# **1. Spring MVC Introduction**

**🔹 What & Why**

**Spring MVC** is the web-framework module of the Spring Framework that implements the **Model–View–Controller** design pattern. It was introduced to provide a **clean separation of concerns** between request handling (controller), data (model), and presentation (view), while leveraging Spring’s dependency injection, AOP, and other core features.

* **Why it Came:**
  1. **Separation of Concerns:** Traditional servlet/JSP code mixed business logic and view rendering in the same classes (servlets or JSP scripts), making maintenance hard.
  2. **Testability:** By decoupling controllers, models, and views, components can be unit-tested in isolation.
  3. **Flexibility:** MVC pattern allows swapping view technologies (JSP, Thymeleaf, FreeMarker, PDF, Excel) without changing controller logic.
  4. **Leverage Spring:** Integrates seamlessly with Spring’s IoC, AOP, data access, transaction management, validation, and security modules.

**📌 Core Rules & Annotations**

| **Annotation** | **Purpose** |
| --- | --- |
| @Controller | Marks a class as a Spring MVC controller. |
| @RestController | @Controller + @ResponseBody; methods return payloads directly (JSON/XML). |
| @RequestMapping | Maps incoming URLs (and HTTP methods) to controller methods. |
| @GetMapping, etc. | Shorthand for @RequestMapping(method = GET), etc. |
| Model or ModelMap | Holds attributes to be passed from controller to view. |
| ModelAndView | Encapsulates both model data and logical view name. |
| @PathVariable | Binds path segments (e.g. /users/{id}) to method parameters. |
| @RequestParam | Binds query parameters (e.g. ?q=term) to method parameters. |
| @ExceptionHandler | Handles exceptions thrown in controller methods (per-controller). |
| @ControllerAdvice | Global exception handling and data binding advice across all controllers. |

**Configuration Essentials:**

* Register **DispatcherServlet** in web.xml (or auto-configured by Spring Boot).
* Define **ViewResolver** beans (e.g., InternalResourceViewResolver for JSP, ThymeleafViewResolver for Thymeleaf).
* Enable component scanning for controllers.

**📍 When & Where to Use**

* **Classic Server-Rendered Web Apps:** When you need to generate dynamic HTML (JSP, Thymeleaf, FreeMarker) on the server.
* **RESTful Services:** Using @RestController to serve JSON/XML to Single-Page Applications and mobile clients.
* **Form-Based Applications:** Handling multi-step form submissions, validation, data binding.
* **Hybrid Apps:** Mixed use of server-views and REST endpoints in the same application.

**✅ Real-World Examples**

1. **Online Banking Portal**
   * **Controllers** handle account summary pages, money-transfer forms, and transaction history.
   * **Views** rendered with Thymeleaf, showing account balances, transactions, charts.
   * **Validation** on forms (e.g., transfer amount > 0, recipient account exists).
2. **University Enrollment System**
   * **Controllers** for student dashboards, course registration pages, and transcript downloads.
   * **ModelAndView** lets the same controller return PDF, Excel, or HTML based on request.
   * **@ControllerAdvice** centralizes handling of EnrollmentException (e.g., seat full).
3. **E-Commerce Storefront**
   * **REST endpoints** (@RestController) for product catalogs consumed by an Angular front-end.
   * **Classic controllers** for server-rendered admin UI (inventory management, order processing).
   * **Interceptors** track user sessions, add cart data to every view automatically.

## **2. Spring MVC Architecture and Working**

**🔹 Architecture Overview**

At the heart of Spring MVC is the **DispatcherServlet**, which orchestrates the request processing through a chain of pluggable components.



**📌 Key Components & Flow**

1. **DispatcherServlet**
   * The **front controller**. All incoming HTTP requests for the Spring MVC application route through this servlet.
   * Initialized via web.xml or auto-configured in Spring Boot.
2. **HandlerMapping**
   * Determines which controller (handler) method should process the request, based on URL, HTTP method, headers, etc.
   * Common implementations: RequestMappingHandlerMapping.
3. **HandlerAdapter**
   * Invokes the chosen controller method, handling method signatures (parameters like @RequestParam, Model, HttpServletRequest, etc.).
   * Example: RequestMappingHandlerAdapter.
4. **Controller (Handler)**
   * Annotated with @Controller or @RestController.
   * Contains request-handling methods annotated with @RequestMapping, @GetMapping, etc.
   * Returns a **ModelAndView** (view name + model data) or directly writes to response (@ResponseBody).
5. **ModelAndView / Model**
   * **Model**: Data to be rendered (key-value pairs).
   * **ModelAndView**: Logical view name plus model data.
6. **ViewResolver**
   * Translates logical view names into actual View implementations.
   * Examples: InternalResourceViewResolver maps "home" → /WEB-INF/jsp/home.jsp; ThymeleafViewResolver resolves .html templates.
7. **View**
   * The actual template engine or JSP that renders HTML.
8. **HandlerInterceptors** *(Optional)*
   * Pre- and post-processing hooks around controller invocation (e.g., logging, auth checks).
   * Configured via WebMvcConfigurer#addInterceptors.
9. **Exception Resolvers**
   * Map thrown exceptions to error views or HTTP responses.
   * Controllers can use @ExceptionHandler; global handlers via @ControllerAdvice.

