versão** | **Data** | **Mudanças** |
| 1.0 | | |

# Sumário

# 1 TILES COM STRUTS

## 1.1 Pré-requisito

Este tutorial é voltado para os desenvolvedores que tenham alguns conhecimentos em JSP e sobre os conceitos de Struts.

## 1.2 Introdução

Para realizar o gerenciamento de layout das aplicações foi padronizada a utilização do gerenciador Tiles disponível no framework Struts.

Normalmente no desenvolvimento de uma aplicação Web, o grupo responsável pela interface com o usuário (UI) cria o Look and Feel (L&F) do site. Baseando-se no Look and Feel, este grupo cria páginas HTML que representam funcionalidades e a forma de navegação. Com uma implementação baseada em Servlet e Java Server Pages (JSP), onde as páginas HTML são convertidas em Servlets e JSP, os designers de UI identificam componentes HTML e JSP comuns, como o cabeçalho (Header), o corpo (Body), e o rodapé da página (Footer), o menu, etc. Este documento apresenta várias soluções para uma efetiva e eficiente organização dos componentes de apresentação (component view).

Como dito anteriormente, para explorar as soluções de Templates e de Layouts, será utilizado o framework Tiles. O “Framework Tiles Component View” é conhecido como Tiles. Este framework usa um arquivo de configuração que além de proporcionar a reutilização de Tiles, também lhe permite organizar layouts.

Com intuito de explorar a flexibilidade e o poder do Tiles, ele será utilizado de maneira integrada com o Struts.

## 1.3 Número de páginas

A solução deve focar-se em reduzir o número de páginas HTML e JSP. O excessivo número de páginas incrementa a complexidade de desenvolvimento, de administração e manutenção da aplicação.

## 1.4 Repetição de código

Quase sempre é ruim, quanto mais código repetido, mais difícil será a tarefa de desenvolver e dar manutenção na aplicação. Uma simples modificação pode desencadear uma série de modificações em cascata em muitas outras páginas diferentes com consequências imprevisíveis. Uma ótima maneira de se adquirir reusabilidade, é eliminando código repetido.

## 1.5 Controle de layout

Se repetir código é ruim, repetir a lógica de layout do código é pior. Expandir a lógica e o comportamento dos componentes views através de muitos JSP's, pode ser a receita para um desastre. Conseguir reutilizar a mesma tela e a lógica dos layouts é uma ótima forma de reutilização, muito melhor que só estar reutilizando os component views.

## 1.6 Soluções

A seguir serão avaliadas soluções usando exemplos básicos de JSP com components views comuns, como cabeçalho e rodapé. Estas soluções estão seguindo uma ordem de complexidade, logo a seguir serão analisados seus critérios de evolução de uma para outra.

### 1.6.1 Solução 1

Considere as seguintes páginas:

page1.jsp

```
<html>
<body>
  Cabeçalho
  <p>
  Corpo da página 1 ...
  <p>
  Rodapé
  <p>
</body>
</html>
```

page2.jsp

```
<html>
<body>
  Cabeçalho
  <p>
  Corpo da página 2 ...
  <p>
  Rodapé
  <p>
</body>
</html>
```

Em muitos casos os desenvolvedores recebem o código HTML dos designers de UI e os convertem literalmente em JSP. Como podemos ver nas páginas page1.jsp e page2.jsp, temos o cabeçalho e o rodapé repetidos em ambas as páginas. A solução 1 não é a desejável, pois qualquer modificação a ser realizada em um component view, desencadeia modificações entre todas as páginas. Esta solução necessita de previsões futuras sobre manutenção.

### 1.6.2 Solução 2

Considere as seguintes páginas:

page1.jsp

```
<html>
  <body>
    <%-- include header --%>
    <jsp:include page="/tiles/header.jsp" />
    Corpo da página 1 ...
    <p>
    <%-- include footer --%>
    <jsp:include page="/tiles/footer.jsp" />
  </body>
</html>
```

page2.jsp

```
<html>
  <body>
    <%-- include header --%>
    <jsp:include page="/tiles/header.jsp" />
    Corpo da página 2 ...
    <p>
    <%-- include footer --%>
    <jsp:include page="/tiles/footer.jsp" />
  </body>
</html>
```

Note que componentes comuns como cabeçalho e rodapé são repetidos usando o mecanismo JSP Include.

header.jsp

```
Cabeçalho
<p>
```

footer.jsp

```
Rodapé
<p>
```

A solução 2 soluciona alguma das limitações maiores em relação a solução 1. Só é necessário modificar os includes utilizados para obtermos a alteração em todo o component view. Esta solução elimina a duplicidade de código e simplifica a manutenção, entretanto aumenta o número de páginas. Esta é uma solução simples, mas é usada em muitas aplicações reais hoje em dia, porém existe um inconveniente, se modificarmos o layout da aplicação, teremos que modificar todas as páginas, o que se supõe que seja uma modificação muito cara, impossibilitando por diversos fatores esta modificação. Além de ser uma solução muito usada hoje pela maioria dos sistemas ainda não conseguimos reaproveitar a lógica de layout.

