

Introdução ao Webwork

Samuel Mota

Aprenda a usar um framework MVC desenvolvendo o login de um sistema com o Webwork utilizando seus principais recursos.

Introdução

Objetivo

O objetivo deste tutorial é apresentar o maior número de recursos existentes no Webwork de forma fácil ao implementar um login que valide o usuário e dê a ele acesso aos recursos do sistema.

Para manter a simplicidade do tutorial não faremos a conexão com o banco de dados, apesar de ser uma tarefa básica e essencial para este sistema.

É importante que à medida que for lendo o tutorial o leitor se atente aos termos utilizados e procure se aprofundar, principalmente nos patterns, pois saber o que está por trás do framework muitas vezes o ajudará a entender o porquê de muitas características.

Banco de Dados

Existem muitas abordagens possíveis para permitir o gerenciamento e uso de conexões, as mais documentadas utilizam o Hibernate (<http://www.hibernate.org>) como framework de mapeamento Objeto/Relacional.

Uma abordagem muito interessante que utiliza *interceptors* (IoC) para esse gerenciamento está documentada no Wiki:

<http://wiki.opensymphony.com/space/Chaining+Interceptor>

Opensymphony

O framework foi desenvolvido e é mantido com o apoio de diversas empresas que diretamente contribuem ao grupo denominado Opensymphony (<http://www.opensymphony.com/>) e que abriga diversos projetos open-source.

Muitos desses projetos tem alguma relação com o Webwork, na maioria das vezes complementando suas funcionalidades para criar uma plataforma extremamente ágil de desenvolvimento de projetos.

O webwork está intimamente ligado ao Xwork que é a base de suas funcionalidades, o Xwork é quem controla a execução em si de todas as ações do sistema por isso será muito citado no tutorial.

Outro projeto que pode ser utilizado em conjunto e irá aumentar sua produtividade é o SiteMesh, não se esqueça ;-).

Documentação

O Webwork não tem uma documentação oficial bem estruturada. Sua principal fonte de consulta será o Wiki disponível em

<http://wiki.opensymphony.com/space/WebWork2>

A melhor fonte de suporte ao Webwork é a lista oficial do desenvolvimento disponível no portal Java.net: <https://webwork.dev.java.net/>

Conceitos e Recursos

Esta seção é para aqueles que queiram entender as opções que existem para utilizar diversos recursos do framework, se você quiser pode ir direto à seção **Aplicação** e voltar para ler esta seção depois.

Xwork

É uma implementação genérica do Command Pattern e é totalmente desacoplado do ambiente Web o que facilita o testes de suas ações.

Ele provê ao processamento das ações uma poderosa linguagem de expressões conhecida como OGNL, um container de IoC, interceptadores, conversão de tipos e um framework de validação.

Webwork

É o framework MVC propriamente dito. É construído totalmente baseado no Xwork com um conjunto de interceptors, results e dispatchers e provê suporte a camada de visualização (View) possibilitando o uso de praticamente qualquer framework como JSP, Velocity, FreeMarker, etc.

ActionSupport

Na distribuição do framework existe a classe `com.opensymphony.xwork.ActionSupport` que implementa diversas interfaces que são usadas pelo framework para prover a maioria dos serviços que desejamos como internacionalização, tratamento e validação de entrada e o comportamento padrão de uma action.

Todas as nossas actions herdarão estes recursos dessa classe, é possível alterar o comportamento do framework e suas características não utilizando esta classe como superclasse, mas este não é nosso objetivo e muito provavelmente você também não precisará.

Interceptors

A maioria dos recursos existentes no Webwork foram codificados como *Interceptors* que gerenciam o processamento passo a passo da requisição e resposta.

Depois de registrados os interceptors podem ser aplicados a ação conforme a necessidade de utilização de cada um.

O framework têm a facilidade de configurarmos pilhas de interceptor e provê inclusive duas pilhas padrão com os serviços básicos que precisaremos.

Validação

O suporte a validação de entradas no Webwork pode ser feito através da implementação de validadores gerenciados pelo Xwork ou da implementação do método `validate()` na action. Pode-se utilizar os dois meios em conjunto criando um suporte de validação extremamente eficiente.

