



**FREE Live
Classes**



 Theory
 Coding



Core Java

by Karthik Ponnusamy

Course Contents:-

- Core Java Basic Concepts
- OOP Concepts
- Fundamentals of Java Programming
- Java Control Flow Statements
- Deep dive into Collection Framework
- Deep dive into Multithreading
- Java 1.8 Features

Batch Starts from Jan - 18 - 2022
Mon to Thurs - 10 PM to 11 PM CST /
9:30 AM to 10:30 AM IST

Day 04: Agenda

Java Servlets & JSPs

Servlets - What is it?

What is Servlet Container?

Servlet Lifecycle - `init()`, `service()`, `destroy()`

Servlets -Practical Implementation

JSP - What is it? Why is it required?

How to write a sample JSP file?

Implicit Objects of JSP

Servlets - What is it?



Servlets are **Java-based server-side components** that are used to process and respond to client requests received by a web server. They are a part of the Java Enterprise Edition (**Java EE**) platform and provide a **powerful mechanism for developing dynamic web applications**.

Servlets work by accepting **HTTP requests** from clients and generating **HTTP responses** that are returned to the clients. They can handle a wide range of request types, including **GET, POST, PUT, DELETE**, and more. Servlets are typically used to implement the business logic of a web application, such as **handling user authentication, processing form data, and accessing databases**.

Servlets are designed to run on a **web server** that supports the Java Servlet specification, such as **Apache Tomcat, Jetty, or GlassFish**.

They can be **deployed as part of a web application** and are typically packaged as a **WAR** (Web Archive) file.

Servlets are often used in combination with other Java technologies, such as JavaServer Pages (**JSP**) and JavaServer Faces (**JSF**), to create complex web applications. They can also be **used to create web services** that provide access to business logic and data over the internet.

Overall, servlets are a **powerful tool for building dynamic web applications in Java**. They provide a flexible and scalable architecture that can be used to create a wide range of web-based applications and services.

Servlets - What is it?



A servlet can be thought of as **a small program that runs on a web server and responds to client requests**. When a client sends a request to the web server, the servlet receives it and processes it, typically by performing some sort of computation or accessing a database. The servlet then generates a response, which is sent back to the client.

Client Request --> Web Server --> Servlet --> Database --> Servlet --> Web Server --> Client Response

In this diagram, the client sends a request to the web server, which passes the request to the appropriate servlet. The servlet then interacts with a database (if necessary) and generates a response, which is sent back to the web server. The web server then sends the response back to the client.

Overall, servlets provide a powerful way to build dynamic web applications, by **allowing developers to write Java code that runs on the server** and responds to client requests in real-time.

Servlets - What is it?



Real world example:-

When you use a website, such as **logging in or filling out a form**, your request is sent to a servlet on the server, which processes the request and generates a response.

Think of it like ordering food at a restaurant. When you place your order, the waiter takes your request to the kitchen (the servlet), where the chef (the servlet) prepares your food (the response) and sends it back to the waiter, who then delivers it to you.

Similarly, when you use a website, your request is sent to the servlet, which performs some kind of computation or accesses a database, and then generates a response that is sent back to your browser.

The servlet is responsible for handling the communication between your browser and the server, making sure that your requests are processed correctly and that the response is delivered back to you in a timely manner.

Overall, **servlets are an important part of building dynamic web applications**, by allowing developers to write Java code that can handle client requests and generate responses in real-time.

What is Servlet Container?



A **Servlet container**, also known as a **Servlet engine**, is a component of a web server that **manages the execution of Servlets and other Java web components**. It **provides a runtime environment** for web applications that use Servlets, JavaServer Pages (JSP), and related Java technologies.

A Servlet container is responsible for several tasks, including:

1. Loading and initializing Servlets and other web components
2. Managing the lifecycle of Servlets
3. Handling incoming client requests and routing them to the appropriate Servlet
4. Managing the threading model and concurrency of Servlets
5. Providing security and access control mechanisms for Servlets
6. Managing the performance and scalability of the web application

