ServletOverview

June 18, 2024

1 What is servlet ? Explain its life cycle. Give its characteristics.

A Servlet is a Java class used to extend the capabilities of servers that host applications accessed by means of a request-response programming model. Servlets are commonly used to extend the applications hosted by web servers, for instance, to process a form data sent by a browser.

Servlet Life Cycle:

1. **Loading and Instantiation:** The servlet container loads the servlet during startup or when the first request is made. The servlet instance is created using the `newInstance()` method of the `Class` class.
2. **Initialization:** The servlet container calls the `init()` method and passes an object implementing the `ServletConfig` interface. This method is called only once throughout the lifecycle.
3. **Request Handling:** For each client request, the servlet container starts a new thread and calls the service method. The `service()` method checks the type of the HTTP request and calls the appropriate method (`doGet()`, `doPost()`, `doPut()`, `doDelete()`, etc).
4. **Removal from Service (Destruction):** The servlet container calls the `destroy()` method before removing the servlet instance, giving it a chance to clean up resources and save state. This method, like `init()`, is called only once.

Characteristics of Servlet:

1. **Platform Independent:** Servlets are written in Java, so they are platform independent.
2. **Performance:** Since servlets are managed by the JVM, it's faster than traditional CGI and also has the capability to handle multiple requests.
3. **Security:** Java has built-in security features, so servlets are secure.
4. **Portability:** Being written in Java, servlets are portable across operating systems and server configurations.
5. **Robust:** Java's strong memory management prevents the servlet from crashing the server.
6. **Persistent:** Servlets remain in memory waiting for requests, so they provide faster responses than processes that must be started up upon each request.
7. **Configurable:** Servlets have configuration parameters that can be used to affect the behavior of the servlet.

2 write a servlet for displaying a string "HELLO WORLD!"

Here is a simple example of a Servlet that displays the string "HELLO WORLD!".

[ ]:
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorldServlet extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html");

