



Bài 11

Tổng quan về Ứng dụng Web

Module: BOOTCAMP WEB-BACKEND DEVELOPMENT



Kiểm tra bài trước

Hỏi và trao đổi về các khó khăn gặp phải trong bài “Câu lệnh truy vấn SQL”

Tóm tắt lại các phần đã học từ bài “Câu lệnh truy vấn SQL”

Mục tiêu



- Giải thích được mô hình Web
- Trình bày được các thành phần của một hệ thống web
- Tạo được một ứng dụng web Java cơ bản
- Sử dụng được Servlet
- Sử dụng được JSP
- Sử dụng được form trong ứng dụng web dựa trên Servlet & JSP
- Sử dụng được các thẻ jstl thông dụng



Mô hình web

Website là gì

Stack web và dynamic web

HTTP

Web Server làm việc thế nào

Một số công nghệ phía server



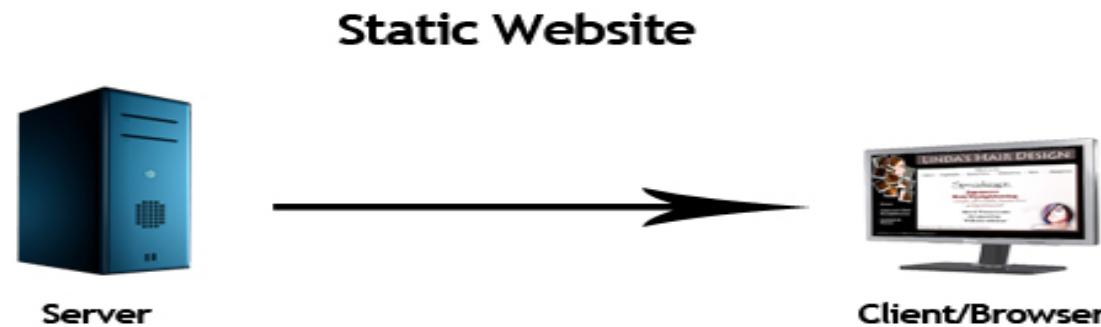
Website là gì?

- Tập hợp các trang web có thể chứa văn bản, hình ảnh, âm thanh, video.
- Trang đầu tiên của website được gọi là trang chủ.
- Mỗi website có địa chỉ cụ thể trên internet gọi là URL
- Website được lưu trữ trên một hoặc nhiều máy chủ (Server) và có thể được truy cập bằng cách tìm tới trang chủ của Website thông qua mạng máy tính.
- Website được quản lý bởi cá nhân, công ty hoặc một tổ chức.
- Có hai loại website cơ bản là web động (dynamic web) và web tĩnh (static web)

Static Web



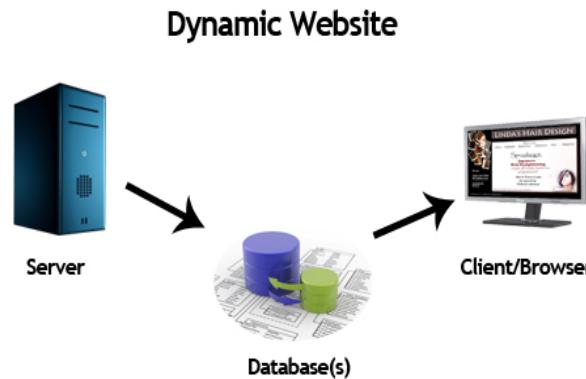
- Là loại website cơ bản được tạo mà không cần phải biết đến các ngôn ngữ lập trình web như Java, PHP, ... hoặc thiết kế cơ sở dữ liệu.
- Những trang web trong website tĩnh được viết bằng mã HTML hoặc thêm CSS, JavaScript để thêm các hiệu ứng.



Dynamic Web



- Website động là website có nội dung thay đổi.
- Sự thay đổi có thể là tùy theo thời gian, tùy theo người dùng, tùy theo ngữ cảnh
- Để tạo được website động, chúng ta thường sử dụng đến các ngôn ngữ phía server (server-side), chẳng hạn như Servlet, JSP, PHP, Python, C#...
- Một website động thường được đặt trên một máy chủ dịch vụ web (Web Server)





HTTP

Giao thức HTTP

Request, Response

Mô hình OSI

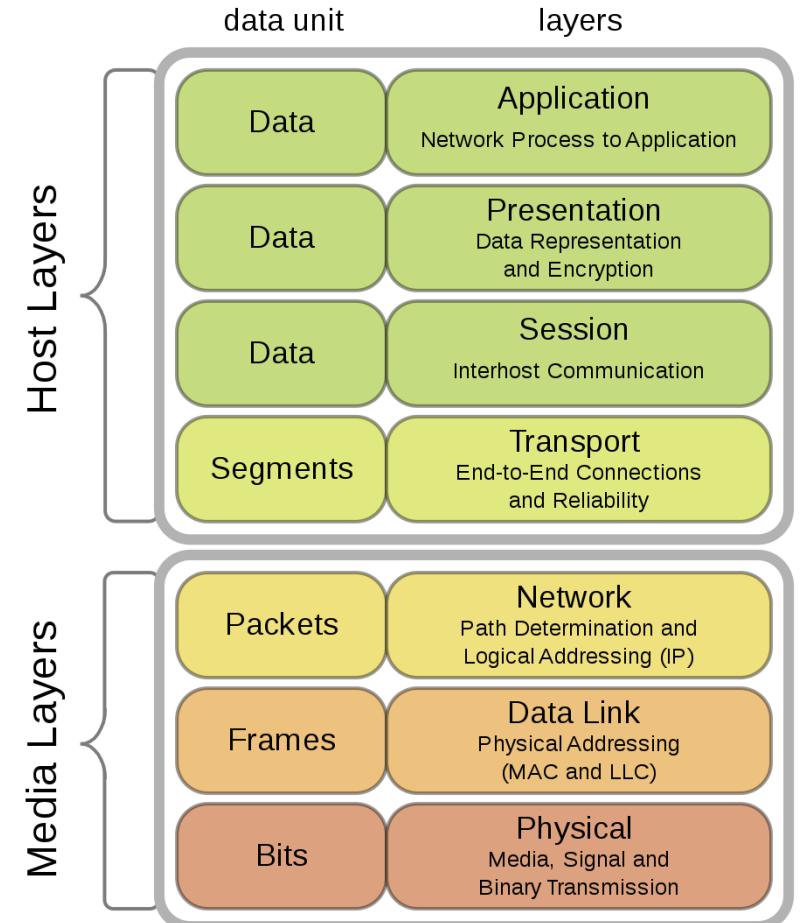


- **Mô hình OSI** (*Open Systems Interconnection Reference Model*, viết ngắn là *OSI Model* hoặc *OSI Reference Model*) - tạm dịch là **Mô hình tham chiếu kết nối các hệ thống mở**
- Là một thiết kế dựa vào nguyên lý tầng cấp, lý giải một cách trừu tượng kỹ thuật kết nối truyền thông giữa các máy vi tính và thiết kế giao thức mạng giữa chúng.
- Mô hình này được phát triển thành một phần trong kế hoạch Kết nối các hệ thống mở (*Open Systems Interconnection*) do ISO và IUT-T khởi xướng. Nó còn được gọi là **Mô hình bảy tầng của OSI**.

Tầng cấp của mău hình OSI



- Tầng 1: Tầng vật lý (Physical Layer)
- Tầng 2: Tầng liên kết dữ liệu (Data-Link Layer)
- Tầng 3: Tầng mạng (Network Layer)
- Tầng 4: Tầng giao vận (Transport Layer)
- Tầng 5: Tầng phiên (Session layer)
- Tầng 6: Tầng trình diễn (Presentation layer)
- Tầng 7: Tầng ứng dụng (Application layer)





Tầng ứng dụng - khái niệm

- **Tầng ứng dụng** là tầng thứ bảy trong bảy tầng cấp của mô hình OSI.
- Tầng này giao tiếp trực tiếp với các tiến trình ứng dụng và thi hành những dịch vụ thông thường của các tiến trình đó
- Tầng này còn gửi các yêu cầu dịch vụ tới tầng trình diễn
- Những dịch vụ thông thường của tầng ứng dụng cung cấp sự chuyển đổi về ngữ nghĩa giữa các tiến trình ứng dụng có liên quan.

