Unit 3

JSP architecture, JSP page life cycle, JSP Directives, JSP scripting elements, JSP Actions, Error handling in JSP, Session tracking techniques in JSP

What is JSP?

- Mostly HTML page, with extension .jsp
- Include JSP tags to enable dynamic content creation
- Translation: JSP → Servlet class
- Compiled at Request time
  (first request, a little slow)
- Execution: Request → JSP Servlet's service method

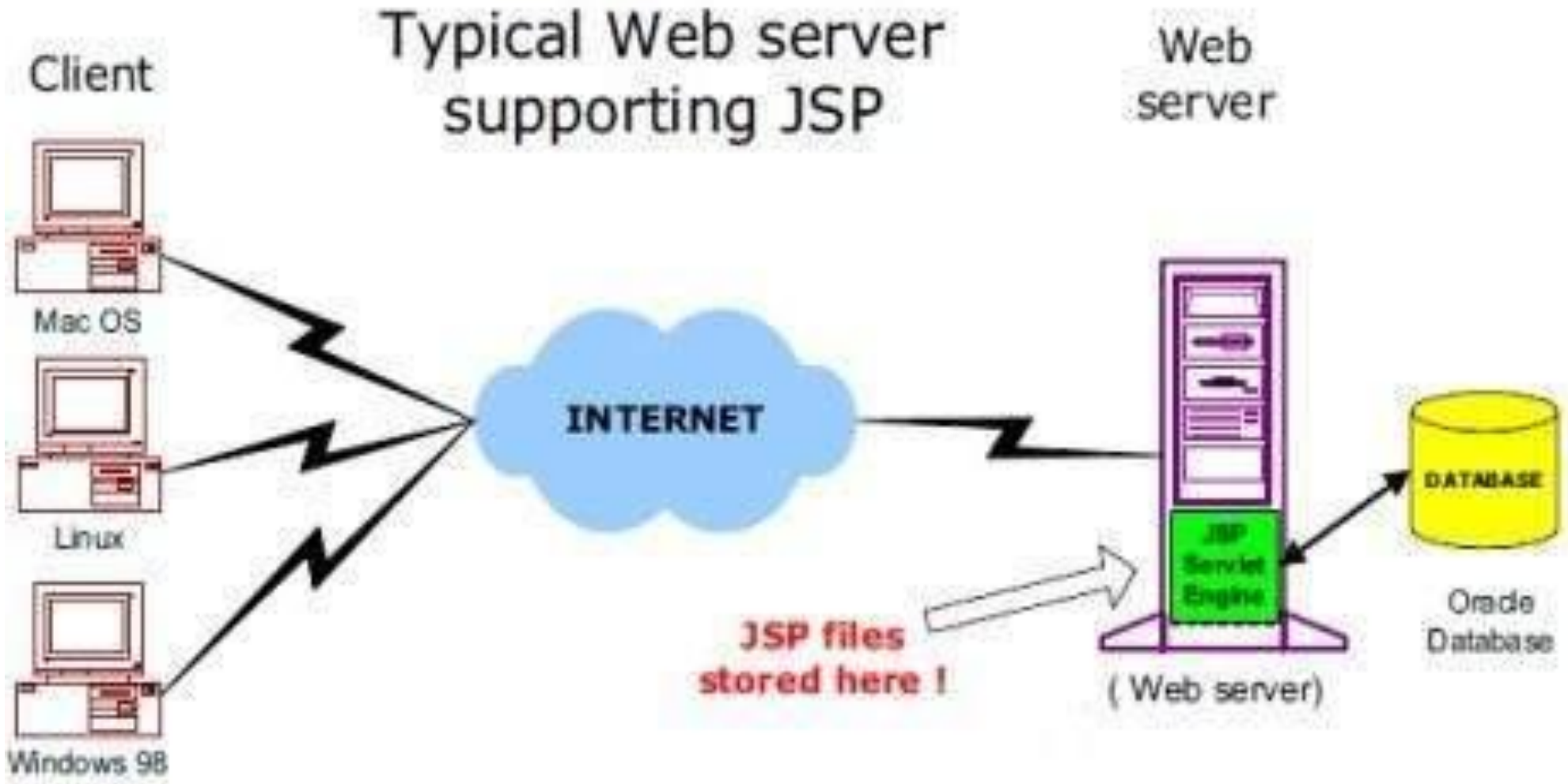
JSP/ASP/PHP vs CGI/Servlets

- CGI & Servlets -- Mostly Code with some HTML via print & out.println
- JSP/ASP/PHP -- Mostly HTML, with code snippets thrown in
  - No explicit recompile
  - Great for small problems
  - Easier to program
  - Not for large computations

