Java Server Pages & Web Design Architectures

Jedrzej Jajor

17 maja 2004

Outline

Outline

Java Servlets

- Java Technology's answer to CGI programming
- A class derived from javax.servlet.GenericServlet, javax.servlet.http.HttpServlet or implementing interface javax.servlet.Servlet
- Constructed by the container, then initialized with init() method
- Calls from the clients handled by the service(...) method
- Destroyed with destroy(), then garbage-collected and finalized

Java Servlets - Advantages

- Efficient - each request handled by a lightweight Java thread
- Convenient - extensive infrastructure for automatical parsing and decoding form data, headers, handling cookies, sessions, ...
- Powerful - can talk directly to a Web Server, maintain state information between requests
- Portable - follow well-standarized API, supported directly or via a plugin on almost every major Web server
- Inexpensive - adding servlet support to a Web server is free or cheap

Java Servlets - Example

Simple Servlet

```
class SimpleServlet extends HttpServlet {
  public SimpleServlet() {}
  public doGet(HttpRequest req, HttpResponse res){
    PrintWriter out = res.getWriter();
    out.println("<html><head><title>");
    out.println("Example</title></head>");
    ...
  }
}
```

Outline

Idea of JSP

- Provides simple interface to write applications based on servlets
- Allows mixing static HTML with dynamically-generated content
- Fast & easy way to create dynamic web content
- Separates layout and code
- Enables rapid development of web-based applications
- Server- and platform-independent

JSP page lifecycle

- Client requests a page
- The container checks if it's compiled and ready
- JSP page is translated to a servlet
- Servlet is compiled and loaded
- Request is serviced

Simple page

index.jsp

```
<%@page contentType="text/html; charset=iso-8859-2"
session="false" import="java.util.Enumeration"%>
<html><head><title>Simple page</title></head>
<body><table><%
<%@include file="WEB-INF/jsp/menu.jsp" %>
<%for(Enumeration e = request.getParameterNames();
    e.hasMoreElements(); ) {
  String name = (String)e.nextElement();
  String val = request.getParameter(name);
  out.println(''<tr><td>'' + name + ''</td>'');
%> <td> <%= val %> </td></tr>
<% } %>
```

Implicit objects

- HttpJspPage page; // Page's servlet instance
- PageContext pageContext; // Provides access to all the namespaces associated with a JSP page and access to several page attributes
- HttpSession session; // User specific session data
- ServletContext application; // Data shared by all application pages
- ServletConfig config; // Servlet configuration information
- JspWriter out; // Output stream for page context
- HttpServletRequest request; // Data included with the HTTP Request
- HttpServletResponse response; // HTTP Response data, e.g. cookies

Outline

Outline

What is MVC?

- Three-layer application model
- Well established pattern for development of interactive application
- Increasingly popular in web application development
- Separates responsibilities among model, view and controller
- Reduces code duplication
- Makes applications easier to maintain
- Used in many web application frameworks

MVC - The Model

- Represents business data and business logic
- Notifies views when it changes
- Provides the ability for the view to query the model state
- Provides the access to application functionality for the controller

MVC - The View

- Renders the contents of the model
- Updates data presentation when the model changes
- Forwards user input to a controller
- Allows controller to select view

