Desenvolvimento Web no Eclipse – Parte I

Vítor E. Silva Souza (vitorsouza@gmail.com)

Saiba como combinar *frame-works* e ferramentas *open-source* para desenvolvimento rápido de aplicações para a Web.

O desenvolvimento de aplicações para a *Web* evoluiu da criação de páginas estáticas ligadas até os enormes sistemas *business-to-consumer* (B2C) e *business-to-business* (B2B) que hoje povoam a Internet. Durante esta evolução, tecnologias como CGI, ColdFusion, PHP, Microsoft ASP e Java Servlets/JSP surgiram para prover uma infra-estrutura que facilitasse o desenvolvimento de *WebApps* (forma abreviada do termo *Web Application*).

Hoje em dia, o estado-da-prática consiste da utilização de *frameworks* ou arquiteturas baseadas em *containers*. A nova versão da plataforma Enterprise JavaBeans (EJB) em conjunto com a tecnologia JavaServer Faces (JSF) formam o padrão definido pelo Java Community Process (JCP) para desenvolvimento *Web*. Tais padrões foram formados a partir de idéias que se popularizaram pela utilização de diversos *frameworks* como Struts, Spring e Hibernate. Mesmo não sendo padrões, estes *frameworks* ainda são bastante utilizados por não requererem um servidor de aplicações completo para sua execução.

Dividido em várias partes, este artigo pretende abordar a criação de *WebApps* na IDE Eclipse, utilizando *frameworks*, Java EE e outras ferramentas *open-source* úteis neste processo. A idéia não é ensinar a fundo nenhum destes *frameworks*, mas sim relatar passos para sua utilização na IDE Eclipse. O leitor familiarizado com os *frameworks* terá bastante facilidade em compreender todos os passos efetuados. Já os demais leitores devem procurar em outros artigos o complemento necessário.

Aqueles que têm facilidade para aprender a partir de exemplos verão nestes artigos um valioso recurso para o aprendizado e aconselhamos a todos que sigam os passos como um tutorial, explorando novos caminhos sempre que se sentirem à vontade para tal. Esta primeira parte foca na instalação de todas as ferramentas necessárias para construção de um primeiro exemplo a partir de uma aplicação *Web* em branco.

Instalação de Ferramentas

A escolha de uma boa ferramenta é um passo fundamental de qualquer processo construtivo. Para o desenvolvimento de WebApps, escolhemos o servidor Apache Tomcat, a IDE Eclipse e seus plugins J2EE Standard Tools e IvyDE. Tanto o Tomcat quanto o Eclipse necessitam de uma máquina virtual Java, portanto começaremos pela instalação desta.

Kit de Desenvolvimento Java

A instalação do JDK (Java Development Kit) depende do sistema operacional. Em certas distribuições Linux já existe um pacote pronto para instalação (ex.: pacote sun-java6-jdk no repositório *Universe* do Ubuntu Linux), já em outras é preciso baixar o instalador da Sun. Este também é o caso do Windows, que possui um assistente para instalação que pode ser baixado do site **`http://java.sun.com/javase/downloads/index.jsp`** (localize o JDK 6, clique no botão *Download*, leia e aceite o contrato e baixe o software).

Apache Tomcat

Apesar do Tomcat já se encontrar na versão 6.0, o Eclipse atualmente dá suporte somente até a versão 5.5. Instale esta versão a partir do seu repositório de pacotes (Linux) ou baixando do site **`http://tomcat.apache.org/`**. Clique no link **`Tomcat 5.x`** abaixo de **`Downloads`**, em seguida localize a seção **`Quick Navigation`**, escolha a versão **`5.5.20`** e baixe a versão **`Core`**. Existem opções de assistente de instalação ou arquivo compactado.

Eclipse

Assim como o JDK e o Tomcat, muitas distribuições também incluem pacotes binários do Eclipse em seus repositórios. Se não for o caso, baixe o Eclipse no site **`http://www.eclipse.org`**, clicando na aba **`Downloads`** (parte superior da página) e, em seguida, no *link* **`Download now: Eclipse SDK 3.2.2`**. Escolha um *mirror*, baixe o arquivo referente ao seu sistema operacional e descompacte-o em alguma pasta do seu computador.

Primeiramente execute o Eclipse como administrador do sistema (usuários Windows domésticos geralmente são administradores, enquanto usuários Linux precisam efetuar *login* como **`root`**) para instalar a extensão para desenvolvimento *Web*. Clique no menu **`Help`** /

Software Updates / **Find and Install...**. Escolha a opção **Search for new features to install** e clique **Next >**. Na listagem **Sites to include in search**, marque a opção **Callisto Discovery Site** e clique **Finish**. Escolha um *mirror* e clique **OK**.

