HTTP (HyperText Transfer Protocol)

1. Giao thức HTTP

HTTP (RFC 1945, RFC 2616) là giao thức truyền các trang siêu văn bản, giúp trao đổi giữa web client (trình duyệt) và web server. HTTP chạy trên nền TCP/IP, dùng port TCP 80.

HTTP dùng mô hình Client/Server: client gửi yêu cầu (request) và server trả về đáp ứng (response):

- HTTP request chứa URL yêu cầu, lệnh HTTP (GET, POST, …), và các tham số request nếu có.

GET /index.html HTTP/1.1	**Request Line**	**HTTP Request**
Date: Thu, 20 May 2007 21:12:55 GMT Connection: close	**General Headers**	
Host: www.myfavorite.com From: joebloe@somewhere.com Accept: text/html, text/plain User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)	**Request Headers**	
	Entity Headers	
	Message Body	

- HTTP response chứa mã đáp ứng, kiểu nội dung content-type (MIME type), và nội dung được yêu cầu (HTML, image, …).

HTTP/1.1 200 OK	**Status Line**	**HTTP Response**
Date: Thu, 20 May 2007 21:12:58 GMT Connection: close	**General Headers**	
Server: Apache/1.3.27 Accept-Ranges: bytes	**Response Headers**	
Content-Type: text/html Content-Length: 170 Last-Modified: Tue, 18 May 2007 10:14:49 GMT	**Entity Headers**	
<html> <head> <title>Welcome to the Amazing Site!</title> </head> <body> This site is under construction. Come back later. Sorry! </body> </html>	**Message Body**	

HTTP là giao thức "stateless" (không lưu trạng thái). Nghĩa là, HTTP server không lưu trạng thái các request của *cùng một phiên giao dịch*, nên HTTP server xem chúng như các request mới không liên quan với nhau.

Các lệnh HTTP (màu trắng là an toàn, không thay đổi trên server):

Lệnh HTTP	Mô tả
GET	Yêu cầu lấy (GET) từ server với URL chỉ định các tài nguyên như: document, chart, web page, …
POST	Gửi (POST) thông tin (credit card, info, …) trong phần thân request đến URL chỉ định (thường là script).
HEAD	Giống GET, nhưng chỉ lấy phần header. Lấy thông tin từ URL chỉ định mà không cần lấy tài nguyên.
TRACE	Cho phép client nhận ngược lại bản sao request. Thường để dò lỗi.
OPTIONS	Yêu cầu danh sách lệnh HTTP mà URL chỉ định có thể đáp ứng, trả về trong một header.
PUT	Yêu cầu server lưu thông tin trong phần thân request tại URL chỉ định. Server không xử lý thông tin.
DELETE	Báo server xóa tài nguyên tại URL chỉ định.

Khi cần chuyển dữ liệu nhiều, GET hạn chế do một số trình duyệt giới hạn chiều dài URL. GET có thể bookmark trong khi POST không thể. POST chuyển thông tin bí mật (password) trong thân HTTP request nên bảo mật hơn.

Các trang siêu văn bản (hypertext) được viết bằng ngôn ngữ HTML. Tài liệu này không đề cập đến các kiến thức về HTML, javascript và CSS; nếu chưa biết, có thể tự học nhanh các kiến thức này tại: http://www.w3schools.com/

- Mã trạng thái (Status Code) trả về cung cấp thông tin về kết quả xử lý request. Nếu kết quả xử lý lỗi, web server đáp ứng bằng trang báo lỗi. Một số mã trạng thái:

Mã trả về	Thông điệp
1xx	Đã nhận request, tiếp tục xử lý.
2xx	Kết quả đáp ứng request.
3xx	Chuyển tiếp (Redirection).
301	Tài nguyên yêu cầu chuyển sang URL mới.
302	Tài nguyên yêu cầu tạm thời chuyển sang URL khác.
4xx	Lỗi do Client, cần các hành động tiếp để hoàn thành request.
400	Request bị sai, server không hiểu request.
401	Cần xác thực (username/password) khi truy cập tài nguyên (Unauthorized).
403	Tài nguyên yêu cầu không được quyền sử dụng.
404	Không tìm thấy tài nguyên yêu cầu.
5xx	Lỗi do Server.
500	Server bị lỗi bên trong.
502	Server nhận một response không hợp lệ từ servlet.

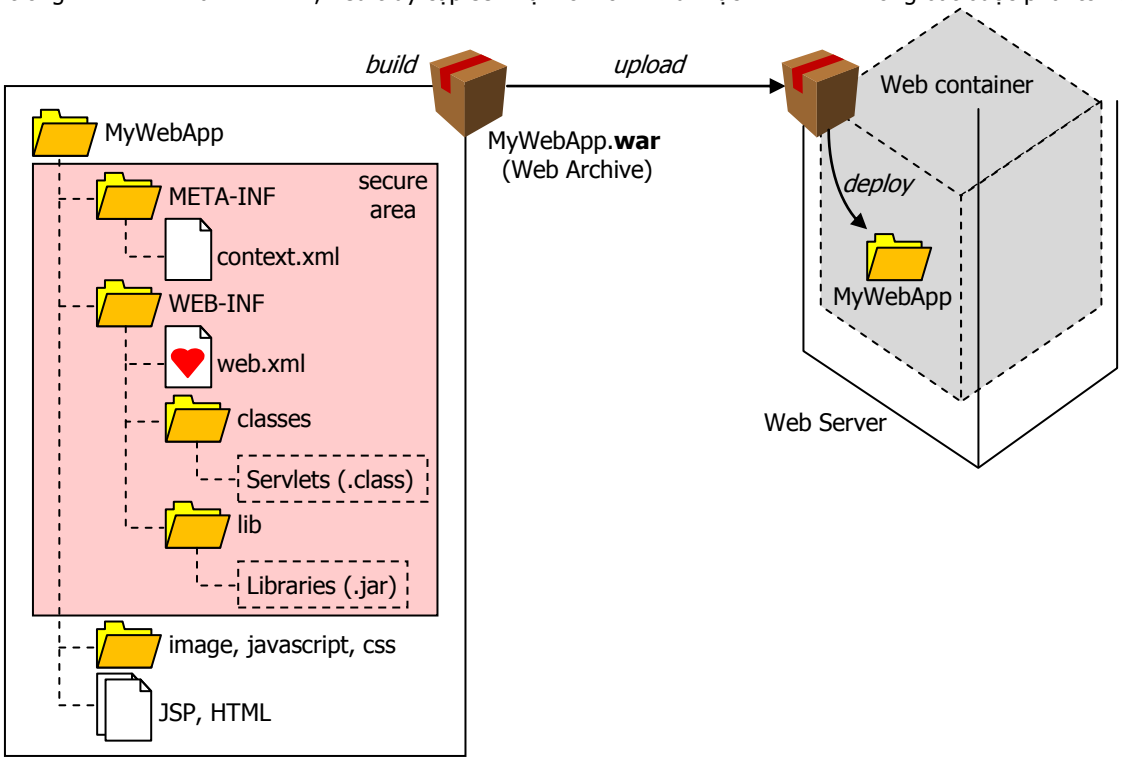
2. URL (Uniform Resource Locator)
URL được dùng để định vị đến tài nguyên. Cần hiểu rõ các thành phần của URL:

	Ví dụ	Ý nghĩa	Mô tả
host	http://	Giao thức	Báo server biết giao thức dùng trao đổi thông tin.
	www.domain.com	Tên miền	Tên miền của server, ánh xạ với địa chỉ IP bằng DNS.
	:80	Port dịch vụ	Cổng dịch vụ, thường là port well-known.
path	/folder/subfolder/	Đường dẫn	Đường dẫn đến tài nguyên. Dấu (/) đầu tiên là ROOT của ứng dụng.
	file.html	Tài nguyên	Tên của tài nguyên yêu cầu.
	?name=value	Tham số request	Các cặp tên-trị kèm theo HTTP request được chuyển đến server.

Một số phần không thấy trong URL, ví dụ:
- port: mặc định với port dịch vụ chuẩn cho giao thức chỉ định.
- tài nguyên: nếu ta không chỉ định, web server sẽ trả về trang mặc định, ví dụ chỉ định trong <welcome-file> của web.xml.

Cấu trúc một ứng dụng Web

1. Cấu trúc gói .WAR (Web ARchive)
J2EE quy định ứng dụng web được đóng gói thành gói **.WAR**, có tên là context[1] của ứng dụng. Bộ nạp lớp (class loader) sẽ nạp các lớp trong WEB-INF/classes trước, rồi đến các thư viện (.JAR) trong WEB-INF/lib. Người dùng không thể truy cập các thành phần trong META-INF và WEB-INF, nếu truy cập sẽ nhận lỗi 404. Thư mục WEB-INF không bắt buộc phải tồn tại.



Cấu trúc gói .WAR

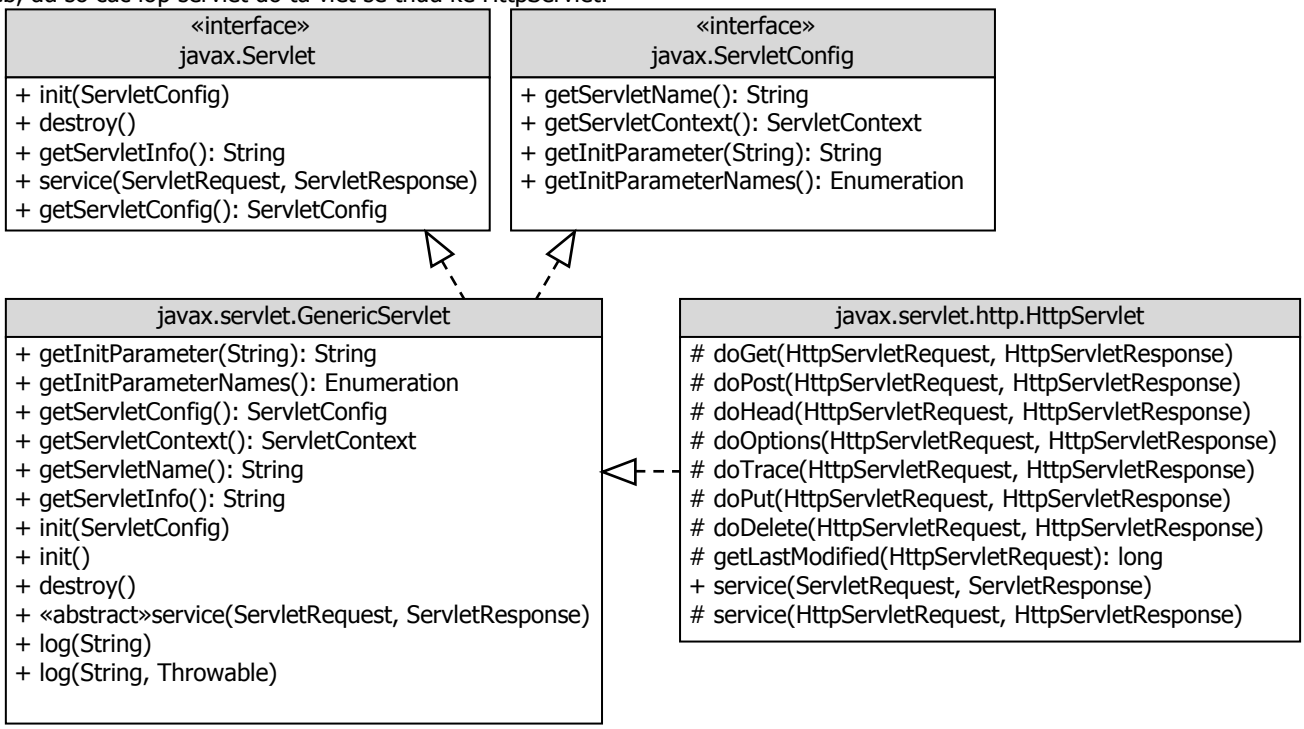
2. Web container
Web container chứa và quản lý các web component (JSP, servlet), có nhiệm vụ:
- Hỗ trợ truyền thông với web server.
- Quản lý vòng đời các component: nạp lớp, tạo thực thể, khởi tạo, triệu gọi dịch vụ, hủy thực thể.
- Hỗ trợ đa luồng (multithread): tự động tạo thread cho mỗi yêu cầu đến servlet. Web container quản lý một thread pool cho công việc này.
- Khai báo bảo mật: hỗ trợ bảo mật dễ dàng bằng khai báo, thuận tiện khi thay đổi bảo mật mà không cần code lại.
- Khai báo tài nguyên: cung cấp khai báo đến các tài nguyên bên ngoài (cơ sở dữ liệu, EJB).
- Hỗ trợ JSP: dịch (translation) trang JSP thành servlet.
Web container cung cấp hai đối tượng quan trọng cho servlet: request và response.
• Đối tượng request chứa thông tin do client chuyển đến Web server.
• Đối tượng response có một stream xuất, stream này lồng vào stream xuất của socket phía server. Ghi vào stream xuất của đối tượng response đồng nghĩa với chuyển dữ liệu ngược trở về client. Tại client, trình duyệt sẽ hiển thị dữ liệu chuyển về.

Servlet

1. Servlet
Servlet là đơn thể phía server, đáp ứng các yêu cầu do client chuyển đến. Các lớp servlet nói chung phải cài đặt giao diện javax.servlet.Servlet. Servlet API cung cấp hai lớp cài đặt từ giao diện này, ta phải thừa kế chúng để tạo ra servlet:

[1] Web server chia sẻ (share) một vùng đĩa với người dùng, vùng đĩa này có gốc là root (/) và chứa các context, được hiểu như "thư mục", chứa các ứng dụng web.

- **GenericServlet** cung cấp các chức năng cơ bản, độc lập giao thức, để tạo servlet. Thừa kế lớp này để tạo các lớp servlet cho dịch vụ không phải HTTP (non-HTTP). Khi thừa kế lớp này, ta cần viết lại phương thức service.
- **HttpServlet** là lớp trừu tượng thừa kế GenericServlet và thêm vào các chức năng cho riêng HTTP. Khi xây dựng ứng dụng web, đa số các lớp servlet do ta viết sẽ thừa kế HttpServlet.



Hai lớp servlet quan trọng: GenericServlet và HttpServlet

Hiểu rõ servlet giúp hiểu rõ JSP.

Cần chú ý:

- Cách ánh xạ giữa lớp servlet và URL gọi servlet trong tập tin web.xml, đây là tập tin mô tả triển khai (deployment descriptor, thường gọi tắt là DD):

```
<servlet>
  <servlet-name>ConvertServlet</servlet-name>
  <servlet-class>samples.ConvertServlet</servlet-class>
</servlet>                          lưu trữ trong /WEB-INF/classes
<servlet-mapping>                     nhưng path xem như từ /
  <servlet-name>ConvertServlet</servlet-name>
  <url-pattern>/ConvertServlet</url-pattern>
</servlet-mapping>          URI để gọi
```

liên kết với nhau thông qua <servlet-name>

- Khi phát triển HttpServlet, cần định nghĩa lại phương thức service() hoặc một hay nhiều phương thức xử lý yêu cầu tương ứng các lệnh HTTP được gọi (doGet, doPost, …).

2. Các phương thức xử lý yêu cầu của servlet

Tùy theo phương thức gọi là XYZ, Web container sẽ gọi phương thức service(HttpServletRequest, HttpServletResponse). Phương thức này sẽ điều hướng xử lý đến phương thức doXYZ(HttpServletRequest, HttpServletResponse) tương ứng. Nếu không có phương thức doXYZ tương ứng, client sẽ nhận lỗi 405.

• doGet: gọi từ URL (nhập URL vào thanh địa chỉ, click vào hyperlink chứa URL), form có method là GET (mặc định nếu form không có method). Tham số gửi bằng GET sẽ hiển thị trong URL.

• doPost: gọi từ form có method là POST. Tham số gửi bằng POST không hiển thị trong chuỗi URL.

Trong servlet do NetBeans tạo ra, doGet() và doPost() đều gọi phương thức **processRequest()**, trong phương thức này ta viết các xử lý yêu cầu của servlet.

• Các phương thức khác: doPut, doDelete, doHead, doOptions, doTrace

Mỗi phương thức trên nhận hai đối số sau:

• **HttpServletRequest**: thường có tên request

Dùng truy xuất thông tin HTTP header:

```
String HttpServletRequest.getHeader(String headerName)
Enumeration HttpServletRequest.get HeaderValues(String headerName)
```

Dùng truy xuất các thông tin do client chuyển đến, gọi là các tham số request:

```
String ServletRequest.getParameter(String paramName)
String[] ServletRequest.getParameterValues(String param)  // với tham số đa trị, ví dụ lấy từ các checkbox cùng tên
```

Dùng truy xuất tất cả cookie liên kết với request đó:

```
Cookie[] HttpServletRequest.getCookies()
```

• **HttpServletResponse**: thường có tên response

Dùng thiết lập Header trả về:

```
HttpServletResponse.setHeader(String headerName, String value)
HttpServletResponse.addHeader(String headerName, String value)
```

Dùng thiết lập kiểu nội dung Content-Type của HTTP response:

```
ServletResponse.setContentType(String value)
```

Lấy các stream xuất của response (byte hoặc character) để trả thông tin ngược về client:

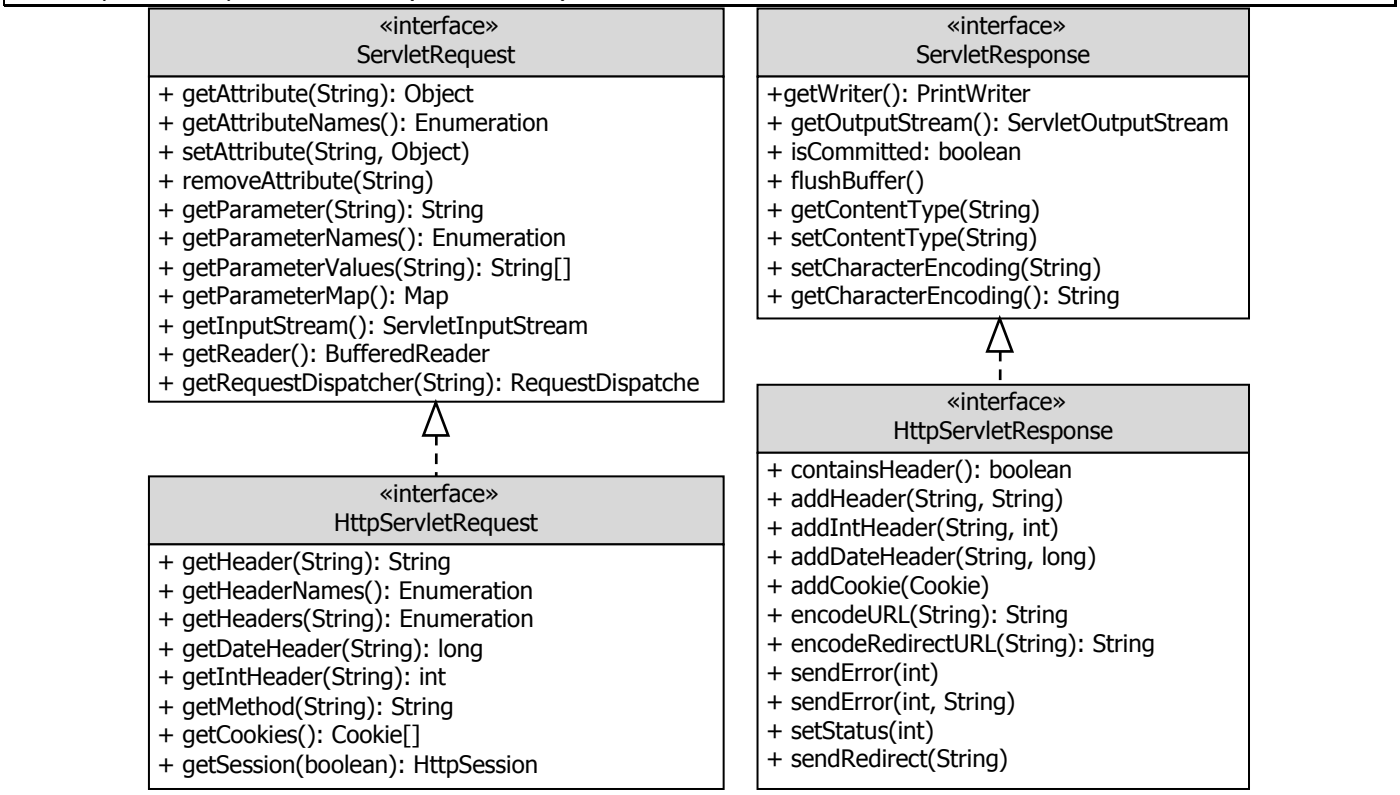
```
ServletOutputStream ServletResponse.getOutputStream()      // sau đó gọi write() của stream
PrintWriter ServletResponse.getWriter()                    // sau đó gọi println() của stream
```