**📍 When & Where Each Part Runs**

* **DispatcherServlet**: on every request to your application’s context path.
* **HandlerMapping/Adapter**: immediately after servlet’s entry; selects and invokes the controller.
* **Controller**: business method executes, may call services/DAOs.
* **ViewResolver & View**: after controller returns; responsible for rendering.
* **Interceptors**: around controller; useful for cross-cutting concerns (authentication, metrics).
* **Exception Resolvers**: if any controller code throws, resolves to safe error response or view.

**✅ Real-World Examples of Architectural Usage**

1. **Multi-module Web Portal**
   * **Interceptor** logs request times and checks SSO tokens before reaching controllers.
   * **Multiple ViewResolvers**: one for Thymeleaf (public pages), another for FreeMarker (admin reports).
   * **Custom HandlerMapping**: URL versioning (e.g. v1/\*\*, v2/\*\*) directs to different controller packages.
2. **REST API with Content Negotiation**
   * **DispatcherServlet** delegates to RequestMappingHandlerAdapter, which requests Accept headers.
   * **MessageConverters** (Jackson, JAXB) serialize/deserialize JSON, XML.
   * **ExceptionHandler** methods produce standardized error payloads ({"error":"Not Found","status":404}).
3. **Dynamic Theming Application**
   * **Controller** selects view name based on user’s theme preference ("themeA/home" or "themeB/home").
   * **ViewResolver** chain: first tries theme-specific templates, then falls back to default.
   * **ModelAttribute** methods populate common data (menus, user info) for all views.

**🧩 Summary**

| **Stage** | **Component** | **Responsibility** |
| --- | --- | --- |
| **Entry Point** | DispatcherServlet | Central request routing. |
| **Routing** | HandlerMapping | Map URL → controller method. |
| **Invocation** | HandlerAdapter | Invoke method, resolve arguments. |
| **Business** | Controller | Handle request, populate model. |
| **Model Setup** | Model / ModelAndView | Store data to render. |
| **View Resolution** | ViewResolver | Map logical view name → actual template/JSP. |
| **Rendering** | View | Generate HTML (or other) output. |
| **Cross-Cutting** | Interceptors/Advice | Logging, security, exception handling, data binding. |

**Spring MVC Quiz**  
  
**1.** In Spring MVC, which annotation is used to mark a class as a controller?  
A) @Service  
B) @Component  
C) @Controller  
D) @RestController  
  
**Answer:** C) @Controller

**2.** Which of the following annotations is used to map a web request to a specific handler method?  
A) @PathVariable  
B) @RequestMapping  
C) @Autowired  
D) @Qualifier  
  
**Answer:** B) @RequestMapping

**3.** What is the role of the DispatcherServlet in Spring MVC?  
A) Handles database transactions  
B) Routes HTTP requests to appropriate controllers  
C) Manages application configuration  
D) Compiles JSP pages  
  
**Answer:** B) Routes HTTP requests to appropriate controllers

**4.** In Spring MVC, which object is used to pass data from a controller to a view?  
A) Model  
B) ResponseEntity  
C) Session  
D) ServletContext  
  
**Answer:** A) Model

**5.** Which annotation in Spring MVC is used to bind a method parameter to a URI template variable?  
A) @RequestParam  
B) @ModelAttribute  
C) @PathVariable  
D) @RequestBody  
  
**Answer:** C) @PathVariable

**6.** Which of the following is true about @RestController in Spring MVC?  
A) It combines @Controller and @ResponseBody  
B) It is used only for SOAP-based services  
C) It requires a separate @ResponseBody annotation for each method  
D) It cannot return JSON responses  
  
**Answer:** A) It combines @Controller and @ResponseBody

**7.** In Spring MVC, which view resolver is commonly used for JSP views?  
A) InternalResourceViewResolver  
B) BeanNameViewResolver  
C) TilesViewResolver  
D) XmlViewResolver  
  
**Answer:** A) InternalResourceViewResolver

**8.** What does the @RequestParam annotation do in Spring MVC?  
A) Maps HTTP request parameters to method parameters  
B) Maps URL path variables to method parameters  
C) Maps JSON payload to Java objects  
D) Maps HTTP headers to method parameters  
  
**Answer:** A) Maps HTTP request parameters to method parameters

**9.** Which of these is **NOT** a feature of Spring MVC?  
A) Loose coupling between view and controller  
B) Built-in REST API support  
C) Automatic database schema creation  
D) Support for multiple view technologies  
  
