### What is J2EE?

J2EE stands for Java 2 Platform, Enterprise Edition. It is a Java-based platform designed for developing, building, and deploying web-based enterprise applications. J2EE simplifies application development in a thin client, tiered environment, making it suitable for large-scale, mainframe-style computing typical of enterprise-level organizations. It provides a set of services, APIs, and protocols to develop multi-tiered, distributed, and web-based applications. J2EE has evolved and is now referred to as Java EE (Java Platform, Enterprise Edition) and currently as Jakarta EE.

**Key Characteristics**

* Platform-independent: J2EE is built on Java, allowing for deployment on various platforms.
* Java-centric: J2EE leverages Java programming language and its ecosystem.
* Enterprise-focused: Designed for developing complex, scalable, and secure enterprise applications.
* Multi-tiered: J2EE applications typically consist of multiple tiers, including presentation, business logic, and data access layers.
* Safe, stable, and quick: J2EE provides a robust framework for developing applications that meet enterprise-level requirements.
* JavaServer Pages ( JSP)
* Java Servlets
* Enterprise JavaBeans (EJB)
* Java Database Connection ( JDBC)
* Java Transaction API ( JTA) and Java Transaction Service ( JTS)
* Java Naming and Directory Interface ( JNDI)
* Java Message Service ( JMS)
* Java IDL and Remote Method Invocation (RMI)
* Java XML

# **What is Advance Java?**

[Java](https://www.javatpoint.com/java-tutorial) is divided into two parts i.e. **Core Java (J2SE)** and **Advanced Java (JEE)**. The core Java part covers the fundamentals (data types, functions, operators, loops, thread, exception handling, etc.) of the Java programming language. It is used to develop general purpose applications. Whereas **Advanced Java** covers the concepts such as database connectivity, networking, Servlet, web-services, etc.

It is a part of Java programming language. It is an advanced technology or advance version of Java specially designed to develop web-based, network-centric or enterprise applications. It includes the concepts like [Servlet](https://www.javatpoint.com/servlet-tutorial), [JSP](https://www.javatpoint.com/jsp-tutorial), JDBC, [RMI](https://www.javatpoint.com/RMI), [Socket programming](https://www.javatpoint.com/socket-programming), etc. It is a specialization in specific domain.

# **RMI (Remote Method Invocation)**

The **RMI** (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM.

The RMI provides remote communication between the applications using two objects *stub* and *skeleton*.

### Understanding stub and skeleton

RMI uses stub and skeleton object for communication with the remote object.

A **remote object** is an object whose method can be invoked from another JVM. Let's understand the stub and skeleton objects:

### stub

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
3. It waits for the result
4. It reads (unmarshals) the return value or exception, and
5. It finally, returns the value to the caller.

### skeleton

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits (marshals) the result to the caller.

# **Java Socket Programming**

Java Socket programming is used for communication between the applications running on different JRE.

Java Socket programming can be connection-oriented or connection-less.

Socket and ServerSocket classes are used for connection-oriented socket programming and DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