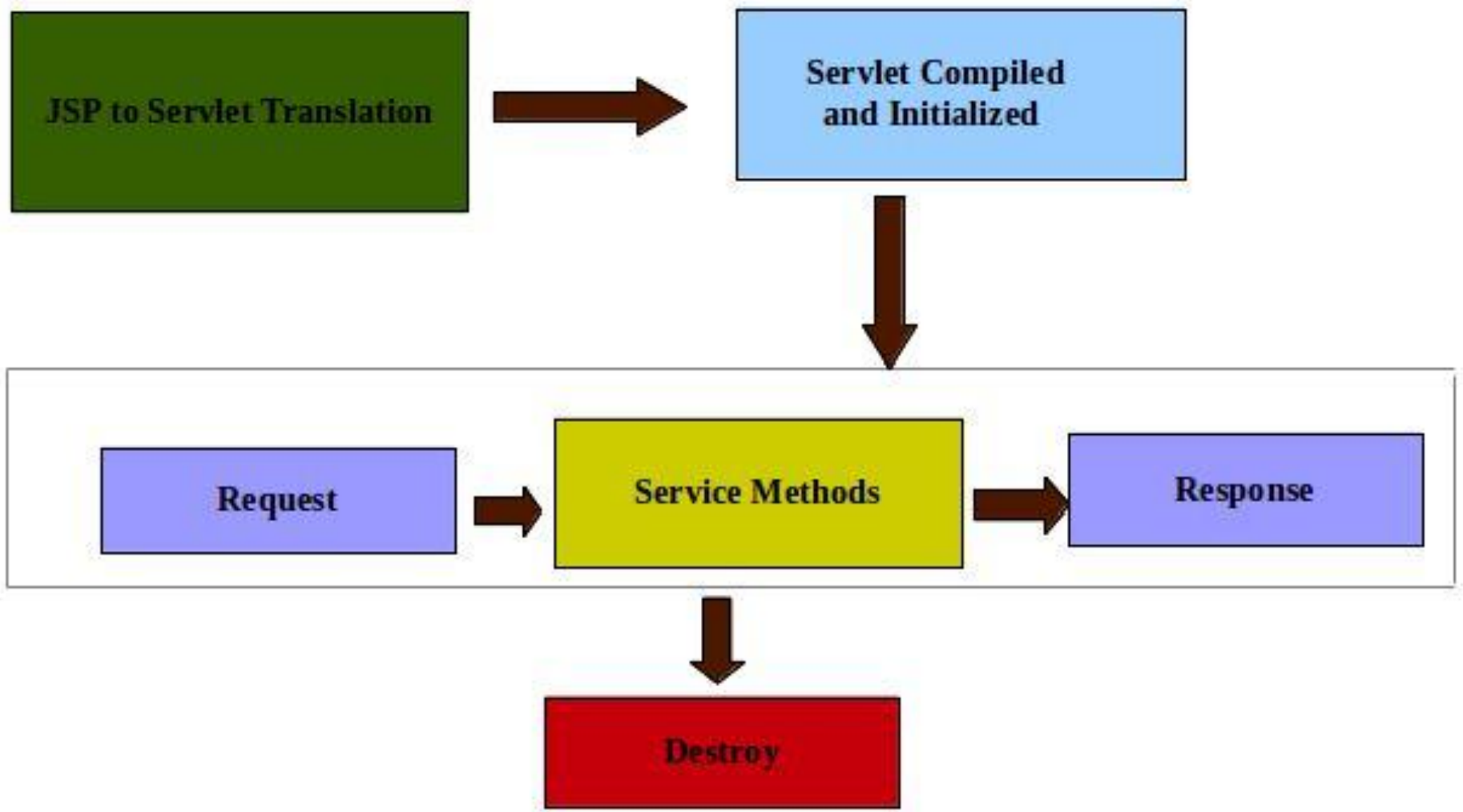
Advantages

- Code -- Computation
- HTML -- Presentation
- Separation of Roles
  - Developers
  - Content Authors/Graphic Designers/Web Masters
  - Supposed to be cheaper... but not really...

