

Java Servlets 3.0

Lesson 7: Inter-Servlet
Communication

Lesson Objectives

- In this lesson, we will learn:
 - Introduction and Need for Inter-Servlet Communication
 - Server / Client side dispatch
 - Communication and sharing data between servlets



Lesson Objectives:

This lesson introduces Inter-Servlet communication. The lesson contents are:

Lesson 07: Inter-Servlet communication

7.1: Introduction and Need for Inter-Servlet Communication

7.2: Server / Client side dispatch

7.3: Communication and sharing data between servlets

7.1: Introduction and Need for Inter-Servlet Communication

Inter-servlet Communication

- In a web application, there are multiple web components running on the same server. A web component can invoke another resource while it is executing.
- Communication between servlets for exchanging data or sharing control is called inter-servlet communication
- Three reasons for using inter-servlet communication are
 - Direct servlet Manipulation
 - Servlet reuse
 - Servlet Collaboration



Copyright © Capgemini 2015. All Rights Reserved 3

Introducing Inter-Servlet Communication:

The communication between servlets is called Inter-Servlet Communication. In fact, servlet being a web component can invoke another web component like html, jsp and other servlets too. Servlets running together in the same server communicate with each other in several ways.

The three major reasons for using inter-servlet communication are:

Direct servlet manipulation : allows gaining access to the other currently loaded servlets and performing certain tasks (through the ServletContext object)

Servlet reuse : allows the servlet to reuse the public methods of another servlet

Servlet collaboration : requires to communicate with each other by sharing specific information (through method invocation). Basically servlets collaborate with each another for exchanging data and sharing control.

In this lesson, we shall be focusing on servlet collaboration

7.2: Server / Client side dispatch

Invoking Web Resources

- A web component can directly invoke another resource while it is executing, in two ways:
 - It can forward the request to another resource
 - The component can include the content of another resource
- To invoke a web resource available on the server, one must first obtain a `RequestDispatcher` object using either a request object or the `ServletContext`.
 - `RequestDispatcher` `ServletRequest.getRequestDispatcher(String path)`
 - `RequestDispatcher` `ServletContext.getRequestDispatcher(String path)`



Copyright © Capgemini 2015. All Rights Reserved 4

Introducing Inter-Servlet Communication:

Invoking other Web Resource:

`RequestDispatcher` is an interface, implementation of which defines an object that receives requests from the client and sends them to any resource (such as a servlet, HTML file, or JSP file) on the server. The servlet container creates the `RequestDispatcher` object which is used as a wrapper around a server resource located at a particular path or given by a particular name.

This interface is intended to wrap servlets, but a servlet container can create `RequestDispatcher` objects to wrap any type of resource.

`RequestDispatcher` object can be obtained from either a `Request Object` or the `ServletContext object` as seen in the method signatures above.

Let us first see the concept of dispatching with `ServletContext` object

7.2: Server / Client side dispatch

Concept of ServletContext

- The ServletContext is an interface which helps us to communicate with the servlet container.
 - The ServletContext is created by the container when the web application is deployed
 - There is only one context per web application. The information in the ServletContext will be common to all the components.
 - ServletContext object can be obtained by:
 - ServletContext context = `getServletContext();`



Copyright © Capgemini 2015. All Rights Reserved 5

RequestDispatcher Interface:

ServletContext:

There is only one ServletContext for the entire web application and the components of the web application can share it. The information in the ServletContext will be common to all the components.

Remember that each servlet will have its own ServletConfig. The ServletContext is created by the container when the web application is deployed and after that only the context is available to each servlet in the web application

7.2: Server / Client side dispatch

Creation of RequestDispatcher

- RequestDispatcher object can be obtained using `getRequestDispatcher()` method of either:
 - ServletRequest object
`RequestDispatcher ServletRequest.getRequestDispatcher(String path)`
 - ServletContext object
`public RequestDispatcher ServletContext.getRequestDispatcher(String path)`
- RequestDispatcher object can also be obtained using `getNamedDispatcher()` of ServletContext object :
 - RequestDispatcher `ServletContext.getNamedDispatcher(String name)`



Copyright © Capgemini 2015. All Rights Reserved 6

RequestDispatcher Interface:

Creation of RequestDispatcher:

The `ServletRequest.getRequestDispatcher()` can take a relative path while `ServletContext.getRequestDispatcher()` cannot (It can only take relative to the current context's root).