**What is JSP?**

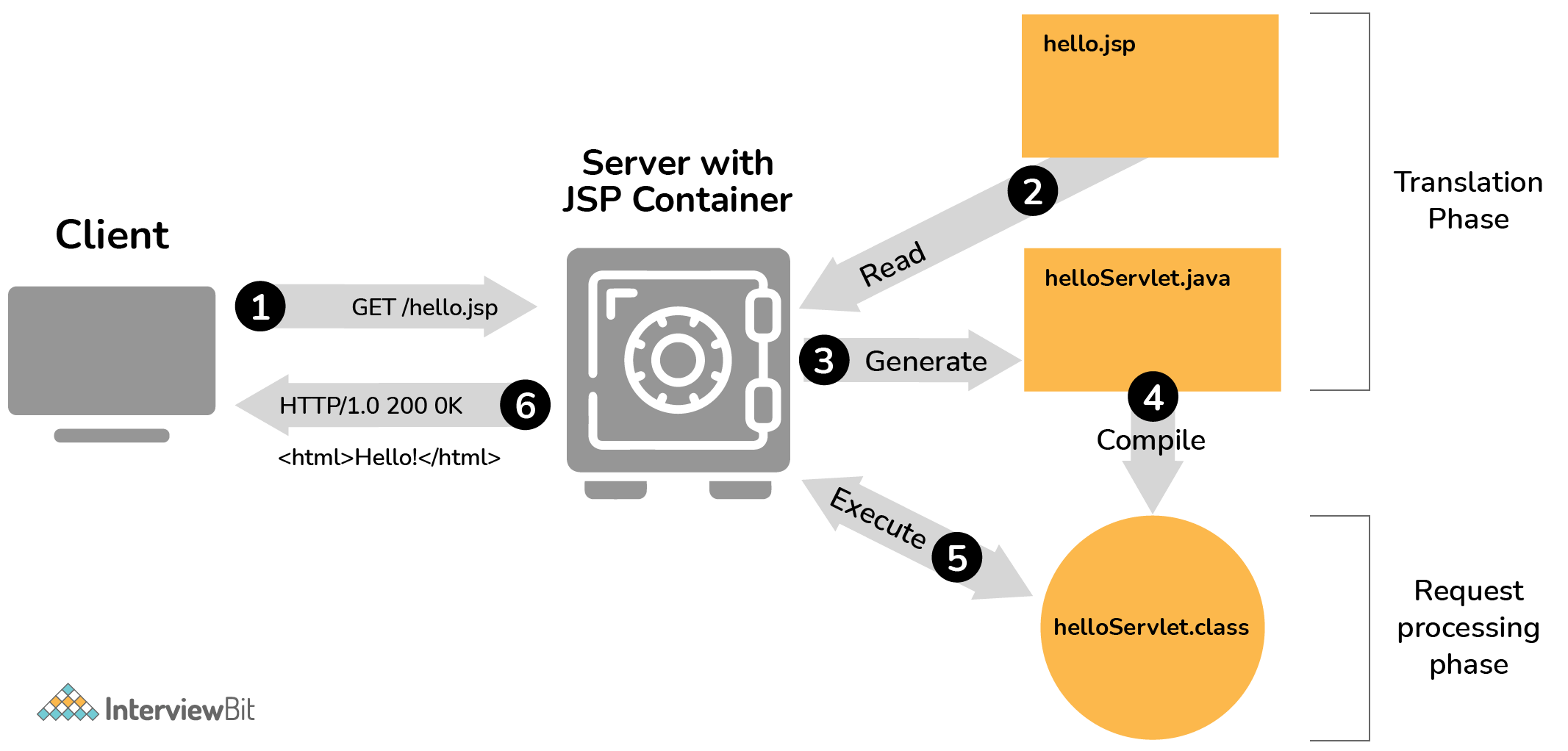
JSP is an abbreviation for **Java Servlet Page**. It is a Server-Side Programming Language used to create dynamic web-pages. It allows programmers to insert Java code into HTML(Hyper Text Markup Langauge) pages using specific JSP tags. The JSP page is implicitly converted into a servlet and it enables some additional features such as Expression Language, Custom Tags, and many more.

### How does JSP processing take place?

The JSP page is turned into a servlet for all the JSP elements to be processed by the server. Then the servlet is executed. The servlet container and the JSP container—are often combined into one package under the name “web container”.

In the translation phase, the JSP container is responsible for converting the JSP page into a servlet and compiling the servlet. This is used to automatically initiate the translation phase for a page when the first request for the page is received.

In the “request processing” phase, the JSP container is also responsible for invoking the JSP page implementation class to process each request and generate the response.



**Why Use JSP When We Have Servlets:**

1. **Separation of Concerns**: JSP focuses on the presentation layer (how the page looks), while servlets handle business logic (what the application does). This separation makes the application easier to develop, maintain, and scale.
2. **Ease of Use for Designers**: JSP allows embedding Java code in HTML, making it more intuitive for web designers. They can work on layout and design without deep knowledge of Java.
3. **Reducing Code Complexity**: JSP simplifies the process of generating HTML, reducing the need for cumbersome out.println statements in servlets.
4. **Reusability with Custom Tags and Tag Libraries**: JSP supports custom tags and tag libraries, allowing complex logic to be encapsulated and reused across pages.
5. **Standardization and Integration**: JSP is a standardized technology with consistent syntax, making it easier to create maintainable and interoperable applications. It also integrates well with other Java EE technologies and frameworks like Spring MVC or Struts.