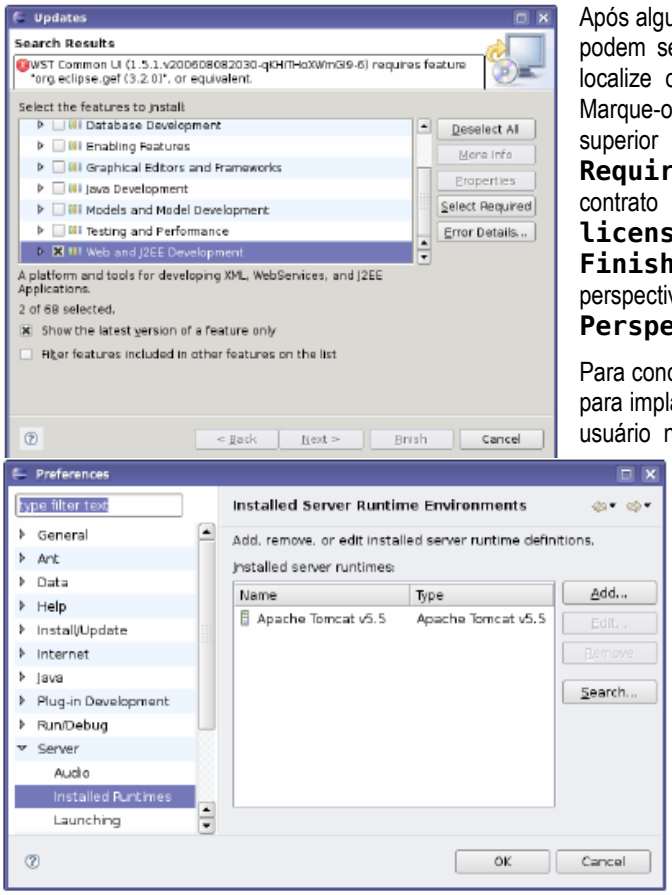

Figura 2. *Configuração do Tomcat no Eclipse.*

Após alguns instantes, a janela **Updates** se abrirá com uma lista de *plugins* que podem ser instalados. Dentro do item **Callisto Discovery Site**, localize o *plugin* **Web and J2EE Development**, ao final da lista. Marque-o para instalação e perceba que o Eclipse apresentará um erro na parte superior da janela, como mostra a figura 1. Clique no boão **Select Required** para corrigir o problema e clique **Next >**. Leia os termos de contrato e marque a opção "**I accept the terms in the license agreement**" para aceitá-los. Clique em **Next >** e em seguida **Finish** para instalar todos os *plugins*. Reinicie o plugin e verifique se a perspectiva J2EE está na listagem apresentada pelo menu **Window** / **Open Perspective** / **Other...**.

Para concluir este passo, precisamos configurar o Eclipse para utilizar o Tomcat para implantação, execução e depuração de *WebApps*. Reinicie o Eclipse como usuário normal, clique no menu **Window** / **Preferences** e localize a opção **Server** / **Installed Runtimes**. Clique no botão **Add**, escolha o servidor **Apache Tomcat v5.5** dentro da pasta **Apache** e clique **Next >**. Informe o diretório onde você instalou o Tomcat e escolha uma das JRE disponíveis.

O servidor será adicionado à lista, como mostra a figura 2. Clique **OK**. Em seguida, clique em **Window** / **Show View** / **Other** e escolha a visão **Servers** na pasta **Server**. Clique com o botão direito no espaço em branco da aba que se abrirá e escolha **New** / **Server**. Na janela **New Server**, escolha novamente **Apache Tomcat v5.5** na pasta **Apache** e clique em **Finish**. O servidor aparecerá na visão **Servers** e você poderá controlá-lo.

IvyDE

Quando usamos bibliotecas e *frameworks* em nossos projetos, devemos configurar as dependências do nosso projeto no Eclipse. Antes disso, no entanto, é preciso obter as dependências. Muitas vezes, alguns *frameworks* dependem de outros, formando uma rede de dependências complicada de manter manualmente. Para evitar este problema, utilizamos o gerenciador de dependências Ivy.

O Ivy (**http://www.jayasoft.org/ivy**) é uma ferramenta que integra-se com o Ant (*build tool* que já vem instalada no Eclipse – **http://ant.apache.org/**). Ela permite que você monte um repositório de dependências (arquivos JAR) e descreva estas dependências em arquivos XML, especificando suas versões e sub-dependências. Assim, seu projeto especifica apenas os *frameworks* que necessita (e suas respectivas versões) e o Ivy cuida de baixar todos os JARs necessários para o diretório **lib/**.

Para instalar o Ivy no Eclipse, abra o Eclipse como administrador do sistema e clique em **Help** / **Software Updates** / **Find and Install...** Escolha a opção **Search for new features to install** e clique **Next >**. Na tela seguinte, clique no botão **New Remote Site**. Dê ao novo *site* remoto o nome **Jayasoft (Ivy)** e a URL **http://update.jayasoft.org**. O novo *site* será adicionado e automaticamente marcado. Clique **Finish** para terminar.

Na tela que ser abrirá após as devidas verificações apresentará um item chamado **Jayasoft (Ivy)**. Marque este item e clique **Next >**. Marque a opção **I accept the terms in the license agreement** após ler o contrato e clique **Next >** novamente. Clique **Finish** para iniciar o *download* e aguarde. Após o *download*, uma tela de verificação se apresentará. Clique no botão **Install All** para concluir a instalação. Por fim, responda **Yes** quando o eclipse perguntar se você deseja reiniciá-lo. Em seguida, feche-o e abra-o novamente como usuário normal.

Para usar o Ivy em nossos *builds* do Ant, é preciso informar ao Eclipse que ele deve utilizar o JAR do Ivy quando for efetuar seus *builds*. Clique em **Window** / **Preferences...** e selecione a opção **Ant** / **Runtime**. Selecione o item **Ant Home Entires (Default)** e clique no botão **Add External JARs**. Dentro do diretório do Eclipse, localize o arquivo **plugins/org.jayasoft.ivyde.eclipse_1.2.0/lib/ant/ivy.jar** e clique **OK**. Clique **OK** novamente para fechar a janela de preferências e pronto.

O Projeto "Bookshelf"

Com as ferramentas prontas para uso, vamos criar um projeto novo bem simples: um sistema de controle de empréstimos de livros para uso pessoal ou de pequenas bibliotecas, como as de laboratórios de informática. Chamamos este projeto de "Bookshelf" (estante, em inglês).

Para este projeto usaremos diversos *frameworks*. Na camada de apresentação, o SiteMesh[1] irá decorar nossas páginas HTML, enquanto o FreeMarker[2] processa modelos de página. No controle, o Struts[23] separa os dados de sua apresentação, integrando-se com os dois *frameworks* anteriores e também com o Spring Framework[4], que será usado para satisfazer as dependências entre as diversas camadas. Finalmente, para persistência usaremos o Hibernate[5] e o banco de dados HSQLDB[6].

Todos estes *frameworks* possuem uma série de arquivos de configuração que precisam ser escritos. Como esta tarefa é bastante tediosa, costuma-se ter um projeto "em branco" com a maioria das configurações prontas. AppFuse[7] e Equinox[8] são exemplos de projetos que propõem exatamente isso: servir como pontapé inicial para outras aplicações que utilizem um determinado conjunto de *frameworks*. Usaremos um projeto do Eclipse em branco (**`blankproject.zip`**) que você pode baixar no *site* **`http://www.inf.ufes.br/~vsouza/?q=node/3`**.