        PrintWriter out = response.getWriter();
        out.println("<h1>HELLO WORLD!</h1>");
    }
}
```

This servlet overrides the `doGet` method from the `HttpServlet` class. When a GET request is made to this servlet, it sets the response content type to "text/html" and then gets a `PrintWriter` object from the response. It uses this `PrintWriter` to send an HTML string to the client. The HTML string contains the message "HELLO WORLD!" inside an `<h1>` tag.

3 What do you mean by Common Gateway Interface (CGI) ? How does CGI work ?

Common Gateway Interface (CGI) is a standard protocol for web servers to execute programs that execute like console applications (also called command-line interface programs) running on a server that generates web pages dynamically. Such programs are known as CGI scripts or simply CGIs; they are usually written in a scripting language, but can be written in any programming language.

How CGI works:

1. **Client Request:** The process starts when the client (usually a web browser) sends a request to the server (web server). This request could be for a static resource (like an HTML file) or a dynamic one (like a CGI script).
2. **Server Response:** If the request is for a CGI script, the web server forwards the request to the appropriate CGI program. The server sets the environment variables with the request information and launches the CGI script.
3. **CGI Script Processing:** The CGI script processes the request. It reads data, processes it, and generates an output. The data can come from various sources like form data, cookies, server variables, etc.
4. **Output Return:** The CGI script sends the output back to the web server, along with a status line and response headers. The output is usually in HTML, but it can be in any format like XML, JSON, plain text, etc.

5. **Server Sends Response:** The web server sends this output back to the client where it's displayed to the user.

The main disadvantage of CGI is that each request starts a new process, which can be expensive in terms of resources and can slow down the server if there are many simultaneous requests. This is why newer technologies like Servlets, ASP.NET, PHP, etc., which allow for persistent processes and thread-based handling, have largely replaced CGI.

4 What are the advantages of using Servlets over CGI?

Servlets have several advantages over CGI, including:

1. **Performance:** In CGI, for each request, a new process is created. In contrast, servlets are based on a multithreaded model where starting a new thread for each request is more efficient than starting a new process. This makes servlets faster and more scalable.
2. **Persistence:** Once a servlet is loaded, it remains in memory and can handle multiple requests. This persistence feature provides a significant performance boost.
3. **Access to the Java Ecosystem:** Servlets are written in Java, which means they have access to the entire Java ecosystem. This includes strong object-oriented programming capabilities, robustness, security, garbage collection, exception handling, and the extensive Java Standard Edition (SE) and Java Enterprise Edition (EE) APIs.
4. **Portability:** Servlets are platform-independent because they are written in Java. They can run on any servlet container that complies with the Java Servlet API.
5. **Session Management:** Servlets provide APIs for session management and cookies, which are not readily available in CGI.
6. **Integration with JSP:** Servlets can be integrated with JSP (Java Server Pages) to create dynamic web content.
7. **Security:** Servlets inherit Java's robust security model.
8. **Cost:** Servlets are free and open source.

5 Explain servlet architecture.

Servlet architecture is based on a set of interfaces and classes that establish a contract between the servlet and the web container in which it runs. Here's a high-level overview:

1. **Servlet Interface:** This is the central abstraction and all servlets must implement this interface, either directly or, more commonly, by extending a class that implements it such as `HttpServlet`.
2. **GenericServlet Class:** This is an abstract class that implements the `Servlet` interface and is protocol-independent. It provides implementations for all the methods of the `Servlet` interface except for the `service()` method.
3. **HttpServlet Class:** This is an abstract class that extends `GenericServlet` and provides a framework for handling the HTTP protocol. It has methods like `doGet()`, `doPost()`, etc., corresponding to HTTP methods.

4. **Web Container:** This is the part of the web server that interacts with the servlets. It 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.

Servlet Workflow:

1. **Loading and Initialization:** The servlet container loads the servlet during startup or when the first request is made. The `init()` method is called by the container to initialize the servlet.
2. **Request Handling:** For each client request, the servlet container starts a new thread and calls the `service()` method. The `service()` method checks the type of the HTTP request and calls the appropriate method (`doGet()`, `doPost()`, etc).
3. **Removal from Service:** The servlet container calls the `destroy()` method before removing the servlet instance, giving it a chance to clean up resources and save state.

This architecture allows servlets to be platform-independent and extensible while providing secure, robust web applications.

6 What are the different methods available in the HttpServlet class?

The `HttpServlet` class provides methods for handling HTTP requests. The methods correspond to the HTTP methods and are called by the `service()` method of the `HttpServlet` class. Here are the main methods:

1. **doGet(HttpServletRequest req, HttpServletResponse resp):** This method handles HTTP GET requests. The GET method requests a representation of the specified resource.
2. **doPost(HttpServletRequest req, HttpServletResponse resp):** This method handles HTTP POST requests. The POST method is used to submit an entity to the specified resource, often causing a change in state or side effects on the server.
3. **doPut(HttpServletRequest req, HttpServletResponse resp):** This method handles HTTP PUT requests. The PUT method replaces all current representations of the target resource with the request payload.
4. **doDelete(HttpServletRequest req, HttpServletResponse resp):** This method handles HTTP DELETE requests. The DELETE method deletes the specified resource.
5. **doHead(HttpServletRequest req, HttpServletResponse resp):** This method handles HTTP HEAD requests. The HEAD method asks for a response identical to that of a GET request, but without the response body.
