



# LẬP TRÌNH VIÊN CÔNG NGHỆ JAVA

## Module 3

☞ Click vào phụ lục để chuyển tới bài cần đọc

### Phụ lục

Bài 1 Tổng quan J2EE .....	2
Bài 2 Servlet.....	22
Bài 3 Java Server Page .....	40
Bài 4 Servlet nâng cao .....	66
Bài 5 JSP nâng cao .....	89
Bài 6 EL & JSTL .....	110
Bài 7 Mô hình MVC với JSP/Servlet .....	122
Bài 8 Web Services .....	135
Bài 9 Bảo mật Website .....	155



Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh  
TRUNG TÂM TIN HỌC

[Go Screen Capture](#)

## LTV CÔNG NGHỆ JAVA

Module 3 – Bài 1: *Tổng quan J2EE*

Ngành LT & CSDL

[www.t3h.vn](http://www.t3h.vn)



2014

5014



## Nội dung

1. Giới thiệu
2. Các thành phần J2EE
3. Các khái niệm cơ bản về Netwoking
4. Giao thức http
5. Cài đặt môi trường
6. Phát triển ứng dụng web tĩnh



## Giới thiệu

- **Công nghệ J2EE hỗ trợ phát triển ứng dụng Web**
- **J2EE cung cấp API cho việc phát triển ứng dụng, nhằm**
  - Giảm thời gian phát triển ứng dụng
  - Giảm độ phức tạp của ứng dụng
  - Tăng hiệu suất ứng dụng



## Mô hình ứng dụng J2EE



- Là kiến trúc ứng dụng đa tầng (multi-tier), với ưu điểm
  - Khả năng mở rộng
  - Khả năng truy cập
  - Khả năng quản lý
- Mô hình kiến trúc chia làm 2 tầng
  - Tầng trình diễn (presentation tier)
  - Tầng nghiệp vụ (business tier)



## Mô hình ứng dụng J2EE



- J2EE cung cấp hệ thống các dịch vụ cho việc phát triển ứng dụng web-based
- Các nhà phát triển có thể dựa vào nền tảng này để cung cấp các giải pháp để khắc phục các vấn đề khó khăn của hệ thống phát triển một dịch vụ đa tầng

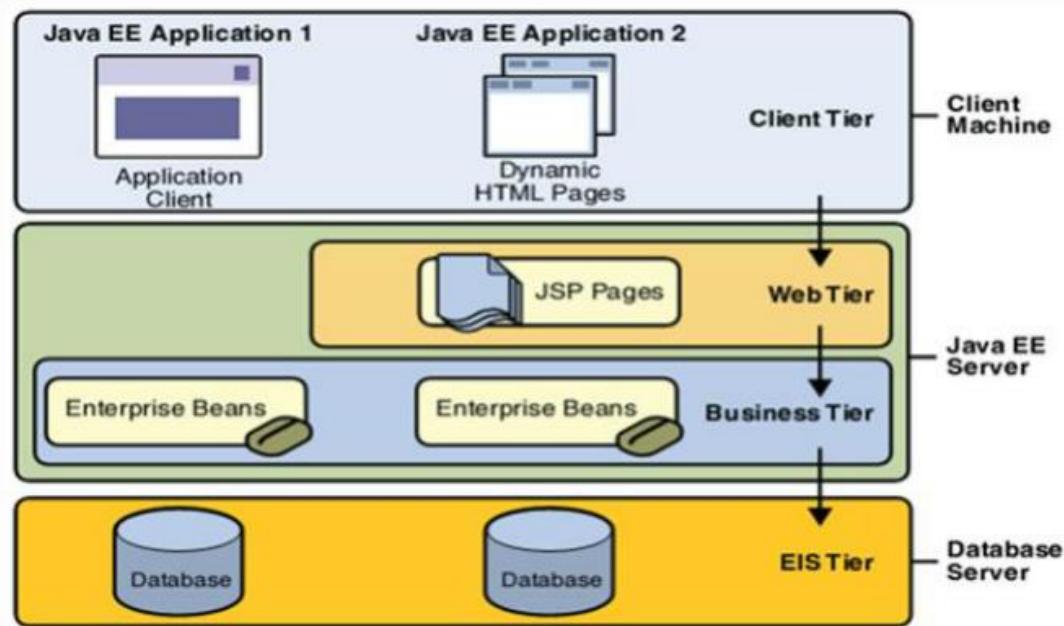


## Nội dung

1. Giới thiệu
2. Các thành phần J2EE
3. Các khái niệm cơ bản về Networking
4. Giao thức http
5. Cài đặt môi trường
6. Phát triển ứng dụng web tĩnh



## Các thành phần J2EE

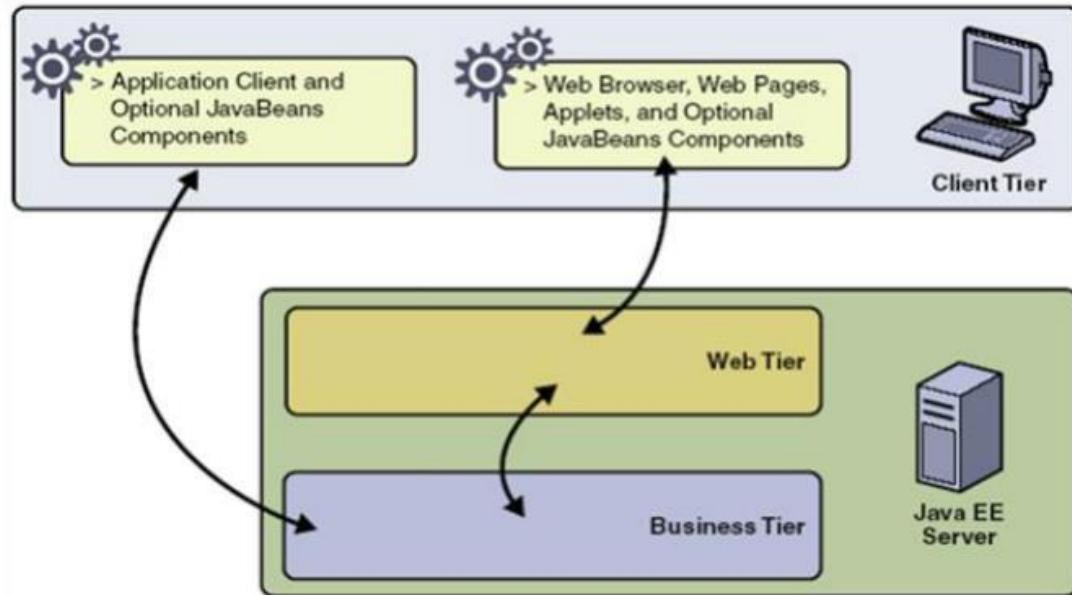


## Java EE client

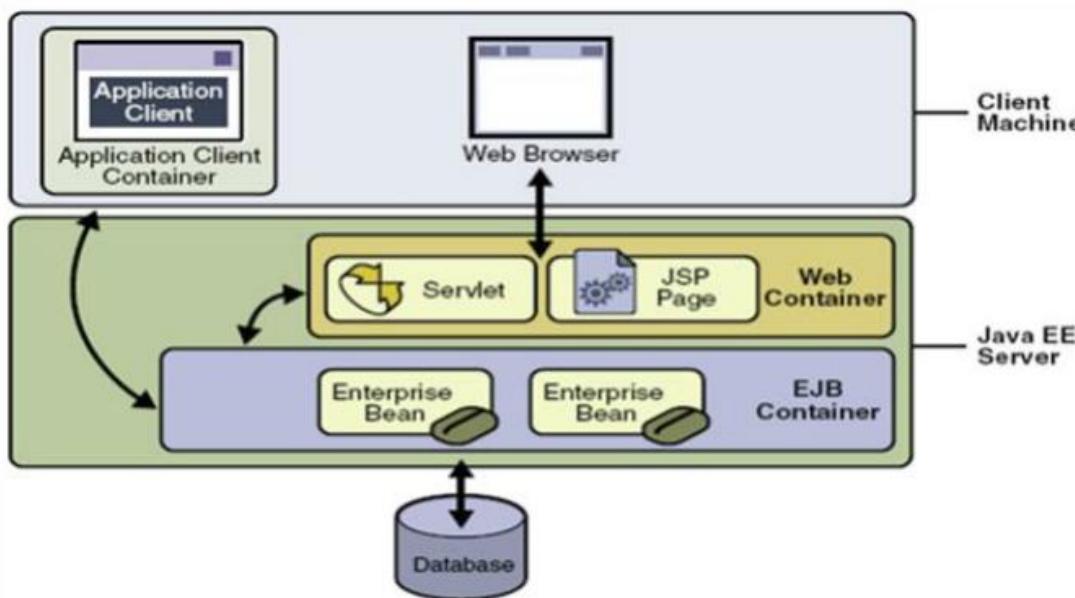
- Web Clients
- Applets
- Application Clients
- JavaBeans Component



## Giao tiếp giữa client-server



## Java EE container



## Các dịch vụ trong Container



- **Container trong JEE cung cấp các dịch vụ**
  - Dịch vụ bảo mật (security service)
  - Dịch vụ giao dịch (transaction service)
  - Dịch vụ JNDI (JNDI lookup service)
  - ...



## Phân loại container



### ▪ Enterprise JavaBeans (EJB) container

- Quản lý việc thực thi của enterprise bean cho các ứng dụng Java EE

### ▪ Web container

- Quản lý việc thực thi của các thành phần của trang JSP và servlet cho các ứng dụng Java EE



## Phân loại container



### ▪ Application client container

- Quản lý việc thực thi của các thành phần ứng dụng client

### ▪ Applet container

- Quản lý việc thực thi các applet. Bao gồm web browser và Java Plug-in chạy trên client với nhau



## Nội dung



1. Giới thiệu
2. Các thành phần J2EE
3. Các khái niệm cơ bản về Networking
4. Giao thức http
5. Cài đặt môi trường
6. Phát triển ứng dụng web tĩnh



## Khái niệm Network



- Là một tập hợp các máy tính và các thiết bị khác mà có thể gửi và nhận dữ liệu từ nơi này qua nơi khác thông qua các phương tiện vật lý



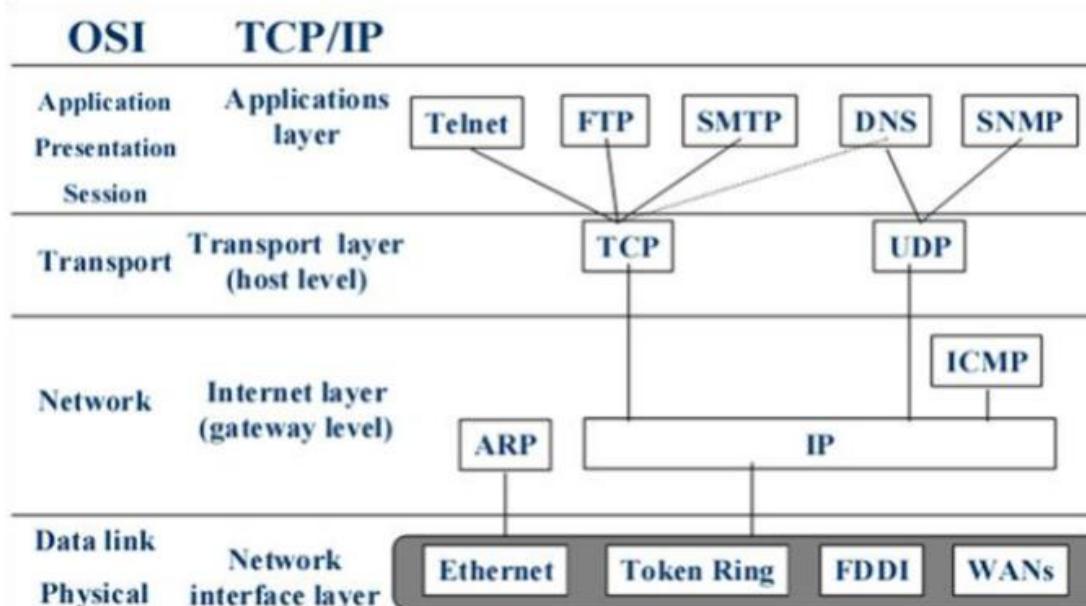
## Giao thức (protocol)



- Là một quy ước hoặc tiêu chuẩn để kiểm soát hoặc kết nối, giao tiếp và truyền dữ liệu giữa hai thiết bị máy tính đầu cuối
- Hầu hết các giao thức truyền thông của Internet được mô tả trong các tài liệu của RFC (Request for Comments) thuộc tổ chức IETF (Internet Engineering Task Force)



## Mô hình network



## Địa chỉ IP



- Là một địa chỉ duy nhất mà thiết bị điện tử hiện đang sử dụng và được dùng để xác định thiết bị trong môi trường mạng
- Phân loại
  - IPv4: 32bit
  - IPv6: 128bit



## URI, URN và URL



- Uniform Resource Identifier (URI) là một chuỗi nhỏ gọn các ký tự được sử dụng để xác định hoặc đặt tên một nguồn tài nguyên
- URI có thể được phân loại như một bộ định vị (URL – Uniform Resource Location) hoặc một cái tên (URN – Uniform Resource Name) hoặc cả hai



## DNS – Domain Name System

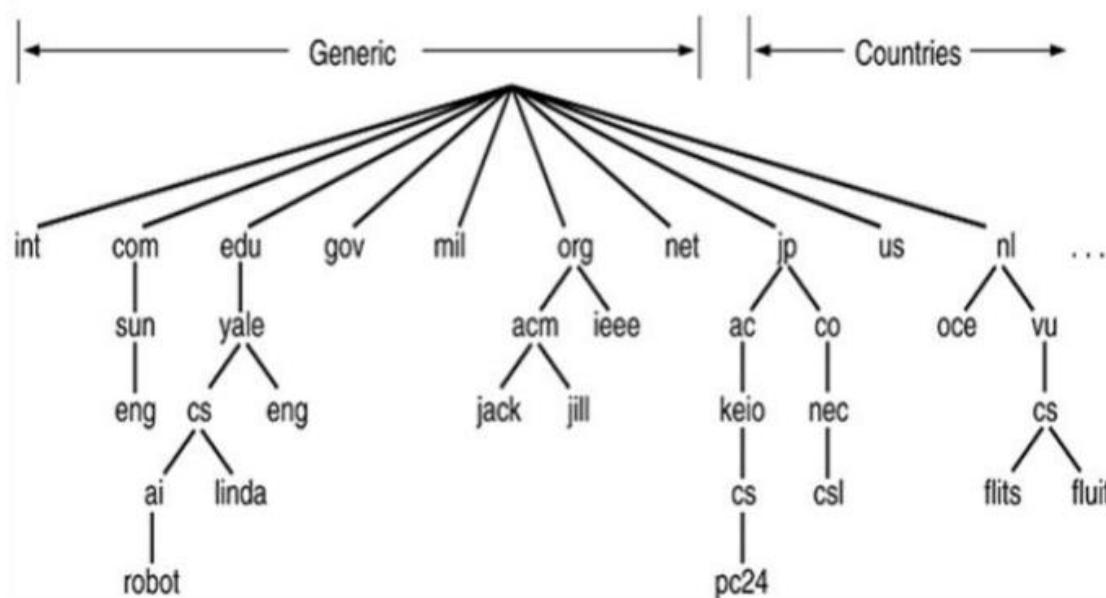


- Ánh xạ tên máy sang địa chỉ IP

192.31.7.130	CISCO.COM
204.71.177.35	YAHOO.COM
152.163.210.7	AOL.COM
198.150.15.234	MAT-MADISON.COM
207.46.131.15	MICROSOFT.COM
192.233.80.9	NOVELL.COM



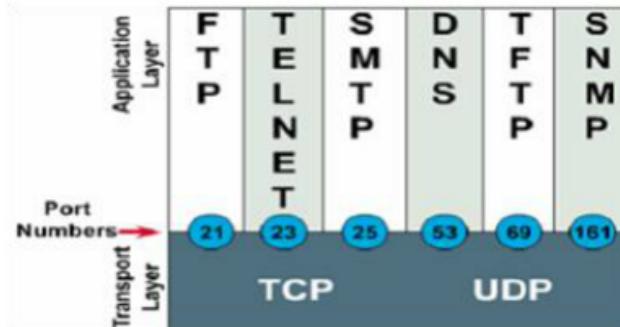
## DNS name space



## Port



- Port là một số đặc biệt hiện diện trong tiêu đề của một gói dữ liệu
- Port thường được sử dụng để ánh xạ dữ liệu cho một quá trình cụ thể đang chạy trên một máy tính



## Nội dung



1. Giới thiệu
2. Các thành phần J2EE
3. Các khái niệm cơ bản về Netwoking
4. Giao thức http
5. Cài đặt môi trường
6. Phát triển ứng dụng web tĩnh



## HTTP



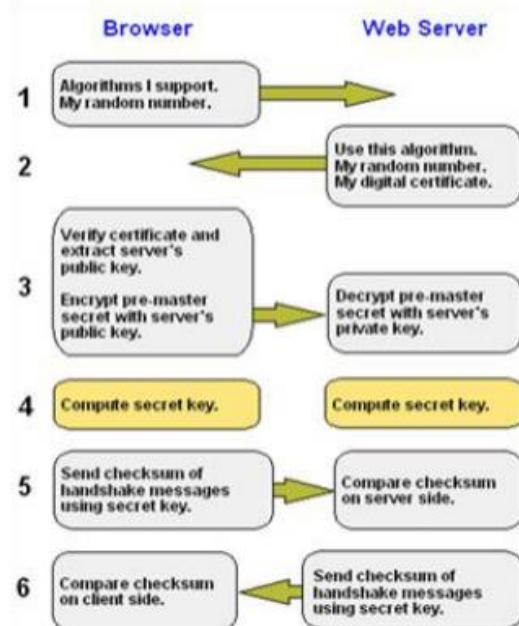
- **Http - Hyper Text Transfer Protocol**
- **Giao thức chuẩn giao tiếp giữa client-server trên www (world wide web)**
- **Port mặc định 80**
- **Phiên bản hiện tại 1.2**



## HTTPs



- **HTTP trên SSL (Secure Socket Layer) hoặc TLS (Transport Layer Security)**
- **Port mặc định là 443**



## HTTP(s)



### ▪ Phương thức

- GET, HEAD, POST, PUT, ...

### ▪ Status code

- 1xx Informational: 101 Switching Protocols
- 2xx Success: 200 OK
- 3xx Redirection: 305 Use Proxy
- 4xx Client Error: 404 File Not Found
- 5xx Server Error: 500 Internal Error



## Nội dung



1. Giới thiệu
2. Các thành phần J2EE
3. Các khái niệm cơ bản về Netwoking
4. Giao thức http
5. Cài đặt môi trường
6. Phát triển ứng dụng web tĩnh

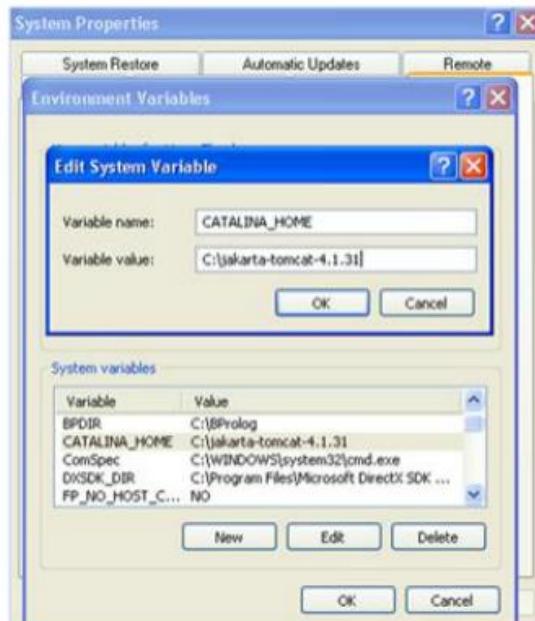


## Môi trường

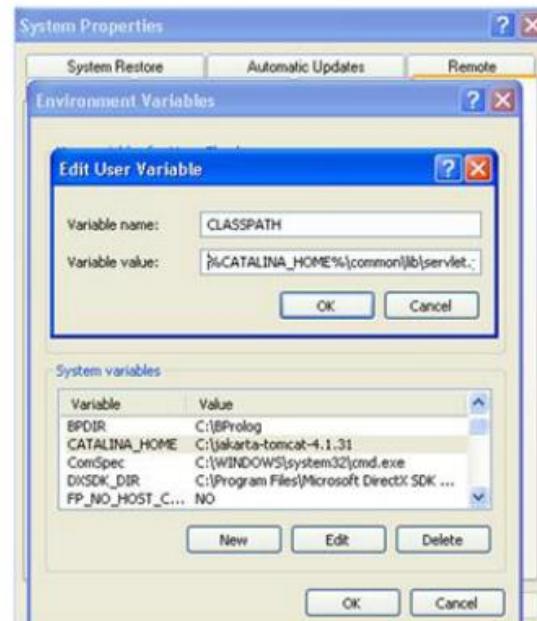
- Java SDK (J2SE)
- Eclipse IDE
- Application Server (Tomcat)
- Tomcat Eclipse plugin



## Thiết lập biến môi trường



Set up CATALINA\_HOME



Set up CLASSPATH



## Nội dung



1. Giới thiệu
2. Các thành phần J2EE
3. Các khái niệm cơ bản về Networking
4. Giao thức http
5. Cài đặt môi trường
6. Phát triển ứng dụng web tĩnh



## Ứng dụng web-based



- Client-side
  - Html, Javascript, css
  - VBScript
  - Applet
- Server-side
  - CGI – Common Gateway Interface
  - Servlet/JSP
  - ASP.Net
  - PHP



## HTML



- **HTML (Hyper Text Markup Language - Ngôn ngữ đánh dấu siêu văn bản)** là một **sự định dạng để báo cho trình duyệt Web (Web browser) biết cách để hiển thị một trang Web.**
- **Các trang Web là văn bản cùng với các thẻ (tag) HTML được sắp xếp đúng cách hoặc các đoạn mã để trình duyệt Web biết cách để thông dịch và hiển thị chúng lên trên màn hình**



## Cấu trúc HTML page



- **<!Doctype>**
  - Phần khai báo chuẩn của html hay xhtml.
- **<head></head>**
  - Phần khai báo ban đầu, khai báo về meta, title, css, javascript...
- **<body></body>**
  - Phần chứa nội dung của trang web, nơi hiển thị nội dung
- **Chú thích sử dụng <!-- và -->.**



## CSS



- **Styles định nghĩa cách các thành phần HTML hiển thị như thế nào.**
- **Các Styles thông thường được lưu trữ trong một Style Sheets**



## CSS



- **Có ba cách chèn Style:**
  - External Style Sheets
  - Internal Style Sheets
  - Inline Style
- **External Style Sheets được lưu trong những tệp có phần mở rộng là CSS.**
- **Hoạt động dựa trên bộ chọn (selectors) & thuộc tính**



## JavaScript



### ▪ Là ngôn ngữ thông dịch

- chương trình nguồn của nó được nhúng hoặc tích hợp vào tập tin HTML
- Khi trang web được tải trong trình duyệt hỗ trợ javascript
- Trình duyệt sẽ thông dịch và thực hiện các lệnh Javascript.