JSP Architecture

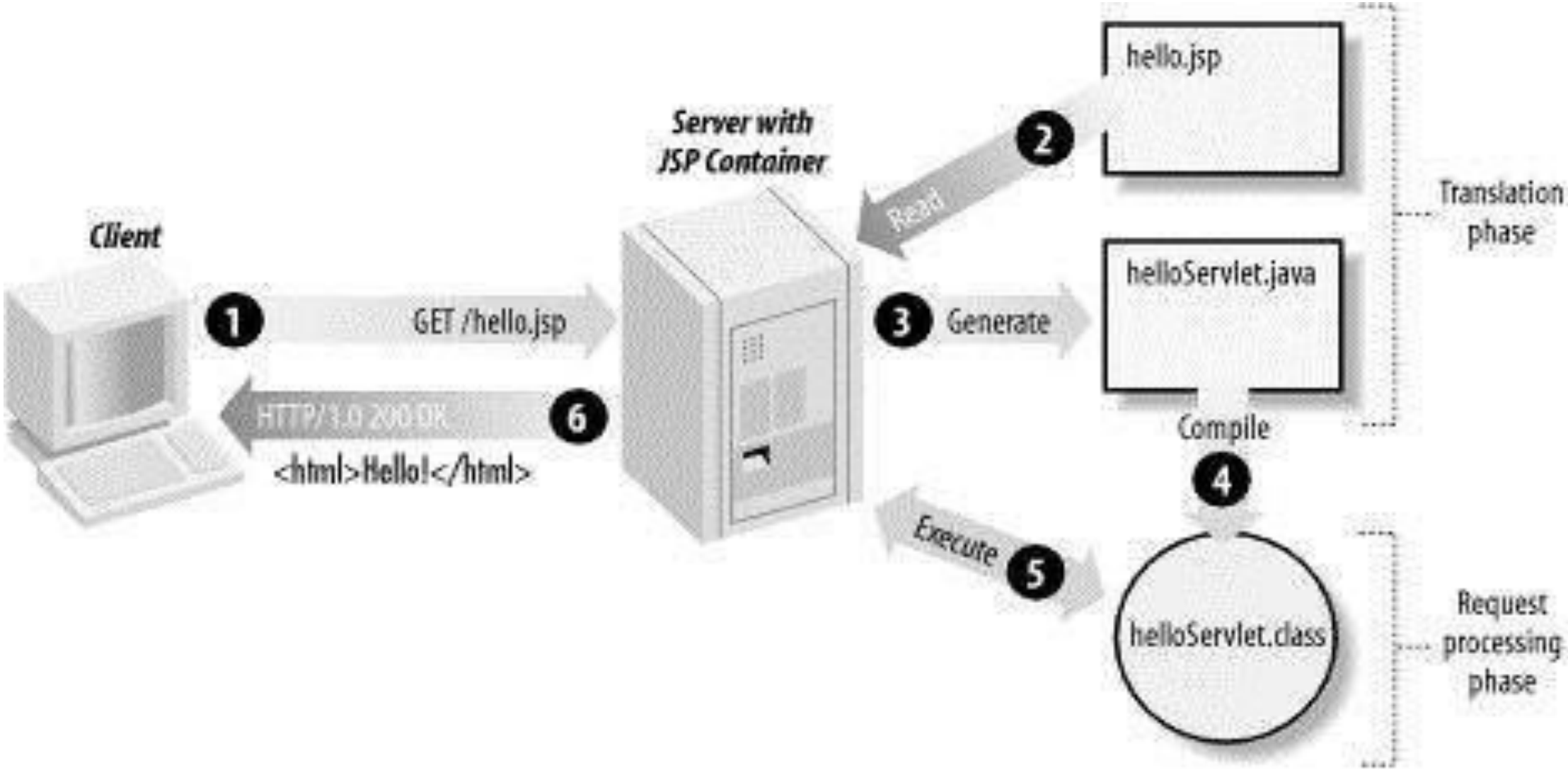


JSP life cycle



JSP Processing

- As with a normal page, **your browser sends an HTTP request to the web server**.
- The web server recognizes that the HTTP request is for a JSP page **and forwards it to a JSP engine**. This is done by using the URL or JSP page which ends with .jsp instead of .html.
- **The JSP engine loads the JSP page from disk and converts it into a servlet content**. This conversion is very simple **in which all template text is converted to println( ) statements and all JSP elements are converted to Java code**. This code implements the corresponding dynamic behavior of the page.
- **The JSP engine compiles the servlet into an executable class** and forwards the original request to a servlet engine.
- A part of the **web server called the servlet engine loads the Servlet class and executes it**. **During execution, the servlet produces an output in HTML format**. The output is further passed on to the web server by the servlet engine inside an HTTP response.
- The **web server forwards the HTTP response to your browser in terms of static HTML content.**
- Finally, the **web browser handles the dynamically-generated HTML page inside the HTTP response exactly as if it were a static page**.