Tầng ứng dụng - Ví dụ



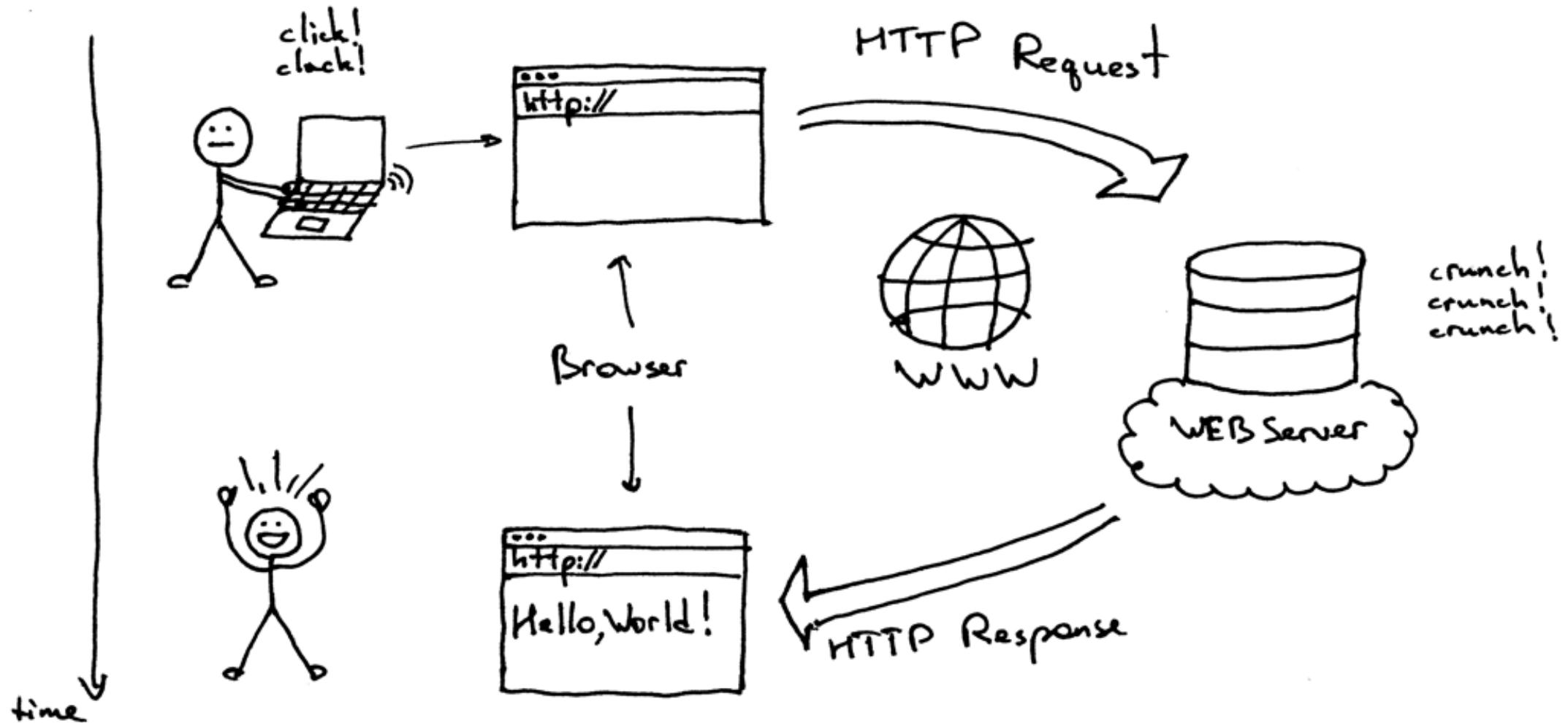
- HTTP, S-HTTP, (Secure) HyperText Transfer Protocol
 - Giao thức truyền siêu văn bản (an toàn)
- MIME, S-MIME:
 - Multipurpose Internet Mail Extensions và Secure MIME
- SMTP
 - Simple Mail Transfer Protocol

Giao thức HTTP



- HyperText Transfer Protocol
- Giao thức truyền tải siêu văn bản
- Là một trong năm giao thức chuẩn về mạng [Internet](#), được dùng để liên hệ thông tin giữa Máy cung cấp dịch vụ (Web server) và Máy sử dụng dịch vụ (Web client)
- Là giao thức Client/Server dùng cho *World Wide Web*-[WWW](#)
- **HTTP** là một giao thức ứng dụng của bộ giao thức [TCP/IP](#) (các giao thức nền tảng cho Internet)

Mô hình Client - Server



HTTP Status Code



- Mã trạng thái HTTP được server phản hồi lại mỗi khi nhận được http request
- Yếu tố Status-Code là một số nguyên 3 ký tự
 - Ký tự đầu tiên của mã hóa trạng thái định nghĩa hạng (loại) phản hồi
 - Hai ký tự cuối không có bất cứ vai trò phân loại nào
- Ký tự đầu gồm có: 1,2,3,4,5 loại lỗi.

HTTP Status Code



5.2.1 1xx: Thông tin

5.2.1.1 [100 \(Continue\)](#)

5.2.1.2 [101 \(Switching protocols\)](#)

5.2.2 2xx: Thành công

5.2.2.1 [200 \(Successful\)](#)

5.2.2.2 [201 \(Created\)](#)

5.2.2.3 [202 \(Accepted\)](#)

5.2.2.4 [203 \(Non-authoritative information\)](#)

5.2.2.5 [204 \(No content\)](#)

5.2.2.6 [205 \(Reset content\)](#)

5.2.2.7 [206 \(Partial content\)](#)

5.2.3 3xx: Sự điều hướng lại

5.2.3.1 [301 \(Moved permanently\)](#)

5.2.4 [302 \(Moved temporarily\)](#)

5.2.5 [304 \(Not modified\)](#)

5.2.6 4xx: Lỗi Client

5.2.6.1 [400 \(Bad request\)](#)

5.2.7 [401 \(Not authorized\)](#)

5.2.7.1 [403 \(Forbidden\)](#)

5.2.7.2 [404 \(Not found\)](#)

5.2.7.3 [405 \(Method not allowed\)](#)

5.2.7.4 [406 \(Not acceptable\)](#)

5.2.7.5 [407 \(Proxy authentication required\)](#)

5.2.8 [408 \(Request timeout\)](#)

5.2.8.1 [409 \(Conflict\)](#)

5.2.8.2 [410 \(Gone\)](#)

5.2.8.3 [411 \(Length required\)](#)

5.2.8.4 [412 \(Precondition failed\)](#)

5.2.8.5 [413 \(Request entity too large\)](#)

5.2.8.6 [414 \(Requested URI is too long\)](#)

5.2.8.7 [416 \(Requested range not satisfiable\)](#)

5.2.8.8 [417 \(Expectation failed\)](#)

5.2.9 5xx: Lỗi Server

5.2.9.1 [500 \(Internal server error\)](#)

5.2.9.2 [501 \(Not implemented\)](#)

5.2.9.3 [502 \(Bad gateway\)](#)

5.2.9.4 [503 \(Service unavailable\)](#)

5.2.9.5 [504 \(Gateway timeout\)](#)

5.2.9.6 [505 \(HTTP version not supported\)](#)



HTTP Request

- Một HTTP client (máy khách) gửi một HTTP request (yêu cầu) lên server (máy chủ) nhờ một thông điệp có định dạng như sau:
 - Một dòng Request-line
 - Không có hoặc có thêm các header (General|Request|Entity) theo sau bởi một ký hiệu CRLF (carriage return line feed - báo hiệu trở về đầu dòng tiếp theo)
 - Một dòng trống (VD một dòng mà không có gì ở trước CRLF) báo hiệu kết thúc phần header
 - Có hoặc không có phần nội dung thông điệp

HTTP Request



```
GET /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

```
POST /cgi-bin/process.cgi HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Content-Type: application/x-www-form-urlencoded
Content-Length: length
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive

licenseID=string&content=string&/paramsXML=string
```

HTTP Response



- Khi nhận và phiên dịch một HTTP Request
- Server sẽ gửi tín hiệu phản hồi là một HTTP Response bao gồm các thành phần sau:
 - Một dòng trạng thái (Status-Line)
 - Không hoặc nhiều hơn các trường Header (General|Response|Entity) được theo sau CRLF
 - Một dòng trống chỉ dòng kết thúc của các trường Header
 - Một phần thân thông báo tùy ý

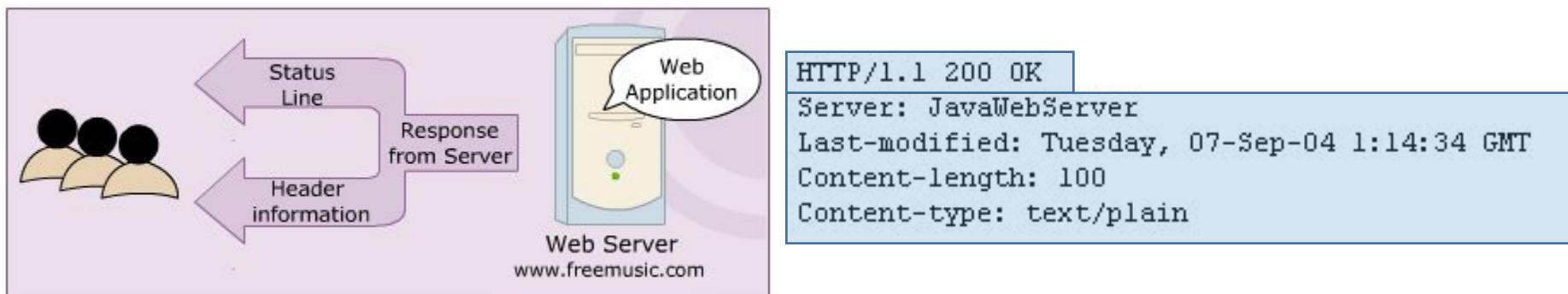
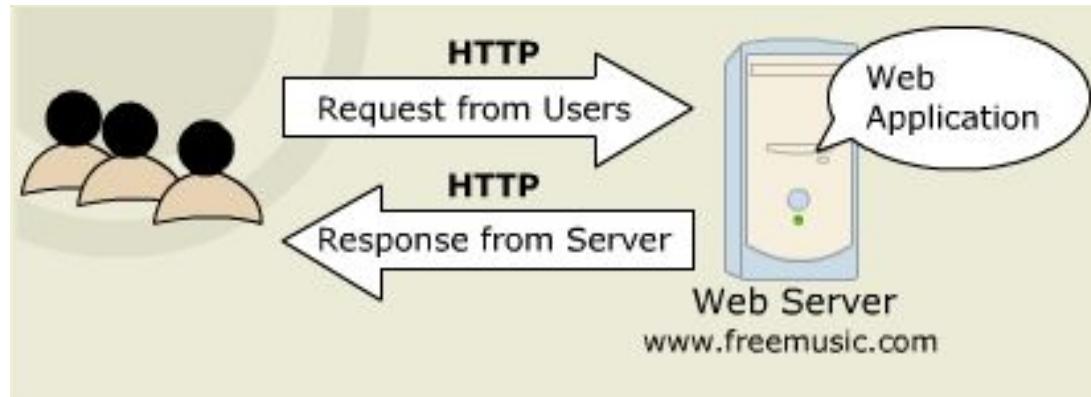
HTTP Response



