Bab 2

Pemrograman MVC dengan Controller Webwork / Struts 2

2.1 Tujuan

Dalam bab ini, kita akan mendiskusikan bagaimana peran controller didalam MVC sekaligus mendiskusikan perkembangan beberapa teknologi controller khususnya Struts 2 yang merupakan pengembangan dari Struts1 dan WebWork.

Pada akhir bab ini, pelajar diharapkan dapat:

- Memahami peran controller dalam MVC
- Memahami prinsip kerja webwork sekaligus mampu untuk mengimplementasikan webwork/struts2 sebagai controller dalam MVC
- Mengetahui penggunaan validator dan interceptor dalam webwork

2.2 Controller dalam MVC

Controller ini adalah sebuah layer yang bekerja untuk mengurusi urusan "antar layer", yang artinya bertanggung jawab terhadap eksekusi aplikasi. Sebenarnya ada banyak controller yang dapat digunakan, seperti JSF, Tapestry dari Apache, atau WebWork yang sekarang lebih dikenal dengan Struts 2.0. Yang mana bab ini akan mendalami teknologi berdasarkan WebWork, yang terkenal merupakan solusi MVC paling mudah didunia Java.

Sebelumnya ada Struts 1.x, yang mana adalah framework paling populer didunia Java. Tetapi tim Apache Struts telah setuju untuk membuat WebWork 2.3 menjadi Struts 2.0. Hal ini dilakukan karena dianggap Struts 1.x setelah ditinggal sang pembuatnya ke JSF, teknologinya mandek, sedangkan WebWork telah masuk ke teknologi berikutnya yang lebih modern.

Gabungan kekuatan nama Struts yang telah mendunia dan teknologi yang mengacu pada WebWork yang sangat stabil dan mature, diharapkan dapat mencounter gerakan JSF. Tetapi saat ini yang terjadi, bukan JSF melawan WebWork/Struts yang terjadi, tetapi Don Brown, salah satu commiter WebWork/Struts2 telah berhasil membuat component development didalam WebWork, sehingga Struts2 sebenarnya adalah project integrasi WebWork dengan JSF.

Beberapa fitur dari WebWork selain kemudahaannya, diantaranya adalah telah terintegrasinya

aplikasi yang umum kita perlukan seperti pembuatan Chart dengan JFreeChart, Reporting dengan Jasper, membuat XML untuk transformasi XHTML. Malahan teknologi AJAX yang baru saja direlease telah diintegrasikan kedalam WebWork sejak 2.2. Yang semua ini hanya dengan mengganti kata result dari html ke chart, atau xml didalam sebuah xml-nya.

Semoga setelah mendalami bab ini, dapat merasakan kekuatan WebWork.

2.3 Pengenalan WebWork

WebWork adalah sebuah projek Java yang mengadopsi model pengembangan MVC (Model – Viewer – Controller), yang mana posisinya lebih tepat diarea C dari MVC, yaitu Controller. WebWork merupakan implementasi dari servlet dispatcher, yang bekerja menghandle semau request dan response dari setiap akses Web.

WebWork diciptakan oleh Richard Oberg, salah satu orang jenius didunia Java dimuka bumi. Beberapa karyanya yang mendunia adalah JBoss dan Xdoclet. Yang mana kedua teknologi ini dibahas juga didalam buku ini.

Versi 2.0 dari WebWork dikembangkan oleh Patrick Lightbody dan Jason Carreira, yang akhirnya membuat WebWork yang semua sebuah project Open Source Java sendirian, masuk ke OpenSymphony.com, sebuah host Open Source Java yang tentu saja tidak sebesar Apache, tetapi telah berhasil menghasilkan banyak perusahaan bernilai jutaan dolar, dengan mengadopsinya, seperti JiveSoftware di NewYork, Amerika dan Atlassian di Sydney, Australia.

Walaupun sekarang web resmi WebWork telah migrasi ke Apache tepatnya http://struts.apache.org, tetapi tetap saja semua diskusi dan tips pengembangan WebWork didiskusikan di OpenSymphony, tepatnya di http://www.opensymphony.com/webwork.