For example, with ServletRequest below statements are valid:

`request.getRequestDispatcher("./html/first.html")` - evaluated relative to the path of the request

`request.getRequestDispatcher("/html/first.html")` - evaluated relative to the root

With ServletContext only :

`context.getRequestDispatcher("/html/first.html")` is valid but not

`context.getRequestDispatcher("./html/first.html")` i.e. it can not evaluate a path other than context root.

The `ServletContext.getNamedDispatcher()` method takes a String argument indicating the name of a servlet known to the ServletContext. If a servlet is found, then it is wrapped with a RequestDispatcher object and the object is returned. If no servlet is associated with the given name, then the method returns null. (A servlet instance can determine its name using `ServletConfig.getServletName()`).

RequestDispatcher allows for direct communication between web resources.

7.2: Server / Client side dispatch

Creation of RequestDispatcher

- The ServletContext and ServletRequest methods that create RequestDispatcher objects using path information allow the optional attachment of query string information to the path.
- For example:

```
ServletContext context = getServletContext();  
String path = "/raisins.html?orderno=5";  
RequestDispatcher rd = context.getRequestDispatcher(path);
```



Copyright © Capgemini 2015. All Rights Reserved 7

RequestDispatcher Interface:

Creation of RequestDispatcher:

Parameters specified in the query string used to create the RequestDispatcher take precedence over other parameters of the same name passed to the included servlet. The parameters associated with a RequestDispatcher are scoped to apply only for the duration of the include or forward call.

7.2: Server / Client side dispatch

Using RequestDispatcher

- A RequestDispatcher object can forward a client's request to a resource or include the resource itself in the response back to the client. It defines two methods for performing the above tasks: forward() and include()
- For example:

```
@WebServlet("/Dispatcher")
public class Dispatcher extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res) {
        RequestDispatcher dispatcher =
            request.getRequestDispatcher("template.html");
        if (dispatcher != null)
            dispatcher.forward(request, response);
    }
}
```


RequestDispatcher Interface:

Using RequestDispatcher:

A RequestDispatcher object can forward a client's request to a resource or include the resource itself in the response back to the client. A resource can be another servlet, HTML file or a JSP file. Methods are:

```
public void forward(ServletRequest req, ServletResponse resp)
```

```
public void include(ServletRequest req, ServletResponse resp)
```

The **forward()** method of the **RequestDispatcher** interface may be called by the calling servlet only when no output has been committed to the client. If non-committed output data exists in the response buffer, the content must be cleared before the target servlet's service method is called. Else, an **IllegalStateException** will be thrown.

In the case of **forward()**, once another web resource is being invoked, we cannot access the previous web resource.

include() includes the content of a resource in the response. The included servlet cannot change the response status code or set headers; any attempt to make a change is ignored.

The target servlet of the **include()** method has access to all aspects of the request object, but its use of the response object is more limited. It can only write information to the **ServletOutputStream** or **Writer** of the response object and commit a response by writing content past the end of the response buffer, or by explicitly calling the **flushBuffer()** method of the **ServletResponse** interface. It cannot set headers or call any method that affects the headers of the response.

In the case of **include()**, once another web resource is being invoked, we can come back with the response embedded to access the previous web resource.

Working with the RequestDispatcher is called as Server side dispatch.

This is because the container is responsible to either forward request to other servlet or include the response of another servlet.

In both of these methods URL pattern does not change. Thus client is not aware which servlet is processing the request.