```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Content-Type: text/html; charset=UTF-8
Content-Encoding: UTF-8
Content-Length: 138
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
ETag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Connection: close

<html>
<head>
    <title>An Example Page</title>
</head>
<body>
    Hello World, this is a very simple HTML document.
</body>
</html>
```

Web Server làm việc như thế nào?

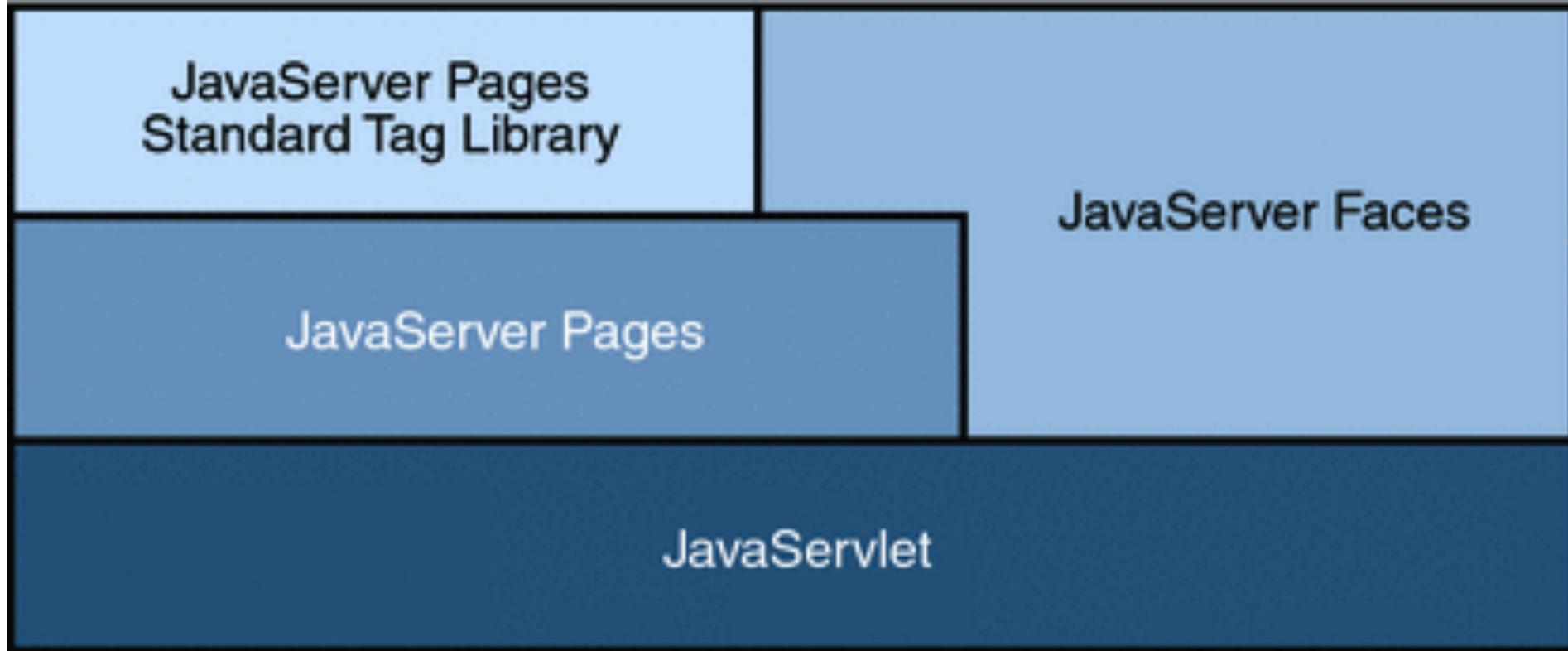


Một số công nghệ phía server



- CGI - Common Gatewway Interface
- SSJS - Server-side Java Scripts
- PHP – Personal Home Page
- Java Servlets
- JSP - Java Server Pages
- ASP - Active Server Pages

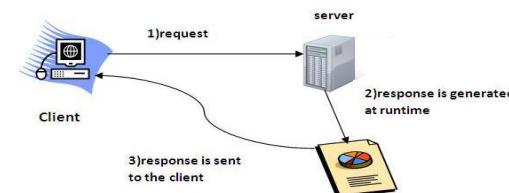
Công nghệ Java Web



Java Servlet



- **Servlet** là một công nghệ được sử dụng để tạo ra các ứng dụng web (được triển khai ở phía server để tạo ra các trang web động).
- Công nghệ Servlet rất "khỏe mạnh" và có khả năng mở rộng nhờ sử dụng ngôn ngữ lập trình java.
- Servlet làm gì?
 - Nhận client request
 - Lấy thông tin từ request
 - Xử lý nghiệp vụ hoặc phát sinh nội dung bằng cách truy cập database, triệu gọi EJB ...
 - Tạo và gửi response tới client hoặc chuyển request tới một trang servlet hoặc JSP khác.



Servlet Container (Bộ chứa các Servlet)

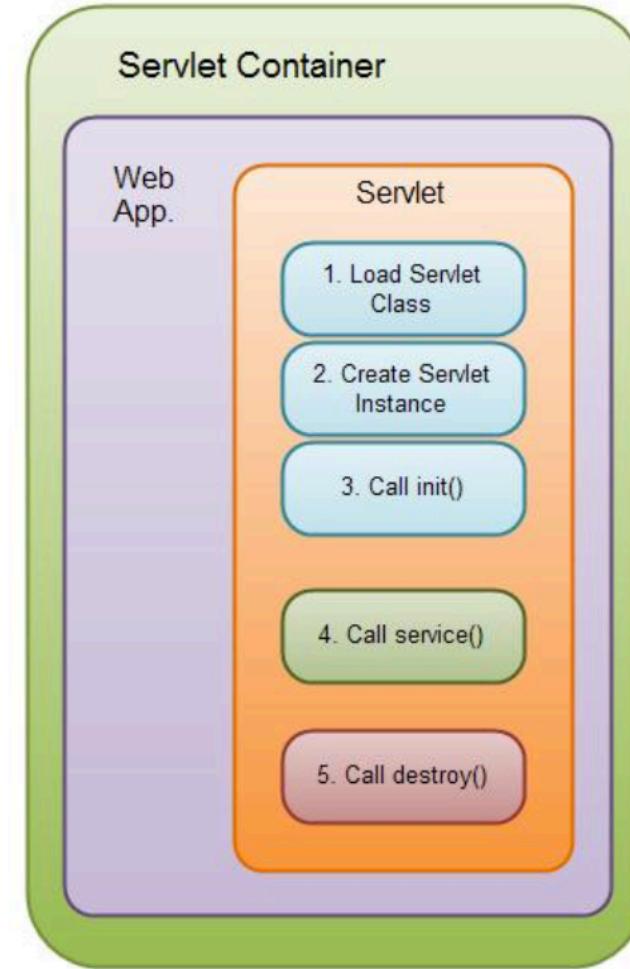


- Servlet container là bộ phận tương tác với servlet để xử lý các yêu cầu từ người dùng tới các trang web động.
- Servlet Container thực hiện các tác vụ dưới đây:
 - Quản lý vòng đời
 - Hỗ trợ xử lý đa luồng
 - Object Pooling
 - Bảo mật

Vòng đời của Servlet



- Servlet Container bảo trì vòng đời của một thực thể servlet.
- Gồm 5 bước:
 - B1: Tải Servlet Class vào bộ nhớ
 - B2: Tạo đối tượng Servlet
 - B3: Gọi phương thức init()
 - B4: Gọi phương thức service()
 - B5: Gọi phương thức destroy()





Vòng đời của Servlet

- Bước 1, 2, 3 được thực thi một lần duy nhất khi mà servlet được nạp lần đầu. Mặc định các servlet không được tải lên cho tới khi nó nhận một đòi hỏi đầu tiên từ người dùng. Bạn có thể buộc Servlet Container tải các servlet khi nó khởi động.
- Bước 4 được thực thi nhiều lần, mỗi khi có đòi hỏi từ phía người dùng tới servlet.
- Bước 5 được thực thi khi Servlet Container trút bỏ Servlet.

Ví dụ: Quan sát vòng đời của Servlet



- Xem ví dụ tại đây: <https://github.com/codegym-vn/java-web-servlet-life-cycle>

```
@WebServlet(name = "ServletLifeCycle")
public class ServletLifeCycle extends HttpServlet {
    public ServletLifeCycle()
    {
        System.out.println("Am from default constructor");
    }

    @Override
    public void init() throws ServletException {
        System.out.println("Am from Init method...!");
    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter pw = res.getWriter();
        pw.println("I am from doGet method");
        pw.close();
    }