**Answer:** C) Automatic database schema creation

**10.** In Spring MVC, which file is typically used to configure the DispatcherServlet in a traditional XML setup?  
A) application.properties  
B) dispatcher-servlet.xml  
C) web.xml  
D) context.xml  
  
**Answer:** B) dispatcher-servlet.xml

# **1. Introduction to JSP(Java Server pages)**

**🔹 What & Why JSP Came**

* **What JSP is:**  
  A server-side view technology that lets you embed Java code (initially via scriptlets) and, more cleanly, use custom tags and Expression Language (EL) inside HTML pages.
* **Why it Came:**
  1. **Simplify Servlet View Generation:** Writing raw HTML in Java HttpServlet’s PrintWriter is painful. JSP lets designers write HTML directly with dynamic placeholders.
  2. **Separation of Concerns (Early):** While early JSPs still mixed logic & view, they paved the way to more MVC-friendly patterns—delegating business logic to servlets or controllers and using JSPs purely for rendering.
  3. **Rapid Page Development:** Non-Java web designers could quickly craft pages with minimal Java knowledge, thanks to JSP’s HTML-centric syntax.

**📌 Core Rules & Best Practices**

| **Rule** | **Description** |
| --- | --- |
| **Avoid Scriptlets** | Use <% … %> sparingly or not at all. Modern practice is to move logic to servlets or backing beans and use EL / JSTL instead. |
| **Use Directives Properly** | <%@ page %> for page settings (e.g. import, contentType), <%@ include %> for static includes, <%@ taglib %> for tag libraries. |
| **Prefer EL over <%= %>** | Expression Language (${user.name}) is cleaner and supports deferred evaluation, null safety, and simplified syntax. |
| **No Business Logic** | JSPs should only render data. All calculations, database access, and business rules belong in controllers/DAOs/services. |
| **Consistent Encoding** | Always set <%@ page contentType="text/html; charset=UTF-8" %> to avoid character-set issues. |

**📍 When & Where to Use JSP**

* **Server-Rendered HTML Pages:** Traditional multi-page web apps with form submissions.
* **View Layer in MVC Apps:** Paired with Spring MVC, Struts, or other frameworks where controllers populate model data.
* **Legacy Systems Maintenance:** Many enterprise apps still use JSP for their view layer.

**✅ Real-World JSP Examples**

1. **User Login Page**
   * **JSP:** <form action="login.do" method="post">…</form>
   * **EL:** Welcome, ${sessionScope.user.fullName}
   * **Directive:** <%@ include file="/WEB-INF/views/header.jsp" %> for common header.
2. **Product Listing**
   * **JSP Loop:**

<c:forEach var="p" items="${products}">

<div>${p.name}: $${p.price}</div>

</c:forEach>

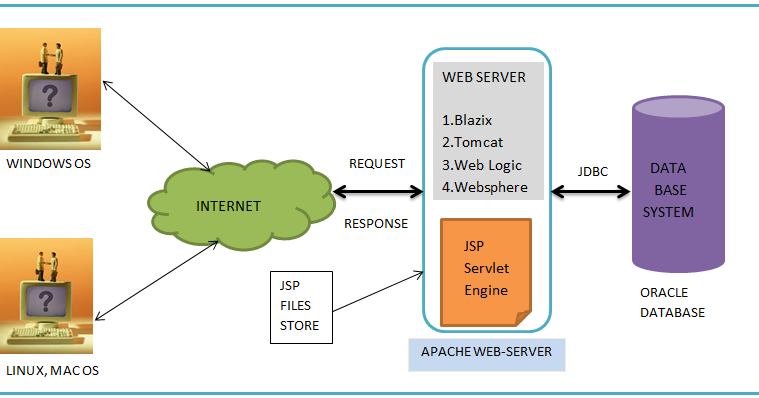
* + Renders an HTML grid of items retrieved from a servlet.

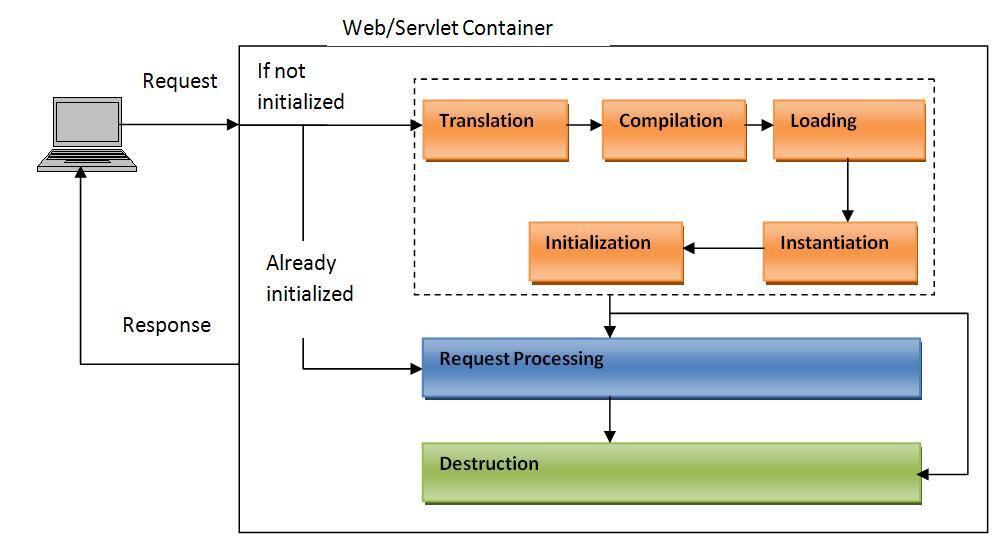
1. **Error Page with Internationalization**
   * **JSP Directive + EL:**

