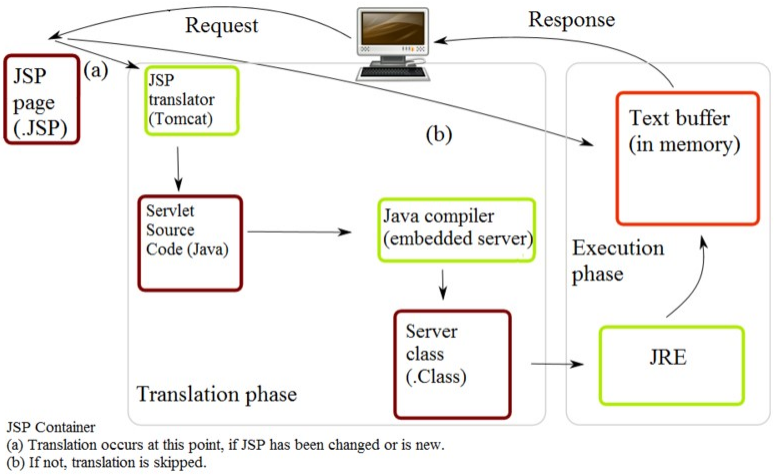
Java servlet

A **Java servlet** is a Java program that extends the capabilities of a server. Although servlets can respond to any types of requests, they most commonly implement applications hosted on Web servers.[1] Such Web servlets are the Java counterpart to other dynamic Web content technologies such as PHP and ASP.NET.



Life of a JSP file

Contents

Introduction

History

Compared with other web application models

Life cycle of a servlet

Example

Container servers

References

External links

Introduction

A Java servlet processes or stores a Java class in Java EE that conforms to the Java Servlet API,[2] a standard for implementing Java classes that respond to requests. Servlets could in principle communicate over any client–server protocol, but they are most often used with the HTTP protocol. Thus "servlet" is often used as shorthand for "HTTP servlet".[3] Thus, a software developer may use a servlet to add dynamic content to a web server using the Java platform. The generated content is commonly HTML, but may be other data such as XML. Servlets can maintain state in session variables across many server transactions by using HTTP cookies, or URL rewriting.

To deploy and run a servlet, a web container must be used. A web container (also known as a servlet container) is essentially the component of a web server that interacts with the servlets. The web container is responsible for managing the lifecycle of servlets, mapping a URL to a particular servlet and ensuring that the URL requester has the correct access rights.

The Servlet API, contained in the Java package hierarchy `javax.servlet` (https://docs.oracle.com/javaee/7/api/javax/servlet/package-summary.html), defines the expected interactions of the web container and a servlet.[3]

A `Servlet` (https://docs.oracle.com/javaee/7/api/javax/servlet/Servlet.html) is an object that receives a request and generates a response based on that request. The basic Servlet package defines Java objects to represent servlet requests and responses, as well as objects to reflect the servlet's configuration parameters and execution environment. The package `javax.servlet.http` (https://docs.oracle.com/javaee/7/api/javax/servlet/http/package-summary.html) defines HTTP-specific subclasses of the generic servlet elements, including session management objects that track multiple requests and responses between the web server and a client. Servlets may be packaged in a WAR file as a web application.

Servlets can be generated automatically from JavaServer Pages (JSP) by the JavaServer Pages compiler. The difference between servlets and JSP is that servlets typically embed HTML inside Java code, while JSPs embed Java code in HTML. While the direct usage of servlets to generate HTML (as shown in the example below) has become rare, the higher level MVC web framework in Java EE (JSF) still explicitly uses the servlet technology for the low level request/response handling via the `FacesServlet` (https://docs.oracle.com/javaee/7/api/javax/faces/webapp/FacesServlet.html). A somewhat older usage is to use servlets in conjunction with JSPs in a pattern called "Model 2", which is a flavor of the model–view–controller.

The current version of Servlet is 3.1.

History

The Servlet1 specification was created by Pavni Diwanji[4] while she worked at Sun Microsystems, with version 1.0 finalized in June 1997. Starting with version 2.2, the specification was developed under the Java Community Process. As of June 9, 2015, the current version of the Servlet specification is 3.1.

In his blog on java.net, Sun veteran and GlassFish lead Jim Driscoll details the history of servlet technology.[5] James Gosling first thought of servlets in the early days of Java, but the concept did not become a product until Sun shipped the *Java Web Server* product. This was before what is now the Java Platform, Enterprise Edition was made into a specification.