## JavaScript



### ▪ Chương trình nguồn Javascript được thông dịch trong trang HTML

- Sau khi toàn bộ trang được load
  - Nhưng trước khi trang được hiển thị
- ### ▪ Có 2 cách để nhúng Javascript vào trong tập tin HTML
- Trực tiếp trong file HTML
  - Sử dụng tập tin javascript bên ngoài







Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh  
TRUNG TÂM TIN HỌC

[Go Screen Capture](#)

## LTV CÔNG NGHỆ JAVA

Module 3 – Bài 2: *Servlet*

Ngành LT & CSDL

[www.t3h.vn](http://www.t3h.vn)



2014

2014



## Nội dung



1. Giới thiệu
2. Servlet Container
3. Servlet API
4. Cấu trúc và triển khai ứng dụng



## Java Servlet



- Các lớp Java cho lập trình phía máy chủ và sử dụng cho tầng xử lý nghiệp vụ (business tier)
- JSR-154
- Phiên bản hiện tại 3.0
- Điều khiển kết nối giữa client – server
  - Tạo kết nối
  - Đáp ứng dữ liệu
  - Đóng kết nối



## Java Servlet

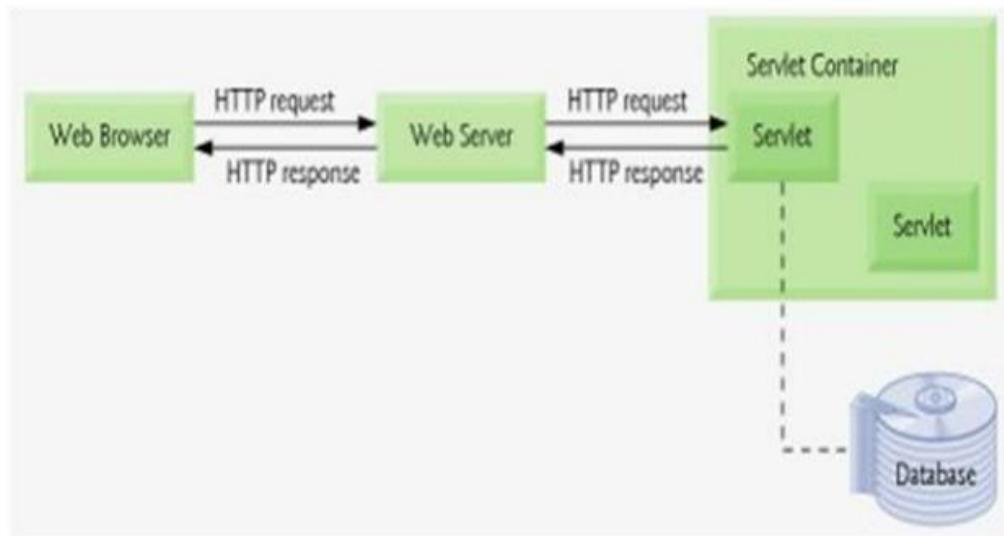


### ▪ Hiện thực mô hình client – server

- Giao thức HTTP
- Giao thức FTP



## Kiến trúc hoạt động của Servlet



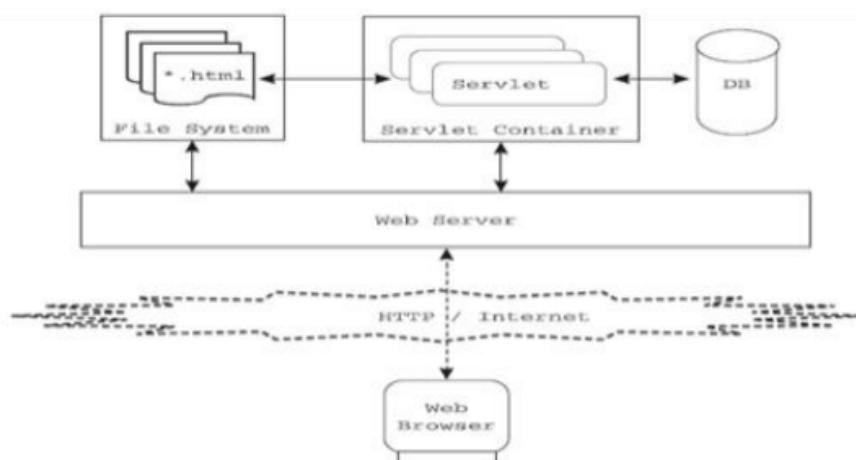
## Nội dung

1. Giới thiệu
2. Servlet Container
3. Servlet API
4. Cấu trúc và triển khai ứng dụng



## Servlet container

- Một phần của Web container
- Môi trường để chạy ứng dụng servlet



## Phân loại Servlet container



### ▪ Standalone

- Web container và servlet container là một phần không thể tách rời của một chương trình duy nhất

### ▪ In-process

- Web container và servlet container là những chương trình khác nhau nhưng chạy trong cùng một web container



## Phân loại Servlet container

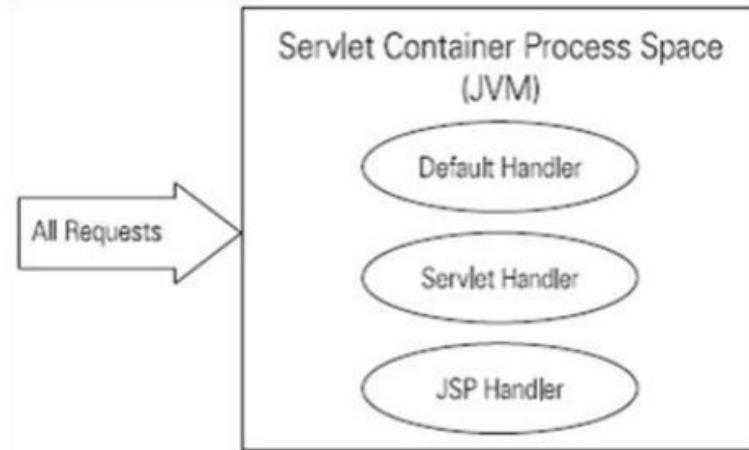


### ▪ Out-of Process

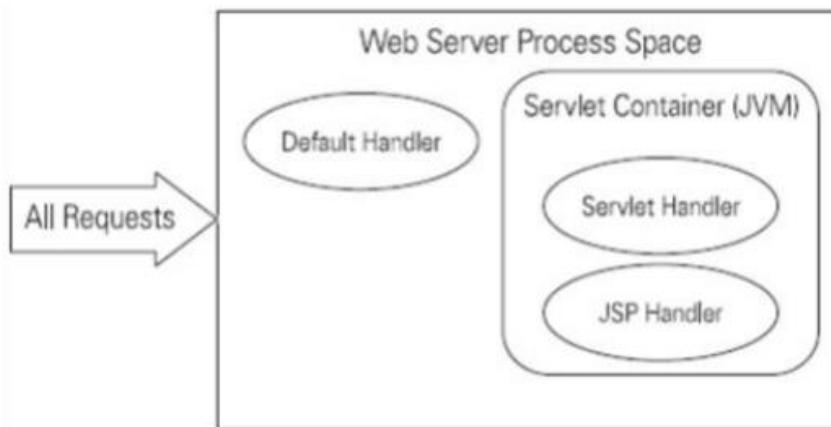
- Web container và servlet container là những chương trình khác nhau và giao tiếp qua mạng



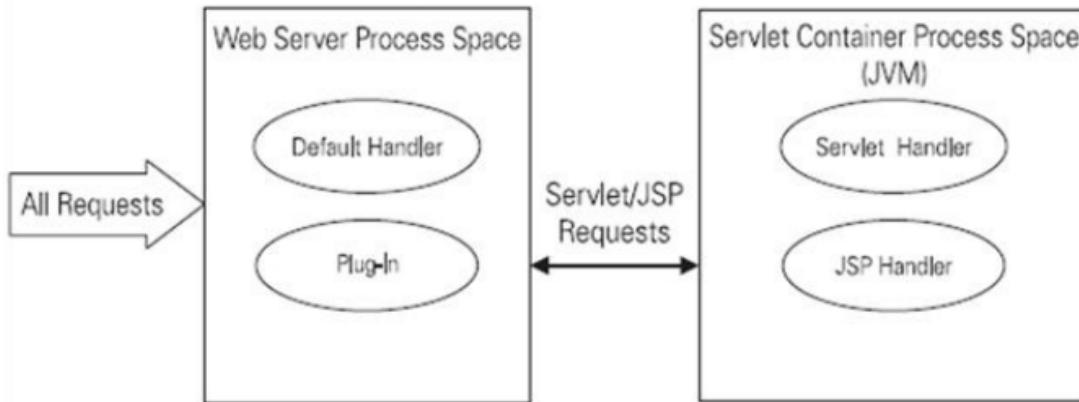
## Standalone



## In-process



## Out-of Process

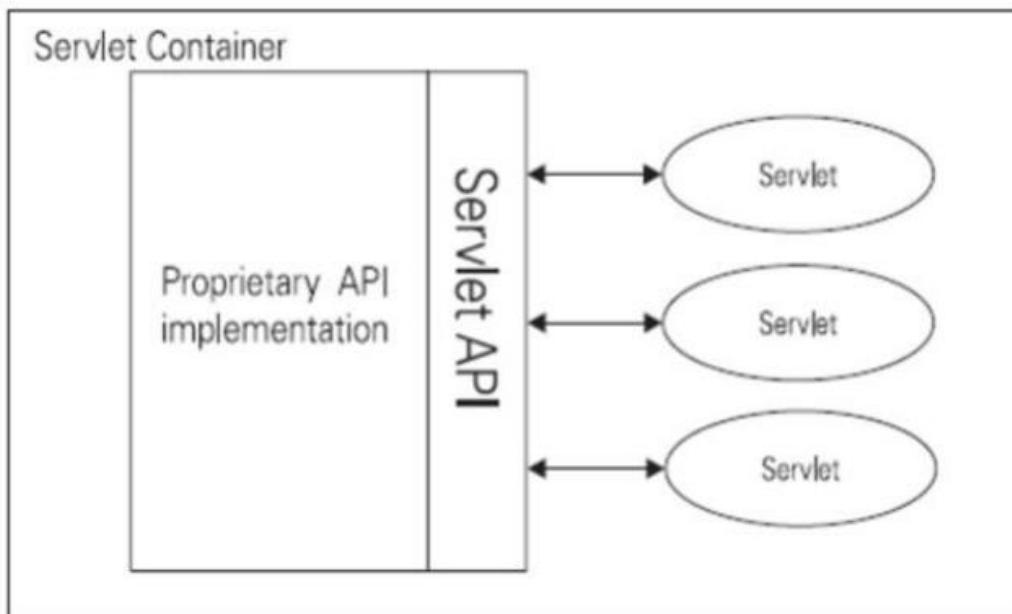


## Nội dung

1. Giới thiệu
2. Servlet Container
3. **Servlet API**
4. Cấu trúc và triển khai ứng dụng



## Servlet API



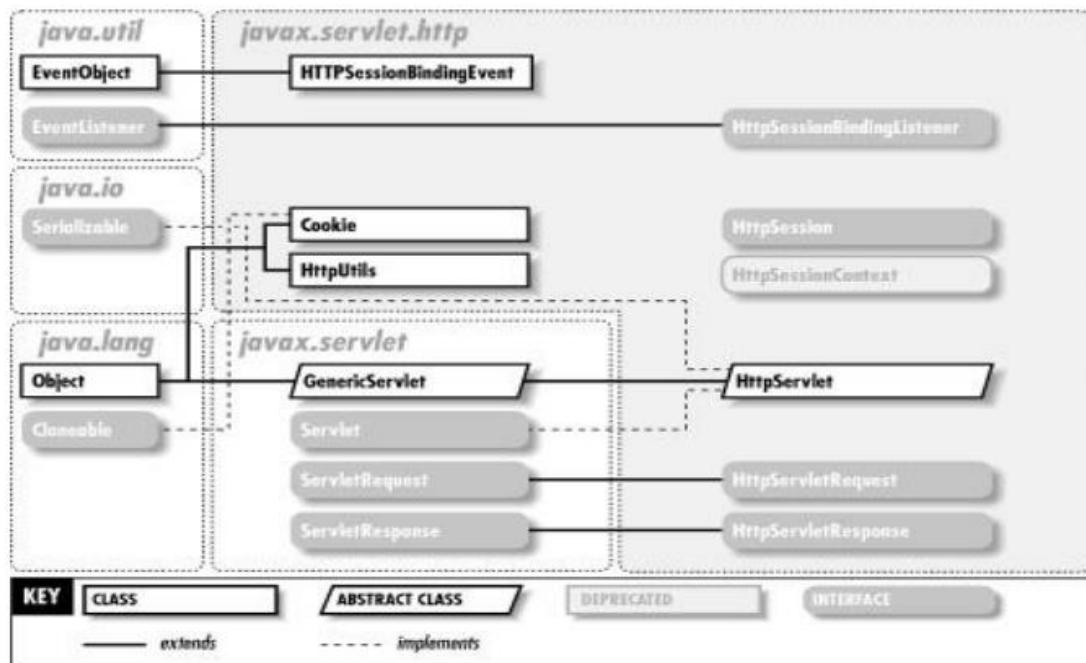
## Servlet API



- **Packages :**
  - javax.servlet
  - javax.servlet.http
- **Gói javax.servlet chứa các giao diện servlet chung và các lớp độc lập của giao thức bất kỳ**
- **Gói javax.servlet.http để xây dựng hỗ trợ cho giao thức HTTP**



## Servlet API



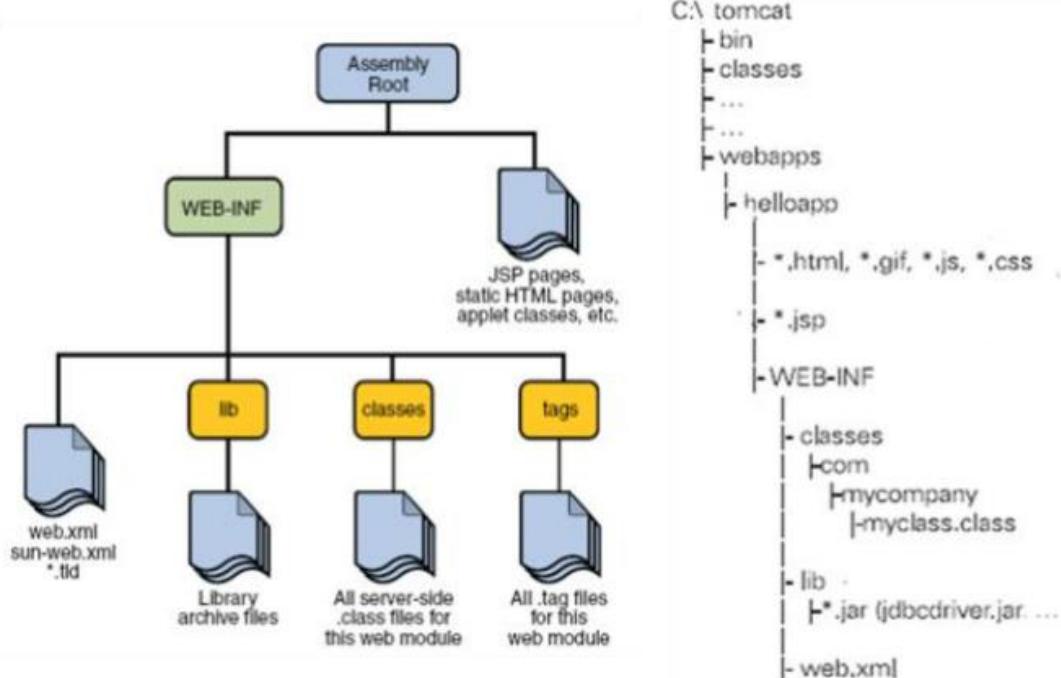
## Nội dung

1. Giới thiệu
2. Servlet Container
3. Servlet API
4. Cấu trúc và triển khai ứng dụng

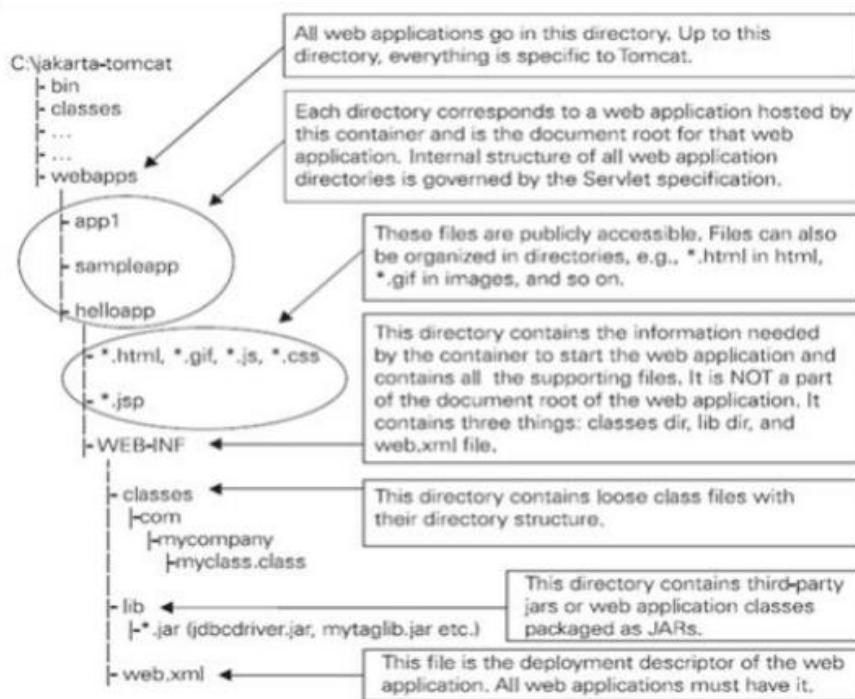




## Cấu trúc ứng dụng Servlet



## Cấu trúc ứng dụng Servlet



## Thư mục WEB-INF



- Nằm trong thư mục gốc của ứng dụng
- Trong thư mục này có:
  - Thư mục classes: các tập tin lớp servlet và các tập tin lớp cần thiết để hỗ trợ các servlet hoặc trang JSP
  - Thư mục lib : Tất cả các tập tin JAR / Zip được sử dụng bởi các ứng dụng web
  - Tập tin web.xml (mô tả triển khai)



## Cấu hình web.xml



- Deployment descriptor
- Mô tả cấu hình trong ứng dụng web trong container
- Theo cấu trúc xml
  - DTD cho xml được chuẩn hóa bởi Sun