Dalam dunia Java, WebWork adalah sebuah sleeping elephant, artinya sebuah teknologi yang sangat stabil, tetapi timnya kurang giat berpromosi, malah produk yang dikembangkan diatas WebWork yang lebih sering dipromosikan. Diantaranya adalah JiveForum, forum Java paling populer yang dipakai oleh java.sun.com, GE, Sony. Jadi kalau kita posting forum di Java.sun.com, secara langsung kita menggunakan WebWork.

2.4 Prinsip Kerja WebWork

Inti dari WebWork adalah XWork, sebuah teknologi controller generic, yang dapat digunakan dalam solusi Web dan non Web, tetapi tentu saja yang solusi Web adalah yang paling mature dan stabil. Bilamana ingin mendalami bagaimana Xwork dikembangkan menjadi solusi berbasis Swing, dapat mendalami pendulum (http://pendulum.sf.net).

WebWork dapat dikatakan web wrappernya XWork, sehingga XWork dapat berjalan didalam lingkungan solusi Web atau servlet.

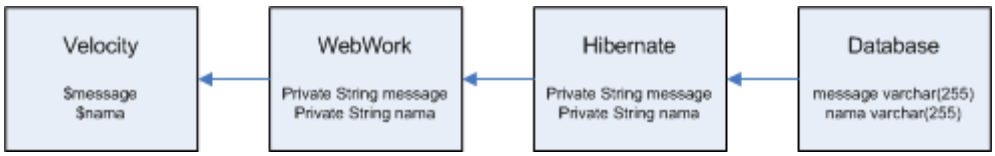
Untuk mengerti bagaimana WebWork bekerja sebenarnya, kita hanya perlu mengerti bagaimaan sebuah POJO dari Java bekerja. POJO yang singkatan dari Plain Old Java Object adalah mekanisme Java "mau-mau kita", yang dalam implementasinya adalah dengan mengimplementasikan get atau set. Yang mana ini adalah mekanisme standar yang harus diketahui dalam pemogramaan Java.

Contohnya adalah berikut

```
package org.blueoxygen.bbook;

import com.opensymphony.xwork.Action;

public class HelloWorld implements Action {
	private String message;
	public String execute() {
			message = "Hello, World!\n";
			message += "The time is:\n";
			message += System.currentTimeMillis();
			return SUCCESS;
	}

	public String getMessage() {
		return message;
	}

	public void setMessage(String message) {
		this.message = message;
	}


}
```

Contoh diatas adalah satu action dari WebWork, perbedaannya dengan POJO adalah memiliki sebuah method execute(), dan mengimplementasikan Action dari xwork.

Setiap variable didalam sebuah POJO yang mengimplementasikan Action, secara otomatis akan dikenal pada presentation layernya MVC yang mengimplementasikan WebWork. Mekanisme ini berlaku saat kita memapping hibernate kedalam MVC.



Dengan konsep ini, secara otomatis, Anda telah mengimplementasikan MVC. Mudah bukan. Itulah WebWork.

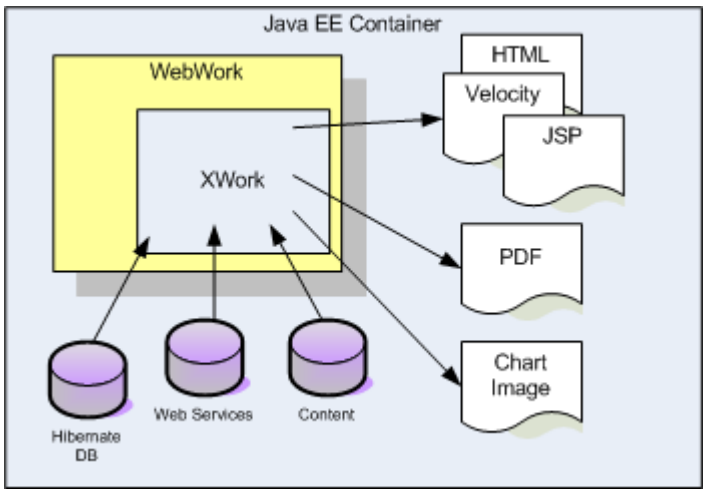
Setelah itu, kita hanya perlu mendaftarkan object yang telah kita buat, yang kita sebut WebWork Action ini kedalam file konfigurasinya yaitu xwork.xml.

