# 2.1 Introduction:

The servlet technology is the foundation of web application development using the Java programming language. It is one of the most important Java technologies, and it is the underlying technology for another popular Java technology for web application development: JavaServer Pages (JSP). Therefore, understanding the servlet technology and its architecture is important if you want to be a servlet developer.Even if you plan to develop your Java web application using JSP pages alone,understanding the servlet technology helps you build a more efficient and effective JSP application.

# 2.2 Servlet Application Architecture

A servlet is a Java class that can be loaded dynamically into and run by a special web server. This servlet-aware web server is called a servlet container, which also was called a servlet engine in the early days of the servlet technology .Servlets interact with clients via a request-response model based on HTTP. Because servlet technology works on top of HTTP, a servlet container must support HTTP as the protocol for client requests and server responses. However, a servlet container also can support similar protocols, such as HTTPS (HTTP over SSL) for secure transactions.





Fig 2.1 Servlet Application Architecture

# 2.3 How a Servlet Works:

A servlet is loaded by the servlet container the first time the servlet is requested. The servlet then is forwarded the user request, processes it, and returns the response to the servlet container, which in turn sends the response back to the user. After that, the servlet stays in memory waiting for other requests—it will not be unloaded from the memory unless the servlet container sees a shortage of memory. Each time the servlet is requested, however, the servlet container compares the timestamp of the loaded servlet with the servlet class file. If the class file timestamp is more recent, the servlet is reloaded into memory. This way, you don't need to restart the servlet container every time you update your servlet.



# Fig 2.2 How a Servlet Works

# 2.4 A Servlet's Life Cycle:

Let there be servlet. This interface in the javax.servlet package is the source of all activities in servlet programming. Servlet is the central abstraction of the Java servlet technology. Every servlet you write must implement this javax.servlet.Servlet interface, either directly or indirectly. The life cycle of a servlet is determined by three of its methods: init, service, and destroy.

#### The init( ) Method

The init method is called by the servlet container after the servlet class has been instantiated. The servlet container calls this method exactly once to indicate to the servlet that the servlet is being placed into service. The init method must complete successfully before the servlet can receive any requests.

You can override this method to write initialization code that needs to run only once, such as loading a database driver, initializing values, and so on. In other cases, you normally leave this method blank.

The signature of this method is as follows:

public void init(ServletConfig config) throws ServletException

#### The service( ) Method

The service method is called by the servlet container after the servlet's init method to allow the servlet to respond to a request.

Servlets typically run inside multithreaded servlet containers that can handle multiple requests concurrently. Therefore, you must be aware to synchronize access to any shared resources, such as files, network connections, and the servlet's class and instance variables. For example, if you open a file and write to that file from a servlet, you need to remember that a different thread of the same servlet also can open the same file" for more on the topic of multithreading and synchronization.

This method has the following signature:

public void service(ServletRequest request, ServletResponse response)

throws ServletException, java.io.IOException

The servlet container passes a ServletRequest object and the ServletResponse object. The ServletRequest object contains the client's request and the ServletResponse contains the servlet's response. These two objects are important because they enable you to write custom code that determines how the servlet services the client request.

The service method throws a ServletException if an exception occurs that interferes with the servlet's normal operation. The service method also can throw a java.io.IOException if an input or output exception occurs during the execution of this method. As the name implies, the service method exists so that you can write code that makes the servlet function the way it is supposed to.

#### The destroy( ) Method

The servlet container calls the destroy method before removing a servlet instance from service. This normally happens when the servlet container is shut down or the servlet container needs some free memory.

This method is called only after all threads within the servlet's service method have exited or after a timeout period has passed. After the servlet container calls this method, it will not call the service method again on this servlet.

The destroy method gives the servlet an opportunity to clean up any resources that are being held (for example, memory, file handles, and threads) and make sure that any persistent state is synchronized with the servlet's current state in memory.

The signature of this method is as follows:

public void destroy()

# 2.5 Demonstrating the Life Cycle of a Servlet:

contains the code for a servlet named PrimitiveServlet, a very simple servlet that exists to demonstrate the life cycle of a servlet. The PrimitiveServlet class implements javax.servlet.Servlet (as all servlets must) and provides implementations for all the five methods of servlet. What it does is very simple. Each time any of the init, service, or destroy methods is called, the servlet writes the method's name to the console.

After you compile the source code into the myApp\WEB-INF\classes directory, add the servlet to the web.xml under the name Primitive

##### The web.xml File for PrimitiveServlet

<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app

PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"

"http://java.sun.com/dtd/web-app\_2\_3.dtd">

<web-app>

<servlet>

<servlet-name>Primitive</servlet-name>

<servlet-class>PrimitiveServlet</servlet-class>

</servlet>

</web-app>

# 2.6 Requests and Responses:

Requests and responses are what a web application is all about. In a servlet application, a user using a web browser sends a request to the servlet container, and the servlet container passes the request to the servlet.