Servlet API history

Servlet API version	Released	JSR Number	Platform	Important Changes
Servlet 4.0	Sep 2017 (https://jcp.org/en/jsr/detail?id=369)	369	Java EE 8	HTTP/2
Servlet 3.1	May 2013 (https://jcp.org/en/jsr/detail?id=340)	340	Java EE 7	Non-blocking I/O, HTTP protocol upgrade mechanism (WebSocket)[6]
Servlet 3.0	December 2009 (http://www.javaworld.com/javaworld/jw-02-2009/jw-02-servlet3.html)	315	Java EE 6, Java SE 6	Pluggability, Ease of development, Async Servlet, Security, File Uploading
Servlet 2.5	September 2005 (http://www.javaworld.com/javaworld/jw-01-2006/jw-0102-servlet.html)	154	Java EE 5, Java SE 5	Requires Java SE 5, supports annotation
Servlet 2.4	November 2003 (http://www.javaworld.com/jw-03-2003/jw-0328-servlet.html)	154	J2EE 1.4, J2SE 1.3	web.xml uses XML Schema
Servlet 2.3	August 2001 (http://www.javaworld.com/jw-01-2001/jw-0126-servletapi.html)	53	J2EE 1.3, J2SE 1.2	Addition of `Filter`
Servlet 2.2	August 1999 (http://www.javaworld.com/jw-10-1999/jw-10-servletapi.html)	902, 903	J2EE 1.2, J2SE 1.2	Becomes part of J2EE, introduced independent web applications in .war files
Servlet 2.1	November 1998 (http://www.javaworld.com/jw-12-1998/jw-12-servletapi.html)	NA	Unspecified	First official specification, added `RequestDispatcher`, `ServletContext`
Servlet 2.0		NA	JDK 1.1	Part of Java Servlet Development Kit 2.0
Servlet 1.0	June 1997	NA		

Compared with other web application models

The advantages of using servlets are their fast performance and ease of use combined with more power over traditional CGI (Common Gateway Interface). Traditional CGI scripts written in Java have a number of performance disadvantages:

- When an HTTP request is made, a new process is created each time the CGI script is called. The overhead associated with process creation can dominate the workload especially when the script does relatively fast operations. Thus, process creation will take more time for CGI script execution. In contrast, for servlets, each request is handled by a separate Java thread *within* the web server process, thereby avoiding the overhead associated with forking processes within the HTTP daemon.

- Simultaneous CGI requests will load the CGI script to be copied into memory once per request. With servlets, there is only one copy that persists across requests and is shared between threads.
- Only a single instance answers all requests concurrently. This reduces memory usage and eases the management of persistent data.
- A servlet can be run by a servlet container in a restrictive environment, called a sandbox. This is similar to an applet that runs in the sandbox of the web browser. This enables restricted use of potentially harmful servlets.[3] CGI programs can of course also sandbox themselves, since they are simply OS processes.

Technologies like FastCGI and its derivatives (including SCGI, AJP) do not exhibit the performance disadvantages of CGI, incurred by the constant process spawning. They are, however, roughly as simple as CGI. They are therefore also in contrast with servlets which are substantially more complex.

Life cycle of a servlet

Three methods are central to the life cycle of a servlet. These are `init()`, `service()`, and `destroy()`. They are implemented by every servlet and are invoked at specific times by the server.

- During initialization stage of the servlet life cycle, the web container initializes the servlet instance by calling the `init()` (http://docs.oracle.com/javaee/7/api/javax/servlet/Servlet.html#init) method, passing an object implementing the `javax.servlet.ServletConfig` (http://docs.oracle.com/javaee/7/api/javax/servlet/ServletConfig.html) interface. This configuration object allows the servlet to access name-value initialization parameters from the web application.
- After initialization, the servlet instance can service client requests. Each request is serviced in its own separate thread. The web container calls the `service()` method of the servlet for every request. The `service()` method determines the kind of request being made and dispatches it to an appropriate method to handle the request. The developer of the servlet must provide an implementation for these methods. If a request is made for a method that is not implemented by the servlet, the method of the parent class is called, typically resulting in an error being returned to the requester.
- Finally, the web container calls the `destroy()` method that takes the servlet out of service. The `destroy()` method, like `init()`, is called only once in the lifecycle of a servlet.