### 1.6.3 Solução 3

Considere as seguintes páginas:

page1.jsp

```
<%@ taglib uri="/WEB-INF/struts-tiles.tld" prefix="tiles" %>
<html>
  <body>
    <%-- include header --%>
    <tiles:insert page="/tiles/header.jsp" flush="true"/>
    Corpo da página 1 ...
    <p>
      <%-- include footer --%>
      <tiles:insert page="/tiles/footer.jsp" flush="true"/>
  </body>
</html>
```

page2.jsp

```
<%@ taglib uri="/WEB-INF/struts-tiles.tld" prefix="tiles" %>
<html>
  <body>
    <%-- include header --%>
    <tiles:insert page="/tiles/header.jsp" flush="true"/>
    Corpo da página 2 ...
    <p>
      <%-- include footer --%>
      <tiles:insert page="/tiles/footer.jsp" flush="true"/>
  </body>
</html>
```

A solução 3 usa o mecanismo insert do Tiles. Usando isto estamos incluindo os component views no site correspondente. Em todos os aspectos, esta solução é idêntica a solução 2, com suas vantagens e desvantagens.

### 1.6.4 Solução 4 (Separando o corpo)

Considere as seguintes páginas:

page1.jsp

```
<%@ taglib uri="/WEB-INF/struts-tiles.tld" prefix="tiles" %>
<html>
      <body>
            <%-- include header --%>
            <tiles:insert page="/tiles/header.jsp" flush="true"/>
            <%-- include body --%>
            <tiles:insert page="/tiles/corpo1.jsp" flush="true"/>
            <%-- include footer --%>
            <tiles:insert page="/tiles/footer.jsp" flush="true"/>
      </body>
</html>
```

page2.jsp

```
<%@ taglib uri="/WEB-INF/struts-tiles.tld" prefix="tiles" %>
<html>
      <body>
            <%-- include header --%>
            <tiles:insert page="/tiles/header.jsp" flush="true"/>
            <%-- include body --%>
            <tiles:insert page="/tiles/corpo2.jsp" flush="true"/>
            <%-- include footer --%>
            <tiles:insert page="/tiles/footer.jsp" flush="true"/>
      </body>
</html>
```

Esta solução separa o núcleo do corpo em páginas individuais, corpo1.jsp e corpo2.jsp

Veja o JSP corpo1.jsp:

```
Corpo da página 1 ...
<p>
```

e o corpo2.jsp:

```
Corpo da página 2 ...
<p>
```

Isto limita modificações no corpo de sua página podendo ser reutilizável em outras partes, eliminando a repetição e a duplicidade. Como as outras soluções, cada página cria ainda seu próprio layout. Portanto ainda não atingimos uma solução quase que ideal.

### 1.6.5 Solução 5 (Templates)

Usando o conceito de criação de templates, podemos definir layouts. A partir destes layouts, podemos inserir marcadores ao invés do atual component view (includes) usando a diretiva insert do Tiles. Assim para todos os componentes, esta página define um layout reutilizável.

Veja este jsp, layout.jsp:

```
<%@ taglib uri="/WEB-INF/struts-tiles.tld" prefix="tiles" %>
<html>
  <body>
    <%-- include header --%>
    <tiles:insert attribute="header"/>
    <%-- include body --%>
    <tiles:insert attribute="body"/>
    <%-- include footer --%>
    <tiles:insert attribute="footer"/>
  </body>
</html>
```

Nas atuais páginas inserimos os componentes usando a diretiva Tiles insert. Usando a diretiva Tiles put, podemos especificar o component view atual para todos os marcadores.