Workflow Interceptor

O método `validate()` não tem retorno nenhum (é void) e, assim como os validadores, não irá impedir a execução da action.

Na `validationWorkflowStack` existe um interceptor chamado `workflow` que é responsável por verificar se existe algum erro nos mapas de erro e se existir interrompe a execução da action retornando diretamente INPUT.

Lembre-se disso quando não for utilizar a `validationWorkflowStack` e quiser fazer validações.

Existem diversos validadores implementados na distribuição e para desenvolver um que ainda não existe basta registra-lo no arquivo `validators.xml` e implementa-lo seguindo as instruções em <http://wiki.opensymphony.com/comments/Xwork+Validation+Framework>

O arquivo `validators.xml` deve ser encontrado na raiz do classpath (`WEB-INF\classes`).

Para configurar a validação criamos arquivos xml com o conteúdo conforme explicado em <http://wiki.opensymphony.com/comments/Xwork+Validation+Framework>.

Esses arquivos são processados seguindo a seguinte seqüência:

- Busca o arquivo com o **NomeDaClasseDaAction-validation.xml** e caso não encontre faz a mesma composição com o nome das super classes da action até que encontre `java.lang.Object`;
- Busca o arquivo com o **NomeDaClasseDaAction-NomeDaAction-validation.xml** onde `NomeDaAction` é o alias da action e irá também fazer a mesma composição com as super classes.

Internacionalização

O webwork oferece ótimo suporte à internacionalização do sistema, isso permite que você crie arquivos, um para cada idioma que desejar suportar, contendo todas as mensagens que utilizará no sistema. Desde as mensagens de erro na validação, até os textos de conteúdo do seu sistema podem ser automaticamente localizados baseado no idioma do usuário.

O webwork se baseia no Locale informado na requisição http, isso normalmente significa que o locale é o do sistema operacional, embora possa ser configurado pelo usuário.

Os arquivos de localização utilizados pelo framework a cada processamento seguem o seguinte padrão:

- Busca a chave em um arquivo `.properties` com o nome da classe da action (**ActionClass.properties**) no mesmo endereço da classe (mesma package), se não encontra -la procura o mesmo padrão para as super classes da action até que encontre `java.lang.Object`;
- Caso não tenha sucesso faz o mesmo processo anterior concatenando ao nome da classe o locale com um underscore. (ex: para usuário com locale `pt_BR` procurará **ActionClass_pt_BR.properties**);
- Se ainda assim não encontrar buscará nos arquivos de recursos registrados como padrão.

Ao iniciar o webwork registra os seguintes arquivos de recursos para recuperar as mensagens: `com/opensymphony/webwork/webwork-messages` e `com/opensymphony/xwork/xwork-messages`.

Componentes

O Xwork implementa IoC (Inversion Of Control) através da Injeção de Interface (Tipo 1), ou seja, para dizer que determinada action irá utilizar um componente basta que esta action implemente uma interface específica.

Introdução ao IoC

Leia o artigo no portal Java.net:

<http://today.java.net/pub/a/today/2004/02/10/ioc.html>

A configuração dos componentes é feita no arquivo `components.xml` que deve estar na raiz do classpath (`WEB-INF\classes`).

O tempo de vida dos componentes é definido pelo escopo que pode ser por requisição (*request*), sessão (*session*) ou aplicação (*application*), a definição desses escopos é análoga ao ciclo dentro da aplicação em um Webserver.

IoC Interceptor

O uso de componentes só é possível se o interceptor chamado *component* estiver aplicado à action, este interceptor não está em nenhuma pilha padrão de interceptors do framework.

View

O Webwork tem uma vasta biblioteca de tags que podemos utilizar em nossa camada de visualização.

Basicamente essas tags trabalham utilizando todos os recursos do framework como internacionalização, tratamento de erros e entradas.

Os templates das tags podem ser alterados ou sobrescritos, os que são utilizados por padrão estão no arquivo JAR do Webwork na package **template**.