In a servlet paradigm, the user request is represented by the ServletRequest object passed by the servlet container as the first argument to the service method. The service method's second argument is a ServletResponse object, which represents the response to the user.

# 2.7 The ServletRequest Interface

The ServletRequest interface defines an object used to encapsulate information about the user's request, including parameter name/value pairs, attributes, and an input stream.

The ServletRequest interface provides important methods that enable you to access information about the user. For example, the getParameterNames method returns an Enumeration containing the parameter names for the current request. To get the value of each parameter, the ServletRequest interface provides the getParameter method.

The getRemoteAddress and getRemoteHost methods are two methods that you can use to retrieve the user's computer identity. The first returns a string representing the IP address of the computer the client is using, and the second method returns a string representing the qualified host name of the computer.



# Fig 2.3 ServletRequest Interface

# 2.8 The ServletResponse Interface

The ServletResponse interface represents the response to the user. The most important method of this interface is getWriter, from which you can obtain a java.io.PrintWriter object that you can use to write HTML tags and other text to the user.



# Fig 2.4 ServletResponse Interface

# 2.9 The GenericServlet Wrapper Class:

Throughout this chapter, we have been creating servlet classes that implement the javax.servlet.Servlet interface. Everything works fine, but there are two annoying things that you've probably noticed:

1. You have to provide implementations for all five methods of the Servlet interface, even though most of the time you only need one. This makes your code look unnecessarily complicated.
2. The ServletConfig object is passed to the init method. You need to preserve this object to use it from other methods. This is not difficult, but it means extra work.

The javax.servlet package provides a wrapper class called GenericServlet that implements two important interfaces from the javax.servlet package: Servlet and ServletConfig, as well as the java.io.Serializable interface. The GenericServlet class provides implementations for all methods, most of which are blank. You can extend GenericServlet and override only methods that you need to use. Clearly, this looks like a better solution.

# 2.10 JSP Basics:

JavaServer Pages (JSP) is another Java technology for developing web applications. JSP was released during the time servlet technology had gained popularity as one of the best web technologies available. JSP is not meant to replace servlets, however. In fact, JSP is an extension of the servlet technology, and it is common practice to use both servlets and JSP pages in the same web applications.

Authoring JSP pages is so easy that you can write JSP applications without much knowledge of the underlying API. If you want to be a really good Java web programmer, however, you need to know both JSP and servlets. Even if you use only JSP pages in your Java web applications, understanding servlets is still very important. For example, in JSP you work with HTTP requests and HTTP responses, request parameters, request attributes, session management, cookies, URL-rewriting, and so on. This chapter explains the relation between JSP and servlets, introduces the JSP technology, and presents many examples that you can run easily.

# 2.11 What's Wrong with Servlets?

The history of web server-side programming in Java started with servlets. Sun introduced servlets in 1996 as small Java-based applications for adding dynamic content to web applications. Not much later, with the increasing popularity of Java, servlets took off to become one of the most popular technologies for Internet development today.

##### Displays All Parameter/Value Pairs in a Request Using a Servlet

Nearly half of the content sent from the doPost method is static HTML. However, each HTML tag must be embedded in a String and sent using the println method of the PrintWriter object. It is a tedious chore. Worse still, the HTML page may be much longer.

Another disadvantage of using servlets is that every single change will require the intervention of the servlet programmer. Even a slight graphical modification, such as changing the value of the <BODY> tag's BGCOLOR attribute from #DADADA to #FFFFFF, will need to be done by the programmer (who in this case will work under the supervision of the more graphic-savvy web designer).

Sun understood this problem and soon developed a solution. The result was JSP technology. According to Sun's web site, "JSP technology is an extension of the servlet technology created to support authoring of HTML and XML pages." Combining fixed or static template data with dynamic content is easier with JSP .What needs to be highlighted is that "JSP technology is an extension of the servlet technology." This means that JSP did not replace servlets as the technology for writing server-side Internet/intranet applications. In fact, JSP was built on the servlet foundation and needs the servlet technology to work.

JSP solves drawbacks in the servlet technology by allowing the programmer to intersperse code with static content, for example. If the programmer has to work with an HTML page template written by a web designer, the programmer can simply add code into the HTML page and save it as a .jsp file. If at a later stage the web designer needs to change the HTML body background color, he or she can do it without wasting the charging-by-the-hour programmer's time. He or she can just open the .jsp file and edit it accordingly.

You can see that <HTML> tags stay as they are. When you need to add dynamic content, all you need to do is enclose your code in <% … %> tags.