Contohnya adalah seperti ini:

```
<!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="action" extends="webwork-default" namespace="/action">

		<action name="helloJSP" class="org.blueoxygen.bbook.HelloWorld">
			<result name="success">helloworld.jsp</result>
```

```
            </action>
            <action name="helloVelocity" class="org.blueoxygen.bbook.HelloWorld">
        <result name="success" type="velocity">helloworld.vm</result>
    </action>
    <action name="helloFreemarker" class="org.blueoxygen.bbook.HelloWorld">
            <result name="success" type="freemarker">helloworld.ftl</result>
    </action>
    <action name="hello" class="org.blueoxygen.bbook.Hello">
        <result name="success" type="velocity">hello.vm</result>
        <result name="input" type="velocity">helloworld.vm</result>
    </action>
     </package>
     <package name="descriptor" extends="webwork-default" namespace="/descriptor">
            <action name="create" class="org.blueoxygen.bbook.Form">
                <result name="success" type="velocity">add.vm</result>
            </action>
            <action name="add" class="org.blueoxygen.bbook.SaveForm">
                <result name="success" type="velocity">simpan.vm</result>
                <result name="error" type="velocity">gagal.vm</result>
            </action>
    </package>
</xwork>
```

Bilamana kita hendak mengganti viewer layer dari JSP ke Velocity, tambahkan type=velocity, atau bilamana hendak melakukan redirecting ke halaman berikutnya ganti dengan type=redirect, ini artinya setelah proses dilakukan web akan diforward ke url lainnya. Gantilah dengan chart, jasper bilamana outputnya ingin PDF. Tentu saja didalam baris method execute(), kita harus menambahkan beberapa script yang memungkinkan output menjadi PDF.



Adapun bentuk hasil akhir (result) default dari script diatas dengan JSP adalah

```
<%@ taglib prefix="ww" uri="webwork" %>
<html>
<head>
<title>Hello Page</title>
</head>
<body>
The message generated by my first action is:
<ww:property value="message"/>
```

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

Sedangkan bilamana menggunakan Velocity adalah

```
<html>
<head>
<title>Hello Page</title>
</head>
<body>
The message generated by my first action is:
$message
</body>
</html>
```

Terlihat velocity lebih simple. Bilamana kita menambahkan variable didalam Action, maka secara langsung akan dikenal dalam presentation layernya. Coba bandingkan dengan script Velocity dibab sebelumnya. Kita harus membuat sebuah variable implementasi dari Java Collection seperti hashmap, atau object Java lainnya, kemudian dimasukan kedalam Context, dan variable context yang diparsing didalam presentation layer. Dengan WebWork, semua tidak diperlukan lagi, semua telah diproses secara otomatis.

