

Simple J2EE Model View Controller Type II Framework

Executive Summary

Application presents content to users in numerous pages containing various data. Also, the engineering team responsible for designing, implementing, and maintaining the application is composed of individuals with different skill sets.

One of the major concerns with the web applications is the separation between the logics that deal with Presentation itself, the data to be presented and the one that controls flow of logic. It is as an answer to such concerns that the Model-View-Controller or MVC pattern was designed.

This paper provides the solution to modularize the user interface functionality of a Web application so that individual parts can be easily modified, that is model view controller framework.

Introduction

The Model-View-Controller (MVC) pattern separates the modeling of the domain, the presentation, and the actions based on user input into three separate classes.

Model: The model manages the behavior and data of the application domain, responds to requests for information about its state (usually from the view), and responds to instructions to change state (usually from the controller).

View: The view manages the display of information.

Controller: The controller interprets the mouse and keyboard inputs from the user, informing the model and/or the view to change as appropriate.

It is important to note that both the view and the controller depend on the model. However, the model depends on neither the view nor the controller. This is one the key benefits of the separation. This separation allows the model to be built and tested independent of the visual presentation. The separation between view and controller is secondary in many rich-client applications, and, in fact, many user interface frameworks implement the roles as one object. In Web applications, on the other hand, the separation between view (the browser) and controller (the server-side components handling the HTTP request) is very well defined.

The solution provided in this paper is used very simple servlet and JSP and plain java objects, using this framework very easily any real time applications can be developed. By following this simple framework most of the complex MVC frameworks can be understood.

Model View Controller Types

MVC Type-I: In this type of implementation, the View and the Controller exist as one entity -- the View-Controller. In terms of implementation, in the Page Centric approach the Controller logic is implemented within the View i.e. with J2EE, it is JSP. All the tasks of the Controller, such as extracting HTTP request parameters, call the business logic (implemented in JavaBeans, if not directly in the JSP), and handling of the HTTP session is embedded within JSP using scriptlets and JSP action tags.

MVC Type-II: The problem with Type-I is its lack of maintainability. With Controller logic embedded within the JSP using scriptlets, the code can get out of hand very easily. So to overcome the problems of maintainability and reusability, the Controller logic can be moved into a servlet and the JSP can be used for what it is meant to be -- the View component. Hence, by embedding Controller logic within a servlet, the MVC Type-II Design Pattern can be implemented.

The major difference between MVC Type-I and Type-II is where the Controller logic is embedded in JSP in Type-I and in Type-II its moved to servlet.

MVC Type-II Framework

In this frame work, Model is a plain old java object, view is a JSP which will render the page using the model , these two are application dependent and this framework has a centralized controller is a servlet, which will populate the model and invokes a method from the action class.

Below is the source of the controller.

SimpleController.java

```
package simple;

import java.io.IOException;
import java.lang.reflect.Method;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class SimpleController extends HttpServlet {

    private ActionBeanMapping mapping;

    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {
        try {
            String strJsp = null;
            String strURI = request.getRequestURI();
```

```

        int startIndex = strURI.lastIndexOf("/");
        int endIndex = strURI.lastIndexOf(".do");
        String strAction =
            strURI.substring(startIndex+1, endIndex);
        this.populateBean(request, strAction);
        SimpleHandler handler =
            (SimpleHandler)mapping.getActionInstance(strAction);
        strJsp = handler.process(request, response);
        request.getRequestDispatcher(strJsp).forward(request, response);
    } catch (Exception e) {
        e.printStackTrace();
    }
    request.getRequestDispatcher("/error.jsp").forward(request, response);
}

public void doPost(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
    this.doGet(request, response);
}

public void init() throws ServletException {
    String strFile = this.getServletContext().getRealPath("/") +
        this.getServletConfig().getInitParameter("actionmappings");
    System.out.println("MAPPING FILE PATH: "+strFile);
    try {
        mapping = new ActionBeanMapping(strFile);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

private void populateBean(HttpServletRequest request, String strAction)
{
    Object obj;
    try {
        obj = mapping.getBeanInstance(strAction);
        Method methods[] = obj.getClass().getMethods();
        for(int i=0; i<methods.length; i++) {
            Method method = methods[i];
            String strName = method.getName();
            if(strName.startsWith("set")) {
                String strField = strName.substring(4);
                strField =
                    String.valueOf(strName.charAt(3)).toLowerCase()+
                    strField;
                String arrayValue[] =
                    request.getParameterValues(strField);
                String strValue = null;
                if(arrayValue != null && arrayValue.length>0){
                    strValue = arrayValue[0];
                }
                try {
                    method.invoke(obj, strValue);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
    }
}