Điều hướng đến trang khác hoặc trang lỗi:

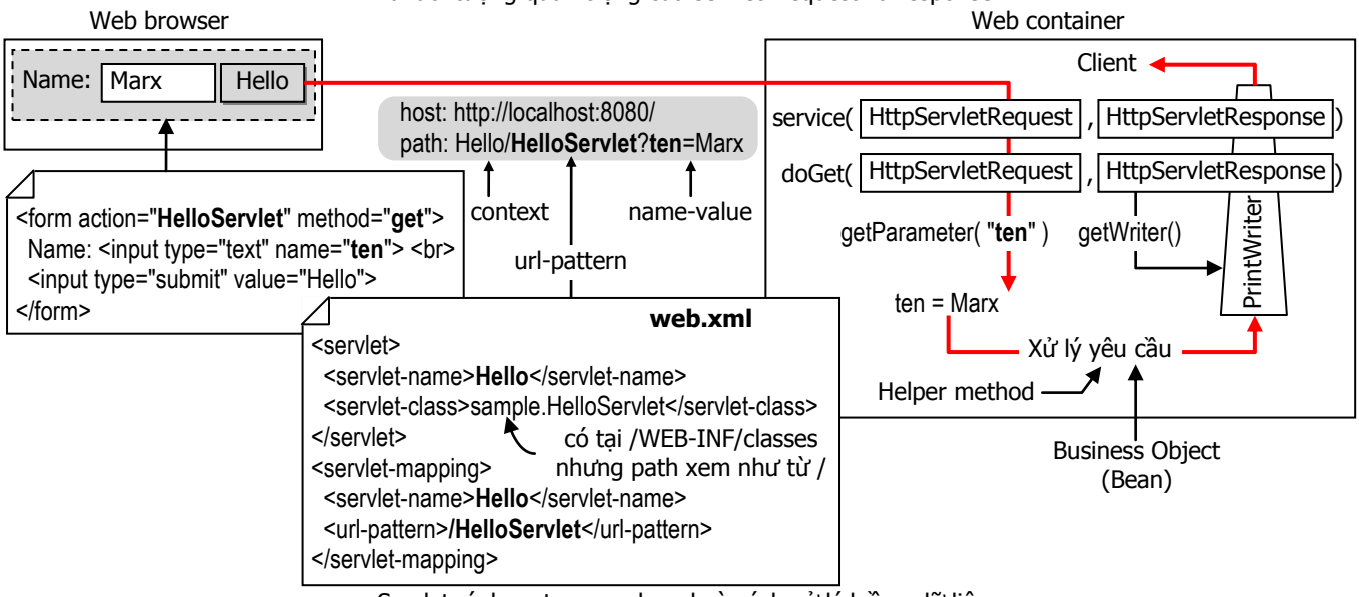
```
HttpServletResponse.sendRedirect(String newURL)
HttpServletResponse.sendError(int sc)                      // sc là mã lỗi, hằng số thuộc lớp HttpServletResponse
```

Dùng ghi cookie đến client:

```
void HttpServletResponse.addCookie(Cookie cookie)
```



Hai đối tượng quan trọng của servlet: request và response



Servlet, ánh xạ trong web.xml và cách xử lý luồng dữ liệu

3. Vòng đời của servlet

- Web container xem web.xml để xác định lớp servlet cần nạp (xem cặp <servlet> và <servlet-mapping>).
- Nạp lớp và tạo thực thể servlet (gọi constructor mặc định).
- Lần lượt gọi các phương thức:

init(ServletConfig) chỉ một lần khi khởi tạo servlet. Sau khi gọi có thể truy xuất ServletConfig (cho servlet) và ServletContext (cho ứng dụng).

service(ServletRequest, ServletResponse) Web container gọi phương thức service, trên một thread cho mỗi request

destroy() chỉ một lần khi hủy servlet.

4. Chuyển tiếp (forward) và điều hướng (redirect) trong servlet

Giao diện **RequestDispatcher** dùng để forward (hoặc include) đến trang khác. Do chuyển tiếp diễn ra trên server, nên URL trong thanh địa chỉ của trình duyệt sẽ không thay đổi.

• Gọi RequestDispatcher từ ServletContext: URI bắt buộc có dấu / ở đầu (đường dẫn tuyệt đối, / là root của ứng dụng).

```
RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/NextServlet");
dispatcher.forward(request, response);
```

hoặc:

```
RequestDispatcher dispatcher = getServletContext().getNamedDispatcher("/NextServlet");
dispatcher.forward(request, response);
```

• hoặc từ HttpServletRequest: URI không bắt buộc có dấu / ở đầu (đường dẫn tương đối).

```
RequestDispatcher dispatcher = request.getRequestDispatcher("NextServlet");
dispatcher.forward(request, response);
```

thường viết tắt:

```
request.getRequestDispatcher("NextServlet").forward(request, response);
```

Khi forward, đối tượng ServletRequest được servlet trước chuyển tiếp đến servlet sau. Nghĩa là servlet sau có thể *dùng chung biến tầm vực request* với servlet trước.

Đối tượng RequestDispatcher thiết lập một số thuộc tính cho đối tượng request, nên từ đối tượng request có thể lấy thông tin về hai servlet này, ví dụ context là MyWebApp, servlet đầu là FirstServlet. Sau khi forward:

- Đường dẫn liên quan NextServlet:

request.getRequestURI()	/MyWebApp/NextServlet
request.getContextPath()	/MyWebApp
request.getServletPath()	/NextServlet
request.getPathInfo()	null
request.getQueryString()	null

- Các thuộc tính của đối tượng request trong NextServlet:

request.getAttribute("javax.servlet.forward.request_uri")	/MyWebApp/FirstServlet
request.getAttribute("javax.servlet.forward.context_path")	/MyWebApp
request.getAttribute("javax.servlet.forward.servlet_path")	/FirstServlet
request.getAttribute("javax.servlet.forward.path_info")	null
request.getAttribute("javax.servlet.forward.query_string")	null

Nếu không cần chuyển tiếp ServletRequest cho servlet khác, ta có thể điều hướng (redirect) servlet. Do điều hướng diễn ra trên client, nên URL trong thanh địa chỉ của trình duyệt sẽ thay đổi:

```
response.sendRedirect("NextServlet");
```

Gửi dữ liệu đến response ngay trước khi chuyển tiếp hoặc điều hướng sẽ ném lỗi IllegalStateException.

ServletConfig và ServletContext

1. ServletConfig và tham số khởi tạo của servlet/JSP

Các tham số khởi tạo, là các cặp tên-trị (name-value) chỉ định một số biến khởi đầu cho một servlet/JSP cụ thể. Chúng có thể được dùng để tùy biến và điều khiển hành vi của servlet/JSP.

- Khai báo các tham số khởi tạo trong web.xml: dùng <init-param> trong tag <servlet> của servlet chỉ định. Không thiết lập bằng code.

```
<servlet>
  <servlet-name>ReadInitParamsServlet</servlet-name>
  <servlet-class>samples.ReadInitParametersServlet</servlet-class>
  <init-param>
    <param-name>emailHost</param-name>
    <param-value>151.68.167.201</param-value>
  </init-param>
</servlet>
<servlet-mapping>
  <servlet-name>ReadInitParamsServlet</servlet-name>
  <url-pattern>/ReadInitParametersServlet</url-pattern>
</servlet-mapping>

<servlet>
  <servlet-name>ReadInitParamJSP</servlet-name>
  <jsp-file>/readInitParamJSP.jsp</jsp-file>
  <init-param>
    <param-name>sys</param-name>
    <param-value>151.68.167.201</param-value>
  </init-param>
```

```
</servlet>
<servlet-mapping>
  <servlet-name>ReadInitParamJSP</servlet-name>
  <url-pattern>/readInitParamJSP.jsp</url-pattern>
</servlet-mapping>
```

- Web container sẽ đọc các tham số khởi tạo từ web.xml và chuyển cho ServletConfig. Từ servlet, *sau khi khởi tạo* bằng phương thức init(), dùng phương thức getInitParameter() của ServletConfig để lấy các tham số khởi tạo này.

```
public void init(ServletConfig config) throws ServletException {      // chỉ có một ServletConfig cho mỗi servlet
  String emailHost  = config.getInitParameter("emailHost");          // nơi truy xuất ServletConfig sớm nhất
}
```

nếu lấy từ phương thức khác:

```
public void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
  String host = getServletConfig().getInitParameter("emailHost");
}
```

Dùng phương thức getInitParameterNames() để lấy tất cả các tham số khởi tạo vào một Enumeration:

```
Enumeration params = config.getInitParameterNames();
while (params.hasMoreElements()) {
  String name = (String)params.nextElement();
  System.out.println("Name: " + name + " Value: " + config.getInitParameter(name));
}
```

Từ JSP: dùng đối tượng ẩn config, nạp chồng jspInit().

```
<%!
  String sys  = null;
  public void jspInit() {
    ServletConfig config = getServletConfig();
    emailHost  = config.getInitParameter("sys");
  }
%>
```

hoặc dùng scriptlet với đối tượng không tường minh config:

```
<%=config.getInitParameter("sys")%>
```

hoặc dùng EL:

```
${initParam.sys}
```

2. ServletContext và tham số khởi tạo của ứng dụng

«interface» ServletContext
+ getInitParameter(String): String + getInitParameterNames(): Enumeration + getAttribute(String): Object + getAttributeNames(): Enumeration + setAttribute(String, Object) + getRequestDispatcher(String): RequestDispatcher + getNamedDispatcher(String): RequestDispatcher + getResource(String): URL + getResourceAsStream(String): InputStream + getResourcePaths(String): String + getRealPath(String): String + getContextPath(): String + getServletContextName():String + getServletInfo(): String + getContext(String): ServletContext + log(String) + log(String, Throwable)

Các tham số khởi tạo của *toàn ứng dụng*, không phải của một servlet/JSP cụ thể nào, có thể lấy từ đối tượng ServletContext. Chỉ có một đối tượng ServletContext cho một ứng dụng Web trên một JVM.

- Khai báo các tham số khởi tạo trong web.xml: dùng <context-param> nằm ngoài tag <servlet>, không thiết lập bằng code.

```
<servlet>
  <servlet-name>initParameterServlet</servlet-name>
  <servlet-class>com.example.InitParameterServlet</servlet-class>
</servlet>
<context-param>
  <param-name>adminEmail</param-name>
  <param-value>admin@domain.com</param-value>
</context-param>
```

- Lấy ServletContext từ phương thức getServletContext() của servlet, rồi từ ServletContext lấy các tham số khởi tạo:

```
Enumeration en = getServletContext().getInitParameterNames();
while (en.hasMoreElements) {
  out.println(getServletContext().getInitParameter(en.nextElement()));
}
out.println(getServletContext ().getInitParameter("adminEmail"));
```

ServletContext cũng có thể lấy từ ServletConfig, dùng khi servlet không dẫn xuất từ GenericServlet hoặc HttpServlet.

```
getServletConfig().getServletContext()
```

3. ServletContextListener

ServletContextListener là lớp dùng lắng nghe hai sự kiện chính trong vòng đời của ServletContext:
- sự kiện khởi tạo: thường dùng để lấy các tham số khởi tạo rồi từ chúng thiết lập các biến tầm vực để truy xuất sau này.
- sự kiện hủy: thường dùng để đóng các kết nối, giải phóng tài nguyên.

Báo cho Web container biết dùng ServletContextListener:

```
<listener>
  <listener-class>samples.MyServletContextListener</listener-class>
</listener>
```

Trong lớp ServletContextListener, cần cài đặt hai phương thức đáp ứng sự kiện:
- contextInitialized(ServletContextEvent event)
- contextDestroyed(ServletContextEvent event)

Khi viết hai phương thức trên, ServletContextEvent được dùng để lấy tham chiếu đến ServletContext:

```
ServletContext servletContext = event.getServletContext();
```

Listener

Các Listener đi kèm theo các đối tượng quan trọng trong servlet để lắng nghe các sự kiện từ các đối tượng này. Các Listener thường cài đặt các phương thức sẽ được kích hoạt với sự kiện tương ứng. Các phương thức kích hoạt nhận tham số là đối tượng Event tương ứng. Chúng sẽ dùng thông tin lấy được từ đối tượng Event được truyền để thực hiện các tác vụ cần thiết.

Tình huống	Giao diện Listener và các phương thức sẽ kích hoạt	Lớp Event
Lắng nghe sự kiện thêm vào, loại bỏ hoặc thay thế một **attribute** tầm vực **context** (tức ứng dụng)	javax.servlet. **ServletContextAttributeListener** • attributeAdded • attributeRemoved • attributeReplaced	ServletContextAttributeEvent
Dùng khi muốn biết **context** được tạo ra hay bị hủy.	javax.servlet.**ServletContextListener** • contextInitialized • contextDestroyed	ServletContextEvent
Dùng khi muốn theo dõi các **session** đang kích hoạt.	javax.servlet.http.**HttpSessionListener** • sessionCreated • sessionDestroyed	HttpSessionEvent
Dùng khi **session** di trú từ JVM này đến JVM khác, trong môi trường clustering.	javax.servlet.http.**HttpSessionActivationListener** • sessionDidActivate • sessionWillPassivate	ServletContextEvent
Lắng nghe sự kiện thêm vào, loại bỏ hoặc thay thế một **attribute** tầm vực **session**.	javax.servlet.http.**HttpSessionAttributeListener** • attributeAdded • attributeRemoved • attributeReplaced	HttpSessionBindingEvent
Dùng khi muốn biết một Bean được lưu trữ hoặc loại bỏ khỏi **session**: <jsp:useBeean scope="session" />	javax.servlet.http.**HttpSessionBindingListener** • valueBound • valueUnbound	HttpSessionBindingEvent
Dùng khi muốn biết mỗi lần có **request** đến, thường để ghi nhận lại.	javax.servlet.**ServletRequestListener** • requestInitialized • requestDestroyed	ServletRequestEvent
Lắng nghe sự kiện thêm vào, loại bỏ hoặc thay thế một **attribute** tầm vực **request**.	javax.servlet. **ServletRequestAttributeListener** • attributeAdded • attributeRemoved • attributeReplaced	ServletRequestAttributeEvent

Các lớp Listener cũng đặt trong WEB-INF/classes của gói .WAR. Chúng được khai báo trong DD bằng cách dùng tag <listener>, tag này chứa nhiều tag <listener-class> liệt kê theo thứ tự được triệu gọi.

```
<listener>
  <listener-class>listeners.ContextListener</listener-class>
</listener>
```

Attribute và Scope

1. Attribute

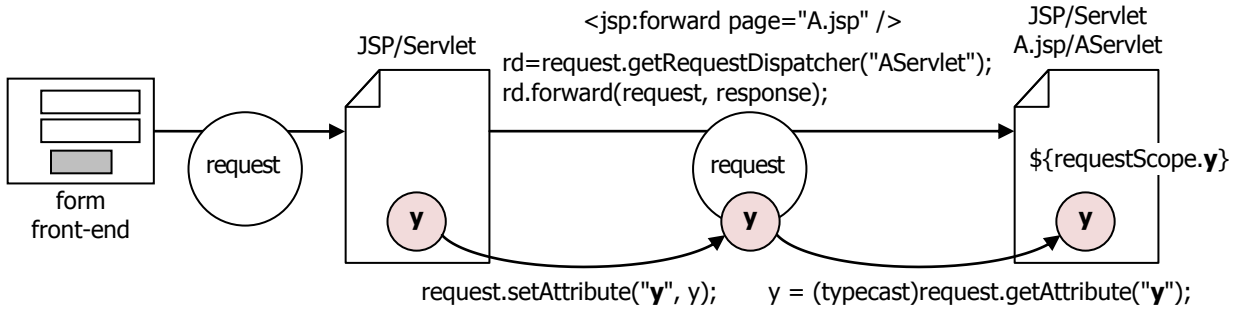
Attribute là một biến liên kết (bound) với một tầm vực dùng trong ứng dụng web. Tầm vực (scope) là phạm vi có thể truy xuất biến này trong ứng dụng. Trong tầm vực bất kỳ, để truy xuất attribute, ta dùng các phương thức:

```
Object getAttribute(String name)                    // đừng quên ép kiểu đối tượng trả về
void setAttribute(String name, Object value)
```

```
void removeAttribute(String attributeName)
Enumeration getAttributeNames()
```

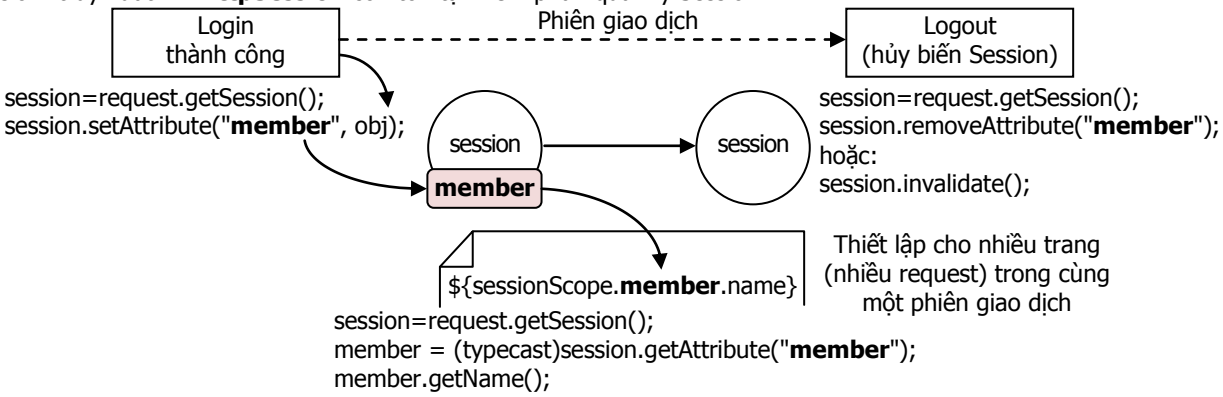
2. Các tầm vực (Scope)

- page: tồn tại và truy xuất được trong một servlet hay một trang JSP. Không có ý nghĩa nhiều vì tầm vực quá hẹp.
- request: truy xuất khi **ServletRequest** còn tồn tại. Thường dùng giữa các trang chuyển tiếp bằng RequestDispatcher.