Novo Projeto *Web*

Inicie o Eclipse e clique no menu **`File`** / **`New`** / **`Project...`** (ou Ctrl+N) e selecione **`Dynamic Web Project`** dentro da pasta **`Web`**. Clique em **`Next >`**, preencha o **`Project name`** com **`Bookshelf`**, escolha **`Apache Tomcat v5.5`** como **`Target runtime`** e clique **`Finish`**.

O Eclipse informará que este tipo de projeto é associado com a perspectiva **`J2EE`** (ele provavelmente se encontra na perspectiva **`Java`**, que é a padrão de instalação). Clique **`Yes`** para mudar para a nova perspectiva. Se quiser alterar de perspectiva em outro momento, vá ao menu **`Window`** / **`Open Perspective`**.

Agora, copie os arquivos do projeto em branco por cima dos arquivos do projeto criado. Descompacte o arquivo **`blankproject.zip`** no diretório do seu novo projeto, sobrescrevendo todos os arquivos que já existirem. Em seguida, retorne ao Eclipse, clique com o botão direito na pasta do seu projeto e peça a opção **`Refresh`**. Uma série de erros será reportada na *view* **`Problems`**. Não se preocupe, trataremos estes erros em seguida.

Obtendo as Dependências

O projeto em branco não traz consigo os diversos JARs que são necessários para compilação e execução do mesmo. Este é o papel do Ivy, como já explicamos anteriormente. Dois arquivos que encontram-se na raiz do projeto serão usados nesta etapa: **`build.xml`** e **`ivy.xml`**. O primeiro é um *script* Ant que evoca o Ivy para *download* das dependências. O segundo é a configuração do Ivy, indicando quais JARs devem ser baixados. Lembre-se que as dependências das dependências são resolvidas automaticamente pelo Ivy segundo as informações do repositório.

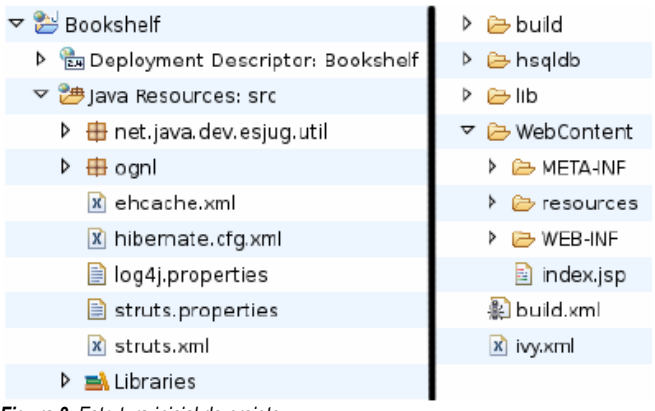

Figura 3. *Estrutura inicial do projeto.*

Se você abrir o arquivo **`build.xml`** notará que o repositório utilizado pelo projeto é o repositório do Laboratório de Engenharia de Software da Universidade Federal do Espírito Santo (**`http://labes.inf.ufes.br/ivy-rep/`**). Você pode visitar o repositório com seu navegador *Web* e ver mais informações sobre cada uma das dependências.

Finalmente, clique no arquivo **`build.xml`** com o botão direito e escolha **`Run As`** / **`Ant Build`**. Aguarde o Ivy baixar todas as dependências, que serão armazenadas no diretório **`lib`**, dentro do projeto. Atualize seu projeto (**`Refresh`**) para vê-las.

Em seguida, clique com o botão direito no nome do projeto e escolha **`Properties.`** Selecione a opção **`J2EE Module`**

1 http://www.opensymphony.com/sitemesh/
2 http://www.freemarker.org/
3 http://struts.apache.org/2.x/
4 http://www.springframework.org/
5 http://www.hibernate.org/
6 http://www.hsqldb.org/
7 https://appfuse.dev.java.net/
8 https://equinox.dev.java.net/

Dependencies no menu à esquerda e clique em **Add JARs...** Selecione todos os arquivos JAR que se localizam abaixo da pasta **lib** e clique **OK.** Aguarde um momento até que as bibliotecas apareçam na listagem e clique **OK** novamente para fechar a janela de propriedades do projeto. Neste momento, o projeto não deve apresentar mais nenhum erro na *view* **Problems** e deve ter a estrutura similar à da figura 3.

Configurações

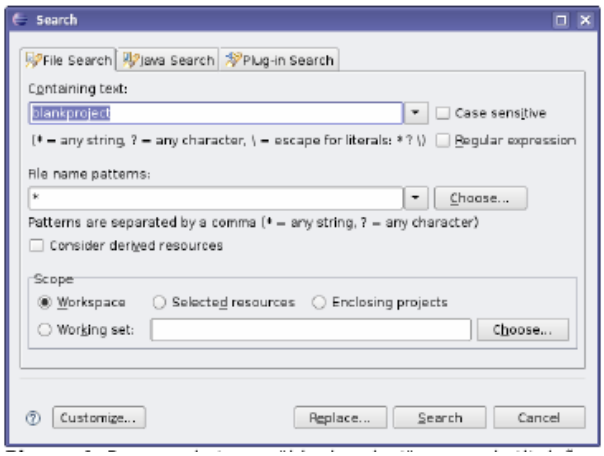

Figura 4. *Busca pelo termo "blankproject" para substituição.*

O projeto em branco traz diversas configurações prontas, porém algumas necessitam de ajustes, pois cada projeto tem um nome diferente, uma estrutura de pacotes diferente, classes diferentes, etc. Para facilitar, todos os pontos que precisam de ajustes possuem a palavra "blankproject". Portanto, clique no menu **Search** / **File...**, preencha os campos de busca como na figura 4 e clique em **Search**.