## Cấu hình web.xml



Web Application Properties	Short Description
Servlet Declarations	Used to specify servlet properties.
Servlet Mappings	Used to specify URL to servlet mapping.
Application Lifecycle Listener classes	Used to specify listener classes for HttpSession-Events and ServletContextAttributeEvent.
ServletContext Init Parameters	Used to specify initialization parameters for the web application.
Error Pages	Used to specify error pages for error conditions.
Session Configuration	Used to specify session timeout.
Security Constraints	Used to specify security requirements of the web application.
Tag libraries	Used to specify the tag libraries required by JSP pages.
Welcome File list	Used to specify the welcome files for the web application.
Filter Definitions and Filter Mappings	Used to specify the filter.
MIME Type Mappings	Used to specify MIME types for common file extensions.
JNDI names	Used to specify JNDI names of the EJBs.



```

<?xml version="1.0" encoding="ISO-8859-1">
<!DOCTYPE web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_3.dtd">
<web-app>
    <display-name>Test Webapp</display-name>
    <context-param>
        <param-name>author</param-name>
        <param-value>john@abc.com</param-value>
    </context-param>
    <servlet>
        <servlet-name>test</servlet-name>
        <servlet-class>com.abc.TestServlet</servlet-class>
    <init-param>
        <param-name>greeting</param-name>
        <param-value>Good Morning</param-value>
    </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>test</servlet-name>
        <url-pattern>/test/*</url-pattern>
    </servlet-mapping>
    <mime-mapping>
        <extension>zip</extension>
        <mime-type>application/zip</mime-type>
    </mime-mapping>
</web-app>

```



## The welcome-file-list



- **Đặc tả thứ tự nạp danh sách trang đầu tiên (trang chủ) của ứng dụng**
  - Trang html hoặc jsp
- **Chứa các thẻ welcome-file**

```
<welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
</welcome-file-list>
```



## The servlet



- **DTD**
  - <!ELEMENT servlet (icon?, servlet-name, display-name?, description?, (servlet-class|jsp-file), init-param\*, load-on-startup?, security-role-ref\*)>

```
<servlet>
    <servlet-name>us-sales</servlet-name>
    <servlet-class>com.xyz.SalesServlet</servlet-class>
    <init-param>
        <param-name>region</param-name>
        <param-value>USA</param-value>
    </init-param>
    <init-param>
        <param-name>limit</param-name>
        <param-value>200</param-value>
    </init-param>
</servlet>
```



## The servlet-mapping



- DTD

- <!ELEMENT servlet-mapping (servlet-name, url-pattern)>

```
<servlet-mapping>
    <servlet-name>accountServlet</servlet-name>
    <url-pattern>/account/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>accountServlet</servlet-name>
    <url-pattern>/myaccount/*</url-pattern>
</servlet-mapping>
```



## Ánh xạ URL tới servlet



- Triệu gọi một yêu cầu tới servlet trải qua 2 bước:

- Servlet container xác định web application

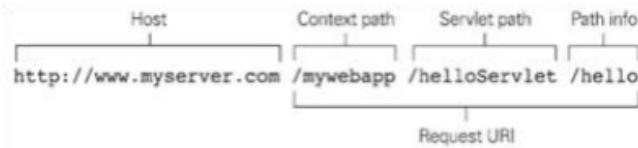
- Tìm servlet phù hợp dựa vào cấu hình web.xml

- Cả 2 bước đều phân chia URI thành 2 chuỗi con:

- Context path

- Servlet path

- Path info



## Cách xác định Servlet path



- **URI được so trùng với servlet-mapping trong web.xml**
- **Nếu thành phần cuối cùng của URI là có phần mở rộng (như .html, .jsp), servlet container so trùng nó với servlet xử lý yêu cầu đó**
  - Trong trường hợp này thì toàn bộ URI là servlet path và path info là null



## Cách xác định Servlet path



- **Nếu Servlet container không tìm thấy thì container điều hướng servlet mặc định**

/colorapp/red	Servlet Used	Servlet Path	Path Info	Comments
/colorapp/red/	RedServlet	/red	null	See Step 1.
/colorapp/red/aaa	RedServlet	/red	/	See Step 2.
	RedServlet	/red	/aaa	See Step 2.



## Cách xác định Servlet path

```
<servlet-mapping>
    <servlet-name>ColorServlet</servlet-name>
    <url-pattern>*.col</url-pattern>
</servlet-mapping>
```

Request URI	Servlet Used	Servlet Path	Path Info	Comments
/colorapp/aa.col	ColorServlet	/aa.col	null	*.col is mapped to ColorServlet. See Step 3.
/colorapp/hello/aa.col	ColorServlet	/hello/aa.col	null	/hello/aa.col matches with *.col, so the servlet path is /hello/aa.col and the path info is null. See Step 3.



## Demo: HelloWorldServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloWorldServlet extends HttpServlet
{
    public void service( HttpServletRequest request,
                        HttpServletResponse response)
                        throws ServletException, IOException
    {
        PrintWriter pw = response.getWriter();
        pw.println("<html>");
        pw.println("<head>");
        pw.println("</head>");
        pw.println("<body>");
        pw.println("<h3>Hello World!</h3>");
        pw.println("</body>");
        pw.println("</html>");
    }
}
```



## Demo: web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
    <display-name>Hello World</display-name>
    <description>
        My First Example
    </description>

    <servlet>
        <servlet-name>HelloWorld</servlet-name>
        <servlet-class>HelloWorldServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>HelloWorld</servlet-name>
        <url-pattern>/servlet/HelloWorld</url-pattern>
    </servlet-mapping>

</web-app>
```



## Triển khai ứng dụng lên Server

- **Đóng gói tập tin war và lưu trữ vào thư mục web của application server**
  - Tomcat là thư mục webapps
- **Tomcat tự động trích xuất nội dung tập tin WAR vào một thư mục dưới ứng dụng web**
- **Tạo tập tin WAR :**  
**jar-CVF hello.war \***







Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh  
TRUNG TÂM TIN HỌC

[Go Screen Capture](#)

## LTV CÔNG NGHỆ JAVA

Module 3 – Bài 3: *Java Server Page*

Ngành LT & CSDL

[www.t3h.vn](http://www.t3h.vn)



2014

5014



## Nội dung



1. JSP là gì?
2. Vòng đời của JSP
3. JSP Tags
4. Biến ẩn trong trang JSP (Implicit variable)
5. Session, Cookies



## JSP



- **JSP là một kỹ thuật tầng web-tier, bổ sung các đặc điểm kỹ thuật của Servlet và rất hữu ích trong việc phát triển các giao diện web**
- **JSP là một công nghệ kết hợp các ngôn ngữ HTML / XML và các yếu tố của ngôn ngữ lập trình Java để trả về nội dung động cho web clients**
- **Công nghệ cho tầng trình diễn**



## JSP



- Trang JSP là một trang chứa Java code kết hợp với html
- JSP định nghĩa một số các thẻ (tag) để tạo ra web page, gọi là JSP tag
- JSR – 245 và phiên bản hiện tại 3.0
- JSP API: trong 2 package
  - javax.servlet.jsp
  - javax.servlet.jsp.tagext



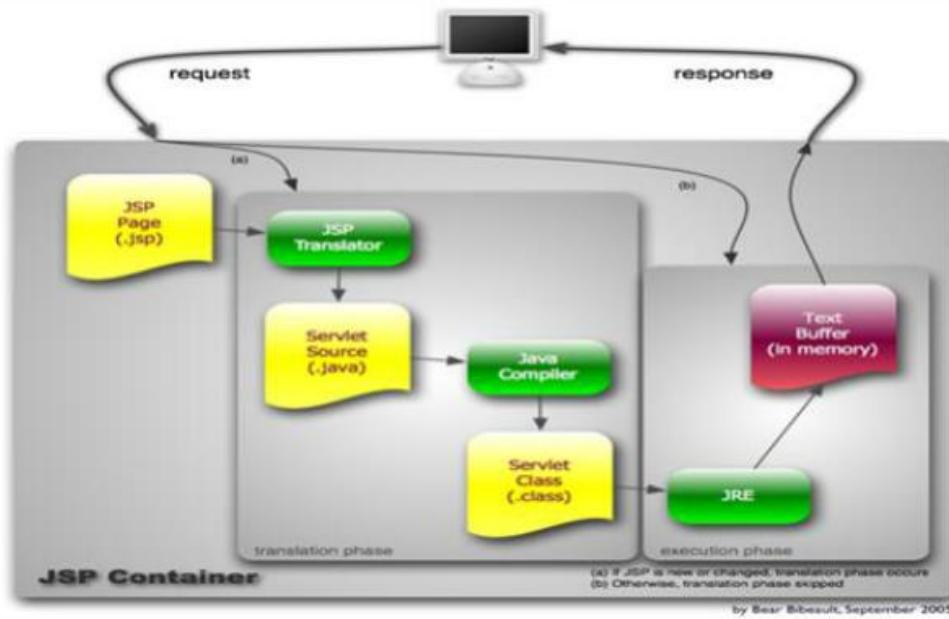
## Nội dung



1. JSP là gì?
2. Vòng đời của JSP
3. JSP Tags
4. Biến ẩn trong trang JSP (Implicit variable)
5. Session, Cookies



## Vòng đời trang JSP



## Vòng đời trang JSP

Phase Name	Description
Page translation	The page is parsed and a Java file containing the corresponding servlet is created.
Page compilation	The Java file is compiled.
Load class	The compiled class is loaded.
Create instance	An instance of the servlet is created.
Call <code>jspInit()</code>	This method is called before any other method to allow initialization.
Call <code>_jspService()</code>	This method is called for each request.
Call <code>jspDestroy()</code>	This method is called when the servlet container decides to take the servlet out of service.



## Translation phase



- JSP Engine đọc, phân tích (parse) và kiểm tra cấu trúc của JSP page
- Ngoài ra, còn có kiểm tra khác
  - Key-values trong khai báo (directives) là hợp lệ
  - Không tồn tại các JavaBean trùng
  - Trong trường hợp dùng Custom Tag Library, kiểm tra có tồn tại các thư viện hợp lệ
  - Cấu trúc của custom tags hợp lệ



## Compilation phase



- Java file được sinh ra trong bước trước được biên dịch
- Kiểm tra tất cả java code hợp lệ
- Có thể biên dịch trang jsp mà không cần thực thi thực sự dùng precompilation request parameter `jsp_precompile`

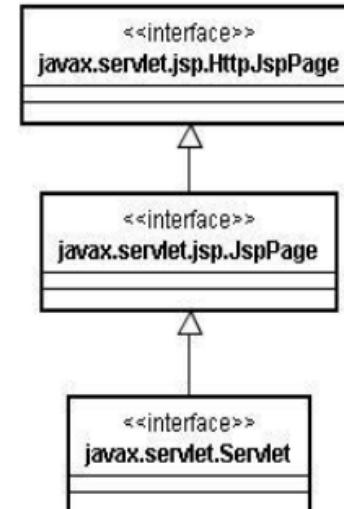
```
http://localhost:8080/counter.jsp?jsp_precompile=true
```



## Execution phase



- Container nạp servlet class vào trong bộ nhớ và tạo ra thực thể của servlet
- Gọi thực thi các phương thức
  - jsplninit()
  - \_jspService()
  - jspDestroy()



```

<% page language="java" import="java.io.*" %>
<!--
    // A variable to maintain the number of visits.
    int count = 0;
    // Path to the file, counter.db, which stores the count
    String dbPath;
    // read the integer value, and initialize the count variable.
    public void jsplninit()
    {
        try
        {
            dbPath = getServletContext().getRealPath("/WEB-INF/counter.db");
            FileInputStream fis = new FileInputStream(dbPath);
            DataInputStream dis = new DataInputStream(fis);
            count = dis.readInt();
            dis.close();
        }
        catch(Exception e)
        {
            getServletContext().log("Error loading persistent counter", e);
        }
    }
    <!--
        This will become a part of the generated _JspService() method
    -->
    <html><body>
    <% count++; %>
    Welcome! You are visitor number
    <% count %>
    </body></html>
<%!
    // This method is called by the container
    public void jspDestroy()
    {
        try
        {
            FileOutputStream fos = new FileOutputStream(dbPath);
            DataOutputStream dos = new DataOutputStream(fos);
            dos.writeInt(count);
            dos.close();
        }
        catch(Exception e)
        {
            getServletContext().log("Error storing persistent counter", e);
        }
    }
%>
  
```



## Nội dung



1. JSP là gì?
2. Vòng đời của JSP
3. **JSP Tags**
4. Biến ẩn trong trang JSP (Implicit variable)
5. Session, Cookies



## Các loại thẻ JSP



JSP Tag Type	Brief Description	Tag Syntax
Directive	Specifies translation time instructions to the JSP engine.	<%@ Directives %>
Declaration	Declares and defines methods and variables.	<%! Java Declarations %>
Scriptlet	Allows the developer to write free-form Java code in a JSP page.	<% Some Java code %>
Expression	Used as a shortcut to print values in the output HTML of a JSP page.	<%= An Expression %>
Action	Provides request-time instructions to the JSP engine.	<jsp:actionName />
Comment	Used for documentation and for commenting out parts of JSP code.	<%-- Any Text --%>



## Directive



- **Cung cấp thông tin tổng quát về các trang JSP cho JSP engine**
- **Có ba loại của các directive**
  - page: Thông báo cho các engine về tính chất tổng thể của một trang JSP
  - include: chỉ thị cho JSP engine để đưa vào các nội dung của tập tin khác
  - taglib: được sử dụng để kết hợp một tiền tố với một thẻ library



## Directive



- **Luôn luôn bắt đầu với <%@ và kết thúc với %>**
- **Cú pháp**

```
<%@ page attribute-list %>
<%@ include attribute-list %>
<%@ taglib attribute-list %>
```

**attribute-list: bao gồm một hoặc nhiều cặp attribute-value**



## Directive



### ▪ **Ghi chú:**

- Tên thẻ, thuộc tính, và giá trị phân biệt chữ hoa chữ thường
- Giá trị phải được đặt trong một cặp dấu nháy đơn hoặc kép
- Không có khoảng trắng giữa dấu bằng (=) và giá trị



## Directive



### page Directive Example

```
<%@ page language="java" %>
```

### include Directive Example

```
<%@ include file="copyright.html" %>
```

### taglib Directive Example

```
<%@ taglib prefix="test" uri="taglib.tld" %>
```



## Thuộc tính Page Directive



Attribute name	Description	Default Value/s
import	A comma-separated list of Java classes and packages that we want to use in the JSP page.	java.lang.*; javax.servlet.*; javax.servlet.jsp.*; javax.servlet.http.*;
session	A Boolean literal specifying whether the JSP page takes part in an HTTP session.	true
errorPage	Specifies a relative URL to another JSP page that is capable of handling errors on behalf of the current page.	null
isErrorPage	A Boolean literal specifying whether the current JSP page is capable of handling errors.	false
language	Any scripting language supported by the JSP engine. As of JSP 1.2, java is the only one allowed.	java
extends	Any valid Java class that implements javax.servlet.jsp.JspPage.	Implementation dependent



## Thuộc tính Page Directive



Attribute name	Description	Default Value/s
buffer	Specifies the size of the output buffer. If a buffer size is specified, it must be in kilobytes (kb). If buffering is not required, specify the string <i>none</i> .	Implementation dependent
autoFlush	A Boolean literal indicating whether the buffer should be flushed when it is full.	true
isThreadSafe	A Boolean literal indicating whether the JSP page is thread safe.	true
info	Any informative text about the JSP page.	Implementation dependent
contentType	Specifies the MIME type and character encoding for the output.	text/html; charset=ISO-8859-1
pageEncoding	Specifies the character encoding of the JSP page.	ISO-8859-1



## Thuộc tính import



- Khai báo các lớp java sử dụng trong trang jsp**

```
<%@ page import="java.util.*, java.io.*, java.text.*,
           com.mycom.*, com.mycom.util.MyClass" %>

<%@ page import="java.util.*" %>
<%@ page import="java.io.*" %>
<%@ page import="java.text.*" %>
<%@ page import="com.mycom.*, com.mycom.util.MyClass" %>
```



## Thuộc tính errorPage và isErrorPage



- Một trang JSP sử dụng các thuộc tính errorPage ủy thác việc xử lý ngoại lệ đối với một trang JSP có xử lý lỗi**
- Thuộc tính isErrorPage: trang hiện tại có thể hoạt động như một trình xử lý lỗi**

```
<%@ page errorPage="errorHandler.jsp" %>
<html>
<body>
  <%
    if (request.getParameter("name")==null)
    {
      throw new RuntimeException("Name not specified");
    }
  %>
  Hello, <%=request.getParameter("name")%>
</body>
</html>
```

```
<%@ page isErrorPage="true" %>
<html>
<body>
  <%=exception.getMessage()%><br>
  Please try again.
</body>
</html>
```



## Thuộc tính contentType và pageEncoding



- Thuộc tính **contentType** đặc tả MIME type và mã hóa kí tự (charater encoding) của output
- **MIME type** và **character coding** cách nhau bởi dấu “;”

```
<%@ page contentType="text/html; charset=ISO-8859-1" %>
```

- Thuộc tính **pageEncoding** đặc tả charater encoding của trang jsp

```
<%@ page pageEncoding="ISO-8859-1" %>
```



## Declaration



- Khai báo biến và các phương thức có thể được sử dụng trong các trang JSP
- Cấu trúc

```
<%! // code here %>
```

```
<%! int count = 0; %>
<%!
    String color[] = {"red", "green", "blue"};
    String getColor(int i)
    {
        return color[i%3];
    }
%>
```



## Scriptlet



- Là đoạn mã Java được nhúng vào trong các trang JSP
- Được thực thi mỗi khi trang web được truy cập (sự khác biệt với directive)
- Cú pháp: <% // code here %>

```
<%@ page language="java" %>
<%! int count = 0; %>

<%
    out.print("<html><body>");
    count++;
    out.print("Welcome! You are visitor number " + count);
    out.print("</body></html>");
%>
```

LTV Công nghệ Java – Module 3

24

## Expression



- Hoạt động như các placeholder cho các biểu thức ngôn ngữ Java
- Được đánh giá xem xét mỗi khi trang web được truy cập, và giá trị của nó sau đó được nhúng trong đầu ra HTML

```
<html><body>
<%@ page language="java" %>
<%! int count = 0; %>

Welcome! You are visitor number <%= ++count %>

</body></html>
```

LTV Công nghệ Java – Module 3

25

## Actions



- **Gồm các thẻ:**

- <jsp:include>
- <jsp:forward>
- <jsp:useBean>
- <jsp:setProperty>
- <jsp:getProperty>
- <jsp:plugin>



## Comment



- **Ghi chú cho người phát triển và không ảnh hưởng trang JSP**
- **Không thể lồng ghi chú JSP trong các ghi chú JSP khác**

```
<html><body>
    Welcome!
    <%-- JSP comment      --%>
    <%   //Java comment    %>
    <!-- HTML comment      -->
</body></html>
```



## Nội dung



1. JSP là gì?
2. Vòng đời của JSP
3. JSP Tags
- 4. Biến ẩn trong trang JSP (Implicit variable)**
5. Session, Cookies



## Biến ẩn trong trang JSP (Implicit variable)



Identifier Name	Class or Interface	Description
application	interface javax.servlet.ServletContext	Refers to the web application's environment
session	interface javax.servlet.http.HttpSession	Refers to the user's session
request	interface javax.servlet.http.HttpServletRequest	Refers to the current request to the page
response	interface javax.servlet.http.HttpServletResponse	Used for sending a response to the client
out	class javax.servlet.jsp.JspWriter	Refers to the output stream for the page
page	class java.lang.Object	Refers to the page's servlet instance
pageContext	class javax.servlet.jsp.PageContext	Refers to the page's environment
config	interface javax.servlet.ServletConfig	Refers to the servlet's configuration
exception	class java.lang.Throwable	Used for error handling



## application



- Biến thuộc javax.servlet.ServletContext
- Tham chiếu đến môi trường của trang JSP theo sau

```
<%  
    String path = application.getRealPath("/WEB-INF/counter.db");  
    application.log("Using: "+path);  
%>  
  
<%  
    String path = getServletContext().getRealPath("/WEB-INF/counter.db");  
    getServletContext().log("Using: "+path);  
%>
```



## session



- Biến thuộc loại javax.servlet.http.HttpSession
- Biến được khai báo khi thuộc tính session của directive page là true (mặc định)

```
<html>  
<body>  
    <%@ page session="false" %>  
  
    Session ID = <%=session.getId()%>  
</body>  
</html>
```



## request & response



- 2 biến tiềm ẩn thuộc HttpServletRequest and HttpServletResponse
- Sử dụng hoàn toàn giống như Servlet

```
<html><body>
<%
    String remoteAddr = request.getRemoteAddr();
    response.setContentType("text/html;charset=ISO-8859-1");
%>

Hi! Your IP address is <%=remoteAddr%>

</body></html>
```



## out



- Biến thuộc javax.servlet.jsp.JspWriter

```
<% out.print("Hello 1"); %>
<%= "Hello 2" %>
```

→

```
public void _jspService(...)

{
    //other code
    out.print("Hello 1");
    out.print("Hello 2");
}
```