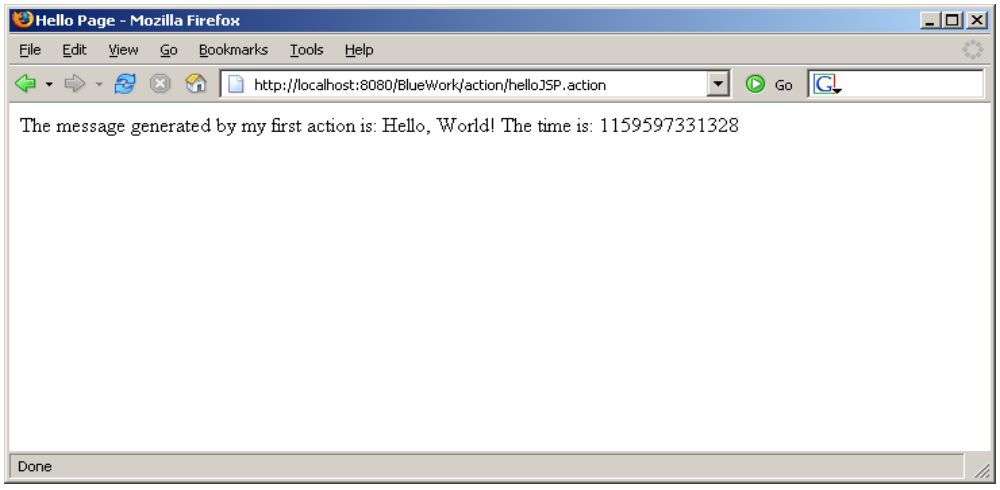
Sedangkan bilamana kita menambahkan variable transformasi mengembalikan variable sebuah String seperti "Dua ratus lima puluh rupiah", hanya perlu dibuat sebuah variable baru saja, dan diisi variable tersebut dengan value yang diperlukan.

Masalah sering terjadi bilamana yang direturn adalah null, karena secara default akan mereturn "null" di layer presentation. Ini tentu saja mengganggu, coba pakai perintah !$message.

Sedangkan untuk memanggil code yang terlah termap antara presentation dan xwork.xml, adalah dengan memanggil code mengikuti namespacenya.

Lihat baris berikut, yang terdapat pada file xwork.xml

```
<package name="action" extends="webwork-default" namespace="/action">
<action name="helloJSP" class="org.blueoxygen.bbook.HelloWorld">
        <result name="success">helloworld.jsp</result>
</action>
```



Cara mengeksekusinya adalah dengan mengetik
http://localhost:8080/BlueWork/helloJSP.action.

Setiap action didalam xwork, akan harus diberikan akhiran .action. Yang artinya menjalankan objek org.blueoxygen.bbook.HelloWorld. Bilamana eksekusi berhasil, akan melempar semua hasil eksekusi ke file helloworld.jsp.

Hal ini berlaku bagi semua tipe result dari WebWork.

2.5 Implementasi Validasi pada WebWork – Standard

Ada lagi satu teknologi yang mungkin sangat bagus didalam WebWork, yaitu validasi. Berita terbaru dari validasi ini, salah satu pembuat WebWork yaitu Jason Carreira, telah menjadi salah satu expert group dari spesifikasi validasi, jadi mungkin saja kedepanyna di dalam JCP akan ada teknologi validasi yang bekerja mirip dengan validasi didalam WebWork. Bilamana realisasi terjadi, bukan tidak mungkin WebWork dan JSF memiliki validasi yang sama.

Untuk membuat sebuah validasi sebenarnya tidaklah sulit, kita hanya perlu membuat sebuah file xml dengan aturan yang sama dengan nama class ditambah kata validation.

Contoh validasi yang didapat pada projek Cimande Thin 1.0 terdapat AddModule yang merupakan file implementasi dari Action, dan sebuah file AddModule-validation.xml

Berikut adalah isi dari file AddModule-validation.xml

```
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
"http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">

<validators>
   <field name="module.name">
      <field-validator type="requiredstring">
         <message>You must enter a name</message>
      </field-validator>
   </field>
   <field name="module.nameSpace">
      <field-validator type="requiredstring">
         <message>NameSpace Cannot be empty</message>
      </field-validator>
   </field>
</validators>
```

Dimana type=”requiredstring” merupakan proses validasi dari class com.opensymphony.xwork.validator.validators.RequiredStringValidator.

Type dari validasi ini merupakan referensi dari file validators.xml yang harus ditemukan didalam folder /WEB-INF/classes.

Bilamana hendak membuat sendiri validasinya, dapat melihat source code dari WebWork, Validator berada dalam com.opensymphony.xwork.validator.validators.