<%@ page isErrorPage="true" %>

<h1>Error: ${exception.message}</h1>

* + Displays a user-friendly message when exceptions bubble up.

  
  
  
**JSP Life Cycle**  
  
**Overview**

The **JSP life cycle** defines the various phases a JSP page goes through — from the time a request is made to the time a response is sent back to the client.  
The JSP lifecycle is **very similar** to the servlet lifecycle because a JSP is ultimately **converted into a servlet** by the container.  
  


**Main Phases of JSP Life Cycle**

A JSP page goes through **six major steps**:

**1. Translation Phase (JSP → Servlet)**

* When a JSP is requested **for the first time** (or modified), the **JSP container** translates it into a Java servlet source file.
* All HTML in the JSP becomes output statements in the servlet.
* All JSP tags and scriptlets become equivalent Java code.

**Example:**

<%= "Hello" %>

becomes

out.print("Hello");

**Key Point:** This step happens **only when the JSP changes**; not on every request.

**2. Compilation Phase (Java → .class)**

* The generated servlet .java file is compiled into a .class file.
* If compilation fails, the server sends an error to the browser.

**3. Loading Phase**

* The compiled servlet class is loaded into the **JVM** by the container's class loader.

**4. Instantiation Phase**

* The container creates an **instance** of the JSP’s servlet class.

**5. Initialization Phase**

* The JSP container calls the **jspInit()** method **once** in the JSP’s lifecycle.
* This is similar to a servlet’s init() method and is used for:
  + Resource initialization
  + Database connections
  + Configuration setup

public void jspInit() {

// Initialization code here

}

**6. Request Processing Phase**

* For **each client request**, the container:
  1. Creates **request** and **response** objects.
  2. Calls the **\_jspService(HttpServletRequest, HttpServletResponse)** method.
  3. This method:
     + Generates dynamic content (Java code from JSP)
     + Sends HTML output back to the browser
* This method is **auto-generated** and **cannot be overridden** by the developer.

**7. Destruction Phase**

* When the JSP is no longer needed, the container calls the **jspDestroy()** method **once** before removing it from service.
* Use this for:
  + Closing database connections
  + Releasing resources

public void jspDestroy() {

// Cleanup code here

}

**2. Introduction to JSTL Tags**

**🔹 What & Why JSTL Came**

* **What JSTL is: (JavaServer Pages Standard Tag Library)**  
  A standardized set of custom tags (and accompanying EL functions) for common tasks: iteration, conditionals, formatting, SQL, XML, and internationalization—so you don’t write Java or scriptlets in your JSP.
* **Why it Came:**
  1. **Eliminate Scriptlets:** Provide a tag-based way to do loops, conditionals, and formatting in JSP.
  2. **Standardization:** Before JSTL, every project rolled its own custom tags. JSTL unified this.
  3. **Cleaner JSPs & Reusability:** Tags encapsulate logic, making JSPs more maintainable and readable.

**📌 Core Rules & Best Practices**

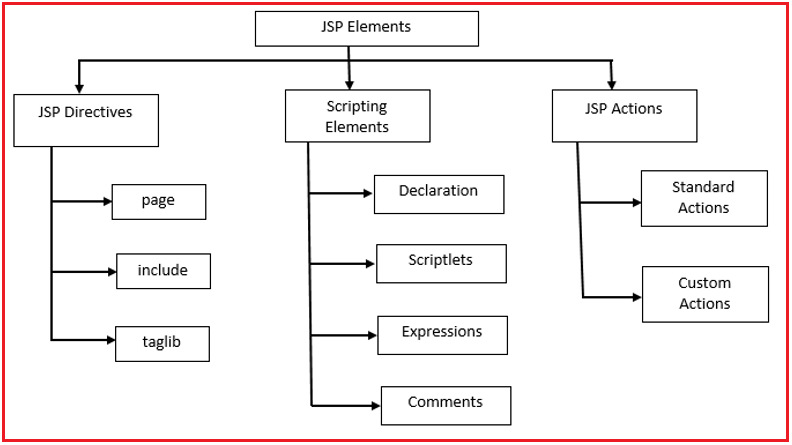
| **Rule** | **Description** |
| --- | --- |
| **Always Declare Taglibs** | At top of JSP: <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>. |
| **Limit Logic to JSTL/Core** | Use <c:if>, <c:forEach>, <c:choose> for flow control; avoid mixing with scriptlets. |
| **Use Formatting Tags** | <fmt:formatDate>, <fmt:formatNumber> for locale-aware output; <fmt:bundle> for message lookups. |
| **No DB Access in JSP** | Even though JSTL has <sql:query>, prefer DAO layers—JSTL SQL tags are mainly for quick demos or protos. |
| **Prefer EL Functions** | For string manipulation, use JSTL functions: |