    @Override
    public void destroy() {
        System.out.println("Am from Destroy methods");
    }
}
```

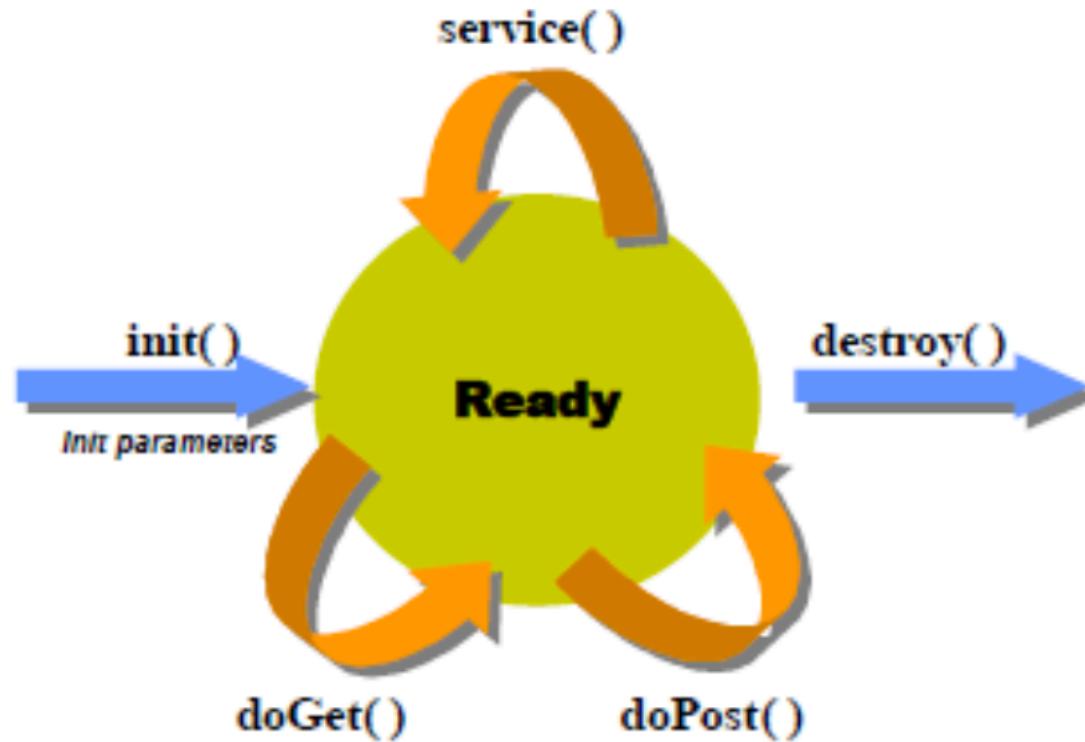
Ví dụ: Quan sát vòng đời của Servlet



- Kết quả trả về:

```
-----  
Connected to server  
[2018-04-12 05:34:31,020] Artifact web:war exploded: Artifact is being deployed, please wait...  
Am from default constructor  
Am from Init method...!  
[2018-04-12 05:34:31,772] Artifact web:war exploded: Artifact is deployed successfully  
[2018-04-12 05:34:31,772] Artifact web:war exploded: Deploy took 752 milliseconds  
12-Apr-2018 17:34:40.096 INFO [ContainerBackgroundProcessor[StandardEngine[Catalina]]] org.apache.catalina.startup.HostConfig.de  
12-Apr-2018 17:34:40.152 INFO [ContainerBackgroundProcessor[StandardEngine[Catalina]]] org.apache.catalina.startup.HostConfig.de  
/Users/VanTT/Documents/apache-tomcat-9.0.7/bin/catalina.sh stop  
NOTE: Picked up JDK_JAVA_OPTIONS: --add-opens=java.base/java.lang=ALL-UNNAMED --add-opens=java.base/java.io=ALL-UNNAMED --add-o  
12-Apr-2018 17:40:51.993 INFO [main] org.apache.catalina.core.StandardServer.await A valid shutdown command was received via the  
12-Apr-2018 17:40:51.993 INFO [main] org.apache.coyote.AbstractProtocol.pause Pausing ProtocolHandler ["http-nio-8080"]  
12-Apr-2018 17:40:52.052 INFO [main] org.apache.coyote.AbstractProtocol.pause Pausing ProtocolHandler ["ajp-nio-8009"]  
Am from Destroy methods  
12-Apr-2018 17:40:52.105 INFO [main] org.apache.catalina.core.StandardService.stopInternal Stopping service [Catalina]  
12-Apr-2018 17:40:52.128 INFO [main] org.apache.coyote.AbstractProtocol.stop Stopping ProtocolHandler ["http-nio-8080"]  
12-Apr-2018 17:40:52.130 INFO [main] org.apache.coyote.AbstractProtocol.stop Stopping ProtocolHandler ["ajp-nio-8009"]  
12-Apr-2018 17:40:52.133 INFO [main] org.apache.coyote.AbstractProtocol.destroy Destroying ProtocolHandler ["http-nio-8080"]  
12-Apr-2018 17:40:52.133 INFO [main] org.apache.coyote.AbstractProtocol.destroy Destroying ProtocolHandler ["ajp-nio-8009"]  
Disconnected from server
```

Các phương thức trong vòng đời Servlet



Các phương thức trong vòng đời Servlet

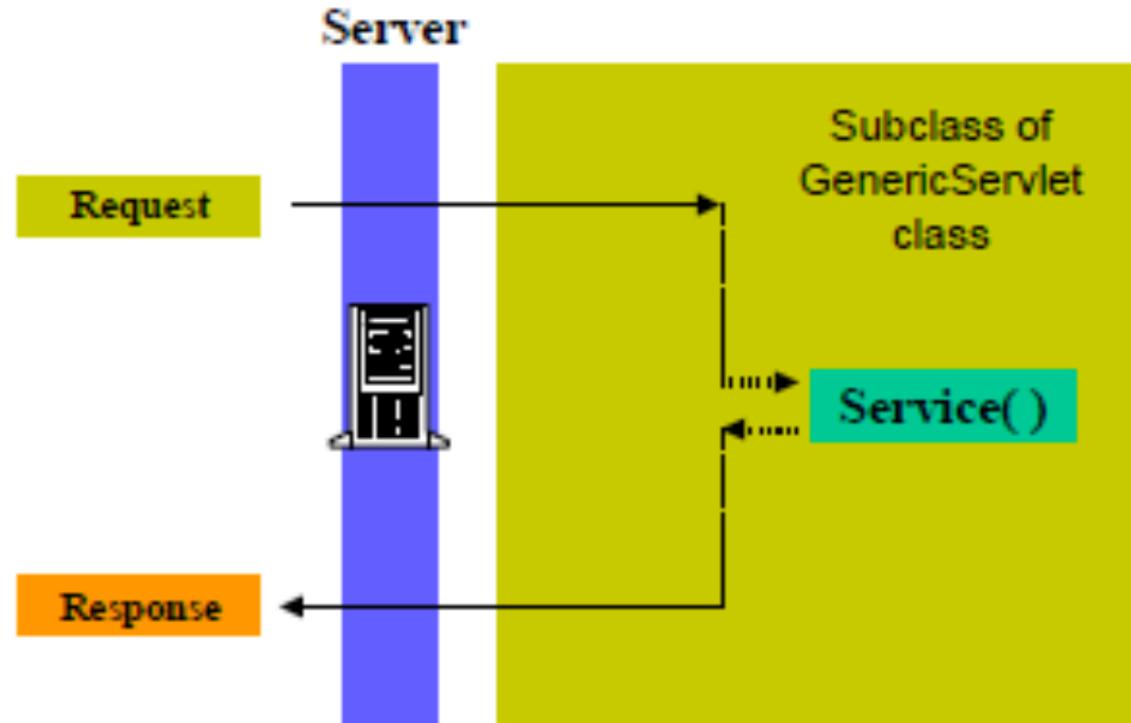


- Phương thức init()
 - Được gọi một lần khi servlet được tạo
 - Thực hiện các khởi tạo trong phương thức này như tạo một kết nối tới CSDL, mở file...
- Phương thức destroy()
 - Được gọi trước khi huỷ một đối tượng servlet
 - Thực hiện thao tác dọn dẹp như đóng file, đóng kết nối CSDL

Các phương thức trong vòng đời Servlet



- Phương thức service()
 - Được gọi mỗi khi có request từ client đến
 - Tuỳ vào loại http request cụ thể nó gọi đến một trong các phương thức doGet() hoặc doPost().
 - Tại các servlet, chúng ta cần ghi đè và xử lý các phương thức này.

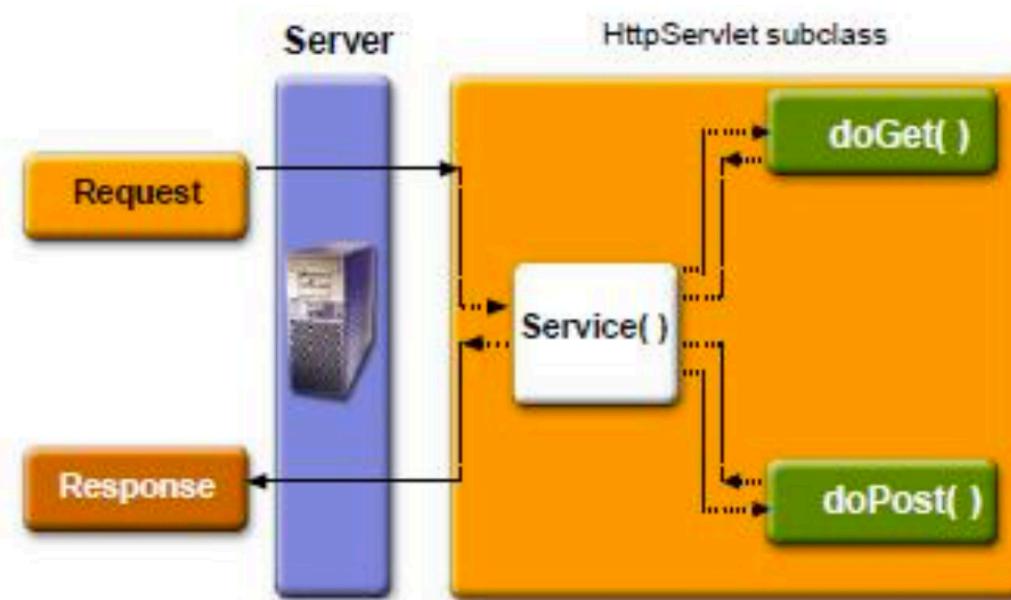


Các phương thức trong vòng đời Servlet



- Phương thức doGet()/doPost()

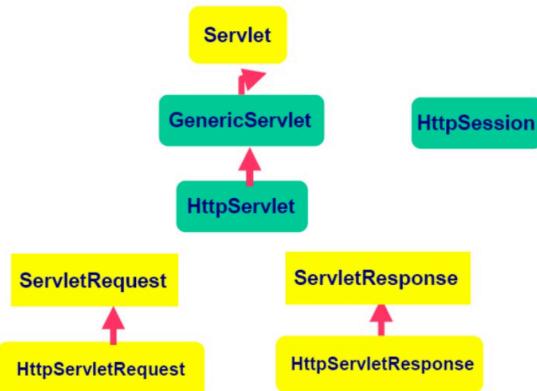
- Trích xuất các thông tin gửi từ client (HTTP parameter) từ HTTP request
- Thiết lập/truy cập các thuộc tính của các scope objects
- Thực hiện các xử lý nghiệp vụ (business logic) hoặc truy cập CSDL
- Tuỳ chọn forward request tới các Web components khác (Servlet hoặc JSP)
- Sinh HTTP response và trả về cho client



Bộ thư viện Servlet -1



- Trong Servlet có 2 gói quan trọng là javax.servlet và javax.servlet.http. Hai gói này cung cấp các interface và lớp để tạo ra các Servlet.
- Interface Servlet định nghĩa các phương thức trong vòng đời của một Servlet
- Lớp GenericServlet thực thi từ Servlet
- Lớp HttpServlet kế thừa GenericServlet, cung cấp các phương thức để xử lý các phương thức HTTP như doGet() xử lý GET, doPost() xử lý post



Bộ thư viện Servlet -2



- Để tạo một Servlet, ta cần thực thi interface Servlet trực tiếp hoặc gián tiếp thông qua GenericServlet hoặc HttpServlet.
- Trong thực tế, lớp Servlet thường kế thừa HttpServlet để có thể xử lý các phương thức HTTP.



Servlet Request

- Khi trình duyệt gửi yêu cầu (request) đến một trang web. Nó gửi rất nhiều thông tin đến web server nhưng những thông tin này không thể đọc trực tiếp vì chúng là một phần của Header trong HTTP request.
- Các phương thức thuộc lớp HttpServletRequest được sử dụng để đọc HTTP Header trong chương trình servlet

Ví dụ: Đọc thông tin HTTP Header



