**BỘ THÔNG TIN VÀ TRUYỀN THÔNG**

**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**



**BÁO CÁO NHÓM**

**LÍ THUYẾT CƠ BẢN VỀ LẬP TRÌNH WEB VỚI JSP**

**GIẢNG VIỄN: NGUYỄN MẠNH SƠN**

1. **NHÓM 14**
2. **THÀNH VIÊN NHÓM:**

* **TRẦN QUANG ĐỨC – B17DCCN149**
* **NGUYỄN QUANG HUY – B17DCCN317**
* **NGUYỄN THỊ HÀ – B17DCCN192**
* **PHẠM MINH HẢI - B17DCCN205**
* **NGUYỄN TUẤN KIỆT - B17DCCN361**

**HÀ NỘI 2019**

**Lí Thuyết Cơ bản về Lập trình Web với JSP**

**Mục Lục**

[**I)JSP là gì? 3**](#_Toc22735248)

[**1)Tại sao lại sử dụng JSP. 3**](#_Toc22735249)

[**\*note: so sánh Servlet và JSP 4**](#_Toc22735250)

[**II) Cấu trúc. 5**](#_Toc22735251)

[**III) Vòng đời của JSP. 7**](#_Toc22735252)

[**1) Biên dịch JSP. 8**](#_Toc22735253)

[**2) Khởi tạo JSP. 8**](#_Toc22735254)

[**3) Thực thi JSP. 8**](#_Toc22735255)

[**HttpServletRequest và Http ServletResponse: 9**](#_Toc22735256)

[**4) Hủy JSP 9**](#_Toc22735257)

[**IV) Cú Pháp 10**](#_Toc22735258)

[**1. Scriptlet, declaration, expression trong JSP 10**](#_Toc22735259)

[**2. Comment trong JSP 10**](#_Toc22735260)

[**3. Directive trong JSP 11**](#_Toc22735261)

[**4. Action trong JSP 11**](#_Toc22735262)

[**5. Các đối tượng ẩn (Implicit) trong JSP 12**](#_Toc22735263)

[**6. Điều khiển luồng trong JSP 12**](#_Toc22735264)

[**7. Các lệnh vòng lặp trong JSP 14**](#_Toc22735265)

[**8. Các toán tử trong JSP 15**](#_Toc22735266)

[**9. Literal trong JSP 16**](#_Toc22735267)

[**IV) Đối tượng ẩn (Implicit) trong JSP. 17**](#_Toc22735268)

[**1) Đối tượng request. 17**](#_Toc22735269)

[**2) Đối tượng response. 17**](#_Toc22735270)

[**NOTE: HttpServletResponse và Status Code 18**](#_Toc22735271)

[**3) Đối tượng out. 19**](#_Toc22735272)

[**4) Đối tượng session. 19**](#_Toc22735273)

[**5) Đối tượng application. 19**](#_Toc22735274)

[**6) Đối tượng config. 20**](#_Toc22735275)

[**7) Đối tượng pageContext. 20**](#_Toc22735276)

[**8) Đối tượng page. 21**](#_Toc22735277)

[**9) Đối tượng exception. 21**](#_Toc22735278)

[**V) Xứ lí form in JSP 21**](#_Toc22735279)

[**1. Phương thức GET 21**](#_Toc22735280)

[**2. Phương thức POST 21**](#_Toc22735281)

[**VI) Upload file in JSP 21**](#_Toc22735282)

[**VII) Client Request trong JSP 23**](#_Toc22735284)

[**Đối tượng HttpServletRequest trong JSP 24**](#_Toc22735285)

[**VIII) Server Response trong JSP 28**](#_Toc22735287)

[**Đối tượng HttpServletResponse trong JSP 30**](#_Toc22735288)

[**XI. HTTP Status Code trong JSP 34**](#_Toc22735289)

[**Phương thức để thiết lập HTTP Status Code 37**](#_Toc22735290)

[**X) Filter trong JSP 38**](#_Toc22735291)

[**Các phương thức JSP Filter 39**](#_Toc22735292)

[**XI) Redirect Page trong JSP 39**](#_Toc22735293)

[**VII) Programming in JSP 40**](#_Toc22735294)

[**\*HTML(thẻ cơ bản) 40**](#_Toc22735295)

[**\*Programming in JSP 41**](#_Toc22735297)

# **I)JSP là gì?**

Hai công nghệ chính của Java Web Server (máy chủ web java) là Servlet và JSP .

JSP là mở rộng của Servlet. Trong thực tế, JSP và servlet được sử dụng đồng thời để phát triển ứng dụng.

**JSP là viết tắt của JavaServer Pages** để phát triển các trang web động. JSP giúp các nhà phát triển chèn java code vào các trang HTML bằng cách sử dụng các thẻ JSP đặc biệt.

**JSP** là một kiểu Java servlet được thiết kế để tạo ra giao diện người dùng cho một ứng dụng Java web. Các nhà phát triển web viết các JSP như các tệp văn bản kết hợp mã HTML hoặc XHTML, các phần tử XML, các action và lệnh JSP.

Sử dụng JSP, có thể thu thập dữ liệu đầu vào từ người dùng thông qua các Form của trang web, trình bày các bản ghi từ một cơ sở dữ liệu hoặc một nguồn khác, và tạo các trang web động.

Các thẻ JSP có thể được sử dụng cho nhiều mục đích khác nhau, chẳng hạn như truy xuất thông tin từ cơ sở dữ liệu hoặc đăng ký mới, truy cập các thành phần JavaBeans, kiểm soát giữa các trang và chia sẻ thông tin giữa các request, các trang vv.

**1)Tại sao lại sử dụng JSP.**

Các trang JavaServer thường phục vụ cùng một mục đích như các chương trình được thực hiện bằng cách sử dụng Common Gateway Interface (CGI). Nhưng JSP cung cấp một số lợi thế so với CGI.

* Hiệu suất tốt hơn đáng kể vì JSP cho phép nhúng các thành phần động trong các trang HTML thay vì có các tệp CGI riêng biệt.
* JSP luôn được biên soạn trước khi chúng được xử lý bởi máy chủ không giống như CGI / Perl, yêu cầu máy chủ tải một trình thông dịch và tập lệnh đích mỗi khi trang được request.
* Giống như Servlet, JSP cũng có quyền truy cập vào tất cả các Enterprise Java APIs, bao gồm JDBC, JNDI, EJB, JAXP vv
* Các trang JSP có thể được sử dụng kết hợp với các servlet xử lý logic nghiệp vụ, model được hỗ trợ bởi Java servlet.

JSP là một phần của Java EE, một nền tảng hoàn chỉnh cho các ứng dụng enterprise.

JSP có những ưu điểm sau:

+JSP có tất cả các lợi thế của Java, tức là viết một lần chạy ở bất cứ đâu.

+Nó sử dụng cú pháp dựa trên ngôn ngữ kịch bản đơn giản hóa để nhúng HTML vào JSP.

+Container JSP cung cấp cách dễ dàng để truy cập các đối tượng và hành động tiêu chuẩn.

+JSP gặt hái tất cả các lợi ích được cung cấp bởi các dịch vụ JAVA và môi trường bộ chứa web, nhưng chúng có thêm một lợi thế là chương trình đơn giản và tự nhiên hơn cho web cho phép nhà phát triển doanh nghiệp.

**JSP có những hạn chế sau:**

+Vì các trang JSP được dịch sang các servlet và được biên dịch, rất khó để theo dõi các lỗi xảy ra trong các trang JSP.

+Nó đòi hỏi gấp đôi không gian đĩa để giữ trang JSP.

+Nó đòi hỏi nhiều thời gian hơn khi truy cập lần đầu tiên vì chúng sẽ được biên dịch trên máy chủ.