A *view* **Search** será apresentada com o resultado da pesquisa. Nove arquivos de configuração possuem o termo e deve ser editados de forma a se adequarem ao projeto **Bookshelf**. A tabela 1 resume o que representa cada arquivo e indica quais alterações devem ser efetuadas para esta adequação.

Arquivo	*Configuração*
hsqldb/server.properties	Configuração do banco de dados HSQLDB em modo servidor. Substitua `blankproject` pelo nome que você quiser dar a sua base de dados. Para nosso projeto, usaremos `bookshelf`.
hsqldb/webserver.properties	Idem, porém configura o HSQLDB em modo servidor *Web*. Este modo é ideal para redes com restrições de acesso impostas por *firewalls*.
src/hibernate.cfg.xml	Configuração do framework de persistência Hibernate. Quando as classes de negócio forem criadas, será preciso declarar seu mapeamento. Por enquanto, no entanto, não mude nada neste arquivo.
src/log4j.properties	Configuração da ferramenta de *logging* Log4J. A linha 9 indica em que arquivo serão gravadas as mensagens de *log*. Substitua seu valor por `/tmp/bookshelf.log`, se você usa Linux, ou `C:\\Temp\\bookshelf.log` se usa Windows.
WebContent/WEB-INF/decorators/defaultl.ftl	Modelo de leiaute padrão para as páginas *Web*. Este arquivo é usado pelo SiteMesh para decorar as páginas. Substitua `blankproject` por `Bookshelf` para que apareça no título de cada uma das páginas. Posteriormente veremos como incrementar este modelo.
WebContent/WEB-INF/pages/home.ftl	Página inicial do sistema, que aparecerá caso tudo corra bem. Substitua o título da página por `Home` e `blankproject` por `Bookshelf` na frase de boas-vindas.
WebContent/WEB-INF/applicationContext-core.xml	Arquivo de configuração do Spring Framework que conterá a definição das classes de serviço e DAO do pacote `core` (único pacote do sistema Bookshelf). No caso de sistemas com múltiplos pacotes, sugerimos a criação de um arquivo separado para cada um. Por enquanto não mude nada neste arquivo.
WebContent/WEB-INF/applicationContext-resources.xml	Arquivo de configuração do Spring Framework que contém a definição da infra-estrutura da aplicação, injetada automaticamente pelo *framework*. Na configuração do *data source* (linha 17), substitua `blankproject` pelo nome da base de dados, no caso, `bookshelf`. Na linha 42, substitua `blankproject.packagename` pelo nome do pacote da sua aplicação. Usaremos `net.java.dev.esjug.bookshelf`. Isso faz com que o Spring utilize AOP para gerenciar transações automaticamente nas classes de serviço (pacote `application`).
WebContent/WEB-INF/web.xml	Configuração da aplicação *Web*. Altere apenas o nome da aplicação, na linha 4. As demais configurações já estão apropriadas para o conjunto de *frameworks* escolhidos.

Tabela 1. Arquivos de configuração do projeto.

Execução

Finalizada a configuração inicial, podemos executar nossa *WebApp* para ver se tudo correu bem. Sempre antes de executar a aplicação, no entanto, é preciso iniciar o banco de dados. Para fazer isso, utilize um dos *scripts* presentes na pasta **`hsqldb`** (**`hsqldb-server`** para modo servidor e **`hsqldb-webserver`** para modo servidor *Web*, em versões Windows – **`.bat`** – e Linux – **`.sh`**). Se preferir usar o modo servidor *Web*, é necessário trocar a URL de acesso ao banco de dados na linha 17 do arquivo **`applicationContext-resources.xml`** para **`jdbc:hsqldb:http://localhost:8090/bookshelf`**.

Abra a *view* **`Servers`**, clique com o botão direito no servidor **`Tomcat`** e escolha **`Add and Remove Projects`**. Seu projeto deve estar na lista de **`Avaliable Projects`**. Mova ele para a lista de **`Configured Projects`** e clique em **`Finish`**. Com o servidor *Web* selecionado, clique no ícone verde para iniciá-lo. É recomendado monitorar o arquivo de *log* neste momento para detectar qualquer problema. Após o início do servidor (o Eclipse mostrará na *view* **`Console`** uma mensagem similar a esta: `INFO: Server startup in 10581 ms`), abra seu navegador *Web* e acesse o endereço **`http://localhost:8180/Bookshelf/`** (a porta – 8180 – pode variar, dependendo da configuração do seu ambiente). Seu navegador deve exibir a página inicial do sistema, que contém apenas uma mensagem de boas-vindas e congratulações.

Aconselhamos o acompanhamento do *log* da aplicação, pois as mensagens apresentadas pelo Tomcat no Console do Eclipse não informam a causa real dos erros, quase sempre fazendo com que o desenvolvedor desista de criar a aplicação após várias horas frustradas de busca na Internet. O arquivo de *log* do Bookshelf foi configurado no arquivo `log4j.properties`, e aconselhamos a utilização da ferramenta *tail* do Linux (`tail -f /tmp/bookshelf.log`) ou do WinTail para o Windows (`http://www.baremetalsoft.com/wintail/`).

Compreendendo os *Frameworks*

Antes de prosseguirmos na construção de nosso sistema de exemplo, é interessante que o leitor entenda como os *frameworks* funcionam. Nesta seção daremos uma visão geral da proposta de cada um deles e apresentaremos arquivos de configuração existentes no projeto em branco relacionados a estes *frameworks*. Para saber mais sobre cada *framework*, consulte as referências ao final do artigo.

Struts²

O Struts² é o controle central da aplicação *Web*. Ele pertence a uma categoria de *frameworks* conhecida como *MVC Frameworks* ou *Front Controller Frameworks*. Seu objetivo é prover um componente que centralize todas as requisições dos clientes (navegadores *Web*), delegue a classes específicas a execução da lógica de negócio e gere as páginas *Web* de resultado como resposta, tudo mediante arquivos de configuração. A figura 5 ilustra este funcionamento.

