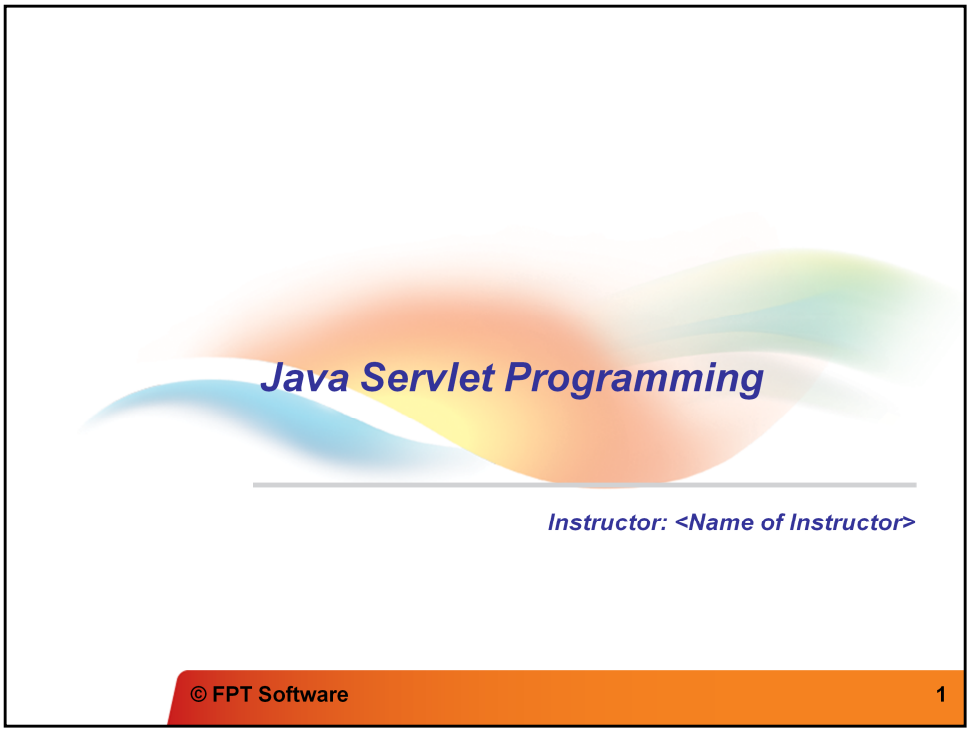


Latest updated by: HanhTT1

Agenda

- Introduction
- Servlet Overview and Architecture
- Handling HTTP get Requests
- Handling HTTP post Requests
- Servlet Context

Introduction

- Java networking capabilities
  - Socket-based and packet-based communications
    - Package `java.net`
  - Remote Method Invocation (RMI)
    - Package `java.rmi`
  - Servlets and Java Server Pages (JSP)
    - Request-response model
    - Packages `javax.servlet`
      `javax.servlet.http`
      `javax.servlet.jsp`
      `javax.servlet.tagext`
    - Form the Web tier of J2EE
- Servlets are modules of Java code that run in a server application to answer client requests.

Servlet Overview and Architecture

- Typical uses for Servlets include:
  - Processing and/or storing data submitted by an HTML form.
  - Providing dynamic content, e.g. returning the results of a database query to the client.
  - Managing state information on top of the stateless HTTP, e.g. for an online shopping cart system which manages shopping carts for many concurrent customers and maps every request to the right customer
- Servlets make use of the Java standard extension classes in the packages javax.servlet (the basic Servlet framework) and javax.servlet.http (extensions of the Servlet framework for Servlets that answer HTTP requests).
- Servlet container (servlet engine)
  - Server that executes a servlet

Servlet Overview and Architecture
Interface Servlet and the Servlet Life Cycle

- Interface **`Servlet`**
  - All servlets must implement this interface
  - All methods of interface **`Servlet`** are invoked automatically
- Servlet life cycle
  - Servlet container invokes the servlet's **`init`** method
  - Servlet's **`service`** method handles requests
  - Servlet's **`destroy`** method releases servlet resources when the servlet container terminates the servlet