\***note: so sánh Servlet và JSP**

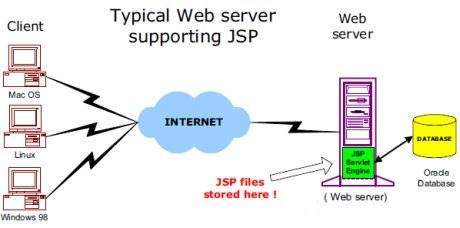
**-** một trong những khái niệm cơ bản gây nhầm lẫn cho nhiều người mới bắt đầu phát triển web Java

| SERVLET | JSP |
| --- | --- |
| Một servlet là một chương trình phía máy chủ và được viết hoàn toàn trên Java. | JSP là một giao diện nằm trên đầu Servlets. Theo cách khác, chúng ta có thể nói rằng JSPs là mở rộng của servlet để giảm thiểu nỗ lực của các nhà phát triển để viết giao diện người dùng bằng cách sử dụng lập trình Java. |
| Servlet chạy nhanh hơn JSP | JSP chạy chậm hơn vì nó có giai đoạn chuyển tiếp để chuyển đổi từ trang JSP sang file Servlet. Một khi nó được chuyển đổi sang một Servlet thì nó sẽ bắt đầu quá trình biên dịch |
| Thực hiện bên trong một máy chủ Web, chẳng hạn như Tomcat | Một chương trình JSP được biên dịch thành một Java servlet. Một khi nó được biên soạn thành một servlet, đó là vòng đời sẽ giống như của servlet. Tuy nhiên, JSP có API của chính nó cho vòng đời. |
| Nhận yêu cầu HTTP từ người dùng và cung cấp phản hồi HTTP | Dễ dàng viết hơn servlet vì nó tương tự như HTML. |
| Không thể xây dựng bất kỳ thẻ tùy chỉnh nào ->sửa đổi giao diện phức tạp. | Có thể xây dựng các thẻ tùy chỉnh bằng cách sử dụng API JSP (có một gói riêng dành cho việc viết các thẻ tùy chỉnh) có thể có sẵn dưới dạng các thành phần có thể tái sử dụng với rất nhiều tính linh hoạt |
| Servlet có các phương pháp vòng đời init (), service () và destroy () | JSP có các phương pháp vòng đời của jspInit (), \_jspService () và jspDestroy () |
| Viết bằng Java, với một vài API bổ sung cụ thể cho loại xử lý này. Vì nó được viết bằng Java nên nó tuân theo tất cả các kỹ thuật lập trình hướng đối tượng. | Các JSP có thể sử dụng JavaBean bên trong các trang web |
| Trong[mô hình MVC](https://itphutran.com/lap-trinh-java/jsp-servlet/mo-hinh-mvc-trong-java-web-jspservlet-la-gi)servlet hoạt động như một controller | Trong [mô hình MVC](https://itphutran.com/lap-trinh-java/jsp-servlet/mo-hinh-mvc-trong-java-web-jspservlet-la-gi), JSP hoạt động như một view. |

# **II) Cấu trúc.**

Web Server cần một JSP engine ví dụ: Container để xử lý các JSP page. JSP container có nhiệm vụ chặn các Request(yc) cho các JSP page.

Một JSP container làm việc với Web Server để cung cấp môi trường runtime và các dịch vụ khác mà một JSP cần đến. Nó biết cách để hiểu các phần tử đặc biệt mà là một phần của JSP.



Sơ đồ sau minh họa

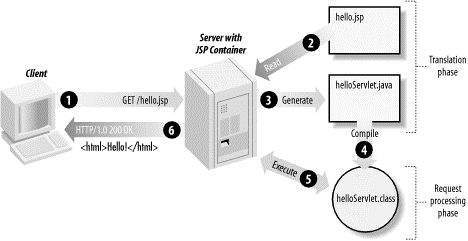
vị trí của JSP container

và JSP file trong một

Web Application

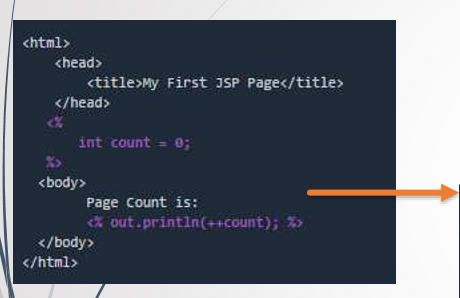
Dưới đây là các bước giải thích cách Web Server tạo các Webpage bởi sử dụng JSP:

* Với một trang thông thường, trình duyệt gửi một HTTP request tới Web server.
* Web server nhận ra rằng HTTP request là cho một JSP page và chuyển nó tới một JSP engine. Điều này được thực hiện bởi sử dụng một URL hoặc một JSP page mà kết thúc với **.jsp** thay vì .html.
* JSP engine tải JSP page từ đĩa và biến đổi nó thành một nội dung Servlet. Sự biến đổi này là rất đơn giản, trong đó tất cả Template Text được biến đổi thành các lệnh printIn() và tất cả phần tử JSP được biến đổi thành Java code mà triển khai các hành vi động tương ứng của trang.
* JSP engine biên dịch Servlet thành một lớp có thể thực thi và chuyển Request ban đầu tới một Servlet engine.
* Một phần của Web server triệu hồi Servlet engine tải lớp Servlet và thực thi nó. Trong quá trình thực thi, Servlet này tạo một output trong định dạng HTML, mà Servlet engine truyền tới Web server bên trong một HTTP response.
* Web server chuyển HTTP Response tới trình duyệt của bạn nội dung HTML tĩnh.
* Cuối cùng, trình duyệt web xử lý trang HTML đã được tạo một cách năng động bên trong HTTP response một cách chính xác như kiểu nó là một trang tĩnh.

Tất cả các bước trên có thể được minh họa trong sơ đồ sau:

Nét đặc trưng là, JSP engine kiểm tra để biết có hay không một Servlet cho một JSP file là đã tồn tại và có hay không ngày sửa đổi JSP là cũ hơn Servlet đó. Nếu JSP là cũ hơn Servlet đã tạo của nó, thì JSP engine giả sử rằng JSP đã không thay đổi và Servlet đã được tạo đó vẫn kết nối với nội dung của JSP. Điều này làm cho tiến trình khác biệt hơn khi so với các ngôn ngữ Scripting khác (ví dụ như PHP) và vì thế nó nhanh hơn.

Trước khi hiển thị ra trình duyệt, tập tin JSP phải được biên dịch thành Servlet, dùng bộ biên dịch JSP (JSP compiler). Bộ biên dịch JSP có thể tạo servlet thành mã nguồn Java trước, rồi biên dịch mã nguồn ra tập tin .class dùng bộ biên dịch Java, hoặc có thể trực tiếp tạo mã byte code cho servlet từ trang JSP. Ngoại trừ giai đoạn phiên dịch, một JSP page được xử lý chính xác giống như một Servlet thông thường, minh họa:

**JSP page -> Servlet ?**

# **III) Vòng đời của JSP.**

Sau đây là các giai đoạn trong vòng đời của của một JSP, 4 giai đoạn chủ yếu:



* Biên dịch
* Khởi tạo
* Thực thi
* Hủy

## 1) Biên dịch JSP.

Khi môt trình duyệt yêu cầu một JSP, đầu tiên JSP engine kiểm tra để biết có hay không nó cần biên dịch trang này. Nếu trang chưa được biên dịch, hoặc nếu JSP đã được sửa đổi từ lần sửa đổi cuối cùng, thì JSP engine biên dịch trang này.

Tiến trình biên dịch gồm 3 bước:

* Parse một JSP(phân tích)
* Biên dịch JSP thành một Servlet
* Biên dịch Servlet đó.

## 2) Khởi tạo JSP.

Khi một container tải một JSP, nó gọi phương thức jspInit() trước khi phục vụ bất kỳ yêu cầu nào. Nếu bạn cần thực hiện sự khởi tạo JSP riêng, ghi đề phương thức jspInit():

public void jspInit(){

// Initialization code...

}

Nét đặc trưng là, việc khởi tạo được thực hiện chỉ một lần và với phương thức int của Servlet, nói chung bạn khởi tạo các kết nối Database, mở các file, và tạo các bảng tra cứu trong phương thức jspInit.

## 3) Thực thi JSP.

Giai đoạn này trong vòng đời JSP biểu diễn tất cả các tương tác với Request tới khi JSP đó bị hủy.