Parâmetros

Lembre-se **sempre** que todos os parâmetros das tags são parseados pelo OGNL que busca executa as expressões, para não executa-las e ter o retorno exatamente com o conteúdo que escrever você deve colocar o texto entre '. Ex: 'textoNãoProcessado'

Aplicação

Agora que conhecemos os principais conceitos relacionados ao framework vamos desenvolver nossa aplicação, passo a passo. Você também pode baixar o projeto (Eclipse) no link de recursos na página do tutorial.

Instalação e Ambiente

No mínimo você precisará de um servlet container, para este artigo estamos utilizando o Jakarta Tomcat 5.0.19 (na verdade a versão não fará grande diferença), faça o download no site oficial <http://jakarta.apache.org/tomcat> e instale seguindo as instruções em <http://jakarta.apache.org/tomcat/tomcat-5.0-doc/setup.html>.

Tomcat e Servlets

Veja o tutorial de introdução ao Tomcat e Servlets no GUJ:
http://www.guj.com.br/user.article.get.chain?page=1&article_id=9

Agora precisamos pegar o framework, faça o download do arquivo webwork-2.0.zip em <https://webwork.dev.java.net/servlets/ProjectDocumentList> e descompacte-o onde preferir.

Crie um projeto em sua IDE preferida que seja uma aplicação no Tomcat, copie todos os arquivos com extensão jar que estão no subdiretório **lib/core** no diretório onde você descompactou o Webwork e também o arquivo **webwork-2.0.jar** e coloque-os no diretório **WEB-INF\lib** de sua aplicação.

web.xml

Para utilizarmos o framework precisamos configurar nossa aplicação no Tomcat para utiliza-lo. O código abaixo é o conteúdo total deste arquivo que deve estar no diretório WEB-INF da aplicação.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <display-name>WebWork Application</display-name>
  <filter>
    <filter-name>container</filter-name>
    <filter-
class>com.opensymphony.webwork.lifecycle.RequestLifecycleFilter</filter-class>
  </filter>
  <filter-mapping>
    <filter-name>container</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
```

```
<listener>
  <listener-
class>com.opensymphony.webwork.lifecycle.SessionLifecycleListener</listener-
class>
  </listener>

  <listener>
  <listener-
class>com.opensymphony.webwork.lifecycle.ApplicationLifecycleListener</listener-
class>
  </listener>

  <servlet>
    <servlet-name>webwork</servlet-name>
    <servlet-
class>com.opensymphony.webwork.dispatcher.ServletDispatcher</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet>
  <servlet-name>AplicationInitializer</servlet-name>
  <servlet-class>br.com.guj.system.Initializer</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>

  <servlet-mapping>
    <servlet-name>webwork</servlet-name>
    <url-pattern>*.action</url-pattern>
  </servlet-mapping>

  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>

  <taglib>
    <taglib-uri>webwork</taglib-uri>
    <taglib-location>/WEB-INF/lib/webwork-2.0.jar</taglib-location>
  </taglib>
</web-app>
```

Os principais pontos dessa configuração são:

- **Filtro:** configuramos o filtro chamado *container* e mapeamos todas as requisições do servidor para passarem por ele, este container é o responsável por controlar diversos recursos do framework;
- **Listeners:** configuramos alguns listeners que são os responsáveis por possibilitar o uso dos componentes, explicados anteriormente, que nos darão acesso a objeto com escopos;
- **Webwork Servlet:** o servlet chamado webwork é que controla a execução das actions por isso ele é mapeado para as requisições com extensão action;
- **Initializer Servlet:** nós vamos usar um servlet para inicializar algumas propriedades específicas de nossa aplicação, por isso criamos este servlet;
- **Taglib:** para utilizarmos a taglib disponível com o framework precisamos nomeá-la aqui.

validators.xml

Este arquivo deverá estar no WEB-INF\classes e o conteúdo inicialmente será apenas registrando os validadores do próprio webwork, se quiser utilizar outros validadores basta declará-los neste arquivo.