We introduce multiple servlets here as a single servlet may not be able to complete the request processing life cycle, or we may require it for load balancing.

7.2: Server / Client side dispatch

Sending Request & Response objects

- The ServletRequest object also exposes three methods to pass/retrieve/remove parameters between requests.
 - `setAttribute()`
 - `getAttribute()`
 - `removeAttribute()`
- For eg:

```
protected void doGet(HttpServletRequest req, HttpServletResponse
    res) {
    req.setAttribute("exception", new ItemNotFoundException());
    getServletContext().getRequestDispatcher("/errorservlet")
        .forward(req,
    res);
}
```



Copyright © Capgemini 2015. All Rights Reserved 10

Communication with Server Resources:

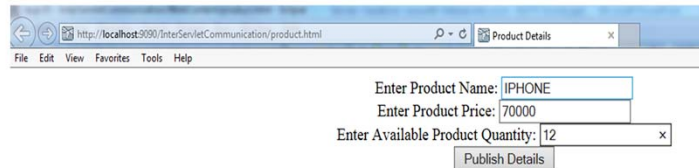
Notice that the `forward()` and `include()` methods have the Request and Response objects as parameters. Thus one servlet can easily pass its Request and Response objects to the target servlet, and thus any request parameters too. However, if user wishes to pass more request parameters and also between just a request, use the `setAttribute()` method of the **ServletRequest object**. The ServletRequest object also exposes the `getAttribute()` and `removeAttribute()` methods, the likes of which has been discussed earlier.

See the code snippet in the slide above. The `setAttribute()` method sets an attribute named "exception" to a new instance of an exception – **ItemNotFoundException**. The `forward()` method while carrying request information will also carry this new attribute to the target servlet!

Thus to communicate among servlets and pass data between them we can make use of **Request object and** its API methods mentioned in the slide.

Demo

- Creating request attributes and retrieving them using [set/get]Attribute method
- ProductServlet, DisplayProductInfo, product.html



Enter Product Name:

Enter Product Price:

Enter Available Product Quantity:



Product Data entered...

Product Name IPHONE Price 70000.0Quantity 12

Shop Name Nova electronics

Mail feedback@nova.com



Copyright © Capgemini 2015. All Rights Reserved 11

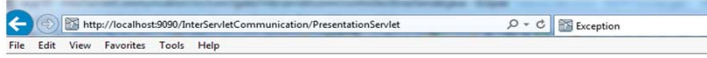

Note:

Refer to ProductServlet, DisplayProductInfo, product.html partial code below:

Execute the code by invoking the product.html first and fill in required fields. Data is fetched into ProductServlet and then forwarded to DisplayProductInfo servlet


Demo

- Working with include method of RequestDispatcher
- PresentationServlet, ItemServlet, ErrorServlet



Error Servlet

com.igate.interservletcomm.controller.PresentationServlet\$ItemNotFoundException: Item not found



From Presentation Servlet : Item set :- coffee:
From Item Servlet: Received : coffee
After ItemServlet

Capgemini
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 12

Note:

Refer to PresentationServlet, ItemServlet, ErrorServlet partial code below:

Execute the code by invoking the PresentationServlet with no item appended as query parameter.

Then execute PresentationServlet with item appended as query parameter

<http://localhost:9090/InterServletCommunication/PresentationServlet>

By appending item as query parameter

<http://localhost:9090/InterServletCommunication/PresentationServlet?item=coffee>

Here the response of ItemServlet is included in PresentationServlet

7.2: Server / Client side dispatch

Client side Dispatch

- In Client side dispatch, method used is `response.sendRedirect("URL Pattern");`
- With this method, a new request is generated and the request-response lifecycle is started again