Servlet Overview and Architecture
HttpServlet Class

- Overrides method **`service`**
- Two most common HTTP request types: get and post
- Method **`doGet`** responds to **`get`** requests
  - Retrieve the content of a URL
- Method **`doPost`** responds to **`post`** requests
  - Post data from an HTML form to a server-side form handler
  - Browsers cache Web pages
- **HttpServletRequest** and **`HttpServletResponse`** objects
  - Created by the servlet container and passed as an argument to the servlet's service methods (doGet, doPost, etc.)
  - **HttpServletRequest** object contains request from the client,
  - **HttpServletResponse** object provides HTTP-specific functionality in sending a response (access HTTP headers, cookies, etc.)

Handling HTTP get Requests
Example: `WelcomeServlet`

```
1  // WelcomeServlet.java
2  // A simple servlet to process get requests.
3  package com.alliant.test.servlets;
4
5  import javax.servlet.*;
6  import javax.servlet.http.*;
7  import java.io.*;
8
9  public class WelcomeServlet extends HttpServlet {
10
11    // process "get" requests from clients
12    protected void doGet( HttpServletRequest request,
13                          HttpServletResponse response )
14          throws ServletException, IOException
15    {
16       response.setContentType( "text/html" );
17       PrintWriter out = response.getWriter();
18
19       // send XHTML page to client
20
21       // start XHTML document
22       out.println( "<?xml version = \"1.0\"?>" );
23
24       out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD " +
25          "XHTML 1.0 Strict//EN\" \"http://www.w3.org" +
26          "/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );
27
28       out.println(
29          "<html xmlns = \"http://www.w3.org/1999/xhtml\">" );
```

Import the **javax.servlet** and **javax.servlet.http** packages.

`WelcomeServlet` that responds to a simple `HTTP get` request.

Extends **HttpServlet** to handle HTTP **get** requests and HTTP **post** requests.

Override method **doGet** to provide custom get request processing.

Uses the **response** object's **getWriter** method to obtain a reference to the **PrintWriter** object that enables the servlet to send content to the client.

Create the XHTML document by writing strings with the **out** object's **println** method.

Handling HTTP get Requests Example: `WelcomeServlet`

```
30
31        // head section of document
32        out.println( "<head>" );
33        out.println( "<title>A Simple Servlet Example</title>" );
34        out.println( "</head>" );
35
36        // body section of document
37        out.println( "<body>" );
38        out.println( "<h1>Welcome to Servlets!</h1>" );
39        out.println( "</body>" );
40
41        // end XHTML document
42        out.println( "</html>" );
43        out.close();  // close stream to complete the page
44     }
45  }
```

`WelcomeServlet` that responds to a simple HTTP get request.

Closes the output stream, flushes the output buffer and sends the information to the client.

Handling HTTP get Requests Example: `WelcomeServlet`

```
1   <?xml version = "1.0"?>
2   <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5   <!-- WelcomeServlet.html -->
6
7   <html xmlns = "http://www.w3.org/1999/xhtml">
8   <head>
9      <title>Handling an HTTP Get Request</title>
10  </head>
11
12  <body>
13     <form action = "/test/welcome1" method = "get">
14
15        <p><label>Click the button to invoke the servlet
16           <input type = "submit" value = "Get HTML Document" />
17        </label></p>
18
19     </form>
20  </body>
21  </html>
```

HTML document in which the form's action invokes WelcomeServlet through the alias welcome1 specified in web.xml.