```

```

        }
    }
    request.setAttribute(SimpleHandler.BEAN, obj);
} catch (Exception e) {
    e.printStackTrace();
}
}
}

```

The servlet's init method is used to initialize the action and bean mappings.

```

public void init() throws ServletException {
    String strFile = this.getServletContext().getRealPath("/") +
this.getServletConfig().getInitParameter("actionmappings");
    System.out.println("MAPPING FILE PATH: "+strFile);
    try {
        mapping = new ActionBeanMapping(strFile);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}

```

Mapping file path is taken from the servlet config, and initialized the ActionBeanMapping helper class.

ActionBeanMapping.java:

```

package simple;

import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;

public class ActionBeanMapping {
    private Properties prop = new Properties();

    public ActionBeanMapping(String propFile) throws IOException {
        this.prop.load(new FileInputStream(propFile));
    }

    public Object getActionInstance(String action) throws Exception {
        String strClass = prop.getProperty("action."+action.trim());
        if(strClass == null)
            throw new NullPointerException("Null action: "+action);
        return Class.forName(strClass).newInstance();
    }

    public Object getBeanInstance(String action) throws Exception {
        String strClass = prop.getProperty("bean."+action);
        if(strClass == null) throw
            new NullPointerException("Null bean: "+action);
        return Class.forName(strClass).newInstance();
    }
}
}

```

This class reads the properties file and provides two methods to instantiate the Action and Bean classes using java reflection for the specified user action.

The GET and POST methods of the request calls the following code in controller.

```
try {
    String strJsp = null;
    String strURI = request.getRequestURI();
    int startIndex = strURI.lastIndexOf("/");
    int endIndex = strURI.lastIndexOf(".do");
    String strAction =
        strURI.substring(startIndex+1, endIndex);
    this.populateBean(request, strAction);
    SimpleHandler handler =
        (SimpleHandler)mapping.getActionInstance(strAction);
    strJsp = handler.process(request, response);
    request.getRequestDispatcher(strJsp).forward(request, response);
} catch (Exception e) {
    e.printStackTrace();
    request.getRequestDispatcher("/error.jsp").forward(request, response);
}
```

This piece of code gets the user action from the URI and instantiates the bean and action class and populates the model and invokes the method on an action class. All the actions classes in the application should implement the interface SimpleHandler. If any error occurs this controller forwards to a generalized error page.

SimpleHandler.java:

```
package simple;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public interface SimpleHandler {
    public static final String BEAN = "simple.BEAN";

    public String process(HttpServletRequest request, HttpServletResponse
response) throws Exception;
}
}
```

All the action classes in the application should implement the process method.

Population of model data from the request object is done by the following controller method.

```
private void populateBean(HttpServletRequest request, String strAction)
{
    Object obj;
    try {
        obj = mapping.getBeanInstance(strAction);
        Method methods[] = obj.getClass().getMethods();
        for(int i=0; i<methods.length; i++) {
            Method method = methods[i];
            String strName = method.getName();
            if(strName.startsWith("set")) {
                String strField = strName.substring(4);
```

```

        strField =
String.valueOf(strName.charAt(3)).toLowerCase()+
strField;
        String arrayValue[] =
request.getParameterValues(strField);
String strValue = null;
if(arrayValue != null && arrayValue.length>0){
            strValue = arrayValue[0];
        }
        try {
            method.invoke(obj, strValue);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
request.setAttribute(SimpleHandler.BEAN, obj);
} catch (Exception e) {
    e.printStackTrace();
}
}

```

This method populates the model data and binds the model to request object, this model is accessed by the action class and JSP.

error.jsp

```

<%@ page language="java" pageEncoding="ISO-8859-1"%>
<html>
<head>
<title>Error page</title>
</head>
<body>
<font color="#ff0000"><b>Error ocured while processing request.</b></font>
</body>
</html>

```

The web configuration is defined below, it's a simple configuration file for controller.

web.xml:

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
    xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
    <servlet>
        <description>Simple J2EE Controller</description>
        <display-name>Simple J2EE Controller</display-name>
        <servlet-name>SimpleController</servlet-name>
        <servlet-class>simple.SimpleController</servlet-class>
        <init-param>
            <param-name>actionmappings</param-name>
            <param-value>WEB-INF/actionmappings.properties</param-value>
        </init-param>
    </servlet>

```

```

    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>SimpleController</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
<welcome-file-list>
  <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
</web-app>

```

The controller servlet is invoked for all the urls which will ends with .do, this servlet loads on server startup, and defines the action mappings file path.

Sample Application using the Framework

Providing sample application to registration to store name, email and phone.

index.jsp:

```

<%@ page language="java" pageEncoding="ISO-8859-1"%>
<html>
  <head>
    <title>Home page</title>
  </head>

  <body>
    <form method="post" action="register.do">
      <table width="200" border="0" align="center">
        <tr>
          <td colspan="2" align="center"><strong>User
Data</strong>&nbsp;</td></tr>
        <tr>
          <td>Name</td>
          <td><input type="text" name="name"></td></tr>
        <tr>
          <td align="left">Email</td>
          <td><input type="text" name="email"></td></tr>
        <tr>
          <td align="center">&nbsp;</td>
          <td align="center">&nbsp;<strong>Phone</td>
          <td><input type="text" name="phone"></td></tr>

        <tr>
          <td colspan="2" align="center">&nbsp;<input type="submit" value="Submit"
name="Submit"></td></tr>
      </table>
    </form>
  </body>
</html>

```

This is a JSP page to enter the data by the user to store in the system. When user clicks the Submit button on the page data is posted to the action register.

Bean and Action mappings are entered into the mappings file.

actionmappings.properties

```
action.register=simple.RegistrationHandler  
bean.register=simple.User
```

all the user actions from the view should present in this properties file.

When user submits the data to controller it populates the data to the following model.

User.java:

```
package simple;  
  
public class User {  
    private String name;  
    private String email;  
    private String phone;  
    public String getEmail() {  
        return email;  
    }  
    public void setEmail(String email) {  
        this.email = email;  
    }  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
    public String getPhone() {  
        return phone;  
    }  
    public void setPhone(String phone) {  
        this.phone = phone;  
    }  
}
```

Following action is called when data is submitted.

```
package simple;  
  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
  
public class RegistrationHandler implements SimpleHandler {  
    public String process(HttpServletRequest request,  
        HttpServletResponse response) throws Exception {  
        User user = (User)request.getAttribute(SimpleHandler.BEAN);  
        return "/success.jsp";  
    }  
}
```

This action class gets the bean from the request; this bean can be saved in the system, after successful processing it returns the success.jsp to controller to forward to this view.

success.jsp:

```
<%@ page language="java" import="simple.*" pageEncoding="ISO-8859-1"%>
<%
User user = (User) request.getAttribute(SimpleHandler.BEAN);
%>
<html>
  <head>
    <title>Success page</title>
  </head>
  <body>
    <table width="200" border="0" align="center">
      <tr>
        <td colspan="2" align="center"><strong>User Data Successfully
Saved</strong>&nbsp;</td></tr>
      <tr>
        <td>Name</td>
        <td><%= user.getName() %></td></tr>
      <tr>
        <td align="left">Email</td>
        <td><%= user.getEmail() %></td></tr>
      <tr>
        <td align="center">&nbsp;<td><%= user.getPhone() %></td></tr>
      <tr>
        <td colspan="2" align="center">&nbsp;<input type="button" value="Home"
onClick="window.location='/MVC';" name="Submit"></td></tr>
    </table>
  </body>
</html>
```

This JSP shows the success message of data processing.

Conclusion

One of the major concerns with the web applications is the separation between the logics that deal with Presentation itself, the data to be presented and the one that controls flow of logic. It is as an answer to such concerns that the Model-View-Controller or MVC pattern was designed.

References

- <http://java.sun.com/blueprints/patterns/MVC-detailed.html>
- <http://msdn2.microsoft.com/en-us/library/ms978748.aspx>
- <http://www.devarticles.com/c/a/Java/J2EE-Design-Patterns-The-Presentation-Layer-Patterns-ModelViewController/>
- http://java.sun.com/j2ee/sdk_1.3/techdocs/api/