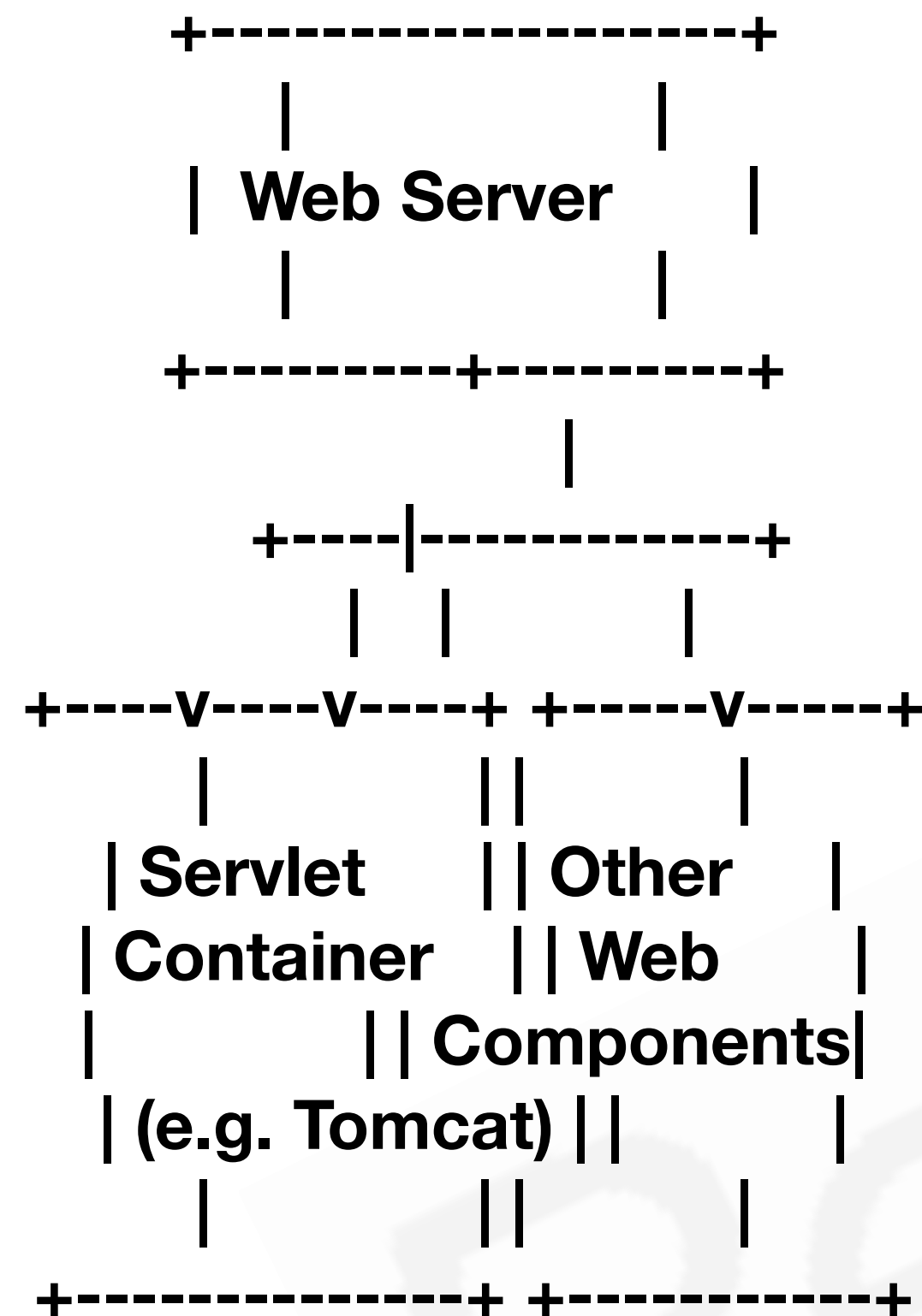
Popular Servlet containers include Apache Tomcat, Jetty, and GlassFish. These containers are typically implemented as Java web applications themselves and run within a Java Virtual Machine (JVM).

In summary, a Servlet container provides a runtime environment for Java web components, including Servlets, and manages the execution of these components within a web server. It is a critical component of Java-based web applications and is **responsible for ensuring the proper functioning and performance of the application**.

What is Servlet Container?



Components of a Servlet container:



In this diagram, the Servlet container is shown as a separate component that runs within the web server. The container is responsible for managing the execution of Servlets and other web components, including loading and initializing them, handling incoming client requests, managing the lifecycle of Servlets, and providing security and access control mechanisms.

The Servlet container typically runs within a Java Virtual Machine (JVM), which provides a runtime environment for executing Java code. The container communicates with the web server via a standard interface, such as the Java Servlet API.

The web server is responsible for handling incoming client requests and routing them to the appropriate Servlet container, which then processes the request and generates a response that is sent back to the client. Other web components, such as JavaServer Pages (JSP), may also run within the Servlet container.