Bất cứ khi nào một trình duyệt yêu cầu một JSP và trang đã được tải và được khởi tạo, thì JSP engine triệu hồi phương thức **\_jspService()** trong JSP đó.

Phương thức \_jspService() nhận  **HttpServletRequest** và  **Http ServletResponse** như là các tham số của nó.

void \_jspService(HttpServletRequest request,

HttpServletResponse response)

{

// Service handling code...

}

Phương thức \_jspService() của một JSP được triệu hồi một lần cho mỗi yêu cầu và nó chịu trách nhiệm tạo Response cho Request đó và phương thức này cũng chịu trách nhiệm tạo các phản hồi tới tất cả 7 phương thức của HTTP, ví dụ: GET, POST, DELETE, HEAD, PUT, CONNECT, OPTIONS.

**HttpServletRequest và Http ServletResponse:**

**+HttpServletRequest:**  Mỗi khi một Client yêu cầu một trang, thì JSP engine tạo một đối mới này để biểu diễn yêu cầu đó.:

+có các phương thức như: Cookie[] getCookies(),Enumeration getAttributeNames(), Enumeration getHeaderNames(),Enumeration getParameterNames()

VD: đối tượng resquest ở trên

**+Http ServletResponse:** Ngay khi Server tạo đối tượng để request, nó cũng tạo một đối tượng để biểu diễn phản hồi tới Client.

+có các phương thức như: String encodeRedirectURL(String url), String encodeURL(String url), boolean containsHeader(String name), boolean isCommitted(),

VD: đối tượng response ở trên)

**4) Hủy JSP**

Giai đoạn hủy một JSP trong vòng đời JSP biểu thị khi nào thì một JSP bị gỡ bỏ khỏi một container.

Phương thức **jspDestroy()** trong JSP là phương thức hủy tương đương với trong Servlet. Ghi đè phương thức jspDestroy khi bạn cần thực hiện bất kỳ quá trình hủy nào, ví dụ như giải phóng kết nối với Database, hoặc đóng các file.

Phương thức jspDestroy() có form sau:

public void jspDestroy()

{

// Your cleanup code goes here.

}

**IV) Cú Pháp**

## Scriptlet, declaration, expression trong JSP

Có 3 kiểu nhúng code vào HTML:

* **Thẻ khai báo (declaration):** định nghĩa biến hoặc phương thức trong JSP.

• Cú pháp: <%! declaration; %>

Hay

<jsp:declaration> code fragment </jsp:declaration>

ví dụ :

* <%! int i = 0; %>
* <%! int a, b, c; %>
* <%! Circle a = new Circle(2.0); %>
* **Thẻ văn lệnh con (scriptlet) :** dùng để chứa biến phương thức trong Java code và sau đó trong Java file

• Cú pháp: jsp:scriptlet> code fragment </jsp:scriptlet>

Ví dụ: <body> Hello World!<br/>

<% out.println("Your IP address is " + request.getRemoteAddr()); %> </body>

* **Thẻ biểu thức (expression):** dùng để hiển thị dùng để hiển thị kết quả của một biểu thức.

• Cú pháp:<% = expression%> hay

<jsp:expression>expression</jsp:expression>

• Ví dụ: <body>   
<p> Today's date:

<%= (new java.util.Date()).toLocaleString()%> </p> </body

1. **Comment trong JSP**

Comment trong JSP đánh dấu text hoặc các lệnh mà JSP container nên bỏ qua. Một JSP comment là hữu ích khi bạn muốn ẩn hoặc bỏ qua phần đó trong JSP page. Cú pháp :

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

Ví dụ :

<html>

<head><title>A Comment Test</title></head>

<body>

<h2>A Test of Comments</h2>

<%-- This comment will not be visible in the page source --%>

</body>

<html>

## Directive trong JSP

Một directive trong JSP ảnh hưởng tới cấu trúc tổng thể của lớp Servlet. Nó thường có form sau:

<%@ directive attribute="value" %>

Có 3 loại thẻ directive trong JSP:

|  |  |
| --- | --- |
| **Directive** | **Miêu tả** |
| <%@ page ... %> | Định nghĩa một thuộc tính page-dependent (phụ thuộc trang), như ngôn ngữ scripting, trang lỗi và các yêu cầu bộ đệm |
| <%@ include ... %> | Include một file trong suốt giai đoạn phiên dịch |
| <%@ taglib ... %> | Khai báo một thư viện thẻ, chứa các action tùy biến, được sử dụng trong trang đó |

## Action trong JSP

Action trong JSP sử dụng các chỉ lệnh trong cú pháp XML để điều khiển hành vi của Servlet engine. Bạn có thể chèn động một file, tái sử dụng các thành phần JavaBeans, chuyển người sử dụng tới trang khác, hoặc tạo HTML cho Java Plugin.

Chỉ có một cú pháp cho phần tử action trong JSP, khi nó phù hợp tiêu chuẩn XML:

<jsp:action\_name attribute="value" />

Về cơ bản, các phần tử action trong JSP là các hàm được định nghĩa trước và có các JSP action sau:

|  |  |
| --- | --- |
| **Cú pháp** | **Mục đích** |
| jsp:include | Include một file tại thời điểm trang đó được yêu cầu |
| jsp:useBean | Tìm kiếm hoặc thuyết minh một JavaBean |
| jsp:setProperty | Thiết lập thuộc tính của một JavaBean |
| jsp:getProperty | Chèn thuộc tính của một JavaBean vào trong output |
| jsp:forward | Chuyển người yêu cầu tới trang mới |
| jsp:plugin | Tạo code cho trình duyệt riêng mà tạo một thẻ OBJECT hoặc EMBED cho Java plugin |
| jsp:element | Định nghĩa động các phần tử XML |
| jsp:attribute | Định nghĩa động thuộc tính của phần tử XML đã được định nghĩa |
| jsp:body | Định nghĩa động phần thân của phần tử XML đã được định nghĩa |
| jsp:text | Sử dụng để viết template text trong các JSP page và tài liệu JSP |

## Các đối tượng ẩn (Implicit) trong JSP

JSP hỗ trợ 9 biến được định nghĩa tự động, mà cũng được gọi là các đối tượng ẩn (implicit object). Sẽ nói trong phần sau

## Điều khiển luồng trong JSP

JSP cung cấp đầy đủ tính năng của Java để được nhúng trong các Web application

* khối **if...else** khởi đầu giống như một Scriptlet thông thường, nhưng Scriptlet được đóng tại mỗi dòng với HTML text được bao giữa các thẻ Scriptlet.

<%! int day = 3; %>

<html>

<head><title>IF...ELSE Example</title></head>

<body>

<% if (day == 1 | day == 7) { %>

<p> Today is weekend</p>

<% } else { %>

<p> Today is not weekend</p>

<% } %>

</body>

</html>

kết quả :

Today is not weekend

* khối **switch...case** mà có một chút khác biệt bởi sử dụng lệnh **out.printIn()** và bên trong các Scriptlet:

<%! int day = 5; %>

<html>

<head><title>SWITCH...CASE Example</title></head>

<body>

<%

switch(day) {

case 0:

out.println("It\'s Sunday.");

break;

case 1:

out.println("It\'s Monday.");

break;

case 2:

out.println("It\'s Tuesday.");

break;

case 3:

out.println("It\'s Wednesday.");

break;

case 4:

out.println("It\'s Thursday.");

break;

case 5:

out.println("It\'s Friday.");

break;

default:

out.println("It's Saturday.");

}

%>

</body>

</html>

kết quả :

It's Friday.