```
<validators>
  <validator
class="com.opensymphony.xwork.validator.validators.RequiredFieldValidator"/>
```

```
<validator name="requiredstring"
class="com.opensymphony.xwork.validator.validators.RequiredStringValidator"/>
<validator name="int"
class="com.opensymphony.xwork.validator.validators.IntRangeFieldValidator"/>
<validator name="date"
class="com.opensymphony.xwork.validator.validators.DateRangeFieldValidator"/>
<validator name="expression"
class="com.opensymphony.xwork.validator.validators.ExpressionValidator"/>
<validator name="fieldexpression"
class="com.opensymphony.xwork.validator.validators.FieldExpressionValidator"/>
<validator name="email"
class="com.opensymphony.xwork.validator.validators.EmailValidator"/>
<validator name="url"
class="com.opensymphony.xwork.validator.validators.URLValidator"/>
<validator name="visitor"
class="com.opensymphony.xwork.validator.validators.VisitorFieldValidator"/>
<validator name="conversion"
class="com.opensymphony.xwork.validator.validators.ConversionErrorFieldValidator"
"/>
</validators>
```

Initializer

Esta classe é um servlet que carregamos na inicialização do servidor para executar as ações que achamos necessária. No exemplo adicionamos outro arquivo de mensagens para ser utilizado pelo framework.

```
package br.com.guj.system;

import javax.servlet.http.HttpServlet;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import com.opensymphony.xwork.util.LocalizedTextUtil;

public class Initializer extends HttpServlet {

    private static final Log log = LogFactory.getLog(Initializer.class);

    public Initializer() {
        log.info("Inicializando sistema");
        LocalizedTextUtil.addDefaultResourceBundle("site-messages");
        log.info("Sistema inicializado!");
    }
}
```

Dica

Quando estiver utilizando o Hibernate este servlet é um bom lugar para carregar todos os mapeamentos do sistema pois não impactará no uso já que componentes do framework e/ou algumas outras abordagens utilizam lazy-evaluation e tornarão seu sistema extremamente lento para o primeiro usuário.

site-messages.properties

Este arquivo é nosso arquivo de mensagens globais conforme adicionamos no servlet de inicialização.

```
# Mensagens globais
pageFooter=Tutorial do GUJ
```

requiredField=Este campo é requerido!
outOfRangeField=O valor do campo deve estar entre \${min} e \${max}, o valor \${bar} é inválido!

User

Criamos agora nossa primeira classe que representará o usuário do sistema, ela contém todas as propriedades que atribuímos ao usuário.

```
package br.com.guj.models;

import java.io.Serializable;

/**
 * @author smota
 *
 * Representa um usuário
 */
public class User
    implements Serializable
{

    /* Dados de acesso */
    private String username;
    private Integer password;

    /* Dados pessoais */
    private String firstName;

    /**
     * @return Returns the username.
     */
    public String getUsername() {
        return username;
    }

    /**
     * @param username The username to set.
     */
    public void setUsername(String username) {
        this.username = username;
    }

    /**
     * @return Returns the firstName.
     */
    public String getFirstName() {
        return firstName;
    }

    /**
     * @param firstName The firstName to set.
     */
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    /**
     * @return Returns the password.
     */
    public Integer getPassword() {
        return password;
    }

    /**
     * @param password The password to set.
     */
}
```

```
    */
    public void setPassword(Integer password) {
        this.password = password;
    }
}
```

Esta classe será o modelo utilizado pela action de login.

SystemAction

Para facilitar futuras expansões do nosso exemplo criaremos uma classe abstrata que herdará diretamente da ActionSupport e obrigará que as actions do nosso sistema tenham algumas características consistentes com nosso projeto, especificamente, todas usarão o ModelDriven e todas utilizarão a seção do usuário.