Overall, the Servlet container is a critical component of Java-based web applications and is responsible for ensuring the proper functioning and performance of the application.

Web Server vs Servlet Container



You need to understand the difference between Servlet Containers and Web Servers.

****Web Server:****

A web server is a software application or a hardware device that handles HTTP requests from clients (typically web browsers) and responds with HTML pages, images, files, or other resources. It's primarily **responsible for serving static content**, such as HTML, CSS, JavaScript, and media files. Web servers also manage security, authentication, and basic request handling. Examples of web servers include **Apache HTTP Server (httpd), Nginx, and Microsoft Internet Information Services (IIS)**.

****Servlet Container:****

A servlet container, also known as a servlet engine, **is a component of a web server or application server** that's responsible **for managing the execution of Java Servlets**. Java Servlets are Java-based programs that extend the capabilities of a web server to provide dynamic content generation and server-side processing. Servlet containers **provide a runtime environment for servlets to run** and manage their lifecycle, handling tasks such as initialization, request processing, and destruction.

Web Server vs Servlet Container



****Functionality:****

- Web Server: Primarily serves static content like HTML, images, and files.

****Content Handling:****

- Web Server: Focuses on delivering static content efficiently.

****Request Handling:****

- Web Server: Handles HTTP requests and serves static content directly.

****Extension:****

- Web Server: Can be extended with modules or plugins for specific functionalities (e.g., URL rewriting, load balancing).

****Examples:****

- Web Server: Apache HTTP Server, Nginx, IIS.

****Deployment:****

- Web Server: Generally easier to set up and configure for serving static content.

****Use Cases:****

- Web Server: Suitable for hosting websites with mainly static content.

- Servlet Container: Executes Java servlets to generate dynamic content and handle server-side processing.
- Servlet Container: Specializes in executing dynamic code and generating dynamic content in response to client requests.
- Servlet Container: Manages the execution of servlets, which generate responses based on dynamic processing.
- Servlet Container: Provides a runtime environment for servlets to run, handling their lifecycle and interactions.
- Servlet Container: Apache Tomcat, Jetty, WildFly (formerly known as JBoss).
- Servlet Container: Requires configuration for servlet deployment and often accompanies Java-based web applications.
- Servlet Container: Ideal for Java-based web applications that require dynamic content generation and server-side processing.

In practical terms, **web servers and servlet containers often work together** in web application deployment. A web server might handle initial requests and static content delivery, while requests that require dynamic processing are forwarded to a servlet container for execution. This combination ensures efficient handling of both static and dynamic aspects of a web application.

Servlet Lifecycle - init(), service(), destroy()

The **lifecycle of a Servlet is managed by the Servlet container** and consists of three main phases:

initialization, request processing, and destruction.

These phases are implemented using the init(), service(), and destroy() methods, respectively.

Initialization: When a Servlet is first loaded by the Servlet container, its init() method is called. This method is used **to initialize the Servlet** and perform any one-time setup tasks, such as opening a database connection or loading configuration data. The init() method is only called once during the lifetime of a Servlet, typically when the Servlet is first deployed or when the server is restarted.

Request Processing: Once the Servlet has been initialized, its service() method is called to process incoming client requests. This method is called **for each request that the Servlet is responsible for handling**. The service() method receives two parameters: an HTTP request object that encapsulates the client request, and an HTTP response object that the Servlet can use to generate the response.

Destruction: When the Servlet container determines that a Servlet is no longer needed, its destroy() method is called. This method is used **to perform any cleanup tasks, such as closing database connections or releasing system resources**. The destroy() method is only called once during the lifetime of a Servlet, typically when the server is shut down or when the Servlet is undeployed.