```
// Method to handle GET method request.  
public void doGet(HttpServletRequest request, HttpServletResponse response)  
    throws ServletException, IOException {  
  
    // Set response content type  
    response.setContentType("text/html");  
  
    PrintWriter out = response.getWriter();  
    String title = "Vi du doc thong tin HTTP Header";  
    String docType = "<!doctype html public "-//w3c//dtd html 4.0 "  
        + "transitional//en\">\n";  
  
    out.println(docType + "<html>\n" + "<head><title>" + title  
        + "</title></head>\n"  
        + "<body bgcolor = \"#f0f0f0\">\n"  
        + "<h1 align = \"center\">" + title + "</h1>\n"  
        + "<table width = \"100%\" border = \"1\" align = \"center\">\n"  
        + "<tr bgcolor = \"#949494\">\n"  
        + "<th>Header Name</th><th>Header Value(s)</th>\n"  
        + "</tr>\n");  
  
    // get header names  
    Enumeration headerNames = request.getHeaderNames();  
  
    while (headerNames.hasMoreElements()) {  
        String paramName = (String) headerNames.nextElement();  
        out.print("<tr><td>" + paramName + "</td>\n");  
        String paramValue = request.getHeader(paramName);  
        out.println("<td> " + paramValue + "</td></tr>\n");  
    }  
    out.println("</table>\n</body></html>");  
}
```

Header Name	Header Value(s)
host	localhost:8080
connection	keep-alive
user-agent	Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36
upgrade-insecure-requests	1
accept	text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
accept-encoding	gzip, deflate, br
accept-language	vi-VN,vi;q=0.8,fr-FR;q=0.6,fr;q=0.4,en-US;q=0.2,en;q=0.2



Servlet Response

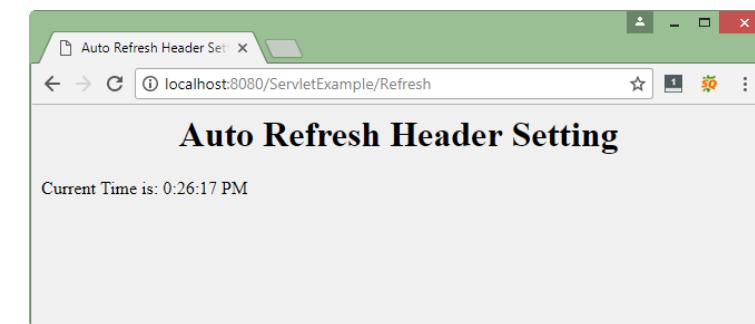
- Khi web server đáp ứng (response) yêu cầu của HTTP request. Một response thông thường bao gồm trạng thái (status), header, blank line và document

```
HTTP/1.1 200 OK
Content-Type: text/html
Header2: ...
...
HeaderN: ...
      (Blank Line)
<!doctype ...>
<html>
  <head>...</head>
  <body>
    ...
  </body>
</html>
```



Ví dụ: Hiển thị ngày giờ hệ thống

```
// Method to handle GET method request.  
public void doGet(HttpServletRequest request, HttpServletResponse response)  
    throws ServletException, IOException {  
  
    // Set refresh, autoload time as 5 seconds  
    response.setIntHeader("Refresh", 5);  
  
    // Set response content type  
    response.setContentType("text/html");  
  
    // Get current time  
    Calendar calendar = new GregorianCalendar();  
    String am_pm;  
    int hour = calendar.get(Calendar.HOUR);  
    int minute = calendar.get(Calendar.MINUTE);  
    int second = calendar.get(Calendar.SECOND);  
  
    if (calendar.get(Calendar.AM_PM) == 0)  
        am_pm = "AM";  
    else  
        am_pm = "PM";  
  
    String CT = hour + ":" + minute + ":" + second + " " + am_pm;  
  
    PrintWriter out = response.getWriter();  
    String title = "Auto Refresh Header Setting";  
    String docType = "<!doctype html public \"-//w3c//dtd html 4.0 "  
        + "transitional//en\">\n";  
  