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

${fn:toUpperCase(user.name)}

**📍 When & Where to Use JSTL**

* **Loops & Conditionals in JSP:** Iterate over collections (<c:forEach>) and make decisions (<c:if>, <c:choose>).
* **Locale-Aware Formatting:** Dates, numbers, currencies via the <fmt> taglib.
* **Simple Message Bundles:** <fmt:message key="welcome.message" /> for i18n.
* **Minimal Logic in Views:** Keep JSPs declarative.

  
  
**Main JSP Elements**

JSP elements define **how** we can embed Java code, configuration, and reusable components in a JSP page.  
There are **four main categories**:

1. **Directives**
2. **Scripting Elements**
3. **Standard Actions**
4. **Implicit Objects** (often considered separately, but used with scripting)

**1. JSP Directives**

Directives give **instructions to the JSP container** and affect the *overall structure* of the page, not just a single request.

**Syntax:**

<%@ directive attribute="value" %>

**Types of Directives:**

* **page directive** → Defines page-level settings (language, imports, error handling, etc.)
* <%@ page language="java" import="java.util.Date" contentType="text/html;charset=UTF-8" %>
* **include directive** → Includes another file at **translation time** (static include).
* <%@ include file="header.jsp" %>
* **taglib directive** → Declares a custom tag library (e.g., JSTL).
* <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

**2. JSP Scripting Elements**

These allow embedding **Java code** directly inside JSP.

**Types:**

* **Declaration** (<%! %>) → Declares variables or methods **once**, available to the page.
* <%! int counter = 0;
* public String greet() { return "Hello JSP!"; }
* %>
* **Scriptlet** (<% %>) → Contains Java statements that run **for each request**.
* <% counter++; %>
* <p>Page visit count: <%= counter %></p>
* **Expression** (<%= %>) → Outputs the result of an expression directly into HTML.
* <p>Current Time: <%= new java.util.Date() %></p>

**Note:** In modern JSP development, we try to avoid putting business logic here — we use JSTL and EL instead.

**3. JSP Standard Actions**

Standard actions are **predefined XML-style tags** that control the behavior of the JSP at request time.

**Common examples:**

* **<jsp:include>** → Includes a resource at **request time** (dynamic include).
* <jsp:include page="menu.jsp" />
* **<jsp:forward>** → Forwards request to another resource.
* <jsp:forward page="welcome.jsp" />
* **<jsp:useBean>** → Instantiates or accesses a JavaBean.
* <jsp:useBean id="user" class="com.example.User" scope="session" />
* <jsp:setProperty name="user" property="name" value="John" />
* <jsp:getProperty name="user" property="name" />

**4. JSP Implicit Objects**

JSP provides **9 built-in objects** that are automatically available without declaration.

| **Object** | **Type** | **Purpose** |
| --- | --- | --- |
| request | HttpServletRequest | Request data |
| response | HttpServletResponse | Response control |
| session | HttpSession | Session tracking |
| application | ServletContext | Application-wide data |
| out | JspWriter | Output to client |
| config | ServletConfig | Servlet config |
| page | Object | Current JSP page |
| pageContext | PageContext | Page attributes |
| exception | Throwable | Error handling |

Example:

<p>Client IP: <%= request.getRemoteAddr() %></p>

**Putting It All Together – Example JSP**

<%@ page language="java" import="java.util.Date" %>

<%! int visitCount = 0; %>

<%

visitCount++;

%>

<html>

<body>

<h2>Welcome to My JSP Page</h2>

<p>Current Time: <%= new Date() %></p>

<p>Visit Count: <%= visitCount %></p>

<jsp:include page="footer.jsp" />

</body>

</html>

**Summary Table**

| **Element Type** | **Syntax Example** | **Purpose** |
| --- | --- | --- |
| Directive | <%@ page import="java.util.\*" %> | Page-level instructions |
| Declaration | <%! int x = 0; %> | Declare vars/methods |
| Scriptlet | <% x++; %> | Java code per request |
| Expression | <%= x %> | Output value |
| Action | <jsp:include page="file.jsp"/> | Predefined tasks |
| Implicit Objects | request.getParameter("name") | Built-in objects |