Again, JSP is not a replacement for servlets. Rather, JSP technology and servlets together provide an attractive solution to web scripting/programming by offering platform independence, enhanced performance, separation of logic from display, ease of administration, extensibility into the enterprise, and most importantly, ease of use.

# 2.12 Writing a JSP File:

A JSP page consists of interwoven HTML tags and Java code. The HTML tags represent the presentation part and the code produces the contents. In its most basic form, a JSP page can include only the HTML part.



Fig 2.5 jsp file interface

# 2.13 How JSP Works:

Inside the JSP container is a special servlet called the page compiler. The servlet container is configured to forward to this page compiler all HTTP requests with URLs that match the .jsp file extension. This page compiler turns a servlet container into a JSP container. When a .jsp page is first called, the page compiler parses and compiles the .jsp page into a servlet class. If the compilation is successful, the jsp servlet class is loaded into memory. On subsequent calls, the servlet class for that .jsp page is already in memory; however, it could have been updated. Therefore, the page compiler servlet will always compare the timestamp of the jsp servlet with the jsp page. If the .jsp page is more current, recompilation is necessary. With this process, once deployed, JSP pages only go through the time-consuming compilation process once.

You may be thinking that after the deployment, the first user requests for a .jsp page will experience unusually slow response due to the time spent for compiling the .jsp file into a jsp servlet. To avoid this unpleasant situation, a mechanism in JSP allows the .jsp pages to be pre-compiled before any user request for them is received. Alternatively, you deploy your JSP application as a web archive file in the form of a compiled servlet.

## 2.13.1 The JSP Servlet Generated Code:

When the JSP is invoked, Tomcat creates two files in the C:\%CATALINA\_HOME%\work\localhost\examples\jsp directory. Those two files are SimplePage\_ jsp.java and SimplePage\_ jsp.class. When you open the SimplePage\_ jsp.java file, you will see the following:

# 2.14 The JSP API

The JSP technology is based on the JSP API that consists of two packages: javax.servlet.jsp and javax.servlet.jsp.tagext. Both packages are given in detail in In addition to these two packages, JSP also needs the two servlet packages—javax.servlet and javax.servlet.http. When you study the javax.servlet.jsp package, you will know why we say that JSP is an extension of servlet technology and understand why it is important that a JSP application programmer understands the servlet technology well.The javax.servlet.jsp package has two interfaces and four classes. The interfaces are as follows:

* JspPage
* HttpJspPage

The four classes are as follows:

* JspEngineInfo
* JspFactory
* JspWriter
* PageContext

In addition, there are also two exception classes: JspException and JspError.

## 2.14.1The JspPage Interface

The JspPage is the interface that must be implemented by all JSP servlet classes. This may remind you of the javax.servlet.Servlet interface , "The Servlet Technology," of course. And, not surprisingly, the JspPage interface does extend the javax.servlet.Servlet interface.

The JSPPage interface has two methods, JspInit and JspDestroy, whose signatures are as follows:

public void jspInit()

public void jspDestroy()

jspInit, which is similar to the init method in the javax.servlet.Servlet interface, is called when the JspPage object is created and can be used to run some initialization. This method is called only once during the life cycle of the JSP page: the first time the JSP page is invoked.

The jspDestroy method is analogous with the destroy method of the javax.servlet.Servlet interface. This method is called before the JSP servlet object is destroyed. You can use this method to do some clean-up, if you want.

Most of the time, however, JSP authors rarely make full use of these two methods. The following example illustrates how you can implement these two methods in your JSP page:

## 2.14.2 The HttpJspPage Interface

This interface directly extends the JspPage interface. There is only one method: \_ jspService. This method is called by the JSP container to generate the content of the JSP page. The \_ jspService has the following signature:

public void \_jspService(HttpServletRequest request,

HttpServletResponse response) throws ServletException, IOException.

You can't include this method in a JSP page, such as in the following code:

## 2.14.3The JspFactory Class

The JspFactory class is an abstract class that provides methods for obtaining other objects needed for the JSP page processing. The class has the static method getDefaultFactory that returns a JspFactory object. From the JspFactory object, a PageContext and a JspEngineInfo object can be obtained that are useful for the JSP page processing. These objects are obtained using the JspFactory class's getEngineInfo method and the getPageContext method, whose signatures are given here:

public abstract JspEngineInfo getEngineInfo()

public abstract PageContext getPageContext (

Servlet requestingServlet, ServletRequest request,

ServletResponse response, String errorPageURL,

boolean needsSession, int buffer, boolean autoFlush)

The following code is part of the \_ jspService method that is generated by the JSP container:

JspFactory \_jspxFactory = null;

PageContext pageContext = null;

jspxFactory = JspFactory.getDefaultFactory();

pageContext = \_jspxFactory.getPageContext(this, request,

response, "", true, 8192, true);

## 2.14.4 The JspEngineInfo Class

The JspEngineInfo class is an abstract class that provides information on the JSP container. Only one method, getSpecificationVersion, returns the JSP container's version number. Because this is the only method currently available, this class does not have much use.