What are the life-cycle methods for a JSP?

|  |  |
| --- | --- |
| **Method** | **Description** |
| public void jspInit() | It is invoked only once, same as init method of the servlet. |
| Public void jspService (ServletRequest req, ServletResponse res )throws ServletException,IOException | It is invoked at each request, same as service() method of the servlet. |
| public void jspDestroy() | It is invoked only once, same as destroy() method of the servlet. |

### [What are the JSP lifecycle phases?](https://www.digitalocean.com/community/tutorials/jsp-interview-questions-and-answers#what-are-the-jsp-lifecycle-phases)

* 1. **Translation** - JSP container checks the JSP page code and parse it to generate the servlet source code. For example in Tomcat you will find generated servlet class files at **TOMCAT/work/Catalina/localhost/WEBAPP/org/apache/jsp** directory. If the JSP page name is home.jsp, usually the generated servlet class name is home\_jsp and file name is home\_jsp.java
  2. **Compilation** - JSP container compiles the jsp class source code and produce class file in this phase.
  3. **Class Loading** - Container loads the class into memory in this phase.
  4. **Instantiation** - Container invokes the no-args constructor of generated class to load it into memory and instantiate it.
  5. **Initialization** - Container invokes the init method of JSP class object and initializes the servlet config with init params configured in deployment descriptor. After this phase, JSP is ready to handle client requests. Usually from translation to initialization of JSP happens when first request for JSP comes but we can configure it to be loaded and initialized at the time of deployment like servlets using load-on-startup element.
  6. **Request Processing** - This is the longest lifecycle of JSP page and JSP page processes the client requests. The processing is multi-threaded and similar to servlets and for every request a new thread is spawned and ServletRequest and ServletResponse object is created and JSP service method is invoked.
  7. **Destroy** - This is the last phase of JSP lifecycle where JSP class is unloaded from memory. Usually it happens when application is undeployed or the server is shut down.

List out some advantages of using JSP.

* Better performance.
* The compilation of JSP is done before it is processed by the server which eradicates the need for loading of interpreter and code script each time.
* JSP has access to all-powerful enterprises.
* Easy to maintain: JSP can be easily managed because we can easily separate our business logic with presentation logic. In Servlet technology, we mix our business logic with the presentation logic.
* JSP can also be used in combination with servlets.
* Ease of Development: Simplifies creating dynamic web pages with embedded Java code in HTML.
* Separation of Concerns: Supports MVC design, separating presentation from business logic.
* Reusability: Custom tags and tag libraries promote reusable and consistent code.
* Integration: Seamlessly integrates with Java EE technologies and frameworks.
* Platform Independence: Runs on any system supporting Java.
* Session Management: Provides built-in mechanisms for managing user sessions.
* Automatic Compilation: Automatically compiles and deploys JSP pages as Servlets.
* Rich Features: Includes expression language, JSTL, and support for internationalization.

What are the JSP implicit objects?

JSP provides nine implicit objects by default. They are as follows:

|  |  |
| --- | --- |
| **Object** | **Type** |
| 1) out | JspWriter |
| 2) request | HttpServletRequest |
| 3) response | HttpServletResponse |
| 4) config | ServletConfig |
| 5) session | HttpSession |
| 6) application | ServletContext |
| 7) pageContext | PageContext |
| 8) page | Object |
| 9) exception | Throwable |

### What are the various action tags used in JSP?

Various action tags used in JSP are as follows:

* jsp:forward: This action tag forwards the request and response to another resource.
* jsp:include: This action tag is used to include another resource.
* jsp:useBean: This action tag is used to create and locates bean objects.
* jsp:setProperty: This action tag is used to set the value of the property of the bean.
* jsp:getProperty: This action tag is used to print the value of the property of the bean.
* jsp:plugin: This action tag is used to embed another component such as the applet.
* jsp:param: This action tag is used to set the parameter value. It is used in forward and includes mostly.
* jsp:fallback: This action tag can be used to print the message if the plugin is working.

### What is EL in JSP?

The Expression Language(EL) is used in JSP to simplify the accessibility of objects. It provides many objects that can be used directly like param, requestScope, sessionScope, applicationScope, request, session, etc.

Most of the times we use JSP for view purposes and all the business logic is present in servlet code or model classes. When we receive client request in the servlet, we process it and then add attributes in request/session/context scope to be retrieved in JSP code. We also use request params, headers, cookies and init params in JSP to create response views. We can use scriptlets and JSP expressions to retrieve attributes and parameters in JSP with java code and use it for view purpose. But for web designers, java code is hard to understand and that’s why JSP Specs 2.0 introduced Expression Language (EL) through which we can get attributes and parameters easily using HTML like tags. Expression language syntax is `${name}` and we can use EL implicit objects and EL operators to retrieve the attributes from different scopes and use them in JSP page.