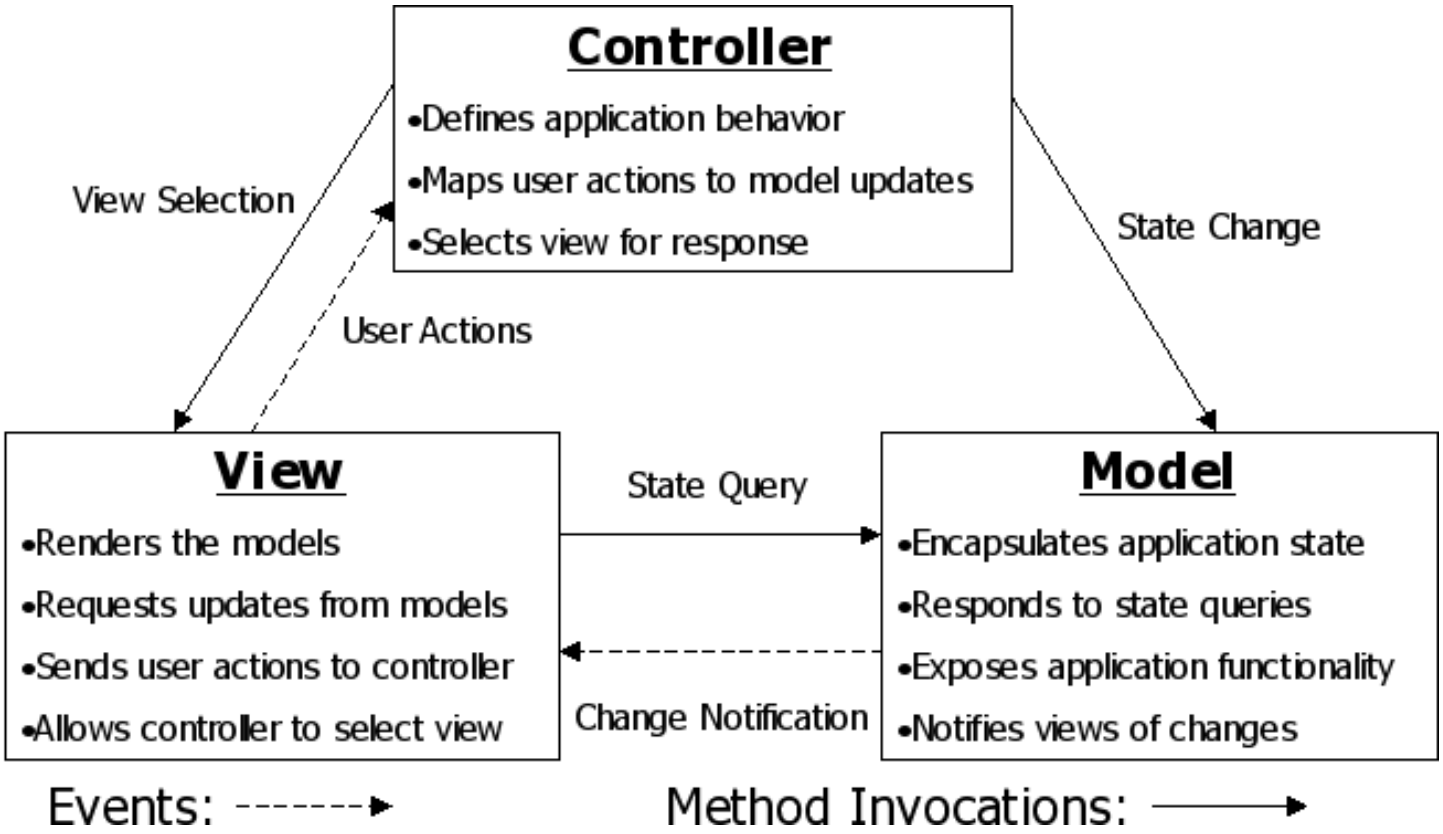
MVC - The Controller

- Defines application behavior
- Interprets user inputs and maps then into actions to be performed by the model
- Dispatches user requests and selects views for presentation
- Typically one controller for each set of related functionality

MVC - Benefits

- Separates design concerns
- Decreases code duplication
- Centralizes control
- Makes the application easy-modifyable
- Clearly defines the responsibilities of participating classes
- Makes bugs easier to track down and eliminate

