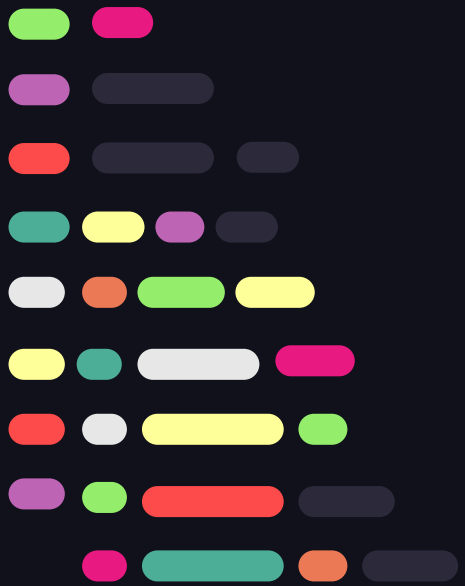
Java Programming Week-09

<Mohan De Zoysa –IIT JAVA CERTIFICATION>

Outline



Java API’s

- HTTP Servlet Basics
- Request and Response
- JavaServer Pages (JSP)
- MVC

HTTP Servlet Basics

- javax.servlet provides the foundation for servlet development
- javax.servlet.http builds upon it by adding classes and interfaces that are specific to handling HTTP requests and responses.
- Developers typically use both packages in conjunction when creating web applications in Java,
- javax.servlet for common servlet functionality and javax.servlet.http for handling HTTP-related tasks.

javax.servlet

Support generic, protocol-independent servlets
Servlet (interface)
GenericServlet (class)
service()
ServletRequest and ServletResponse
Provide access to generic server requests and responses

javax.servlet.http

Extended to add HTTP-specific functionality
HttpServlet (extends GenericServlet)
doGet()
doPost()
HttpServletRequest and HttpServletResponse
Provide access to HTTP requests and responses

User-defined Servlets

Inherit from HttpServlet

- Override doGet() and doPost()
- To handle GET and POST requests
- Have no main() method

doGet() and doPost()

```
protected void doGet(HttpServletRequest req,HttpServletResponse resp)

protected void doPost(HttpServletRequest req,HttpServletResponse resp)
```

Hello World

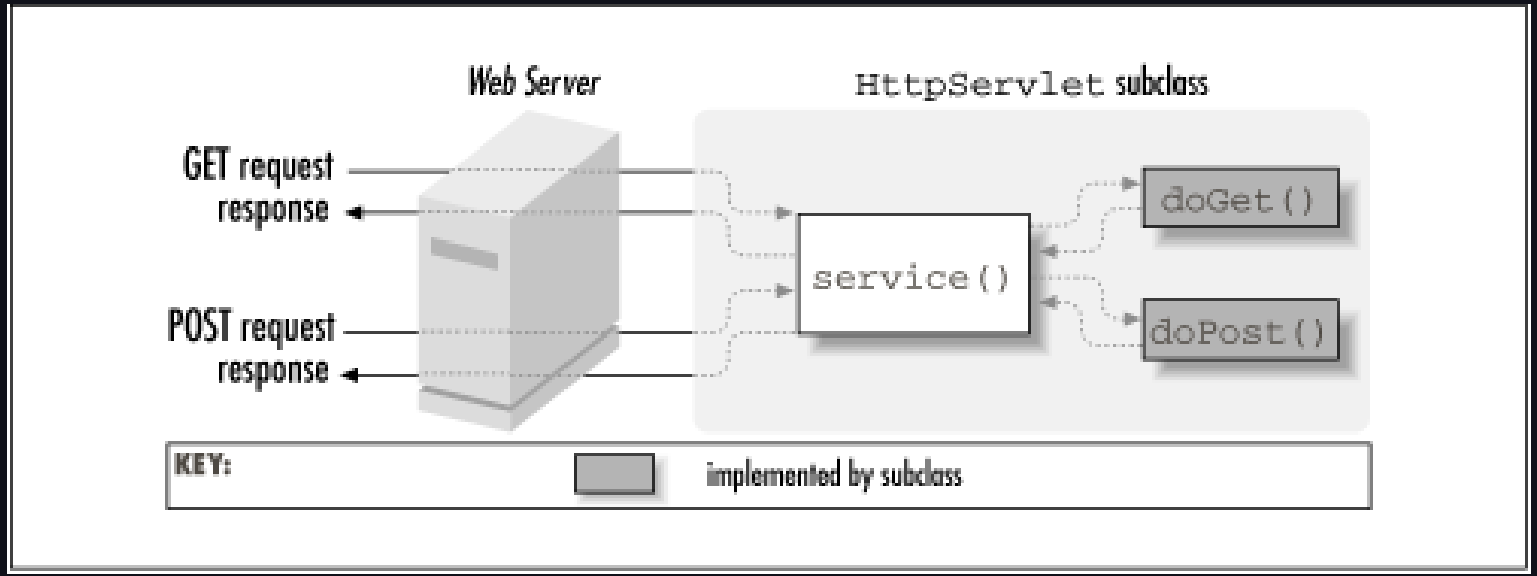
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloWorld extends HttpServlet {

public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
res.setContentType("text/html");
PrintWriter out = res.getWriter();
out.println("<html>");
out.println("<head><title>hello world</title></head>");
out.println("<body>");
out.println("<big>hello world</big>");
out.println("</body></html>");
}

}
```

Hello World Cont.