Handling HTTP get Requests
Setting Up the Apache Tomcat Server

- Download Tomcat, latest version, from below link
  - http://tomcat.apache.org
- Define environment variables
  - JAVA_HOME
  - CATALINA_HOME
- Start the Tomcat server:
  - CATALINA_HOME\bin\startup.bat
- Access default web applications via
  - http://localhost:8080/
- Shutdown the Tomcat server:
  - CATALINA_HOME\bin\shutdown.bat
- Further information about configuring and running Tomcat can be found on the website (http://tomcat.apache.org)

Handling HTTP get Requests
Deploy a Web application

- Web applications
  - JSPs, servlets and their supporting files
- Deploying a Web application
  - Directory structure
    - Context root
  - Web application archive file (WAR file) OR Web application folder
  - Deployment descriptor
    - `web.xml`

Handling HTTP get Requests
Deploy a Web application (cont.)

Directory	Description
context root	This is the root directory for the Web application. The name of this directory is chosen by the Web application developer. All the JSPs, HTML documents, servlets and supporting files such as images and class files reside in this directory or its subdirectories. The name of this directory is specified by the Web application creator. To provide structure in a Web application, subdirectories can be placed in the context root. For example, if your application uses many images, you might place an images subdirectory in this directory.
`WEB-INF`	This directory contains the Web application *deployment descriptor* (**`web.xml`**).
`WEB-INF/classes`	This directory contains the servlet class files and other supporting class files used in a Web application. If the classes are part of a package, the complete package directory structure would begin here.
`WEB-INF/lib`	This directory contains Java archive (JAR) files. The JAR files can contain servlet class files and other supporting class files used in a Web application.

Web application standard directories.

Handling HTTP get Requests
Deploy a Web application (cont.)

```
1   <!DOCTYPE web-app PUBLIC
2      "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
3      "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
4
5   <web-app>
6
7      <!-- General description
8      <display-name>
9         Sample
10        Servlet Examples
11     </display-name>
12
13     <description>
14        This is the Web application in which we
15        demonstrate our JSP and Servlet examples.
16     </description>
17
18     <!-- Servlet definition
19     <servlet>
20        <servlet-name>welcome1</servlet-name>
21
22        <description>
23           A simple servlet that handles an HTTP get request.
24        </description>
25
26        <servlet-class>
27           com.alliant.test.servlets.WelcomeServlet
28        </servlet-class>
29     </servlet>
30
```

Deployment descriptor (web.xml) for the Web ...on.

Element **web-app** defines the configuration of each servlet in the Web application and the servlet mapping for each servlet.

Element **display-name** specifies a name that can be displayed to the administrator of the server on which the Web application is installed.

Element **description** specifies a description of the Web application that might be displayed to the administrator of the server.

Element **servlet** describes a servlet.

Element **servlet-name** is the name for the servlet.

Element **description** specifies a description for this particular servlet.

Element **servlet-class** specifies compiled servlet's fully qualified class name.

Handling HTTP get Requests
Deploy a Web application (cont.)

```
31      <!-- Servlet mappings -->
32      <servlet-mapping>
33         <servlet-name>welcome1</servlet-name>
34         <url-pattern>/welcome1</url-pattern>
35      </servlet-mapping>
36
37   </web-app>
```

Element **servlet-mapping** specifies **servlet-name** and **url-pattern** elements.

ment
ptor
ml) for
the Web
application.

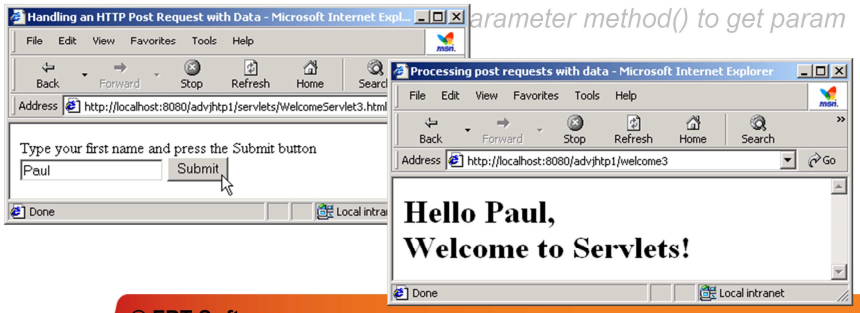
- Invoke **`WelcomeServlet`** example
  - ✓ /test/welcome1
    - **`/test`** specifies the context root
    - **`/welcome1`** specifies the URL pattern
- Web application directory, file structure
  - ✓ test

```
servlets\WelcomeServlet.html
WEB-INF
    web.xml
    classes\com\test\servlets
        WelcomeServlet.class
```

