1. Introduction

Contents

[1. Introduction 1](#_Toc1727)

[2. Architecture of Servlets 1](#_Toc31713)

[3. Servlets Life Cycle 1](#_Toc25352)

[4. Setting up Tomcat Web Server 1](#_Toc9244)

[5. Summary 2](#_Toc24245)

# 1. Introduction

=>slides: Pg. 1-2

. In this course, we shall learn programming Servlets with the support of some modules. This course is for the Java people who want to start web application development using Servlets. We shall start from scratch, understanding the architecture and lifecycle of Servlets and slowly by the end of this course, the learner will get the complete idea on programming Servlets for developing web applications.

=>slides: Pg. 3

This course assumes that the learner has a prior background on Core Java fundamentals, and a knowledge on JDBC and HTML will be an added advantage, but it is not compulsory. T

=>slides: Pg. 4

he first module can be considered as a prerequisite session for understanding all the remaining modules of programming Servlets. In this module, we shall understand how web technology and CGI work, what are Servlets, architecture of Servlets, Servlet lifecycle, and also we will learn how to set up the Tomcat web server for supporting for the Servlets. Now let us first understand how the web technology works.

=>slides: Pg. 5

In the web, the user clicks on a link or types in the URL on the web browser, the web browser formats that request and sends it to the server. Here, a browser is a software that knows how to communicate with the server, for example, Chrome, IE, Safari, etc. Server accepts the request and finds the requested page. And then the server formats the response and sends the response back to the client, that is, the browser. Once the web browser gets the response, it renders the content into a display for the user. Whenever we say server, then we mean either the physical machine hardware or the web server application software. Throughout this course if the differences between server hardware and software matters, then I will explicitly say which one--hardware or software--I'm talking about. From now on, whenever we use the term client, it usually won't care whether we are talking about the human user or the browser app. Or in simple terms to remember that client is the browser app.

=>slides: Pg. 6

Now let us understand why we require Servlets. Web servers load to static web pages whenever the client sends a request to the web server, the server accepts the request, finds the resource,

=>slides: Pg. 7

and returns the relevant resource to the client. Sometimes that resource is an HTML page. Sometimes it might be a picture or a song file or even a PDF document. Static pages just sit there indirectly. The server finds it and hands it back to the client. The processing of the page happens at the client, that is, web browser. Every client will see the same information, but sometimes we need more than that.

=>slides: Pg. 8

We need to perform some kind of computations at the server site like interacting with the database or platforms or processing some kind of business logic. But the web server application can't do those computations.

=>slides: Pg. 9

It requires another helper application on the server, which can handle this task.

=>slides: Pg. 10

The web server will take care of getting the request to the right helper app. It will also take that application's response and send it back to the client.

=>slides: Pg. 11

The first technology which was used for building the helper application is CGI,

=>slides: Pg. 12

which stands for Common Gateway Interface. CGI programs can be written using PERL scripts, C and Python.

=>slides: Pg. 13

Now let us understand how a CGI program works. Whenever the user clicks on a link that has a URL to a CGI program, instead of a static page, the web server application identifies that the request is for a helper program so the web server launches and runs the program. The web server app also sends the parameters from a GET or POST request. Once the helper app receives the request, it constructs a brand-new page and sends the HTML back to the server. As far as the web server is concerned, the HTML from the helper app is considered as a static page.

=>slides: Pg. 14

Once the server receives the HTML content, the helper application will shut down, and the client receives an HTML page as a part of its dynamic content.

=>slides: Pg. 15

But whenever we use the CGI programs, performance-related issues will arise because the server has to launch a heavyweight process for each and every request for the resource. Further, there's no possibility for a CGI program to keep database connections open over a number of requests. For each request a new database connection must be established, which leads to a significant performance cost in database-centric applications. So, a microsystem answer to the CGI technology limitations are Java Servlets. In the next section, first we shall understand what Servlet is and also we shall understand the architecture of Servlets in detail.

# Architecture of Servlets

=>slides: Pg. 16

Before we start understanding the architecture of Servlets, we need to understand what Servlets are.

=>slides: Pg. 17

A Servlet is a java programming language class that is used to extend the capabilities of servers that host applications accessed by means of a request-response programming model. Although Servlets can respond to any type of request, they are commonly used to extend applications hosted by web servers. Java Servlets often serve the same purpose as programs implemented using the CGI.

=>slides: Pg. 18

But Servlets offer several advantages in comparison with the CGI. Servlets execute within the address space of a web server. It is not necessary to create a separate process to handle each client request, so performance is significantly better. Servlets are platform independent because they are written in Java. Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. So Servlets are trusted. The full functionality of the Java class libraries is available to a Servlet. It can communicate with applets, databases, or other software.