Biến tầm vực request, là một thuộc tính của đối tượng request

- session: truy xuất khi **HttpSession** còn tồn tại. Xem phần quản lý Session.



Biến tầm vực session, là một thuộc tính của đối tượng session

Biến thuộc tầm vực session có thể không an toàn thread khi người dùng có hơn một cửa sổ trình duyệt dùng chung một session. Cách giải quyết, đồng bộ hóa trên HttpSession:

```
HttpSession session = request.getSession();
synchronized(session) {
  session.setAttribute("attributeName", "attributeValue");
  out.println(session.getAttribute("attributeName"));
}
```

- application (context): truy xuất mọi nơi trong ứng dụng, không an toàn thread do nhiều servlet trong nhiều thread có thể cùng truy xuất biến thuộc tầm vực này. Không có ý nghĩa nhiều vì tầm vực quá dài.

Quản lý Session

1. Các phương thức của HttpSession

Một session bao gồm nhiều request có liên quan, các request này tạo thành một phiên giao dịch.

Ví dụ về phiên giao dịch: shopping (nhiều request chọn/trả hàng), quiz (nhiều request chọn cho mỗi câu trắc nghiệm), forum (nhiều request chọn các trang trong một đợt truy cập).

Do HTTP là giao thức "stateless", cần biến tầm vực session để theo dõi các request của cùng một phiên giao dịch.

Đối tượng session là một thực thể của lớp cài đặt giao diện javax.servlet.http.HttpSession. Trạng thái của người dùng được lưu như một thuộc tính của biến session, bằng cách sử dụng các phương thức sau:

Phương thức	Mô tả
public String getId()	Trả về chuỗi chứa định danh duy nhất của session.
public ServletContext getServletContext()	Trả về ServletContext của session.
public boolean isNew()	Trả về true nếu server có tạo session và người dùng chưa truy xuất đến session đó.
public long getCreationTime()	Trả về thời điểm session được tạo, tính bằng miligiây kể từ 0h00 1/1/2003 GMT.
public long getLastAccessedTime()	Trả về thời điểm truy nhập (request) cuối của session, tính bằng miligiây kể từ 0h00 1/1/1970 GMT.
public int getMaxInactiveInterval()	Trả về số giây server sẽ chờ trước khi session bị hủy (thời gian timeout). Nếu trả về số âm, session sẽ không timeout.
public void setMaxInactiveInterval(int interval)	Chỉ định số giây server sẽ chờ trước khi session bị hủy. Nếu thiết lập số âm, session sẽ không timeout.
public void invalidate()	Hủy session cùng các biến trong nó, giống setMaxInactiveInterval(0).
public java.util Enumeration getAttributeNames()	Trả về tập hợp "tên" tất cả các đối tượng có trong session.

public Object getAttribute(String *name*)	Trả về đối tượng liên kết với *name* chỉ định trong session này (biến tầm vực session) hoặc null nếu không tìm thấy.
public void setAttribute(String name, Object value)	Kết nối một Object với một *name* chỉ định (gọi là attribute). Nếu *name* đã tồn tại thì Object được truyền này sẽ thay cho Object trước.
public void removeAttribute(String name)	Loại đối tượng liên kết với *name* chỉ định ra khỏi session.

Cách thuận tiện nhất để có đối tượng HttpSession là:

```
request.getSession();
```

getSession() và getSession(true) giống nhau, đối tượng HttpSession sẽ được tạo nếu nó không tồn tại sẵn. getSession(false) sẽ trả về null nếu không có đối tượng HttpSession nào tồn tại sẵn.
Ngoài ra, có thể lấy đối tượng HttpSession từ phương thức của HttpSessionListener thông qua đối tượng HttpSessionEvent.

```
public void sessionCreated(HttpSessionEvent event) {
  HttpSession session = event.getSession();
}
```

2. Cookie, Session ID và URL Rewriting

Cookie là thông tin do server gửi đến client, client lưu dưới dạng tập tin văn bản. Sau đó, chuỗi request của cùng một phiên giao dịch gửi từ client sẽ được server nhận dạng, vì cookie nhận được sẽ có mặt trong từng request:
- Trong header của HTTP response đầu tiên, server gửi: Set-Cookie: username=JohnSmith
- Trong header của chuỗi HTTP request kế tiếp, client gửi: Set-Cookie: username=JohnSmith

Các phương thức truy xuất cookie:

```
Cookie[] HttpServletRequest.getCookies()
HttpServletResponse.addCookie()
```

Các phương thức của cookie:

```
Cookie cookie = new Cookie("cookieName", cookieValue);      // tên Cookie không chứa space và [ ] ( ) = , " / ? @ : ;
getDomain() và setDomain(String)          // tên miền cấp cookie
getMaxAge() và setMaxAge(int)             // thời gian sống của cookie, tính bằng giây, ví dụ: 60*60*24 cho một ngày
getPath() và setPath(String)              // vị trí trong context, nơi yêu cầu client phải trình cookie
getValue() và setValue(String)
getSecure() và setSecure(boolean)         // dùng SSL, HTTPS
String getName()                          // ASCII, không ";", " ", không bắt đầu với $
```

Thời gian sống bằng 0, cookie sẽ bị hủy ngay. Thời gian sống bằng -1, cookie sẽ bị hủy khi thoát trình duyệt.
Ví dụ về truy xuất cookie:

```
Cookie[] cookies = request.getCookies();
for (int i = 0; i < cookies.length; ++i) {
  Cookie cookie = cookies[i];
  if (cookie.getName().equals("cookieName")) {
    out.println(cookie.getValue());
    break;
  }
}
```

Khi Session được tạo, một session ID tương ứng được sinh ra. Có ba cách theo dõi Session:
- Cookie: session ID mới tạo sẽ lưu trên client dưới dạng Cookie (MaxAge = -1), có tên JSESSIONID. Các request tiếp theo của client phải chứa session ID này, để server theo dõi Session.

```
Set-Cookie: JSESSIONID=49EBBB19A1B2F8D10EE075F6F14CB8C9; Path=/
```

- URL rewriting: dùng khi Cookie bị cấm trên client. Lúc này, tất cả URL của ứng dụng được gắn kèm theo session ID. Ví dụ:

```
http://www.domain.com:80/folder/subfolder/file.jsp;jsessionid=12345
```

• Gắn kèm session ID đến URL.

```
HttpServletResponse.encodeURL(String url)
```

• Gắn kèm session ID lên URL để chuyển cho phương thức response.sendRedirect(String url).

```
HttpServletResponse.encodeRedirectURL(String url)
```

- SSL information: SSL có cơ chế phân biệt nhiều request của cùng một client, xem như một phần của phiên giao dịch.

3. Kết thúc Session

Có ba cách kết thúc Session:
- Dùng phương thức invalidate() của HttpSession.
- Hết thời gian (timeout).

Thiết lập timeout bằng code, thời gian *tính bằng giây*:

```
HttpSession.setMaxInactiveInterval(int)
```

Thiết lập timeout bằng khai báo trong web.xml, thời gian *tính bằng phút*:

```
<session-config>
  <session-timeout>30</session-timeout>
</session-config>
```

Trị timeout bằng 0 hoặc nhỏ hơn 0, Session sẽ *không timeout* cho đến khi gọi invalidate().

HttpSessionBindingListener được dùng bởi các đối tượng muốn nhận cảnh báo khi thêm hoặc loại biến khỏi Session. Nó có hai phương thức:

```
void valueBound(HttpSessionBindingEvent e)
void valueUnound(HttpSessionBindingEvent e)
```

- Đóng trình duyệt để hủy Sesssion ID lưu phía client.

Filter và Wrapper

1. Filter

Filter là các component cho phép chặn đón request TRƯỚC khi nó được gửi đến servlet, hoặc chặn đón response SAU khi servlet hoàn thành kết xuất nhưng TRƯỚC khi response gửi trở về client.

Filter thường dùng trong mẫu thiết kế Intercepting Filter, có chức năng đa dạng: xác thực, nén dữ liệu, mã hóa dữ liệu, kích hoạt truy cập tài nguyên, ghi nhận kiểm tra, bộ lọc dùng XSLT cho XML.

Mục tiêu:

- Chặn đón request
  - Kiểm tra bảo mật, kiểm tra dữ liệu hợp lệ (validator)
  - Định dạng lại request (header + body)
  - Ghi nhận (log), kiểm soát (audit) request
- Chặn đón response
  - Nén dữ liệu trong stream trả về
  - Nối thêm hoặc thay đổi stream trả về
  - Tạo một response hoàn toàn khác

Các filter có thể chạy kế tiếp nhau thành chuỗi (filter chain).

Filter viết giống như một servlet:

```
public class ExampleFilter implements javax.servlet.Filter {
  private FilterConfig config;
  public void init(FilterConfig config) throws ServletException {
  this.config = config;
}

public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
    throws ServletException, IOException {
  HttpServletRequest httpRequest = (HttpServletRequest) request;
  …
  chain.doFilter(request, response);  // gọi filter kế hoặc servlet
}

public void destroy() {
    …
  }
}
```

Filter cũng có các tham số khởi tạo (init parameter), khai báo trong web.xml và truy cập thông qua các phương thức getInitParameter(String), getInitParameterNames() của đối tượng javax.servlet.FilterConfig.

Filter có thể lấy tham chiếu đến đối tượng ServletContext bằng phương thức getServletContext().

2. Khai báo filter

Web container quyết định gọi filter dựa trên khai báo trong web.xml. Người triển khai sẽ ánh xạ filter với URL pattern của request tương ứng trong web.xml. Khai báo một filter:

```
<filter>
  <filter-name>ExampleFilter</filter-name>
  <filter-class>com.example.ExampleFilter</filter-class>
    <init-param>
      <param-name>userLogFile</param-name>
      <param-value>userLog.txt</param-value>
    </init-param>
  </filter>
</web-app>
```

Ánh xạ filter với một servlet hoặc JSP mà nó chặn đón:

```
<filter-mapping>
  <filter-name>ExampleFilter</filter-name>
  <servlet-name>ExampleServlet</servlet-name>
  (hoặc)
  <url-pattern>*.jsp</url-pattern>
</filter-mapping>
```

Ngoài ra, thứ tự của <filter-mapping> trong web.xml xác định thứ tự Web container áp dụng filter cho servlet.

3. Vòng đời filter

Giống như servlet, Web container quản lý vòng đời của filter.

init() → doFilter() → destroy()
Các phương thức trong vòng đời của filter:
• init(FilterConfig): được gọi bởi container khi khởi động ứng dụng.
• doFilter(ServletRequest, ServletResponse): được gọi bởi container cho mỗi request có URL ánh xạ với filter này.
• destroy(): được gọi bởi container khi kết thúc ứng dụng.

4. Wrapper
Filter có thể hiệu chỉnh lại request và thay đổi response. Để thực hiện điều này, filter dùng các lớp Wrapper:
- Gói javax.servlet định nghĩa lớp ServletRequestWrapper và ServletResponseWrapper.
- Gói javax.servlet.http định nghĩa lớp HttpServletRequestWrapper và HttpServletResponseWrapper.
Đối tượng thuộc lớp thừa kế 4 lớp này có constructor nhận request và response làm đối số. Điều này cho phép chúng ta viết đè các phương thức của request và response. Ví dụ như hiệu chỉnh lại header của request hoặc lồng thêm stream vào stream xuất của response.

Security

1. Các cơ chế bảo mật
- Authentication (xác thực): xác thực người dùng là người đăng ký hợp lệ, thực hiện bằng cách hỏi username/password.
- Authorization (phân quyền): xác thực người dùng có quyền truy cập tài nguyên yêu cầu, thực hiện bằng cách kiểm tra quyền người dùng trong ACL (Access Control List).
- Data integrity (toàn vẹn dữ liệu): bảo đảm dữ liệu truyền không bị thay đổi, thực hiện bằng digest.
- Confidentiality (bí mật giữ liệu): bảo đảm người không liên quan không sử dụng được dữ liệu, thực hiện bằng mã hóa.
Thiết lập bảo mật được thực hiện trong tập tin web.xml.

2. Ràng buộc bảo mật (<security constraint>)
Cần định nghĩa:
- <web-resource-collection> (ít nhất một) tham chiếu đến các nguồn tài nguyên cần được bảo vệ, bao gồm nhiều bộ {<web-resource-name>, <url-pattern>, <http-method>}.
- <auth-constraint> (tùy chọn) tham chiếu đến các vai trò (<role-name>) được truy cập tài nguyên cần bảo vệ.
- <user-data-constraint (tùy chọn) chỉ định lớp bảo vệ. NONE là không bảo vệ (HTTP), INTEGRAL là bảo đảm toàn vẹn dữ liệu (HTTPS), CONFIDENTIAL bảo đảm bí mật dữ liệu (HTTPS).

3. Cấu hình đăng nhập (<login-config>)
Cần định nghĩa phương thức xác thực (<auth-method>) và các tag có liên quan:
Các phương thức xác thực:
- BASIC: lỗi 401 khi truy cập tài nguyên cần xác thực sẽ được trả về trình duyệt, lỗi này kích hoạt trình duyệt bật *form đăng nhập mặc định*. Thông tin đăng nhập username/password được mã hóa Base64 và gửi cho server.
Ưu điểm:
• Dễ thiết lập.
• Hỗ trợ bởi mọi trình duyệt.
Khuyết điểm:
• Không bảo mật do username/password không mã hóa.
• Hộp thoại login tùy theo trình duyệt.
• Không logout được.
- DIGEST: khi client yêu cầu đăng nhập, server gửi trị nonce (number once) "thử thách". Client xác thực bằng cách gửi hash(userid, password, nonce) thay vì gửi (userid, password) dưới dạng rõ.
Ưu điểm:
• Bảo mật.
Khuyết điểm:
• Không được hỗ trợ bởi mọi trình duyệt.
- FORM: lỗi 401 khi truy cập tài nguyên cần xác thực sẽ điều hướng đến form quy định, với tên form (j_security_check), tên user (j_username) và password (j_password) bắt buộc. Thông tin đăng nhập username/password được mã hóa Base64 và gửi cho server.
Ưu điểm:
• Dễ thiết lập.
• Hỗ trợ bởi mọi trình duyệt.
• Tùy biến được form dùng login.
Khuyết điểm:
• Không bảo mật do username/password không mã hóa, nếu không dùng HTTPS.
- CLIENT-CERT: thông tin truyền được mã hóa bất đối xứng, sử dụng public key trao đổi trong certificate.
Ưu điểm:
• Rất bảo mật.
• Hỗ trợ bởi mọi trình duyệt.
Khuyết điểm:
• Tốn chi phí triển khai PKI cho public key.

4. Vai trò bảo mật (<security-role>)
Cần định nghĩa <role-name>.

Servlet có thể tham chiếu <role-name> dưới một tên khác. Ví dụ sau khai báo một role thực là supervisor, nhưng servlet tham chiếu trong code với tên là manager.

```
<security-role>
  <role-name>supervisor</role-name>
</security-role>
<servlet>
  <servlet-name>SecureServlet</servlet-name>
  <servlet-class>samples.SecureServlet</servlet-class>
  <security-role-ref>
    <role-name>manager</role-name>
    <role-link>supervisor</role-link>
  </security-role-ref>
</servlet>
```

5. Trang báo lỗi

Các lỗi HTTP được web server báo bằng các trang lỗi chuẩn. Có thể thay các trang này bằng trang riêng, khai báo trong DD web.xml:

```
<error-page>
  <error-code>404</error-code>
  <location>/pageNotFound.jsp</location>
</error-page>
```

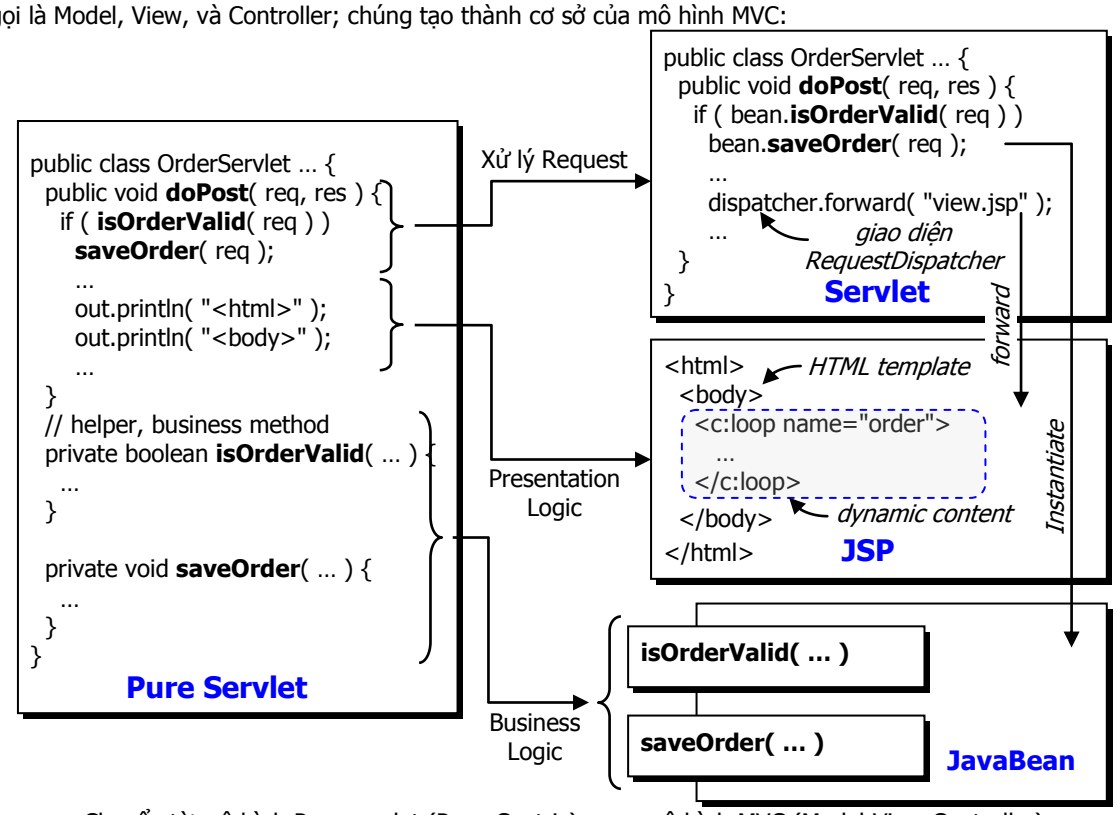
Cũng có thể dùng khai báo trên để báo các lỗi exception bằng trang riêng:

```
<error-page>
  <exception-type>exception.BookNotFoundException</exception-type>
  <location>/errorPage.html</location>
</error-page>
```

Mô hình MVC

1. Mô hình MVC

Ý tưởng là tách phần hiển thị dữ liệu và phần xử lý dữ liệu, thêm thành phần để phối hợp chúng với nhau. Cả ba thành phần này được gọi là Model, View, và Controller; chúng tạo thành cơ sở của mô hình MVC:



Chuyển từ mô hình Pure servlet (Page Centric) sang mô hình MVC (Model-View-Controller)

• Model – giữ dữ liệu và trạng thái của ứng dụng, thực hiện các thao tác nghiệp vụ cho ứng dụng, phổ biến là truy xuất cơ sở dữ liệu. Dữ liệu từ Model được Controller chuyển đến View để hiển thị. Model thường là bean, phát triển thành EJB trong kiến trúc phân tán.