## Các lệnh vòng lặp trong JSP

* vòng lặp **for** trong JSP:

<%! int fontSize; %>

<html>

<head><title>FOR LOOP Example</title></head>

<body>

<%for ( fontSize = 1; fontSize <= 3; fontSize++){ %>

<font color="green" size="<%= fontSize %>">

JSP Tutorial

</font><br />

<%}%>

</body>

</html>

kết quả :

JSP Tutorial

JSP Tutorial

JSP Tutorial

* vòng lặp **while**  :

<%! int fontSize; %>

<html>

<head><title>WHILE LOOP Example</title></head>

<body>

<%while ( fontSize <= 3){ %>

<font color="green" size="<%= fontSize %>">

JSP Tutorial

</font><br />

<%fontSize++;%>

<%}%>

</body>

</html>

Nó sẽ cho kết quả sau:

JSP Tutorial

JSP Tutorial

JSP Tutorial

## Các toán tử trong JSP

JSP hỗ trợ tất cả toán tử logic và toán tử số học được hỗ trợ bởi Java.

|  |  |  |
| --- | --- | --- |
| **Loại** | **Toán tử** | **Thứ tự ưu tiên** |
| Postfix | () [] . (toán tử dot) | Trái sang phải |
| Unary | ++ - - ! ~ | Phải sang trái |
| Tính nhân | \* / % | Trái sang phải |
| Tính cộng | + - | Trái sang phải |
| Dịch chuyển | >> >>> << | Trái sang phải |
| Quan hệ | > >= < <= | Trái sang phải |
| Cân bằng | == != | Trái sang phải |
| Phép AND bit | & | Trái sang phải |
| Phép XOR bit | ^ | Trái sang phải |
| Phép OR bit | | | Trái sang phải |
| Phép AND logic | && | Trái sang phải |
| Phép OR logic | || | Trái sang phải |
| Điều kiện | ?: | Phải sang trái |
| Gán | = += -= \*= /= %= >>= <<= &= ^= |= | Phải sang trái |
| Dấu phảy | , | Trái sang phải |

## Literal trong JSP

JSP Expression Language (EL) định nghĩa các literal sau:

* **Boolean:** true và false
* **Integer:** như trong Java
* **Floating point:** như trong Java
* **String:** với các trích dẫn đơn và kép; " được thay thế bởi \", ' được thay thế bởi \', và \ được thay thế bởi \\.
* **Null:** null

# **IV) Đối tượng ẩn (Implicit) trong JSP.**

Đối tượng ẩn (Implicit Object) trong JSP là các đối tượng Java mà JSP container làm nó có sẵn tới các lập trình viên trong mỗi trang và lập trình viên có thể gọi chúng một cách trực tiếp mà không phải khai báo rõ ràng. Đối tượng ẩn trong JSP cũng được gọi là các biến tiền định nghĩa.

JSP hỗ trợ 9 đối tượng ẩn sau đây:

|  |  |
| --- | --- |
| **Action** | **Mô tả** |
| **request** | Đây là đối tượng **HttpServletRequest** được liên kết với request. |
| **response** | Đây là đối tượng **HttpServletResponse** được liên kết với response tới client. |
| **out** | Đây là đối tượng **PrintWriter** được sử dụng để gửi dữ liệu tới client. |
| **session** | Đây là đối tượng **HttpSession** được liên kết với request. |
| **application** | Đây là đối tượng **ServletContext** được liên kết với application context. |
| **config** | Đây là đối tượng **ServletConfig** được liên kết với page. |
| **pageContext** | Điều này đóng gói việc sử dụng các tính năng cụ thể của server như **JspWriters** với hiệu suất cao hơn. |
| **page** | Đơn giản là một từ đồng nghĩa với **this**, được sử dụng để gọi các phương thức được định nghĩa bởi lớp servlet được biên dịch. |
| **Exception** | Đối tượng **Exception** cho phép các dữ liệu ngoại lệ được truy cập bằng JSP được chỉ định. |

### **1) Đối tượng request.**

Đối tượng request là một thể hiện của **javax.servlet.http.HttpServletRequest**. Mỗi lần client request một trang thì JSP engine tạo ra một đối tượng mới để đại diện cho request đó.

Đối tượng request cung cấp các phương thức để lấy thông tin HTTP header bao gồm Form data, cookie, HTTP method vv.

### **2) Đối tượng response.**

Đối tượng response là một thể hiện của **javax.servlet.http.HttpServletResponse**. Cũng như việc server tạo ra các đối tượng resquest, nó cũng tạo ra một đối tượng response để đại diện cho các phản hồi cho client.

Đối tượng response cũng định nghĩa các giao diện để tạo ra HTTP header mới. Thông qua đối tượng này, lập trình viên JSP có thể thêm cookie mới hoặc date stamps, HTTP status code, vv.

**NOTE:** **HttpServletResponse và Status Code**

**+** Được sử dụng thiết lập HTTP Status code trong chương trình JSP:

+[Status Code trong JSP](javascript:void(0)) **:** trong các thông báo HTTP request và HTTP response sẽ có 1 đoạn là status và code với một mã ứng với 1 thông báo vd:

Xét một một header phản hồi từ Server

HTTP/1.1 200 OK

Content-Type: text/html

Header2: ...

...

HeaderN: ...

(Blank Line)

<!doctype ...>

<html>

<head>...</head>

<body>

...

</body>

</html>

Dòng Status gồm phiên bản HTTP (trong ví dụ là HTTP/1.1)+ một status code ( ví dụ là 200), và một thông báo rất ngắn tương ứng với Status code đó (vd là OK).

+ Nó tạo ra đối tượng có các phương thức như: public void setStatus ( int statusCode ), public void sendRedirect(String url), public void sendError(int code, String message).

Ví dụ sau sẽ gửi code lỗi là 407 tới trình duyệt Client và trình duyệt sẽ hiển thị thông báo "Need authentication".

<html>

<head>

<title>Setting HTTP Status Code</title>

</head>

<body>

<%

// Set error code and reason.

response.sendError(407, "Need authentication!!!" );

%>

</body>

</html>

### **3) Đối tượng out.**

Đối tượng ngầm ẩn là một thể hiện của **javax.servlet.jsp.JspWriter** và được sử dụng để gửi nội dung trong một phản hồi.

Đối tượng JspWriter ban đầu được khởi tạo bằng cách khác nhau tùy thuộc vào việc trang được buffered hay không. Buffer có thể dễ dàng bị tắt bằng cách sử dụng thuộc tính **buffered = ‘false’** của page directive.

Đối tượng JspWriter chứa hầu hết các phương thức tương tự như lớp **java.io.PrintWriter**. Tuy nhiên, JspWriter có một số phương thức bổ sung được thiết kế để đối phó với buffering. Không giống như đối tượng PrintWriter, JspWriter throw IOExceptions.

Bảng sau liệt kê ra các phương thức quan trọng mà chúng ta sẽ sử dụng để ghi kiểu dữ liệu char, boolean, int, double, object, String , vv

|  |  |
| --- | --- |
| **Phương thức** | **Mô tả** |
| **out.print(dataType dt)** | In giá trị của kiểu dữ liệu. |
| **out.println(dataType dt)** | In giá trị kiểu dữ liệu với ký tự dòng mới. |
| **out.flush()** | stream. |

### **4) Đối tượng session.**

Đối tượng session là một thể hiện của **javax.servlet.http.HttpSession** và hoạt động chính xác theo cách mà các đối tượng session trong Java Servlets.

Đối tượng session được sử dụng để theo dõi phiên của các request của client.

### **5) Đối tượng application.**

Đối tượng application là thể hiện của một đối tượng **javax.servlet.ServletContext**.

Đối tượng này là một đại diện của trang JSP thông qua toàn bộ vòng đời của nó. Đối tượng này được tạo ra khi trang JSP được khởi tạo và sẽ bị xóa khi trang JSP bị xóa bởi phương thức **jspDestroy()**.

Bằng cách thêm thuộc tính vào application, bạn có thể đảm bảo rằng tất cả các tệp JSP của ứng dụng web của bạn đều có quyền truy cập vào nó.

### **6) Đối tượng config.**

Đối tượng config là thể hiện của một đối tượng **javax.servlet.ServletConfig**.

Đối tượng này cho phép lập trình viên JSP truy cập vào các tham số khởi tạo Servlet hoặc JSP engine như đường dẫn hoặc vị trí tập tin, vv.

Phương thức sau trả về tên servlet, là chuỗi chứa trong phần tử **<servlet-name>** được định nghĩa trong file **WEB-INF\web.xml**.

|  |
| --- |
| config.getServletName(); |

### **7) Đối tượng pageContext.**

Đối tượng pageContext là một thể hiện của **javax.servlet.jsp.PageContext**. Đối tượng pageContext được sử dụng để đại diện cho toàn bộ trang JSP.