=>slides: Pg. 19

Servlets don't have a main method for their execution. They request some help. When a request comes in, someone has to instantiate the Servlet. Someone has to get the request and the response to the Servlet. Someone has to manage the life, death, and resources of the Servlet. And that someone is the web container, which is also popularly called as Servlet container or Servlet engine. The point we need to remember is since the Servlets don't have a main method, they are under the control of another Java application called as a web container. Servlets are supported by different web containers such as Tomcat, Jetty, Java Web Server, and Resin, etc.

=>slides: Pg. 20

Now let us understand the high level architecture of Servlets. Whenever a client types in a URL at the browser and sends an HTTP request for the Servlet instead of a static page, the web server application like Tomcat, the web container app, that is running at the web server machine will get the request, and the container sees that the request is for a Servlet, so the container creates two objects--HttpServletRequest and HttpServletResponse. Then the container finds the correct Servlet based on the URL in the request and creates or allocates a thread for that request

=>slides: Pg. 21

and then passes the request and response objects to the Servlet thread.

=>slides: Pg. 22

The container calls the Servlet's service method depending on the type of request. The service method calls either the doGet or doPost method. For this scenario, as I mentioned, the client types in the URL, the request will be an HTTP-GET. I will discuss in detail about this method by explaining the Servlet's lifecycle.

=>slides: Pg. 23

The doGet method generates the dynamic page and stuffs the page into the response object. We need to remember that the container still has a reference to the response object. The thread completes, and the container converts the response object into an HTTP response

=>slides: Pg. 24

and sends it back to the client.

=>slides: Pg. 25

It then deletes the request and response objects. In the next section, we shall understand the Servlet class hierarchy and Servlet lifecycle in detail, which plays a major role in understanding the Servlet programming.

# Servlets Life Cycle

=>slides: Pg. 26

In the previous section, I have used some terms like service, doGet, doPost while explaining the Servlet's architecture. Now let us first understand these terms while understanding the class hierarchy of Servlets, which will also help in understanding the Servlet's lifecycle.

=>slides: Pg. 27

The center abstraction in the Servlet API is the Servlet interface. All Servlets have to implement this interface either directly or by extending a class such as GenericServlet or HttpServlet. The Servlet interface provides the following methods to manage the Servlet and its communications with clients. Destroy--this method cleans up whatever resources are being held and makes sure that any persistent state is synchronized with the Servlet's current in-memory state. getServletConfig--this method returns a Servlet config object which contains any initialization parameters and startup configuration for the Servlet. getServletInfo--this method returns a string containing information about the Servlet such as its author, version, and copyright. Init--this method is used to initialize the Servlet, and this method will execute once before any request is serviced, and this method throws Servlet exceptions. Service--this method carries out a single request from the client and throws Servlet exceptions and IO exceptions. Servlet writers provide some or all of these methods when developing the Servlet.

=>slides: Pg. 28

GenericServlet is an abstract class that implements Servlet interface, and this class is protocol independent. That is, it handles all types of protocols, HTTP, SMTP, FTP, etc., and uses service method to handle any type of request, whereas HttpServlet class is also an abstract class, which is a direct subclass of GenericServlet, and it is protocol dependent. It handles only HTTP protocol and supports doGet, doPost, doPut, doDelete based on the requested type. We shall understand these classes and methods in detail at a later part of this course while discussing working with form and query string data.

=>slides: Pg. 29

Now as we have an idea on the class hierarchy of Servlets, let us understand the Servlets lifecycle. A Servlet lifecycle can be defined as the entire process from its creation until the destruction, and will be controlled by the web container. The lifecycle contains the following steps--loading of Servlet class, creating the Servlet instance, calling the init method, calling the service method, calling the destroy method. The first three steps are executed only once when the Servlet is initially loaded. By default, the Servlet is not loaded until the first request is received for it. We can force the container to load the Servlet when it starts up. Service method will execute multiple attempts, once for every HTTP request to the Servlet. Destroy method is executed whenever the container unloads the Servlet. Now let us understand the lifecycle in detail. Loading the Servlet class--before a Servlet can be invoked, the container must first load the particular class definition. This happens just like any other class definition getting loaded. Creating the Servlet instance--when the Servlet class is loaded, the Servlet container creates an instance of the Servlet. Remember, only a single instance of the Servlet is created. After that, the same identical requests are executed on the same Servlet instance with a \_\_\_\_\_. Again, this is up to the Servlet container to determine. But generally there is just one instance. Calling the init method--when a Servlet instance is created, the container invokes this init method. The init method gives us the privilege to configure a particular Servlet before the very first request is processed. We can specify init parameters to the Servlet in the web. xml file or via annotations. I will explain the importance of web. xml and annotations in a later part of this course. Calling the service method--for every request received to the Servlet, the container calls Servlet's service method. Our Servlet extends HttpServlet class or GenericServlet. To process the request, we must have to override one of the doGet or doPost or service methods because these methods are typically called by the container internally. As long as the Servlet is active in the container, the service method can be called. Calling the destroy method--when a Servlet is unloaded by the Servlet container, the destroy method gets a call. This happens only once because a Servlet is unloaded once in its lifecycle. With all these details, now we are ready to get started. In the next section, we shall understand how to set up the Tomcat web server, which will be our web container for supporting Servlet programming.