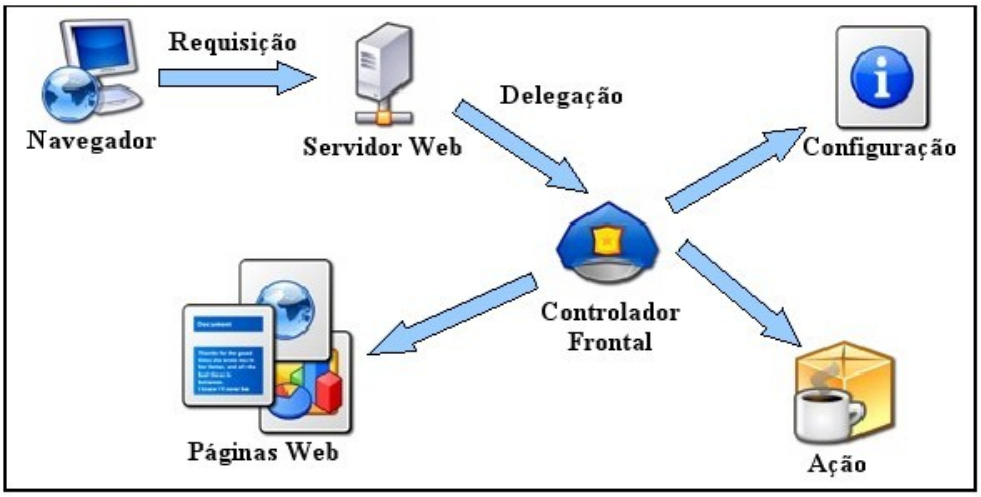

Figura 5. *funcionamento do padrão arquitetural Front Controller.*

Já podemos notar o Struts² em nossa pequena aplicação *Web*. A listagem 1 mostra um trecho do arquivo `web.xml`, que configura nossa *WebApp*. Um filtro do Struts² e um Servlet do framework Ajax DWR foram configurados. O primeiro faz com que todas as requisições (mapeamento `/*`) recebidas pelo servidor *Web* sejam filtradas pelo Struts² (que, por padrão, trata apenas requisições que terminam em `.action`). É neste momento que o *framework* toma conta da situação e faz seu trabalho. O segundo direciona todas as requisições que começarem com /dwr (mapeamento `/dwr/*`) para o Servlet do DWR. O Struts² utiliza este mapeamento para construção de scripts de

validação de formulários que utilizam AJAX.

```
<!-- Struts: the MVC framework (see src/struts.xml). -->
<filter>
  <filter-name>strutsFilter</filter-name>
  <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class>
</filter>

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

<!-- DWR Servlet: required if Struts AJAX validation is used. -->
<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>

<!-- Maps the servlets declared above to the URLs defined below. -->
<servlet-mapping>
  <servlet-name>dwr</servlet-name>
  <url-pattern>/dwr/*</url-pattern>
</servlet-mapping>
```

Listagem 1. *Trechos do arquivo WebContent/WEB-INF/web.xml referentes ao framework Struts[2].*

O filtro do Struts[2] possui sua própria configuração no arquivo `struts.xml`. Este arquivo, mostrado na listagem 2, indica que as configurações *default* do Struts2 devem ser importadas e cria um pacote de ações *default* para nossa aplicação. Neste pacote, definimos o FreeMarker como resultado padrão e configuramos a ação `home`, que não executa nenhuma lógica e simplesmente exibe a página `home.ftl`, mostrado na listagem 3. Veja que esta página contém a mesma mensagem de boas-vindas que você viu ao executar a *WebApp* no passo anterior. Isso ocorre porque a página `index.jsp` (listagem 4) redireciona automaticamente para `home.action`.

```
<!-- Struts actions configuration. -->
<struts>
  <!-- Import default configurations. -->
  <include file="struts-default.xml" />

  <!-- Home package: includes general actions (not related to any specific use case). -->
  <package name="default" extends="struts-default">
     <!-- Configures FreeMarker as the default result. -->
     <result-types><result-type name="freemarker" default="true"
           class="org.apache.struts2.views.freemarker.FreemarkerResult" /></result-types>

     <!-- Default action (used when an unexisting action name is given). -->
     <default-action-ref name="home" />

     <!-- Home: displays the home page. -->
     <action name="home"><result>/WEB-INF/pages/home.ftl</result></action>
  </package>
</struts>
```

Listagem 2. *Arquivo src/struts.xml que configura o Struts².*

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Home</title>
  <meta http-equiv="Content-Type" content="application/xhtml+xml; charset=utf-8" />
</head>
<body>
  <h1>Welcome to the Bookshelf.</h1>
  <p>Congratulations: your project is running.</p>
</body>
</html>
```

Listagem 3. *Trecho do arquivo WebContents/WEB-INF/pages/home.ftl – página inicial do sistema.*

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

Listagem 4. Arquivo WebContents/index.jsp – redireciona para a ação home.

O *framework* Struts² possui um último arquivo de configuração chamado `struts.properties`. Neste arquivo (veja listagem 5) que permite que sejam especificadas configurações diversas. Note que a última linha deste arquivo indica o Spring como o *container* de injeção de dependências a ser usado.

```
## Struts configuration.