Client
Server with
JSP Container
hello.jsp
Read
2
Translation
phase
helloServlet.java
1
GET /hello.jsp
3
Generate
HTTP/1.0 200 OK
6
<html>Hello!</html>
Compile
4
Execute
5
helloServlet.class
Request
processing
phase

JSP container operational phases

- **Translation time** Generating the Java servlet source code from a .jsp file
- **Request time** Invoking the servlet to handle an HTTP request

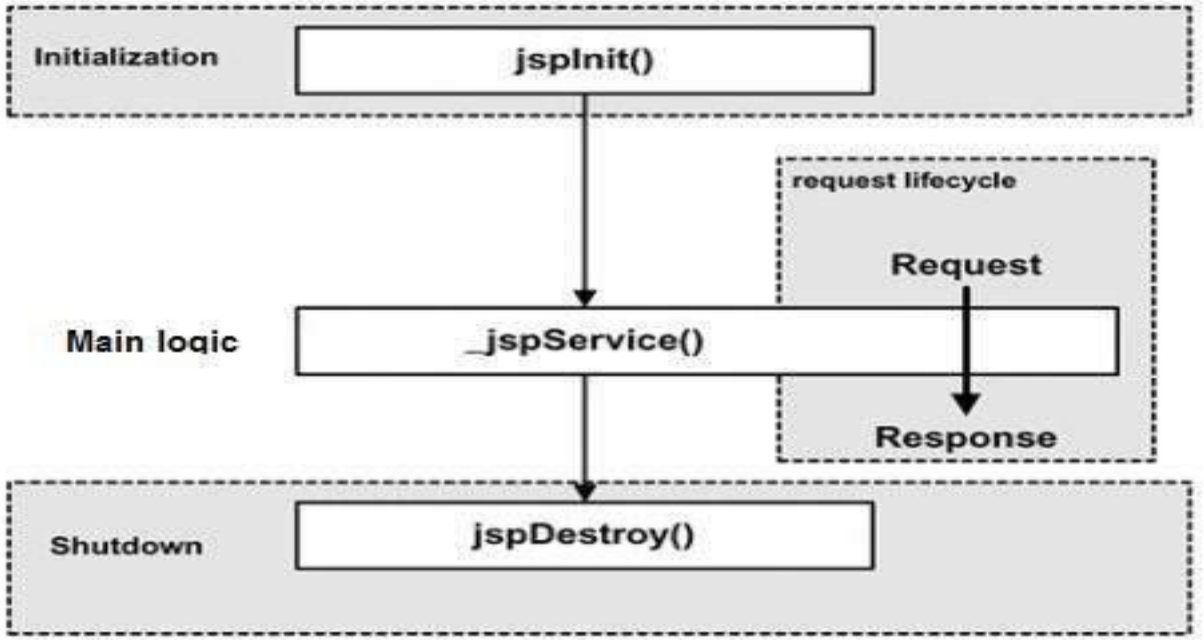
The following are the paths followed by a JSP −

- Compilation
- Initialization
- Execution
- Cleanup

Paths followed by a JSP

The following are the paths followed by a JSP

1. Compilation
    1. Parsing the JSP.
    2. Turning the JSP into a servlet.
    3. Compiling the servlet.
2. Initialization: **jspInit()**
3. Execution: **_jspService()**
4. Cleanup: **jspDestroy()**



1. JSP Compilation

- When a browser asks for a JSP, **the JSP engine first checks to see whether it needs to compile the page**. If the page has **never been compiled**, or if the JSP **has been modified** since it was last compiled, the **JSP engine compiles the page**.
- The compilation process involves three steps −
  - Parsing the JSP.
  - Turning the JSP into a servlet.
  - Compiling the servlet.

2. JSP Initialization

- When a **container loads a JSP it invokes the jspInit()** method before servicing any requests. If you need to perform JSP-specific initialization, override the jspInit() method −

```
public void jspInit(){
  // Initialization code...}
```

Typically, **initialization is performed only once** and as with the servlet init method, you generally initialize database connections, open files, and create lookup tables in the jspInit method.