## page



- Biến là một tham chiếu kiểu Object đến Servlet hiện tại
- Được khai báo như sau :

```
Object page = this;
```

- Ghi nhớ rằng page là kiểu Object, cần ép kiểu trước khi sử dụng

```
<%= page.getServletInfo() %>  

<%= ((Servlet)page).getServletInfo() %>
```



## pageContext



- Thuộc javax.servlet.jsp.PageContext
- PageContext là một lớp trừu tượng mà các nhà cung cấp JSP engine cung cấp
- Các nhiệm vụ :
  - Lưu trữ tham chiếu đến các biến tiềm ẩn
  - Cung cấp phương thức get/set và thiết lập các thuộc tính trong phạm vi khác nhau
  - Cung cấp phương pháp thuận tiện để chuyển các yêu cầu tới các tài nguyên khác trong ứng dụng web



## config



- Thuộc javax.servlet.ServletConfig
- Cấu hình các tham số trong mô tả triển khai

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
    <servlet>
        <servlet-name>InitTestServlet</servlet-name>
        <jsp-file>/initTest.jsp</jsp-file>
        <init-param>
            <param-name>region</param-name>
            <param-value>North America</param-value>
        </init-param>
    </servlet>
</web-app>
```

### Get information

```
<html><body>
    Servlet Name = <%=config.getServletName()%><br>
    Parameter region = <%=config.getInitParameter("region")%>
</body></html>
```



## Tầm vực của biến trong JSP



Scope Name	Existence and Accessibility
Application	Limited to a single web application
Session	Limited to a single user session
Request	Limited to a single request.
Page	Limited to a single page (translation unit) and a single request.



## Nội dung



1. JSP là gì?
2. Vòng đời của JSP
3. JSP Tags
4. Biển ẩn trong trang JSP (Implicit variable)
5. Session, Cookies



## Trạng thái



- **Thông tin trong quá trình tương tác giữa client – server**
- **Giao thức http không có trạng thái**
  - Không có cách để xác định các yêu cầu gửi cùng một client hay không
- **Tuy nhiên, một số trường hợp ứng dụng web cần lưu trữ trạng thái của client**
  - Ứng dụng shopping cart



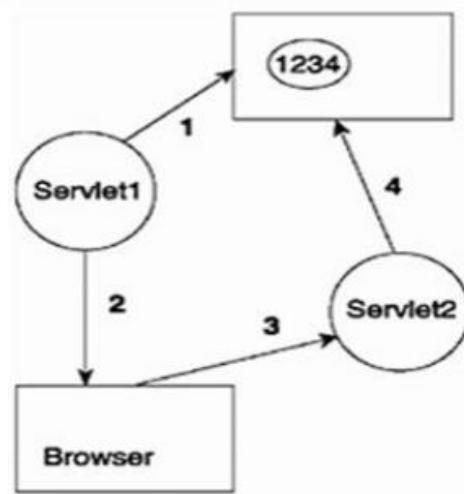
## Session



- Session giúp server lưu trữ trạng thái giao tiếp của client
- Session bắt đầu khi client gửi yêu cầu đầu tiên đến web application và kết thúc khi client đóng kết nối hoặc quá thời gian tương tác (timeout)



## Cơ chế hoạt động



## Cơ chế hoạt động



- Server tạo ra session ID cho lần yêu cầu đầu tiên và gửi cho client
- Client gửi yêu cầu lại sẽ chứa session ID và server dựa trên session ID để xác định trạng thái giao dịch của client đó



## Giao diện HttpSession



- Package javax.servlet.http
- Hiện thực bởi các servlet container và cung cấp cách theo dõi trạng thái giao dịch của client
- Servlet container tạo ra đối tượng session khi bắt đầu phiên giao dịch



## Dùng HttpSession



### ▪ 3 bước xử lý :

- Lấy Session kết hợp với các yêu cầu
- Thêm hoặc loại bỏ các cặp tên-giá trị của các thuộc tính trong Session
- Đóng hoặc không kích hoạt Session, nếu cần

Method	Description
HttpSession getSession(boolean create)	This method returns the current HttpSession associated with this request, or if there is no current session and the create parameter is true, then it returns a new session.
HttpSession getSession()	This method is equivalent to calling getSession(true).



## Dùng HttpSession



```

HttpSession session = req.getSession(true);
List listOfItems = (List) session.getAttribute("listofitems");
if(listOfItems == null) {
    listOfItems = new Vector();
    session.setAttribute("listofitems", listOfItems);
}

```



## Xử lý sự kiện Session



- **Hiện thực 4 giao diện:**
  - HttpSessionAttributeListener
  - HttpSessionBindingListener
  - HttpSessionListener
  - HttpSessionActivationListener



## HttpSessionBindingListener



- **Được gọi khi thêm mới hoặc hủy một đối tượng tới một session**
- **Servlet container gọi phương thức sự kiện khi session timeout**

Method	Description
void valueBound(HttpSessionBindingEvent event)	Notifies the object that it is being bound to a session
void valueUnbound(HttpSessionBindingEvent event)	Notifies the object that it is being unbound from a session



## HttpSessionListener



- Nhận được thông báo khi một Session là tạo ra hoặc bị tiêu huỷ

Method	Description
void sessionCreated(HttpSessionEvent se)	This method is called when a session is created.
void sessionDestroyed(HttpSessionEvent se)	This method is called when a session is destroyed.



## HttpSessionActivationListener



- Giao diện này được sử dụng bởi thuộc tính Session để nhận thông báo khi một Session đang được di chuyển trên các JVM trong một môi trường phân tán

Method	Description
void sessionDidActivate(HttpSessionEvent se)	This method is called just after the session is activated.
void sessionWillPassivate(HttpSessionEvent se)	This method is called when the session is about to be passivated.



## Phương thức khác



- Hidden field
- Cookies





Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh  
TRUNG TÂM TIN HỌC

[Go Screen Capture](#)

## LTV CÔNG NGHỆ JAVA

Module 3 – Bài 4: *Servlet nâng cao*

Ngành LT & CSDL

[www.t3h.vn](http://www.t3h.vn)



2014

5014

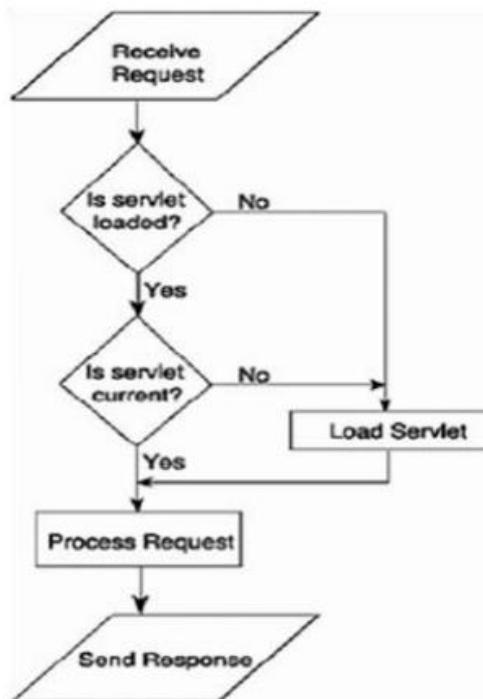


## Nội dung

1. Chu trình sống của Servlet
2. Events, Listeners
3. Filter



## Chu trình sống của Servlet



## Giao diện Servlet



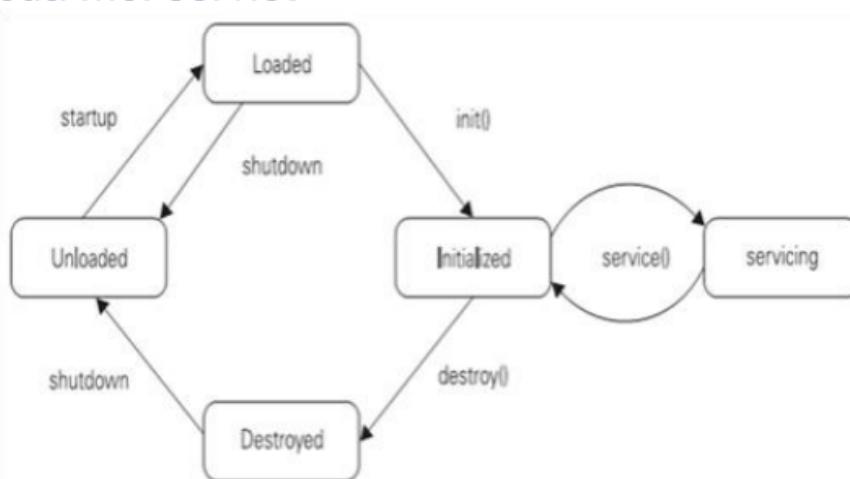
- **javax.servlet.Servlet**
- **Mỗi servlet phải hiện thực giao diện servlet một cách trực tiếp hoặc gián tiếp**
- **Chu kỳ sống của servlet được xác định bởi 3 phương thức**
  - Init
  - Service
  - Destroy



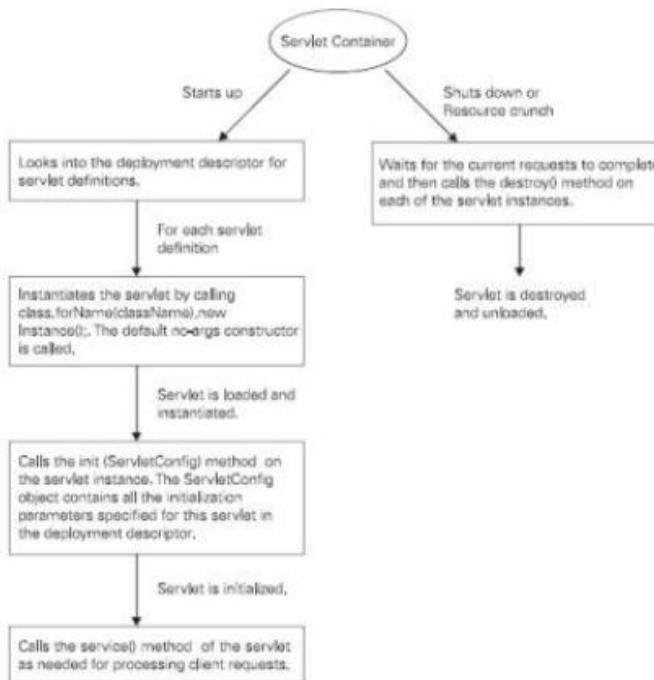
## Sơ đồ chuyển trạng thái servlet



- **Servlet container**
  - Đọc cấu hình (web.xml) và tạo ra các thực thể của mỗi servlet



## Container



## Phương thức init



- **public void init (ServletConfig config)  
throws ServletException**
- **Đối tượng ServletConfig chứa các  
cấu hình ghi trong tập tin web.xml  
cho các ứng dụng**
- **Khi một vấn đề tồn tại trong một  
servlet, ứng dụng ném một  
ServletException**



## Phương thức service



- **public void service (ServletRequest req,ServletResponse resp) throws ServletException**
- **Phương thức dịch vụ được gọi bởi servlet container sau khi phương thức init cho phép servlet đáp ứng yêu cầu**
- **Servlet thường chạy đa luồng trong servlet container để có thể xử lý đồng**



## Phương thức destroy



- **public void destroy()**
- **Servlet container gọi phương thức này trước khi loại bỏ một thể hiện servlet**
- **Phương thức này được gọi là chỉ sau khi tắt cả luồng trong phương thức service() của servlet đã thoát hoặc sau một thời gian chờ**



```

import javax.servlet.*;
import java.io.IOException;
.

public class PrimitiveServlet implements Servlet {
    public void init(ServletConfig config)
        throws ServletException {
        System.out.println("init");
    }
    public void service(ServletRequest request, ServletResponse response)
        throws ServletException, IOException {
        System.out.println("service");
    }
    public void destroy() {
        System.out.println("destroy");
    }
    public String getServletInfo() { return null; }
    public ServletConfig getServletConfig() { return null; }
}

```



## Servlet Config

- Thông tin trong tập tin web.xml có thể được lấy bằng cách sử dụng đối tượng **ServletConfig**
- Cấu hình các đối số cho một servlet nào

ServletConfig methods for retrieving initialization parameters

Method	Description
String getInitParameter(String name)	Returns the value of the parameter or null if no such parameter is available.
Enumeration getInitParameterNames()	Returns an Enumeration of Strings for all the parameter names.
ServletContext getServletContext()	Returns the ServletContext for this servlet.
String getServletName()	Returns the servlet name as specified in the configuration file.



## Servlet Config

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
    "http://java.sun.com/j2ee/dtds/web-app_2.2.dtd">

<web-app>
    <servlet>
        <servlet-name>ConfigDemo</servlet-name>
        <servlet-class>ConfigDemoServlet</servlet-class>
        <init-param>
            <param-name>adminEmail</param-name>
            <param-value>admin@hcmuns.edu.vn</param-value>
        </init-param>
        <init-param>
            <param-name>adminContactNumber</param-name>
            <param-value>04298371237</param-value>
        </init-param>
    </servlet>
</web-app>
```



## Servlet Config

```
public void init(ServletConfig config) throws ServletException {
    //get init parameter
    Enumeration parameters = config.getInitParameterNames();
    while (parameters.hasMoreElements()) {
        //get parameter name
        String parameter = (String) parameters.nextElement();
        System.out.println("Parameter name : " + parameter);
        //get init parameter value
        System.out.println("Parameter value : " +
                           config.getInitParameter(parameter));
    }
}
```



## Servlet Config



- **Tạo đối tượng ServletConfig toàn cục lưu trữ cấu hình ứng dụng Web**

```
ServletConfig servletConfig;
public void init(ServletConfig config)
    throws ServletException {
    servletConfig = config;
}
public ServletConfig getServletConfig() {
    return servletConfig;
}
```



## Servlet Context



- **Trong cùng một ứng dụng web, các Servlet chia sẻ cùng một môi trường**
- **Servlet container thể hiện môi trường trong Servlet thông qua giao diện Servlet Context**
- **Servlet API cũng định nghĩa giao diện cho phép giao tiếp giữa servlet và servlet container**



## Servlet context



### ▪ Khởi tạo Servlet Context

- Servlet Context được khởi tạo trong thời điểm ứng dụng được nạp
- Mỗi ứng dụng sẽ có duy nhất 1 Servlet Context
- Các tham số được khởi tạo qua Servlet context thường được dùng để cung cấp một số thông tin đặc biệt như :
  - Thông tin về người phát triển
  - Thông tin kết nối cơ sở dữ liệu



## Servlet context



### ▪ Đối tượng ServletContext được chứa trong ServerConfig

- Phương thức getServletContext()



```

public void service(ServletRequest request, ServletResponse response)
                    throws ServletException, IOException {
    //getting servlet context
    ServletContext servletContext = servletConfig.getServletContext();

    //get attribute
    Enumeration attributes = servletContext.getAttributeNames();
    while (attributes.hasMoreElements()) {
        String attribute = (String) attributes.nextElement();
        System.out.println("Attribute name : " + attribute);
        System.out.println("Attribute value : "
                           + servletContext.getAttribute(attribute));
    }
    //other attribute
    System.out.println("Major version : " +
                        servletContext.getMajorVersion());
    System.out.println("Minor version : " +
                        servletContext.getMinorVersion());
    System.out.println("Server info : " +
                        servletContext.getServerInfo());
}

```

## Servlet context trong web.xml

```

<web-app>
    <context-param>
        <param-name>user</param-name>
        <param-value>scott</param-value>
    </context-param>

    <context-param>
        <param-name>pass</param-name>
        <param-value>tiger</param-value>
    </context-param>

    <servlet>
        <servlet-name>Demo Context</servlet-name>
        <servlet-class>ServletServlet</servlet-class>
    </servlet>
</web-app>

```

## Requests và Responses



- **Giao diện ServletRequest tạo ra đối tượng đóng gói các thông tin**

- Parameters
- Attributes
- Luồng nhập (input stream)

- **Giao diện ServletResponse**

- Gọi hàm getWriter để tạo ra đối tượng PrintWriter



## ServletRequest



```

System.out.println("Server Port: " + request.getServerPort());
System.out.println("Server Name: " + request.getServerName());
System.out.println("Protocol: " + request.getProtocol());
System.out.println("Character Encoding: " + request.getCharacterEncoding());
System.out.println("Content Type: " + request.getContentType());
System.out.println("Content Length: " + request.getContentLength());
System.out.println("Remote Address: " + request.getRemoteAddr());
System.out.println("Remote Host: " + request.getRemoteHost());
System.out.println("Scheme: " + request.getScheme());

Enumeration parameters = request.getParameterNames();
while (parameters.hasMoreElements()) {
    String parameterName = (String) parameters.nextElement();
    System.out.println("Parameter Name: " + parameterName);
    System.out.println("Parameter Value: " + request.getParameter(parameterName));
}
Enumeration attributes = request.getAttributeNames();
while (attributes.hasMoreElements()) {
    String attribute = (String) attributes.nextElement();
    System.out.println("Attribute name: " + attribute);
    System.out.println("Attribute value: " + request.getAttribute(attribute));
}

```



## ServletResponse



```
PrintWriter out = response.getWriter();
out.println("<HTML>");
out.println("<HEAD>");
out.println("<TITLE>");
out.println("ServletResponse");
out.println("</TITLE>");
out.println("</HEAD>");
out.println("<BODY>");
out.println("<B>Demonstrating the ServletResponse object</B>");
out.println("<BR>");
out.println("<BR>Server Port: " + request.getServerPort());
out.println("</BODY>");
out.println("</HTML>");
```



## GenericServlet class



- Là abstract class
- Hiện thực 2 giao diện Servlet và ServletConfig
- Có thể tạo một servlet bằng cách thừa kế GenericServlet và thực hiện lại một số hàm được ghi đè



## GenericServlet class

```
public class SimpleServlet extends GenericServlet {
    public void service(ServletRequest request, ServletResponse
        response) throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        out.println("<HTML>");
        out.println("<HEAD>");
        out.println("<TITLE>");
        out.println("Extending GenericServlet");
        out.println("</TITLE>");
        out.println("</HEAD>");
        out.println("<BODY>");
        out.println("Extending GenericServlet makes code simpler.");
        out.println("</BODY>");
        out.println("</HTML>");
    }
}
```



## HttpServlet class

- Package javax.servlet.http
- Hiện thực các hàm trong giao thức http

- Phụ thuộc phương thức của request thì thực thi hàm tương ứng trong HttpServlet

- Thừa kế GenericServlet

HTTP Method	HttpServlet Method
GET	doGet()
HEAD	doHead()
POST	doPost()
PUT	doPut()
DELETE	doDelete()
OPTIONS	doOptions()
TRACE	doTrace()



## Redirect



- Một servlet có thể chuyển sang (redirect) sang một tài nguyên khác  
`response.sendRedirect("http://www.cnn.com");`
- Không thể redirect khi ServletResponse đã sẵn sàng trả về cho client



## RequestDispatcher



- Có thể dùng giao diện RequestDispatcher để triệu gọi tài nguyên khác
- Tạo ra đối tượng RequestDispatcher bằng cách gọi hàm `getRequestDispatcher()` của đối tượng:
  - ServletContext
  - ServletRequest



## Nội dung



1. Chu trình sống của Servlet
2. Events, Listeners
3. Filter



## Event



- **Sự khởi tạo hoặc hủy bỏ Servlet**  
**Context được gọi là events**
- **Được sử dụng để thực hiện các hành động**
  - Ghi log khi context được khởi tạo
  - Hỗ trợ khi context bị hủy
  - Đặc tả Servlet định nghĩa giao diện Listener là con đường nhận các thông báo khi có các sự kiện quan trọng có liên quan tới ứng dụng xảy đến



## Hiện thực Listener



- Đầu tiên viết 1 lớp thực thi giao diện listener có liên quan tới sự kiện và phải hiện thực giao diện:
  - ServletContextListener
  - ServletContextAttributeListener
  - HttpSessionAttributeListener
- Cấu hình lớp hiện thực vào mô tả triển khai (web.xml)



## ServletContextListener



Method	Description
void contextDestroyed(ServletContextEvent sce)	This method is called when the context is destroyed.
void contextInitialized(ServletContextEvent sce)	This method is called when the context is initialized.



```

package com.abcinc;
import javax.servlet.*;
import java.sql.*;
public class MyServletContextListener implements
ServletContextListener
{
    public void contextInitialized(ServletContextEvent sce)
    {
        try{
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            String url = (String)sce.getServletContext().
getInitParameter("dburl");
            Connection c = DriverManager.getConnection(url, "", "");
            sce.getServletContext().setAttribute("connection", c);
        }catch(Exception e) { }
    }
    public void contextDestroyed(ServletContextEvent sce)
    {
        try {
            Connection c = (Connection)
sce.getServletContext().getAttribute("connection");
            c.close();
        }catch(Exception e) { }
    }
}

```



## ServletContextAttributeListener

Method	Description
void attributeAdded(ServletContextAttributeEvent scae)	This method is called when a new attribute is added to the servlet context.
void attributeRemoved(ServletContextAttributeEvent scae)	This method is called when an existing attribute is removed from the servlet context.
void attributeReplaced(ServletContextAttributeEvent scae)	This method is called when an attribute of the servlet context is replaced.