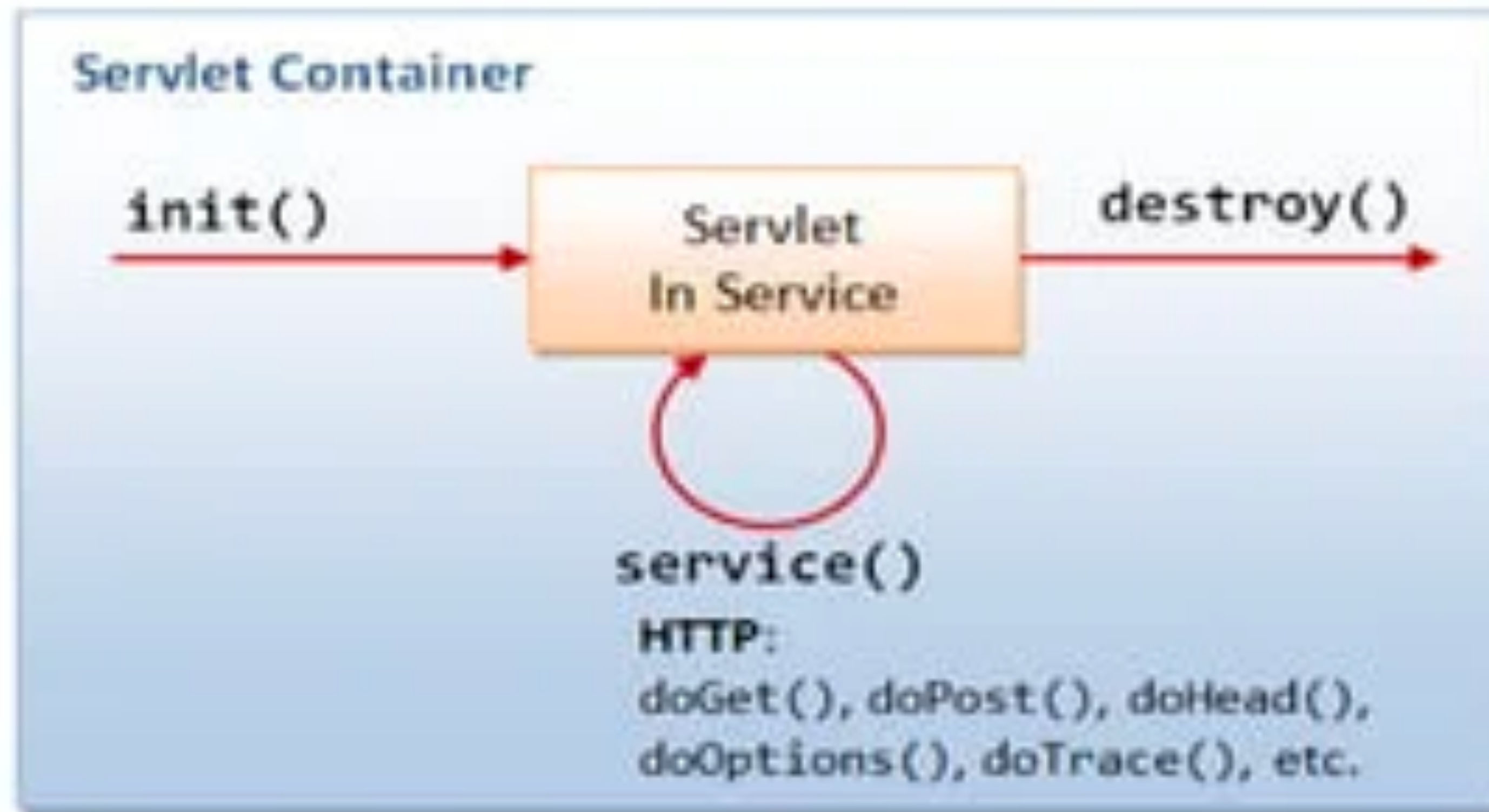
Here is a summary of the three main methods in the Servlet lifecycle:

init(): Called once when the Servlet is first loaded by the Servlet container. Used to perform one-time setup tasks.

service(): Called for each incoming client request. Used to process the request and generate a response.

destroy(): Called once when the Servlet is no longer needed by the Servlet container. Used to perform cleanup tasks.

Servlet life cycle



Servlets - Why required?



Servlets are widely used in web application development because they provide a number of benefits:

Platform independence: Servlets are written in Java, which is platform-independent, meaning they can run on any platform that supports the Java Virtual Machine (JVM). **Servlets -> Chef and Platforms -> Flavors (spice, sweet etc,)**

High performance: Servlets are designed to be efficient and lightweight, allowing them to handle a large number of requests quickly and with low overhead.

Scalability: Servlets can be deployed to a Servlet container, which can manage multiple instances of the same Servlet to handle increased traffic and ensure high availability.

Robustness: Servlets are part of the Java EE standard, which provides a well-defined framework for web application development and ensures interoperability with other Java EE components.

Security: Servlets can be secured using a variety of mechanisms, such as HTTPS, SSL, and authentication and authorization frameworks.

Flexibility: Servlets can be combined with other Java EE technologies, such as JSP, to provide a rich and flexible web application development platform.