3. JSP Execution

- This phase of the **JSP life cycle represents all interactions with requests** until the JSP is destroyed.
- Whenever **a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the _jspService()** method in the JSP.
- The _jspService() method takes an HttpServletRequest and an HttpServletResponse as its parameters as follows −

```
void _jspService(HttpServletRequest request, HttpServletResponse response) {
  // Service handling code...
}
```

The _jspService() method of a JSP is invoked on request basis. This is responsible for **generating the response for that request** and this method is also responsible for generating responses to all seven of the HTTP methods, i.e, GET, POST, DELETE, etc.

4. JSP Cleanup

- The **destruction phase of the JSP life cycle** represents when a **JSP is being removed from use by a container**. The jspDestroy() method is the JSP equivalent of the destroy method for servlets. **Override jspDestroy when you need to perform any cleanup, such as releasing database connections or closing open files**. The jspDestroy() method has the following form –

```
public void jspDestroy() {
  // Your cleanup code goes here.
}
```

JSP Declarations

- A declaration declares one or more variables or methods that you can use in Java code later in the JSP file. You must declare the variable or method before you use it in the JSP file.
- <%! int i = 0; %>
- <%! int a, b, c; %>
- <%! Circle a = new Circle(2.0); %>

JSP Expression

- A JSP expression element **contains a scripting language expression** that is **evaluated, converted to a String, and inserted** where the expression appears in the JSP file.
- Because the value of an expression is converted to a String, you can use an expression within a line of text, whether or not it is tagged with HTML, in a JSP file.
- The expression element can contain any expression that is valid according to the Java Language Specification but you cannot use a semicolon to end an expression.

JSP Comments

- JSP comment marks text or statements that the JSP container should ignore. A JSP comment is useful when you want to hide or "comment out", a part of your JSP page.
- <%-- This comment will not be visible in the page source --%>

JSP Documents

- JSP docs are XHTML Documents containing:
  - Fixed-Template Data: *FTD*
    - *HTML Components*
    - *XML markup*
  - JSP Components: <*JSP*>

Components of a JSP Page

- JSP elements: are instructions to the **JSP container about what code to generate and how it should operate**
- fixed template data: have **specific start and end tags** that identify them to the JSP compiler. Template data is everything else that is not recognized by the JSP container. Template data (usually HTML) is passed through unmodified, so the HTML that is ultimately generated contains the template data exactly as it was coded in the .jsp file.
- any combination of the two

JSP Objects

- Request: This is the HttpServletRequest object associated with the request.
- Response : This is the HttpServletResponse object associated with the response to the client.
- Out: This is the PrintWriter object used to send output to the client.
- Session: This is the HttpSession object associated with the request.
- Application: This is the ServletContext object associated with the application context.
- Config: This is the ServletConfig object associated with the page.
- pageContext: This encapsulates use of server-specific features like higher performance JspWriters.
- Page: This is simply a synonym for this, and is used to call the methods defined by the translated servlet class.
- Exception: The Exception object allows the exception data to be accessed by designated JSP.

JSP Actions

- JSP actions use constructs in XML syntax to control the behavior of the servlet engine. Unlike directives, actions are re-evaluated each time the page is accessed.
- You can dynamically insert a file, reuse JavaBeans components, forward the user to another page, or generate HTML for the Java plugin.

jsp:include	Includes a file at the time the page is requested.
jsp:useBean	Finds or instantiates a JavaBean.
jsp:setProperty	Sets the property of a JavaBean.
jsp:getProperty	Inserts the property of a JavaBean into the output.
jsp:forward	Forwards the requester to a new page.
jsp:plugin	Generates browser-specific code that makes an OBJECT or EMBED tag for the Java plugin.
jsp:element	Defines XML elements dynamically.
jsp:attribute	Defines dynamically-defined XML element's attribute.
jsp:body	Defines dynamically-defined XML element's body.
jsp:text	Used to write template text in JSP pages and documents.

Common Attributes

- Id attribute: The id attribute **uniquely identifies the Action element**, and **allows the action to be referenced inside the JSP page**. If the Action creates an instance of an object, the id value can be used to reference it through the implicit object PageContext.
- Scope attribute: This attribute **identifies the lifecycle of the Action element**. The id attribute and the scope attribute are directly related, as the scope attribute **determines the lifespan of the object associated with the id**. The scope attribute has four possible values: **(a) page, (b)request, (c)session**, and **(d) application**.