    out.println(docType + "<html>\n" +  
        "<head><title>" + title + "</title></head>\n" +  
        "<body bgcolor = \"#f0f0f0\"\n" +  
        "<h1 align = \"center\">" + title + "</h1>\n" +  
        "<p>Current Time is: " + CT + "</p>\n");  
}
```





Apache Tomcat

Apache Tomcat

Tạo project trên IntelliJ



Apache Tomcat

- Apache Tomcat là một ứng dụng chủ (Application Server), là một phần mềm mã nguồn mở được cung cấp bởi Apache.
- Tomcat thi hành các ứng dụng [Java Servlet](#) và [JavaServer Pages \(JSP\)](#) từ [Sun Microsystems](#), và cung cấp một máy chủ HTTP cho ngôn ngữ [Java](#) thuận túy để thực thi các chương trình lệnh viết bằng ngôn ngữ [Java](#).
- Một số phiên bản:

Tomcat 3.x (1999): Reference Implementation (RI) for Servlet 2.2 and JSP 1.1.

Tomcat 4.x (2001): RI for Servlet 2.3 and JSP 1.2.

Tomcat 5.x (2002): RI for Servlet 2.4 and JSP 2.0.

Tomcat 6.x (2006): RI for Servlet 2.5 and JSP 2.1.

Tomcat 7.x (2010): RI for Servlet 3.0, JSP 2.2 and EL 2.2.

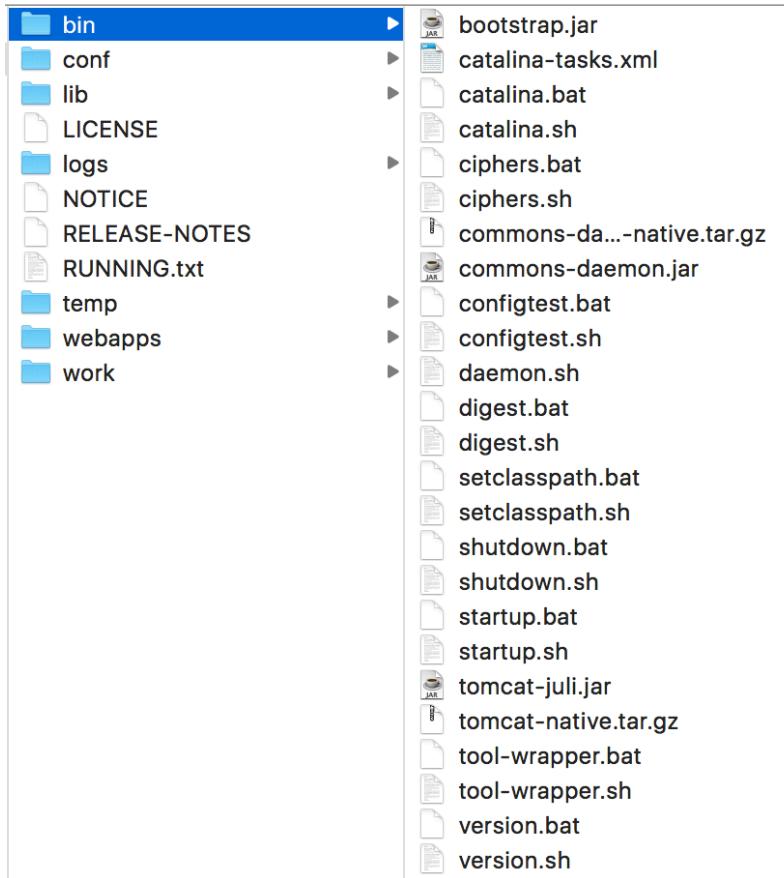
Tomcat 8.x (2013): RI for Servlet 3.1, JSP 2.3, EL 3.0 and WebSocket 1.0.

Tomcat 9.x (2018): RI for Servlet 4.0, JSP 2.3, EL 3.0, WebSocket 1.0, JASPI 1.1.

Cài đặt Web server Tomcat



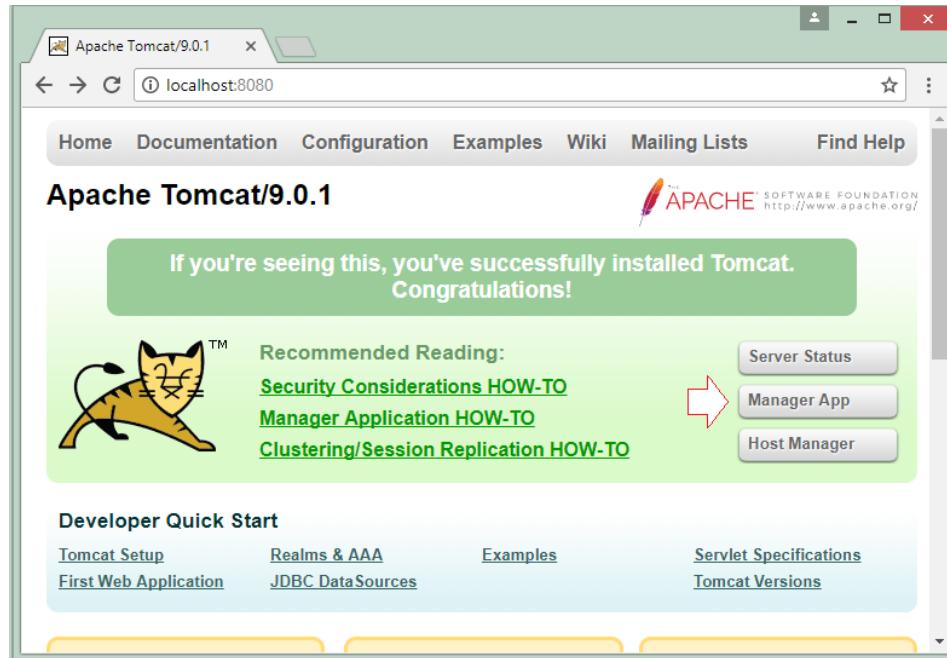
- Tải và cài Tomcat tại: <http://tomcat.apache.org>
- Cấu trúc thư mục tomcat:





Web server Tomcat

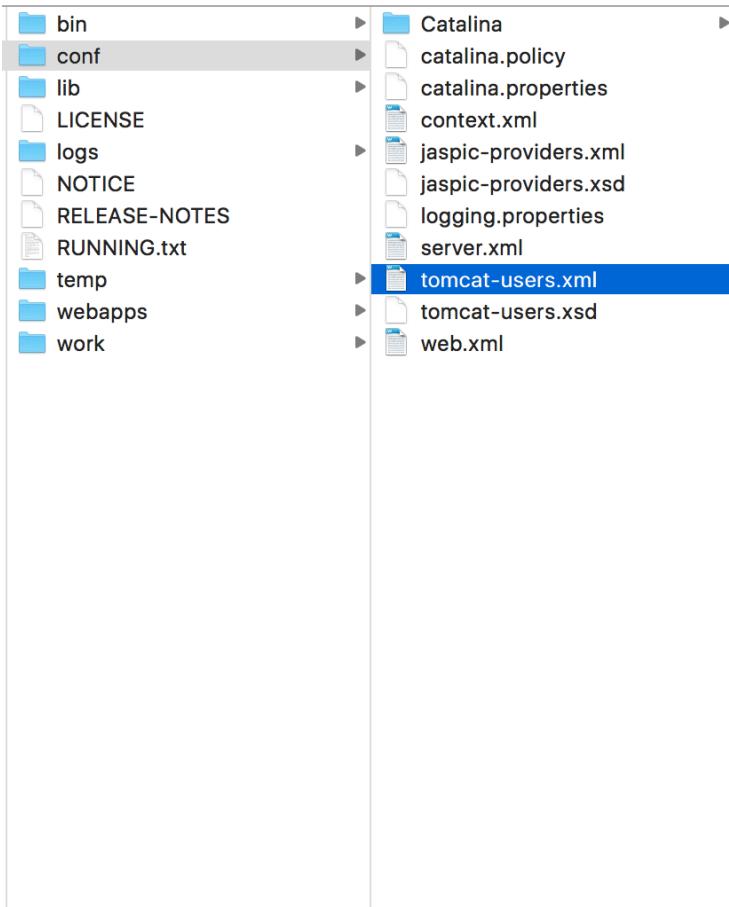
- Start Tomcat Server: Sử dụng startup.bat/ startup.sh trong window/linux
- Stop Tomcat Server: Sử dụng shutdown.bat/ shutdown.sh trong window/linux
- Sau khi start tomcat, trên trình duyệt, truy cập vào địa chỉ:
<http://localhost:8080>



Web server Tomcat



- Cấu hình các **user** được phép sử dụng **Tomcat**. Mở file **tomcat-users.xml**



```
See the License for the specific language governing permissions and
limitations under the License.

-->
<tomcat-users xmlns="http://tomcat.apache.org/xml"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:schemaLocation="http://tomcat.apache.org/xml tomcat-users.xsd"
               version="1.0">

<!--
  NOTE: By default, no user is included in the "manager-gui" role required
        to operate the "/manager/html" web application. If you wish to use this app,
        you must define such a user - the username and password are arbitrary. It is
        strongly recommended that you do NOT use one of the users in the commented out
        section below since they are intended for use with the examples web
        application.
--&gt;
&lt;!--
  NOTE: The sample user and role entries below are intended for use with the
        examples web application. They are wrapped in a comment and thus are ignored
        when reading this file. If you wish to configure these users for use with the
        examples web application, do not forget to remove the &lt;!... ..&gt; that surrounds
        them. You will also need to set the passwords to something appropriate.
--&gt;
&lt;!--
  &lt;role rolename="tomcat"/&gt;
  &lt;role rolename="role1"/&gt;
  &lt;user username="tomcat" password="123456" roles="tomcat"/&gt;
  &lt;user username="both" password=&lt;must-be-changed&gt; roles="tomcat,role1"/&gt;
  &lt;user username="role1" password=&lt;must-be-changed&gt; roles="role1"/&gt;
  &lt;user username="admin" password="admin" roles="manager-gui,manager-script,manager-
jmx,manager-status,admin-gui,admin-script"/&gt;
  &lt;user username="tomcat" password="123456" roles="manager-gui"/&gt;
--&gt;
&lt;role rolename="manager-gui"/&gt;
&lt;user username="tomcat" password="s3cret" roles="manager-gui,manager-script,manager-
jmx,manager-status,admin-gui,admin-script"/&gt;
&lt;/tomcat-users&gt;</pre>

tomcat-users.xml


```

Web server Tomcat



- **Tomcat** đã định nghĩa định nghĩa trước 4 vai trò (role) sau:

```
manager-gui - allows access to the HTML GUI and the status pages  
manager-script - allows access to the text interface and the status pages  
manager-jmx - allows access to the JMX proxy and the status pages  
manager-status - allows access to the status pages only
```

- Thêm vào file tomcat-users.xml như sau:

```
<?xml version="1.0" encoding="UTF-8"?>  
<!--...-->  
<tomcat-users xmlns="http://tomcat.apache.org/xml"  
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
    xsi:schemaLocation="http://tomcat.apache.org/xml tomcat-users.xsd"  
    version="1.0">  
    <!--...-->  
    <!--...-->  
    <!--...-->  
    <role rolename="manager-gui"/>  
    <role rolename="manager-gui"/>  
    <role rolename="manager-script"/>  
    <role rolename="manager-jmx"/>  
    <role rolename="manager-status"/>  
  
    <user username="tomcat" password="s3cret"  
        roles="manager-gui,manager-script,manager-jmx,manager-status,admin-gui,admin-script"/>  
</tomcat-users>
```

Demo: Tạo project trên IntelliJ





Java Server Page

Tìm hiểu về JSP

Luồng hoạt động của một ứng dụng web sử dụng JSP

Vòng đời JSP

JSP là gì?



- JSP (JavaServer Pages) là một ngôn ngữ kịch bản phía server, cho phép người dùng tạo ra các trang web động.
- JSP được phát hành vào năm 1999 bởi Sun Microsystems, được chạy trên nền JDK 1.3 trở về sau, là một công nghệ không thể thiếu của Java EE. Phiên bản mới nhất của JSP là 2.2.
- JSP là một tài liệu text có thể trả về cả static content (như HTML, XML, text) và dynamic content (như mã nguồn java, các thuộc tính của các lớp Java Bean, các custom tag) cho trình duyệt. Static content và dynamic content có thể đan xen lẫn nhau.
- Các file JSP được lưu với phần mở rộng là .jsp.
- Không giống như Servlet, các file JSP sẽ tự động biên dịch và triển khai khi có bất kỳ thay đổi nào.



Ví dụ file index.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1><% out.print("Hello World"); %></h1>
    </body>
</html>
```

So sánh JSP và Servlet



- Về bản chất, JSP cũng chính là Servlet. Vì trong quá trình biên dịch, JSP sẽ được chuyển thành Servlet rồi Servlet này sẽ được biên dịch. Servlet của trang JSP có vòng đời giống như các Servlet thông thường.
- **Giống nhau**
 - Đều nằm ở phía server.
 - Đều xử lý dữ liệu “động”.
 - Đều chạy với web container.
 - Đều là những công nghệ quan trọng của Java EE.