# Default locale for this web application.
struts.locale = pt_BR

# Developer mode: indicates if modifications should be reloaded and if errors that
# could be ignored should be reported.
struts.devMode = true

# Maximum size for upload (10MB = 10485760).
struts.multipart.maxSize = 10485760

# Uses Spring Framework as Dependency Injection container.
struts.objectFactory = spring
```

Listagem 5. *Arquivo src/struts.properties – página inicial do sistema.*

O projeto em branco que utilizamos traz diversas configurações prontas para o Struts², restando a nós agora escrever as ações para nosso projeto.

SiteMesh

SiteMesh é um *framework* decorador. Seguindo o padrão de projeto *Decorator*, seu objetivo é interceptar respostas do servidor *Web* ao cliente e aplicar um leiaute padronizado à resposta HTML crua vinda do Struts[2] (veja figura 6). Desta maneira, e contando com uma configuração bastante flexível, podemos aplicar um mesmo leiaute a várias páginas de forma fácil e até mesmo alterar o desenho de todas as páginas da nossa *WebApp* em pouco tempo.

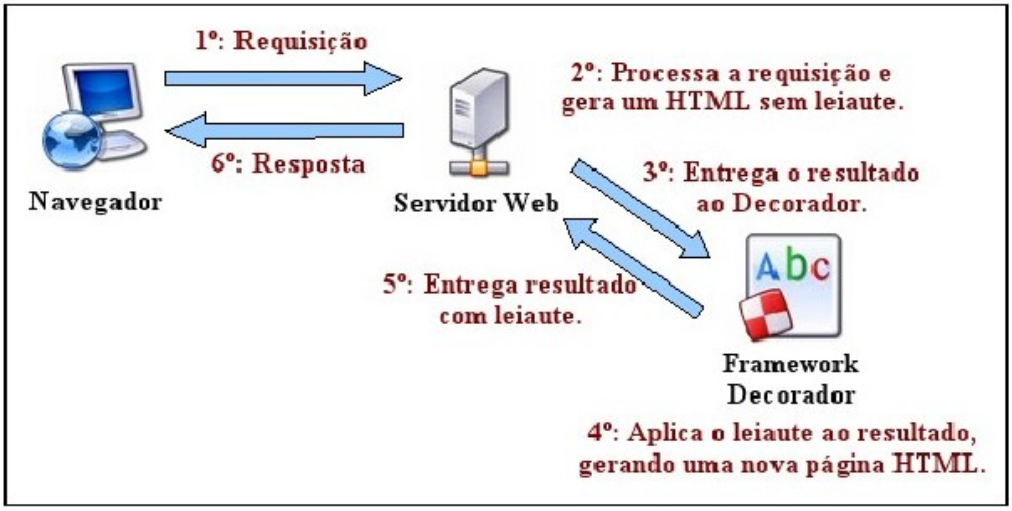

Figura 6. *funcionamento de um framework decorador.*

O SiteMesh é ativado pelo servidor *Web* a partir de uma configuração no arquivo web.xml. Os trechos referentes a esta configuração são exibidos na listagem 6. Note que, para que os dados que o Struts[2] disponibiliza estejam disponíveis para o decorador do SiteMesh, é preciso adicionar um filtro extra, o `strutsCleanup`. Observe com atenção a ordem dos filtros no arquivo `web.xml`, pois é algo bastante importante.

```
<!-- Struts Cleanup: required if SiteMesh is being used. -->
<filter>
   <filter-name>strutsCleanup</filter-name>
   <filter-class>org.apache.struts2.dispatcher.ActionContextCleanUp</filter-class>
</filter>

<!-- Sitemesh: applies layout to raw web pages using decorators. -->
<filter>
   <filter-name>sitemesh</filter-name>
   <filter-class>org.apache.struts2.sitemesh.FreeMarkerPageFilter</filter-class>
</filter>

<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>
```

Listagem 6. Trechos do arquivo WebContent/WEB-INF/web.xml referentes ao framework FreeMarker.

É possível que você já tenha percebido o SiteMesh em ação quando executou o projeto Bookshelf pela primeira vez. Repare que na página `home.ftl` (listagem 3) o título da página é *Home* e há uma mensagem de boas-vindas e congratulações. No entanto, ao executar a *WebApp*, vemos que o título é *Bookshelf :: Home* e além das mensagens de boas-vindas há também uma nota ao final da página.

O que acontece é que o SiteMesh pega o resultado cru vindo do processamento de `home.ftl` e aplica a este resultado um decorador, de acordo com as configurações existentes no arquivo `decorators.xml`. Este último, exibido na listagem 7, indica que os decoradores encontram-se na paste `/WEB-INF/decorators`, que os arquivos dentro da pasta `/resources` não devem ser decorados (imagens, folhas de estilo, scripts JavaScript, etc.) e que todas as páginas devem ser decoradas pelo decorador

`default.ftl`, que mostramos na listagem 8.

```
<!-- Sitemesh decorator configuration. -->
<decorators defaultdir="/WEB-INF/decorators">
  <!-- Resources should not be decorated. -->
  <excludes><pattern>/resources/*</pattern></excludes>

  <!-- Default decorator - used in most pages. -->
  <decorator name="default" page="default.ftl">
     <pattern>/*</pattern>
  </decorator>
</decorators>
```

Listagem 7. *Arquivo WebContents/WEB-INF/decorators.xml – configuração do SiteMesh.*

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Bookshelf :: ${title}</title>
  <meta name="description" content="" />
  <meta name="keywords" content="" />
  <meta name="author" content="" />
  <meta http-equiv="Content-Type" content="application/xhtml+xml; charset=utf-8" />
</head>
<body>
  ${body}
  <p> </p>
  <p>This is the layout that comes with the blank project. I suggest you visit
     <a href="http://www.oswd.org" target="_blank">OSWD</a> and change it.</p>
</body>
</html>
```

Listagem 8. Arquivo WebContents/WEB-INF/decorators/default.ftl – decorador padrão da aplicação.

As variáveis `${title}` e `${body}` indicam onde serão inseridos, respectivamente, o título e o corpo da página crua recebida do Struts[2] como resposta a ser enviada ao cliente. O SiteMesh faz isso momentos antes do servidor *Web* enviar a resposta, resultando na página que vemos ao executarmos nossa aplicação *Web* Bookshelf.

FreeMarker

FreeMarker é um *template engine*. O termo, que não possui tradução direta, define uma ferramenta que permite a criação de documentos de diversos tipos a partir de modelos (*templates*). A figura 7 exemplifica como um *template engine* funciona.