```
package br.com.guj.actions;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import com.opensymphony.xwork.ActionSupport;
import com.opensymphony.xwork.ModelDriven;
import br.com.guj.components.UserSession;
import br.com.guj.components.UserSessionAware;

/**
 * @author smota
 *
 * Classe base para todas as Action do sistema.
 * - Exige que todas implementem o ModelDriven
 * - Adiciona suporte a identificação de usuário
 */
public abstract class SystemAction
    extends ActionSupport
    implements ModelDriven,
               UserSessionAware
{
    private static final Log log = LogFactory.getLog(SystemAction.class);

    /** Objeto armazenado na sessão com info do usuário */
    protected UserSession userSession;

    /** Construtor de inicialização */
    protected SystemAction() {
    }

    /**
     * @see br.com.guj.components.UserSessionAware#setUserSession(UserSession)
     */
    public void setUserSession(UserSession usr) {
        this.userSession = usr;
    }

    /**
     * @see com.opensymphony.xwork.ModelDriven#getModel()
     */
    abstract public Object getModel();
}
```

Agora nossas classes de actions não mais herdarão a ActionSupport e sim esta superclasse.

É importante notar que o objeto **userSession** será sempre setado com a chamada do método **setUserSession(UserSession usr)** que é feita pelo container IoC do framework e na sua configuração definimos que esse objeto tem o escopo da sessão do usuário, por isso nós o utilizaremos como nosso objeto que identifica o usuário e todas as suas propriedades.

Login

A classe de login é bastante simples e neste ponto deveríamos fazer o acesso ao banco de dados para validar, mas neste caso exigimos o usuário *guj* e a senha *1234*.

Alguns comentários adicionais:

- Criamos uma interface chamada **DisableSecurityAction** para que nosso interceptor (você verá mais para frente) de segurança não exija que o usuário esteja logado para executar esta ação, afinal esta ação é que irá cuidar do login;
- O model desta classe é o próprio **User** que vimos acima, isso significa que os campos na tela devem ter o mesmo nome das propriedades deste classe;
- Para utilizarmos o recurso de internacionalização as mensagens que adicionamos durante o processamento da action usam o método **getText(String chave)**;
- **addActionError** adiciona um erro global a action enquanto **addFieldError** adiciona um erro diretamente ligado ao campo, a diferença de uso está na forma que você usa esses erros. No caso das tags do próprio framework os erros de campo são adicionados acima dele.

```
package br.com.guj.actions;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import br.com.guj.models.User;
import br.com.guj.system.DisableSecurityAction;

/**
 * @author smota
 */
public class Login
    extends SystemAction
    implements DisableSecurityAction
{

    private static final Log log = LogFactory.getLog(Login.class);

    private User model = new User();

    public String execute() {

        // --> AQUI deveria validar de verdade o usuário
        if(!model.getUsername().equals("guj")) {
            addActionError(getText("login.invalidCredentials"));
            addFieldError("username", getText("login.unknownUser"));
            return ERROR;
        }

        if(model.getPassword().intValue()!=1234) {
            addActionError(getText("login.invalidCredentials"));
            addFieldError("password", getText("login.wrongPassword"));
            return ERROR;
        }

        //logou com sucesso? Põe o usuário na sessão.
        this.userSession.setUser(model);

        return SUCCESS;
    }
}
```

```
public Object getModel() {  
    return this.model;  
}  
}
```

Login.jsp

Este arquivo é responsável por montar a tela de login, apresentar os erros da action e também dos campos, exibe também os erros de conversão.

Observe que os erros nos campos aparecem em vermelho, isso é feito através da definição de um estilo que é usado na montagem do código html pelas tags do framework. Existem outros estilos, veja o fonte html.

Todos os textos dessa página são localizados e para permitir o uso do framework quando o usuário aponta direto para a página e não executa a action utilizamos a tag **i18n**.

É muito importante lembrar sempre na montagem da view utilizando as tags do framework que **todos** os parâmetros das tags podem ser expressões OGNL e serão parseados a menos que estejam entre aspas simples.

