



Cognizant
Technology
Solutions



Cognizant
Academy



Introduction to Servlets

Basic

C3: Protected

About the Author

Created By:	Mohamed, Hajura (Cognizant), 117960
Credential Information:	Executive – Cognizant Academy
Version and Date:	J2EE/PPT/0306/1.0

Cognizant Certified Official Curriculum



Icons Used



Questions



Hands-on Exercise



A Welcome Break



Test Your Understanding



Coding Standards



Reference



Demo









Key Contacts

Introduction:

Initially, Common Gateway Interface (CGI) scripts were used to generate dynamic content. Although widely used, CGI scripting technology has a number of shortcomings, including platform dependence and lack of scalability. To address these limitations, Java servlet technology was created as a portable way to provide dynamic, user-oriented content.

Objective:

After completing this chapter, you will be able to:

-  Explain a servlet
-  Explain the servlet life cycle
-  Describe the process of information sharing
-  Initialize a servlet
-  Apply servlet for getting information from requests and constructing responses
-  Invoke other Web resources

What is a Servlet?

- Servlets are Java objects used to process server-side requests and generate response for clients.
- Servlets typically execute within the context of an HTTP Web server.
- Servlets are commonly used to extend the applications hosted by Web servers through a request-response programming model.

The functionality of servlets are:

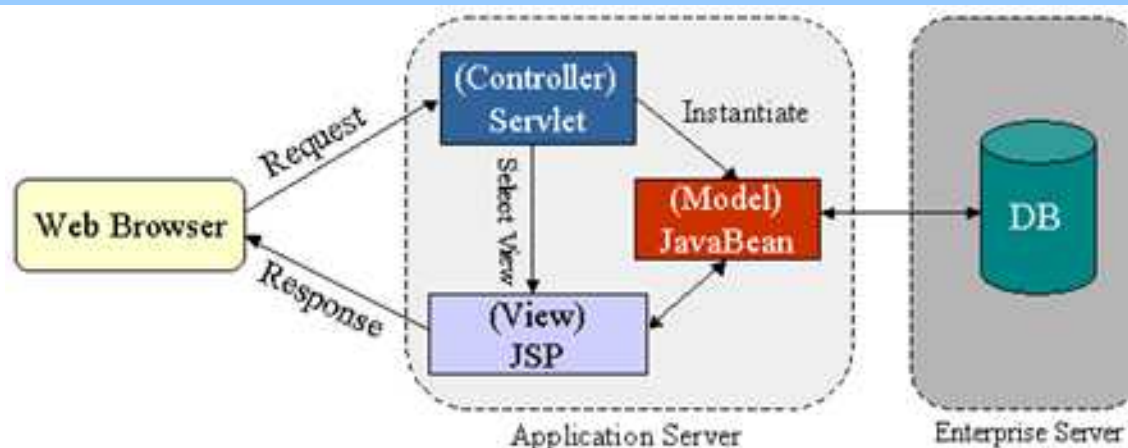
- Receive client request (mostly in the form of HTTP request)
- Extract the information from the request
- Do content generation or business logic processing (possibly by computing, accessing database, invoking EJBs, and so on)
- Create and send responses to clients (mostly in the form of HTTP response) or forward the request to another servlet

Advantages of Servlet

The following are the advantages of servlet:

- No CGI limitations
- Abundant third-party tools and Web servers that support servlets
- Access to the entire family of Java APIs
- Reliable, better performance, and scalability
- Platform and server independent
- Secure
- Most servers allow automatic reloading of servlets by administrative action

Request: Request Paradigm



This model promotes the use of the Model View Controller (MVC) architectural style design and pattern:

- JSP pages are used for the presentation layer and servlets for processing tasks.
- The servlet acts as a controller responsible for processing requests and creating any beans needed by the JSP page.
- The controller is also responsible for deciding to which JSP page to forward the request.
- The JSP page retrieves objects created by the servlet and extracts dynamic content for insertion within a template.

- **The `init()` method:**

- Is invoked after the servlet is loaded and instantiated but before the servlet is delivered
- Can be used to:
 - Read persistent data configurations
 - Initialize resource
 - Perform any other one time activity

Example

```
public class CatalogServlet extends HttpServlet {  
    private BookDBAO bookDB;  
    public void init() throws ServletException {  
        bookDB = (BookDBAO)getContext()  
            .getAttribute("bookDB");  
        if (bookDB == null) throw new  
            UnavailableException("Couldn't get database.");  
    }  
}
```