Đối tượng này được dùng như một phương tiện để truy cập thông tin về trang trong khi vẫn tránh được hầu hết các chi tiết thực hiện.

Đối tượng này chứa các tham chiếu tới các đối tượng request và đối tượng response cho mỗi yêu cầu. Các đối tượng **application, config, session,**và **out** được bắt nguồn bằng cách truy cập các thuộc tính của đối tượng này.

Đối tượng pageContext cũng chứa thông tin về các chỉ thị được cấp cho trang JSP, bao gồm thông tin về bộ đệm, errorPageURL và phạm vi trang.

Lớp **PageContext** định nghĩa một số trường, bao gồm định nghĩa bốn phạm vi **PAGE\_SCOPE**, **REQUEST\_SCOPE**, **SESSION\_SCOPE** và **APPLICATION\_SCOPE**. Nó cũng hỗ trợ hơn 40 phương thức, khoảng một nửa trong số đó được kế thừa từ lớp **javax.servlet.jsp.JspContext**.

Một trong những phương thức quan trọng là **removeAttribute**. Phương thức này chấp nhận một hoặc hai đối số. Ví dụ, **pageContext.removeAttribute (“attrName”)** loại bỏ các thuộc tính từ tất cả các phạm vi, trong khi mã sau chỉ loại bỏ nó từ phạm vi trang:

|  |
| --- |
| pageContext.removeAttribute("attrName", PAGE\_SCOPE); |

### **8) Đối tượng page.**

Đối tượng này là một tham chiếu thực tế tới thể hiện của một page. Nó có thể được coi như một đối tượng đại diện cho toàn bộ trang JSP.

Đối tượng page là tương đương vơi đối tượng this.

### **9) Đối tượng exception.**

Đối tượng ngoại lệ là một wrapper có chứa ngoại lệ được ném ra từ trang trước. Nó thường được sử dụng để tạo ra một phản hồi phù hợp với điều kiện lỗi.

**V) Xứ lí form in JSP**

Phương thức GET và phương thức POST: truyền một số thông tín từ trình duyệt tới Webserver và sau cùng tới chương trình backend.

## Phương thức GET

Phương thức GET gửi thông tin người dùng đã được mã hóa phụ thêm tới yêu cầu trang. Trang này và thông tin đã được mã hóa được phân biệt bởi ký tự ‘?’

*http://www.test.com/hello****?****key1=value1&key2=value2*

Phương thức GET là phương thức mặc định để truyền thông tin từ trình duyệt tới Web Server và nó tạo ra một chuỗi dài mà xuất hiện trong Location:box của trình duyệt. Không bao giờ sử dụng phương thức GET nếu bạn truyền tới Server thông tin password hoặc thông tin nhạy cảm khác.

Phương thức **GET** có kích cỡ giới hạn: chỉ có 1024 ký tự trong một chuỗi yêu cầu.

Thông tin này được truyền bởi sử dụng **QUERY\_STRING Header** và sẽ có thể truy cập thông qua biến môi trường **QUERY\_STRING** mà có thể được xử lý bởi sử dụng phương thức **getQueryString()** và **getParameter()** của đối tượng request trong JSP.

1. **Phương thức POST**

Nó gói thông tin theo cách giống như phương thức GET, nó gửi thông tin như là một thông báo riêng biệt. Thông báo này tới chương trình backend trong form của input chuẩn mà có thể parse và sử dụng.

JSP xử lý kiểu yêu cầu này bởi sử dụng phương thức **getParameter()** để đọc các tham số đơn giản và phương thức **getInputStream()** để đọc các luồng dữ liệu nhị phân tới từ Client.

**VI) Upload file in JSP**

Để upload các file (Một file được tải lên có thể là file dạng text, nhị phân hoặc hình ảnh hoặc bất kỳ tài liệu nào khác) tới Server một JSP có thể được sử dụng với một thẻ HTML form

## Tạo một File Upload Form trong JSP

HTML code sau tạo một Upload Form với những điểm quan trọng :

* Thuộc tính **Phương thức** của thẻ form nên được thiết lập là phương thức **POST** và phương thức **GET** không thể được sử dụng.
* Thuộc tính **enctype** của thẻ form nên được thiết lập là **multipart/form-data**.
* Thuộc tính **action** trong thẻ form nên được thiết lập là một JSP file mà sẽ xử lý việc upload file tại Backend Server. Ví dụ sau sử dụng chương trình **uploadFile.jsp** để upload file.
* Để upload một file đơn, sử dụng một thẻ <input .../> đơn với thuộc tính là type="file". Để upload nhiều file lên, sử dụng nhiều thẻ input với các giá trị thuộc tính khác nhau. Trình duyệt liên kết một nút Browse với từng giá trị đó.

<html>

<head>

<title>File Uploading Form</title>

</head>

<body>

<h3>File Upload:</h3>

Select a file to upload: <br />

<form action="UploadServlet" method="post"

enctype="multipart/form-data">

<input type="file" name="file" size="50" />

<br />

<input type="submit" value="Upload File" />

</form>

</body>

</html>

# **VII) Client Request trong JSP**

Khi một trình duyệt yêu cầu một Webpage, nó gửi nhiều thông tin tới Web server, mà không thể được đọc một cách trực tiếp bởi vì thông tin này là một phần trong Header của HTTP request.

Thông tin header quan trọng mà tới từ Brower Side .

|  |  |
| --- | --- |
| **eader** | **Miêu tả** |
| Accept | Trường này xác định loại MIME mà trình duyệt hoặc Client khác có thể xử lý. Các giá trị **image/png** hoặc **image/jpeg** là hai khả năng phổ biến nhất |
| Accept-Charset | Trường này xác định bộ ký tự mà trình duyệt có thể sử dụng để hiển thị thông tin. Ví dụ: ISO-8859-1 |
| Accept-Encoding | Trường này xác định kiểu mã hóa mà trình duyệt biết để xử lý. Các giá trị của **gzip** hoặc **compress** là hai khả năng phổ biến nhất |
| Accept-Language | Trường này xác định ngôn ngữ ưa thíc của Client trong trường hợp Servlet có thể tạo các kết quả trong nhiều ngôn ngữ. Ví dụ en, en-us, ru, .v.v. |
| Authorization | Trường này được sử dụng bởi Client để nhận diện chính nó khi truy cập vào các trang web được bảo vệ bởi password |
| Connection | Trường này chỉ dẫn có hay không Client có thể xử lý các kết nối HTTP dạng Persistent. Các kết nối Persistent cho phép Client hoặc trình duyệt thu nhận nhiều file với một yêu cầu đơn. Một giá trị của **Keep-Alive** nghĩa là các kết nối Persistent này nên được sử dụng |
| Content-Length | Trường này chỉ thích hợp với các yêu cầu POST và cung cấp kích cỡ của dữ liệu POST bằng giá trị byte |
| Cookie | Trường này trả về các cookie tới các Server mà trước đó đã gửi chúng tới trình duyệt |
| Host | Trường này xác định host và cổng (port) khi được cung cấp trong URL ban đầu |
| If-Modified-Since | Trường này chỉ rằng Client muốn trang này chỉ khi nó đã được thay đổi sau ngày đã xác định. Server gửi một code là 304, nghĩa là **Not Modified** Header nếu không có kết quả nào mới hơn là có sẵn |
| If-Unmodified-Since | Trường này là ngược với trường If-Modified-Since. Nó xác định hoạt động là thành công chỉ khi tài liệu là cũ hơn ngày đã xác định |
| Referer | Trường này chỉ URL của trang web đang tham chiếu đó. Ví dụ, nếu bạn tại trang Webpage 1 và click vào một link tới trang Webpage 2, thì URL của Webpage 1 được bao trong trường Referer Header khi trình duyệt yêu cầu trang Webpage 2 |
| User-Agent | Trường này nhận diện trình duyệt hoặc Client đang tạo yêu cầu và có thể được sử dụng để trả về nội dung khác nhau cho các loại trình duyệt khác nhau |

## HttpServletRequest trong JSP

Đối tượng request là một minh họa của một đối tượng thuộc javax.servlet.http.HttpServletRequest. Mỗi khi một Client yêu cầu một trang, thì JSP engine tạo một đối tượng mới để biểu diễn yêu cầu đó.