Copyright © Capgemini 2015. All Rights Reserved 13

Client Side Dispatch

Make use of method, `response.sendRedirect("URL Pattern");`

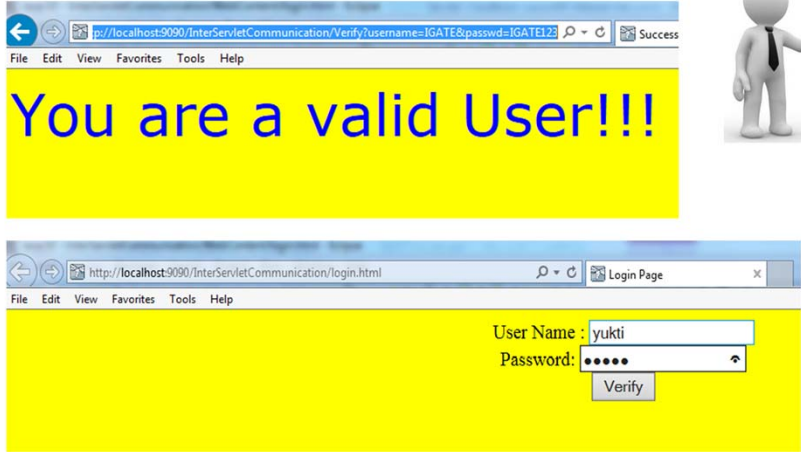
In this, a servlet may not be able to process the request completely and sends it back to the browser with HTTP **Response Code as 300** for **Redirection**.

This will result in a new URL to be created and start of another Request-Response lifecycle.

The URL pattern will change in browser as contrast to Server side dispatch

Demo

- Execute login.html, Verify.java, success.html



The image shows two browser screenshots. The top one displays a yellow background with the text 'You are a valid User!!!' in blue. The bottom one shows a login form with fields for 'User Name' (containing 'yukti') and 'Password' (masked with dots), and a 'Verify' button. A 3D character stands next to a 'Demo' sign.

Capgemini
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 14

Run the URL Pattern:

<http://localhost:9090/InterServletCommunication/login.html>

If user credentials are valid, display Success.html page

And if invalid user credentials display -> login.html back

7.2: Server / Client side dispatch

Comparison

Server – Side Dispatch	Client – Side Dispatch
Container is responsible for dispatch	Browser is responsible for redirect
RequestDispatcher interface used with forward and include methods, with URL Pattern specified	Response.sendRedirect("URL Pattern") method is used
URL Pattern in browser during Request-Response lifecycle does not change	URL Pattern in browser during Request-Response lifecycle changes
It is an cheap process	It is expensive process
Forward / Include can happen only within the same Web application	Redirection can happen in same web application as well as to another URL (outside web application) also
Data can be transferred easily between web components by making use of Request object API like request.setAttribute(String s,Object); and request.getAttribute(String s);	Data transferring here becomes tricky due to the new Request-Response lifecycle. However we could make use of ServletContext object for data sharing among web components

7.3: Communication and Sharing Data between Servlets

Sharing Data between Servlets

- Servlet Context API allows sharing named objects between all the servlets in a servlet context, by binding the objects to the servlet context object.
- Methods of Servlet Context API :
 - void ServletContext.setAttribute(String s, Object o)
 - Object ServletContext.getAttribute(String s)
 - void ServletContext.removeAttribute(String s)
 - Enumeration ServletContext.getAttributeNames()



Copyright © Capgemini 2015. All Rights Reserved 16

Sharing Data between Servlets:

One of the reasons for inter-servlet communication that we listed is to share data between the servlets. Servlet API offers way of sharing named objects between all the Servlets in a Servlet context (and also other contexts, as we'll see in next page) by binding the objects to the ServletContext object which is shared by several Servlets.

The **ServletContext** class has several methods for accessing the shared objects:

`public void setAttribute(String name, Object object)` adds a new object or replaces an old object by the specified name. The attribute name should follow the same naming convention as a package name.

Just like a custom ServletRequest attribute, an object which is stored as a ServletContext attribute should also be serializable to allow attributes to be shared by Servlets which are running in different JVMs on different machines in a load-balancing server environment.