**✅ Real-World JSTL Examples**

1. **Displaying Search Results**

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

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

<div>${item.title} — ${item.snippet}</div>

</c:forEach>

1. **Conditional Navigation Menu**

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

<c:choose>

<c:when test="${user.admin}">

<a href="admin/dashboard">Admin Dashboard</a>

</c:when>

<c:otherwise>

<a href="user/profile">My Profile</a>

</c:otherwise>

</c:choose>

1. **Locale-Aware Date & Number Formatting**

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

<fmt:formatDate value="${order.date}" pattern="dd MMM yyyy" />

<fmt:formatNumber value="${order.total}" type="currency" currencyCode="${user.currency}" />

**🔗 Summary Comparison**

| **Aspect** | **JSP Only (No JSTL)** | **JSP + JSTL** |
| --- | --- | --- |
| Flow Control | Scriptlets <% for(...) { %> | <c:forEach> <c:if> <c:choose> |
| Expression Syntax | <%= user.getName() %> | ${user.name} |
| Formatting | Manual in scriptlet or bean calls | <fmt:formatDate>, <fmt:formatNumber> |
| Internationalization | Custom taglibs or scriptlets | <fmt:bundle> + <fmt:message> |

## **1. Core Tags (<c:…>)**

**Purpose:** Iteration, conditionals, URL construction, import, redirection, and more  
**Taglib Declaration:**

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

**Rules & When to Use:**

* Use <c:forEach> instead of scriptlet loops.
* Use <c:if> and <c:choose> for branching logic.
* Keep all business logic in the controller; JSTL core tags are for view-level flow control only.

**Real-World Examples**

1. **Display a List of Products**

<c:forEach var="prod" items="${products}">

<div>${prod.name} — $${prod.price}</div>

</c:forEach>

*Renders each product returned by the controller.*

1. **Show “Admin” Menu Only for Admin Users**

<c:if test="${sessionScope.user.role == 'ADMIN'}">

<a href="/admin/dashboard">Admin Dashboard</a>

</c:if>

*Conditionally renders admin links.*

1. **Choose Message Based on Order Status**

<c:choose>

<c:when test="${order.status == 'SHIPPED'}">

Your order has shipped!

</c:when>

<c:otherwise>

Order is being processed.

</c:otherwise>

</c:choose>

*Selects one of multiple possible messages.*

**2. Formatting Tags (<fmt:…>)**

**Purpose:** Internationalization (i18n) and locale-aware formatting of dates, numbers, and messages  
**Taglib Declaration:**

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

**Rules & When to Use:**

* Always set a locale (via <fmt:setLocale> or Accept-Language).
* Use <fmt:message> for externalized message bundles, not hard-coded strings.
* Use formatting tags instead of Java API calls in JSP.

**Real-World Examples**

1. **Locale-Aware Date Display**

<fmt:setLocale value="${param.lang != null ? param.lang : 'en\_US'}"/>

<fmt:formatDate value="${order.date}" pattern="dd MMM yyyy"/>

*Displays order date in user’s locale.*

1. **Currency Formatting**

<fmt:formatNumber value="${prod.price}" type="currency" currencyCode="${user.currency}"/>

*Formats a price according to the user’s preferred currency.*

1. **Message Bundle Lookup**

<fmt:bundle basename="messages">

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

</fmt:bundle>

*Retrieves “welcome.user” from messages.properties.*

**3. Function Tags (<fn:…>)**

**Purpose:** String and array utilities for use in EL expressions  
**Taglib Declaration:**

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

**Rules & When to Use:**

* Only for simple operations (length, contains, substring).
* Avoid heavy logic—use functions sparingly in the view.

**Real-World Examples**

1. **Check if a String Contains a Substring**

<c:if test="${fn:contains(user.email, '@gmail.com')}">

<p>Using Gmail account</p>

</c:if>

1. **Limit Display of a Long Description**

<c:out value="${fn:length(article.body) > 200

? fn:substring(article.body, 0, 200) + '...'

: article.body}" />

1. **Join Array Elements for Display**

<c:out value="${fn:join(order.tags, ', ')}" />

*(If you provide a custom join function via your own taglib.)*

**4. SQL Tags (<sql:…>) (Use sparingly!)**

**Purpose:** Quick demo-level database access from JSP without a DAO layer  
**Taglib Declaration:**

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

**Rules & When to Use:**

* **Avoid** in production—bypass JSTL SQL tags in favor of a DAO/service layer.
* Use only for simple demos or prototyping.
* Always parameterize queries to prevent SQL injection (<sql:param>).

**Real-World Examples**

1. **Query All Users (Demo Only)**

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

url="jdbc:mysql://localhost/shop" user="root" password="pass"/>

<sql:query dataSource="${db}" var="users">

SELECT \* FROM users

</sql:query>

<c:forEach var="u" items="${users.rows}">

${u.username} — ${u.email}<br/>

</c:forEach>

1. **Parameterized Search**

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

SELECT \* FROM products WHERE name LIKE ?

<sql:param value="${param.q}%" />

</sql:query>

1. **Update Operation**

<sql:update dataSource="${db}">

UPDATE orders SET status='SHIPPED' WHERE id=?