• View – chứa thao tác trình bày (presentation logic) phía server. View hiển thị dữ liệu chứa trong Model cho người dùng. View cũng cho phép người dùng tương tác với hệ thống và thông báo cho Controller các tương tác của người dùng. View thường là JSP, được hỗ trợ bởi các thư viện tag và EL. Thao tác trình bày nói một cách tóm tắt là dàn trang (layout): kết hợp giữa template HTML tĩnh và dữ liệu động lấy từ Model.

• Controller – quản lý toàn bộ luồng xử lý. Controller khởi tạo Model và View, liên kết View với Model. Tùy theo yêu cầu của của ứng dụng, Controller có thể khởi tạo nhiều View và kết hợp chúng với cùng một Model. Controller nhận tương tác của người dùng và điều khiển Model thích hợp để đáp ứng. Controller thường là servlet.
Nhiều framework phát triển ứng dụng web xuất phát từ mô hình MVC, như : Struts (tập tring phần Controller), Spring, JSF (tập trung phần View).

2. Luồng xử lý trong mô hình MVC
- Nhận request: servlet (Controller) nhận request từ form front-end, lấy thông tin của client do request chuyển đến. Sau đó, servlet:
• tạo Bean.
• thiết lập các thông tin nhận được cho Bean (Model) bằng setter của Bean.
• gọi phương thức nghiệp vụ của Bean.
- Xử lý request: Bean (Model) xử lý tập trung các request. Bean là Business Object, chuyên xử lý nghiệp vụ.
- Trả kết quả: kết quả được servlet chuyển tiếp (forward) đến JSP, JSP (View) thực hiện dàn trang (layout) kết quả, bao gồm 2 thành phần:
• Template HTML tĩnh (khó thực hiện linh hoạt với out.println() của servlet): chứa HTML, XML, CSS, ...
• Dữ liệu động: do Bean trả về, thường chứa trong các POJO theo mẫu hiết kế Transfer Object.
Business Logic: Bean, thành phần có thể dùng lại được. Có thể nhận kết xuất từ Bean bằng 2 cách:
• Servlet đưa kết quả mà Bean trả về vào biến có tầm vực request hay session rồi chuyển đến trang JSP, JSP truy xuất dữ liệu từ các biến tầm vực này.
• JSP gọi trực tiếp Bean từ tag hoặc EL, Bean phải có getter/setter tương ứng.
Presentation Logic: JSP là Presentation logic phía server. Phần servlet (Controller) của MVC có thể viết bằng JSP nhưng không ý nghĩa vì JSP cũng chuyển sang servlet, nên viết bằng servlet sẽ thực hiện nhanh hơn.

JSP (Java Server Page)

1. Các giai đoạn xử lý JSP
JSP là ngôn ngữ kịch bản phía server, thực chất JSP sẽ được chuyển thành servlet và biên dịch rồi mới thực thi được.

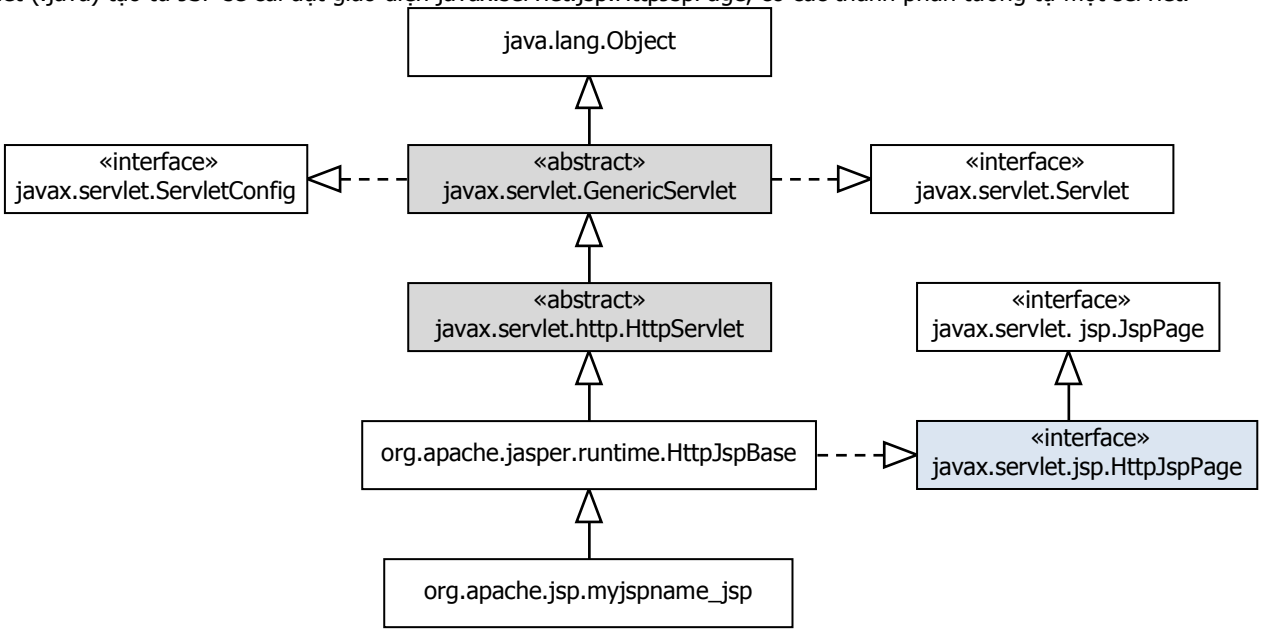
```
<servlet>
  <servlet-name>convert</servlet-name>
  <jsp-file>/convert.jsp</jsp-file>
</servlet>
<servlet-mapping>
  <servlet-name>convert</servlet-name>
  <url-pattern>/Convert</url-pattern
</servlet-mapping>
```

trang JSP thật sự được gọi khi dùng URL ánh xạ

liên kết với nhau thông qua <servlet-name>

URI để gọi

Bản chất JSP là XML, cho nên tên tag, các thuộc tính, các trị, … phân biệt chữ hoa/chữ thường.
Ba giai đoạn xử lý trang JSP:
- Giai đoạn 1: page translation, "dịch" trang JSP. JSP sẽ được chuyển (parse – phân tích cú pháp) từ tập tin kịch bản (.jsp) thành tập tin nguồn servlet (.java):
• Directive trong JSP cho biết servlet sẽ được tạo ra như thế nào: import, extends, ngôn ngữ sử dụng, ...
• Scriptlet, action (tag), biểu thức EL: chuyển thành code trong **_jspService()**
Servlet (.java) tạo từ JSP sẽ cài đặt giao diện javax.servlet.jsp.HttpJspPage, có các thành phần tương tự một servlet.



Để xem code servlet sinh ra từ JSP, click phải lên trang JSP trong tab Projects, chọn View Servlet từ menu tắt.
- Giai đoạn 2: page compilation, biên dịch trang. Servlet sinh ra từ trang JSP được biên dịch (.class). Nếu dịch bị lỗi, client sẽ nhận được lỗi 500 (Server Error).
- Giai đoạn 3: execution, thực thi trang JSP. Thực chất servlet (.class) tương ứng được thực thi. Quá trình thực thi như sau:

Nạp lớp → Tạo thực thể (instance) → Gọi jspInit() → Gọi _jspService() → Gọi jspDestroy()
Trong lần gọi sau, servlet (.class) sẽ thực thi, hai giai đoạn trước chỉ thực hiện lại khi trang JSP thay đổi.

2. Vòng đời của JSP
jspInit() và **jspDestroy()** chỉ gọi một lần, không tham số, không exception, có thể nạp chồng.
Định nghĩa trong giao diện javax.servlet.jsp.JspPage.
Nạp chồng jspInit(): trong trang JSP, nếu muốn nạp chồng jspInit() của servlet sinh từ JSP, viết:

```
<%! public void jspInit() {
        ServletConfig config = getServletConfig();
        String parameterValue = config.getInitParameter("parameterName ");
        ServletContext context = getServletContext();
        context.setAttribute("parameterName", parameterValue);
      }
%>
```

vì jspInit() được gọi sau phương thức init(), nó có thể truy xuất ServletConfig và ServletContext.
_jspService() có 2 tham số (HttpServletRequest, HttpServletResponse), 2 exception (ServletException, IOException) và cũng có stream xuất JspWriter.
Phương thức này *không thể nạp chồng*, có thể gọi nhiều lần, một lần một thread cho mỗi request.
Định nghĩa trong giao diện javax.servlet.jsp.HttpJspPage.
Tương tự servlet, có thể cấu hình các tham số khởi tạo cho JSP trong web.xml:

```
<servlet>
  <servlet-name>Example<servlet-name>
  <jsp-file>/example.jsp</jsp-file>
  <init-param>
    <param-name>parameterName</param-name>
    <param-value>parameterValue</param-value>
  </init-param>
</servlet>
```

Các thành phần của JSP

JSP bao gồm các thành phần:

1. Template HTML tĩnh
Các chuỗi của template không cần phiên dịch mà được chuyển trực tiếp đến JspWriter.

2. Comment (ghi chú)

<!-- HTML comment -->	có trong kết xuất HTML nhưng trình duyệt không hiển thị
<%-- JSP comment --%>	sẽ bị loại trong kết xuất HTML
<% // Java code comment (trong scriptlet) %>	

3. Directive (chỉ thị)
Dùng trong giai đoạn translation trang JSP thành servlet.
• <%@taglib %>, định nghĩa thư viện tag dùng trong trang JSP. Ví dụ: <%@ taglib prefix="pre" uri="lib.tld" %>
Tên của tập tin TLD chỉ định bởi thuộc tính uri, vị trí tập tin TLD do Web container tìm.
• <%@include %>, xử lý trang chèn tĩnh vào khi *translation*. Ví dụ: <%@ include file="filename.suf" %>, có thuộc tính *file*.
Phân biệt với <jsp: include page="result.jsp"> xử lý trang chèn vào khi *biên dịch* (chèn động), có thuộc tính *page*.
• <%@page %>, định nghĩa các thuộc tính chỉ định trang (page-specific), cho biết cách tạo ra servlet từ JSP này. Ngoại trừ pageEncoding và contentType phải đặt đầu trang, các thuộc tính còn lại có thể ở vị trí bất kỳ, áp dụng cho toàn trang.

Thuộc tính	Ví dụ	Mô tả
language	language="java"	Định nghĩa ngôn ngữ script dùng trong trang. Mặc định là Java.
extends	extends="…"	Servlet sinh từ JSP có thể thừa kế một lớp khác. Lớp này phải cài đặt giao diện JspPage hoặc HttpJspPage.
import	import="java.io.*, java.util.Date"	Các lớp mà servlet sinh từ JSP sẽ import. Một số không tường minh: java.lang, javax.servlet, javax.servlet.http, javax.servlet.jsp
session	session="true"	Servlet sinh từ JSP có tạo HttpSession không, mặc định là true.
buffer	buffer="8kb"	Chỉ định kích thước buffer, mặc định 8KB. Cần cho <jsp:forward> hoạt động.
autoflush	autoFlush="true"	"Súc" buffer khi đầy, mặc định là true.
isThreadSafe	isThreadSafe= "true"	Định nghĩa cấp an toàn của các thread. Nếu false, dùng SingleThreadModel.
info	info="…"	Mô tả trang. Dùng bởi công cụ quản lý server.
errorPage	errorPage="…"	Chỉ định trang báo lỗi sẽ điều hướng đến nếu trang này có lỗi.
isErrorPage	isErrorPage= "true"	Chỉ định đây là trang báo lỗi nếu isErrorPage bằng true.
contentType	contentType=" text/html "	Chỉ định kiểu nội dung, kiểu MIME và tập mã dùng cho trang trả về. Mặc định là text/html. Tương tự response.setContentType().
isELIgnored	isELIgnored="false"	Chỉ định bỏ qua EL khi "translation" trang JSP.
pageEncoding		Định nghĩa tập mã cho JSP, mặc định là ISO-8859-1.

4. Scriptlet

Code Java viết nhúng trong template HTML tĩnh, kết thúc bằng (;) cuối mỗi dòng lệnh.
Ví dụ: <% **if (1+2==true) {** %> yippee!! <% **}** **else {** %> boohoo!! <% **}** %>
• Script: code Java, bao trong cặp <% %>
• Khai báo:
<%**!** String name="JSP"; %>: khai báo *biến lớp* - instance variable, nằm ngoài _jspService.
<% String name="JSP"; %>: khai báo *biến cục bộ*, nằm trong _jspService.
Có thể khai báo một phương thức bằng <%**!** %>, tuy nhiên phương thức này không truy xuất trực tiếp các đối tượng không tường minh, phải truyền chúng đến phương thức đó như tham số. Nói chung, các Business Logic nên được tách thành Bean và được JSP gọi, không nên khai báo phương thức trong scriptlet.
Không thể khai báo một phương thức trong <% %>, vì Java không chấp nhận phương thức lồng.
• Biểu thức: <%out.println("Name: " + student.getName());%>
tương đương <%**=**"Name: " + student.getName()%> (chú ý không kết thúc bằng ;)
Lỗi từ scriptlet sẽ được bắt trong giai đoạn translation.

5. XML-based Tag
Khi dùng cú pháp XML-based Tag, toàn trang phải bao trong <jsp:root> </jsp:root> dù rỗng.
• Directive: <jsp:directive.{page|include} attrib="value" />
Taglib được khai báo như một xml namespace trong <jsp:root>: <jsp:root …xmlns:taglib-prefix= "uri-of-taglib" … > … </jsp:root>
• Declaration:<jsp:declaration> … </jsp:declaration>
• Expression:<jsp:expression> … </jsp:expression>
• Scriptlet:<jsp:scriptlet> … </jsp:scriptlet>

6. Action
Còn gọi là tag, tư tưởng chính của JSP là dùng tag phối hợp với EL.
• tập tag chuẩn (không cần khai báo).
<jsp:include>, ví dụ: <jsp:include page="relativeURL" flush="true" />, xem RequestDispatcher.include
<jsp:forward>, ví dụ: <jsp:forward page="relativeURL" />, xem RequestDispatcher(URI).forward(request, response)
<jsp:param>, thường dùng như tag lồng bên trong <jsp:include> hoặc <jsp:forward>, truyền tham số kèm theo trang được include hoặc forward. Ví dụ: chèn trang kèm tham số: other.jsp?abc=123

```
<jsp:include page='other.jsp' >
  <jsp:param name='abc' value='123' />
</jsp:include>
```

<jsp:useBean>, <jsp:setProperty>, <jsp:getProperty> xem phần Java Bean.
• các tag tùy biến (custom) do người dùng tạo ra.
Thư viện tag chuẩn JSTL giải quyết hầu hết các công việc cần thiết. Cần thêm thư viện JSTL vào project khi dùng.

7. Implicit Object (đối tượng không tường minh)
Các đối tượng không tường minh là các đối tượng ta sử dụng được ngay trong JSP mà không khai báo, không khởi tạo.
Thực chất đây là các đối tượng được tạo sẵn (với tên mặc định) trong servlet tự sinh tương ứng với trang JSP.
In/Out: **request** (HttpServletRequest), **response** (HttpServletResponse), **out** (JSPWriter của HttpServletResponse)
Scope: **page** (Object), **request** (HttpServletRequest), **session** (HttpSession), **application** (ServletContext)
Servlet: **config** (ServletConfig), **pageContext** (PageContext), **exception** (JspException, chỉ có trong trang báo lỗi)
Từ đối tượng **pageContext**, có thể truy xuất các đối tượng không tường minh khác, có thể lấy attribute của tầm vực bất kỳ.
Đối tượng exception chỉ có trong trang báo lỗi:
• Trang báo lỗi: có directive <%@page isErrorPage="true" %>, trong trang này ta lấy lỗi từ đối tượng **exception**.
Ví dụ: ${exception}
• Trang có thể bị lỗi: có directive <%@page errorPage="error.jsp" %>, khi xuất hiện lỗi, sẽ điều hướng đến trang error.jsp.

8. Ngôn ngữ EL
Hỗ trợ cho việc dùng tag, mô tả phần Expression Language. EL không được dùng chung với scriptlet.

Java Bean

- Lớp bean cần nằm trong package. Theo quy ước, lớp Java Bean phải:
• cung cấp một constructor mặc định (không có đối số) và public.
• cung cấp getters/setters cho các thuộc tính. Các thuộc tính của bean thường là các tham số cần chuyển cho bean để bean thực hiện thao tác nghiệp vụ của nó. Setter nhận đối số cùng kiểu với trị trả về của getter cùng cặp. Getter trả về boolean bắt đầu bằng "is". Ví dụ: String getName() với void setName(String), boolean isMember()
Nếu bean được dùng với JSP, các thuộc tính của bean phải có kiểu String và các kiểu cơ bản.
- Sử dụng bean: servlet đóng vai trò Controller tạo bean, thiết lập các thuộc tính của bean, gọi phương thức nghiệp vụ của bean.
Nếu gọi bean từ JSP, xem ví dụ chuyển tiếp bean giữa hai trang, first.jsp và second.jsp sau:
• Trang firts.jsp: <jsp:useBean> tạo thực thể bean, do không tìm thấy bean **bean** trong tầm vực request.
Chú ý khi lồng tag <jsp:setProperty> trong <jsp:useBean>, CHỈ thiết lập thuộc tính cho bean *mới được tạo*.

```
<jsp:useBean id="bean" class="samples.RegBean" scope="request">
  <jsp:setProperty name="bean" property="*" />
</jsp:useBean>
<jsp:forward page="second.jsp">
```

tương đương với:

```
<%
  samples.RegBean bean = new samples.RegBean();
  // giả sử form nhập có input: name
  String name = request.getParameter("name");
  bean.setName(name);
  request.setAttribute("bean", bean);
  request.getRequestDispatcher("second.jsp").forward(request, response);
%>
```

Tầm vực (scope) mặc định cho bean được tạo bằng <jsp:useBean> là page.
Chú ý, tạo bean cũng có thể dùng đa hình:

```
<jsp:useBean id="person" type="com.example.Person" class="com.example.Employee" />
```

tương đương:

```
<% Person person = new Employee(); %>
```

- Khi dùng <jsp:setProperty>, thuận tiện nhất là thiết lập hàng loạt thuộc tính của bean bằng property="*". Điều kiện là thuộc tính **name** của các input trong form nhập (nghĩa là các tham số trong request) phải *so trùng* với các field trong lớp bean.