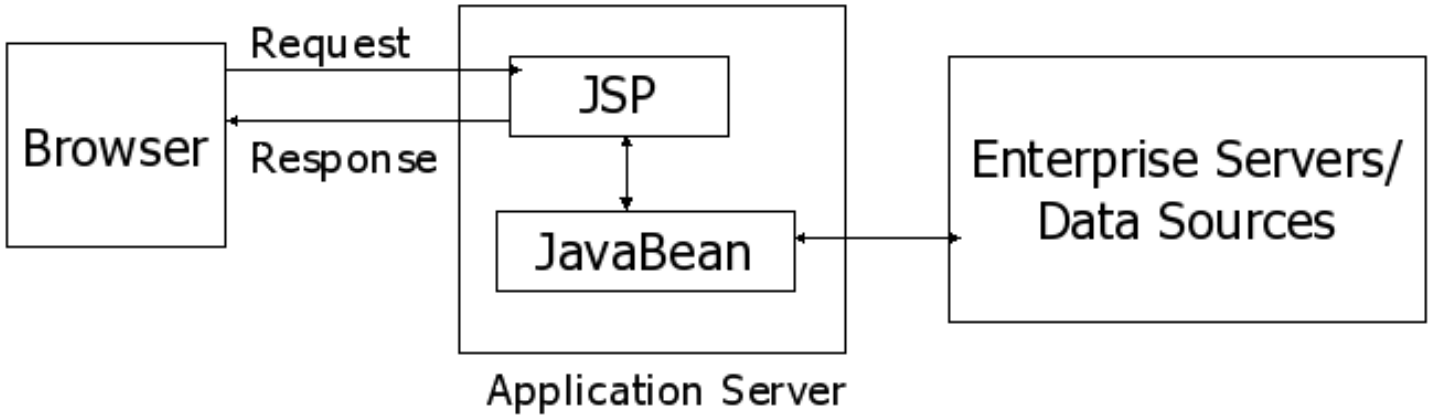
MVC scheme


Controller
•Defines application behavior
•Maps user actions to model updates
•Selects view for response
View Selection
User Actions
State Change
View
•Renders the models
•Requests updates from models
•Sends user actions to controller
•Allows controller to select view
State Query
Change Notification
Model
•Encapsulates application state
•Responds to state queries
•Exposes application functionality
•Notifies views of changes
Events:
Method Invocations:

Model 1 architecture

- Web browser directly accesses JSP pages
- Control is decentralized
- Next view is determined by hyperlinks or request parameters
- Each page or servlet processes its own input
- JSP pages access JavaBeans that represent the model

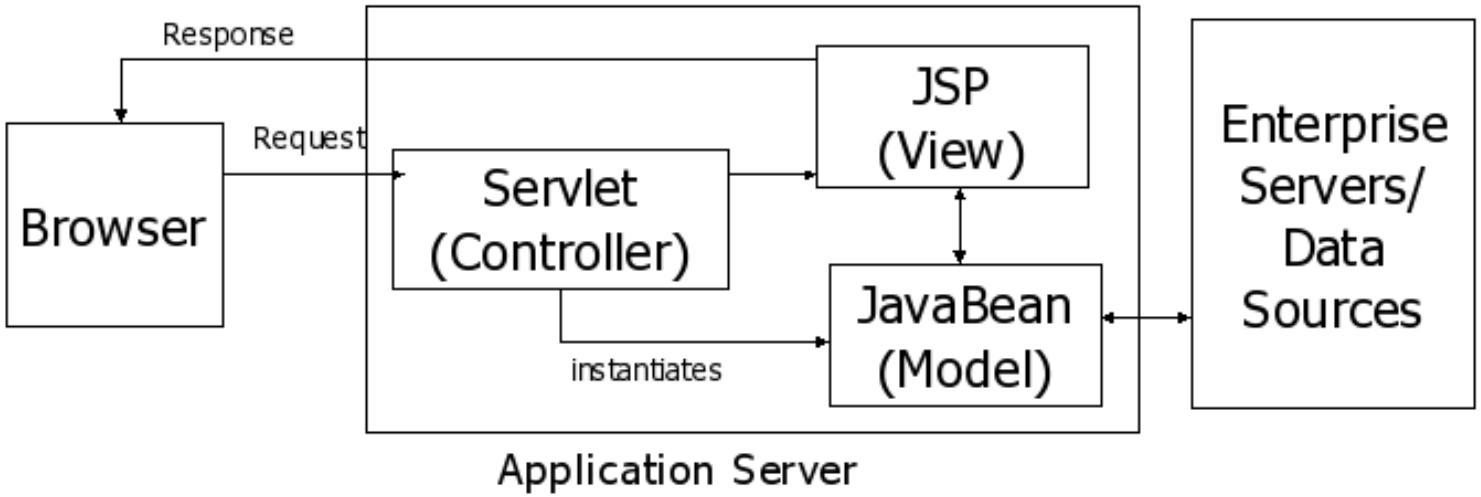
Model 1 scheme



Model 2 architecture

- Introduces a controller servlet between the browser and JSP pages
- Centralizes the logic for dispatching requests
- Next view based on URL, input parameters and application state
- Views do not refer to each other directly
- Applications are easier to maintain and extend

Model 2 scheme



To use or not to use...