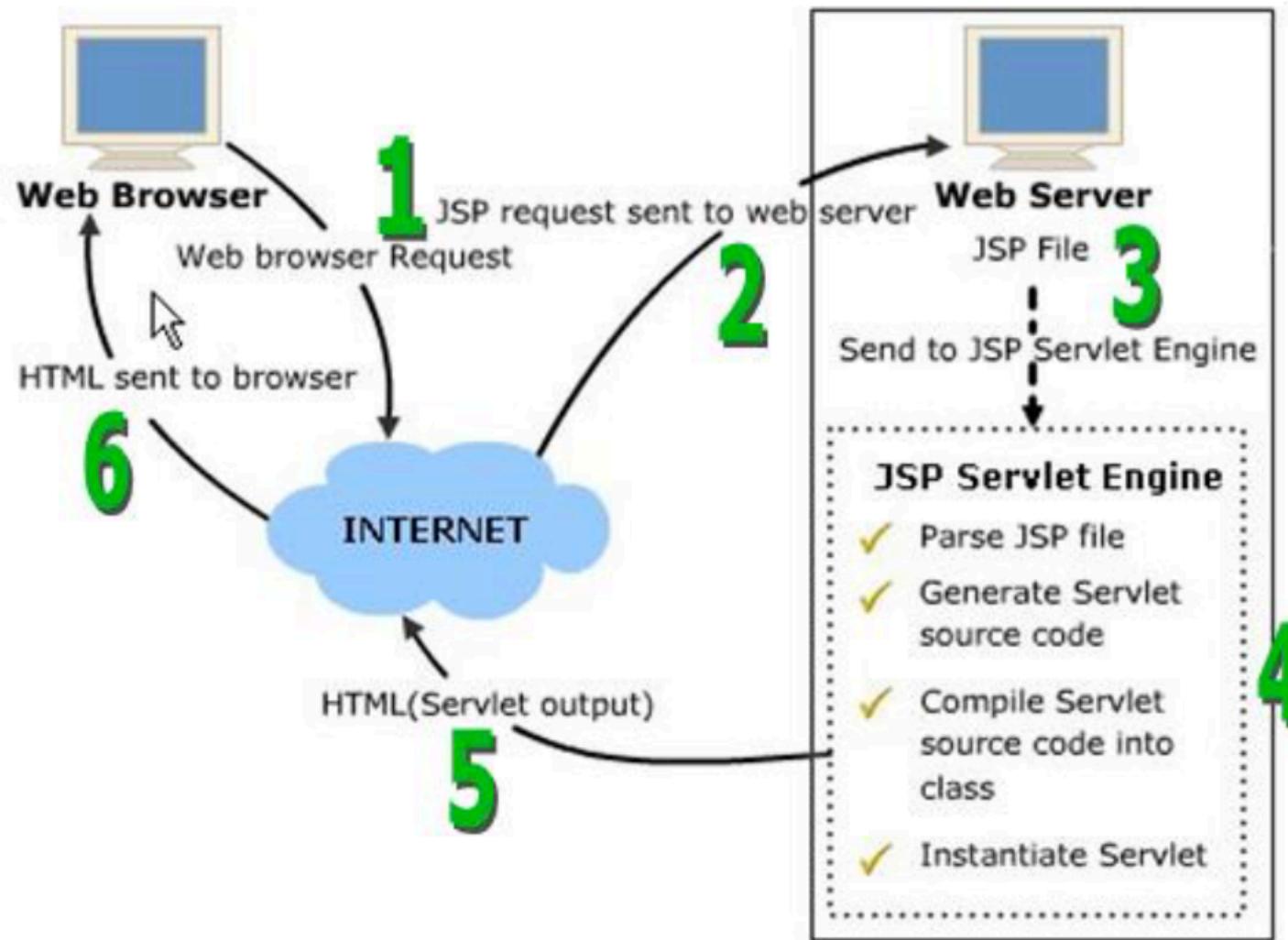
So sánh JSP và Servlet



• Khác nhau

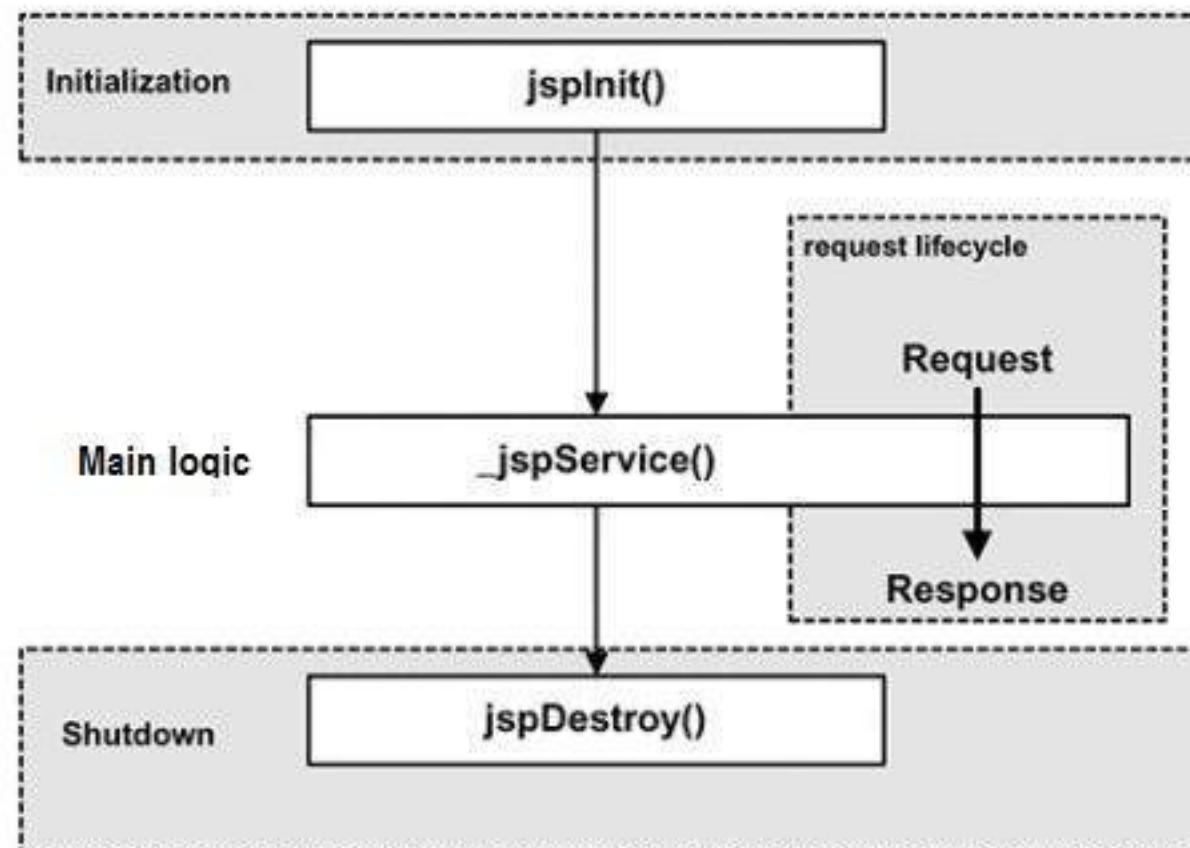
- Servlet rất mạnh về xử lý và điều phối, nhưng lại rất yếu về tạo giao diện và bảo trì web.
- JSP mạnh về xử lý hiển thị nhưng lại yếu về xử lý nghiệp vụ và điều phối.
- Trong thực tế, chúng ta kết hợp sức mạnh của Servlet và JSP vào mô hình MVC thì: Servlet đóng vai trò Controller, thì JSP đóng vai trò View.
- Ở Servlet, mã HTML nằm trong mã Java. Còn ở JSP, mã Java nằm trong mã HTML.

Cách xử lý trang JSP



Vòng đời của một trang JSP

- Vòng đời của một JSP được tính từ khi JSP đó được tạo ra cho đến khi bị hủy bỏ.
- Các giai đoạn trong vòng đời trang JSP
 - Biên dịch
 - Khởi tạo
 - Thực thi
 - Huỷ





1. Biên dịch

- Khi trình duyệt yêu cầu 1 trang JSP, JSP engine đầu tiên sẽ kiểm tra xem có cần biên dịch trang đó không. Nếu trang JSP chưa bao giờ được biên dịch hoặc đã được chỉnh sửa kể từ lần biên dịch cuối cùng thì JSP engine sẽ biên dịch trang JSP.
- Quá trình biên dịch gồm 3 bước:
 - Phân tích trang JSP
 - Chuyển trang JSP sang Servlet
 - Biên dịch Servlet



2. Khởi tạo

- Khi JSP container nạp trang JSP, nó sẽ gọi phương thức `jsplnIt()` trước khi trả lời các request khác.
- Nếu bạn cần thực hiện sự khởi tạo JSP riêng, ghi đè phương thức `jsplnIt()`:

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

- Việc khởi tạo được thực hiện chỉ một lần và với phương thức `int` của `Servlet`
- Trong `jsplnIt()`, chúng ta có thể khởi tạo kết nối tới CSDL, mở file, ...