Đối tượng request cung cấp các phương thức để nhận thông tin HTTP header gồm Form data, cookie, phương thức HTTP, …

Các phương thức quan trọng có thể được sử dụng để đọc HTTP header trong chương trình JSP . Các phương thức này có sẵn với đối tượng *HttpServletRequest* mà biểu diễn Client Request tới Webserver.

|  |  |
| --- | --- |
| **STT** | **Phương thức & Miêu tả** |
| 1 | Cookie[] getCookies()  Trả về một mảng chứa tất cả đối tượng Cookie mà Client gửi với yêu cầu này |
| 2 | Enumeration getAttributeNames()  Trả về một Enumeration chứa các tên của thuộc tính có sẵn cho yêu cầu này |
| 3 | Enumeration getHeaderNames()  Trả về một Enumeration của tất cả tên Header mà yêu cầu này chứa |
| 4 | Enumeration getParameterNames()  Trả về một Enumeration của các đối tượng String chứa tên của Parameter được chứa trong yêu cầu này |
| 5 | HttpSession getSession()  Trả về session hiện tại, mà liên kết với yêu cầu này, hoặc yêu cầu đó không có một session, nó sẽ tạo ra một session |
| 6 | HttpSession getSession(boolean create)  Trả về HttpSession hiện tại, mà liên kết với yêu cầu này, hoặc nếu không có session hiện tại nào và create là true, nó trả về một session mới |
| 7 | Locale getLocale()  Trả về Locale mà Client sẽ chấp chận nội dung trong đó, dựa trên trường Accept-Language Header |
| 8 | Object getAttribute(String name)  Trả về giá trị của thuộc tính name như là một Object, hoặc trả về null nếu không có thuộc tính nào trong name đã cung cấp tồn tại |
| 9 | ServletInputStream getInputStream()  Thu nhận thân yêu cầu dạng dữ liệu nhị phân bởi sử dụng một ServletInputStream |
| 10 | String getAuthType()  Trả về tên của Authentication Scheme được sử dụng để bảo vệ Servlet, ví dụ: "BASIC" hoặc "SSL", hoặc trả về null nếu JSP không được bảo vệ |
| 11 | String getCharacterEncoding()  Trả về tên của Mã hóa ký tự được sử dụng trong thân yêu cầu này |
| 12 | String getContentType()  Trả về kiểu MIME của thân yêu cầu, hoặc trả về null nếu không biết kiểu này |
| 13 | String getContextPath()  Trả về phần URI yêu cầu mà chỉ dẫn context của yêu cầu đó |
| 14 | String getHeader(String name)  Trả về giá trị của trường Header đã xác định dưới dạng một String |
| 15 | String getMethod()  Trả về tên của phương thức HTTP mà yêu cầu này được tạo, ví dụ: GET, POST, hoặc PUT |
| 16 | String getParameter(String name)  Trả về giá trị của Parameter dưới dạng một String, hoặc trả về null nếu Parameter này không tồn tại |
| 17 | String getPathInfo()  Trả về bất kỳ thông tin path bổ sung, mà liên kết với URI mà Client đã gửi khi nó tạo yêu cầu này |
| 18 | String getProtocol()  Trả về tên và phiên bản của giao thức |
| 19 | String getQueryString()  Trả về chuỗi truy vấn, được chứa trong URL sau path đó |
| 20 | String getRemoteAddr()  Trả về địa chỉ Internet Protocol (IP) của một Client mà gửi yêu cầu này |
| 21 | String getRemoteHost()  Trả về tên đầy đủ của Client mà gửi yêu cầu này |
| 22 | String getRemoteUser()  Trả về login của người sử dụng tạo yêu cầu này, nếu người dùng đã được xác nhận, hoặc trả về null nếu người dùng chưa được xác nhận |
| 23 | String getRequestURI()  Trả về phần URL của yêu cầu này từ tên giao thức dưới dạng một chuỗi truy vấn trong dòng đầu tiên của HTTP Request |
| 24 | String getRequestedSessionId()  Trả về ID của session đã được xác định bởi Client |
| 25 | String getServletPath()  Trả về phần URL của yêu cầu này mà gọi JSP |
| 26 | String[] getParameterValues(String name)  Trả về một mảng đối tượng String chứa tất cả các giá trị mà các parameter đã cung cấp có, hoặc trả về null nếu parameter này không tồn tại |
| 27 | boolean isSecure()  Trả về một Boolean chỉ rằng có hay không yêu cầu này được tạo bởi sử dụng một đường an toàn, như HTTPS |
| 28 | int getContentLength()  Trả về độ dài, bằng giá trị byte, của thân yêu cầu và làm nó có sẵn bởi Input Stream, hoặc trả về -1 nếu độ dài là không được biết |
| 29 | int getIntHeader(String name)  Trả về giá trị của Request header đã cho dưới dạng một int |
| 30 | int getServerPort()  Trả về số hiệu cổng mà trên đó yêu cầu được nhận |

## Tên và giá trị tương ứng của HTTP Header Request trong JSP

|  |  |
| --- | --- |
| **Tên Header** | **Giá trị Header** |
| accept | \*/\* |
| accept-language | en-us |
| user-agent | Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Trident/4.0; InfoPath.2; MS-RTC LM 8) |
| accept-encoding | gzip, deflate |
| host | localhost:8080 |
| connection | Keep-Alive |
| cache-control | no-cache |

# **VIII) Server Response trong JSP**

Khi một Web Server phản hồi một HTTP Request tới trình duyệt, phản hồi đặc trưng bao gồm một dòng Status, một số trường Header, một dòng trống, và tài liệu.

Dưới đây là các trường Header phổ biến trong phản hồi HTTP :

|  |  |
| --- | --- |
| **Header** | **Miêu tả** |
| Allow | Trường này xác định các phương thức yêu cầu (GET, POST, .v.v.) mà Server hỗ trợ |
| Cache-Control | Trường này xác định các hoàn cảnh mà trong đó tài liệu phản hồi có thể được cache một cách an toàn. Nó có thể có các giá trị **publics**, **private** hoặc **no-cache**, .v.v.. Trong đó, publics nghĩa là tài liệu có thể được cache, private nghĩa là tài liệu dành cho một người sử dụng và chỉ có thể được lưu giữ riêng (không được chia sẻ) và no-cache nghĩa là tài liệu không bao giờ được cache |
| Connection | Trường này chỉ dẫn trình duyệt có sử dụng persistent trong các kết nối HTTP hoặc không. Một giá trị **close** chỉ rằng trình duyệt không sử dụng các kết nối HTTP dạng persistent và **keep-alive** nghĩa là sử dụng các kết nối persistent |
| Content-Disposition | Trường này cho bạn yêu cầu rằng trình duyệt hỏi người sử dụng để lưu phản hồi lên địa trong một file với tên đã cho |
| Content-Encoding | Trường này xác định cách mà trang được mã hóa trong khi truyền tải |
| Content-Language | Trường này xác định ngôn ngữ mà tài liệu được viết. Ví dụ en, en-us, ru, … |
| Content-Length | Trường này xác định số byte trong phản hồi. Thông tin này chỉ cần thiết nếu trình duyệt đang sử dụng một kết nối HTTP dạng persistent (giá trị keep-alive) |
| Content-Type | Trường này kiểu MIME (Multipurpose Internet Mail Extension) của tài liệu phản hồi |
| Expires | Trường này xác định thời gian tại đó nội dung nên được xem là hết hạn và không bao giờ được cache nữa |
| Last-Modified | Trường này chỉ dẫn khi tài liệu được thay đổi lần cuối. Sau đó Client có thể cache tài liệu này và cung cấp một date bởi một trường yêu cầu **If-Modified-Since** trong các yêu cầu sau đó |
| Location | Trường này nên được bao với tất cả phản hồi mà có Status code là 300. Nó thông báo cho trình duyệt về địa chỉ tài liệu. Trình duyệt tự động kết nối lại tới vị trí này và thu nhận tài liệu mới |
| Refresh | Trường này xác định khi nào trình duyệt nên đề nghị một trang đã được update. Bạn có thể xác định thời gian với đơn vị là số giây sau đó một trang sẽ được refresh |
| Retry-After | Trường này có thể được sử dụng khi kết hợp với phản hồi 503 (Service Unavailable) để nói cho Client khi nào thì nó có thể lặp lại yêu cầu |
| Set-Cookie | Trường này xác định một cookie mà liên kết với trang |

## HttpServletResponse trong JSP

Đối tượng response là một minh họa của đối tượng javax.servlet.http.HttpServletResponse. Ngay khi Server tạo đối tượng request, nó cũng tạo một đối tượng để biểu diễn phản hồi tới Client.