```
<%@ taglib prefix="ww" uri="webwork" %>  
<ww:i18n name="br/com/guj/actions/login" >  
  
<html>  
<head>  
    <title><ww:text name="login.title" /></title>  
</head>  
  
<style>  
.errorMessage {  
    color: red;  
}  
</style>  
  
<body>  
  
<p align="center"><ww:text name="login.saudacao" /></p>  
  
<ww:if test="hasErrors()" >  
    <p align="center">  
        ATENÇÃO: Verifique o(s) erro(s) abaixo.  
        <ww:iterator value="actionErrors" >  
            <li><ww:property value="getText(top)" />  
        </ww:iterator>  
        <ww:if test="hasFieldErrors()" >  
            <li> Verifique os erros em destaque no formulário!  
        </ww:if>  
    </p>  
</ww:if>  
  
<p align="center">  
<ww:form name="login" action="login.action" method="POST">  
    <ww:textfield label="getText('login.username')" name="username"  
value="username" size="15" />  
    <ww:password label="getText('login.password')" name="password"  
value="password" size="15" />  
    <ww:submit value="getText('login.enter')" />  
</ww:form>  
</p>  
  
<div align="center"><ww:text name="pageFooter" /></div>  
</body>
```

```
</html>
</ww:i18n>
```

login.properties

Este arquivo mantém as mensagens específicas da action de login.

```
# Mensagens da tela de login
login.title=Entrada do sistema
login.saudacao=Bem vindo a página de login
login.username=Nome do usuário
login.password=Senha
login.enter=Entrar

login.invalidCredentials=Dados de login incorretos!
login.unknownUser=Usuário não encontrado!
login.wrongPassword=Senha inválida para o usuário!

invalid.fieldvalue.password=A senha deve ser numérica!
```

A chave **invalid.fieldvalue.fieldname** (no exemplo nosso campo é o **password**) é uma definição padrão do framework e será procurada toda vez que houver um erro de conversão (no exemplo conversão para Integer), caso não exista a chave o Webwork utilizará uma mensagem padrão (em Inglês).

login-validation.properties

Neste arquivo definimos os validadores que utilizaremos na tela de login.

```
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator
1.0//EN"
"http://www.opensymphony.com/xwork/xwork-validator-1.0.dtd">
<validators>
  <field name="username">
    <field-validator type="requiredstring">
      <message key="requiredField">Campo exigido!</message>
    </field-validator>
  </field>
  <field name="password">
    <field-validator type="required">
      <message key="requiredField">Campo exigido!</message>
    </field-validator>
    <field-validator type="int">
      <param name="min">1000</param>
      <param name="max">9999</param>
      <message key="outOfRangeField">bar must be between ${min} and ${max},
current value is ${password}.</message>
    </field-validator>
  </field>
</validators>
```

Veja que utilizamos sempre uma chave para aproveitarmos os recursos de localização, caso a chave não exista será usada a mensagem deste arquivo.

ApplicationSecurityInterceptor

Aproveitando o recurso de interceptors criamos esta classe que tem como função não permitir a execução de ações sem uma sessão do usuário.

Inicialize a aplicação e tente acessar a action **interno.action** e você será redirecionado para a página de login.

```
package br.com.guj.system;

import br.com.guj.components.UserSession;
import br.com.guj.components.UserSessionAware;

import com.opensymphony.xwork.Action;
import com.opensymphony.xwork.ActionInvocation;
import com.opensymphony.xwork.interceptor.Interceptor;

/**
 * ApplicationSecurityInterceptor
 * @author smota
 */
public class ApplicationSecurityInterceptor
    implements Interceptor
{
    //~ Methods
    ///////////////////////////////////////////////////////////////////

    /**
     * Called to let an interceptor clean up any resources it has allocated.
     */
    public void destroy() {
    }

    /**
     * Called after an Interceptor is created, but before any requests are
    processed using the intercept() methodName. This
     * gives the Interceptor a chance to initialize any needed resources.
     */
    public void init() {
    }

    /**
     * Allows the Interceptor to do some processing on the request before
    and/or after the rest of the processing of the
     * request by the DefaultActionInvocation or to short-circuit the
    processing and just return a String return code.
     *
     * @param invocation
     * @return
     * @throws Exception
     */
    public String intercept(ActionInvocation invocation) throws Exception {
        Action action = invocation.getAction();

        if(action instanceof DisableSecurityAction) {
            return invocation.invoke();
        } else { //devemos validar
            UserSession session = ((UserSessionAware)
    action).getUserSession();
            if(session.getUser()==null) { //não há usuário logado
                return Action.LOGIN;
            }
        }

        return invocation.invoke();
    }
}
```

```
}
```

UserSession

Este é nosso componente armazenado na sessão do usuário. O framework é responsável por instanciarlo e passa-lo para todas as actions que implementem sua interface de marcação **UserSessionAware**.