Jangan lupa memasukan baris <interceptor-ref name="validation"/>, pada package action, berikut adalah contohnya:

```
<action name="add" class="org.blueoxygen.cimande.thin.module.action.AddModule">
```

```
        <interceptor-ref name="params"/>
        <interceptor-ref name="validation"/>
        <interceptor-ref name="component"/>
        <interceptor-ref name="workflow"/>
        <result name="success" type="redirect">create.action</result>
        <result name="input" type="redirect">create.action</result>
</action>
```

2.6 Interceptor

Telah dibahas sebelumnya, penggunaan interceptor pada setting xwork.xml. Ternyata sebenarnya interceptor itu ada banyak didalam xwork yang siap kita gunakan.

Interceptor adalah sebuah mekanisme yang memungkinkan sebuah objek dijalankan sebelum proses atau setelah proses Action. Interceptor bekerja seperti filter sebenarnya pada pemograman servlet, dimana kalau di JavaEE filter disetup di web.xml, sedangkan interceptor didalam WebWork diletakan diantara action-name.

Contohnya adalah

```
        <package name="default" extends="webwork-default">
                <result-types>
                <result-type name="chart"
class="org.blueoxygen.cimande.dashboard.chart.ChartResult"/>
                </result-types>
      <interceptors>
         <interceptor-stack name="defaultComponentStack">
            <interceptor-ref name="component"/>
            <interceptor-ref name="model-driven"/>
            <interceptor-ref name="validationWorkflowStack"/>
         </interceptor-stack>

          <interceptor-stack name="chainingComponentStack">
            <interceptor-ref name="defaultComponentStack"/>
            <interceptor-ref name="chain"/>
         </interceptor-stack>
      </interceptors>

      <default-interceptor-ref name="defaultComponentStack"/>
   </package>
```

Berikut adalah diagram interceptor yang disediakan dan siap digunakan:

Interceptor	Name	Description
Alias Interceptor	alias	Converts similar parameters that may be named differently between requests.
Chaining Interceptor	chain	Makes the previous action's properties available to the current action. Commonly used together with <result type="chain"> (in the previous action).
Component Interceptor	component	Enables and makes the components available to the Actions. Refer to components.xml
Conversion Error Interceptor	conversionError	adds conversion errors from the ActionContext to the Action's field errors
Create Session Interceptor	createSession	create an HttpSession automatically, usefull with certain interceptor (eg. TokenInterceptor) when an HttpSession is required in order to work properly
Execute and Wait Interceptor	execAndWait	an interceptor that executes the action in the background and then sends the user off to an intermediate waiting page.
Exception Interceptor	exception	Maps exceptions to a result.
File Upload Interceptor	fileUpload	an interceptor that adds easy access to file upload support. See the javadoc for more info
I18n Interceptor	i18n	remembers the locale selected for a user's session
Logger Interceptor	logger	Outputs the name of the action
Model Driven Interceptor	model-driven	If the action implements ModelDriven, pushes the getModel() result onto the valuestack.
Parameters Interceptor	params	Sets the request parameters onto the action.
Prepare Interceptor	prepare	If the action implements Preparable, calls its prepare() method.
Scope Interceptor	scope	simple mechanism for storing action state in the session or application scope
Servlet Config Interceptor	servlet-config	Give access to HttpServletRequest and HttpServletResponse (think twice before using this since this ties you to the Servlet api)
Static Parameters Interceptor	static-params	Sets the xwork.xml defined parameters onto the action. These are the <param> tags that are direct children of the <action> tag.
Timer Interceptor	timer	Outputs how long the action (including nested interceptors and view) takes to execute
Token Interceptor	token	Checks for valid token presence in action, prevents duplicate form submission
Token Session Interceptor	token-session	Same as above, but storing the submitted data in session when handed an invalid token
Validation Interceptor	validation	Performs validation using the validators defined in {Action}-validation.xml
Workflow Interceptor	workflow	Calls the validate method in your action class. If action errors created then it returns the INPUT view.
Parameter Filter Interceptor	N/A	Removes parameters from the list of those available to actions etc.