Considere as páginas:

page1.jsp

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

<tiles:insert page="/tiles/layout.jsp" flush="true">
    <tiles:put name="header" value="/tiles/header.jsp"/>
    <tiles:put name="body" value="/tiles/corpo1.jsp"/>
    <tiles:put name="footer" value="/tiles/footer.jsp"/>
</tiles:insert>
```

page2.jsp

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

<tiles:insert page="/tiles/layout.jsp" flush="true">
    <tiles:put name="header" value="/tiles/header.jsp"/>
    <tiles:put name="body" value="/tiles/corpo2.jsp"/>
    <tiles:put name="footer" value="/tiles/footer.jsp"/>
</tiles:insert>
```

A maior vantagem de se usar esta solução, é que encapsula o comportamento do layout, reduzindo drasticamente o acoplamento do mesmo entre os componentes. Entretanto incrementa a complexidade introduzindo outra página para o layout. Entender e implementar o mecanismo de templates também pode ser difícil a princípio.

Esta solução ainda poderia eliminar a criação das páginas utilizadas dentro do corpo body (corpo1.jsp, corpo2.jsp, ...) que são as únicas que variam de conteúdo neste exemplo com o com o truque de fornecer a tag put o valor final do conteúdo, é necessário apenas especificar type=”String” ao invés de especificar o caminho da página para que o Tiles não cometa um erro esperando o caminho de uma página e escrever o próprio código JSP no corpo da tag put.

As páginas então ficariam assim:

page1.jsp

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

<tiles:insert page="/tiles/layout.jsp" flush="true">
    <tiles:put name="header" value="/tiles/header.jsp"/>
    <tiles:put name="body" type="string">

      Corpo da página 1 ...
      <p>

    </tiles:put>
    <tiles:put name="footer" value="/tiles/footer.jsp"/>
</tiles:insert>
```

page2.jsp

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

<tiles:insert page="/tiles/layout.jsp" flush="true">
    <tiles:put name="header" value="/tiles/header.jsp"/>
    <tiles:put name="body" type="string">

      Corpo da página 2 ...
      <p>

    </tiles:put>
    <tiles:put name="footer" value="/tiles/footer.jsp"/>
</tiles:insert>
```

### 1.6.6 Solução 6 (Struts e Tiles)

A página de layout anterior layout.jsp contém código HTML e JSP para organizar os componentes. As página page1.jsp e page2.jsp não contém código HTML, entretanto contém as diretivas Tiles para inserir os componentes necessários.

Você não acharia agradável especificar todos as páginas de conteúdo em um só arquivo de configuração XML?

Vamos chamar este arquivo de tiles-defs.xml e especificar as páginas como:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

 <!DOCTYPE tiles-definitions PUBLIC
       "-//Apache Software Foundation//DTD Tiles Configuration//EN"
       "http://jakarta.apache.org/struts/dtds/tiles-config.dtd">

<tiles-definitions>
    <definition name="page1Def" path="/tiles/layout.jsp">
        <put name="header" value="/tiles/header.jsp"/>
        <put name="footer" value="/tiles/footer.jsp"/>
        <put name="body" value="/tiles/corpo1.jsp"/>
    </definition>

    <definition name="page2Def" path="/tiles/layout.jsp">
        <put name="header" value="/tiles/header.jsp"/>
        <put name="footer" value="/tiles/footer.jsp"/>
        <put name="body" value="/tiles/corpo2.jsp"/>
    </definition>
</tiles-definitions>
```

A solução 6 elimina as páginas de conteúdo page1.jsp e page2.jsp, colocando suas definições em um arquivo XML. Se não existir um recurso chamado page1.jsp, como podemos solicitá-lo? E o mais importante, como faremos para o Tiles interpretar o arquivo XML tiles-defs.xml?

É aí que acontece a integração entre Struts e Tiles. Para o Struts encontrar e

reconhecer a definição do Tiles, estaremos inserindo a seguinte bloco de configuração de plug-in no arquivo struts-config.xml, antes do fechamento da tag struts-config:

```
<struts-config>
....
    <plug-in className="org.apache.struts.tiles.TilesPlugin" >
      <set-property property="definitions-config"
                    value="/WEB-INF/tiles-defs.xml" />
    </plug-in>
</struts-config>
```

Não podemos invocar uma definição diretamente do navegador, porém podemos invocar através do struts como se fosse um recurso default do mesmo. Definimos as actions Struts no arquivo struts-config.xml como vemos abaixo:

```
      <action path="/page1" forward="page1Def"/>
      <action path="/page2" forward="page2Def"/>
```

Agora invocaremos a action Struts solicitando page1.do e page2.do respectivamente para desenvolvermos o recurso desejado.

A solução 6 tem a vantagem de estabelecer todas as definições em um arquivo de configuração XML, reduzindo o número de páginas.

### 1.6.7 Solução 7 (herança em Tiles)

No arquivo de configuração é onde todas as definições são feitas. Estas definem três componentes, cabeçalho, corpo e rodapé da página, dos quais (rodapé e cabeçalho) são sempre iguais. Uma poderosa característica do Tiles é a herança entre definições. Portanto podemos criar uma definição base e definir que as demais herdem desta. A definição base somente deve conter as características. A definição filha somente deve definir seus componentes únicos.

A seguir observe o seguinte código usando herança em um arquivo de configuração XML.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

 <!DOCTYPE tiles-definitions PUBLIC
       "-//Apache Software Foundation//DTD Tiles Configuration//EN"
       "http://jakarta.apache.org/struts/dtds/tiles-config.dtd">

<tiles-definitions>
    <definition name="baseDef" path="/tiles/layout.jsp">
        <put name="header" value="/tiles/header.jsp"/>
        <put name="footer" value="/tiles/footer.jsp"/>
        <put name="body" value=""/>
    </definition>

    <definition name="page1Def" extends="baseDef">
        <put name="body" value="/tiles/corpo1.jsp"/>
    </definition>

    <definition name="page2Def" extends="baseDef">
        <put name="body" value="/tiles/corpo2.jsp"/>
    </definition>

</tiles-definitions>
```

Isto elimina parte do código repetido.

## 1.7 Qual é a melhor solução

A melhor solução depende muito da necessidade do projeto, dos requerimentos do mesmo e é claro, de suas habilidades e conhecimentos em desenvolvimento e manutenção de aplicações web. Para soluções onde você já está usando Struts, aproveite a boa fluência de integração que Tiles possui com Struts para desenvolver uma poderosa solução.

## 1.8 Exemplo prático

Para este exemplo suponha que o layout da aplicação contém elementos comuns, como um menu e um rodapé como o primeiro modelo da figura a seguir. Agora imagine que o cliente queira modificar o layout básico da aplicação do primeiro para o segundo modelo. Quantas páginas seriam necessárias alterar um projeto grande? Isso se não estivesse sendo utilizado um mecanismo de templates como o Tiles.

| Topo | | | Topo |
|---|---|---|---|
| Menu | Conteúdo | | Menu |
| | | | Conteúdo |
| Rodapé | | | Rodapé |

Aplicações web baseadas no tiles são montadas com “peças”. Assim é possível definir e modelar uma vez e replicar o layout em um grupo de páginas ou mesmo em todas as páginas da aplicação.

Para este exemplo é necessário criar um página (layout.jsp) que servirá de base para o layout das outras páginas da aplicação. A estrutura usada neste exemplo é definida usando tabelas HTML. Para o primeiro layout é necessário então criar as “regiões” topo, menu, rodape e conteudo.

layout.jsp:

```
<%@ taglib uri="/WEB-INF/struts-tiles.tld" prefix="tiles" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<html:html locale="true">
 <head>
  <title>Tiles com Struts</title>
</head>

<body bgcolor="#FFFFFF" leftmargin="0" topmargin="0">
<div align="left">
  <table width="100%" border="0" cellspacing="0">
    <tr>
      <td colspan="2"><tiles:get name="topo"/></td>
    </tr>
    <tr>
      <td width="17%"><tiles:get name="menu"/></td>
      <td width="83%"><tiles:get name="conteudo"/></td>
    </tr>
    <tr>
      <td colspan="2"><tiles:get name="rodape"/></td>
    </tr>
  </table>
</div>
</body>
</html:html>
```

No começo é feita a declaração da taglib do tiles:

<%@ taglib uri=”/WEB-INF/struts-tiles.tld” prefix=”tiles” %>

Em seguida são declaradas as porções dentro do código HTML usando a taglib tiles: <tiles:get name=”topo”/>, <tiles:get name=”menu”/> etc. Na execução da página, estas tags serão substituídas pelas JSPs informadas no arquivo de definição tiles-defs.xml que neste exemplo tem o seguinte conteúdo para a página home deste exemplo:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

 <!DOCTYPE tiles-definitions PUBLIC
       "-//Apache Software Foundation//DTD Tiles Configuration 1.1//EN"
       "http://jakarta.apache.org/struts/dtds/tiles-config_1_1.dtd">