## HttpSessionAttributeListener



Method	Description
void attributeAdded(HttpSessionBindingEvent se)	This method is called when an attribute is added to a session.
void attributeRemoved(HttpSessionBindingEvent se)	This method is called when an attribute is removed from a session.
void attributeReplaced(HttpSessionBindingEvent se)	This method is called when an attribute is replaced in a session.



## Thẻ Listener



- **Chỉ định các lớp lắng nghe cho các sự kiện ứng dụng**
- **Chứa một và chỉ một class xác định tên đầy đủ của lớp thực hiện các listener interface**

```
<listener>
    <listener-class>
        com.abcinc.MyServletContextListener
    </listener-class>
</listener>
```



## Nội dung

1. Chu trình sống của Servlet
2. Events, Listeners
3. Filter

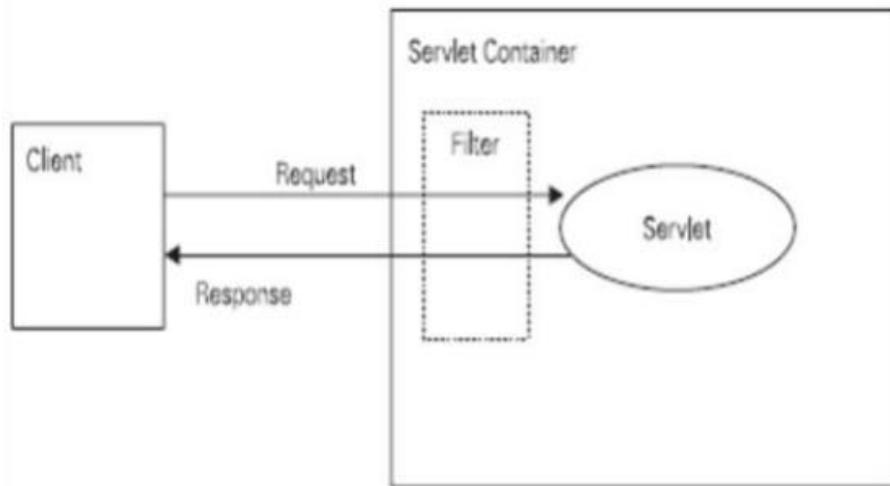


## Filter

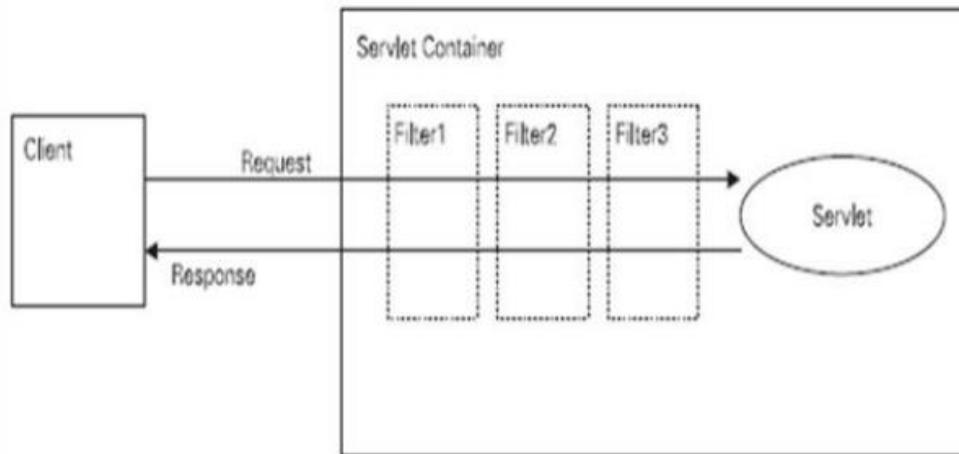
- Trong kỹ thuật, filter là một đối tượng chặn thông điệp giữa dữ liệu nguồn và đích sau đó lọc dữ liệu được chuyển qua chúng
  - Nó hoạt động như một người bảo vệ ngăn chặn những thông tin không mong muốn được truyền từ một điểm đến một điểm khác
- Trong ứng dụng web, filter là một thành phần web ở web server lọc các request và response giữa client và tài nguyên



## Single filter



## Multiple filters



## Các ứng dụng của Filter



- **Bộ lọc chứng thực**
- **Bộ lọc đăng nhập**
- **Bộ lọc chuyển đổi hình ảnh**
- **Bộ lọc nén dữ liệu**
- **Bộ lọc mã hóa**
- **Bộ lọc Tokenizing**
- **Bộ lọc kích hoạt các sự kiện truy cập tài nguyên**
- ...



## Hiện thực lớp Filter



- **Tạo lớp hiện thực giao diện Filter (package javax.servlet) và hiện thực 3 hàm**
  - init()
  - doFilter()
  - destroy()
- **Cấu hình lớp hiện thực trong cấu hình triển khai (web.xml)**
  - **Đặc tả thẻ filter và filter-mapping**



```

import java.io.*;
import javax.servlet.*;
public class HelloWorldFilter implements Filter
{
    private FilterConfig filterConfig;
    public void init(FilterConfig filterConfig)
    {
        this.filterConfig = filterConfig;
    }
    public void doFilter(
        ServletRequest request,
        ServletResponse response,
        FilterChain filterChain
    ) throws ServletException, IOException
    {
        PrintWriter pw = response.getWriter();
        pw.println("<html>");
        pw.println("<head>");
        pw.println("</head>");
        pw.println("<body>");
        pw.println("<h3>Hello Filter World!</h3>");
        pw.println("</body>");
        pw.println("</html>");
    }
    public void destroy()
    {
    }
}

```



```

<web-app>
    ...
    <!-- specify the Filter name and the Filter class -->
    <filter>
        <filter-name>HelloWorldFilter</filter-name>
        <filter-class>HelloWorldFilter</filter-class>
    </filter>
    <!-- associate the Filter with a URL pattern -->
    <filter-mapping>
        <filter-name>HelloWorldFilter</filter-name>
        <url-pattern>/filter/*</url-pattern>
    </filter-mapping>
    ...
</web-app>

```







Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh  
TRUNG TÂM TIN HỌC

[Go Screen Capture](#)

## LTV CÔNG NGHỆ JAVA

Module 3 – Bài 5: *JSP nâng cao*

Ngành LT & CSDL

[www.t3h.vn](http://www.t3h.vn)



2014

2014



## Nội dung



1. Khái quát về Custom Tag
2. Hiện thực Custom Tag
3. Sử dụng lại thành phần web
4. Java Beans



## Giới thiệu



- **Thẻ JSP có phần hạn chế và không cung cấp hỗ trợ cho các trình bày logic cần thiết cho định dạng dữ liệu động**
- **Công nghệ JSP cung cấp một tính năng cho phép nhà phát triển tạo ra các thẻ mới và định nghĩa hành vi của chúng**
- **Các thẻ người dùng định nghĩa được gọi là Custom Tags**



## Một số khái niệm



### ▪ Tag handler

- Là một đối tượng được quản lý bởi Container
- Tag handler đánh giá hành động tùy chỉnh trong khi thực hiện một trang JSP
- Là một lớp Java thực hiện giao diện Tag, IterationTag, hoặc BodyTag
- Trong gói javax.servlet.jsp.tagext



## Một số khái niệm



### ▪ Tag library

- Một tập của các hành động gói gọn một số chức năng được sử dụng bên trong một trang JSP
- Chúng ta sẽ thiết kế và phát triển một tập hợp các thẻ làm việc cùng nhau và giúp giải quyết các yêu cầu thường gặp
- Một bộ các thẻ tùy chỉnh như vậy được gọi là một thư viện thẻ



## Một số khái niệm



### ▪ Tag library descriptor

- Khi chúng ta sử dụng các thẻ tùy chỉnh trong một trang JSP, JSP engine cần phải biết các lớp xử lý cho các thẻ này
- Trong đó thẻ thư viện chúng nằm ở đâu, và làm thế nào chúng được sử dụng
- Siêu dữ liệu này được lưu trữ trong một tập tin gọi là thư viện thẻ mô tả (TLD)



## Nội dung



1. Khái quát về Custom Tag
2. Hiện thực Custom Tag
3. Sử dụng lại thành phần web
4. Java Beans



## Hiện thực Custom Tag



- Sun Microsystems xây dựng tập các thẻ theo chuẩn JSTL
- Qui trình xây dựng và sử dụng Custom Tag
  - Định nghĩa Tag Libraries Descriptor (TLD)
  - Xây dựng lớp Tag Handler
  - Cấu hình TLD trong web.xml
  - Sử dụng các Tag Libraries trong trang JSP



## Tag Library Descriptor



```

<?xml version="1.0" encoding="ISO-8859-1">
<!DOCTYPE taglib PUBLIC
  "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
  "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd" >
<taglib>
  <lib-version>1.0</lib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>test</short-name>
  <uri>http://www.manning.com/sampleLib</uri>
  <tag>
    <name>greet</name>
    <tag-class>sampleLib.GreetTag</tag-class>
    <body-content>empty</body-content>
    <description>Prints Hello and the user name</description>
    <attribute>
      <name>user</name>
      <required>false</required>
      <rtpexprvalue>true</rtpexprvalue>
    </attribute>
  </tag>
</taglib>

```



## Tag Library Descriptor



- DTD thẻ taglib

```
<!ELEMENT taglib (tlib-version, jsp-version, short-name,
    uri?, display-name?, small-icon?, large-icon?,
    description?, validator?, listener*, tag+)>
```

- DTD thẻ tag

```
<!ELEMENT tag (name, tag-class, tei-class?, body-content?,
    display-name?, small-icon?, large-icon?,
    description?, variable*, attribute*, example?)>
```



## Thẻ body-content



- Phần tử này không có phần tử con
- Có 3 giá trị cho phần tử
  - Empty: nội dung của Tag phải là rỗng
  - JSP: nội dung của Tag có thể chấp nhận bất kỳ đặc tả nào của JSP (mặc định)
  - Tagdependent: nội dung của Tag không được biên dịch bởi JSP engine và là một tag độc lập



```
<test:greet />
<test:greet user="john" />
<test:greet></test:greet>
<test:greet user="john"></test:greet>
```

**JSP**

**Empty**

```
<test:dbQuery>
    SELECT count(*) FROM USERS
</test:dbQuery>
```

**Tagdependent**

```
<test:if condition="true" />
<test:if condition="true"> </test:if>
<test:if condition="true">&nbsp; </test:if>
<test:if condition="true">
    <test:greet user="john" />
    <% int x = 2+3; %>
    2+3 = <%= x %>
</test:if>
```



## Thẻ attribute

```
<!ELEMENT attribute (name, required? , rtxprvalue?,
                    type?, description?) >
```

Element	Description
name	The name of the attribute.
required	A value that specifies whether the attribute is required or optional. The default is false, which means optional. If this is set to true, then the JSP page must pass a value for this attribute. Possible values are true, false, yes, and no.
rtxprvalue	A value that specifies whether or not the attribute can accept request-time expression values. The default is false, which means it cannot accept request-time expression values. Possible values are true, false, yes, and no.
type	The data type of the attribute. This may be used only when <rtxprvalue> is set to true. It specifies the return type of the expression, using a request-time attribute expression: <%= %>. The default value is java.lang.String.
description	Some text describing the attribute for documentation purposes.



## Tag Handler



- Tag Handler là những lớp java hiện thực một trong 3 giao diện sau

Interface Name	Description
Tag	The Tag interface is the base interface for all tag handlers and is used for writing simple tags. It declares six tag life-cycle methods, including the two most important ones: doStartTag() and doEndTag(). We implement this interface if we want to write a simple tag that does not require iterations or processing of its body content.
IterationTag	IterationTag extends the Tag interface and adds one more method for supporting iterations: doAfterBody().
BodyTag	BodyTag extends IterationTag and adds two methods for supporting the buffering of body contents: doInitBody() and setBodyContent().



## Hiện thực giao diện Tag

- Giao diện Tag là giao diện cơ sở cho tất cả xử lý thẻ tùy chỉnh
- Các phương thức :

Method Name	Description
int doStartTag()	Called when the opening tag is encountered.
int doEndTag()	Called when the closing tag is encountered.
Tag getParent()	Returns the handler class object of the closest enclosing tag of this tag.
void release()	Called on a tag handler to release resources.
void setPageContext(PageContext)	Sets the current page context.
void setParent(Tag)	Sets the parent (closest enclosing tag handler) of this tag handler.



## Hiện thực giao diện Tag

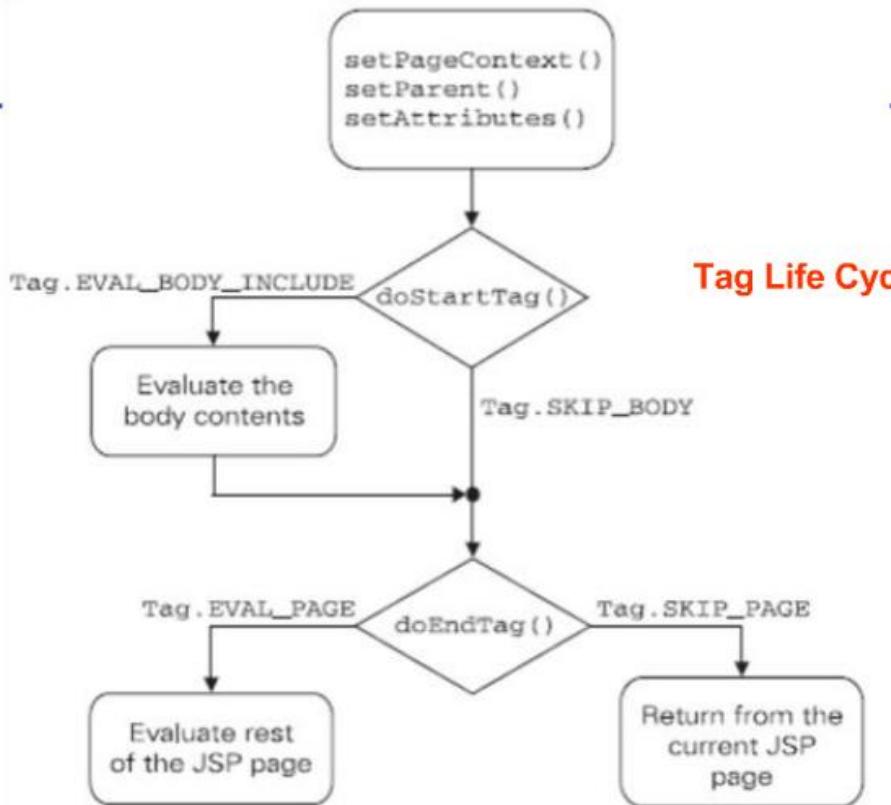


### ▪ Các tham số

Constant	Description
EVAL_BODY_INCLUDE	Return value for doStartTag() Instructs the JSP engine to evaluate the tag body and include it in the output.
SKIP_BODY	Return value for doStartTag() Instructs the JSP engine not to evaluate the tag body and not to include it in the output.
EVAL_PAGE	Return value for doEndTag() Instructs the JSP engine to evaluate the rest of the page and include it in the output.
SKIP_PAGE	Return value for doEndTag() Instructs the JSP engine not to evaluate the rest of the page and not to include it in the output.



### Tag Life Cycle



## Hiện thực giao diện Tag



- **Ngữ cảnh sử dụng :**

- Một thẻ trống mà chỉ cần in văn bản HTML
- Một thẻ trống chấp nhận một thuộc tính
- Thẻ không trống (một thẻ với body) mà bỏ qua hoặc bao gồm các phần body của mình



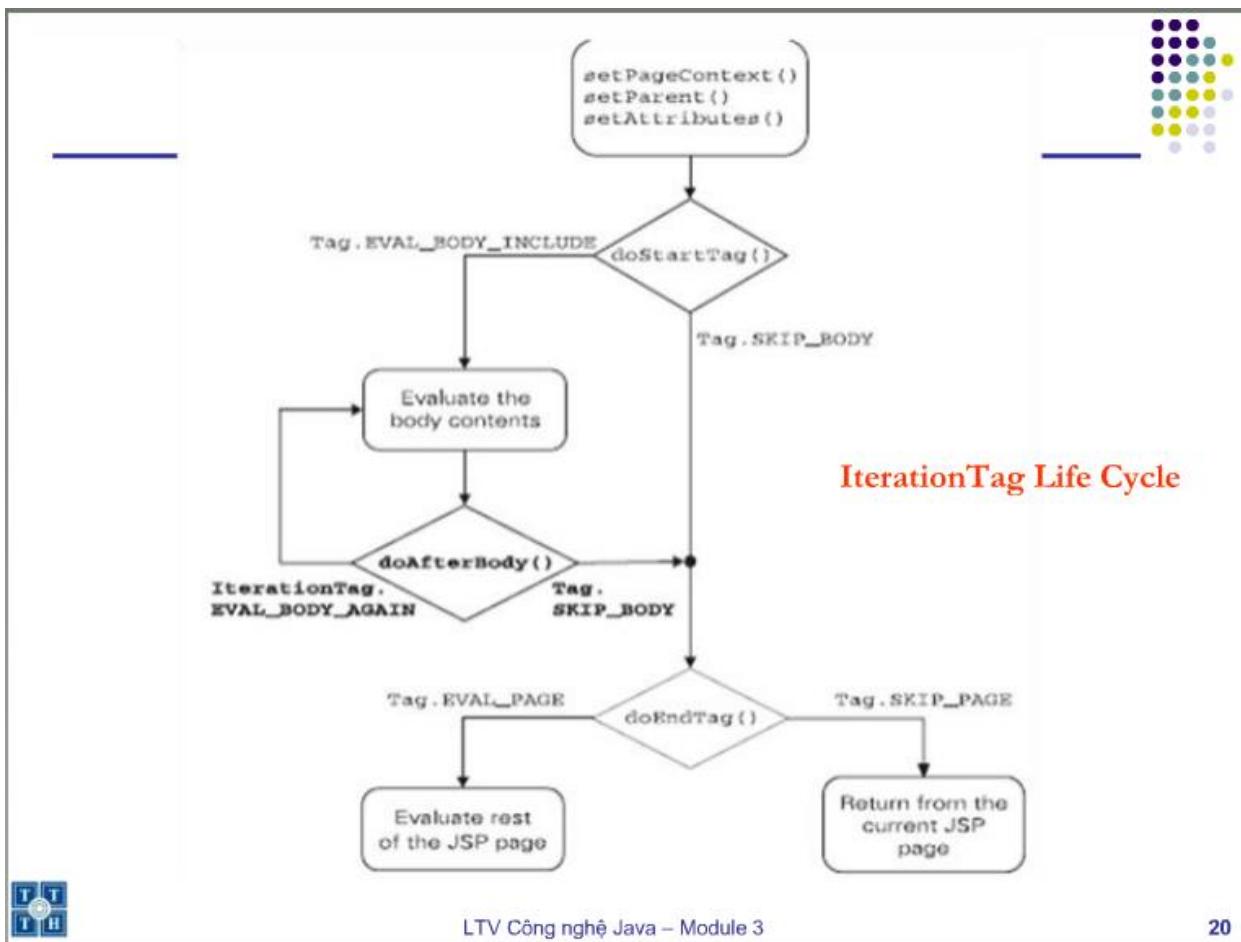
## Hiện thực giao diện IterationTag



- **IterationTag mở rộng Tag và cho phép chúng bao nội dung body nhiều lần**
- **IterationTag khai báo một phương thức và một hằng số**

Method Name	Description
int doAfterBody()	This method is called after each evaluation of the tag body. It can return either of two values: IterationTag.EVAL_BODY_AGAIN or Tag.SKIP_BODY. The return value determines whether or not the body needs to be reevaluated.
Constant	Description
EVAL_BODY_AGAIN	Return value for doAfterBody(). This constant instructs the JSP engine to evaluate the tag body and include it in the output.