Overall, Servlets provide a powerful and flexible platform for building robust, scalable, and high-performance web applications.

Servlet's - Advantages & Disadvantages

Advantages of Servlets:

Platform independence: Servlets are platform-independent, which means they can run on any platform that supports Java.

Performance: Servlets are faster than traditional CGI scripts because they are loaded once and reused for subsequent requests.

Scalability: Servlets are highly scalable, which means they can handle a large number of requests at the same time without affecting the performance of the application.

Security: Servlets provide a more secure environment for web applications, as they run within the Java Virtual Machine and can be subject to Java's security features.

Flexibility: Servlets are highly flexible, as they can interact with various databases, JavaBeans, and other Java APIs to provide dynamic content.

Disadvantages of Servlets:

Complexity: Servlets require a deeper understanding of Java and web application architecture, which can make them more complex to develop than traditional CGI scripts.

Lack of standardization: The servlet API is not as standardized as other Java APIs, which can lead to inconsistencies and compatibility issues across different platforms.

Code maintenance: Because Servlets are highly customizable, they can become difficult to maintain and update as they grow in complexity.

Limited support for dynamic content: While Servlets can provide dynamic content, they are not as powerful as other server-side technologies like JSP and ASP.

Servlet container limitations: The performance and capabilities of Servlets are limited by the capabilities of the Servlet container that they run in.

Overall, Servlets are a powerful technology for building dynamic and scalable web applications, **but they require a deep understanding of Java and web application architecture to use effectively.**

JSP - What is it? Why is it required?



JSP stands for **JavaServer Pages**, which is **a technology used for creating dynamic web pages using Java**. JSP pages are **similar to HTML pages, but they allow you to embed Java code into the HTML code to create dynamic content**.

When a JSP page is requested by a client, the **web server processes the JSP page and generates an HTML page that is sent back to the client**. The Java code in the JSP page is executed on the server side, allowing you to access databases, manipulate data, and perform other server-side operations.

JSP pages can be used to create a variety of web-based applications, including **e-commerce websites, online forums, and social networking sites**. They are often used in combination with other Java-based technologies, such as Servlets and JDBC, to provide a complete web development platform.

JSP pages are easy to learn and use, and they provide a high degree of flexibility and customization for web developers. They also **provide a clear separation between the presentation layer (the JSP page) and the business logic layer (the Java code)**, making it easier to maintain and update web applications.

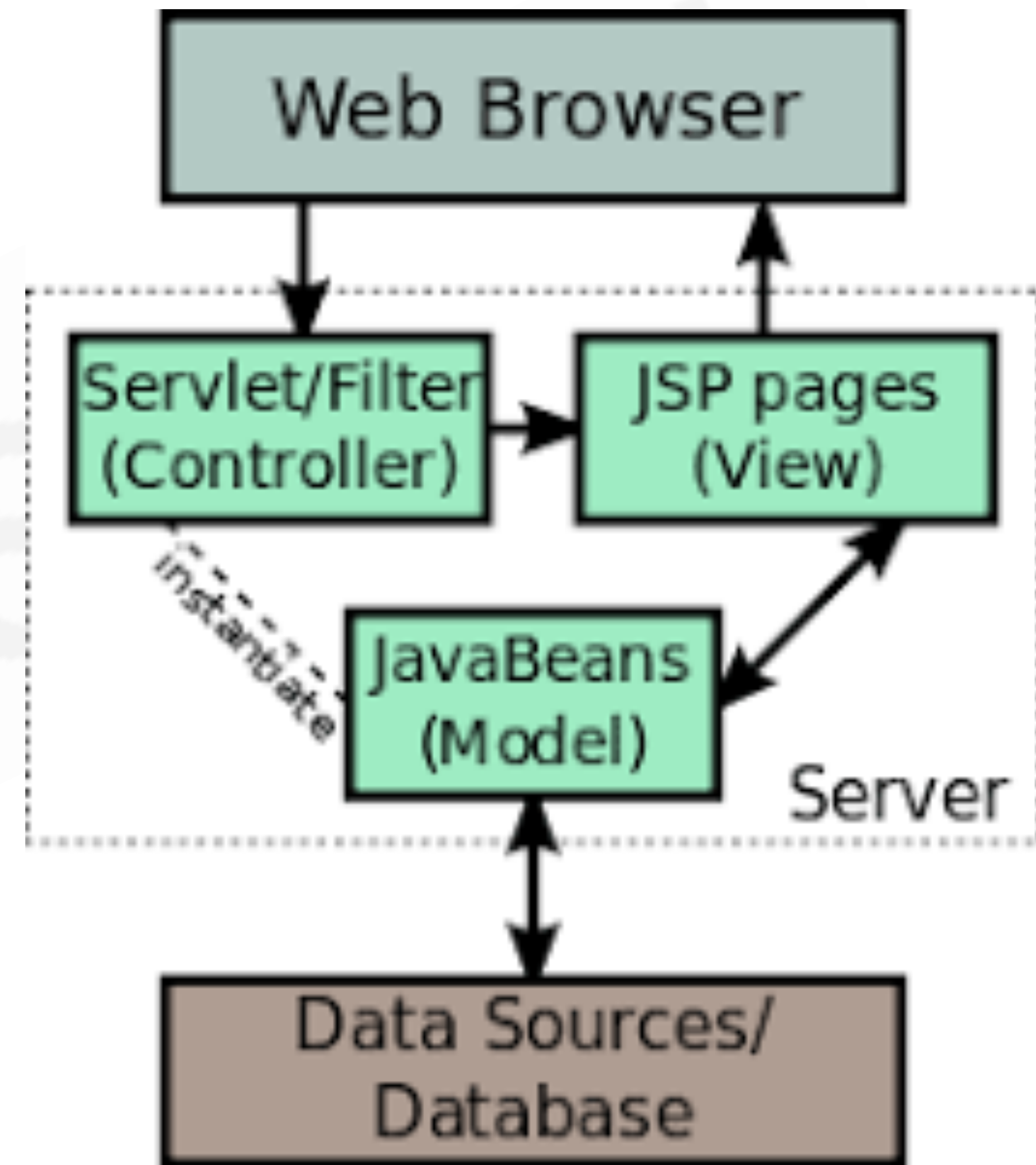
JSP - What is it? Why is it required?