<tiles-definitions>
      <definition name="exampleHomeDef" path="/tiles/example/layout.jsp">
            <put name="topo" value="/tiles/example/topo.jsp" />
            <put name="menu" value="/tiles/example/menu.jsp" />
            <put name="conteudo" value="/tiles/example/home.jsp" />
            <put name="rodape" value="/tiles/example/rodape.jsp" />
      </definition>
</tiles-definitions>
```

O arquivo tiles-defs.xml contém a especificação das definições do Tiles além de ser a parte principal do mecanismo e responsável pela flexibilidade na manutenção de layouts. Para o exemplo aqui é necessário criar as definições das demais páginas usando herança, redefinindo algumas regiões e reaproveitando o layout da página home.

O conteúdo então do arquivo tiles-defs.xml passa a ser o seguinte:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

 <!DOCTYPE tiles-definitions PUBLIC
       "-//Apache Software Foundation//DTD Tiles Configuration 1.1//EN"
       "http://jakarta.apache.org/struts/dtds/tiles-config_1_1.dtd">

<tiles-definitions>

      <definition name="exampleHomeDef" path="/tiles/example/layout.jsp">
            <put name="topo" value="/tiles/example/topo.jsp" />
            <put name="menu" value="/tiles/example/menu.jsp" />
            <put name="conteudo" value="/tiles/example/home.jsp" />
            <put name="rodape" value="/tiles/example/rodape.jsp" />
      </definition>

      <definition name="exampleErroDef" extends="exampleHomeDef">
            <put name="conteudo" value="/tiles/example/erro.jsp" />
      </definition>

      <definition name="exampleBeanDef" extends="exampleHomeDef">
            <put name="conteudo" value="/tiles/example/bean.jsp" />
      </definition>

      <definition name="exampleTilesDef" extends="exampleHomeDef">
            <put name="conteudo" value="/tiles/example/tiles.jsp" />
      </definition>
</tiles-definitions>
```

Observe que para as definições exampleErroDef, exmpleBeanDef e exampleTilesDef apenas a região conteúdo precisou ser redefinida.

Seguem agora os códigos das demais páginas utilizadas neste exemplo:

topo.jsp

```
<table width="100%" border="0" bgcolor="#000099">
  <tr>
    <td><font face="Arial, Helvetica, sans-serif" size="5"><b><font
color="#FFFFFF">Tiles
      com Struts </font></b></font></td>
  </tr>
</table>
```

menu.jsp

```
<table width="100%" border="1" cellspacing="0" bordercolor="#0000AA">
  <tr>
    <td><a href="home.do"><b><font
color="#000099">home</font></b></a></td>
  </tr>
  <tr>
    <td><a href="bean.do"><b><font color="#000099">struts-
bean.tld</font></b></a></td>
  </tr>
  <tr>
    <td><a href="html.do"><b><font color="#000099">struts-
html.tld</font></b></a></td>
  </tr>
  <tr>
    <td><a href="logic.do"><b><font color="#000099">struts-
logic.tld</font></b></a></td>
  </tr>
  <tr>
    <td><a href="tiles.do"><b><font color="#000099">struts-
tiles.tld</font></b></a></td>
  </tr>
  <tr>
    <td> </td>
  </tr>
</table>
```

rodape.jsp

```
<div align="center">
  <hr size="1">
  <font face="Verdana, Arial, Helvetica, sans-serif, Trebuchet MS"
size="2"><b>Tiles
  com Struts</b></font>
</div>
```

home.jsp

```
<div align="center">
<br><br>
Layout da página Home
<br><br>
</div>
```

erro.jsp

```
<font face="Comic Sans MS" size=4>
  <center>
  <h1><font color=red>Erro 404</font></h1>
  <p>Desculpe, a página que você solicitou não existe.
  </center>
</font>
```

bean.jsp

```
<div align="center">
<br><br>
Corpo da página de opção "struts-bean.tld" do menu
<br><br>
</div>
```

tiles.jsp

```
<div align="center">
<br><br>
Corpo da página de opção "struts-tiles.tld" do menu
<br><br>
</div>
```

O próximo passo é alterar o arquivo de configuração do Struts. Primeiro configurar o plug-in do Tiles, depois criar os actions foward para disparar o trabalho de substituição. Estes devem apontar para o nome da definição feita no tiles-defs.xml, o arquivo struts-config.xml contém então o seguinte:

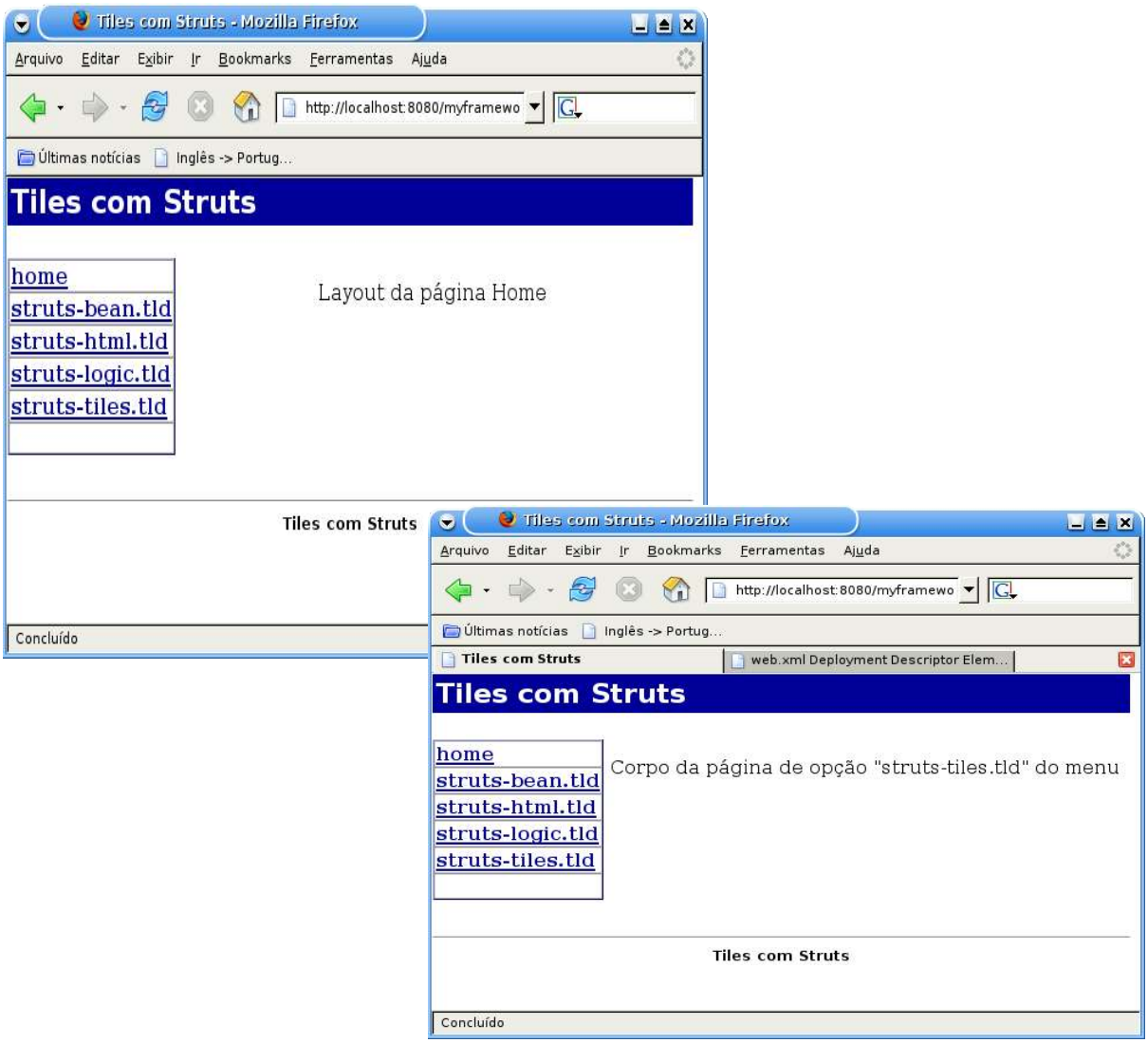
```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE struts-config PUBLIC
          "-//Apache Software Foundation//DTD Struts Configuration
1.2//EN"
          "http://jakarta.apache.org/struts/dtds/struts-config_1_2.dtd">

<struts-config>

      <form-beans></form-beans>
      <global-exceptions></global-exceptions>
      <global-forwards></global-forwards>

      <action-mappings>
            <action path="/erro" forward="exampleErroDef" />
            <action path="/home" forward="exampleHomeDef" />
            <action path="/bean" forward="exampleBeanDef" />
            <action path="/tiles" forward="exampleTilesDef" />
      </action-mappings>

      <controller
         processorClass="org.apache.struts.tiles.TilesRequestProcessor" />
      <message-resources
         parameter="ApplicationResources" />

      <plug-in className="org.apache.struts.tiles.TilesPlugin">
            <set-property property="definitions-config"
                  value="/WEB-INF/tiles-defs.xml" />
      </plug-in>

</struts-config>
```