JSP Directives

- JSP directives are **the elements of a JSP source code that guide the web container on how to translate the JSP page** into it's respective servlet.
- Syntax : <%@ directive attribute = "value"%>
- Directives can have a number of attributes which you can list down as key-value pairs and separated by commas. The blanks between the @ symbol and the directive name, and between the last attribute and the closing %>, are optional.

	Directive & Description
1	**<%@ page ... %>** Defines page-dependent attributes, such as scripting language, error page, and buffering requirements.
2	**<%@ include ... %>** Includes a file during the translation phase.
3	**<%@ taglib ... %>** Declares a tag library, containing custom actions, used in the page

Error Handling in JSP

- **Checked exceptions**: A checked exception is an exception that is typically a **user error or a problem that cannot be foreseen by the programmer**. For example, if a file is to be opened, but the file cannot be found, an exception occurs. These exceptions cannot simply be ignored at the time of compilation.
- **Runtime exceptions**: A runtime exception is an exception that probably could have been avoided by the programmer. As opposed to the checked exceptions**, runtime exceptions are ignored at the time of compilation**.
- Errors: These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. **Errors are typically ignored in your code because you can rarely do anything about an error**. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

JSP Session Tracking Techniques

While using a web application, a client may trigger multiple requests, or a server might be handling multiple requests from different clients at a given point of time.
At times like these, a need for Session tracking arises so that the server identifies a client. It is an efficient way to maintain the state of requests that originate from the same browser for a given period of time.
In order to do so, a unique session id is assigned to a user for the given session.

Need for Session Tracking in JSP

- HTTP is a stateless protocol that implies that whenever a client makes a request, then for each request a new connection to the web server is established. As new connections are made every time, it becomes difficult for servers to identify that requests are coming from the same user or not. In addition to this, the server doesn’t keep the information of the previous request, if a new request is made.
- For example, a client is using a shopping web application; a client may keep adding items to the cart. A server should know that if it is the same client or not. In scenarios like these session tracking is required.
- To solve such problems, whenever a user introduces itself to the server, it should possess and provide a unique identifier. This will help to maintain a complete conversation between client and server. There are four methods available to maintain a session between server and client.

Maintaining Session Between Web Client And Server

1) Cookies: A webserver can assign a unique session ID as a cookie to each web client and for subsequent requests from the client they can be recognized using the received cookie. This may not be an effective way as the browser at times does not support a cookie. It is not recommended to use this procedure to maintain the sessions.

- There are two types of cookies:
  - **Session cookies** - are temporary cookies and are deleted as soon as user closes the browser. The next time user visits the same website, server will treat it as a new client as cookies are already deleted.
  - **Persistent cookies** - remains on hard drive until we delete them or they expire.
- If cookie is associated with the client request, server will associate it with corresponding user session otherwise will create a new unique cookie and send back with response.

Advantage: A user can be identified using this method.
Disadvantage: This method is not recommended for session tracking as it fails for the users that keep their cookies blocked as no information can be saved on the client's side.

```
Cookie myCookie = new Cookie("SessionID", "Session Value");
myCookie.setMaxAge(60*60*24)
Response.addCookie(myCookie);
```

Maintaining Session Between Web Client And Server

2) Hidden Form Fields:

- Hidden fields are similar to other input fields with the only difference is that these fields are not displayed on the page but its value is sent as other input fields. For example

  <input type=”hidden” name=”sessionId” value=”unique value”/>

- is a hidden form field which will not displayed to the user but its value will be send to the server and can be retrieved using request.getParameter(“sessionId”) .
- With this approach ,we have to have a logic to generate unique value and HTML does not allow us to pass a dynamic value which means we cannot use this approach for static pages. In short with this approach, HTML pages cannot participate in session tracking.
- **Advantage:** The sessionId remains hidden and can be passed through simple HTML pages.
  Disadvantage: However, we will need a logic that will dynamically generate unique values. So, this way of session tracking is not suitable for dynamic pages. Also, form submission can’t occur by clicking on a simple hypertext link. Thus this way is not much suitable.