### Which directive is used in JSP custom tag?

The JSP taglib directive.

### What is JSTL?

JSP Standard Tag Library is a library of predefined tags that ease the development of JSP.

**Tags provided in JSTL?**

There is 5 type of JSTL tags.

### 1. ****Core Tags****

* **Purpose**: General-purpose tags for common tasks such as iteration, conditionals, and importing content.
* **Examples**:

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<!-- Conditional statement -->

<c:if test="${userLoggedIn}">

Welcome, ${username}!

</c:if>

<!-- Iteration -->

<c:forEach var="item" items="${itemList}">

${item.name} - ${item.price} <br/>

</c:forEach>

<!-- URL import -->

<c:import url="header.jsp" />

### 2. ****Formatting Tags****

* **Purpose**: Formatting and internationalization support for numbers, dates, and messages.
* **Examples**:

<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>

<!-- Formatting numbers -->

<fmt:formatNumber value="${price}" type="currency" />

<!-- Formatting dates -->

<fmt:formatDate value="${currentDate}" pattern="MM/dd/yyyy" />

<!-- Message formatting -->

<fmt:message key="welcome.message">

<fmt:param value="${username}" />

</fmt:message>

### 3. ****SQL Tags****

* **Purpose**: Tags for interacting with relational databases, such as executing queries and updates.
* **Examples**:

<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>

<!-- Setting up a data source -->

<sql:setDataSource var="dbSource" driver="com.mysql.jdbc.Driver"

url="jdbc:mysql://localhost:3306/mydb"

user="root" password="password" />

<!-- Executing a query -->

<sql:query var="result" dataSource="${dbSource}">

SELECT \* FROM users

</sql:query>

<!-- Iterating over query results -->

<c:forEach var="row" items="${result.rows}">

${row.username} - ${row.email} <br/>

</c:forEach>

### 4. ****XML Tags****

* **Purpose**: Tags for parsing and transforming XML documents.
* **Examples**:

<%@ taglib uri="http://java.sun.com/jsp/jstl/xml" prefix="x" %>

<!-- Parsing XML -->

<x:parse var="doc" xml="${xmlString}" />

<!-- Selecting XML nodes -->

<x:out select="$doc/book/title" />

### 5. ****Function Tags****

* **Purpose**: Utility functions for manipulating strings and other data types.
* **Examples**:

<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>

<!-- String manipulation -->

<c:set var="upperCaseName" value="${fn:toUpperCase(username)}" />

## **JSP Scripting elements**

The scripting elements provides the ability to insert java code inside the jsp. There are five types of scripting elements:

**Comments**

* **Purpose**: To include comments in the JSP code that are not sent to the client. These comments are for the developer's reference and are ignored by the JSP engine.
* **Syntax**: <%-- comment --%>
* **Example**:

<%-- This is a JSP comment --%>

**Directives**

* **Purpose**: To provide global information about the JSP page, such as importing classes, defining page-level settings, and including other files.
* **Syntax**: <%@ directive attribute="value" %>
* **Types**: Page, Include, Taglib
* **Examples**:

**Page Directive**:

<%@ page language="java" contentType="text/html; charset=UTF-8" %>

**Include Directive**:

<%@ include file="header.jsp" %>

**Taglib Directive**:

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

**Declarations**

* **Purpose**: To declare variables and methods that can be used throughout the JSP page.
* **Syntax**: <%! Java code %>
* **Example**:

<%!

private int counter = 0;

public int getCounter() {

return counter++;

}

%>

**Scriptlets**

* **Purpose**: To embed Java code that will be executed when the JSP page is requested.
* **Syntax**: <% Java code %>
* **Example**:

<%

String username = "John Doe";

out.println("Hello, " + username);

%>

**Expressions**

* **Purpose**: To output the value of a Java expression directly to the client.
* **Syntax**: <%= expression %>
* **Example**:

<%

int num1 = 5;

int num2 = 10;

%>

Sum: <%= num1 + num2 %>

### Which methods are used for reading form data using JSP?

JSP is used to handle the form data parsing automatically. It dies so by using the following methods depending on the situation:

* **getParameter()** − To get the value of a form parameter, call the request.getParameter() method.
* **getParameterValues()** − If a parameter appears more than once and it returns multiple values, call this method.
* **getParameterNames()** − This method is used if, in the current request, you want a complete list of all parameters.
* **getInputStream()** − This method is used for reading binary data streams from the client.