Đối tượng Response cũng định nghĩa các Interface mà thực hiện việc tạo các HTTP header mới. Thông qua đối tượng này, có thể thêm các Cookie mới hoặc Date, HTTP Status code, …

Dưới đây là các phương thức có thể được sử dụng để thiết lập HTTP response Header . Những phương thức này có sẵn với đối tượng **HttpServletResponse** mà biểu diễn Server Response.

|  |  |
| --- | --- |
| **STT** | **Phương thức và Miêu tả** |
| 1 | **String encodeRedirectURL(String url)**  Mã hóa URL đã cho để sử dụng trong phương thức sendRederect, hoặc nếu mã hóa này không được đòi hỏi, thì trả về URL như cũ |
| 2 | **String encodeURL(String url)**  Mã hóa URL đã cho bằng việc thêm ID của session trong nó, hoặc nếu mã hóa này không được đòi hỏi, thì trả về URL như cũ |
| 3 | **boolean containsHeader(String name)**  Trả về một Boolean chỉ rằng có hay không header phản hồi với tên đã cho đã được thiết lập |
| 4 | **boolean isCommitted()**  Trả về một Boolean chỉ rằng nếu phản hồi đã được commit |
| 5 | **void addCookie(Cookie cookie)**  Thêm cookie đã cho tới phản hồi |
| 6 | **void addDateHeader(String name, long date)**  Thêm một header phản hồi với tên và giá trị date đã cho |
| 7 | **void addHeader(String name, String value)**  Thêm một header phản hồi với tên và giá trị đã cho |
| 8 | **void addIntHeader(String name, int value)**  Thêm một header phản hồi với tên và giá trị integer đã cho |
| 9 | **void flushBuffer()**  Force bất kỳ nội dung nào trong Buffer để được ghi tới Client đó |
| 10 | **void reset()**  Xóa bất dữ liệu nào tồn tại trong buffer cũng như Status code và các trường header |
| 11 | **void resetBuffer()**  Xóa nội dung của buffer nằm dưới trong phản hồi mà không xóa các trường header và status code |
| 12 | **void sendError(int sc)**  Gửi một phản hồi lỗi tới Client bởi sử dụng Status code đã cho và xóa buffer |
| 13 | **void sendError(int sc, String msg)**  Gửi một phản hồi lỗi tới Client bởi sử dụng Status code đã cho |
| 14 | **void sendRedirect(String location)**  Gửi một phản hồi Redirect tạm thời tới Client bởi sử dụng địc chỉ URL để Redirect đã cho |
| 15 | **void setBufferSize(int size)**  Thiết lập kích cỡ buffer cho thân phản hồi |
| 16 | **void setCharacterEncoding(String charset)**  Thiết lập mã hóa ký tự (MIME Charset) của phản hồi đang được gửi tới Client, ví dụ, thiết lập là UTF-8 |
| 17 | **void setContentLength(int len)**  Thiết lập độ dài nội dung của thân phản hồi trong HTTP Servlet, phương thức này thiết lập trường header là HTTP Content-Length |
| 18 | **void setContentType(String type)**  Thiết lập kiểu nội dung của phản hồi đang được gửi tới Client, nếu phản hồi vẫn chưa được commit |
| 19 | **void setDateHeader(String name, long date)**  Thiết lập một header phản hồi với tên và giá trị date đã cho |
| 20 | **void setHeader(String name, String value)**  Thiết lập một header phản hồi với tên và giá trị đã cho |
| 21 | **void setIntHeader(String name, int value)**  Thiết lập một header phản hồi với tên và giá trị integer đã cho |
| 22 | **void setLocale(Locale loc)**  Thiết lập Locale của phản hồi, nếu phản hồi vẫn chưa được commit |
| 23 | **void setStatus(int sc)**  Thiết lập Status code cho phản hồi này |

# **XI. HTTP Status Code trong JSP**

Định dạng của các thông báo HTTP request và HTTP response là tương tự nhau và sẽ có cấu trúc sau:

* Một dòng status + CRLF ( Carriage Return + Line Feed
* Bảng dưới liệt kê các HTTP Status code và các thông báo tương ứng mà có thể được trả về từ Web Server:

|  |  |  |
| --- | --- | --- |
| **Code** | **Thông báo** | **Miêu tả** |
| 100 | Continue | Chỉ một phần của yêu cầu đã được nhận bởi Server, nhưng chỉ cần nó không bị loại bỏ, Client nên tiếp tục với yêu cầu này |
| 101 | Switching Protocols | Server chuyển đổi giữa các giao thức |
| 200 | OK | Yêu cầu là OK |
| 201 | Created | Yêu cầu là hoàn thành, và một nguồn mới được tạo |
| 202 | Accepted | Yêu cầu được chấp nhận để xử lý, nhưng việc xử lý chưa hoàn thành |
| 203 | Non-authoritative Information |  |
| 204 | No Content |  |
| 205 | Reset Content |  |
| 206 | Partial Content |  |
| 300 | Multiple Choices | Một danh sách các link. Người dùng có thể chọn một link và tới trang đó. Tối đa là 5 địa chỉ |
| 301 | Moved Permanently | Trang được yêu cầu đã chuyển tới một URL mới |
| 302 | Found | Trang được yêu cầu đã chuyển tới một URL mới |
| 303 | See Other | Trang được yêu cầu có thể được tìm với một URL khác |
| 304 | Not Modified |  |
| 305 | Use Proxy |  |
| 306 | *Unused* | Code này được sử dụng trong một phiên bản trước. Nó không còn được sử dụng nữa, nhưng vẫn được lưu trữ |
| 307 | Temporary Redirect | Trang được yêu cầu đã tạm thời chuyển tới một URL mới |
| 400 | Bad Request | Server không hiểu yêu cầu |
| 401 | Unauthorized | Trang được yêu cầu cần thiết một username và một password |
| 402 | Payment Required | *Bạn không thể sử dụng code này nữa* |
| 403 | Forbidden | Sự truy cập là bị cấm với trang được yêu cầu |
| 404 | Not Found | Server không tìm thấy trang được yêu cầu |
| 405 | Method Not Allowed | Phương thức đã được xác định trong yêu cầu là không được cho phép |
| 406 | Not Acceptable | Server chỉ có thể tạo một phản hồi mà không được chấp nhận bởi Client |
| 407 | Proxy Authentication Required | Bạn phải xác nhận với một Proxy server trước khi yêu cầu này có thể được phục vụ |
| 408 | Request Timeout | Yêu cầu tốn thời gian quá lâu so với thời gian server đợi |
| 409 | Conflict | Yêu cầu này không thể hoàn thành bởi vì có một conflict |
| 410 | Gone | Trang được yêu cầu không có sẵn nữa |
| 411 | Length Required | "Content-Length" không được định nghĩa. Server sẽ không chấp nhận yêu cầu mà không có nó |
| 412 | Precondition Failed | Điều kiện tiên quyết (precondition) đã cung cấp trong yêu cầu được tính toán là false bởi Server |
| 413 | Request Entity Too Large | Server sẽ không chấp nhận yêu cầu này, bởi vì đối tượng yêu cầu là quá lớn |
| 414 | Request-url Too Long | Server sẽ không chấp nhận yêu cầu này, bởi vì URL là quá dài. Xảy ra khi bạn biến đổi một yêu cầu POST thành một yêu cầu GET với một thông tin truy vấn quá dài |
| 415 | Unsupported Media Type | Server sẽ không chấp nhận yêu cầu, bởi vì MIME type là không được hỗ trợ |
| 417 | Expectation Failed |  |
| 500 | Internal Server Error | Yêu cầu chưa được hoàn thành. Server gặp một trường hợp không mong muốn |
| 501 | Not Implemented | Yêu cầu chưa được hoàn thành. Server không hỗ trợ tính năng được yêu cầu |
| 502 | Bad Gateway | Yêu cầu chưa được hoàn thành. Server này nhận một phản hồi không hợp lệ từ Server tuyến trên |
| 503 | Service Unavailable | Yêu cầu chưa được hoàn thành. Server tạm thời đang quá tải hoặc hỏng |
| 504 | Gateway Timeout | Gateway là hết hạn |
| 505 | HTTP Version Not Supported | Server không hỗ trợ phiên bản "http protocol" |