Maintaining Session Between Web Client And Server

3) URL Rewriting

- URL Rewriting is the approach in which a session (unique) identifier gets appended with each request URL so server can identify the user session. For example if we apply URL rewriting on **http://localhost:8080/jsp-tutorial/home.jsp**, it will become something like **?jSessionId=XYZ** where jSessionId=XYZ is the attached session identifier and value XYZ will be used by server to identify the user session.
- **Advantage:** This technique is independent of the browser. Unlike the approaches above, even if the user has disabled cookies, this method is useful. We also need not pass hidden values. Another advantage is , we need not to submit extra hidden parameter.
- **Disadvantage:** The client needs to keep a track of this unique identifier until the conversation between client and server completes. Thus the client needs to regenerate the URL and append the sessionId with each request made in that session.

http://localhost:8080/jsp/home.jsp?sessionid=”ABC”

Maintaining Session Between Web Client And Server

4) Session Object

- Session object is representation of a user session. User Session starts when a user opens a browser and sends the first request to server. Session object is available in all the request (in entire user session) so attributes stored in Http session in will be available in any servlet or in a jsp.
- When session is created, server generates a unique ID and attach that ID with the session. Server sends back this Id to the client and there on , browser sends back this ID with every request of that user to server with which server identifies the user
- **How to get a Session Object** – By calling getSession() API on HttpServletRequest object (remember this is an implicit object)

a) HttpSession session = request.getSession()

b) HttpSession session = request.getSession(Boolean)

- **Destroy or Invalidate Session –** This is used to kill user session and specifically used when user logs off. To invalidate the session use –

session.invalidate();

Maintaining Session Between Web Client And Server

4) Session Object

Other important methods

- **Object getAttribute(String attributeName)** – this method is used to get the attribute stored in a session scope by attribute name. Remember the return type is Object.
- **void setAttribute(String attributeName, Object value)**- this method is used to store an attribute in session scope. This method takes two arguments- one is attribute name and another is value.
- **void removeAttribute(String attributeName)**- this method is used to remove the attribute from session.
- **public boolean isNew()**- This method returns true if server does not found any state of the client.
- *Browser session and server sessions are different. Browser session is client session which starts when you opens browser and destroy on closing of browser where as server session are maintained at server end.*

Browser Session vs. Server Session

- By default session tracking is based on cookie (session cookie) which is stored in browser memory and not on hard drive. So as soon as browser is closed, that cookie is gone and will not be retrieved again.
- Corresponding to this session identifier, a server session is created at server and will be available until it is invalidated or maximum inactive time has been reached. When user closes the browser without logging off, server does not know that browser has been closed and thus server session remains active till configured maximum inactive time has been reached.
- Maximum inactive session time (in minutes) can be configured in web.xml like below

```
<session-config>
    <session-timeout>15</session-timeout>
</session-config>
```

Store Attributes in Session object wisely

- At a first glance it looks very convenient to store attributes in session as it is available on any JSP or servlet within the same session. It need to be very careful while storing attributes in session as it can be modified by another JSP or servlet accidently which will result in undesirable behaviour.
- Lets consider the application with two jsp. One JSP (FetchBalance) calls some database system and pulls out bank account balance and stores in a session variable (“accountBalance”) for later use.
- Another JSP “UpdateBalance” which updates the database with the value of “accountBalance” session variable.
- There is another developer who is also working on this web application and does not know that it have created a session variable “accountBalance” so he also thought of creating a session variable with same name and stores it value as 0.
- Now in any flow if prior the execution of UpdateBalance, if JSP written by second developer gets executed then it will change the session variable “accountBalance” value ot 0 and then UpdateBalance will update the account balance in database to 0

Deleting Session Data options

- Remove a particular attribute − You can call the public void removeAttribute(String name) method to delete the value associated with the a particular key.
- Delete the whole session − You can call the public void invalidate() method to discard an entire session.
- Setting Session timeout − You can call the public void setMaxInactiveInterval(int interval) method to set the timeout for a session individually.
- Log the user out − The servers that support servlets 2.4, you can call logout to log the client out of the Web server and invalidate all sessions belonging to all the users.
- web.xml Configuration − If you are using Tomcat, apart from the above mentioned methods, you can configure the session time out in web.xml file as follows.

```
<session-config>
  <session-timeout>15</session-timeout>
</session-config>
```

The timeout is expressed as minutes, and overrides the default timeout which is 30 min. in Tomcat.

<%@ page session="false"%> - way to prevent session in JSP

sesssion.invalidate(); not create session for this page .

Session termination function

HttpSession session = request.getSession(false); // so if no session is active no session is created

if (session != null)

session.setMaxInactiveInterval(1); // so it expires immediately

- session.invalidate()
- request.getSession(false)
- session.close()

Big Picture – Web Apps

End User #1

End User #2

Web Server
(Servlets/JSP)

Database

Legacy Applications

Java Applications

Web Service

Other…?

Anatomy of Request Processing

A simple JSP can extract what it needs from the request object, perform the necessary calculations and other logic, and then create output using the response object.