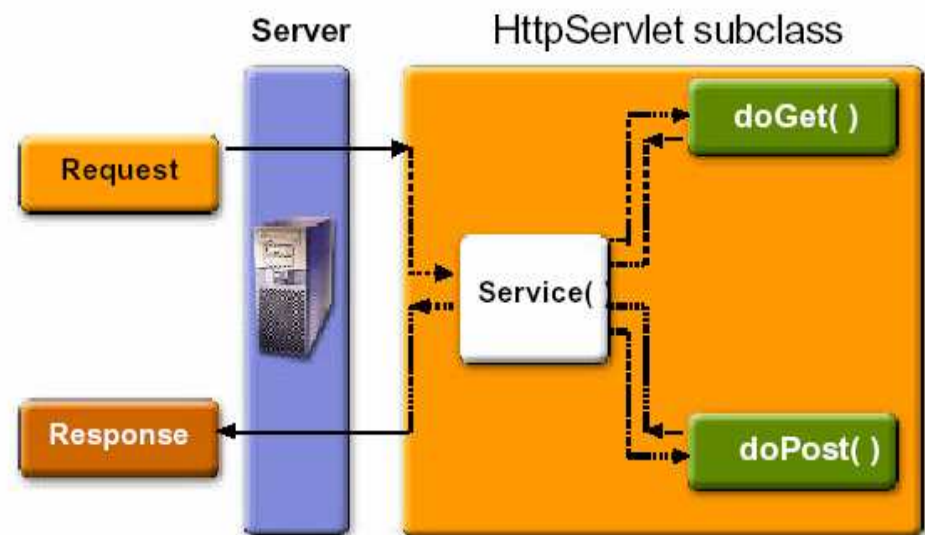
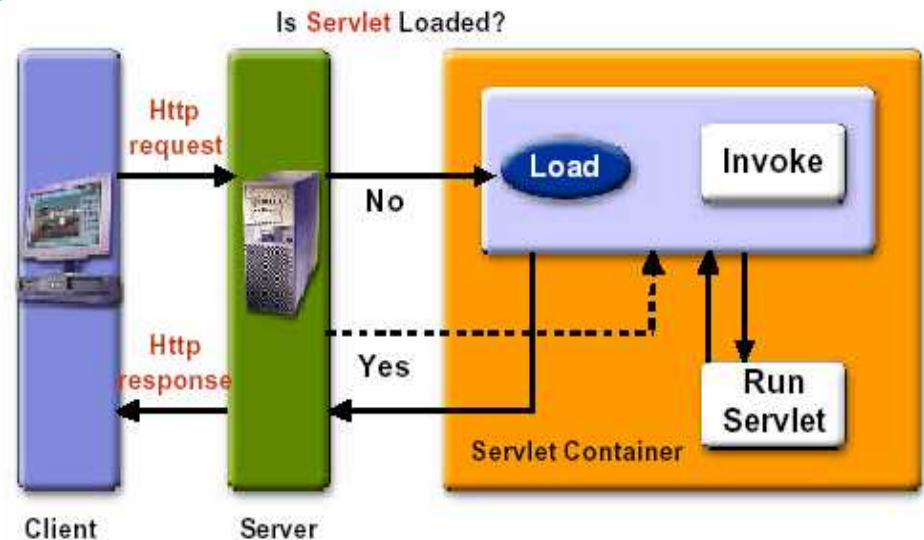


The Servlets Life Cycle



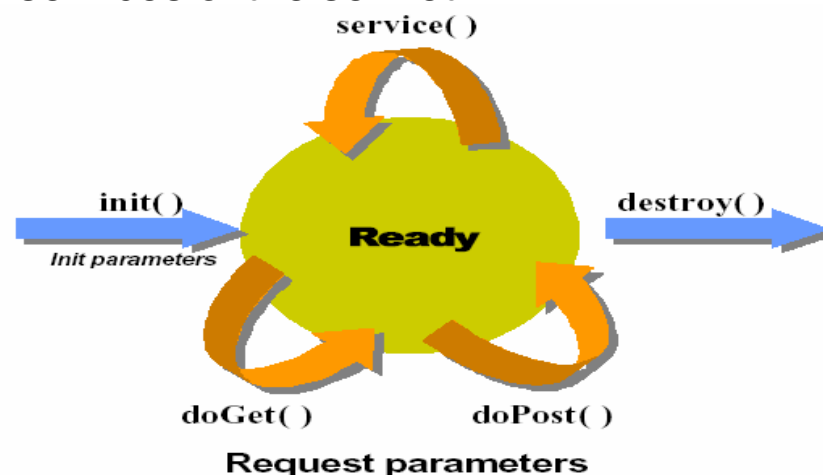
When a request is mapped to the servlet, the container:

1. Looks for the instance of the servlet class
2. If it does not exist, container loads the servlet class and instantiates it
3. Initializes the servlet class by calling the `init` method
4. Invokes the service method when the servlet needs to be removed the container
5. Terminates the servlet by calling the `destroy` method



The Servlets Life Cycle (Contd.)