Vài cách dùng <jsp:setProperty> khác:

```
// gán trị cho thuộc tính từ một biểu thức
<jsp:setProperty  name="address" property="city" value="<%=JSPExpression%>" />
// thuộc tính name của input phải giống field của bean
<jsp:setProperty name="aName" property="aProp" />
// thuộc tính name của input có thể khác field của bean
<jsp:setProperty name="aName" property="aProp" param="aParam"/>
```

• Trang second.jsp: tìm thấy bean **bean** trong các biến có tầm vực request, nên không tạo ra bean mới. Sau đó dùng <jsp:getProperty> để in trị, trong ví dụ dưới ta in thuộc tính name của bean, cũng có thể dùng EL: ${bean.name}

```
<jsp:useBean id="bean" class="samples.RegBean" scope="request" />
<jsp:getProperty name="bean" property="name" />
```

tương đương với:

```
<%
  samples.RegBean bean = (samples.RegBean)request.getAttribute("bean");
  if (bean == null) {
    bean = new samples.RegBean();
    request.setAttribute("bean", bean);
  }
  String name = bean.getName();
  out.println(name);
%>
```

Expression Language (EL)

EL là ngôn ngữ script cho phép truy xuất dễ dàng các đối tượng không tường minh, truy xuất các đối tượng có kiểu phức tạp, tính toán các biểu thức số học và luận lý. EL hỗ trợ việc dùng tag trở nên dễ dàng, tự nhiên.
Thường có dạng: ${firstArg.secondArg}
    firstArg: là *đối tượng* không tường minh (cookie, param, header, …) hoặc *Map* chứa các biến tầm vực (requestScope, sessionScope, …)
    secondArg: *key* của một Map, *thuộc tính* của Bean, …
EL cung cấp hai toán tử duyệt:
• toán tử dấu chấm: ${leftVariable.rightVariable}
    leftVariable: Map hoặc Bean.
    rightVariable: key của Map hoặc thuộc tính của Bean.
• toán tử ngoặc vuông []: ${leftVariable["content"]}
    leftVariable: Bean, List, Array, Map. (ghi nhớ: BLAM)
    content: key của Map, thuộc tính của Bean, chỉ số của List, chỉ số của Array.

1. Các toán tử
• số học: {+ - * / (hoặc div) % (hoặc mod)}. Ví dụ: ${34 mod 5} định trị 4
    phép div cho 0 trả về infinity mà không ném exception; phép mod cho 0 ném ArithmeticException.
• quan hệ: {< > <= >= == !=} hoặc {lt gt le ge eq ne}, ví dụ: ${4 >= 8} định trị false.
  Khi so sánh các đối tượng EL sẽ triệu gọi phương thức equals() hoặc compareTo() của đối tượng.
• luận lý: {&& || !} hoặc {and or not}, ví dụ: ${(5 > 3) && !(3 div 4 == 2)} định trị true.
• empty, ví dụ: ${empty sessionScope.member} kiểm tra biến member thuộc tầm vực session có null không.
  EL xem null như: chuỗi rỗng, biểu thức định trị 0, biểu thức logic định trị false.
EL không có nâng cấp kiểu, vì vậy ${"some string" + 3} sẽ ném ra javax.servlet.jsp.el.ELException.

2. Các đối tượng không tường minh
EL dễ dàng truy xuất các đối tượng không tường minh:
• pageContext, từ đó có thể truy xuất servletContext, request, response, session, application, out.

• param và paramValues, trả về Map chứa tham số theo tên của request (param) hoặc trả về tập các tham số (paramValues).
• initParam, trả về Map chứa các tham số khởi tạo của context, không phải của servlet.
• header và headerValues, trả về Map chứa một thành phần theo tên của header (header) hoặc trả về tập các thành phần của header (headerValues).
• cookie, trả về Map chứa các cookie, tương tự getCookies(). Ví dụ: ${cookie.cname} lấy cookie tên cname.

3. Các biến tầm vực
EL dễ dàng truy xuất các attribute thuộc các tầm vực khác nhau, trả về Map chứa các attribute và trị của chúng. Các tầm vực gồm: pageScope, request-Scope, sessionScope, và applicationScope.
Dùng toán tử truy xuất thuộc tính (. hoặc []) lấy attribute từ Map trả về .
${pageScope} ${requestScope} ${sessionScope} ${applicationScope}

```
${requestScope.book.id} hoặc ${requestScope["book"]["id"]}
```

Cặp "" có thể thiếu. List và array chỉ dùng toán tử [].

4. Một số cách dùng EL
- Lấy các tham số request: ví dụ truyền enroll.jsp?name=Marx, có một tham số tên **name**

```
${param.name}
```

tương đương:

```
<%=request.getParameter("name")%>
```

- Kiểm tra một điều kiện: kiểm tra collection books, biến tầm vực request, có rỗng không

```
<c:if test="${empty requestScope.books}">
  <h2>Database empty!</h2>
</c:if>
```

tương đương:

```
<%
 ArrayList<Book> books = (ArrayList<Book>) request.getAttribute("books");
 if (books.isEmpty()) {
   out.println("<h2>Database empty!</h2>");
 }
%>
```

- Truy xuất biến tầm vực là collection:

```
<c:forEach var="book" items="${requestScope.books}" varStatus="index">
  <tr>
    <td>${index.count}</td>
    <td>${book.name}</td>
    <td align="center">${book.price}</td>
  </tr>
</c:forEach>
```

tương đương:

```
<%
 ArrayList<Book> books = (ArrayList<Book>) request.getAttribute("books");
 int index = 0;
 for (Book book : books) {
   out.println("<tr>");
   out.println("<td>" + (++index) + "</h2>");
   out.println("<td>" + book.getName() + "</h2>");
   out.println("<td>" + book.getPrice() + "</h2>");
   out.println("</tr>");
 }
%>
```

- Viết code tùy biến EL:
• Viết code: lớp Java public với các phương thức public static, lưu tại /WEB-INF/classes hoặc /WEB-INF/lib nếu đóng gói .JAR.

```
public Example {
  public static String upper(String x) {
    return x.toUpperCase();
  }
}
```

• Khai báo trong TLD và trong web.xml:
Trong TLD:

```
<uri>myTldUri</uri>
<function>
  <name>Upper</name>
  <function-class>Example</function-class>
  <function-signature>java.lang.String Upper(java.lang.String)</function-signature>
</function>
```

Trong web.xml:

```
<taglib>
  <taglib-uri>myTldUri</taglib-uri>
  <taglib-location>/WEB-INF/example.tld</taglib-location>
</taglib>
```

• Khai báo directive taglib và sử dụng:

```
<%@ taglib prefix="up" uri="myTldUri" %>
${up:Upper(stringvar)}
```

JSTL (JSP Standard Tag Library)

JSTL 1.1 không thuộc về đặc tả JSP nên cần phải thêm thư viện JSTL vào ứng dụng khi lập trình, các gói JAR cần thiết sẽ có mặt trong WEB-INF/lib. Khi dùng JSTL, *nhất thiết phải có directive taglib*, ví dụ:

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

Các tập tag chính:

- Core (c): xử lý các tác vụ cơ bản.

• Các tác vụ chung: <c:set> <c:remove> <c:out> <c:catch>

```
<c:set var="color" scope="request" value="blue">
  BEFORE: color: ${color}
  <c:remove var="color" scope="request" />
  AFTER: color: ${color}
</c:set>
```

<c:set> dùng "var" cho biến tầm vực (attribute), dùng "target" cho thuộc tính của bean hoặc trị của Map.
<c:catch> dùng tương tự khối try-catch.

• Các tác vụ điều kiện: <c:if> <c:choose> <c:when> <c:otherwise>, giống cấu trúc if else.

```
<c:choose>
  <c:when test="${userType eq 'executive'}">
    Executive-specific statement
  </c:when>
  <c:when test="${userType eq 'manager'}">
    Manager-specific statement
  </c:when>
  <c:otherwise>
    Employee-specific statement (Everyone else)
  </c:otherwise>
</c:choose>
```

• Các tác vụ lặp: <c:forEach> <c:forTokens>, giống vòng lặp forEach.

```
<table>
  <c:forEach var="movie" items"${movieList}" varStatus="counter">
    <tr>
      <td>${counter.count}</td>
      <td>${movie}</td>
    </tr>
  </c:forEach>
</table>
```

• Các tác vụ liên quan URL:

<c:import> chèn động, có thuộc tính *url*.

```
<c:import url="http://www.domain.com/includes/header.jsp" />
```

<c:url> tự động xử lý URL Rewriting, thêm session ID cuối URL. Thường dùng kết hợp <c:param>

```
<c:url var="employeeProfile" value="/profile.jsp">
  <c:param name="firstName" value="${firstName}" />
  <c:param name="lastName" value="${lastName}" />
</c:url>
```

<c:redirect>

- SQL (sql): dùng truy xuất cơ sở dữ liệu.

setDataSource, query, update, transaction, param

- Format (fmt): dùng định dạng dữ liệu.
- XML (xml): hỗ trợ xml.

Custom Tag

1. Quy trình viết và sử dụng tag

1.1. Khai báo thư viện tag TLD (Tag Library Descriptor)

Tập tin TLD nói chung có hai phần:

- Mô tả thông tin về thư viện tag.
    - <tlib-version> khai báo phiên bản của thư viện tag.

- <short-name> được dùng bởi các công cụ (IDE) dùng thư viện này.
- <uri> tên duy nhất (không phải vị trí) được dùng trong directive <%@taglib%>

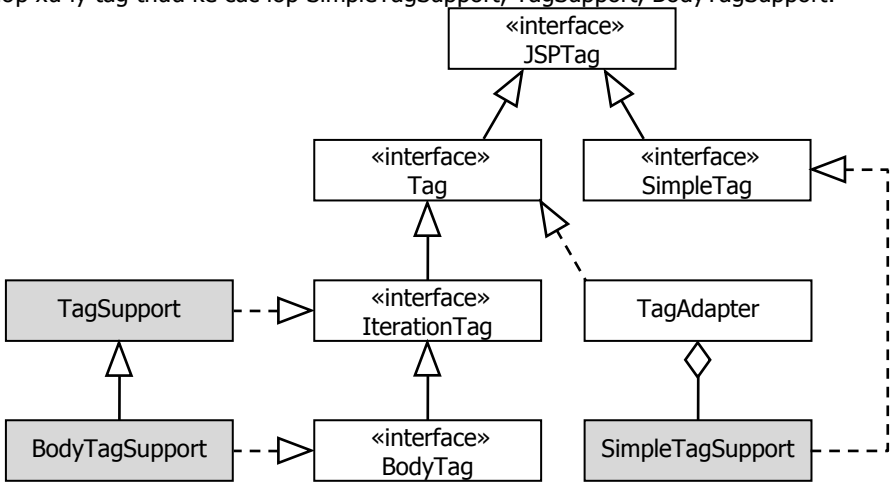
- mô tả các tag có trong thư viện bằng các <tag>, mỗi <tag> chứa:
  - <description> mô tả mục đích dùng tag.
  - <name> tên tag.
  - <tag-class> tên lớp xử lý tag.
  - <body-content> định nghĩa kiểu thân tag. Gồm: empty (tag rỗng), scriptless (thân không chứa scriptlet), tagdependent (thân xem như text), JSP (thân chứa bất kỳ thành phần nào của JSP).
  - <attribute> Các thuộc tính của tag được mô tả bằng các <attribute>, mỗi <attribute> chứa:
    <name> tên thuộc tính.
    <required> true là bắt buộc khai báo thuộc tính, mặc định là false.
    <rtexprvalue> true thì trị của thuộc tính có thể là biểu thức, mặc định là false. Biểu thức dùng cho trị của thuộc tính có thể là EL, biểu thức scriptlet và <jsp:attribute>

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<taglib xmlns="..." xmlns:xsi="..." xsi:schemaLocation="..." version="2.0">
  <tlib-version>0.9</tlib-version>
  <short-name>RandomTags</short-name>
  <uri>randomThings</uri>
  <tag>
    <description>Random Advice</description>
    <name>advice</name>
    <tag-class>samples.AdvisorTagHandler</tag-class>
    <body-content>empty</body-content>
    <attribute>
      <name>user</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
    </attribute>
  </tag>
</taglib>
```

Lúc đầu, thư viện tag rỗng nên chưa có mô tả <tag>. Với mỗi lớp xử lý tag được tạo, NetBeans hỗ trợ đưa mô tả <tag> tương ứng với lớp xử lý tag đó vào tập tin TLD.

1.2. Viết lớp xử lý tag (Tag Handler Class)

Tùy loại tag cần viết, lớp xử lý tag thừa kế các lớp SimpleTagSupport, TagSupport, BodyTagSupport:



Các lớp và giao diện xử lý tag

Ví dụ viết tag đơn giản,một lớp xử lý tag đơn giản phải thừa kế SimpleTagSupport và chỉ cài đặt cho phương thức doTag():

```
public class ExampleTag extends SimpleTagSupport {
  public void doTag() throws JspException, IOException {
    getJspContext.getOut.write("Example");
    getJspBody().invoke(null);                    // getJspBody() trả về JspFragment, gọi phương thức invoke,
  }                                               // không xử lý mà trực tiếp xuất thân của tag đến JspWriter
}
```

Thân của tag đơn giản có thể chứa text và biểu thức EL, nhưng không chứa scriptlet (khai báo, biểu thức, code. Nói cách khác, <body-content> của tag đơn giản không thể là JSP.

Trong lớp xử lý tag, ta có thể lấy các đối tượng không tường minh:

Các đối tượng không tường minh	Lấy các đối tượng không tường minh	
	Dùng phương thức	Dùng hằng
application	pageContext.getServletContext()	pageContext.getAttribute(PageContext.APPLICATION)
session	pageContext.getSession()	pageContext.getAttribute(PageContext.SESSION)
request	pageContext.getRequest()	pageContext.getAttribute(PageContext.REQUEST)

response	pageContext.getResponse()	pageContext.getAttribute(PageContext.RESPONSE)
out	pageContext.getOut()	pageContext.getAttribute(PageContext.OUT)
config	pageContext.getConfig()	pageContext.getAttribute(PageContext.CONFIG)
page	pageContext.getPage()	pageContext.getAttribute(PageContext.PAGE)
pageContext		pageContext.getAttribute(PageContext.PAGECONTEXT)
exception	pageContext.getException()	pageContext.getAttribute(PageContext.EXCEPTION)

Trong lớp xử lý tag, ta có thể lấy các attribute thuộc các tầm vực khác nhau:

Các tầm vực (scope)	Lấy attribute thuộc các tầm vực khác nhau	
	Dùng phương thức	Dùng hằng
application	pageContext.getServletContext() .getAttribute("name")	pageContext.getAttribute("name", PageContext.APPLICATION_SCOPE)
session	pageContext.getSession() .getAttribute("name")	pageContext.getAttribute("name", PageContext.SESSION_SCOPE)
request	pageContext.getRequest() .getAttribute("name")	pageContext.getAttribute("name", PageContext.REQUEST_SCOPE)
page	pageContext.getAttribute("name")	pageContext.getAttribute("name", PageContext.PAGE_SCOPE)

Kết quả ta có tập tin .JAR (trong /WEB-INF/lib) hoặc các tập tin class (trong /WEB-INF/classes).

1.3. Khai báo TLD trong web.xml

Nếu chỉ định tường minh <taglib-location> trong web.xml, Web container sẽ dùng nó. Nếu không, Web container sẽ *mặc định tìm TLD* trong: WEB-INF, META-INF của tập tin JAR có trong WEB-INF/lib.

```
<web-app>
  <jsp-config>
    <taglib>
      <taglib-uri>randomThings</taglib-uri>
      <taglib-location>/WEB-INF/myFunctions.tld</taglib-location>
    </taglib>
  </jsp-config>
</web-app>
```

1.4. Dùng tag trong JSP