JSP or JavaServer Pages is a technology used to create dynamic web pages. Think of it like **building a website, but with the ability to add functionality using Java code.**

When a user requests a JSP page, the server processes the JSP and generates an HTML page that is sent back to the user. This allows you to create web pages that can do things like access databases, interact with other systems, and perform other server-side tasks.

JSP pages can be used to create a variety of web-based applications, such as e-commerce websites, social networking sites, and online forums. They are often used in combination with other Java-based technologies, like Servlets and JDBC, to provide a complete web development platform.

Overall, JSP provides a flexible and customizable way to build dynamic web pages using Java code.



JSP - What is it? Why is it required?



JSP or JavaServer Pages is required for a number of reasons:

Dynamic web pages: JSP allows you to create dynamic web pages that can change based on user input, database queries, or other server-side factors. This allows you to create websites that are more interactive and engaging for users.

Code reusability: JSP pages can be reused across multiple pages, allowing you to avoid duplication of code and make your web development process more efficient.

Separation of concerns: JSP allows you to separate the presentation layer (the JSP page) from the business logic layer (the Java code), making it easier to maintain and update your web application.

Java-based: JSP is based on Java, which is a widely used and powerful programming language. This makes it easier to find developers with Java skills and to integrate with other Java-based technologies, such as Servlets and JDBC.

Customization: JSP provides a high degree of customization, allowing you to create web pages that are tailored to your specific needs and requirements.

Overall, JSP is required for building dynamic web pages that are more interactive and engaging for users, and for creating a web development process that is more efficient and maintainable.

JSPs - Advantages & Disadvantages



Advantages of JSP:

Easy to learn: JSP is easy to learn and understand, especially for developers with experience in Java programming language.

Improved performance: JSP pages can be compiled into servlets, which can improve the performance of your web application by reducing the time needed to process requests and responses.

Code reusability: JSP pages can be reused across multiple pages, allowing you to avoid duplication of code and make your web development process more efficient.

Separation of concerns: JSP allows you to separate the presentation layer (the JSP page) from the business logic layer (the Java code), making it easier to maintain and update your web application.

Integration with Java-based technologies: JSP can be easily integrated with other Java-based technologies, such as Servlets and JDBC.

Disadvantages of JSP:

Steep learning curve: Although JSP is easy to learn for developers with experience in Java programming language, it may be difficult for beginners who are not familiar with Java.

Limited control over HTML output: JSP pages generate HTML output, which can be limiting if you need full control over the HTML generated by your web application.

Mixing of business logic and presentation: If not properly structured, JSP pages can lead to a mixing of business logic and presentation, which can make it difficult to maintain and update your web application.