`public Object getAttribute(String name)` returns the named object or null if the attribute does not exist.

7.3: Communication and Sharing Data between Servlets

Sharing Information: Usage

- Setting the context, request information:

```
ServletContext sc=getServletContext();
sc.setAttribute("company" , "IGATE");
req.setAttribute("coursename","J2EE");
sc.getRequestDispatcher("/print").forward(req , res);
```

- Getting this context, request information in the included servlet:

```
ServletContext sc=getServletContext();
out.println(" context variables (application) " + "<br>");
out.println("company " + " : " + sc.getAttribute("company") + "<br>");
out.println(" Request attributes " + "<br>");
out.println("name" + " : " + req.getAttribute("coursename"));
```



Copyright © Capgemini 2015. All Rights Reserved 17

Sharing Data between Servlets:

Sharing Information: Usage

In addition to the user-defined attributes, there may also be predefined attributes that are specific to the Servlet engine and provide additional information about a Servlet(Context)'s environment.

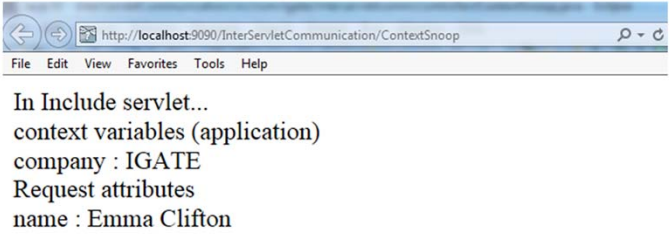

public Enumeration **getAttributeNames()** returns an Enumeration of the names of all available attributes.

public void **removeAttribute(String name)** removes the attribute with the specified name if it exists.


The separation of Servlets into Servlet contexts depends on the Servlet engine. The ServletContext object of a Servlet with a known **local URI** can be retrieved with the method public ServletContext getContext(String uripath) of the Servlet's own ServletContext. This method returns null if there is no Servlet for the specified path or if this Servlet is not allowed to get the ServletContext for the specified path due to security restrictions.

Demo

- Creating context attributes and retrieving them using [set/get]Attribute method
 - ContextSnoop.java
 - PrintServlet.java



In Include servlet...
context variables (application)
company : IGATE
Request attributes
name : Emma Clifton

 Capgemini
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 18

Note:

Refer to the ContextSnoop Servlet and Include servlet; partial code of both given below:

ContextSnoop.java

PrintServlet.java :

Execute the ContextSnoop servlet as :

<http://localhost:9090/InterServletCommunication/ContextSnoop>

Notice how the ContextSnoop servlet first sets the request-level and ServletContext-level attributes and then forwards to print servlet!

For the curious, invoke the Print servlet directly; and see what happens

Summary

- In this lesson, we have learnt:
 - Inter-Servlet Communication
 - RequestDispatcher Interface
 - Communication with Server Resources
 - Sharing data between servlets



Add the notes here.

Review Question

- Question 1: Which of the following can be used to obtain RequestDispatcher object?

- Option 1: ServletConfig.getRequestDispatcher()
- Option 2: ServletRequest.getRequestDispatcher()
- Option 3: ServletResponse.getRequestDispatcher()
- Option 4: ServletContext.getRequestDispatcher()



- Question 2: What is the difference between include() and forward() of RequestDispatcher ?

- Option 1: Forward() transfers a request from a servlet to another resource on the server while include() embeds the content of a resource in the response
- Option 2: Forward() embeds the content of a resource in the response while include() transfers a request from a servlet to another resource on the server

Add the notes here.

Review Question

- Question 3: How is the data shared between servlets ?
 - Option 1: Using put/get value methods of Servlet Context
 - Option 2: Using set/get Attribute methods of ServletContext
 - Option 3: Using value bound/unbound methods
 - Option 4: Using set/get attribute methods of ServletResponse



Add the notes here.