```
public void doPost(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
doGet(req, res);
}
```

```
<servlet>
  <servlet-name>product</servlet-name>
  <servlet-class>com.controller.ProductController</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>product</servlet-name>
  <url-pattern>/product</url-pattern>
</servlet-mapping>
```

Handling GET and POST Request



End-to-end Process

Client

- Makes a request to a servlet

Web Server

- Receives the request
- Identifies the request as a servlet request
- Passes the request to the servlet container

Servlet Container

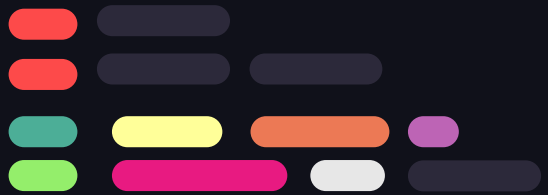
- Locates the servlet
- Passes the request to the servlet

Servlet

- Executes in the current thread
- The servlet can store/retrieve objects from the container
- Output is sent back to the requesting browser via the web server
- Servlet continues to be available in the servlet container

{..

Request and Response



}..

HttpServlet Request

Encapsulate all information from the client request

- HTTP request header and request body

Methods to retrieve data

- Inherited from ServletRequest
  - getParameter()
  - getParameterNames()
  - getParameterValues()
  - getInputStream()
  - getReader()

HttpServletResponse

HttpServletResponse

Encapsulate all data to be returned to client

HTTP response header and response body (optional)

Set HTTP response header

Primitive manipulation

setStatus(), setHeader(), addHeader()

Convenience methods

setContentType(), sendRedirect(), sendError()

Set HTTP response Body

Obtain a PrintWriter or ServletOutputStream to return data to the client

getWriter(), getOutputStream()

GET vs. POST

GET

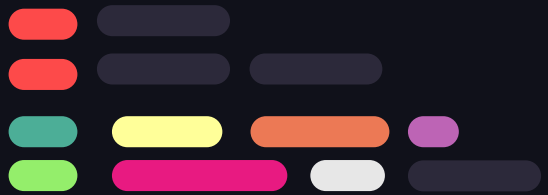
- All form parameters are embedded in the URL
- If you reload, or bookmark and return, the query will get executed a second time with the same parameters
- Bad if page is a credit card order confirmation – 2 charges!
- Use GET to obtain info

POST

- Form parameters are included in the request body
- On reload
  - Browser will ask if it should re-post
  - On bookmark and return
  - Query will proceed with no parameters
  - Use POST to change state

{ . .

JavaServer Pages (JSP)



} . .

JavaServer Pages

Embed Java servlet code in HTML pages

Purposes

“Enable the separation of dynamic and static content”

“Enable the authoring of Web pages that create dynamic content easily but with maximum power and flexibility”

Server automatically creates, compiles, loads, and runs a special servlet for the page

Simple JSP Example

```
<html>
<head>
<title>JSP Example</title>
</head>
<body>
<h1>JSP Example</h1>
<hr/>
<p>
<% out.println("Hello " + request.getParameter("name")); %>
<%= "Hello again, " + request.getParameter("name") + "!"%>
</p>
</body>
</html>
```

Elements in JSP

XHTML

Scriptlet

<% servlet code for service() %>

<%-- comment --%>

Expressions and declarations

<%= expression %>

Eliminates the clutter of out.println()

<%! code for outside of service() %>

Declare static or instance variables, and define new methods

Directives

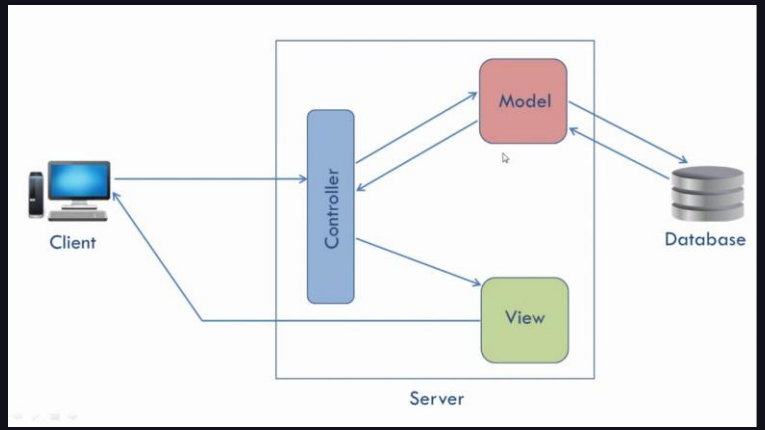
Control different aspects of the workhorse servlet

<%@ directiveName attribName=“attribValue” %>

<%@ page contentType=“text/plain” %>

{ . .

MVC - Model View Controller Design Pattern



} . .

MVC stands for Model-View-Controller, which is a design pattern commonly used in software development for building user interfaces and organizing code in a structured and modular way. It separates an application into three interconnected components, each with its specific responsibilities:

1 **Model:**

1. The Model represents the application's data and business logic. It encapsulates the data and the rules for manipulating and processing tha data.
2. It is responsible for retrieving and storing data from and to a database, as well as performing any necessary calculations or data transformations.
3. The Model component is independent of the user interface and user interaction.

2.**View:**

1. The View is responsible for presenting the data to the user and displaying the user interface.
2. It is concerned with the presentation and user experience aspects of the application.
3. In web development, the View typically generates the HTML, CSS, and JavaScript required for rendering the user interface.

3 **Controller:**

1. The Controller acts as an intermediary between the Model and the View.
2. It receives user input, such as mouse clicks or button presses, and processes it.
3. Based on the input, the Controller interacts with the Model to update data or retrieve data, and it also updates the View to reflect any changes in the Model.
4. It contains the application's control logic and handles user interactions.