The following is a typical user scenario of these methods.

1. Assume that a user requests to visit a URL.
    - The browser then generates an HTTP request for this URL.
    - This request is then sent to the appropriate server.
2. The HTTP request is received by the web server and forwarded to the servlet container.
    - The container maps this request to a particular servlet.
    - The servlet is dynamically retrieved and loaded into the address space of the container.
3. The container invokes the `init()` method of the servlet.
    - This method is invoked only when the servlet is first loaded into memory.
    - It is possible to pass initialization parameters to the servlet so that it may configure itself.
4. The container invokes the `service()` method of the servlet.
    - This method is called to process the HTTP request.
    - The servlet may read data that has been provided in the HTTP request.
    - The servlet may also formulate an HTTP response for the client.

5. The servlet remains in the container's address space and is available to process any other HTTP requests received from clients.
    - The `service()` method is called for each HTTP request.
6. The container may, at some point, decide to unload the servlet from its memory.
    - The algorithms by which this decision is made are specific to each container.
7. The container calls the servlet's `destroy()` method to relinquish any resources such as file handles that are allocated for the servlet; important data may be saved to a persistent store.
8. The memory allocated for the servlet and its objects can then be garbage collected.

Example

The following example servlet prints how many times its `service()` method was called.

Note that `HttpServlet` is a subclass of `GenericServlet`, an implementation of the `Servlet` interface.

The `service()` method of `HttpServlet` class dispatches requests to the methods `doGet()`, `doPost()`, `doPut()`, `doDelete()`, and so on; according to the HTTP request. In the example below `service()` is overridden and does not distinguish which HTTP request method it serves.

```
import java.io.IOException;

import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class ServletLifeCycleExample extends HttpServlet {

    private int count;

    @Override
    public void init(final ServletConfig config) throws ServletException {
        super.init(config);
        getServletContext().log("init() called");
        count = 0;
    }

    @Override
    protected void service(final HttpServletRequest request, final HttpServletResponse response) thro
ws ServletException, IOException {
        getServletContext().log("service() called");
        count++;
        response.getWriter().write("Incrementing the count to " + count);
    }

    @Override
    public void destroy() {
        getServletContext().log("destroy() called");
    }
}
```

Container servers

The specification for Servlet technology has been implemented in many products. See a list of implementations on the Web container page.

References

1. "servlet" (http://www.webopedia.com/TERM/S/servlet.html). http://www.webopedia.com/: WEBOPEDIA. Retrieved 2011-04-27. "A small java program that runs on a server. The term usually refers to a Java applet which runs within a web server environment. This is analogous to a Java applet that runs within a web browser environment."
2. "Servlet (Java(TM) EE 7 Specification APIs)" (http://docs.oracle.com/javaee/7/api/javax/servlet/Servlet.html). *oracle.com*. Retrieved 22 November 2016.
3. "Servlet Essentials - Chapter 1" (http://www.novocode.com/doc/servlet-essentials/chapter1.html). *novocode.com*. Retrieved 2016-11-22.
4. "Pavni Diwanji" (https://www.fosi.org/people/pavni-diwanji/). *Family Online Safety Institute*. Retrieved 12 November 2016.
5. "Servlet History | community.oracle.com" (https://community.oracle.com/blogs/driscoll/2005/12/10/servlet-history). Weblogs.java.net. 2005-12-10. Retrieved 2013-06-14.
6. "What's new in Servlet 3.1 ? - Java EE 7 moving forward (Arun Gupta, Miles to go ...)" (https://blogs.oracle.com/arungupta/entry/what_s_new_in_servlet). *oracle.com*. Retrieved 22 November 2016.

External links

- JSR 369 (https://www.jcp.org/en/jsr/detail?id=369) - Java servlet 4.0 documentation
- JSR 340 (https://www.jcp.org/en/jsr/detail?id=340) - Java servlet 3.1 documentation
- JSR 315 (https://www.jcp.org/en/jsr/detail?id=315) - Java servlet 3.0 documentation
- JSR 154 (https://www.jcp.org/en/jsr/detail?id=154) - Java servlet 2.4 documentation
- JSR 53 (https://www.jcp.org/en/jsr/detail?id=53) - Java servlet 2.3 documentation