```
package br.com.guj.components;

import java.io.Serializable;

import br.com.guj.models.User;

/**
 * @author smota
 *
 * Componente para manipulação da sessão do usuário.
 *
 */
/**
 * Não é um componente direto com o User para poder colocar
 * na seção mais objetos relacionados ao usuário.
 * */
public class UserSession
    implements Serializable
{

    private User user;

    /**
     * @return Returns the user.
     */
    public User getUser() {
        return user;
    }
    /**
     * @param user The user to set.
     */
    public void setUser(User user) {
        this.user = user;
    }
}
```

components.xml

Este arquivo define os componentes, e suas configurações, do sistema.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<components>
  <component>
    <scope>session</scope>
    <class>br.com.guj.components.UserSession</class>
    <enabler>br.com.guj.components.UserSessionAware</enabler>
  </component>
</components>
```

xwork.xml

Este é o arquivo onde são configuradas as actions da aplicação. Para este exemplo optou-se por manter essa configuração simples dando ênfase ao recursos do framework em si, mas existem muitos recursos para configuração das actions que podem ser conhecidos na documentação do framework, os principais são:

- Parâmetros estáticos passados neste arquivo para as actions;
- Configuração de várias packages com namespaces específicos;
- Herança nas packages e namespaces permitindo facilmente a criação de skins para sua aplicação;
- Os parâmetros dos *results* podem ser propriedades da action ($\${propriedade}$).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE xwork PUBLIC "-//OpenSymphony Group//XWork 1.0//EN"
"http://www.opensymphony.com/xwork/xwork-1.0.dtd">

<xwork>
  <include file="webwork-default.xml" />

  <package name="default" extends="webwork-default">

    <interceptors>
      <interceptor
name="appsecurity" class="br.com.guj.system.ApplicationSecurityInterceptor"/>

      <interceptor-stack name="systemDefaultStack">
        <interceptor-ref name="model-driven"/>
        <interceptor-ref name="component"/>
      </interceptor-stack>
      <interceptor-ref name="appsecurity"/>
      <interceptor-ref name="validationWorkflowStack"/>
      <interceptor-ref name="logger"/>
      <interceptor-ref name="timer"/>
    </interceptors>

    <default-interceptor-ref name="systemDefaultStack"/>

    <global-results>
      <result name="login" type="redirect">
        <param name="location"/>/login.jsp</param>
      </result>
    </global-results>

    <action name="interno" class="br.com.guj.actions.Interno">
      <result name="success" type="dispatcher">
        <param name="location"/>/interno.jsp</param>
      </result>
    </action>

    <action name="login" class="br.com.guj.actions.Login">
      <result name="input" type="dispatcher">
        <param name="location"/>/login.jsp</param>
      </result>
      <result name="error" type="dispatcher">
        <param name="location"/>/login.jsp</param>
      </result>
      <result name="success" type="redirect">
        <param name="location"/>/interno.action</param>
      </result>
    </action>

  </package>
```

</xwork>

Conclusão

É importante que o leitor esteja a vontade com todos os conceitos envolvidos em um framework MVC para tirar melhor proveito do Webwork ou seja qual for o framework MVC que escolher usar.

Este tutorial apresentou um exemplo com muitos recursos do framework, mas ele de maneira alguma está limitado a estes recursos e muitas vezes pode permitir o mesmo resultado com abordagens diferentes.

O melhor agora é baixar o código do tutorial e explorá-lo adicionando novos recursos à aplicação.

Espero que este tutorial seja útil e as dúvidas surgidas em sua leitura poderão e deverão ser sanadas no fórum do GUJ.

Samuel Mota (smota@pinguimdesign.com.br) é programador Java à 2 anos e atua como consultor de desenvolvimento e projeto em diversas empresas principalmente com plataformas Unix utilizando C e Java.