3. Thực thi

- Khi trình duyệt yêu cầu một trang JSP, trang này đã được nạp và khởi tạo, thì JSP engine sẽ gọi phương thức `_jspService()`.
- Phương thức `_jspService()` nhận một `HttpServletRequest` và một `HttpServletResponse` như là các tham số của nó.
- Phương thức `_jspService()` của JSP được triệu hồi một lần cho mỗi yêu cầu và nó chịu trách nhiệm tạo Response cho Request đó.

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

- Phương thức này cũng chịu trách nhiệm tạo các phản hồi tới tất cả phương thức của HTTP, ví dụ: GET, POST, DELETE, ...



4. Huỷ

- Giai đoạn hủy một JSP trong vòng đời JSP biểu thị khi nào thì một JSP bị gỡ bỏ khỏi một container.
- Ghi đè phương thức `jspDestroy` khi bạn cần thực hiện bất kỳ quá trình hủy nào, ví dụ như giải phóng kết nối với Database, hoặc đóng các file.

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



Ví dụ

index.jsp

```
1 <%-- Created by IntelliJ IDEA. --%>
2 <%@ page contentType="text/html;charset=UTF-8" language="java" %>
3 <%!
4     int counter;
5     public void jspInit() {
6         counter = 0;
7         System.out.println("The lifecycle jsp has been initialized");
8     }
9 %>
10
11 <html>
12 <head>
13     <title>JSP Life Cycle Example</title>
14 </head>
15 <body>
16 <%
17     System.out.println("The lifecycle jsp has received a request");
18     counter++;
19 %>
20 <h2>JSP Life cycle : Request counter</h2>
21 <p>This page has been called <%=counter %> times </p>
22 </body>
23 </html>
24 <%!
25     public void jspDestroy() {
26         System.out.println("The lifecycle jsp is being destroyed");
27     }
28 %>
```

Cú pháp cơ bản trong JSP



- Để chèn các đoạn mã nguồn java vào bên trong trang JSP sử dụng 3 dạng cơ bản sau:
 - Expressions: <%= Expressions %>
 - Scriptlets: <% Code %>
 - Declarations: <%! Declarations%>



Các đối tượng ẩn

- Đối tượng ẩn (Implicit Object) là các đối tượng Java mà JSP Container cung cấp cho các nhà phát triển trong mỗi trang.
- Các đối tượng này được gọi trực tiếp mà không cần khai báo. Còn được gọi là các biến được định nghĩa trước.



Các đối tượng ẩn

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



- JSTL là bộ thư viện thẻ chuẩn được bổ sung với mục đích tối ưu lập trình **trong** JSP
- Các thư viện cần thiết cho JSTL gồm
 - Jstl-api.jar
 - Jstl-impl.jar
- Trong JSTL có rất nhiều bộ thẻ để xử lý các vấn đề khác nhau
 - Core: Chứa các thẻ điều khiển cơ bản
 - Format: Chứa các thẻ định dạng và đa ngôn ngữ
 - XML: Chứa các thẻ xử lý tài liệu xml
 - SQL: Chứa các thẻ làm việc với CSDL
 - Function: Chứa các thẻ cung cấp các hàm hỗ trợ cho Expression Language

Trong phạm vi bài học này, chúng ta sẽ sử dụng bộ thẻ cơ bản



Các thẻ cơ bản trong jslt

- Các thẻ cơ bản (Core Tags)

- Chứa các lệnh điều khiển như như if, choose when otherwise,vòng lặp như forEach.... Để sử dụng JSTL core tags, bạn cần khai báo vào trong trang JSP đang sử dụng như sau:

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

Các thẻ JSTL cơ bản



Thẻ	Mô tả
<u>Thẻ <c:out > trong JSTL</u>	Giống <%= ... >, nhưng cho các Expression
<u>Thẻ <c:set > trong JSTL</u>	Thiết lập kết quả của một ước lượng Expression trong một 'scope'
<u>Thẻ <c:remove > trong JSTL</u>	Gỡ bỏ một biến mục tiêu (từ một biến scope cụ thể, nếu đã xác định)
<u>Thẻ <c:catch> trong JSTL</u>	Bắt bất kỳ Throwable mà xuất hiện trong thân của nó và trưng bày nó một cách tùy ý
<u>Thẻ <c:if> trong JSTL</u>	Thẻ điều kiện đơn giản, mà ước lượng phần thân của nó nếu điều kiện đã cho là true
<u>Thẻ <c:choose> trong JSTL</u>	Thẻ điều kiện đơn giản mà thiết lập một context cho các hoạt động điều kiện loại trừ, được đánh dấu bởi <when> và <otherwise>
<u>Thẻ <c:when> trong JSTL</u>	Thẻ phụ của <choose> mà include phần thân của nó nếu điều kiện được ước lượng là true

Các thẻ JSTL cơ bản



Thẻ	Mô tả
<u>Thẻ <c:otherwise> trong JSTL</u>	Thẻ phụ của <choose> mà theo sau thẻ <when> và chỉ chạy nếu tất cả điều kiện trước được ước lượng là 'false'
<u>Thẻ <c:import> trong JSTL</u>	Thu nhận một URL tuyệt đối hoặc quan hệ và trưng bày nội dung của nó tới hoặc trang đó, một String trong 'var', hoặc một Reader trong 'varReader'
<u>Thẻ <c:forEach> trong JSTL</u>	Thẻ lặp cơ bản, chấp nhận nhiều kiểu tập hợp khác nhau và hỗ trợ subsetting (chia tập con) và tính năng khác
<u>Thẻ <c:forTokens> trong JSTL</u>	Lặp qua các token, được phân biệt bởi các dấu phân tách (delimiter) đã cung cấp
<u>Thẻ <c:param> trong JSTL</u>	Thêm một parameter tới một URL của thẻ đang chứa 'import'
<u>Thẻ <c:redirect> trong JSTL</u>	Redirect tới một URL mới
<u>Thẻ <c:url> trong JSTL</u>	Tạo một URL với các tham số truy vấn tùy ý



Ví dụ sử dụng c:out

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>c:out example</title>
</head>
<body>
<h2>c:out example</h2>
<c:out value="${'This is true: 10 > 1 '}" />
<br/>
Tag: <c:out value="${'<atag> , &'}" />
</body>
</html>
```

c:out example

This is true: 10 > 1
Tag: <atag> , &



Ví dụ sử dụng c:set

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"
%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>c:set example</title>
</head>
<body>
<h2>c:set example</h2>
<c:set scope="request" var="greeting" value="Hello every
body" />
Greeting: <c:out value="${greeting}" />
</body>
</html>
```

c:set example

Greeting: Hello every body



Ví dụ sử dụng c:if

```
<%@ taglib uri = "http://java.sun.com/jsp/jstl/core" prefix = "c" %>
<html>
<head>
    <title><c:if> Tag Example</title>
</head>
<body>
<c:set var = "salary" scope = "session" value = "${2000*2}" />
<c:if test = "${salary > 2000}">
    <p>My salary is: <c:out value = "${salary}" /></p>
</c:if>
</body>
</html>
```

My salary is: 4000



Ví dụ sử dụng c:choose - c:when - c:otherwise

```
<%@ taglib uri = "http://java.sun.com/jsp/jstl/core" prefix = "c" %>
<html>
<head>
<title><c:choose> Tag Example</title>
</head>
<body>
<c:set var = "salary" scope = "session" value = "${2000*2}" />
<p>Your salary is : <c:out value = "${salary}" /></p>
<c:choose>
<c:when test = "${salary <= 0}">
    Salary is very low to survive.
</c:when>
<c:when test = "${salary > 1000}">
    Salary is very good.
</c:when>
<c:otherwise>
    No comment sir...
</c:otherwise>
</c:choose>
</body>
</html>
```

Your salary is : 4000
Salary is very good.

c:forEach



```
<%@ taglib uri =
"http://java.sun.com/jsp/jstl/core" prefix = "c" %>
<html>
<head>
<title><c:forEach> Tag Example</title>
</head>
<body>
<c:forEach var = "i" begin = "1" end = "5">
    Item <c:out value = "${i}" /><p>
</c:forEach>
</body>
</html>
```

Item 1
Item 2
Item 3
Item 4
Item 5

Tóm tắt bài học



- HTTP là giao thức truyền tải siêu văn bản, sử dụng để thiết lập giao tiếp giữa máy chủ cung cấp dịch vụ web (Server) và máy khách sử dụng dịch vụ web (Client)
- Apache Tomcat là một ứng dụng chủ (Application Server), là một phần mềm mã nguồn mở được cung cấp bởi Apache.
- Servlet là một công nghệ được sử dụng để tạo ra các ứng dụng web.
- Các phương thức trong vòng đời của Servlet: init(), service(), destroy(), doGet(), doPost()
- JSP (JavaServer Pages) là một ngôn ngữ kịch bản phía server, cho phép người dùng tạo ra các trang web động.
- Toàn bộ trang JSP được thông dịch sang Servlet (một lần) và Servlet được thực thi khi yêu cầu của client gửi đến.



Hướng dẫn

Hướng dẫn làm bài thực hành và bài tập

Chuẩn bị bài tiếp theo: *Kiến trúc MVC*