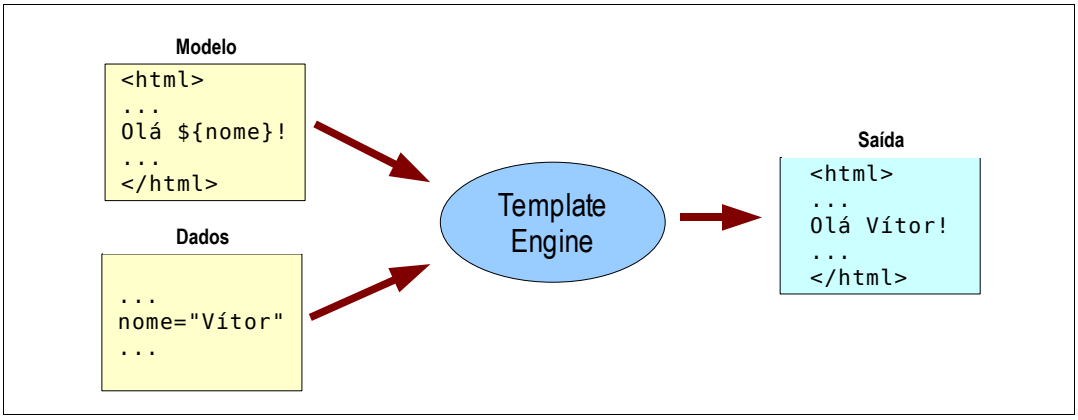

Figura 7. *funcionamento de um template engine.*

Em nossa *WebApp* utilizamos FreeMarker em duas situações que já apresentamos: na construção de decoradores para o SiteMesh e no desenvolvimento das páginas *Web* que o Struts[2] processará para dar o resultado. A utilização de um *template engine* permite uma maior separação entre os dados e sua apresentação, além de trazer algumas facilidades na hora de construir as páginas (sugiro que o leitor dedique algum tempo conhecendo os *built-ins* do FreeMarker!).

O Struts2 possui uma forte integração com o FreeMarker, utilizando-o para definição de suas *tags*, além de disponibilizar automaticamente todos os dados da ação para o *template*, quando usamos o resultado `freemarker` no `struts.xml` (veja listagem 2). Se a ação do Struts2 disponibiliza uma informação, digamos, por meio do método `getNome()`, basta colocar a interpolação `${nome}` que o método será chamado e seu resultado impresso na página.

Spring Framework

O Spring é um *framework* de aplicação em camadas para Java SE/EE. Surgiu a partir da publicação do livro Expert One-on-One J2EE Design and Development, de Rod Johnson em 2002, oferecendo uma alternativa ao padrão EJB 2.1, vigente na época. Mesmo depois de lançada a versão 3.0 dos EJBs, que corrigiu várias falhas apontadas por Johnson como motivação para construção do Spring, o *framework* permanece como uma alternativa bastante popular ao uso de um servidor de aplicações completo para se obter serviços como gerência de transações, acesso remoto, etc.

No projeto Bookshelf, usaremos o Spring para injeção de dependências e gerenciamento de transações declarativas via AOP. Primeiro, é preciso configurá-lo no arquivo `web.xml` (veja listagem 9).

```
<!-- Indicates where Spring should look for its configuration files. -->
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/applicationContext-*.xml</param-value>
</context-param>

<!-- Spring's Hibernate "open session in view" filter. -->
<filter>
  <filter-name>hibernateFilter</filter-name>
  <filter-class>org.springframework.orm.hibernate3.support.OpenSessionInViewFilter</filter-class>
   <init-param>
      <param-name>singleSession</param-name>
      <param-value>false</param-value>
   </init-param>
</filter>

<!-- Spring filter: provides Spring's bean factory, AOP services, etc. -->
<filter>
  <filter-name>springFilter</filter-name>
  <filter-class>org.springframework.web.filter.RequestContextFilter</filter-class>
</filter>

<!-- Applies the filters declared above in the order defined below. -->
<filter-mapping>
  <filter-name>hibernateFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
<filter-mapping>
  <filter-name>springFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>

<!-- Spring's listener. -->
<listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
```

Listagem 9. Trechos do arquivo WebContent/WEB-INF/web.xml referentes ao framework Spring.

- Os arquivos de configuração do Spring devem ser encontrados dentro da pasta `WEB-INF`, começarem com `applicationContext-` e terminarem com a extensão `.xml`;
- O Spring deve gerenciar transações com o Hibernate, usando o padrão *Open Session in View*. Neste padrão, a seção do Hibernate é aberta quando o usuário faz a requisição e fechada quando o servidor *Web* envia a resposta;
- Um filtro e um *listener* do Spring também devem ser configurados para que a injeção de dependências funcione integrada com a criação das ações do Struts2.

Dentro do padrão `/WEB-INF/applicationContext-*.xml`, encontramos dois arquivos já preparados no projeto em branco: applicationContext-resources.xml (listagem 10) e applicationContext-core.xml (listagem 11). O primeiro contém configurações do Hibernate

e do gerenciamento de transações via AOP. O segundo contém exemplos comentados de definições de *beans* do Spring para Injeção de Dependências.

```
<!-- The data source: pool of connections required by Hibernate to connect to the database. -->
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
  <property name="driverClassName" value="org.hsqldb.jdbcDriver"/>
  <property name="url" value="jdbc:hsqldb:hsql://localhost/bookshelf"/>
  <property name="username" value="sa"/>
  <property name="password" value=""/>
  <property name="maxActive" value="100"/>
  <property name="maxIdle" value="30"/>
  <property name="maxWait" value="1000"/>
  <property name="defaultAutoCommit" value="true"/>
  <property name="removeAbandoned" value="true"/>
  <property name="removeAbandonedTimeout" value="60"/>
</bean>

<!-- Hibernate's session factory. -->
<bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
  <property name="dataSource" ref="dataSource" />
  <property name="configurationClass">
     <value>org.hibernate.cfg.AnnotationConfiguration</value>
  </property>
  <property name="configLocation"><value>classpath:/hibernate.cfg.xml</value></property>
</bean>

<!-- The transaction manager. -->
<bean id="transactionManager"
        class="org.springframework.orm.hibernate3.HibernateTransactionManager">
  <property name="sessionFactory"><ref local="sessionFactory" /></property>
</bean>

<!-- An AOP advice that applies transaction management. -->
<tx:advice id="transactionAdvice">
  <tx:attributes><tx:method name="*" /></tx:attributes>
</tx:advice>
<aop:config>
  <aop:advisor pointcut="execution(* net.java.dev.esjug.bookshelf.application.*.*(..))" advice-
                ref="transactionAdvice" />
</aop:config>
```

Listagem 10. Trechos do arquivo WebContent/WEB-INF/applicationContext-resources.xml.