<sql:param value="${order.id}" />

</sql:update>

<p>${result} rows updated.</p>

**5. XML Tags (<x:…>)**

**Purpose:** Simplified XML data parsing and iteration inside JSP  
**Taglib Declaration:**

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

**Rules & When to Use:**

* Use only when your view needs to process XML payloads (e.g., SOAP responses).
* Better moved to a controller or service layer in production.

**Real-World Examples**

1. **Parse and Display RSS Feed Items**

<x:parse var="rss" doc="${param.feedUrl}" />

<x:forEach select="$rss/rss/channel/item" var="item">

<h3><x:out select="$item/title"/></h3>

<p><x:out select="$item/description"/></p>

</x:forEach>

1. **XSLT Transformation**

<x:transform var="html" xml="${xmlDoc}" xslt="${pageContext.servletContext.getRealPath('/WEB-INF/xslt/template.xsl')}" />

<c:out value="${html}" escapeXml="false"/>

1. **Extracting Configuration from an XML File**

<x:parse var="config" doc="/WEB-INF/config.xml"/>

<c:out value="<x:out select='$config/config/logging/level'/>"/>

**Summary Table**

| **Tag Library** | **Prefix** | **Use Case** | **Production Suitability** |
| --- | --- | --- | --- |
| Core | c | Loops, conditionals, URL, import/redirect | ✔️ Recommended |
| Formatting | fmt | i18n, date/number formatting, bundles | ✔️ Recommended |
| Functions | fn | String/array utilities | ✔️ Recommended |
| SQL | sql | Quick DB prototyping | ❌ Discouraged |
| XML | x | XML parsing/transform in view | ❌ Discouraged |

## **JSP Quiz**

**1.** JSP stands for:  
A) Java Standard Pages  
B) Java Servlet Pages  
C) JavaServer Pages  
D) Java Script Pages  
  
**Answer:** C) JavaServer Pages

**2.** Which directive in JSP is used to include other files during page translation?  
A) <%@ page %>  
B) <%@ include %>  
C) <% include %>  
D) <%@ taglib %>  
  
**Answer:** B) <%@ include %>

**3.** Which of the following is a *JSP implicit object*?  
A) request  
B) thread  
C) main  
D) console  
  
**Answer:** A) request

**4.** What is the default scripting language used in JSP?  
A) Python  
B) Java  
C) JavaScript  
D) PHP  
  
**Answer:** B) Java

**5.** Which JSP tag is used to insert Java code into a JSP?  
A) <% %>  
B) <%! %>  
C) <%= %>  
D) All of the above  
  
**Answer:** D) All of the above

**6.** In JSP, the <%= %> syntax is used for:  
A) Declaring variables  
B) Writing comments  
C) Outputting the value of an expression  
D) Importing packages  
  
**Answer:** C) Outputting the value of an expression

**7.** Which JSP directive is used to define page-level settings like content type and buffering?  
A) <%@ page %>  
B) <%@ taglib %>  
C) <%@ include %>  
D) <%! page %>  
  
**Answer:** A) <%@ page %>

**8.** Which of these is NOT an implicit object in JSP?  
A) request  
B) response  
C) session  
D) connection  
  
**Answer:** D) connection

**9.** In JSP, which scope means the object is available until the web application is stopped?  
A) request  
B) session  
C) application  
D) page  
  
**Answer:** C) application

**10.** Which method is called by the container to initialize a JSP page?  
A) jspInit()  
B) init()  
C) \_jspInit()  
D) service()  
  
**Answer:** A) jspInit()

**11.** The JSP expression language (EL) syntax uses which symbols?  
A) ${ }  
B) <# #>  
C) <% %>  
D) << >>  
  
**Answer:** A) ${ }

**12.** Which JSP action tag is used to forward a request to another resource?  
A) jsp:include  
B) jsp:useBean  
C) jsp:forward  
D) jsp:param  
  
**Answer:** C) jsp:forward

**13.** In JSP, the session implicit object is an instance of which class?  
A) HttpSession  
B) SessionContext  
C) JSPsession  
D) ServletSession  
  
**Answer:** A) HttpSession

**14.** Which directive is used to import Java classes in JSP?  
A) <%@ taglib %>  
B) <%@ page import="..." %>  
C) <% import ... %>  
D) <%@ import %>  
  
**Answer:** B) <%@ page import="..." %>

**15.** Which statement about JSP is correct?  
A) JSP runs on the client browser directly.  
B) JSP is converted into a servlet before execution.  
C) JSP cannot use Java code.  
D) JSP cannot access databases.  
  
**Answer:** B) JSP is converted into a servlet before execution.

## **Creating a Simple MVC Application using Spring MVC and JSP**

Building a web application with Spring MVC and JSP lets you cleanly separate **three concerns**—Model (data), View (presentation), and Controller (request handling)—while leveraging Spring’s powerful features (dependency injection, AOP, validation, etc.). Below is a deep dive into why this approach exists, the key rules and annotations you need, when/where to apply it, and three real-world scenarios.