- The Model 1 architecture can provide a more lightweight design for small, static applications.
- Model 1 architecture is suitable for applications that have very simple page flow, have little need for centralized security control or logging, and change little over time.
- Model 1 applications can often be refactored to Model 2 when application requirements change

Bad Example

controller.jsp

```
< %String creditCard = request.getParameter(
    "creditCard");
if (creditCard.equals("VisaElectron")) { %>
  <jsp:forward page="/processVisaEl.jsp"/>
<% } else if (creditCard.equals("Maestro")) { %>
  <jsp:forward page="/processMaestro.jsp"/>
<% } %>
```

Bad Practice:

JSP Page acting as a Controller

Bad Example 2

controller.java

```
public class ControllerServlet extends HttpServlet {
  protected void doPost(HttpServletRequest req,
      HttpServletResponse res) throws ... {
    String card = req.getParameter("creditCard");
    String page = "/process" + card + ".jsp";
    ServletContext sc = getServletContext();
    RequestDispatcher rd =
      sc.getRequestDispatcher(page);
    rd.forward(req, res);
  }
}
```

Outline

Controller Design

- Identifying the operation
- Invoking Model methods
- Controlling dynamic screen flow

Identifying the operation

- Indicate the operation in a hidden form field

Operation in hidden form field

```
<form method="POST" action=
    "http://myServer/myApp/myServlet">
  <input type="HIDDEN" name="OP"
    value="createUser"/>
  <!-- other form contents... -->
</form>
```

Identifying the operation

- Indicate the operation in a GET query string parameter

Operation in query string

```
http://myHost/myApp/servlets/myServlet?op=createUser
```

Identifying the operation

- Use a servlet mapping to map certain URLs to a servlet

Operation in URL

```
<servlet-mapping>
  <servlet-name>myServlet</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

:)

Servlet mappings provide the most flexible way to control where to route URLs based on patterns in the URLs.

Invoking Model methods

- Big if-then-else statement leads to large service method
- Better approach - use a Command pattern
- (wstawić diagram UML)

Invoking Model methods

Action abstraction:

Action.java

```
public abstract class Action {
  protected Model model;
  public Action(Model model) {this.model = model;}
  public abstract String getName();
  public abstract Object perform(
      HttpServletRequest req);
};
```

Invoking Model methods

Concrete action example:

CreateUserAction.java

```
public class CreateUserAction extends Action {
  public CreateUserAction(Model model) {
    super(model);
  }
  public String getName() { return "createUser"; }
  public Object perform(HttpServletRequest req) {
    String userName = req.getAttribute("user");
    String UserPass = req.getAttribute("pass");
    return model.createUser(userName, userPass);
  }
}
```

Controlling dynamic screen flow

The next view to display may depend on:

- The current view
- The results of any operation on the application model, returned by model method invocations
- Possibly other server-side state, kept in PageContext, ServletRequest, HttpSession, and ServletContext

Controller Servlet

ControllerServlet.java

```
public class ControllerServlet extends HttpServlet {
  private HashMap actions;
  public void init() throws ServletException {
    actions = new HashMap();
    CreateUserAction cua =
        new CreateUserAction(model);
    actions.put(cua.getName(), cua);
    //... create and add more actions
  }
// to be continued ...
```

Controller Servlet

ControllerServlet.java

```
public void doPost(HttpServletRequest req,
  HttpServletResponse resp) throws ... {
  // First identify operation from URL.
  String op = getOperation(req.getRequestURL());
  // Then find and execute corresponding Action
  Action action = (Action)actions.get(op);
  // to be continued...
```

Controller Servlet

ControllerServlet.java