Invoking Dynamic Code (from JSPs)

- Call Java Code Directly

  (Expressions, Declarations, Scriptlets)
- Call Java Code Indirectly

  (Separate Utility Classes, JSP calls methods)
- Use Beans

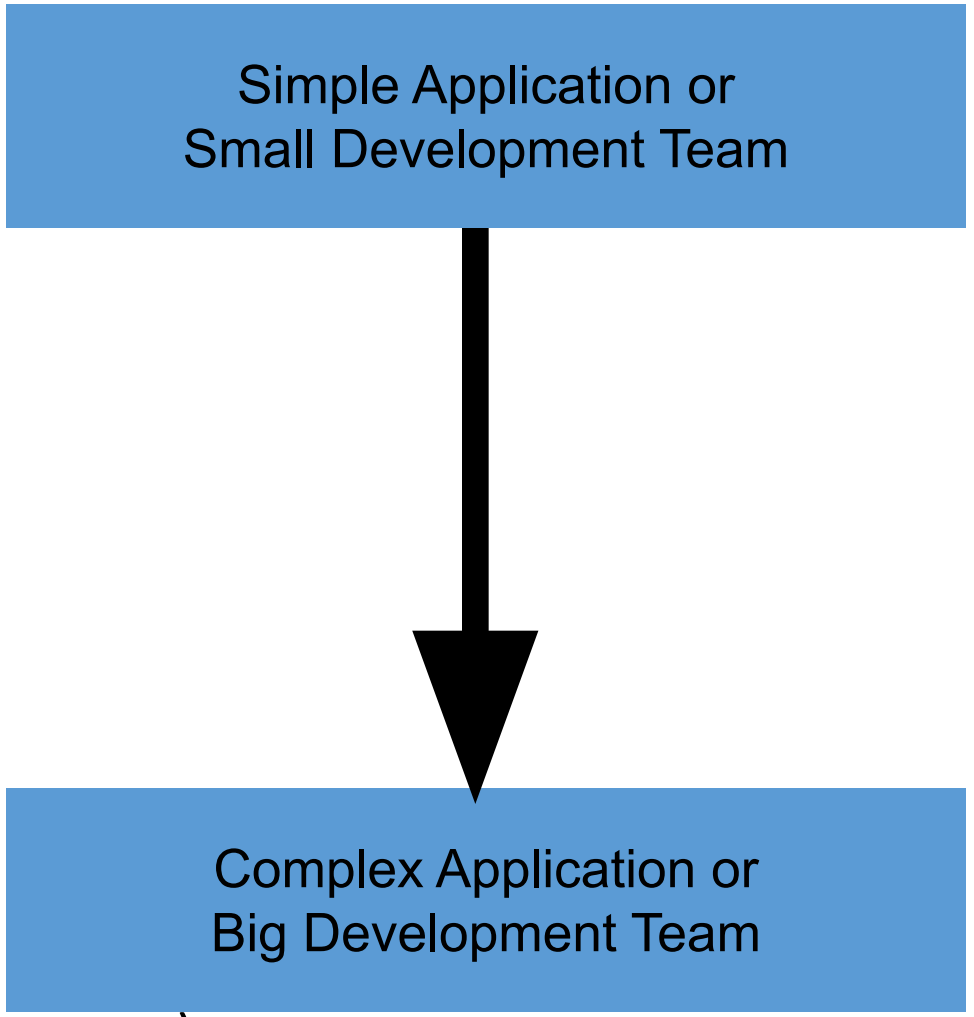
  (jsp:useBean, jsp:getProperty, jsp:setProperty)
- Use MVC architecture (servlet, JSP, JavaBean)
- Use JSP expression Language

  (shorthand to access bean properties, etc)
- Use custom tags

  (Develop tag handler classes; use xml-like custom tags)

Invoking Dynamic Code (from JSPs)

- Call Java Code Directly
  (Expressions, Declarations, Scriptlets)
- Call Java Code Indirectly
  (Separate Utility Classes, JSP calls methods)
- Use Beans
  (jsp:useBean, jsp:getProperty, jsp:setProperty)
- Use MVC architecture (servlet, JSP, JavaBean)
- Use JSP expression Language
  (shorthand to access bean properties, etc)
- Use custom tags
  (Develop tag handler classes; use xml-like custom tags)



Implicit Objects

- Application Scope
  - application
- Session Scope
  - session
- Request Scope
  - request
- Page Scope
  - response, out, page, pageContext