- URL pattern formats
  - ✓ Exact match
    - /test/welcome1
  - ✓ Path mappings
    - /test/example/*
  - ✓ Extension mappings
    - *.jsp
  - ✓ Default servlet
    - /test/example/

Handling HTTP post Requests

- HTTP post request
  - Post data from an HTML form to a server-side form handler
  - Browsers cache Web pages
- Practices time:
  - *Let's try to see differences between HTTP post and HTTP get requests*



Servlet Context

- Defined by an object of ServletContext type.
- Defines a servlet's view of the web application within which the servlet is running.
- Allows a servlet to access resources available to it.
- Using such an object, a servlet can log events, obtain URL references to resources, and set and store attributes that other servlets in the context can use.

Servlet Context
Sample

- We can change the init() method of the SurveyServlet as below:

```
// set up database connection and prepare SQL statements
public void init( ServletConfig config ) throws ServletException
{
   String dbDriver, dbURL;
   ServletContext context = config.getServletContext();

   dbDriver = context.getInitParameter("DB_Driver");
   dbURL = context.getInitParameter("DB_URL");

   // attempt database connection and create PreparedStatements
   // ...
}
```

- Then we need to amend the web.xml file to specify the initial context parameters:

```
<context-param>
    <param-name>DB_URL</param-name>
    <param-value>jdbc:mysql://localhost:3306/test</param-value>
</context-param>

<context-param>
    <param-name>DB_Driver</param-name>
    <param-value>com.mysql.jdbc.Driver</param-value>
</context-param>
```

Servlet Context
Servlet Lifecycle Events

- From Java Servlet 2.3
- New events framework
- More global control than any one servlet or JSP can provide
- Support event notifications for state changes in ServletContext and HttpSession objects
- Scope
  - ServletContext: manage state held at a VM level for the application
  - HttpSession: manage state or resources associated with a series of requests from the same user/session
  - In distributed containers, one listener instance /deployment descriptor declaration / java VM

Servlet Context
ServletContext and HttpSession

- Interesting things on the servlet contexts:
  - Manage
  - Startup/shutdown
  - Attribute changes
- Interesting events on HTTP sessions:
  - Creation and invalidation
  - Changes in attributes
  - Migration across distributed containers
- Attribute changes to both objects may occur concurrently
  - No synchronization support in container
  - Listener classes need to support data integrity

Servlet Context
Java Servlet 2.3 API - Listening Interfaces

- ServletContextListener
  - contextInitialized/Destroyed(ServletContextEvent)
- ServletContextAttributeListener
  - attributeAdded/Removed/Replaced(ServletContextAttributeEvent)
- HttpSessionListener
  - sessionCreated/Destroyed(HttpSessionEvent)
- HttpSessionAttributeListener
  - attributedAdded/Removed/Replaced(HttpSessionBindingEvent)

Servlet Context
Basic Steps for Implementing Event Listeners

- Create a new class that implements the appropriate interface
- Override the methods needed to respond to the events of interest
- Obtain access to the important Web application objects
  - Servlet context
  - Servlet context attribute, its name and value
  - Session, session attribute name and value
- Use these objects
  - e.g. Servlet context: getInitParameter(), setAttribute() and getAttribute()
- Declare the listener: configure listener, listener-class in web.xml
- Provide any needed initialization parameters