## Hiện thực giao diện BodyTag

- **Giao diện BodyTag mở rộng IterationTag**
- **Nó thêm một chức năng mới cho phép thẻ xử lý đánh giá nội dung body của nó trong một vùng đệm tạm thời**
- **Tính năng này cho phép thẻ xử lý nội dung được tạo ra theo ý muốn**
  - Ví dụ, sau khi đánh giá, xử lý thẻ có thể xem các phần body, loại bỏ nó hoàn toàn, sửa đổi nó, hoặc thêm nhiều dữ liệu hơn trước khi gửi nó

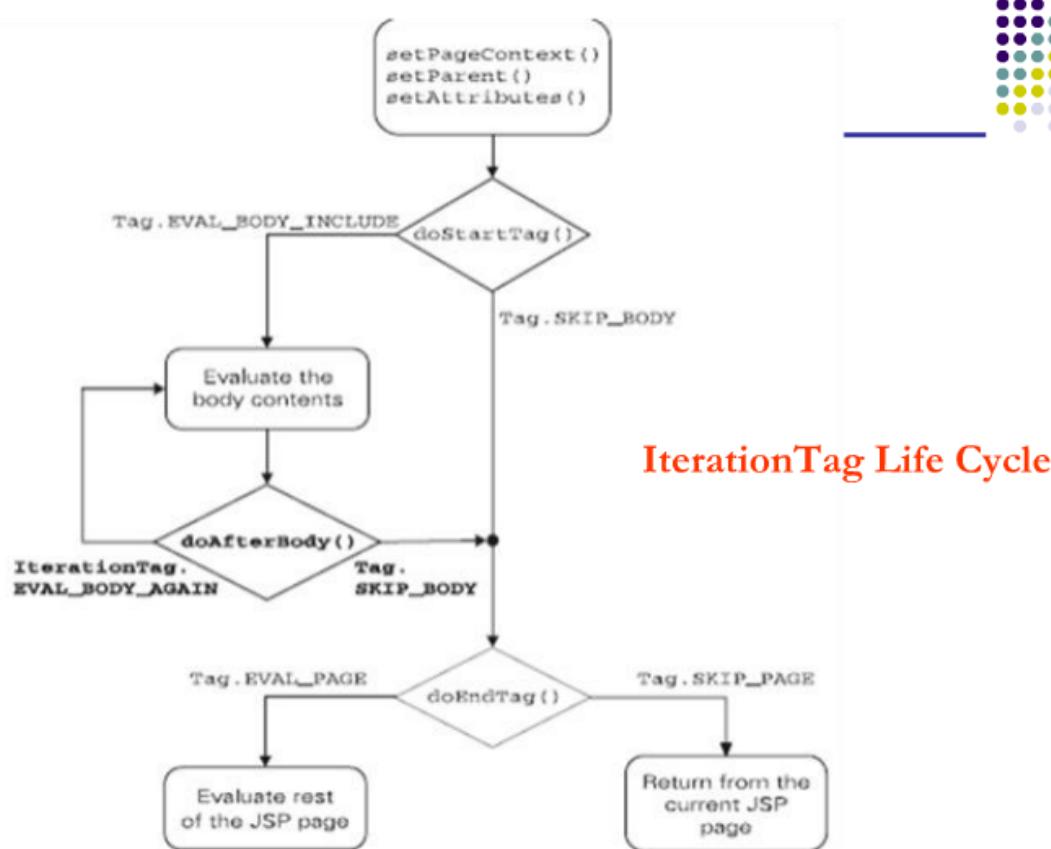


## Hiện thực giao diện BodyTag



### ▪ Phương thức và tham số

Method Name	Description
void setBodyContent(BodyContent)	Called by the JSP container to pass a reference to a BodyContent object.
void doInitBody()	Called by the JSP container after calling setBodyContent(), to allow the tag handler class to perform initialization steps on BodyContent.
Constant	Description
EVAL_BODY_BUFFERED	A constant defined as a return value for doStartTag() and doAfterBody() for tags that want to buffer the evaluation of their content.



## Cấu hình TLD trong web.xml



- Sử dụng phần tử trong web.xml đặc tả URIs của TLD

```
<!ELEMENT taglib (taglib-uri, taglib-location)>
```



```
<taglib>
  <taglib-uri>
    http://www.manning.com/studyKit
  </taglib-uri>
  <taglib-location>
    /myLibs/studyKit.tld
  </taglib-location>
</taglib>
```



## Sử dụng Tag Libraries trong JSP



```
<%@ taglib uri="sampletaglib.tld" prefix="test" %>

<test:if condition="true">
  Anything that is to be printed when the condition is true goes here.
  Name is: <%=request.getParameter("name")%>
</test:if>
```



## Nội dung



1. Khái quát về Custom Tag
2. Hiện thực Custom Tag
3. Sử dụng lại thành phần web
4. Java Beans



## JSP bao gộp trang JSP khác



- **Sử dụng lại thành phần web cho phép một trang JSP có thể bao gồm (include) một trang JSP khác**
- **Có 2 cách hiện thực include**
  - Tĩnh (Static inclusion)
  - Động (Dynamic inclusion)



## Static inclusion



- Dùng JSP Directive

```
<%@ include file="relativeURL" %>
<jsp:directive.include file="relativeURL" />
```

- Thuộc tính file tham chiếu đến file (có thể là text-based file, htm, JSP hoặc XML)
- Đường dẫn file là đường dẫn tương đối



## Static inclusion



File a.jsp

```
<html><body>
<b>Welcome</b>
<%@ include
  file="b.jsp"
%>
<b>Good Bye</b>
</body></html>
```

a.jsp's Translation Time

File b.jsp

```
<pre>
Once upon a time
...
</pre>
```

```
//in _jspService()
out.write(
  <html><body>
  <b>Welcome</b>
  </body></html>
);
out.write(
  <pre>
  Once upon a time
  ...
  </pre>
);
out.write(
  <b>Good Bye</b>
  </body></html>
);
```

Generated servlet for a.jsp

Request Time

```
<html><body>
<b>Welcome</b>
<pre>
Once upon a time
...
</pre>
<b>Good Bye</b>
</body></html>
```

Output HTML



## Dynamic inclusion



- Dùng JSP actions

```
<jsp:include page="relativeURL" flush="true" />
```

```
<jsp:forward page="relativeURL" />
```

- Đường dẫn tương đối có thể là biểu thức

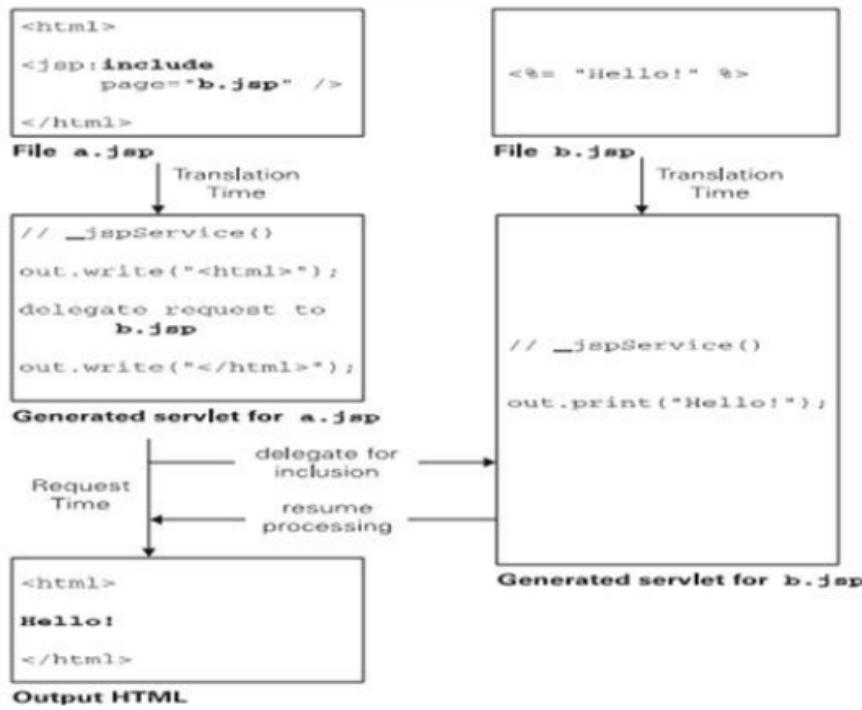
```
<% String pageURL = "other.jsp" %>
<jsp:include page="<%=pageURL%>" />
```

- Có thể truyền đối số cho dynamic inclusion

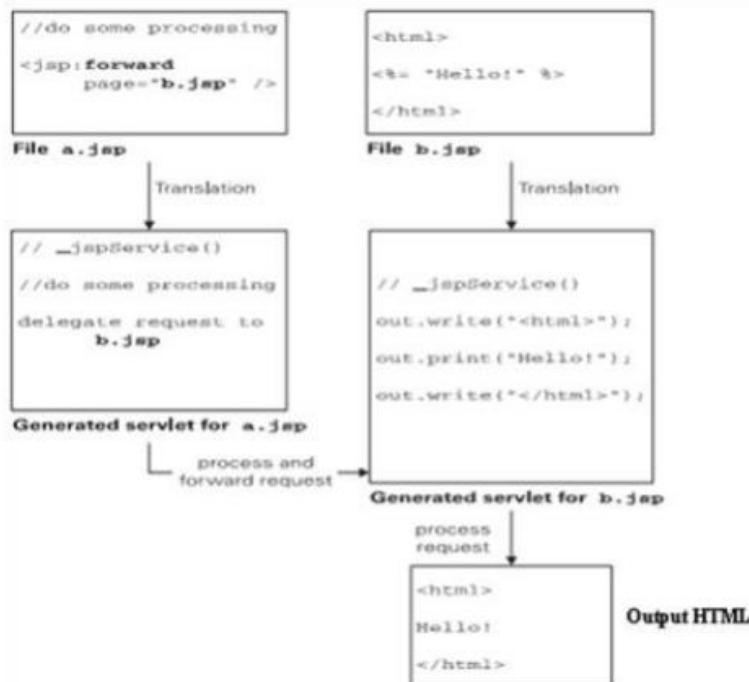
```
<jsp:include page="somePage.jsp">
  <jsp:param name="name1" value="value1" />
  <jsp:param name="name2" value="value2" />
</jsp:include>
```



## JSP include



## JSP forward



## Nội dung

1. Khái quát về Custom Tag
2. Hiện thực Custom Tag
3. Sử dụng lại thành phần web
4. Java Beans



## JavaBeans



- **Mô hình các thành phần phần mềm được thiết kế sao cho dễ tái sử dụng và thay thế nhau**
- **Các thành phần tái sử dụng được viết trong java và đóng gói các hành vi, trạng thái**
- **JavaBeans định nghĩa ra những giao diện và những lớp trong package java.bean để tạo ra bean và điều khiển các thuộc tính của nó**



## Cấu trúc của JavaBeans



- **Là lớp java phải có hàm khởi tạo không có đối số**
- **Tất cả các thuộc tính phải đặc tả private**
  - Phương thức getter để truy cập giá trị thuộc tính nào đó trong bean
  - Phương thức setter để thiết lập giá trị cho một thuộc tính nào đó trong bean



## Ví dụ - AddressBean



```

package data.beans;
public class AddressBean implements java.io.Serializable{
    //properties
    private String street;
    private String city;
    private String state;
    private String zip;
    //setters
    public void setStreet(String street){ this.street = street; }
    public void setCity(String city) { this.city = city; }
    public void setState(String state) { this.state = state; }
    public void setZip(String zip) { this.zip = zip; }
    //getters
    public String getStreet(){ return this.street; }
    public String getCity() { return this.city; }
    public String getState() { return this.state; }
    public String getZip() { return this.zip; }
}

```



## Truy cập Beans trong JSP



- **Thẻ JavaBean được kết hợp với phần tử JSP**
- **Thẻ JavaBean được chuyển đổi vào một lớp Servlet trên server**
- **Dùng phần tử <jsp:useBean> để tạo một tham chiếu đến một đối tượng**
  - <jsp:setProperty> để thiết lập giá trị thuộc tính
  - <jsp:getProperty> để lấy giá trị thuộc tính



```

<%@ page import='data.beans.AddressBean' %>
<jsp:useBean id='address' class="scwcdkit.AddressBean"
              scope='session'/>

<html><body>
  <table>
    <tr>
      <td>Street</td>
      <td><jsp:getProperty name='address' property='street' /></td>
    </tr>
    <tr>
      <td>City</td>
      <td><jsp:getProperty name='address' property='city' /></td>
    </tr>
    <tr>
      <td>State</td>
      <td><jsp:getProperty name='address' property='state' /></td>
    </tr>
    <tr>
      <td>Zip</td>
      <td><jsp:getProperty name='address' property='zip' /></td>
    </tr>
  </table>
</body></html>

```



## Tầm vực của JavaBeans

- **Khả năng truy cập JavaBean là tầm vực của JavaBean**
- **JavaBean chứa 4 tầm vực**
  - Page: truy cập trong trang hiện tại (mặc định)
  - Request: truy cập trong lời triệu gọi hiện tại
  - Session: truy cập trong một phiên làm việc
  - Application: truy cập trong một ứng dụng







Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh  
TRUNG TÂM TIN HỌC

[Go Screen Capture](#)

## LTV CÔNG NGHỆ JAVA

Module 3 – Bài 6: **EL & JSTL**

Ngành LT & CSDL

[www.t3h.vn](http://www.t3h.vn)



2014

5014



## Nội dung



1. EL - Expression Language
2. JSTL - JSP Standard Tag Library



## JSP Expression Language



- **EL cung cấp cấu trúc cho phép truy xuất dữ liệu của JavaBean, maps, arrays và danh sách được lưu trữ trong các thuộc tính của ứng dụng web**
- **JSP EL cho phép bạn tạo ra các biểu thức cả số học và logic.**
- **Biểu thức JSP EL có thể sử dụng số nguyên, số thực, chuỗi, boolean và giá trị null**



## Ưu điểm



- Ngắn gọn và súc tích hơn JSP chuẩn
- Cho phép truy cập các thuộc tính lồng cấp
- Cho phép truy cập collections như map, arrays, list
- Xử lý giá trị null tốt hơn JSP chuẩn
- Cung cấp nhiều chức năng



## Nhược điểm



- Không tạo ra JavaBean nếu chưa tồn tại
- Không cung cấp cách để thiết lập giá trị cho thuộc tính



## Cú pháp EL



- **Định dạng**

**\$ { Expression }**

- **Expression có thể là:**

- Literals
- Operators
- Variables (đối tượng tham chiếu)
- Gọi hàm thông qua các thuộc tính



## EL Literals



Literals	Literal Values
Boolean	true or false
Integer	Similar to Java e.g. 243, -9642
Floating Point	Similar to Java e.g. 54.67, 1.83
String	Any string delimited by single or double quote e.g. "hello" , 'hello'
Null	null

**\${ false } <%-- evaluate to false --%>**

**\${ 8\*3 } <%-- evaluate to 24 --%>**





## EL Operators

Type	Operator
Arithmetic	+ - * / (div) % (mod)
Grouping	( )
Logical	&& (and)    (or) ! (not)
Relational	== (eq) != (ne) < (lt) > (gt) <= (le) >= (ge)
Empty	prefix operation to determine value is null or empty, returns boolean value
Conditional	? :

`${ (6*5) + 5 } <%-- evaluate to 35 --%>`

`${ (x >= min) && (x <= max) }`

`${ empty name }`



## EL Identifiers

- Thể hiện tên của các biến đối tượng
- Có 11 biến ẩn

Category	Implicit Object	Description
JSP	pageContext	used to access JSP implicit objects
Scopes	pageScope	A Map associating names & values of page scoped attributes
	requestScope	A Map associating names & values of request scoped attributes
	sessionScope	A Map associating names & values of session scoped attributes
	applicationScope	A Map associating names & values of page scoped attributes



## EL Identifiers



Category	Operator	Description
Request Parameters	Param	Maps a request parameter name to a single String parameter value
	paramValues	Maps a request parameter name to an array of values
Request Headers	header	Maps a request header name to a single header value
	headerValues	Maps a request header name to an array of values
Cookies	cookie	A <i>Map</i> storing the cookies accompanying the request by name
Initialization parameters	initParam	A <i>Map</i> storing the context initialization parameters of the web application by name



## EL Identifiers



### ▪ Examples

`${ pageContext.response }`

- Trả về đối tượng response của trang JSP

`${ param.name }`

- Tương ứng với `request.getParameter("name")`

`${ cookie.name.value }`

- Tương đương
 

```
if (cookie.getName().equals("name")) {
    String val = cookie.getValue();
}
```



## Toán tử “.”



- Được dùng để truy cập các thuộc tính của đối tượng
- Ví dụ

`$ { person.name }`

`${ user.address.city }`



## Toán tự [ ]



- Với arrays và collection hiện thực giao diện List
  - Dùng index để truy cập các phần tử
  - `$ { personList[2] }` → truy xuất phần tử thứ 3
- Với collection map
  - Đặc tả key bên trong []
  - `$ { myMap["id"] }`



## Nội dung



1. EL - Expression Language
2. JSTL - JSP Standard Tag Library



## JSTL



- **Một tập hợp các thẻ JSP hữu ích gói gọn chức năng cốt lõi chung cho nhiều ứng dụng JSP.**
- **JSTL hỗ trợ cho các thao tác phổ biến, cấu trúc như :lặp và điều kiện, thẻ cho các thao tác các tài liệu XML, thẻ quốc tế, và các thẻ SQL. Nó cũng cung cấp một khuôn khổ cho việc tích hợp tùy chỉnh thẻ hiện có với các thẻ JSTL**



## Phân loại JSTL



- Core Tags
- Formatting tags
- SQL tags
- XML tags
- JSTL Functions



## Core Tags



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

Tag	Description
<c:out>	Like <%= ... >, but for expressions.
<c:set>	Sets the result of an expression evaluation in a 'scope'
<c:remove>	Removes a scoped variable (from a particular scope, if specified).
<c:catch>	Catches any Throwable that occurs in its body and optionally exposes it.
<c:if>	Simple conditional tag which evaluates its body if the supplied condition is true.
<c:choose>	Simple conditional tag that establishes a context for mutually exclusive conditional operations, marked by <when> and <otherwise>
<c:when>	Subtag of <choose> that includes its body if its condition evaluates to 'true'.
<c:otherwise>	Subtag of <choose> that follows <when> tags and runs only if all of the prior conditions evaluated to 'false'.
<c:import>	Retrieves an absolute or relative URL and exposes its contents to either the page, a String in 'Var', or a Reader in 'VarReader'.
<c:forEach>	The basic iteration tag, accepting many different collection types and supporting subsetting and other functionality.
<c:forTokens>	Iterates over tokens, separated by the supplied delimiters.
<c:param>	Adds a parameter to a containing 'import' tag's URL.
<c:redirect>	Redirects to a new URL.
<c:url>	Creates a URL with optional query parameters



## Formatting tags



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

Tag	Description
<fmt:formatNumber>	To render numerical value with specific precision or format.
<fmt:parseNumber>	Parses the string representation of a number, currency, or percentage.
<fmt:formatDate>	Formats a date and/or time using the supplied styles and pattern
<fmt:parseDate>	Parses the string representation of a date and/or time
<fmt:bundle>	Loads a resource bundle to be used by its tag body.
<fmt:setLocale>	Stores the given locale in the locale configuration variable.
<fmt:setBundle>	Loads a resource bundle and stores it in the named scoped variable or the bundle configuration variable.
<fmt:timeZone>	Specifies the time zone for any time formatting or parsing actions nested in its body.
<fmt:setTimeZone>	Stores the given time zone in the time zone configuration variable
<fmt:message>	To display an internationalized message.
<fmt:requestEncoding>	Sets the request character encoding



## SQL tags



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

Tag	Description
<sql:setDataSource>	Creates a simple DataSource suitable only for prototyping
<sql:query>	Executes the SQL query defined in its body or through the sql attribute.
<sql:update>	Executes the SQL update defined in its body or through the sql attribute.
<sql:param>	Sets a parameter in an SQL statement to the specified value.
<sql:dateParam>	Sets a parameter in an SQL statement to the specified java.util.Date value.
<sql:transaction>	Provides nested database action elements with a shared Connection, set up to execute all statements as one transaction.



## XML tags



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

**Trước khi sử dụng cần copy thư viện xử lý xml vào thư mục lib:**

XercesImpl.jar: Download it from <http://www.apache.org/dist/xerces/ji/>  
 xalan.jar: Download it from <http://xml.apache.org/xalan-j/index.html>

Tag	Description
<x:out>	Like <%= ... %>, but for XPath expressions.
<x:parse>	Use to parse XML data specified either via an attribute or in the tag body.
<x:set>	Sets a variable to the value of an XPath expression.
<x:if>	Evaluates a test XPath expression and if it is true, it processes its body. If the test condition is false, the body is ignored.
<x:forEach>	To loop over nodes in an XML document.
<x:choose>	Simple conditional tag that establishes a context for mutually exclusive conditional operations, marked by <when> and <otherwise>
<x:when>	Subtag of <choose> that includes its body if its expression evaluates to 'true'
<x:otherwise>	Subtag of <choose> that follows <when> tags and runs only if all of the prior conditions evaluated to 'false'
<x:transform>	Applies an XSL transformation on a XML document
<x:param>	Use along with the transform tag to set a parameter in the XSLT stylesheet



## JSTL Functions



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

Function	Description
fn:contains()	Tests if an input string contains the specified substring.
fn:containsIgnoreCase()	Tests if an input string contains the specified substring in a case insensitive way.
fn:endsWith()	Tests if an input string ends with the specified suffix.
fn:escapeXml()	Escapes characters that could be interpreted as XML markup.
fn:indexOf()	Returns the index within a string of the first occurrence of a specified substring.
fn:join()	Joins all elements of an array into a string.
fn:length()	Returns the number of items in a collection, or the number of characters in a string.
fn:replace()	Returns a string resulting from replacing in an input string all occurrences with a given string.
fn:split()	Splits a string into an array of substrings.
fn:startsWith()	Tests if an input string starts with the specified prefix.
fn:substring()	Returns a subset of a string.
fn:substringAfter()	Returns a subset of a string following a specific substring.
fn:substringBefore()	Returns a subset of a string before a specific substring.
fn:toLowerCase()	Converts all of the characters of a string to lower case.
fn:toUpperCase()	Converts all of the characters of a string to upper case.







Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh  
TRUNG TÂM TIN HỌC

[Go Screen Capture](#)

## LTV CÔNG NGHỆ JAVA

Module 3 – Bài 7: *Mô hình MVC với JSP/Servlet*

Ngành LT & CSDL

[www.t3h.vn](http://www.t3h.vn)



2014

5014



## Nội dung



1. Giới thiệu
2. Các thành phần trong mô hình MVC
3. Mô hình hoạt động giữa các thành phần trong MVC
4. Ưu và nhược điểm của mô hình MVC
5. Phát triển Web JEE với MVC



## Mẫu thiết kế phần mềm (Design pattern)



- Một design pattern là một giải pháp (solution) chung cho một vấn đề thông thường trong công nghiệp phát triển phần mềm
- Đưa ra mô tả và cách giải quyết vấn đề trong các trường hợp khác nhau



## Mẫu thiết kế phần mềm (Design pattern)



- **Những mẫu (Patterns) là cách giải quyết vấn đề được chuẩn hóa (formalized) cho việc phát triển ứng**
- **Một số Design pattern**
  - OOP
  - Connection pooling
  - Observer pattern
  - MVC
  - ...



## MVC



- **MVC (Model – View – Controller)**
- **Là mẫu thiết kế (design pattern) trong công nghệ phần mềm**
- **Đưa ra cách thiết kế và phát triển phần mềm có sự tách biệt giữa xử lý ứng dụng (application logic) với tầng hiển thị (presentation)**



## Nội dung



1. Giới thiệu
2. Các thành phần trong mô hình MVC
3. Mô hình hoạt động giữa các thành phần trong MVC
4. Ưu và nhược điểm của mô hình MVC
5. Phát triển Web JEE với MVC

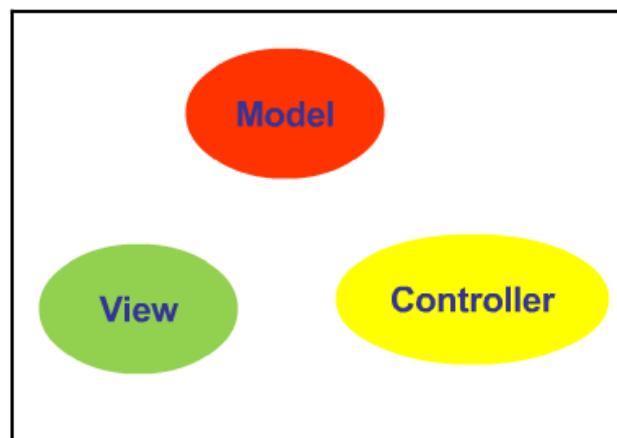


## Kiến trúc MVC



- **Kiến trúc phần mềm MVC chia thành phần model, giao diện (user interface) và xử lý vào 3 thành phần**

- Model
- View
- Controller



## Model



- Domain data
- Business logic
- Cơ chế persistence



## View



- Các phần tử giao diện (UI) cho phép truy cập và xử lý dữ liệu trong model
- UI có thể là:
  - GUI trong Java (AWT, Swing)
  - HTML
  - XML



## Controller



- **Thành phần trung gian giao tiếp giữa view – model**
- **Ánh xạ hành động trên view xuống nghiệp vụ dưới model**
- **Lựa chọn view trả về cho người dùng**



## Nội dung



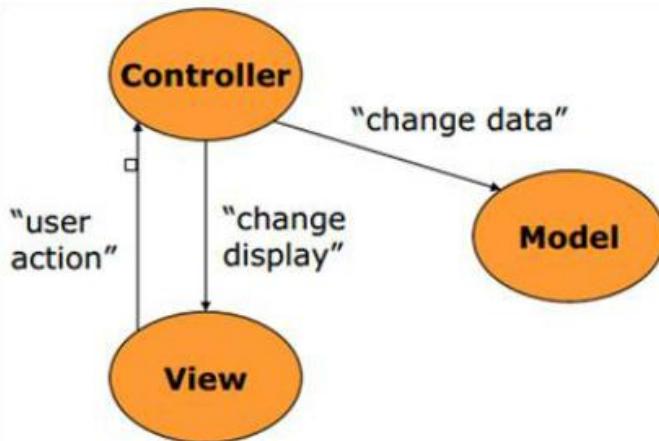
1. Giới thiệu
2. Các thành phần trong mô hình MVC
3. **Mô hình hoạt động giữa các thành phần trong MVC**
4. Ưu và nhược điểm của mô hình MVC
5. Phát triển Web JEE với MVC



## Passive view pattern



- Controller là thành phần xử lý sự kiện người dùng và nghiệp vụ tầng giao diện (presentation logic)



## Passive view pattern trong Swing



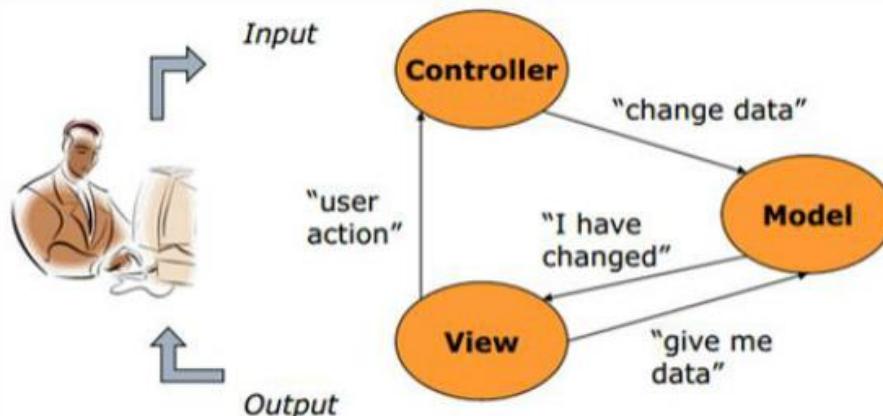
- Ánh xạ các lớp vào MVC
  - View là Swing GUI (JFrame, JButton,...)
  - Controller là bộ lắng nghe (listener)
  - Model là lớp java (hoặc database)
- Thay thế ánh xạ:
  - View là Swing GUI bao gồm Listener
  - Controller là lớp java cho business logic
  - Model là lớp java (database)



## Supervising controller pattern



- Controller đóng vai trò giám sát quá trình tương tác giữa View và Model



## Supervising controller pattern trong Swing



- Tương tự như Observer pattern
  - Một đối tượng được thông báo bởi thay đổi của đối tượng khác
  - Trong mô hình này, View là observer của Model



## Nội dung



1. Giới thiệu
2. Các thành phần trong mô hình MVC
3. Mô hình hoạt động giữa các thành phần trong MVC
- 4. Ưu và nhược điểm của mô hình MVC**
5. Phát triển Web JEE với MVC



## Ưu điểm



- **Giảm độ phức tạp code**
- **Tính tái sử dụng**
- **Tăng tính linh động**
- **Tính lỏng lẻo (loose-couple)**



## Nhược điểm



### ▪ Đối với dự án nhỏ:

- Mô hình MVC gây cồng kềnh
- Tốn thời gian phát triển
- Tốn thời gian trung chuyển dữ liệu giữa các tầng



## Nội dung



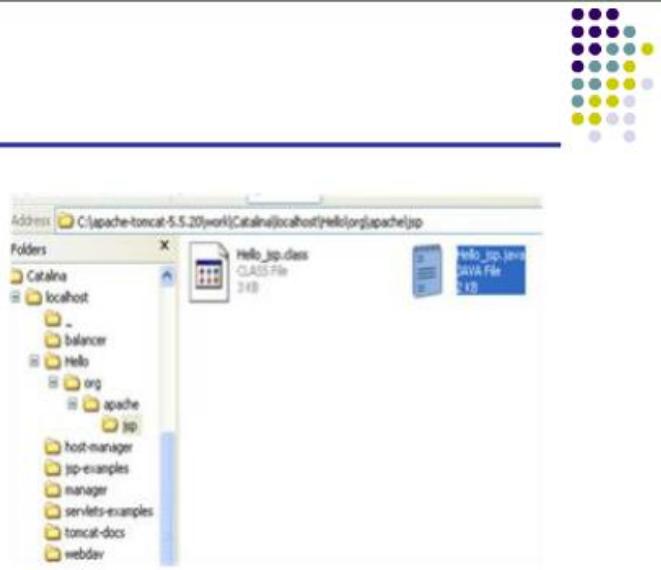
1. Giới thiệu
2. Các thành phần trong mô hình MVC
3. Mô hình hoạt động giữa các thành phần trong MVC
4. Ưu và nhược điểm của mô hình MVC
5. Phát triển Web JEE với MVC



## JSP và Servlet

### ▪ JSP

- Dễ viết
- Tầng giao diện



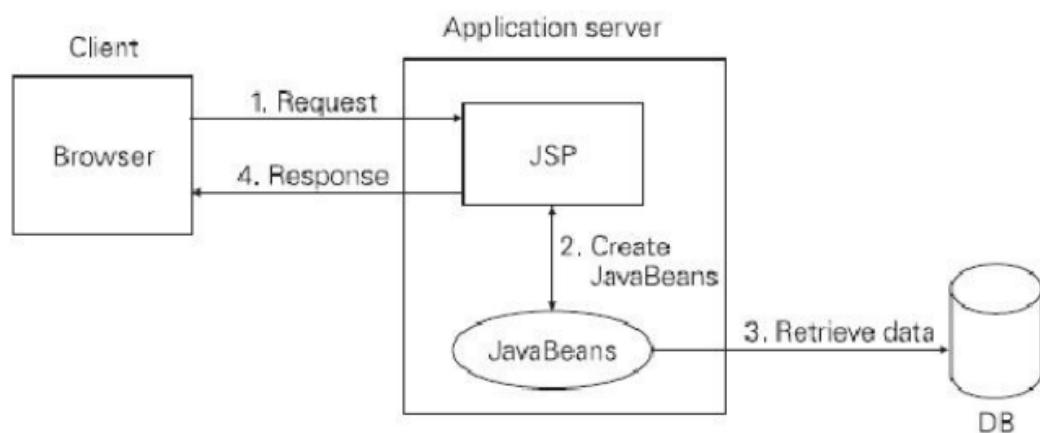
### ▪ Servlet

- Khó viết
- Tầng nghiệp vụ

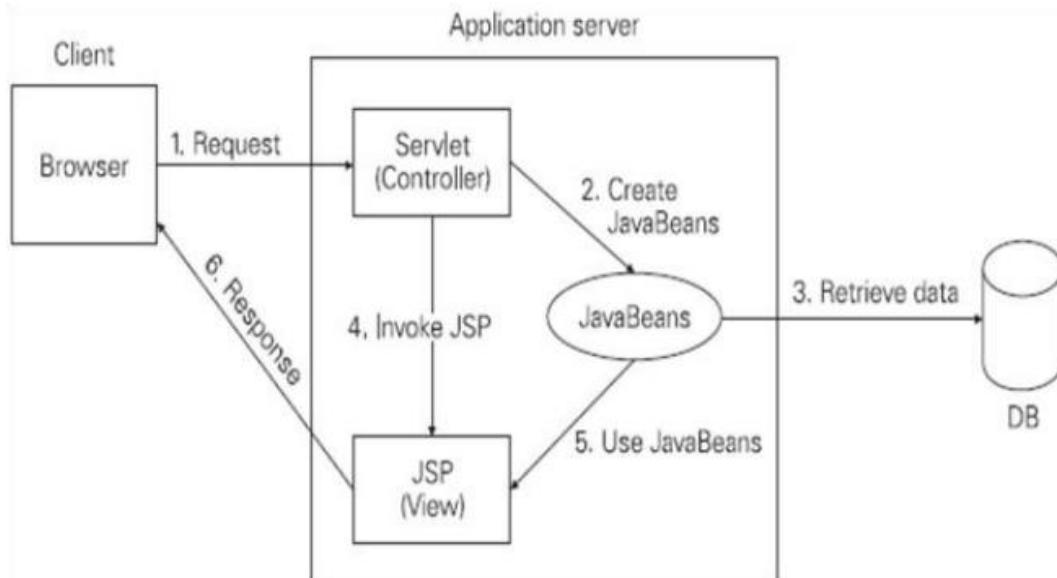
→ JSP dựa trên servlet



## Mô hình 1



## Mô hình 2 (MVC)



## Mô hình 2 (MVC)

- **Controller**
  - Servlet
- **View**
  - HTML, JSP, JSTL, ...
- **Model**
  - Java classes
  - POJOs







Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh  
TRUNG TÂM TIN HỌC

[Go Screen Capture](#)

## LTV CÔNG NGHỆ JAVA

Module 3 – Bài 8: Web Services

Ngành LT & CSDL

[www.t3h.vn](http://www.t3h.vn)



2014

2014



## Nội dung



1. Giới thiệu
2. Kiến trúc Web Services
3. Phát triển Web Services với J2EE
4. Restful Web Services



## SOA – Service-Oriented Architecture



- **Mô tả kiến trúc của một phần mềm gồm:**
  - Một tập các dịch vụ (services)
  - Cách giao tiếp giữa các dịch vụ
  - Kết nối lỏng lẻo (couple-loosely)
- **Ưu điểm**
  - Khả năng mở rộng
  - Tái sử dụng
  - Dễ bảo trì



## SOA – Service-Oriented Architecture



### ▪ Một số kỹ thuật hiện thực SOA

- REST
- RPC
- DCOM
- CORBA
- Web services
- ...



## Web Services



### ▪ Hệ thống phần mềm được thiết kế để hỗ trợ giao tiếp giữa client-server (machine-to-machine) trên môi trường mạng

- Cung cấp giao diện được đặc tả trong định dạng xml (WSDL)
- Truy cập Web Services dùng giao thức http và SOAP



## Web Services



- Cung cấp cho client giao tiếp với server nhiều cách hơn trình duyệt
- Dùng giao thức http truyền messages (được định theo giao thức SOAP)
- Được mô tả trong WSDL (Web Services Description Language)
- Nhiều công cụ sinh web servie client từ WSDL
- Đăng ký và tìm thấy trong UDDI



## Web Services



- Mô hình ứng dụng Web Services trên J2EE dựa trên công nghệ
  - XML
  - Java
  - TCP/IP
  - HTML

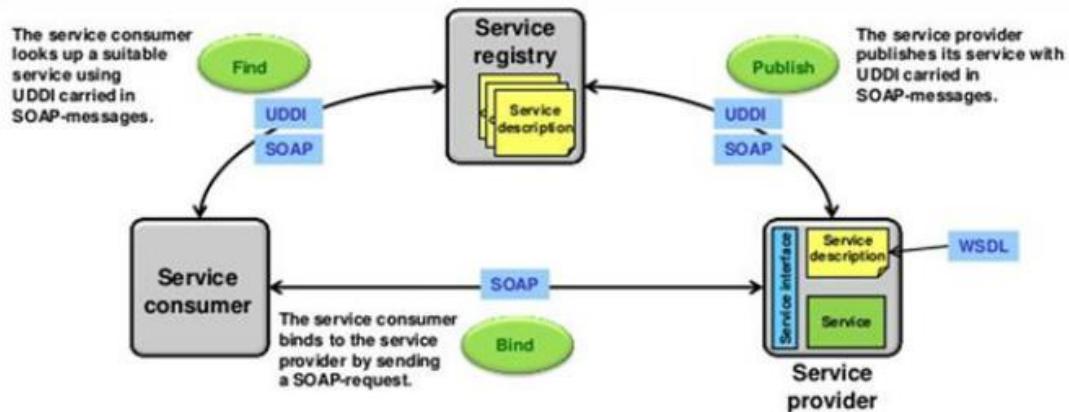


## Nội dung

1. Giới thiệu
2. Kiến trúc Web Services
3. Phát triển Web Services với J2EE
4. Restful Web Services



## Kiến trúc Web Service



## Kiến trúc Web Services



### ▪ Các phần tử

- Giao thức HTTP: giao thức giao tiếp trên mạng
- SOAP (Simple Object Access Protocol)
- UDDI (Universal Description, Discovery and Integration)
- WSDL (Web Services Description Language)



## SOAP



- Là chuẩn giao tiếp độc lập
- Là định dạng cho message gửi nhận giữa client-server, dựa trên chuẩn XML-based
- SOAP message được đóng gói bên HTTP request message và HTTP response message
- SOAP message được đóng trong một envelope gồm header và body



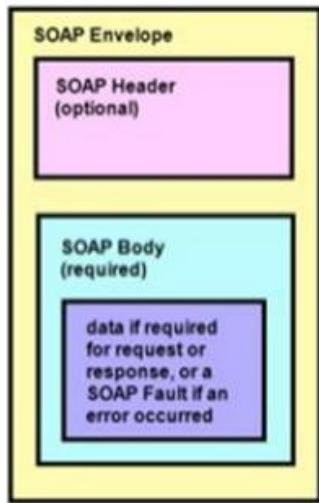
## SOAP



- **SOAP message là chuẩn XML có các thẻ (tag)**
  - Envelope: xác định xml là chuẩn SOAP message
  - Header: chứa thông tin header
  - Body: chứa thông tin gọi và trả lời giữa client-server
  - Fault: chứa các thông tin về lỗi và trạng thái



## SOAP



```
<?xml version="1.0"?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

    <soap:Body xmlns:m="http://www.example.org/stock">
        <m:GetStockPrice>
            <m:StockName>IBM</m:StockName>
        </m:GetStockPrice>
    </soap:Body>
</soap:Envelope>

<?xml version="1.0"?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

    <soap:Body xmlns:m="http://www.example.org/stock">
        <m:GetStockPriceResponse>
            <m:Price>34.5</m:Price>
        </m:GetStockPriceResponse>
    </soap:Body>
</soap:Envelope>
```



## WSDL



- WSDL dùng để xác định và mô tả Web Services
- WSDL chuẩn dựa trên XML-based và là chuẩn được qui định trong W3C
- Các công cụ Web Services dùng mô tả WSDL để sinh ra source code bên client



## WSDL



- Các phần tử trong mô tả WSDL
  - Type
    - Mô tả kiểu dữ liệu phức tạp và những phần tử được sử dụng bất kì đâu trong đặc tả WSDL
  - Import
    - Tương tự như phần tử import trong XML schema
    - Được dùng để import một đặc tả WSDL vào một đặc tả WSDL khác



## WSDL



### ▪ Các phần tử trong mô tả WSDL

- Message: Mô tả thông tin input và output của message và dùng
  - Kiểu được xây dựng trong XML schema
  - Kiểu dữ liệu phức tạp
  - Phần tử type của WSDL
  - Hoặc được định nghĩa trong đặc tả WSDL khác thông qua phần tử import



## WSDL



### ▪ Các phần tử trong mô tả WSDL

- PortType và operation
  - Mô tả giao diện của Web Service và các phương thức của nó
  - Tương ứng với java interface và các phương thức khai báo trong interface đó
- Binding
  - Gán portType và operation tới một giao thức cụ thể và kiểu mã hóa (encoding)



## WSDL



### ▪ Các phần tử trong mô tả WSDL

- Service

- Gán địa chỉ mạng cho một binding cụ thể

- Documentation

- Giải thích một vài khía cạnh của đặc tả WSDL hướng người đọc
- Bất kỳ một đặc tả WSDL nào cũng chứa phần tử documentation



## WSDL



```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="BookQuoteWS" targetNamespace="http://www.Monson-
Haefel.com/jwsbook/BookQuote" xmlns:mh="http://www.Monson-
Haefel.com/jwsbook/BookQuote" xmlns:soapbind="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/">
    <!-- message elements describe the input and output parameters -->
    <message name="GetBookPriceRequest">
        <part name="isbn" type="xsd:string" />
    </message>
    <message name="GetBookPriceResponse">
        <part name="price" type="xsd:float" />
    </message>
    <!-- portType element describes the abstract interface of a Web service -->
    <portType name="BookQuote">
        <operation name="getBookPrice">
            <input name="isbn" message="mh:GetBookPriceRequest"/>
            <output name="price" message="mh:GetBookPriceResponse"/>
        </operation>
    </portType>
```



## WSDL

```
<!-- binding tells us which protocols and encoding styles are used -->
<binding name="BookPrice_Binding" type="mh:BookQuote">
  <soapbind:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="getBookPrice">
    <soapbind:operation style="rpc" soapAction= "http://www.Monson-
      Haefel.com/jwsbook/BookQuote/GetBookPrice"/>
    <input>
      <soapbind:body use="literal" namespace="http://www.Monson-
        Haefel.com/jwsbook/BookQuote" />
    </input>
    <output>
      <soapbind:body use="literal" namespace="http://www.Monson-
        Haefel.com/jwsbook/BookQuote" />
    </output>
  </operation>
</binding>
```