```
<%@ taglib uri="randomThings" prefix="my" %>
<my:advice user="${username}" />
```

Các prefix không được phép dùng: jsp, jspx, servlet, java, javax, sun, sunw.

2. Các giao diện tag

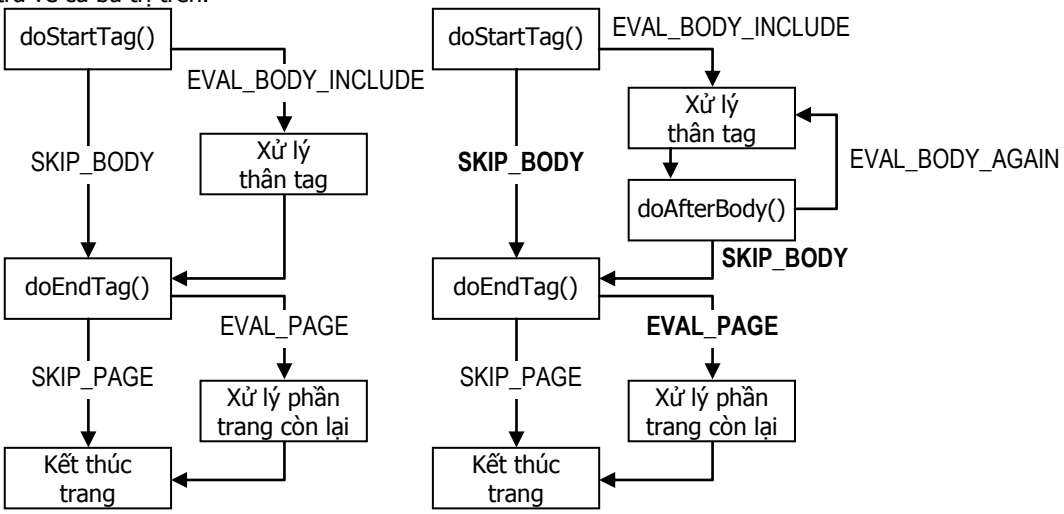
Giao diện Tag định nghĩa các phương thức có thể tùy biến trong lớp xử lý tag:

- **doStartTag()**

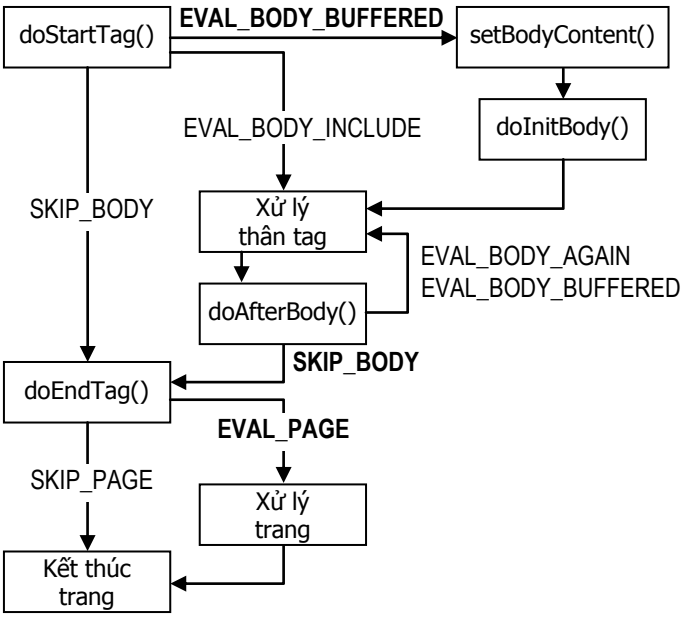
Được gọi sau khi JSP engine phân tích (parse) xong tag mở. Trước khi gọi doStartTag(), JSP engine gọi các phương thức sau:
setPageContext() → setParent() → setters cho các thuộc tính của tag.

doStartTag() trả về 3 trị:
SKIP_BODY: báo bỏ qua, không xử lý nội dung của thân tag.
EVAL_BODY_INCLUDE: báo xử lý thân tag như với mã JSP bình thường.
EVAL_BODY_BUFFERED: báo xử lý thân tag như với mã JSP, nhưng đệm lại kết xuất, không gửi đến client.

Giao diện IterationTag thừa kế giao diện Tag, lớp cài đặt giao diện này có phương thức doStartTag() chỉ trả về SKIP_BODY hoặc EVAL_BODY_INCLUDE. Giao diện BodyTag thừa kế giao diện IterationTag, lớp cài đặt giao diện này có phương thức doStartTag() trả về cả ba trị trên.



Trái : luồng xử lý tag cài đặt giao diện Tag, không có doAfterBody()
Phải: luồng xử lý tag cài đặt giao diện IterationTag, nhấn đậm là mặc định



Luồng xử lý tag cài đặt giao diện BodyTag, không có doAfterBody(), nhấn đậm là mặc định

- **doAfterBody()**

Được gọi khi JSP engine xử lý xong thân tag. doAfterBody() không được gọi với các tag chỉ cài đặt giao diện Tag; nói cách khác, giao diện Tag không có doAfterBody().

doAfterBody() được gọi lần thứ nhất, nếu:

Tag cài đặt giao diện IterationTag và doStartTag() trả về EVAL_BODY_INCLUDE.
Tag cài đặt giao diện BodyTag và doStartTag() trả về EVAL_BODY_INCLUDE.
Tag cài đặt giao diện BodyTag và doStartTag() trả về EVAL_BODY_BUFFERED.

Thân của tag có thể được xử lý lặp, bằng cách gọi lặp doAfterBody(), nếu:

Tag cài đặt giao diện IterationTag và lần gọi doAfterBody() trước trả về EVAL_BODY_AGAIN.
Tag cài đặt giao diện BodyTag và lần gọi doAfterBody() trước trả về EVAL_BODY_AGAIN.
Tag cài đặt giao diện BodyTag và lần gọi doAfterBody() trước trả về EVAL_BODY_BUFFERED.

doAfterBody() trả về 3 trị:

EVAL_BODY_AGAIN: xử lý thân tag lần nữa, không dùng đệm.
EVAL_BODY_BUFFERED: xử lý thân tag lần nữa, đệm kết xuất.
SKIP_BODY: bỏ qua, không xử lý thân tag lần nữa, không gọi doAfterBody() lần nữa.

Giao diện IterationTag định nghĩa doAfterBody(), lớp cài đặt giao diện này có phương thức doAfterBody() chỉ trả về SKIP_BODY hoặc EVAL_BODY_AGAIN. Giao diện BodyTag thừa kế giao diện IterationTag, lớp cài đặt giao diện này có phương thức doAfterBody() trả về: EVAL_BODY_TAG (lạc hậu), EVAL_BODY_AGAIN, EVAL_BODY_BUFFERED và SKIP_BODY.

- **doEndTag()**

Luôn luôn được gọi sau khi kết thúc xử lý tag.

doEndTag() trả về 2 trị:

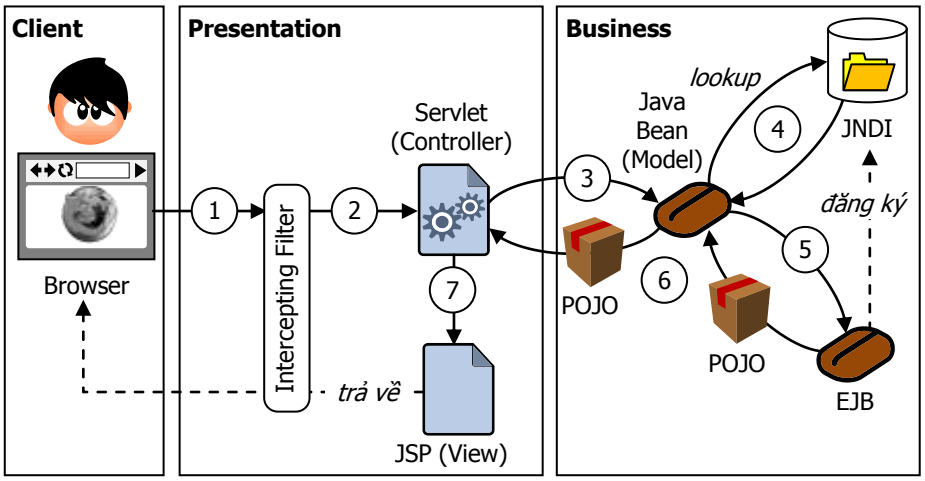
SKIP_PAGE: không xử lý phần còn lại của trang JSP. Bỏ qua hoàn toàn.
EVAL_PAGE: xử lý phần còn lại của trang JSP như với mã JSP bình thường.

Khi viết lớp xử lý tag, có thể dùng các đối tượng không tường minh, lấy thông qua pageContext.

Design Patterns

Trong J2EE, khi phát triển một ứng dụng web, thường áp dụng sáu mẫu thiết kế sau (chữ đậm):

① Một request được chặn đón bởi một **intercepting filter**.
② Nếu filter cho phép nó, request được chuyển đến một **front controller** (Controller) xử lý trung tâm, thường là servlet.
③ Front controller gọi phương thức nghiệp vụ từ một **business delegate** (Model), thường là Java Bean.
④ Business delegate dùng một **service locator**, nó tham chiếu JNDI để định vị đối tượng nghiệp vụ phân tán, thường là EJB.
⑤ Business delegate ủy nhiệm thao tác nghiệp vụ cho EJB thực hiện.
⑥ Dữ liệu được trả về chứa trong một **transfer object**, thường là các POJO.



⑦ Front controller chuyển tiếp dữ liệu trả về để xử lý tách biệt phần hiển thị với nghiệp vụ trong trang JSP (View) của **MVC**.

Bài tập

1. Tomcat

1.1. Cài đặt Tomcat 7.x

- Download J2SE 6 từ http://www.oracle.com/technetwork/java/javase/downloads/index.html. Cài đặt vào thư mục chỉ định, ví dụ: C:/java/j2se6

Thiết lập biến môi trường **JAVA_HOME**:

Nhấn +pause để vào hộp thoại System Properties, chọn tab Advanced rồi click nút Environment Variables. Trong phần User variables, nhấn nút New để thêm biến môi trường: variable name=JAVA_HOME, variable value=C:/java/j2se6.

Thêm C:/java/j2se6/bin vào biến môi trường PATH.

Logoff rồi vào lại để các biến môi trường có hiệu lực.

- Download Tomcat 7.x từ http://jakarta.apache.org/tomcat, cài đặt vào thư mục chỉ định (tạm gọi là TOMCAT_HOME), ví dụ: C:/java/tomcat.

Khi dùng NetBeans, Tomcat được cài đặt chung với NetBeans hoặc có thể cài đặt bổ sung. Vào Tools > Servers, chọn Tomcat và chú ý Catalina Home, chính là nơi Tomcat được cài đặt. Chú ý port quản lý (Server Port), thường là 8084.

Phải kích hoạt hỗ trợ ứng dụng web của NetBeans bằng cách tạo một project web.

- Bổ sung JDBC Driver Type 4 dùng truy xuất Microsoft SQLServer, tập tin sqljdbc4.jar, vào thư mục TOMCAT_HOME/lib. Ứng dụng web trên Tomcat dùng driver này để truy xuất cơ sở dữ liệu.

2.1. Chạy, triển khai ứng dụng Web và dừng Tomcat (bằng tay)

- Trong TOMCAT_HOME/bin, chạy startup.bat. Tomcat sẽ chạy trong cửa sổ Console, cho thấy quá trình khởi động. Khi xuất hiện dòng: INFO: Server startup in XXXX ms, Tomcat đã khởi động xong.
- Gói ứng dụng Web, ví dụ ABC.war, được sao chép thẳng vào thư mục TOMCAT_HOME/webapps. Tomcat sẽ tự động triển khai thành thư mục ABC (gọi là Web Application Context), là thư mục con của thư mục trên.

Thực tế tập tin ABC.war sẽ được upload lên Tomcat Server bằng công cụ quản trị. URL: http://localhost:8080/manager/html, vào phần Deploy > War file to deploy để upload.

- Chạy ứng dụng Web ABC trên bằng cách mở Web Browser và nhập vào URL: http://localhost:8080/ABC.
- Dừng Tomcat bằng cách vào TOMCAT_HOME/bin, chạy shutdown.bat.

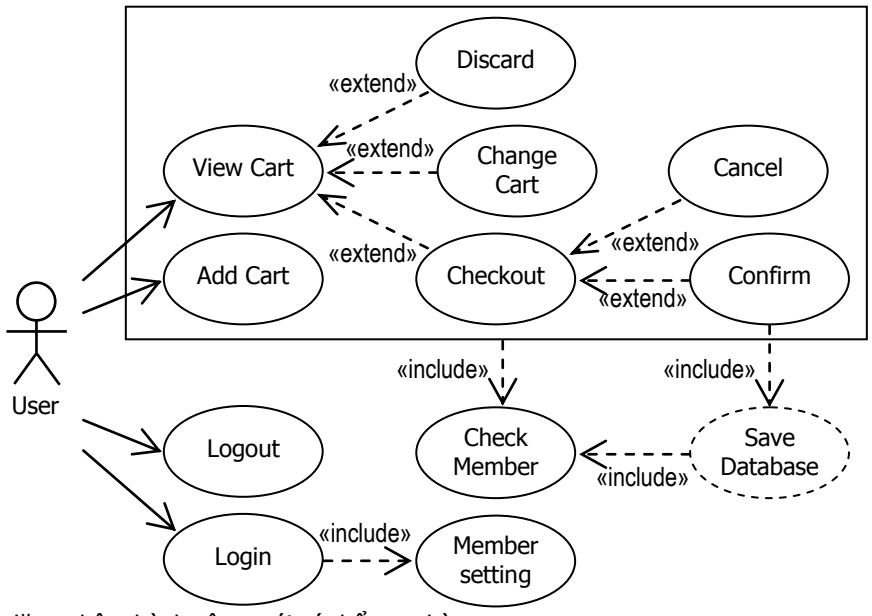
Trong NetBeans, có thể khởi động Tomcat rồi triển khai trực tiếp ứng dụng Web từ IDE.

2. Quản lý phiên giao dịch

Project xây dựng phân hệ Shopping Cart thường được dùng trong các trang mua hàng trực tuyến. Shopping Cart và phân quyền bằng lập trình là các ví dụ minh họa điển hình cho cách dùng biến thuộc tầm vực Session, sử dụng để quản lý trạng thái phiên giao dịch.

2.1. Đặc tả

Sơ đồ Use case:



- Login: khách hàng phải đăng nhập thành công mới có thể mua hàng.
- Member setting: sau khi khách hàng đăng nhập thành công với quyền user, đối tượng **member** chứa thông tin của khách hàng sẽ được đưa vào tầm vực Session, dùng để kiểm tra phân quyền và lấy thông tin khách hàng nếu cần.
- Check member: phải kiểm tra để chắc rằng khách hàng đã đăng nhập và có quyền user, trước khi cho phép khách hàng thực hiện các thao tác với Shopping Cart.
- Logout: loại bỏ tất cả thông tin phiên giao dịch có liên quan đến khách hàng. Cụ thể, các biến thuộc tầm vực Session của khách hàng sẽ bị loại bỏ.
- Add Cart: khách hàng thêm sản phẩm vào cart. Nếu mặt hàng đã có trong cart, quantity của mặt hàng sẽ tăng.
- View Cart: khách hàng xem các sản phẩm đã lựa chọn, nếu chưa có sản phẩm nào, sẽ báo cart rỗng. Khi đang xem cart, khách hàng có thể thực hiện các thao tác sau:

  + Discard: khách hàng loại bỏ cart, không mua nữa. Thao tác này giống như Checkout nhưng ta không xuất Invoice.

+ Change Cart: khách hàng loại bớt sản phẩm khỏi cart bằng cách click vào link Remove, một lần loại một đơn vị sản phẩm. Nếu loại bỏ tất cả sản phẩm, sẽ báo cart rỗng.
+ Checkout: khách hàng kết thúc lựa chọn hàng, xuất Invoice cho khách hàng. Chuyển tất cả nội dung của cart ra Invoice và loại bỏ biến tầm vực Session **cart**. Invoice cũng chứa thông tin khách hàng, lấy từ biến tầm vực Session **member**.
+ Khách hàng cũng có thể quay trở lại trang chủ, tiếp tục mua thêm hàng.

Welcome **hugo** [Logout]

Shopping Cart

Buy more | Checkout | Discard

No	Name	Price	Quantity	Remove
1	Nobody's Baby But Mine	2.9	2	Remove
2	Master and Magarita	9.7	3	Remove
		Total = 34.9		

© Copyright by **Memorial Duck**
Fri, May 10, 2013

Khi xem Invoice khách hàng có thể thực hiện các thao tác sau:
+ Cancel: khách hàng không mua hàng, quyết định bỏ Invoice.
+ Confirm: khách hàng xác nhận mua hàng, thông tin trong Invoice sẽ được lưu xuống cơ sở dữ liệu.

- Save to Database: lưu thông tin Invoice xuống cơ sở dữ liệu, để bộ phận bán hàng xử lý tiếp. Phần này đơn giản, không hiện thực trong project, mà xem như bài tập (chèn thông tin vào bảng Orders và OrderItems).

Welcome **hugo** [Logout]

Invoice

Invoice Number : INVVIC4015
Date : May 10, 2013
Customer : Victor Hugo
Address : 45 Napoleon Str., Paris
Mobile : 0903859412

No	Name	Price	Quantity	Total Line
1	Nobody's Baby But Mine	2.9	2	5.8
2	Master and Magarita	9.7	3	29.1
			Total:	34.9

Confirm Discard

© Copyright by **Memorial Duck**
Fri, May 10, 2013

2.2. Cơ sở dữ liệu

a) Tạo nguồn dữ liệu (JDBC Data Source)

Tomcat cung cấp dịch vụ naming gọi qua JNDI, cho phép đăng ký một Data Source và tham chiếu (lookup) đến nó từ ứng dụng web. Tomcat cũng cung cấp khả năng quản lý kết nối (container-managed connection pooling).

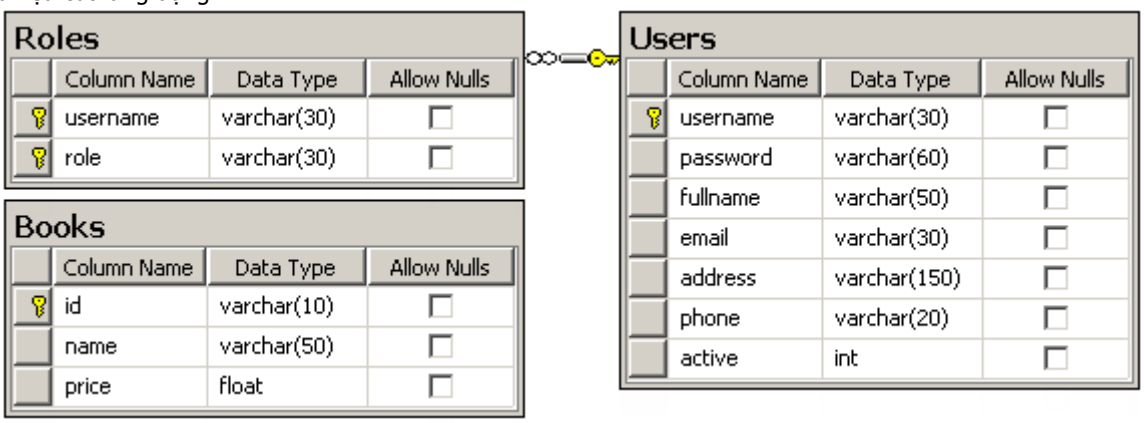
Trong ứng dụng web, Data Source được khai báo trong /META-INF/context.xml. Các bước như sau:

- Tải về driver JDBC Type 4 của SQL Server 2005, **sqljdbc4.jar**. Đặt trong thư mục TOMCAT_HOME/lib.
- Khai báo một driver mới: tab Services, nhánh Databases > Drivers, click phải, chọn New Driver… Click nút Add…, chỉ đến tập tin sqljdbc4.jar. Kết quả ta có thêm driver: **Microsoft SQL Server 2005**.

- Tạo một kết nối mới: tab Services, nhánh Databases, click phải, chọn New Connection…
Bước **Locate Driver**, chọn Driver là Microsoft SQL Server 2005.
Bước **Customize Connection**, chọn Host:Port là localhost:1433, Database và User Name:Password như đã dùng trong SQL Server. Chú ý, phải bảo đảm kết nối SQL Server thông qua kết nối TCP/IP, chế độ xác thực là SQL Server Authentication.
Click nút Test Connection để kiểm tra kết nối đến JDBC URL.
Bước **Choose Database Schema**, chọn schema là **dbo**.
Đổi tên kết nối mới tạo cho dễ dùng, ví dụ BooksDB.
- Trong một tập tin .java nào đó của project, chọn Insert Code… (Alt+Insert) > Use Database…, thực hiện các bước để thêm Data Source. Kiểm tra /META-INF/context.xml để chắc rằng nguồn dữ liệu đã được khai báo tự động. Chú ý đến tên của nguồn dữ liệu, trong ví dụ này là BooksDB. Nếu triển khai ứng dụng từ mã nguồn do chúng tôi cung cấp, cần chú ý hiệu chỉnh tập tin context.xml này. Tập tin .java trên có thể loại bỏ nếu không cần thiết.
Trong servlet, dùng nguồn dữ liệu (BooksDB) vừa tạo bằng annotation Resource:

```
@Resource(name = "BooksDB")
private DataSource ds;
```

b) Cơ sở dữ liệu của ứng dụng



2.3. Phân trang
Trang index.jsp có thể viết theo mô hình MVC, nhưng trong project này ta tập trung vào phần Shopping Cart nên ta chỉ đơn giản dùng tag **sql:query** của JSTL để liệt kê danh sách sản phẩm. Các bước như sau:
- Trong project, click phải nhánh Libraries, chọn Add Library… và chọn JSTL 1.1.
- Trong Palette (Ctrl+Shift+8) bên phải, phần Database, kéo thả icon DB Report vào trang index.jsp:
  + Nhập tên Data Source: vừa tạo phần trên, BooksDB.
  + Nhập câu truy vấn Query Statement: phát biểu SQL để hiển thị thông tin trong bảng Books.
- Điều chỉnh code vừa sinh để có bảng kết quả như ý muốn. Thêm link **View Cart** và cột **Buy** cho bảng.
Trang kết quả không có phân trang (paging), ta tiến hành phân trang bằng truy vấn.
Cần các biến sau:
- **rows**: số record của kết quả, ta dùng một tag <sql:query> thực hiện một truy vấn để lấy số record này.
- **start**: chỉ số của record bắt đầu một trang, lưu trong biến **start**, khi chuyển giữa các trang ta dùng tham số **no** để chuyển. Quan sát tham số **no** này trong các link của phân trang.
- **perpage**: số record trong mỗi trang, định nghĩa trong biến **perpage**. Như vậy trong mỗi trang ta dùng tag <sql:query> với startRow="${start}" và maxRows="${perpage}" để lấy đúng số record yêu cầu.
Để tạo các link cho phân trang, ta dùng tag <c:forEach> tạo vòng lặp xuất các link. Chú ý các thuộc tính step="${perpage}" và varStatus="index", chúng giúp ta dễ dàng tính chỉ số trang và tham số **no** cần chuyển.
Phân trang phát sinh một số vấn đề cần giải quyết. Ví dụ, khách hàng đang ở trang 3 của index.jsp. Khách hàng thêm một sản phẩm vào giỏ hàng hoặc chuyển sang xem giỏ hàng (View Cart); hơn nữa, khi đang xem giỏ hàng, khách hàng thực hiện một thao tác chuyển tiếp nào đó, như xóa bớt sản phẩm đã chọn. Ta cần phải bảo đảm khi khách hàng quay trở về trang index.jsp, họ vẫn về đúng trang 3.
Chạy thử chương trình. Chú ý, để quản trị các server chạy trong NetBeans, dùng tab Services, nhánh Server và chọn server cần quan tâm; ta có thể khởi động, dừng server, mở xem tập tin log một cách thuận tiện.

2.4. Shopping Cart
Ta tạo một đối tượng lớp Cart riêng (**cart**) để lưu danh sách các sản phẩm khách đã chọn:
- Cart có một HashMap (**books**) lưu các sản phẩm, mỗi sản phẩm được lưu với khóa là ID của sản phẩm.
- Các phương thức lớp Cart là những thao tác cơ bản mà khách hàng thực hiện với giỏ hàng của mình: **add** (thêm sản phẩm), **remove** (loại bớt sản phẩm), **getContents** (xem danh sách các sản phẩm đã chọn), **getTotal** (tổng giá tiền giỏ hàng).
- HashMap của Cart chứa các đối tượng sản phẩm, thuộc lớp Book. Có vài điểm cần chú ý trong lớp này:
  + Biến thực thể **quantity**, mặc định là 1. Khi thêm hoặc xóa sản phẩm trong HashMap cần biến này: thêm một sản phẩm đã có chỉ cần tăng quantity của nó, xóa một sản phẩm có quantity bằng 1 nghĩa là loại bỏ nó ra khỏi HashMap.
  + Link **Buy** truyền id của sản phẩm. Phương thức static findByID của lớp Book lấy thông tin sản phẩm dựa trên id này, trả về một đối tượng Book để thêm vào Cart.
  + Phương thức **getTotalLine()** giúp tính nhanh tổng giá tiền cho một sản phẩm.

Như vậy, chỉ cần gán tầm vực Session cho đối tượng **cart**, nó có thể lưu trạng thái mua hàng của khách hàng, tồn tại xuyên qua hàng loạt các request trong phiên giao dịch của khách hàng. Khi khách hàng thực hiện các request, dựa trên tham số action khác nhau lấy từ đối tượng request, CartServlet sẽ gọi phương thức thích hợp của biến cart để cập nhật trạng thái.
Khi khách hàng kết thúc lựa chọn hàng, việc xuất Invoice cho khách hàng dựa trên các thông tin:
- Mã của Invoice: INV + 3 ký tự đầu fullname (viết hoa) của khách hàng + 4 ký tự cuối của chuỗi thời gian hiện hành.
- Thông tin khách hàng: lấy từ biến tầm vực Session **member**.
- Thông tin đơn hàng: trước khi xóa biến tầm vực Session **cart**, ta chuyển thông tin nó chứa cho biến tầm vực Request **invoice**. Thông tin của đơn hàng chứa trong biến invoice này sẽ được trang invoice.jsp hiển thị.
Chạy thử chương trình, trước khi cài đặt bảo mật.

2.5. Bảo mật
Sau khi project hoàn tất và chạy thử ổn định, ta mới tiến hành viết phần bảo mật. Nói chung, phần bảo mật được thực hiện cuối cùng trong quy trình, do Deployer chịu trách nhiệm.
Xác thực để đăng nhập dùng hai bảng: Users và Roles. Sau khi người dùng đăng nhập thành công, biến tầm vực Session **member** sẽ lưu thông tin người dùng. Sau đó dựa trên biến này ta có vài cách thực hiện bảo mật bằng lập trình:
- Dùng filter để lọc các truy cập đến JSP hoặc Servlet. Việc bảo đảm xác thực (authentication) đơn giản, nhưng lại phức tạp khi phân quyền (authorization).
- Dùng code kiểm tra với từng trang JSP và Servlet. Tuy nhiên, không thể bảo vệ một vùng web, ví dụ không thể ngăn chặn việc truy cập một tập tin ảnh tại một vùng web nào đó.

a) Login
- Model: lớp **LoginBean**, phương thức nghiệp vụ là **getMember()** trả về một đối tượng thuộc lớp Member nếu đăng nhập thành công. Nếu đăng nhập thất bại, getMember() trả về null.
- Controller: lớp **LoginServlet**, tạo thực thể bean (LoginBean), nhận thông tin username và password từ client, rồi thiết lập cho các thuộc tính tương ứng của bean, cuối cùng gọi phương thức nghiệp vụ getMember(). Đối tượng Member trả về được gán vào biến tầm vực Session **member** và được kiểm tra tại đầu các trang JSP hoặc Servlet. Đối tượng member trong tầm vực Session hữu ích khi ta cần lấy thông tin của khách hàng khi tạo Invoice.
Khi thực hiện điều này, bạn có thể phải sao chép nhiều lần code có chứa tag JSTL, chú ý không quên sao chép khai báo directive taglib kèm theo, tag JSTL chỉ hoạt động khi có khai báo này.
Một đối tượng lớp Member có nhiều role, chứa trong một ArrayList các role. Phương thức getMember() sau khi xác định người đăng nhập xác thực thành công, sẽ truy xuất tiếp cơ sở dữ liệu để lấy đủ các role của người dùng.
Để bảo mật cơ sở dữ liệu, password của người dùng được lưu dưới dạng MD5 (hàm băm bảo mật mã hóa một chiều). LoginBean sẽ mã hóa MD5 password người dùng nhập trước khi kiểm tra. Như vậy kẻ xâm nhập truy cập được cơ sở dữ liệu và ngay cả admin cũng không biết được mật khẩu thật của người dùng.

b) Logout
- Controller dùng phương thức **invalidate()** của đối tượng Session để xóa các biến thuộc tầm vực Session đang tồn tại.
Sau đó chuyển về trang chủ index.jsp.
Chạy thử chương trình.

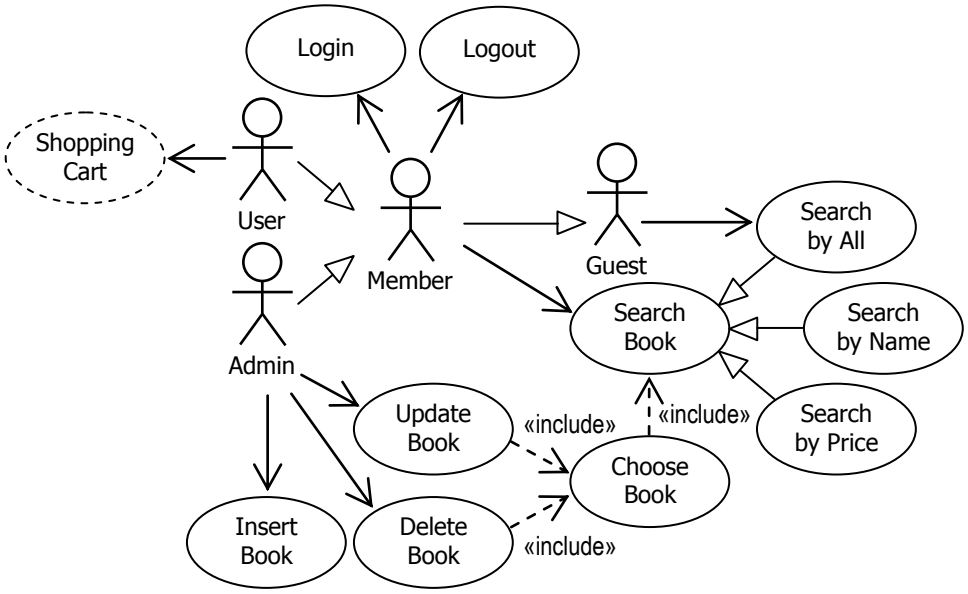
2.6. Cookie
Khái niệm quản lý phiên giao dịch bằng đối tượng Session gắn liền với cookie. Bài tập nhỏ sau minh họa cách dùng cookie lưu thông tin đăng nhập của người dùng.

Chọn checkbox để lưu thông tin đăng nhập từ form đăng nhập vào cookie **user**. Thông tin đăng nhập có định dạng: username:password
- Form đăng nhập sẽ kiểm tra cookie **user**, nếu không có thì hiển thị form trống.
- Khi đăng nhập, nếu chọn checkbox, cookie **user** sẽ lưu vào máy khách thông tin đăng nhập để hiển thị sẵn trong lần đăng nhập sau.
Xem log để thấy cookie được tạo mới và được ghi lại.
Tìm tập tin chứa cookie trong máy client, xóa cookie này và thấy form đăng nhập không hiển thị thông tin đã lưu.

3. Mô hình MVC
Project tập trung vào phân hệ quản lý sản phẩm và minh họa mô hình MVC khi xây dựng ứng dụng web. Project cũng cài đặt các tính năng cần thiết cho ứng dụng web như: i18n, phân quyền người dùng. Ngoài ra, project ghép thêm phân hệ mua hàng từ project Shopping Cart. Các bước chính thực hiện project:
- Thực hiện đầy đủ các chức năng yêu cầu như: CRUD (Create-Read-Update-Delete), tìm kiếm sản phẩm, shopping cart.
- Cài đặt tính năng i18n, cung cấp đa ngôn ngữ cho ứng dụng web.

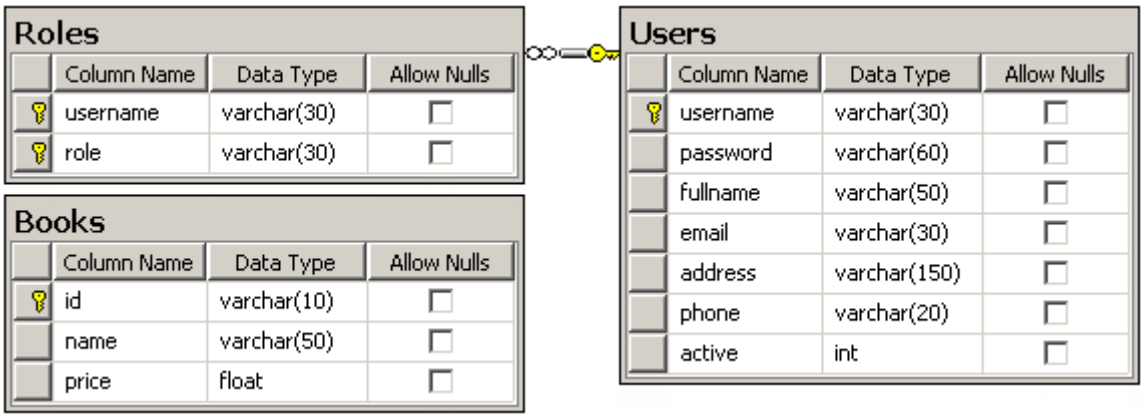
- Cài đặt bảo mật cho project, cung cấp chức năng đăng nhập và phân quyền.
Sau từng giai đoạn thực hiện, chạy thử chương trình để kết thúc công việc tại mỗi giai đoạn và dễ khoanh vùng lỗi.

3.1. Đặc tả
Sơ đồ Use case:



Phân quyền trong hệ thống:
- Người dùng Guest (hay Anonymous) chỉ có quyền xem thông tin sản phẩm.
- Thành viên có đăng nhập (Admin, User) được quyền tìm sản phẩm theo nhiều cách: xem tất cả (như Guest), tìm theo tên (Search by Name) hoặc tìm theo giá (Search by Price) với giá nằm trong giới hạn chỉ định.
- Người dùng có quyền User có thể mua hàng, phần này ghép từ project Shopping Cart trước.
- Người dùng có quyền Admin có thể hiệu chỉnh cơ sở dữ liệu, nghĩa là có quyền thêm (Insert Book), xóa (Delete Book), sửa (Update Book) sản phẩm chỉ định. Tuy nhiên, Admin không có quyền mua hàng.

3.2. Cơ sở dữ liệu



3.3. Thực hiện
a) Show
Chức năng Show minh họa việc sử dụng JSTL là dễ dàng trong NetBeans, vì vậy chức năng này không kiến trúc theo mô hình MVC. Thực tế, không nên dùng JSP thực hiện phần Controller.
Trong Palette (Ctrl+Shift+8) bên phải, phần Database, kéo thả icon DB Report vào index.jsp. Nhập tên Data Source là **BooksDB** và Query Statement (phát biểu SQL để hiển thị thông tin trong bảng Books).
Thêm các cột cần thiết cho các link gọi chức năng: **Buy**, **Update**, **Delete**. Link **Buy** và link **Delete** truyền thông tin của sản phẩm bằng chuỗi khóa chính **id**, phương thức static getBook(String) của lớp Book truy vấn cơ sở dữ liệu để lấy thông tin của đối tượng Book cụ thể.
Trong trang này và các trang khác tiếp sau, menu liệt kê đầy đủ tất cả các chức năng. Việc phân quyền sẽ thực hiện sau cùng, khi phân quyền xong, menu chỉ hiển thị các chức năng tương ứng với quyền của người dùng đang đăng nhập.
Chạy thử chương trình, bảo đảm bảng có dữ liệu thử để hiển thị.

b) Delete
Ta thực hiện chức năng này trước vì nó đơn giản nhất. Trong trang index.jsp, ta dùng một đoạn javascript nhỏ để hỏi người dùng trước khi gọi Controller (DeleteServlet) thực hiện việc loại bỏ sản phẩm.
- Model: lớp **DeleteBean**, phương thức nghiệp vụ là **deleteBook()** thực hiện lệnh SQL DELETE để xóa record yêu cầu, khóa id cần xóa được truyền đến trường id của lớp này.

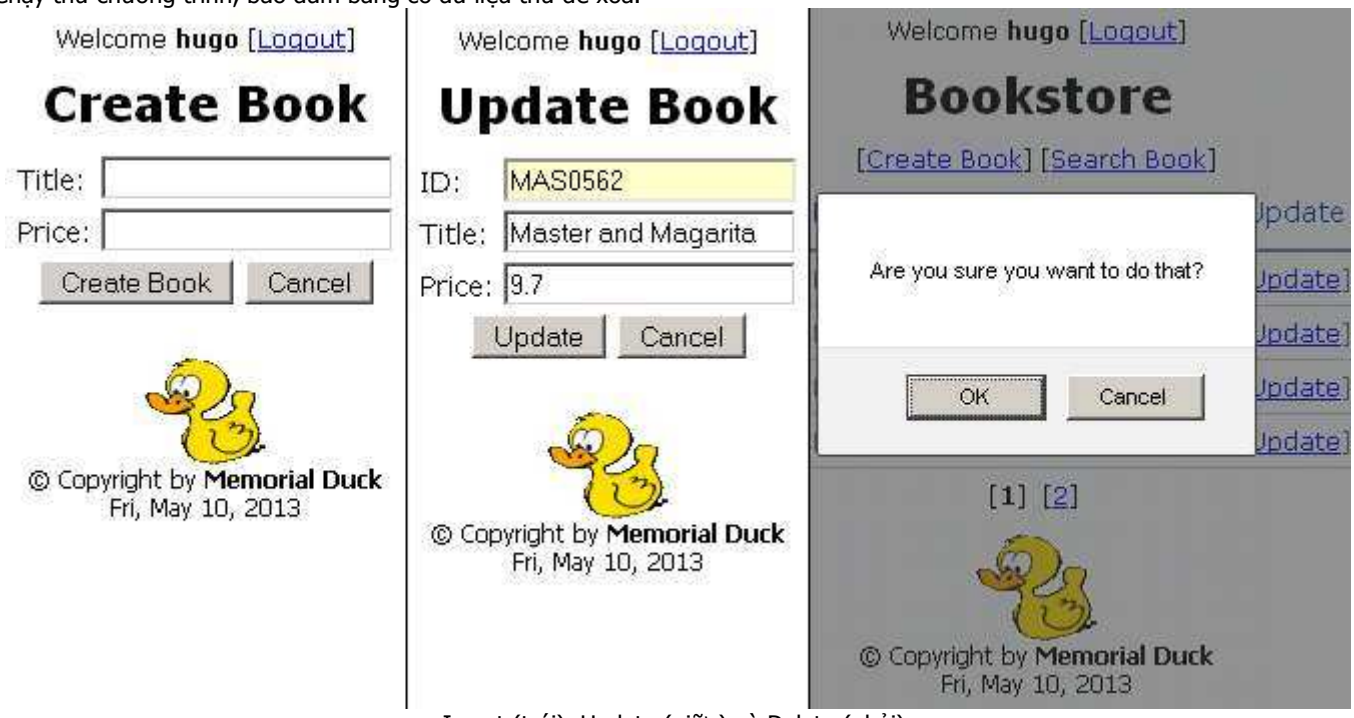
- Controller: lớp **DeleteServlet**, tạo thực thể **bean** (lớp DeleteBean), nhận id từ client rồi thiết lập cho thuộc tính id của bean, gọi phương thức nghiệp vụ deleteBook() của bean để loại bỏ sản phẩm, sau đó chuyển đến url là trang index.jsp. Trang đến login.jsp dùng cho phân quyền sau này.

Khi khởi tạo cho bean, ta trao cho constructor của bean hai tham số:

- Tham chiếu đến servlet gọi bean, mục đích để bean có thể sử dụng context log từ servlet gọi nó.
- Nguồn dữ liệu, được khai báo nhờ annotation @Resource trong servlet rồi chuyển cho bean sử dụng.

Với các cặp servlet - bean (Controller - Model) khác ta cũng thực hiện tương tự.

Chạy thử chương trình, bảo đảm bảng có dữ liệu thử để xóa.



Insert (trái), Update (giữa) và Delete (phải).

c) Insert

Thao tác Insert chia thành 2 giai đoạn:

+ Bước 1: gọi trang **insert.jsp** (form front-end) để người dùng nhập các thông tin cần thiết cho việc tạo một sản phẩm mới. Trang này được kiểm thử (validate) phía client bằng javascript. Như vậy là đủ do trang này chỉ có admin sử dụng.
+ Bước 2: trang insert.jsp gọi Controller để thực hiện việc chèn record mới vào cơ sở dữ liệu.

- Model: lớp **InsertBean**, phương thức nghiệp vụ là **insertBook()** thực hiện lệnh SQL INSERT để chèn mới record yêu cầu, thông tin đầy đủ cho record này (trừ id) được chuyển thông qua các thuộc tính của lớp bean.

Phương thức nghiệp vụ insertBook() tự tạo id của record theo luật: 3 ký tự đầu của tên sản phẩm viết hoa + 7 ký tự cuối của chuỗi thời gian hiện tại. Ta không sinh id trong constructor của lớp Book do constructor này cũng cần cho cập nhật Bean, nhưng việc sinh id lại chỉ dùng cho record mới.

Để giải quyết vấn đề lưu trị float với độ chính xác gốc, ta có thể sử dụng hàm ép kiểu trong phát biểu SQL:

```
pstmt = conn.prepareStatement("INSERT INTO Books VALUES (?, ?, CAST(? AS NUMERIC(5, 2)))");
pstmt.setFloat(3, price);
```

- Controller: lớp **InsertServlet**, tạo thực thể **bean** (lớp InsertBean), nhận thông tin chi tiết về sản phẩm mới từ client (không có id), thiết lập cho các thuộc tính của bean, gọi phương thức nghiệp vụ insertBook() của bean, chuyển đến url là trang index.jsp. Trang đến login.jsp dùng cho phân quyền sau này.

Học viên có khuynh hướng viết một Controller chung hoặc Model chung, như vậy khó phân công công việc, các luồng xử lý nên được viết tách biệt.

Chạy thử chương trình.

d) Update

Thao tác Update cũng chia thành 2 giai đoạn:

+ Bước 1: gọi trang **update.jsp** (from front-end), truyền trực tiếp các thông tin để điền vào form. Người dùng có thể hiệu chỉnh các form này để cập nhật thông tin mới. Riêng trường id chỉ hiển thị mà không cho phép thay đổi. Trang này được kiểm thử (validate) phía client bằng javascript. Như vậy là đủ do trang này chỉ có admin sử dụng.
+ Bước 2: trang update.jsp gọi Controller để thực hiện việc cập nhật record trong cơ sở dữ liệu với id yêu cầu.

- Model: lớp **UpdateBean**, phương thức nghiệp vụ là **updateBook()** thực hiện lệnh SQL UPDATE để chèn mới record yêu cầu, thông tin đầy đủ cho record này (cả id) được chuyển thông qua các thuộc tính của lớp bean.
- Controller: lớp **UpdateServlet**, tạo thực thể **bean** (lớp UpdateBean), nhận thông tin hiệu chỉnh sản phẩm từ client , thiết lập cho các thuộc tính của bean, gọi phương thức nghiệp vụ updateBook() của bean, chuyển đến url là trang index.jsp. Trang đến login.jsp dùng cho phân quyền sau này.

Chạy thử chương trình.

e) Search

Thao tác Search chia thành 3 giai đoạn:

+ Bước 1: gọi trang **search.jsp** (from front-end), người dùng sẽ chọn điều kiện search và nhập thông tin cần thiết. Trang này được kiểm thử (validate) bằng javascript. Như vậy là đủ do trang này chỉ có user sử dụng, quyền **guest** không được search. Chú ý hai nút của form này có thuộc tính name giống nhau là **find**, SearchServlet dùng trị nhận được của tham số find này để phân biệt nút nào được click.
+ Bước 2: trang search.jsp gọi Controller để thực hiện tìm trong cơ sở dữ liệu với điều kiện tìm do người dùng yêu cầu.
+ Bước 3: Controller sẽ chuyển tiếp (forward) kết quả đến trang **result.jsp**.

- Model: lớp **SearchBean**, phương thức nghiệp vụ là **findByName()** và **findByPrice()** thực hiện lệnh SQL SELECT WHERE theo các điều kiện yêu cầu, các trường của bean sẽ cung cấp thông tin cần thiết cho hai phương thức này, tuy nhiên ta chỉ thiết lập và sử dụng các trường có liên quan đến phương thức nghiệp vụ được gọi.
- Controller: lớp **SearchServlet**, tạo thực thể **bean** (lớp SearchBean), nhận khóa tìm kiếm từ client, xác định nút nào của form đã được nhấn (thông qua tham số **find**), thiết lập các thuộc tính cần thiết để tìm cho bean, sau đó gọi phương thức nghiệp vụ tương ứng.

Cả hai phương thức đều trả về ArrayList<Book>, đưa kết quả này vào biến request **books** rồi chuyển tiếp đến trang result.jsp bằng phương thức forward của giao diện RequestDispatcher.

- View: trang **result.jsp**. Đầu vào của trang này là biến request **books**, bản chất là một collection. Ta dùng tag <c:forEach> của JSTL để duyệt thông tin trong collection này và hiển thị trong bảng với các cột cần thiết. Trước đó, ta kiểm tra collection có rỗng hay không để báo trong trường hợp không có kết quả tìm theo yêu cầu đưa ra.

Chạy thử chương trình.

f) Phân trang

Như đã trình bày trong project Shopping Cart, ta phân trang cho trang index.jsp bằng cơ sở dữ liệu, dựa trên tag <sql:query> của JSTL. Ta cũng muốn trang kết quả tìm kiếm sản phẩm (result.jsp) được phân trang như vậy.

Tuy nhiên, trang này là phần View của mô hình MVC, nên việc phân trang có khác.

- Trong Controller, lớp **SearchServlet**, ta chuẩn bị các biến cần cho việc phân trang:
  + **rows**: số record kết quả tìm kiếm được.
  + **perpage**: số record hiển thị trong một trang kết quả.
  + **start**: chỉ số của record đầu trong một trang kết quả.
- Trong Model, lớp Search Bean, tiến hành hai bước cho một đợt tìm kiếm, ví dụ tìm kiếm theo tên sản phẩm:
  + Phương thức FindByName: truy vấn để biết rows, tức số record kết quả tìm được.
  + Phương thức FindByNameNext: truy vấn tiếp để lấy đúng perpage record, từ chỉ số start + 1 đến chỉ số start + perpage.

Để thuận tiện cho người dùng quyền User, khi đang mua hàng, ta cho phép họ tìm sản phẩm, chọn sản phẩm từ kết quả tìm kiếm được để đưa vào Shopping Cart.

Chạy thử chương trình.

g) Shopping Cart

Phần mua hàng thuộc về project Shopping Cart, được ghép vào project này với những hiệu chỉnh thích hợp.

Chạy thử chương trình.

h) i18n

Tích hợp tính năng i18n, giao diện tiếng Trung quốc.

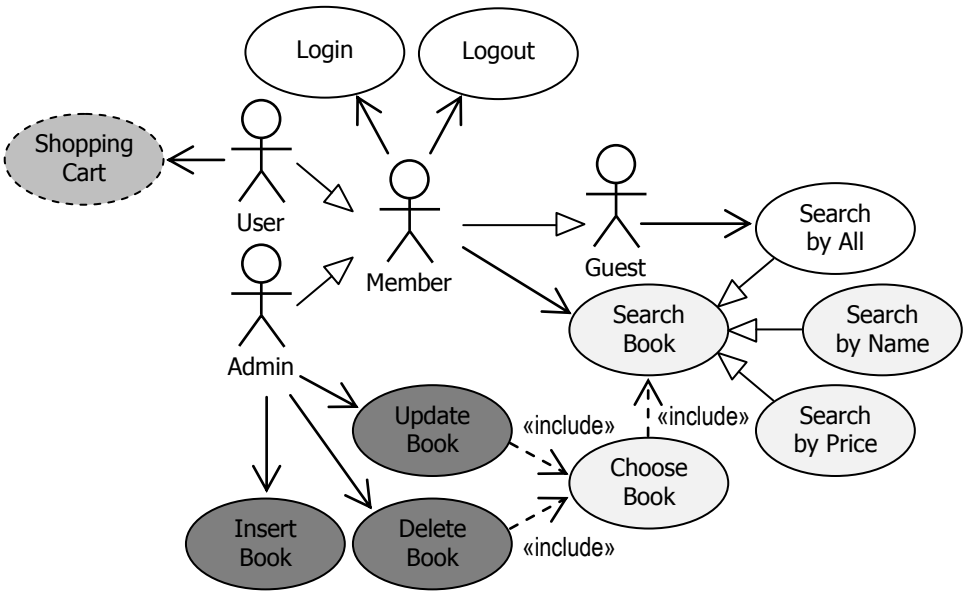
Để tích hợp tính năng i18n, cần phải chuẩn bị ngay từ khâu thiết kế giao diện. Tất cả các đoạn text cần chuyển ngữ, các đường dẫn đến hình ảnh thích hợp, được liệt kê trong tập tin tài nguyên package.properties, gọi là gói ngôn ngữ (bundle) của ứng dụng. Mỗi ngôn ngữ có một tập tin properties tương ứng.

Trong tập tin properties, thông tin được lưu có dạng key=value; key là khóa dùng tham chiếu đến đoạn text cần chuyển ngữ, value là đoạn văn bản thật sự cần hiển thị, có nội dung tùy theo gói ngôn ngữ chỉ định.
Ví dụ: cặp key-value tương ứng cho tiếng Anh (trong package.properties) và tiếng Trung quốc (trong package_cn.properties).
errors.password=Password is not empty!
errors.password=密碼不是空的！
Sau đó, trong giao diện ứng dụng web, dùng tag <fmt:message> của JSTL thay thế tất cả các đoạn văn bản, các đường dẫn đến hình ảnh thích hợp, với key của tag <fmt:message> tham chiếu đến key trong tập tin properties tương ứng với ngôn ngữ cần hiển thị. Ví dụ:
<fmt:message key="**errors.password**" />
Trong trang index.jsp, khi người dùng chọn ngôn ngữ, biến tầm vực Request **locale** chuyển cho JSTL dùng thiết lập ngôn ngữ. Sau đó, tag <fmt:bundle> của JSTL trong mỗi trang JSP bảo đảm hiển thị trang với ngôn ngữ chỉ định bằng cách lấy trị cần thiết trong tập tin properties tương ứng với locale đó.
Chạy thử chương trình.

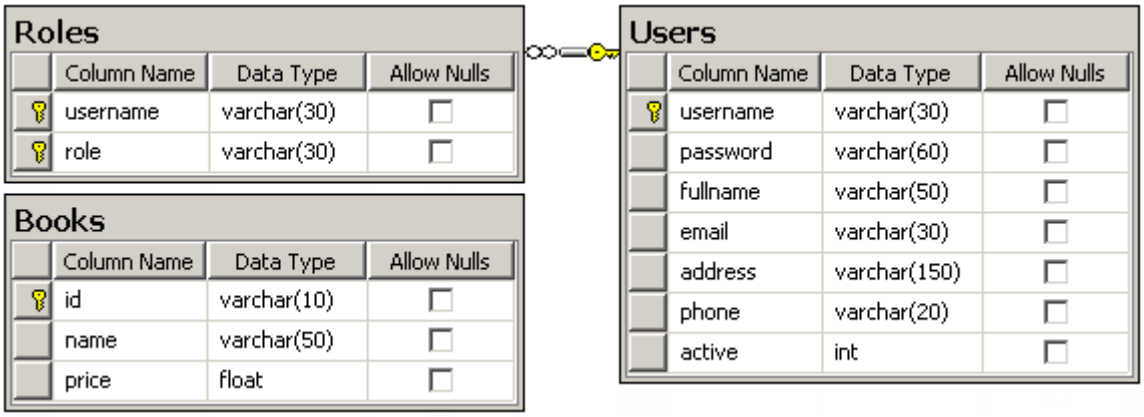
i) Bảo mật
Project này được bảo mật bằng lập trình. Vì vậy, bạn nên lưu một bản sao project tại thời điểm này để thực hiện *bảo mật bằng khai báo* trong một project khác.
Để lưu bản sao, trong tab Projects, click phải vào project cần sao lưu, chọn Copy… trong menu tắt.
Ta tiến hành bảo mật bằng lập trình giống như project Shopping Cart, tuy nhiên trong project này ta phân quyền chi tiết hơn.
Chạy thử chương trình.
Người dùng với quyền User, có thể mua hàng khi tìm kiếm sản phẩm.

4. Bảo mật bằng khai báo
Có hai cách bảo mật cho một ứng dụng web:
- Bảo mật bằng lập trình, đã trình bày trong project MVC.
- Bảo mật bằng khai báo, trình bày trong project này.
Trong project MVC, trước khi tiến hành bảo mật bằng lập trình và trước khi tiến hành i18n, ta đã lưu lại một bản sao. Bây giờ, ta thực hiện bảo mật bằng khai báo cho project bản sao đó.
Các bước chính thực hiện project:
- Thực hiện đầy đủ các chức năng yêu cầu, không quan tâm đến phân quyền. Kết quả ta có project bản sao của project MVC chưa tiến hành bảo mật.
- Thực hiện tách rời phần bảo mật cho project, cài đặt bảo mật bằng cách khai báo.
Thực hiện bảo mật bằng cách khai báo có vẻ phức tạp hơn, nhưng thực tế lại đơn giản vì chỉ cấu hình trong web.xml.

4.1. Đặc tả
Sơ đồ Use case:



4.2. Cơ sở dữ liệu
Project bản sao của project MVC, chưa tiến hành bảo mật. Như vậy ta chưa có login.jsp, LoginBean, LoginServlet và LogoutServlet. Ngoài ra, lớp Member chỉ chứa thông tin về khách hàng.
Ta cần điều chỉnh CartServlet và checkout.jsp, vì thông tin khách hàng trong checkout.jsp không lấy từ biến tầm vực Session **member**. Ta phải viết một **InfoBean** khác để lấy thông tin khách hàng khi cần tạo Invoice, CartServlet sẽ gọi InfoBean và chuyển thông tin khách hàng trong biến tầm vực Request **member** cho checkout.jsp.
Cơ sở dữ liệu:



4.3. Security Realm

Tomcat có nhiều kiểu "lãnh địa" bảo mật: JDBCRealm, DataSourceRealm, JNDIRealm, UserDatabaseRealm, JAASRealm, MemoryRealm. Đặc biệt có các kiểu lồng như CombinedRealm, LockOutRealm. Ví dụ:

```
<Realm className="org.apache.catalina.realm.LockOutRealm"
  failureCount="3" lockOutTime="300" cacheSize="1000" cacheRemovalWarningTime="3600">
  <Realm className="org.apache.catalina.realm.DataSourceRealm"
    dataSourceName="BooksDB" digest="MD5" localDataSource="true"
    userTable="Users" userNameCol="username" userCredCol="password"
    userRoleTable="Roles" roleNameCol="role"/>
</Realm>
```

Kiểu LockOutRealm trên cho phép tạm khóa tài khoản sau ba lần đăng nhập sai.

Trong META-INF/context.xml, ta thêm tag <Realm> khai báo "lãnh địa" bảo mật, kiểu DataSourceRealm:

```
<Context antiJARLocking="true" path="/MVCSecurityProject">
  <Resource name="BooksDB" type="javax.sql.DataSource" auth="Container"
            driverClassName="com.microsoft.sqlserver.jdbc.SQLServerDriver"
            maxActive="8" username="sa" password=""
            url="jdbc:sqlserver://localhost:1433;DatabaseName=GREEN;" />
  <Realm className="org.apache.catalina.realm.DataSourceRealm"
         dataSourceName="BooksDB" digest="MD5" localDataSource="true"
         userTable="Users" userNameCol="username" userCredCol="password"
         userRoleTable="Roles" roleNameCol="role" />
</Context>
```

Các thông tin cho realm này là:

- Tên nguồn dữ liệu (dataSourceName) dùng cho realm. Do sử dụng nguồn dữ liệu cục bộ, cùng khai báo trong context.xml, nên thuộc tính localDataSource là true.
- Cơ sở dữ liệu đăng nhập: bảng user (userTable), bảng role (userRoleTable), tên cột username là khóa liên kết hai bảng (userNameCol), tên cột password trong bảng user (userCredCol), tên cột role trong bảng role (roleNameCol). Chú ý các bảng này cũng giống project MVCProject để ta có thể so sánh hai cách bảo mật bằng lập trình và bằng khai báo.
- Mã hóa: để bảo mật cơ sở dữ liệu, password của người dùng được lưu dưới dạng MD5 (hàm băm bảo mật mã hóa một chiều). Web container sẽ mã hóa MD5 password do người dùng nhập rồi tiến hành so sánh với password mã hóa MD5 lưu trong cơ sở dữ liệu. Như vậy kẻ xâm nhập truy cập được cơ sở dữ liệu và ngay cả admin cũng không biết được mật khẩu của người dùng.

Ta dễ nhận thấy nếu dùng cơ chế khóa tài khoản riêng (cột active trong bảng Users), các kiểu "lãnh địa" bảo mật không hỗ trợ. Tuy nhiên, Tomcat cho phép viết kiểu "lãnh địa" bảo mật riêng, ta thừa kế DataSourceRealm và viết lại phương thức Principal authenticate(String username, String credentials) để thực hiện những yêu cầu bảo mật mong muốn.

4.4. Bảo mật ứng dụng web

a) Ràng buộc bảo mật

Quyền của một role được ràng buộc bảo mật với từng vùng web cụ thể (URL pattern). Cần hiểu rằng các vùng web này không nhất thiết phải vật lý mà chỉ là *vùng web ảo*, vì vậy không cần thiết đưa các trang vào *thư mục vật lý cụ thể*.

Các vùng web ảo bố trí như sau:

- Vùng admin, dùng cho role admin.
- Vùng user, dùng cho role user.
- Vùng member, dùng chung cho cả admin và user, chú ý là không cần role member.

Trước hết cần tạo các role. Trong web.xml, phần Security > Security Roles, click nút Add... thêm hai role là admin và user.

Tiếp theo, tiến hành ràng buộc bảo mật cho các role: phần Security > Security Constraints, click Add Security Constraint, thêm:

- adminConstraint có Web Resource Collection tên (Name) là adminResource, vùng (URL Pattern) là: /admin/*, /member/*. Phần Enable Authentication Constraint, click nút Edit và thêm role admin.
- userConstraint có Web Resource Collection tên (Name) là userResource, vùng (URL Pattern) là: /user/*, /member/*. Phần Enable Authentication Constraint, click nút Edit và thêm role user.

Trong web.xml, phần XML, xem các thay đổi: tag <security-role> và tag <security-constraint>.

Ta nhận thấy admin và user có quyền chung là Search Book, ta giải quyết bằng cách cho cả hai role này có quyền trên vùng web ảo /member/*.

b) Login và Logout

Phương pháp xác thực Basic không cho phép logout nên kém linh động. Trong project này, ta dùng phương pháp xác thực bằng Form. Thông tin đăng nhập lấy từ cơ sở dữ liệu.

Tạo trang **login.jsp**, chú ý các tên bắt buộc trong form đăng nhập: action là **j_security_check**, name của input User name là **j_username**, name của input Password là **j_password**.

Trong web.xml, phần Security > Login Configuration, click chọn Form, phần Form Login Page và Form Error Page đều là /login.jsp. Như vậy, khi đăng nhập không thành công, trang hiển thị vẫn là login.jsp.

Khi logout gọi LogoutServlet, servlet này dùng phương thức invalidate() của đối tượng session để xóa các biến tầm vực Session đang tồn tại. Sau đó chuyển về trang chủ index.jsp.

c) Thiết lập bảo mật

Trong web.xml, phần Servlets, vào từng servlet, ánh xạ lại các URL Pattern(s) cho phù hợp:

- Vùng admin: InsertServlet, DeleteServlet, UpdateServlet. Với các trang JSP, insert.jsp và update.jsp, dùng nút Add Servlet Element... ánh xạ thành các tên Insert và Update rồi cũng đưa vào vùng admin.
- Vùng user: CartServlet và Cart (ánh xạ cart.jsp), Checkout (ánh xạ checkout.jsp).
- Vùng member: SearchServlet, Show (ánh xạ index.jsp), Search (ánh xạ search.jsp) và Result (ánh xạ result.jsp).

Các servlet LoginServlet và LogoutServlet không cần bảo mật.

Bây giờ ta có các vấn đề cần giải quyết:

- Tên của người đăng nhập (username), dùng ${pageContext.servletContext.remoteUser}
- Vai trò của người đăng nhập (role), dùng ${pageContext.request.isUserInRole('admin')} (ví dụ để xác định role đó có phải là admin).
- Điều chỉnh đường dẫn (path) trong các servlet và trang JSP, dùng ${pageContext.servletContext.contextPath} với URL Pattern chính xác để gọi các servlet và các trang JSP. Nói cách khác, dùng đường dẫn tuyệt đối nếu gọi bên ngoài thư mục ảo.

Chạy thử chương trình.

Thư mục META-INF và WEB-INF của ứng dụng web cũng là vùng bảo mật "tự nhiên", nên thường dùng chứa các tập tin không muốn người dùng truy cập được bằng đường dẫn.

Welcome **hugo** [Logout]

Bookstore

[Create Book] [Search Book]

No	ID	Name	Price	Update	Delete
1	LOR1140	Lord of Scoundrels	3.4	[Update]	[Delete]
2	MAS0562	Master and Magarita	9.7	[Update]	[Delete]
3	MCK4625	McKenzie's Mountain	4.8	[Update]	[Delete]
4	MOR7328	Morning Glory	7.2	[Update]	[Delete]

[1] [2]

© Copyright by **Memorial Duck**
Fri, May 10, 2013

Người dùng với quyền Admin, có thể thêm, xóa, sửa và tìm kiếm sản phẩm.