**1. Why It Came**

* **Tangled Code in Early Java Web Apps**  
  Before MVC frameworks, servlets and JSPs often mixed business logic, HTML generation, and database calls in the same class or page, making maintenance and testing difficult.
* **Need for Separation of Concerns**  
  The Model–View–Controller pattern cleanly divides responsibilities:
  + **Model** holds application data (Java beans, DTOs)
  + **View** renders HTML (JSP + JSTL/EL)
  + **Controller** responds to requests and invokes services
* **Leverage Spring Ecosystem**  
  Spring MVC integrates seamlessly with Spring’s IoC container, data access, validation, security, and other modules—so your web layer can reuse existing services and DAOs without custom wiring.

**2. Core Rules & Annotations**

| **Concept / Annotation** | **Purpose** |
| --- | --- |
| **DispatcherServlet** | Front controller servlet; intercepts all web requests and delegates to handlers. |
| **@Controller** | Marks a class as a web controller containing request-handling methods. |
| **@RequestMapping**, **@GetMapping**, **@PostMapping** | Map HTTP paths and methods to controller methods |

**Configuration Essentials**

* **Spring XML or JavaConfig**: Register DispatcherServlet, enable component scanning, configure a ViewResolver bean.
* **Directory Structure**:

src/main/java/.../controller/MyController.java

src/main/webapp/WEB-INF/views/home.jsp

**3. When & Where to Use**

* **Server-Rendered HTML Applications**  
  When your application’s primary interface is HTML pages (e.g., corporate intranet, admin panels).
* **Form-Based Workflows**  
  For multi-step forms, complex validations, file uploads—Spring MVC’s data binding and validation shine here.
* **Hybrid Apps**  
  When you need both server-rendered pages and REST endpoints in the same codebase (use @Controller for JSP views, @RestController for JSON).
* **Legacy Modernization**  
  If you’re migrating a classic JSP/Servlet app, Spring MVC provides a familiar paradigm while adding powerful features.

**4. Real-World Examples**

**Example 1: Employee Directory Portal**

* **Controller**

@Controller

public class EmployeeController {

@GetMapping("/employees")

public String list(Model model) {

List<Employee> list = employeeService.findAll();

model.addAttribute("employees", list);

return "employeeList";

}

}

* **View (employeeList.jsp)**

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

<html><body>

<h2>Employees</h2>

<table>

<c:forEach var="e" items="${employees}">

<tr>

<td>${e.id}</td><td>${e.name}</td><td>${e.department}</td>

</tr>

</c:forEach>

</table>

</body></html>

* **Use Case:** Internal HR system to browse and search employee information.

**Example 2: Product Catalog & Search (E-Commerce)**

* **Controller**

@Controller

public class ProductController {

@GetMapping("/products")

public String showAll(Model m) {

m.addAttribute("products", productService.findAll());

return "productList";

}

@PostMapping("/products/search")

public String search(@RequestParam String q, Model m) {

m.addAttribute("products", productService.searchByName(q));

return "productList";

}

}

* **View (productList.jsp)**

<form method="post" action="products/search">

<input name="q"/><button>Search</button>

</form>

<c:forEach var="p" items="${products}">

<div>${p.name} — $${p.price}</div>

</c:forEach>

* **Use Case:** Retail storefront allowing customers to browse and filter products.

**Example 3: Student Enrollment Form (University Portal)**

* **Controller**

@Controller

public class EnrollmentController {

@GetMapping("/enroll")

public String showForm(Model m) {

m.addAttribute("student", new StudentDTO());

return "enrollForm";

}

@PostMapping("/enroll")

public String submit(@Valid @ModelAttribute("student") StudentDTO s, BindingResult br) {

if (br.hasErrors()) return "enrollForm";

enrollmentService.enroll(s);

return "redirect:/enroll/success";

}

}

* **View (enrollForm.jsp)**

<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

<html><body>

<form:form method="post" modelAttribute="student">

<form:label path="name">Name:</form:label>

<form:input path="name"/>

<form:errors path="name"/>

<!-- more fields... -->

<button>Submit</button>

</form:form>

</body></html>

* **Use Case:** University site where students register for courses with server-side validation.

**5. Best Practices**

1. **Keep Controllers Thin**  
   Delegate all business logic to @Service beans; controllers should only orchestrate request → service → view.
2. **Use DTOs for Binding**  
   Avoid exposing JPA entities directly; use simple POJOs or DTOs annotated with @Valid for form data.
3. **Organize Views Consistently**  
   Store JSP files under /WEB-INF/views/ and configure a single InternalResourceViewResolver with a clear prefix/suffix.
4. **Leverage @ControllerAdvice**  
   Centralize exception handling and common model attributes (e.g., user details, navigation menus).
5. **Internationalization**  
   Use JSTL <fmt:message> and property bundles for all user-facing text.