```
// do Post - continued
   Object result = null;
    try {
      result = action.perform(req);
    } catch (NullPointerException npx) {
      //... handle "no such action" exception:
    }
    // ... Use result to determine next view
  }
  //... other methods...
}
```

View Design

- View displays data produced by the Model
- Usually JSP Pages or servlets, also HTML, XHTML, XML, PDF, graphics ...
- Lightweight and heavyweight clients
- Often used templates

View Design - Example

RegisterUser.jsp

```
<%@page content="text/html; charset=iso-8859-2"
session="true"%> <html><head>
<title>Register New User</title></head><body>
<%@include page="header.jsp"%>
<jsp:include page="menu.jsp" flush="true">
  <jsp:param name="parametr" value="wartosc"/>
</jsp:include>
<a href="<%= response.encodeURL("nextLink.jsp") %>">
link</a>
<%--in Struts we can put:
<html:link href="nextLink.jsp">Link</html:link> --%>
</body></html>
```

View Design - Example 2

myTemplate.vm

```
<html>
<body>
  Hello $customer.Name !
  <table>
    #foreach( $product in $customerSales )
      #if ( $customer.hasPurchased($product) )
        <tr>
          <td>$product</td>
        </tr>
      #end
    #end
  </table>
</body></html>
```

View Design - Example 2

list.java

```
Velocity.init();
VelocityContext context = new VelocityContext();
context.put("customerSales", new CustSalEnum(...) );
context.put("customer", this->getCustomer());
Template template = null;
try {
   template = Velocity.getTemplate("myTemplate.vm"); }
catch( ResourceNotFoundException rnfe ) {}
catch( ParseErrorException pee ) {}
catch( MethodInvocationException mie ) {}
catch( Exception e ) {}
StringWriter sw = new StringWriter();
template.merge( context, sw )
```

Model Design

- Represents business data
- Implements business logic
- Model API and technology are important design considerations.
- Often used Enterprise Java Beans (bigger applications) or JavaBeans (simpler applications)

Model Design

User.java

```
class User {
  private String login;
  private String password;
  private String name;
...
  public User() {
  }
  public String getLogin() {
    return login;
  }
  public void setLogin(String login) {
    this.login = login;
  }
```

Outline

J2EE BluePrints Web Application Framework

- Suitable for small, non-critical applications, and for learning

Features:

- Front Controller servlet
- Abstract action class
- Templating service
- Generic custom tags
- Internationalization support

Jakarta Struts

- Powerful, industrial-strength framework suitable for large applications

Additional features:

- Highly configurable
- Utility classes for XML
- Automatic population of server-side JavaBeans
- Web forms with validation
- Custom tags for accessing server-side state, creating HTML, presentation and templating

Online:http://jakarta.apache.org/struts

Jakarta Turbine

- Similar to Struts, but has richer set of features

Features:

- Integrates with other templating engines (Cocoon, WebMacro, FreeMarker, Velocity)
- Caching services,
- Integration with Object-Relational Mapping tools
- Job scheduler
- Based on "Model 2 + 1"

Online:http://jakarta.apache.org/turbine

Enhydra Barracuda

- Combination of MVC and Component frameworks
- Supports three different content-generation techniques:
  - Simple Servlets,
  - Template Engines (JSP, TeaServlet, WebMacro, FreeMarker, Velocity),
  - Document Object Model manipulation
- Provides better component composition and reuse than Turbine of Struts
- :-(Still not a standard

Online:

http://www.barracudamvc.org

New Framework - JavaServer Faces

- currently under development
- architecture ans a set of APIs for dispatching requests to JavaBeans, maintaining stateful, server-side components, supporting i18n, validation, multiple client types
- currently offers means for creating general content

Online:

http://java.sun.com/j2ee/javaserverfaces

New Pattern - Active View

- New pattern in the area of web application business layer
- Intent: Manage a Web active (dynamic) view
- Refocuses the Observer pattern in a Web architecture
- Communication between subservlet and observlet uses standard protocol

Online:

http://shrike.depaul.edu/~darchamb/