6. **doOptions(HttpServletRequest req, HttpServletResponse resp):** This method handles HTTP OPTIONS requests. The OPTIONS method is used to describe the communication options for the target resource.
7. **doTrace(HttpServletRequest req, HttpServletResponse resp):** This method handles HTTP TRACE requests. The TRACE method performs a message loop-back test along the path to the target resource.

8. **service(HttpServletRequest req, HttpServletResponse resp):** This method dispatches client requests to the methods listed above. It's called by the server (via the `service` method) to allow a servlet to handle a GET, POST, PUT, DELETE, HEAD, OPTIONS, TRACE request.
9. **init() and destroy():** These are lifecycle methods inherited from the `Servlet` interface. The `init` method is called once when the servlet is first loaded, and the `destroy` method is called once before the servlet is unloaded.
10. **getServletInfo():** This method returns information about the servlet, such as author, version, and copyright. It's not specific to `HttpServlet`, but is part of the `Servlet` interface.

7 Write short notes on servlet interface.

The `Servlet` interface is the central abstraction in the Java Servlet API. All servlets must implement this interface, either directly or, more commonly, by extending a class that implements it such as `HttpServlet`.

The `Servlet` interface defines methods to manage the servlet lifecycle and to service client requests. Here are the methods defined in the `Servlet` interface:

1. **init(ServletConfig config):** This method is called by the servlet container to indicate to a servlet that the servlet is being placed into service. It is called only once during the lifecycle of a servlet, before it starts servicing any requests.
2. **service(ServletRequest req, ServletResponse res):** This method is called by the servlet container to allow the servlet to respond to a request. This method is called for each request that the servlet services and is the main method to perform actual task.
3. **destroy():** This method is called by the servlet container to indicate to a servlet that the servlet is being taken out of service. It is called only once, at the end of the servlet's lifecycle.
4. **getServletConfig():** This method returns a `ServletConfig` object, which contains initialization and startup parameters for the servlet. The `ServletConfig` object returned is the one passed to the `init` method.
5. **getServletInfo():** This method returns information about the servlet, such as author, version, and copyright.

The `Servlet` interface provides a standard way for the servlet container to interact with the servlet. The container uses the `init` method to initialize the servlet, calls `service` to process requests, and calls `destroy` to take the servlet out of service. The `getServletConfig` and `getServletInfo` methods provide the container with configuration information and metadata about the servlet.

8 JSP lifecycle example

The lifecycle of a JSP (Java Server Pages) involves the following steps:

1. **Translation:** The web container translates the JSP page into a servlet class. This is a one-time process that happens when the JSP page is requested for the first time or when the JSP page has changed.

2. **Compilation:** The servlet class generated in the translation phase is compiled by the web container.
3. **Loading and Instantiation:** The web container loads the servlet class and creates an instance of it.
4. **Initialization:** The web container calls the `init()` method to initialize the servlet instance. This method is called only once throughout the lifecycle.
5. **Request Processing:** For each client request, the web container starts a new thread and calls the `_jspService()` method. Unlike servlets, JSPs have the `_jspService()` method instead of the `service()` method. This method is called for each request and it can't be overridden.
6. **Destroy:** The web container calls the `destroy()` method to remove the servlet instance. This method is called only once, just like the `init()` method.

Here is an example of a simple JSP page:

[ ]:
```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"␣
 ↪pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Sample JSP Page</title>
</head>
<body>
    <h1>Hello, World!</h1>
    <p>Current time: <%= new java.util.Date() %></p>
</body>
</html>
```

In this JSP, the `<%= new java.util.Date() %>` is a scriptlet that gets executed on the server side and its result is sent to the client. When this JSP is requested for the first time, it goes through the lifecycle steps described above. The JSP is translated into a servlet, compiled, loaded, and then the `_jspService()` method is called to process the request. The result is an HTML page with the current date and time.

9 Write note on Tomcat server?

Apache Tomcat, often referred to as Tomcat Server, is an open-source Java Servlet Container developed by the Apache Software Foundation (ASF). It implements several Java EE specifications including Java Servlet, JavaServer Pages (JSP), Java EL, and WebSocket, and provides a "pure Java" HTTP web server environment in which Java code can run.

Key Features of Tomcat Server:

1. **Lightweight and Flexible:** Tomcat is lighter and requires less resources compared to other application servers. It's often used for simple web applications or microservices.

2. **Java Servlet Container:** Tomcat implements the Java Servlet and the JavaServer Pages (JSP) specifications from Oracle, providing an environment for Java code to run in.
3. **Webserver:** Tomcat includes its own HTTP server. It is often used as a standalone server for applications that only require servlet/JSP processing.
4. **Integration:** Tomcat can also be used in conjunction with other web servers like Apache HTTP Server for serving static content, while dynamically generated content is handled by Tomcat.
5. **Tooling:** It provides tools for configuration and management, which can be graphical or command line based. It can also be configured by editing XML configuration files.
6. **Security:** Tomcat includes a realm mechanism that can be configured to support a variety of authentication methods, including SSL and integration with enterprise security systems.
7. **Community Support:** Being open-source, it has a large community of developers who contribute to its development and provide support.
8. **Portability:** Applications developed with Tomcat and the Servlet API are portable to other Servlet/JSP containers.

Tomcat is not a full Java EE server. It doesn't support the full range of Java EE technologies such as Java Message Service (JMS), Enterprise JavaBeans (EJBs), or Java Transaction API (JTA). However, it's often used in development environments and for applications that don't require the full Java EE platform.

10 Thank You!