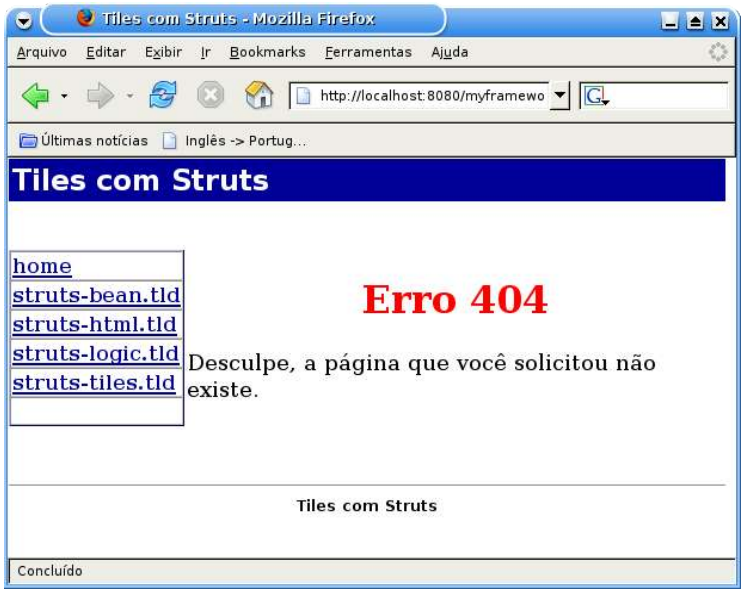
Como neste exemplo ainda foi definida uma página de erro que será utilizada para quando o servidor retornar o erro 404, é necessário ainda definir que quando esse erro ocorrer a página de erro (erro.do) seja chamada, para isso acontecer é necessário adicionar o seguinte bloco de configuração no arquivo web.xml da aplicação:

```
<error-page>
   <error-code>404</error-code>
   <location>/erro.do</location>
</error-page>
```

Repare no arquivo menu.jsp que existem dois links (html.do e logic.do) que não foram definidos no struts-config.xml justamente para simular o erro de página.

Ao rodar a aplicação o resultado é o seguinte:






Todas as páginas ficam de acordo com o modelo do primeiro layout proposto.

Agora se for necessário mudar o layout para o segundo modelo como foi dito anteriormente, é necessário criarmos uma nova pagina de layout.

layout2.jsp

```
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-tiles.tld" prefix="tiles" %>
<html:html locale="true">
 <head>
 <title>Tiles com Struts</title>
</head>

<body vlink="#551a8b" alink="#ee0000" link="#0000ee"
style="background-color: rgb(232, 232, 232); color: rgb(0, 0, 0);"
leftmargin="0" topmargin="0">
<div align="left">
      <table width="100%" border="0" cellspacing="0">
            <tr>
                  <td colspan="2"><tiles:get name="topo"/><br></td>
            </tr>
            <tr>
                  <td colspan="2"><tiles:get name="menu"/><br></td>
            </tr>
            <tr>
                  <td colspan="2"><tiles:get name="conteudo"/><br></td>
            </tr>
            <tr>
                  <td colspan="2"><tiles:get name="rodape"/><br>
            </td>
            </tr>
      </table>
</div>
</body>
</html:html>
```

Observe que as regiões continuam sendo as mesmas, o que muda é a disposição

das mesmas, neste caso foi alterada também a cor de fundo da tela.

Para ficar de acordo com o o novo layout é necessário também alterar a disposição dos itens de menu, neste exemplo foi criado um novo arquivo:

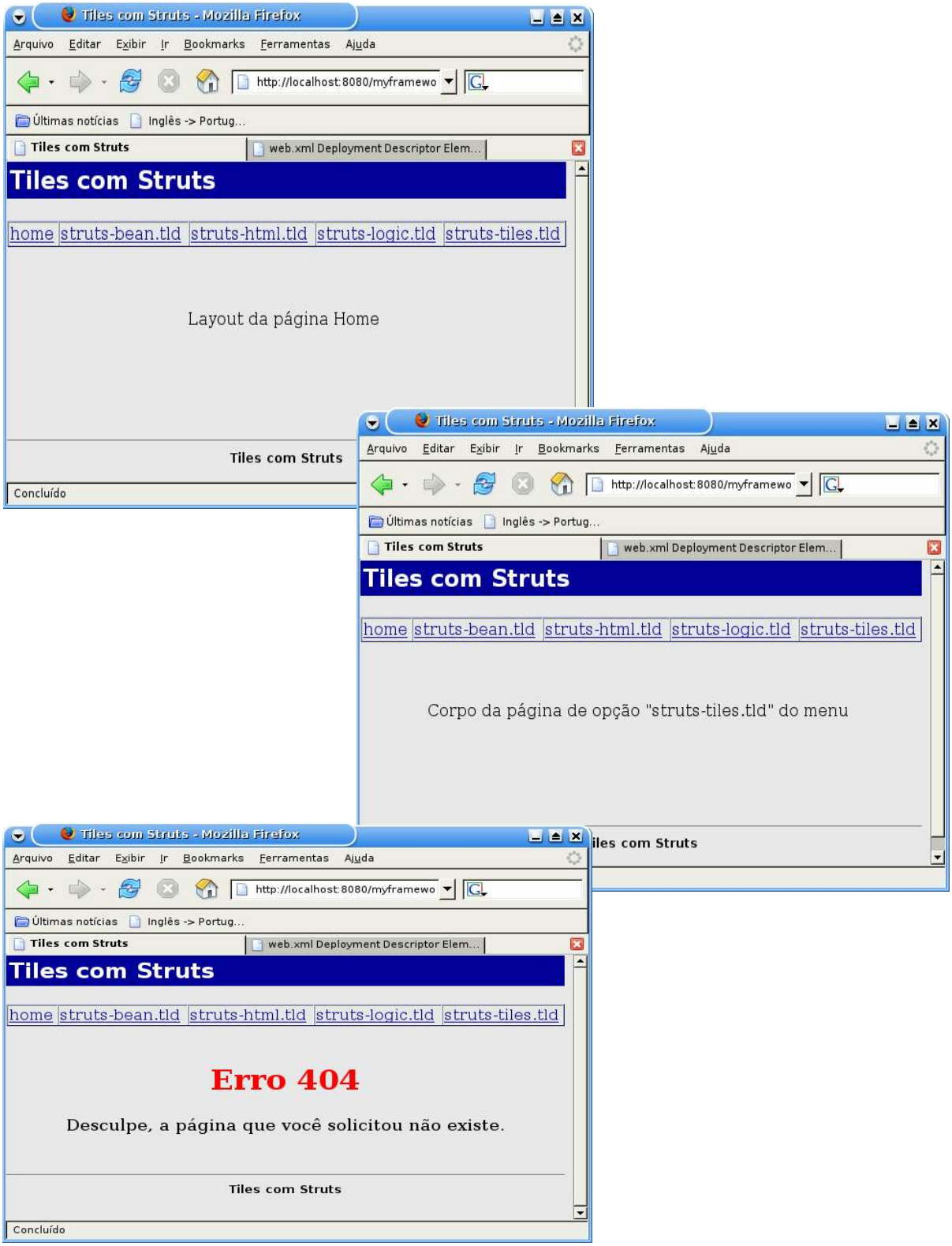
menu2.jsp

```
<table width="100%" border="1" cellspacing="0" bordercolor="#0000AA">
  <tr>
    <td><a href="home.do"><font color="#000099">home</font></a></td>
    <td><a href="bean.do"><font color="#000099">struts-
bean.tld</font></a></td>

    <td><a href="html.do"><font color="#000099">struts-
html.tld</font></a></td>

    <td><a href="logic.do"><font color="#000099">struts-
logic.tld</font></a></td>

    <td><a href="tiles.do"><font color="#000099">struts-
tiles.tld</font></a></td>
  </tr>
</table>
```

Após criados esses dois novos arquivos é necessário alterar a configuração no tiles-defs.xml como mostrado no seguinte bloco para que o Tiles passe a utilizar os novos arquivos:

```
...
<definition name="exampleHomeDef" path="/tiles/example/layout2.jsp">
      <put name="topo" value="/tiles/example/topo.jsp" />
      <put name="menu" value="/tiles/example/menu2.jsp" />
      <put name="conteudo" value="/tiles/example/home.jsp" />
      <put name="rodape" value="/tiles/example/rodape.jsp" />
</definition>
...
```

Após essas alterações o resultado é o seguinte:




Todas as páginas passam a ficar de acordo com o modelo do segundo layout proposto.

Como demonstrado a utilização do Tiles ajuda na reutilização de layouts e

outros elementos de design, contolando de forma centralizada os layouts utilizados na aplicação além de facilitar possíveis alterações de layout.