Scalability issues: As your web application grows, the number of JSP pages can become unmanageable, leading to scalability issues.

Overall, JSP is a powerful technology for building dynamic web pages, but it has its limitations and requires proper planning and structuring to avoid potential issues.

How to write a sample JSP file?



```
<!DOCTYPE html>
<html>
<head>
  <title>My JSP Page</title>
</head>
<body>
  <h1>Hello World!</h1>
  <p>The current time is <%= new java.util.Date() %></p>
</body>
</html>
```

This JSP file outputs "Hello World!" as an HTML heading and the current date and time as a paragraph. The `<%= ... %>` syntax is used to include Java code that will be executed at runtime and its output will be included in the HTML response. Note that JSP files are typically saved with a .jsp extension and are deployed on a web server along with other web application resources such as Servlets, HTML, CSS, and JavaScript files.

Can I execute a JSP file using Browser? - NO



To run a JSP (JavaServer Pages) page in your browser, **you need to set up a server environment** that supports JSP execution. **Apache Tomcat** is a popular choice for this purpose. Here's a step-by-step guide to help you get started:

Install Apache Tomcat: Download the latest version of Apache Tomcat from the official website: <https://tomcat.apache.org/download-10.cgi>

Deploy Your JSP Page: Once Tomcat is installed, navigate to the Tomcat installation directory. Inside the "webapps" directory, create a new folder for your application. For example, if your JSP file is named "mypage.jsp," create a folder named "mypage." Copy your JSP file into the newly created folder.

Start Tomcat: Go to the "bin" directory within the Tomcat installation. Run the startup script: startup.sh (Linux/Mac) or startup.bat (Windows). Tomcat will start, and you should see console output indicating that the server is running.

Access Your JSP Page: Open a web browser. In the address bar, enter the URL: `http://localhost:8080/your-folder-name/mypage.jsp`
Replace your-folder-name with the name of the folder you created in the "webapps" directory.
Replace mypage.jsp with the name of your JSP file.

View Your JSP Page: After entering the URL, press Enter. Your browser will send a request to Tomcat, which will process the JSP and send the generated HTML back to the browser. The browser will render the HTML content generated by your JSP.

Remember that the **default port for Tomcat is 8080**, but it might be different if you've configured it otherwise. If you encounter any issues, make sure to check the Tomcat logs for any error messages.

Please note that this is a basic setup for running a single JSP page. In real-world applications, you would typically create more structured projects and use frameworks to manage your web application.

Browser can't read the JSP page directly?? - NO



Browsers cannot directly read and interpret JSP files like they do with HTML files. JSP files contain Java code that needs to be executed on the server-side before being sent to the browser as HTML.

Client (Browser) Request:

- When a user enters a URL in their browser to access a JSP page, the browser sends an HTTP request to the server.



1

Server-side Processing:

- The server receives the HTTP request.
- The JSP file is not directly sent to the browser. Instead, the server processes the JSP file using the Java code embedded within it.

2

JSP Compilation:

- The server compiles the **JSP file into a servlet class**. This servlet class contains the Java code from the JSP and additional code to handle the request and response objects.

3

Apache Tomcat

Response to Browser:

- Once the servlet processing is complete, the generated HTML (or other output) is sent as the HTTP response back to the browser.

5

Servlet Execution:

- The servlet container executes the servlet class. The Java code in the servlet class generates dynamic content, typically HTML, by combining Java logic with static content from the JSP.

4

The servlet file that is generated from a JSP file is stored:

<TOMCAT_HOME>/work/Catalina/localhost/<your-web-app>/org/apache/jsp/

Implicit Objects of JSP



Implicit objects in JSP are **predefined objects** that are automatically available for use in a JSP file **without the need for any explicit declaration or instantiation**. They are available at runtime and can be used to perform various operations in JSP pages.

Here are some of the commonly used implicit objects in JSP:

request: The request object provides **access to the client's request information** such as parameters, headers, cookies, etc.

response: The response object provides **access to the server's response information** such as headers, cookies, etc.

out: The out object is used to **send output to the client's browser**.

session: The session object provides **access to session information** such as session ID, session attributes, etc.

application: The application object provides **access to application-level information** such as context parameters, servlet context, etc.

pageContext: The pageContext object provides **access to various JSP page context** attributes such as request, response, session, and application.

config: The config object provides **access to the JSP configuration information** such as initialization parameters.

exception: The exception object is **used to handle any exception** that occurs during the execution of the JSP page.