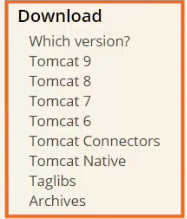
# Setting up Tomcat Web Server

=>slides: Pg. 30

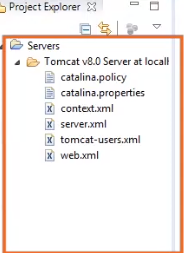
As we have a clear idea on how Servlets work and also the importance of web container or Servlet container, now let us understand how to set up the Tomcat web server for supporting the Servlet's programming. First, we need to download the Tomcat Apache web server. I already opened the web browser.



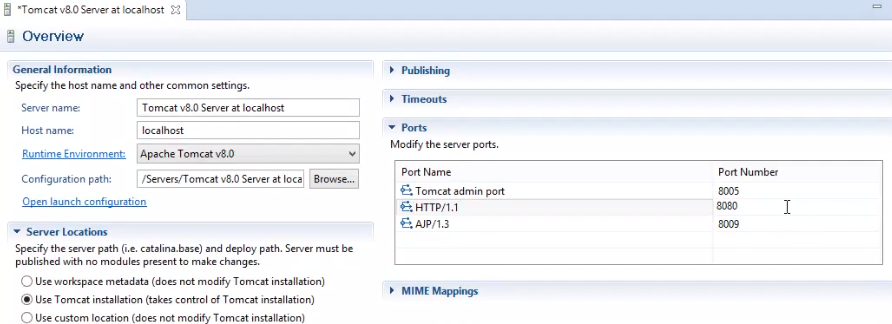
And, also, I typed in the URL from where we can download the software.



Under the download menu, we can find various versions of Tomcat Apache web server. Currently as of this recording, version 9 is in alpha, and version 8. 5 is in beta. So we can download either 8 or 7 versions. If we click on Tomcat 8, under the Binary Distribution code, we can find various links to download. Since I'm using Windows, let me click on that zip to download. Once the file gets downloaded, we need to extract the downloaded zip file. So let me extract the file. It is better to extract in some folder where it is easy to refer. I extract in the C: drive root. Once the file gets extracted, then open Esectionse, click on the Servers tab. If the Servers tab is not visible, then click on the Window menu, click on Show View, and then click on Servers. We can observe a message: No servers are available. So let me click on the link to create a server. It will provide an option for selecting the server type in the list. Let me expand the Apache and select the Apache Tomcat version downloaded. As I have downloaded 8, let me select Apache Tomcat version 8 and click on Next. Now, we need to specify the Tomcat installation folder. So let me click on the Browse button and navigate to the folder where we have extracted the Tomcat Apache web server and select the folder above bin directory. In this case, apache-tomcat, and the version. Click on OK button, then click on Next button. If any projects are already available, it will prompt us to configure the project. But as we don't have any project, we can just click on Finish button. Now we can observe there is a server which has been displayed, and also we can observe there is a new package created with the name Servers under Project Explorer.



When we expand the Servers folder at Project Explorer, we can find all the XML files used by the Tomcat web server. Let us verify if the Tomcat web server is configured or not. To do, let me right-click on Tomcat server at localhost and click on Start to start the server. Sometimes, it might raise an error stating Port 8080 is already in use. The reason for that is many web servers default the port number will be 8080. Since it is already used by another server configured in my system, when I start Tomcat server, this error has been raised. Now let me first fix that before we continue.



Double-click on the Tomcat server at localhost, and then we can observe the default HTTP/1. 1 port is 8080. Let me change the port number to 8081 and then save the configuration by pressing Ctrl+S. And before saving the configuration, make sure that server location is set to Use Tomcat installation radio button. Let us observe what will happen if I have the default workspace metadata as the server location. Now once again, let me right-click on Tomcat web server and click on Start. In the Console tab, we can observe the log messages generated by the server. And we can observe a message stating Server startup. This indicates that Tomcat web server has been started. To verify let me open a web browser, and let me type in http://localhost:8081 and press Enter. We can observer there is a 404 error. So once again, let me flip to Esectionse. Within the server properties, let me update the server location to Use Tomcat installation, and let me save. Once again, let me restart the server. And now let me flip to the browser again and refresh the page. We can observe the Tomcat web server contents are displayed. This indicates that our Esectionse is configured with the Tomcat web server.

# Summary

=>slides: Pg. 31

In this module, we have understood clearly how a web server processes the request and provides the response. And, also, we have understood the architecture of Servlets, lifecycle of Servlets, and also we have learned how to configure Tomcat server for processing Servlets. In the next module, we shall understand how to create a basic Servlet page and apply the Servlet to the Tomcat web server.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*