Na listagem 10 podemos ver:

- A definição do *bean* `dataSource`, indicando todas as configurações de banco de dados que o Hibernate precisa para efetuar a persistência dos objetos;
- A definição do *bean* `sessionFactory`, a fábrica de sessões do Hibernate, que as classes que fazem persistência de dados precisam para recuperar, gravar e excluir objetos de domínio do banco de dados. Note que a fábrica é configurada para utilizar anotações e ler as configurações em `hibernate.cfg.xml`;
- A definição do *bean* `transactionManager`, o gerenciador de transações do Spring para o Hibernate. O gerenciador de transações funciona via AOP, configurada logo a seguir;
- A configuração de AOP, indicando que todos os métodos públicos de todas as classes do pacote `net.java.dev.esjug.bookshelf.application` devem ser transacionais. Isso significa que antes de cada método será aberta uma transação e após seu encerramento a mesma sofrerá *rollback* ou *commit*, dependendo se uma exceção foi lançada ou não. Neste pacote ficarão as classes de aplicação, que implementam os casos de uso.

A listagem 11 traz exemplos comentados de definição de DAOs e classes de aplicação. Veremos mais sobre estas classes posteriormente. A idéia é que haja um arquivo destes para cada pacote da aplicação. Como o Bookshelf só tem um pacote (que chamamos de *core*), só há este arquivo, além do *resources*, que define recursos gerais da aplicação.

```
<!-- Example of declaration of a DAO class:
<bean id="blankProjectDAO" class="blankproject.packagename.HibernateSpringBlankProjectDAO">
  <property name="sessionFactory"><ref bean="sessionFactory" /></property>
</bean>
-->

<!-- Example of declaration of a service class that depends on the DAO class above:
<bean id="blankProjectService" class="blankproject.packagename.BlankProjectServiceImpl">
  <property name="blankProjectDAO"><ref local="blankProjectDAO" /></property>
</bean>
-->
```

Listagem 11. Trechos do arquivo WebContent/WEB-INF/applicationContext-core.xml.

Hibernate

Por fim, o Hibernate fará a persistência transparente dos objetos num banco de dados relacional (no nosso caso, o HSQLDB). O Hibernate é o *framework* de mapeamento objeto/relacional (ORM – *Object Relational Mapping*) mais conhecido e mais usado pela comunidade Java.

Como já vimos, sua configuração está vinculada à configuração do Spring. Restam apenas dois arquivos específicos do Hibernate: `hibernate.cfg.xml` (configura vários aspectos da persistência) e `ehcache.xml` (configuração da ferramenta de *cache* utilizada pelo Hibernate). Mostraremos apenas o primeiro, na listagem 12.

```
<hibernate-configuration>
  <session-factory>
     <!-- SQL dialect -->
     <property name="dialect">org.hibernate.dialect.HSQLDialect</property>

     <!-- Enable Hibernate's automatic session context management -->
     <property name="current_session_context_class">thread</property>

     <!-- Disable the second-level cache  -->
     <property name="cache.provider_class">org.hibernate.cache.EhCacheProvider</property>
     <property name="cache.use_query_cache">true</property>
     <property name="cache.use_second_level_cache">true</property>

     <!-- Echo all executed SQL to stdout -->
     <property name="show_sql">false</property>

     <!-- Drop and re-create the database schema on startup -->
     <property name="hbm2ddl.auto">update</property>

     <!-- Mappings. -->
     <!--  mapping class="blankproject.packagename.ClassName" / -->
  </session-factory>
</hibernate-configuration>
```

Listagem 12. Arquivo src/hibernate.cfg.xml, configuração do framework Hibernate.

Os pontos que gostaríamos de destacar neste arquivo são: a escolha do `dialect`, que deve corresponder ao dialeto do banco de dados que estamos utilizando, a geração automática das tabelas no banco de dados (opção `hbm2ddl.auto`) e o exemplo comentado de mapeamento no final do arquivo, que deve ser substituído pelos mapeamentos reais após criadas as classes de domínio.

Conclusões

Nesta primeira parte da série de artigos sobre desenvolvimento *Web* no Eclipse, vimos como instalar diversas ferramentas que, combinadas, facilitam o trabalho de desenvolver e testar aplicações *Web*. Começamos nosso projeto Bookshelf a partir de um projeto em branco e compreendemos o papel de cada *framework* envolvido, detalhando seus arquivos de configuração.

O resultado desta primeira parte está disponível para *download* no *site* **`http://www.inf.ufes.br/~vsouza/?q=node/3`**. O código-fonte pode ser importado para um novo *Dynamic Web Project* do Eclipse como fizemos com o projeto em branco, explicado anteriormente.

Na próxima parte, continuaremos o desenvolvimento da *WebApp* Bookshelf utilizando os recursos dos *frameworks* e também da IDE Eclipse, terminando com uma aplicação simples desenvolvida em pouquíssimo tempo, graças a toda a infra-estrutura provida pela utilização dos vários *frameworks*.

Referências:

- Documentação do Struts[2]: http://struts.apache.org/2.x/docs/home.html;
- Documentação do SiteMesh: http://www.opensymphony.com/sitemesh/index.html;
- Documentação do FreeMarker: http://www.freemarker.org/docs/index.html;
- Documentação do Spring Framework: http://www.springframework.org/documentation;
- Documentação do Hibernate: http://www.hibernate.org/5.html;
- Grupo de Usuários de Java do Estado do Espírito Santo: http://esjug.dev.java.net/.