## WSDL

```
<!-- service tells us the Internet address of a Web service -->
<service name="BookPriceService">
  <port name="BookPrice_Port" binding="mh:BookPrice_Binding">
    <soapbind:address location= "http://www.Monson-
      Haefel.com/jwsbook/BookQuote" />
  </port>
</service>
</definitions>
```



## UDDI



- Là dịch vụ thư mục và nơi các công ty đăng ký và tìm kiếm các Web Services
- Là thư mục cho việc lưu trữ thông tin về Web Services
- Là thư mục các giao diện của Web Service được mô tả bởi WSDL
- UDDI giao tiếp thông qua SOAP



## UDDI



## Nội dung



1. Giới thiệu
2. Kiến trúc Web Services
3. Phát triển Web Services với J2EE
4. Restful Web Services



## Công nghệ



- Java Web Services hiện tại hiện thực chuẩn là JAX-WS 2.0 và được hiện trong JEE Framework
- Ngoài ra, một số công nghệ phổ biến phát triển Java Web Services:
  - Axis: hỗ trợ phiên bản trước
  - Axis2: hỗ trợ cho REST và JAX-WS
  - JBoss-WS
  - Metro (Glassfish)



## JAX-WS



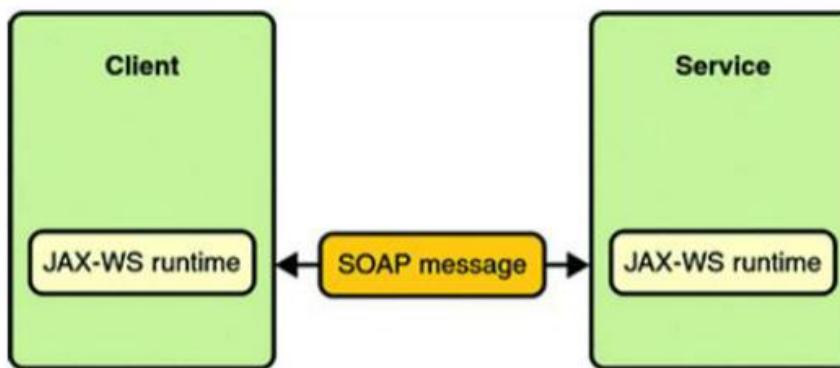
- Java API for XML Web Services
- Công nghệ cho việc xây dựng Web Services và client giao tiếp dùng XML
- Phát triển message-oriented tương tự RPC-oriented Web Services
- Hệ thống JAX-WS chuyển đổi API gọi thành SOAP message gửi và nhận nên Developer không cần phải phân tích SOAP message



## JAX-WS



- JAX-WS thay thế JAX-RPC trong JEE 5



## Mô hình phát triển



### ▪ Bottom-up

- Phát triển hiện thực service
- Sinh ra những lớp hỗ trợ
- Sinh ra WSDL
- Triển khai



## Mô hình phát triển



### ▪ Top-down

- Tạo ra đặc tả WSDL
- Sinh ra hiện thực dịch vụ
- Sinh ra các lớp hỗ trợ
- Phát triển lớp hiện thực dịch vụ
- Triển khai



## Phát triển Web Services với JAX-WS



### ▪ Server-side

- Viết một hiện thực service
- Thêm một số Annotation vào hiện thực service
- Sinh WSDL và các lớp hỗ trợ wsgen
- Đưa service ra ngoài dùng lớp endpoint

### ▪ Client-side

- Lấy file WSDL
- Sinh ra Stub từ WSDL dùng wsimport
- Gọi hàm sinh ra từ Stub



## Nội dung



1. Giới thiệu
2. Kiến trúc Web Services
3. Phát triển Web Services với J2EE
4. Restful Web Services



## Restful Web Services



- REST (Representational State Transfer)
- Được sử dụng rộng rãi để thay thế cho Web Services dùng SOAP và WSDL
- Đưa ra kiến trúc phần mềm tương tự như WWW (world wide web)
- Đơn giản hóa giao diện dùng XML trên HTTP mà không cần SOAP



## Đặc điểm chính của REST



- Dùng các giao thức http (GET, POST, PUT, DELETE,...) để gọi đến một tác vụ cụ thể
- Dùng đối số của URL như đối số của phương thức
- URI đặc tả Web Services cần gọi



## Lợi ích



- Lightweight framework
- Không sử dụng WSDL và SOAP phức tạp
- Dễ sử dụng
- Giảm sự phụ vào các thành phần thư viện ngoài (third-party libraries)



## Hiện thực REST



- JAX-WS hỗ trợ đầy đủ cho việc xây dựng Restful Web Services
- JAX-WS cho phép xây dựng RESTful endpoint thông qua giao diện javax.xml.ws.Provider
- Provider là giao diện tổng quát được hiện thực bởi một lớp và lớp hiện thực có thể được triển khai trong java EE container hoặc chế độ stand-alone với JAX-WS endpoint API



## Hiện thực REST



- **Lỗi trong REST tương tự như mã lỗi trong HTTP**
  - 400 – Bad Request
  - 403 – Forbidden
  - ...
- **Message lỗi trả về như là một phần của response**



## Server-side code



```

@WebServiceProvider
@ServiceMode(value=Service.Mode.PAYLOAD)
public class MyProvider implements Provider<Source> {
    public Source invoke(Source source)
    {
        String replyElement = new String("<p>hello world</p>");
        StreamSource reply = new StreamSource( new StringReader(replyElement));
        return reply;
    }

    public static void main(String args[])
    {
        Endpoint e = Endpoint.create( HTTPBinding.HTTP_BINDING, new MyProvider());
        e.publish("http://127.0.0.1:8084/hello/world");
        // Run forever
        e.stop();
    }
}

```



## Client-side code



```
Service service = Service.create(qname);
service.addPort(qname, HTTPBinding.HTTP_BINDING, url + "acceptPO");
Dispatch<Source> dispatcher = service.createDispatch(qname, Source.class,
Service.Mode.MESSAGE);
Map<String, Object> requestContext = dispatcher.getRequestContext();
requestContext.put(MessageContext.HTTP_REQUEST_METHOD, "POST");
Source result = dispatcher.invoke(new StreamSource(new StringReader(poXML)));
printSource(result);
```





Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh  
TRUNG TÂM TIN HỌC

[Go Screen Capture](#)

## LTV CÔNG NGHỆ JAVA

Module 3 – Bài 9: **Bảo mật Website**

Ngành LT & CSDL

[www.t3h.vn](http://www.t3h.vn)



2014

5014



## Nội dung



1. Giới thiệu
2. Bảo mật hệ thống
3. Phân loại bảo mật J2EE
4. Cấu hình bảo mật



## Giới thiệu



- Trên mạng, hàng triệu người đang truy cập và truyền thông tin cá nhân như mua hàng trên mạng
- Rất nhiều giao dịch kinh doanh, chứng khoán, ngân hàng diễn ra liên tục trên mạng
- Do đó, cần phải có những cơ chế bảo mật cho các ứng dụng đó



## Một số khái niệm cơ bản



### ▪ Authentication

- Là quá trình xác định một người hoặc cả một hệ thống. Nó xác minh rằng người sử dụng là (những) ai

### ▪ Authorization

- Là quá trình xác định xem một người dùng được phép truy cập vào một (nhiều) tài nguyên cụ thể đã được yêu cầu
- ACL (Access Control List)



## Một số khái niệm cơ bản



### ▪ Access Control: một cơ chế để hạn chế truy cập vào các đối tượng hệ thống toàn quyền

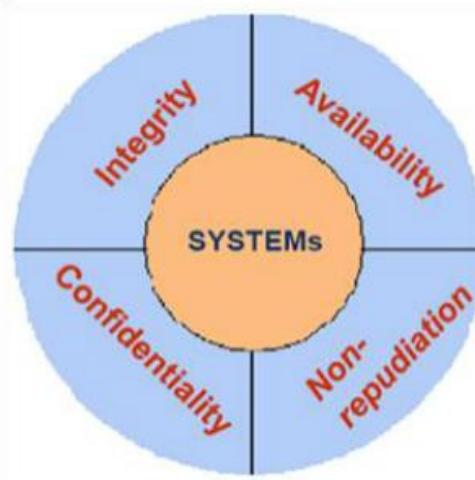


## Nội dung

1. Giới thiệu
2. Bảo mật hệ thống
3. Phân loại bảo mật J2EE
4. Cấu hình bảo mật



## Yêu cầu bảo mật



## Yêu cầu bảo mật



### ▪ Confidentiality

- Là quá trình đảm bảo rằng không một ai, ngoại trừ người sử dụng có thể truy cập thông tin nhạy cảm

### ▪ Integrity

- Là quá trình đảm bảo rằng dữ liệu không bị can thiệp khi đang chuyển từ người gửi đến người nhận



## Yêu cầu bảo mật



### ▪ Availability:

- Một hệ thống an toàn làm cho dữ liệu có sẵn cho người sử dụng có thẩm quyền

### ▪ Non-repudiation

- Người dùng không thể từ chối những gì đã thực hiện



## Một số kiểu tấn công Website



### ▪ Secrecy attacks

- Cố gắng ăn cắp thông tin bí mật bằng cách nghe lén các thông tin liên lạc giữa hai máy tính

### ▪ Integrity attacks

- Nỗ lực để thay đổi thông tin trong quá trình vận chuyển bằng mã độc hại
- Malicious code : đoạn mã gây thiệt hại cho hệ thống máy tính



## Một số kiểu tấn công Website



### ▪ Denial-of-service attacks (hoặc availability attacks)

- Cố gắng làm tràn ngập hệ thống với các yêu cầu giả để hệ thống không sẵn sàng cho các yêu cầu hợp pháp



## Một số biện pháp



### ▪ Access Control

- DAC – Discretionary Access Control
- RDAC – Role-based DAC
- MAC – Mandatory Access Control

### ▪ Flow Control

- Encryption
  - Symmetric Algorithm (DES, AES)
  - Asymmetric Algorithm (RSA, OTP)



## Một số biện pháp



### ▪ Inference Control

### ▪ Auditing



## Nội dung

1. Giới thiệu
2. Bảo mật hệ thống
- 3. Phân loại bảo mật J2EE**
4. Cấu hình bảo mật



## Cơ chế xác thực (Authentication)

- **Công nghệ Servlet hỗ trợ 4 cơ chế xác thực người dùng**
  - HTTP Basic authentication
  - HTTP Digest authentication
  - HTTPS Client authentication
  - HTTP FORM-based authentication
- **Tất cả các cơ chế đều xác thực dựa trên tài khoản và mật khẩu người dùng (username/password)**



## HTTP Basic Authentication



- Được định nghĩa trong đặc tả HTTP 1.0
- Gồm 5 bước

- B1 : Người dùng gửi yêu cầu

```
GET /servlet/SalesServlet HTTP/1.0
```



## HTTP Basic Authentication



- Gồm 5 bước

- B2 : Máy chủ quan sát thấy rằng tài nguyên được bảo vệ
  - Gửi 401 Unauthorized về client
  - Trong tin nhắn, bao gồm trong header cho trình duyệt biết rằng xác thực cơ bản là cần thiết để truy cập vào các tài nguyên
  - Header cũng xác định bối cảnh trong đó xác thực sẽ có giá trị. Bối cảnh này được gọi là realm



## HTTP Basic Authentication



```

HTTP/1.0 401 Unauthorized
Server: Tomcat/4.0.1
WWW-Authenticate: Basic realm="sales"
Content-Length=500
Content-Type=text/html

<html>
...detailed message
</html>

```



## HTTP Basic Authentication



- **Gồm 5 bước**

- B3: Trình duyệt mở một prompt yêu cầu username/password



## HTTP Basic Authentication



- **Gồm 5 bước**

- B4: Trình duyệt gửi lại yêu cầu đính kèm trong header một field Authorization

```
GET /servlet/SalesServlet HTTP/1.0
Authorization: Basic am9objpqamo=
```

- B5: Server nhận yêu cầu, kiểm tra user/pass, nếu đúng sẽ trả về tài nguyên yêu cầu, ngược lại trả về lỗi 401 Unauthorized



## HTTP Basic Authentication



- **Ưu điểm**

- Dễ cài đặt
- Hỗ trợ bởi tất cả trình duyệt

- **Khuyết điểm :**

- Không bảo mật vì user/pass không được mã hóa
- Không thể điều chỉnh cảm quan của form đăng nhập



## HTTP Digest Authentication



- Giống HTTP Basic Authentication nhưng có mã hóa user/pass nên bảo mật hơn
- **Ưu điểm:**
  - Bảo mật hơn Basic
- **Khuyết điểm:**
  - Chỉ được hỗ trợ trong IE  $\geq 5$
  - Nó không được hỗ trợ bởi nhiều servlet container từ đặc tả kỹ thuật không uỷ quyền



## HTTPS Client Authentication



- **Chứng thực được thực hiện khi kết nối SSL được thành lập giữa trình duyệt và máy chủ**
- **Tất cả các dữ liệu được truyền dưới hình thức mã hóa bằng cách sử dụng khóa công khai**
  - Trong suốt đối với người phát triển servlet
- **Ưu điểm:**



## HTTPS Client Authentication



- **Ưu điểm:**

- Bảo mật nhất trong 4 loại
- Hỗ trợ bởi hầu hết các trình duyệt thông dụng

- **Khuyết điểm:**

- Đòi hỏi một giấy chứng nhận từ một cơ quan chứng nhận (CA)
- Rất tốn kém để thực hiện và duy trì



## HTTP FORM-Based Authentication



- **Sử dụng HTML form nhận username/password**

- **Nhà phát triển sẽ xử lý xác thực**

- **Ưu điểm:**

- Dễ dàng cài đặt
- Các trình duyệt đều hỗ trợ
- Có thể điều chỉnh form đăng nhập tùy ý



## HTTP FORM-Based Authentication



### ▪ Khuyết điểm:

- Không bảo mật vì username/password không mã hóa
- Chỉ sử dụng được khi có session duy trì qua cookies hoặc HTTPS



## Nội dung



1. Giới thiệu
2. Bảo mật hệ thống
3. Phân loại bảo mật J2EE
4. Cấu hình bảo mật



## Định nghĩa tài khoản xác thực



- Tomcat định nghĩa tất cả user trong tomcat-user.xml
- RBAC - Role Base Access Control

```
<tomcat-users>
    <role rolename="tomcat"/>
    <role rolename="role1"/>
    <role rolename="employee"/>
    <role rolename="supervisor"/>

    <user username="tomcat" password="tomcat" roles="tomcat"/>
    <user username="both" password="tomcat" roles="tomcat,role1"/>
    <user username="role1" password="tomcat" roles="role1"/>
    <user name="john" password="jjj" roles="employee" />
    <user name="mary" password="mmm" roles="employee" />
    <user name="bob" password="bbb" roles="employee, supervisor" />
</tomcat-users>
```



## Cấu hình cơ chế xác thực



- Đặc tả thẻ <login-config> trong cấu hình triển khai web.xml
- DTD

```
<!ELEMENT login-config (auth-method?, realm-name?, form-login-config?)>
```

- <auth-method>: đặc tả phương thức xác thực gồm 4 giá trị là BASIC, DIGEST, CLIENT-CERT và FORM
- <realm-name>: đặc tả tên của realm dùng trong Basic authentication
- <form-login-config>: đặc tả trang login và trang lỗi dùng cho cơ chế xác thực Form



## Cấu hình cơ chế xác thực

```

<login-config>
    <auth-method>BASIC</auth-method>          Basic Authentication
    <realm-name>sales</realm-name>
</login-config>

<login-config>                                Form Authentication
    <auth-method>FORM</auth-method>
    <!--realm-name not required for FORM based authentication -->
    <form-login-config>
        <form-login-page>/formlogin.html</form-login-page>
        <form-error-page>/formerror.html</form-error-page>
    </form-login-config>
</login-config>

<html>                                         <html>
    <body>                                         <body>
        <h4>Please login:</h4>                     <h4>Sorry, your username and password do not match.</h4>
        <form method="POST" action="j_security_check">   </body>
            <input type="text" name="j_username">           </html>
            <input type="password" name="j_password">
            <input type="submit" value="OK">
        </form>
    </body>
</html>

```



## Khai báo bảo mật

- **Servlet cho phép đặc tả chi tiết yêu cầu bảo mật trong web.xml**
- **Mặc định, tất cả các tài nguyên trong ứng dụng được truy cập bởi tất cả người dùng**
- **Để hạn chế truy cập tài nguyên thì cấu hình thẻ < security-constraint> trong mô tả triển khai web.xml**



## Security Constraint



- **DTD**

```
<!ELEMENT security-constraint (display-name?,  
web-resource-collection+, auth-constraint?, user-  
data-constraint?)>
```

- **Display name:**

- Tên của security-constraint được dùng để phân biệt

- **Web resource collection**

- Xác định tài nguyên trong một ứng dụng



## Security Constraint



- **Authorization constraint**

- Xác định roles mà người dùng được gán để truy cập tài nguyên

- **User data constraint**

- Đặc tả phương thức dữ liệu được truyền tải giữa bên gửi và bên nhận



## Web Resource Collection



- **<!ELEMENT web-resource-collection (web-resource-name, description?, url-pattern\*, http-method\*)>**
  - web-resource-name
  - Description
  - url-pattern
  - http-method



## Web Resource Collection



```

<security-constraint>
    <web-resource-collection>
        <web-resource-name>reports</web-resource-name>

        <url-pattern>/servlet/SalesReportServlet/*</url-pattern>
        <url-pattern>/servlet/FinanceReportServlet/*</url-pattern>
        <url-pattern>/servlet/HRReportServlet/*</url-pattern>

        <http-method>GET</http-method>
        <http-method>POST</http-method>
    </web-resource-collection>
    ...
</security-constraint>

```



## Authorization constraint



- <!ELEMENT auth-constraint (description?, role-name\*)>
- **Role name**
  - Đặc tả role được phép truy cập tài nguyên đó
  - "\*" là chấp nhận tất cả các roles
  - Phải là tên của role được định nghĩa trong thẻ <security-role>



## Authorization constraint



```

<security-role>
    <role-name>manager</role-name>
    <role-name>director</role-name>
    <role-name>employee</role-name>
</security-role>

...
<security-constraint>
    ...
    <auth-constraint>
        <description>accessible to all managers and
                    directors</description>
        <role-name>manager</role-name>
        <role-name>director</role-name>
    </auth-constraint>
    ...
</security-constraint>

```



## User data constraint



- **<!ELEMENT user-data-constraint (description?, transport-guarantee)>**
- **Transport guarantee: có thể chứa giá trị**

- **NONE**

&lt;security-constraint&gt;

...

&lt;user-data-constraint&gt;

<description>requires the data transmission  
to be integral</description>

<transport-guarantee>INTEGRAL</transport-guarantee>

&lt;/user-data-constraint&gt;

...

&lt;/security-constraint&gt;

- **INTEGRAL**

- **CONFIDENTIAL**



```

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
    <servlet>
        <servlet-name>SecureServlet</servlet-name>
        <servlet-class>SecureServlet</servlet-class>
    </servlet>
    <security-constraint>
        <web-resource-collection>
            <web-resource-name>declarative security test</web-resource-name>
            <url-pattern>/servlet/SecureServlet</url-pattern>
            <http-method>POST</http-method>
        </web-resource-collection>
        <auth-constraint>
            <role-name>supervisor</role-name>
        </auth-constraint>
        <user-data-constraint>
            <transport-guarantee>NONE</transport-guarantee>
        </user-data-constraint>
    </security-constraint>
    <login-config>
        <auth-method>FORM</auth-method>
        <form-login-config>
            <form-login-page>/formlogin.html</form-login-page>
            <form-error-page>/formerror.html</form-error-page>
        </form-login-config>
    </login-config>
    <security-role>
        <role-name>supervisor</role-name>
    </security-role>
</web-app>

```



## Lập trình bảo mật



- Có thể sử dụng một số hàm API để quản lý thông tin bảo mật trong ứng dụng
- Giao diện HttpServletRequest cung cấp 3 phương thức

Method	Description
String getRemoteUser	This method returns the login name of the user, if the user has been authenticated, or null if the user has not been authenticated.
Principal getUserPrincipal()	This method returns a java.security.Principal object containing the name of the current authenticated user. It returns null if the user is not authenticated.
boolean isUserInRole(String rolename)	This method returns a Boolean indicating whether the authenticated user is included in the specified logical role. It returns false if the user is not authenticated.



## Lập trình bảo mật



```

String username = req.getRemoteUser();    ← Gets the username
if(username != null)
    pw.println("<h4>Welcome, "+username+"!</h4>");

if(req.isUserInRole("manager"))    ← Determines if the user is a manager   Code
{
    pw.println("<b>Manager's Page!</b>");
}
else
{
    pw.println("<b>Employee's Page!</b>");
}

<servlet>
<servlet-name>SecureServlet</servlet-name>
<servlet-class>SecureServlet</servlet-class>

<security-role-ref>
    <role-name>manager</role-name>    ← Role name hard-coded
    <role-link>supervisor</role-link>    ← Role name defined in
</security-role-ref>                                the servlet container

</servlet>

```

Web.xml