- Servlets are loaded by the servlet engine either when called for the first time (delayed loading) or when the servlet engine starts (preloading).
- Preloading is a feature often provided by servlet engines that can remove the dreaded "first-request" speed bump. It does not affect servlets or how you write them in any way.
- Once a servlet class is instantiated, the `init` method is immediately called with a `ServletConfig` object optionally passed as a parameter. The `ServletConfig` object carries, among other things, parameters that are set by the user of the system.
- A servlet is inactive until a client-generated request arrives at the server that specifically requests the services of the servlet.



Web components, like most objects, usually work with other objects, such as scope objects, to accomplish their tasks. The following table lists the ways they can do this.

Scope Object	Class	Accessible From
Web context	<code>javax.servlet.ServletContext</code>	Web components within a Web context
Session	<code>javax.servlet.http.HttpSession</code>	Web components handling a request that belongs to the session
Request	<code>javax.servlet.HttpServletRequest</code>	Web components handling the request
Page	<code>javax.servlet.jsp.JspContext</code>	The JSP page that creates the object



Getting Information from Requests and Constructing Responses



Getting information from requests:

- A request contains data passed between a client and the servlet. All requests implement the `ServletRequest` interface.
- The `ServletRequest` interface defines methods for accessing the following information:
 - Parameters, which are typically used to convey information between clients and servlets
 - Object-valued attributes, which are typically used to pass information between the servlet container and a servlet, or between collaborating servlets
 - Information about the protocol used to communicate the request and about the client and server involved in the request
 - Information relevant to localization

Example:

```
String bookId = request.getParameter("Add"); /*Getting Information  
from Requests*/  
if (bookId != null)  
    BookDetails book = bookDB.getBookDetails(bookId);
```



Getting Information from Requests and Constructing Responses (Contd.)



Constructing Responses:

- Contains data passed from servlet to client
- All servlet responses implement `ServletResponse` interface to:
 - Retrieve an output stream
 - Indicate content type
 - Indicate whether to buffer output
 - Set localization information
- `HttpServletResponse` extends `ServletResponse` by using:
 - HTTP response status code
 - Cookies



Getting Information from Requests and Constructing Responses (Contd.)



Example:

```
public class DateRefresh extends HttpServlet
{
    public void doGet(HttpServletRequest req,
        HttpServletResponse res)
        throws ServletException, IOException
    {
        res.setContentType("text/plain");
        PrintWriter out = res.getWriter();
        res.setHeader("Refresh", "5");
        out.println(new Date().toString());
    }
}
```

Invoking Other Web Resources

Web components can invoke other Web resources in two ways:

- **Invoking other Web resources indirectly:**

A Web component indirectly invokes another Web resource by embedding a URL that points to another Web component in the content returned to a client.

Example:

`MytServlet` indirectly invokes the `ShoppingServlet` through the embedded URL `/mystore/shop`.

- **Invoking other Web resources directly:**

For invoking Web resources directly, the Web component can include the content of another resource, or it can forward a request to another resource:

- Obtaining the `RequestDispatcher` object from either a request or the Web context. The only difference is that a request can take a relative path (a path that does not begin with a `/`), but the Web context requires an absolute path.
- Getting a reference to `RequestDispatcher` interface:
 - `ServletContext.getRequestDispatcher(String resource)`
 - `ServletRequest.getRequestDispatcher(String resource)`

Including other resources in the response

```
RequestDispatcher dispatcher =  
getServletContext().getRequestDispatcher("/shop")  
if (dispatcher != null)  
    dispatcher.include(request, response);
```

where /shop is the context root.

Transferring control to another Web component

```
RequestDispatcher dispatcher =  
getServletContext().getRequestDispatcher("/shop");  
if (dispatcher != null)  
    dispatcher.include(request, response);
```

Web.XML Configuration

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Example configuration for Servlet 2.2 Containers -->
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
  <description>Your application name.</description>
  <servlet>
    <servlet-name>FilterServlet</servlet-name>
    <servlet-class>org.chwf.servlet.filter.FilterServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>FilterServlet</servlet-name>
    <url-pattern>*.cmd</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>FilterServlet</servlet-name>
    <url-pattern>*.view</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>index.view</welcome-file>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>
</web-app>
```

This servlet is at this location

When server sees a path, like this, in url , call this servlet whose location it knows already.

Example source HTML for a form:

Maps to mapping in web.xml refer to an earlier slide