## Phương thức để thiết lập HTTP Status Code

* Các phương thức này có sẵn với đối tượng *HttpServletResponse*.

|  |  |
| --- | --- |
| **STT** | **Phương thức & Miêu tả** |
| 1 | **public void setStatus ( int statusCode )**  Phương thức này thiết lập một Status code tùy ý. Phương thức setStatus nhận một int (là status code) như là một tham số. Nếu phản hồi của bạn bao gồm một Status code đặc biệt và một tài liệu, đảm bảo bạn đã gọi setStatus trươc khi thực sự trả về bất kỳ nội dung vào với *PrintWriter* |
| 2 | **public void sendRedirect(String url)**  Phương thức này tạo một phản hồi là 302 cùng với một *Location* header cung cấp URL của tài liệu mới đó |
| 3 | **public void sendError(int code, String message)**  Phương thức này gửi một Status code (thường là 404) cùng với một thông báo ngắn mà tự động được định dạng bên trong một tài liệu HTML và được gửi tới Client |

# **X) Filter trong JSP**

Servlet và JSP Filter là các lớp Java, có thể được sử dụng trong lập trình Servlet và JSP với các mục đích sau:

* Để chặn các yêu cầu từ một Client trước khi chúng truy cập một nguồn tại backend.
* Để thao tác các phản hồi từ Server trước khi chúng được gửi trở lại tới Client.

Sau đây chúng tôi đề nghị một số loại Filter sau:

* Authentication Filters − Các Filter xác nhận
* Data compression Filters − Các Filter nén dữ liệu
* Encryption Filters − Các Filter mật mã hóa
* Các Filter mà kích hoạt các sự kiện truy cập nguồn
* Image Conversion Filters − Các Filter biến đổi hình ảnh
* Logging and Auditing Filters − Các Filter kiểm tra log và thống kê
* MIME-TYPE Chain Filters.
* Tokenizing Filters .
* XSL/T Filter − Các Filter biến đổi nội dung XML

Các Filter được triển khai trong **web.xml** file và sau đó ánh xạ tới hoặc các tên Servlet hoặc các tên JSP hoặc các URL patter trong ứng dụng của bạn. Deployment Descriptor web.xml file có thể được tìm thấy trong thư mục *<Tomcat-installation-directory>\conf*.

Khi JSP container bắt đầu trình duyệt web của bạn, nó tạo một minh họa của mỗi Filter mà bạn đã khai báo trong Deployment Descriptor. Các Filter này thực thi theo thứ tự mà chúng đã được khai báo trong Deployment Descriptor đó.

## Các phương thức JSP Filter

Về cơ bản, một Filter là một lớp Java mà triển khai javax.servlet.Filter Interface. Interface này định nghĩa 3 phương thức:

|  |  |
| --- | --- |
| **STT** | **Phương thức và Miêu tả** |
| 1 | **public void doFilter (ServletRequest, ServletResponse, FilterChain)**  Phương thức này được gọi bởi container mỗi khi một cặp request/response được truyền thông qua chain. |
| 2 | **public void init(FilterConfig filterConfig)**  Phương thức này được triệu hồi bởi Web container để chỉ một Filter mà đang được đặt bên trong service |
| 3 | **public void destroy()**  Phương thức được gọi bởi Web container để chỉ một Filter bị hủy khỏi service |

# **XI) Redirect Page trong JSP**

Nói chung, Redirect page được sử dụng khi một tài liệu di chuyển tới một địa chỉ mới và chúng ta cần gửi Client tới vị trí mới này hoặc có thể vì cân bằng tải (load balancing) hoặc là do ngẫu nhiên.

Cách đơn giản nhất để Redirect một yêu cầu tới page khác là sử dụng phương thức **sendRedirect()** của đối tượng response. Sau đây là cú pháp của phương thức:

public void response.sendRedirect(String location)

throws IOException

Phương thức này gửi phản hồi trở lại trình duyệt cùng với status code và vị trí trang mới. Bạn cũng có thể sử dụng setStatus() và setHeader() cùng nhau để nhận được kết quả tương tự:

....

String site = "http://www.newpage.com" ;

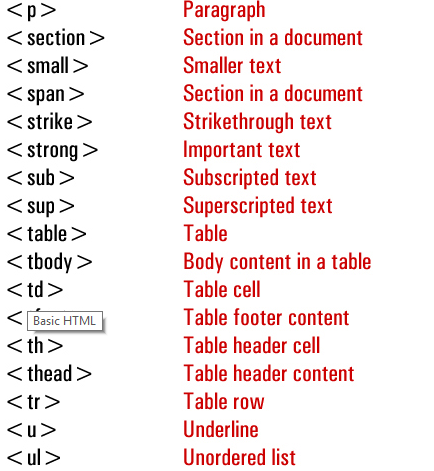
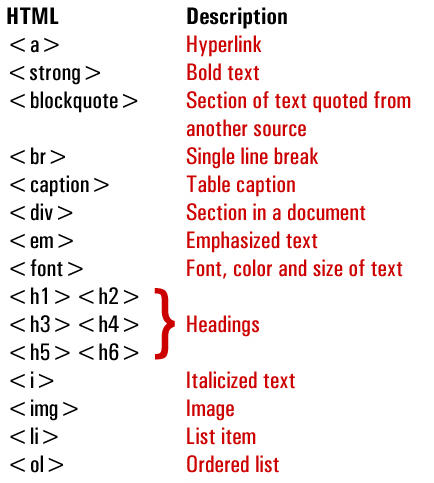
response.setStatus(response.SC\_MOVED\_TEMPORARILY);

response.setHeader("Location", site);

....

**VII) Programming in JSP**

**\*HTML(thẻ cơ bản)**

****

**\*Programming in JSP**

* JSP API bao gồm các lớp sau:

+Lớp ErrorData

+Lớp JSPWriter

+Lớp PageContext

**- Lớp ErrorData:**

+Xác định thông tin lỗi cho các trang lỗi.

+Đặt giá trị của chỉ thị trang, **isErrorPage** thành **true**, để chỉ ra rằng một trang là một trang lỗi.

+Chứa các phương thức khác nhau, đó là:

* getRequestURL (): Trả về URL được yêu cầu dưới dạng một chuỗi.
* SetServletName (): Trả về tên của servlet được gọi dưới dạng chuỗi.
* GetStatusCode (): Trả về mã trạng thái của lỗi dưới dạng một số nguyên.
* GetThrowable (): Trả về ngoại lệ Ném được gây ra lỗi.

**- Lớp JspWriter**:

+Viết dữ liệu hành động và mẫu trong một trang JSP.

+Đề xuất đối tượng của lớp JspWriter bằng biến ẩn.

Chứa các phương thức khác nhau, đó là:

* clear ()
* close ()
* flush ()
* getBufferSize ()
* print ()
* println ()

**- Lớp PageContext:**

+Cung cấp thông tin ngữ cảnh khi sử dụng JSP trong môi trường servlet.

+Chứa các phương thức khác nhau, đó là:

* Forward ()
* getPage ()
* getRequest ()
* getResponse ()
* getServletConfig ()
* getServletContext ()
* getSession ()
* include ()