You can obtain a JspEngineInfo object using the getEngineInfo method of the JspFactory class.

## 2.14.5 The PageContext Class

PageContext represents a class that provides methods that are implementation-dependent. The PageContext class itself is abstract, so in the \_ jspService method of a JSP servlet class, a PageContext object is obtained by calling the getPageContext method of the JspFactory class.

The PageContext class provides methods that are used to create other objects. For example, its getOut method returns a JspWriter object that is used to send strings to the web browser. Other methods that return servlet-related objects include the following:

* getRequest, returns a ServletRequest object
* getResponse, returns a ServletResponse object
* getServletConfig, returns a ServletConfig object
* getServletContext, returns a ServletContext object
* getSession, returns an HttpSession object

## 2.14.6 The JspWriter Class

The JspWriter class is derived from the java.io.Writer class and represents a Writer that you can use to write to the client browser. Of its many methods, the most important are the print and println methods. Both provide enough overloads that ensure you can write any type of data. The difference between print and println is that println always adds the new line character to the printed data.

Additional methods allow you to manipulate the buffer. For instance, the clear method clears the buffer. It throws an exception if some of the buffer's content has already been flushed. Similar to clear is the clearBuffer method, which clears the buffer but never throws any exception if any of the buffer's contents have been flushed.

# 2.15 JSP Syntax:

"JSP Basics," you learned that a JSP page can have Java code and HTML tags. More formally, you can say that a JSP page has elements and template data. The elements, which also are called JSP tags, make up the syntax and semantics of JSP. Template data is everything else. Template data includes parts that the JSP container does not understand, such as HTML tags.

There are three types of elements:

* Directive elements
* Scripting elements
* Action elements

To write an effective JSP page, you need to understand all these elements well. Elements have two forms: the XML form and the <% … %> alternative form. Template data remains as it is, normally passed through the client uninterrupted.

This chapter discusses the three types of JSP elements and comments. It also presents examples on how to use these elements. You will also learn how incorporating these elements affects the JSP servlets—servlets that result from the translation of JSP pages.

## 2.15.1 Directives

Directives are messages to the JSP container containing information on how the JSP container must translate a JSP page into a corresponding servlet. Directives have the following syntax:

<%@ directive (attribute="value")\* %>

The asterisk (\*) means that what is enclosed in the brackets can be repeated zero or more times. The syntax can be re-written in a more informal way as follows:

<%@ directive attribute1="value1" attribute2="value2" ... %>

White spaces after the opening <%@ and before the closing %> are optional, but are recommended to enhance readability.

## 2.15.2 The Page Directive

The Page directive has the following syntax:

<%@ page (attribute="value")\* %>

Or, if you want to use the more informal syntax:

<%@ page attribute1="value1" attribute2="value2" ... %>

An example of the use of the Page directive is as follows:

<%@ page buffer="16384" session="false" %>

With JSP, you can specify multiple page directives in your JSP page, such as the following:

<%@ page buffer="16384" %>

<%@ page session="false" %>

##### The extends Attribute

The extends attribute defines the parent class that will be inherited by the generated servlet. You should use this attribute with extra care. In most cases, you should not use this attribute at all. In Tomcat, the parent class that will be subclassed by the resulting servlet is HttpJspBase.

##### The import Attribute

The import attribute is similar to the import keyword in a Java class or interface. The attribute is used to import a class or an interface or all members of a package. You will definitely use this attribute often. Whatever you specify in the import attribute of a page directive will be translated into an import statement in the generated servlet class. By default, Tomcat specifies the following import statements in every generated servlet class. You don't need to import what has been imported by default:

import javax.servlet.\*;

import javax.servlet.http.\*;

import javax.servlet.jsp.\*;

import javax.servlet.jsp.tagext.\*;

import org.apache.jasper.runtime.\*;

As an example, consider the following JSP page that imports the java.io package and the java.util.Enumeration interface:

<%@ page import="java.io.\*" %>

<%@ page import="java.util.Enumeration" %>

The two will be added before the default import statements in the generated servlet class, as described in the following code fragment:

import java.io.\*;

import java.util.Enumeration;

import javax.servlet.\*;

import javax.servlet.http.\*;

import javax.servlet.jsp.\*;

import javax.servlet.jsp.tagext.\*;

import org.apache.jasper.runtime.\*;