```
< FORM METHOD=POST ACTION= "test.view" >  
  Enter your name:< INPUT <INPUT TYPE=text NAME=your_name>  
  < INPUT TYPE=submit VALUE="Test this form" >  
< /FORM >
```

- When the user clicks the **Submit** button, the browser collects the values of each of the input fields (text typed by the user, and so on) and sends them to the Web server identified in the ACTION keyword of the FORM open tag.
- The Web server passes that data to the program identified in the ACTION, using the METHOD specified.
- Action is mapped in the web.xml file to call the respective servlet to handle the user request. Otherwise Form "Action" can directly be mapped to open another HTML page.

Note: Each form must have exactly one field of the submit type.

Demo 1- HTML



```
<html>
<head><title>MyHTML</title>
<body>
<FORM  name="f1"  action="/myservlet.do"  method="POST"  >
  Type a Color: <br>
  <INPUT type="text" NAME="color" size="20">
Type a Flower: <br>
  <INPUT type="text" NAME="flower" size="20">
Type a Car: <br>
  <INPUT type="text" NAME="car" size="20">

  <INPUT type="submit" NAME="S1" size="20">
  </FORM>
</body>
</html>
```

Demo 1- Servlet



```
package com.cts.mypack;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
public class MyServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException{
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String title = "Reading The Color Parameter";
        out.println(ServletUtilities.headWithTitle(title) + "<BODY>\n" + "<H1
        ALIGN=CENTER>" + title + "</H1>\n" + "<UL>\n" + " <LI>param1: " +
        request.getParameter("color") + "\n" + " <LI>param2: " +
        request.getParameter("flower") + "\n" + " <LI>param3: " +
        request.getParameter("car") + "\n" + "</UL>\n" + "</BODY></HTML>");
    }
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        doGet(request, response);
    }
}
```

Demo 1- web.xml



```
<web-app>
  <description>XML containing descriptors for
  MyServlet</description>
  <servlet>
    <servlet-name>MyServlet</servlet-name>
    <servlet-class>com.cts.mypack.MyServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>MyServlet</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
</web-app>
```



```
<html>
<head><title>MyHTML</title>
<body>
<FORM  name="f1"  action="/dispatch.do"  method="POST"  >
  Type your name: <br>
  <INPUT type="text" NAME="color" size="20">
Type your age: <br>
  <INPUT type="text" NAME="flower" size="20">
Type your designation: <br>
  <INPUT type="text" NAME="car" size="20">

  <INPUT type="submit" NAME="S1" size="20">
  </FORM>
</body>
</html>
```

Demo 2 - Servlet



```
package com.cts.dispatch;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
public class Dispatcher extends HttpServlet {
public void doPost(HttpServletRequest request, HttpServletResponse
    response){
    RequestDispatcher dispatcher =
    request.getRequestDispatcher("/template.jsp");
        if (dispatcher != null)
    dispatcher.forward(request, response);    //invoking a web resource
    //or you can use
    //dispatcher.include(request, response);
}
```


Demo 2 - web.xml



```
<web-app>
  <description>XML containing descriptors for
  Dispatcher</description>
  <servlet>
    <servlet-name>DispatchServlet</servlet-name>
    <servlet-class>com.cts.dispatch.Dispatcher</servlet-
    class>
  </servlet>
    <servlet-mapping>
      <servlet-name>DispatchServlet</servlet-name>
      <url-pattern>*.do</url-pattern>
    </servlet-mapping>
</web-app>
```



Cognizant
Technology
Solutions



Cognizant
Academy

- **Allow time for questions from participants**



Introduction to Servlets: Summary

- The servlet technology was introduced to overcome CGI Limitations and to provide platform independence for Web Resident Programs.
- Servlets are Java objects used to process server-side requests and generate response for clients.
- The Request model promotes the use of the Model View Controller (MVC) architectural style design and pattern.
- The `init()` method is invoked after the servlet is loaded and instantiated but before the servlet is delivered.
- A servlet is inactive until a client-generated request arrives at the server that specifically requests the services of the servlet.
- Preloading is a feature often provided by servlet engines that can remove the dreaded "first-request" speed bump. It does not affect servlets or how you write them in any way.
- A request contains data passed between a client and the servlet. All requests implement the `ServletRequest` interface.
- Web components can invoke other Web resources in two ways:
 - Invoking other Web resources indirectly
 - Invoking other Web resources directly

Introduction to Servlets: Source

- http://java.sun.com/j2ee/tutorial/1_3-fcs/index.html
- Professional Java Server Programming J2EE Edition By Wrox Author Team

Disclaimer: Parts of the content of this course is based on the materials available from the Web sites and books listed above. The materials that can be accessed from linked sites are not maintained by Cognizant Academy and we are not responsible for the contents thereof. All trademarks, service marks, and trade names in this course are the marks of the respective owner(s).



Cognizant
Technology
Solutions



Cognizant
Academy



**You have successfully
completed
Introduction to Servlets.**