These implicit objects provide a convenient way to access various information and functionality required in JSP pages without the need for additional code or configuration.

Implicit Objects of JSP

Here are some practical examples of using the JSP implicit objects:



← → ↺ ⓘ localhost:8080/Redsystech/IMPL_OBJ_REQ_RESP.jsp?name=RED

Hello: RED

Hello Karthik!

1. request and response objects:

jsp

Copy code

<%

```
String name = request.getParameter("name");  
response.setContentType("text/html");  
response.getWriter().println("Hello " + name);  
%>
```

In this example, we use the `request` object to get the value of the `name` parameter from the client's request. We then use the `response` object to set the content type of the response and to write a message to the client's browser.

Implicit Objects of JSP

Here are some practical examples of using the JSP implicit objects:

localhost:8080/Redsystech/IMPL_OBJ_SESSION.jsp

Hello: john

2. session object:

```
<%
```

```
session.setAttribute("user","j  
ohn");
```

```
String user =  
(String)session.getAttribute("  
user");
```

```
out.println("Hello: "+user);
```

```
%>
```

jsp

Copy code

```
<%  
session.setAttribute("user", "John");  
String user = (String)session.getAttribute("user");  
out.println("Welcome " + user);  
%>
```

In this example, we use the `session` object to set a session attribute called `user` with the value of "John". We then retrieve the `user` attribute from the session using the `getAttribute()` method and use it to display a personalized welcome message to the client.

Implicit Objects of JSP

Here are some practical examples of using the JSP implicit objects:

```
<%
Integer hitsCount =
(Integer)application.getAttribute("hitCounter")
;
if( hitsCount ==null || hitsCount == 0 ){
    /* First visit */
    out.println("Welcome to my
website!");
    hitsCount = 1;
}else{
    /* return visit */
    out.println("Welcome back to
my website!");
    hitsCount += 1;
}
application.setAttribute("hitCounter",
hitsCount);
%>
```

3. application object:

jsp Copy code

```
<%
application.setAttribute("count", 0);
int count = (Integer)application.getAttribute("count");
application.setAttribute("count", count + 1);
out.println("This page has been viewed " + count + " times");
%>
```

In this example, we use the `application` object to set an application attribute called `count` to 0. We then retrieve the `count` attribute from the application using the `getAttribute()` method and increment it by 1 before setting it back to the application using the `setAttribute()` method. We then display the updated count value to the client.



Implicit Objects of JSP

Here are some practical examples of using the JSP implicit objects:

```
<%  
    try{  
        int z = 1/0;  
    }  
    catch(Exception ex){  
        out.println("Error occurred here!"  
+ ex.getMessage());  
    }  
%>
```

← → ↺ ⓘ localhost:8080/Redsystemtech/IMPL_OBJ_EXCEP.jsp

Error occurred here!/ by zero

4. exception object:

jsp

Copy code

```
<%  
try {  
    // some code that may throw an exception  
} catch (Exception e) {  
    out.println("An error occurred: " + e.getMessage());  
}  
%>
```

In this example, we use the `try-catch` block to catch any exception that may occur during the execution of the JSP page. We then use the `exception` object to retrieve the exception message and display it to the client.

Implicit Objects of JSP

Here are some practical examples of using the JSP implicit objects:

1. PageContext object:

jsp

Copy code

```
<%
```

```
PageContext pageContext = (PageContext)out.getJspContext();
```

```
pageContext.setAttribute("user", "John", PageContext.PAGE_SCOPE);
```

```
String user = (String)pageContext.getAttribute("user", PageContext.PAGE_SCOPE);
```

```
out.println("Welcome " + user);
```

```
%>
```


In this example, we use the `PageContext` object to set a page scope attribute called `user` with the value of "John". We then retrieve the `user` attribute from the page context using the `getAttribute()` method and use it to display a personalized welcome message to the client.

Implicit Objects of JSP

Here are some practical examples of using the JSP implicit objects:

2. ServletConfig object:

jsp

 Copy code

```
<%  
ServletConfig servletConfig = getServletConfig();  
String greeting = servletConfig.getInitParameter("greeting");  
out.println(greeting + " World!");  
%>
```

In this example, we use the `ServletConfig` object to retrieve an initialization parameter called `greeting` from the servlet configuration. We then use the value of `greeting` to display a customized greeting message to the client. The `getServletConfig()` method is an implicit method that returns the `ServletConfig` object for the current JSP page.