Servlet Context
Example – A Session Counter Listener

- **Implement the session counter:**

```
/*
    SessionCounter.java
*/
import javax.servlet.http.HttpSessionListener;
import javax.servlet.http.HttpSessionEvent;

public class SessionCounter implements HttpSessionListener {

    private static int activeSessions = 0;

    public void sessionCreated(HttpSessionEvent se) {
        activeSessions++;
    }

    public void sessionDestroyed(HttpSessionEvent se) {
        if(activeSessions > 0)
            activeSessions--;
    }

    public static int getActiveSessions() {
        return activeSessions;
    }
}
```

Servlet Context
Example – A Session Counter Listener (cont.)

- **Show session counter status:**

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ActiveSessions extends HttpServlet
{
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
                throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        out.println("<html>");
        out.println("<head>");
        out.println("<title>Active Sessions</title>");
        out.println("</head>");
        out.println("<body bgcolor=\"white\">");
        out.println("<p>Number of sessions currently in the memory: "
                    + SessionCounter.getActiveSessions() + "</p>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

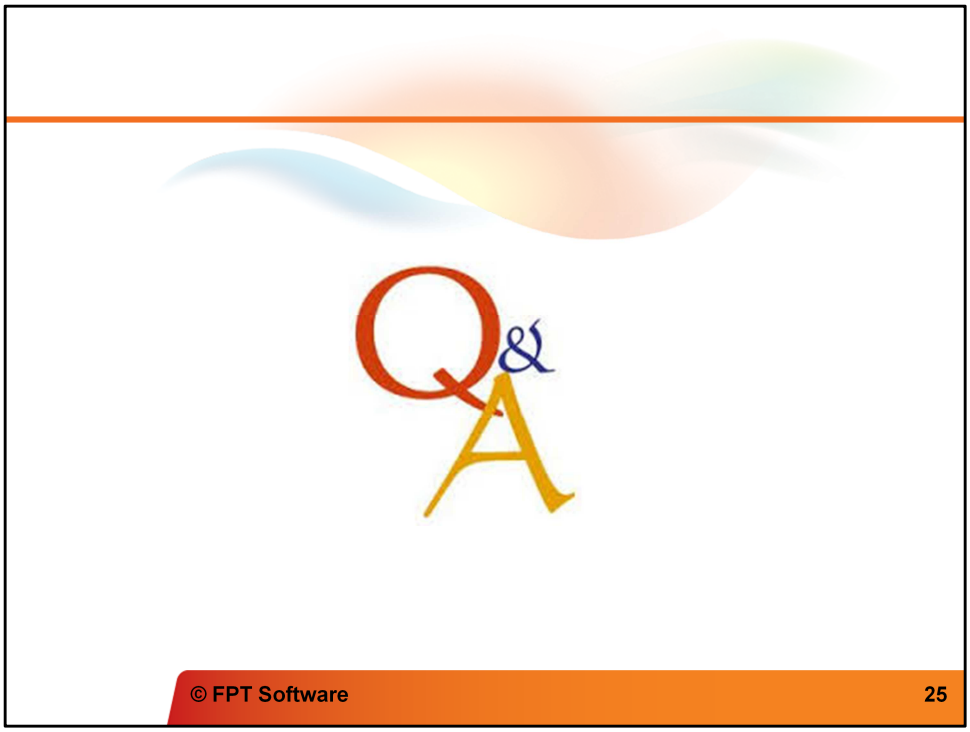
Servlet Context

Example – A Session Counter Listener (cont.)

- **Configure the web.xml**

```
<servlet>
    <servlet-name>ActiveSessions</servlet-name>
    <servlet-class>ActiveSessions</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>ActiveSessions</servlet-name>
    <url-pattern>/servlet/activesessions</url-pattern>
</servlet-mapping>

 <!-- listeners -->
 <listener>
    <listener-class>SessionCounter</listener-class>
 </listener>
```


Q&A
© FPT Software
25