

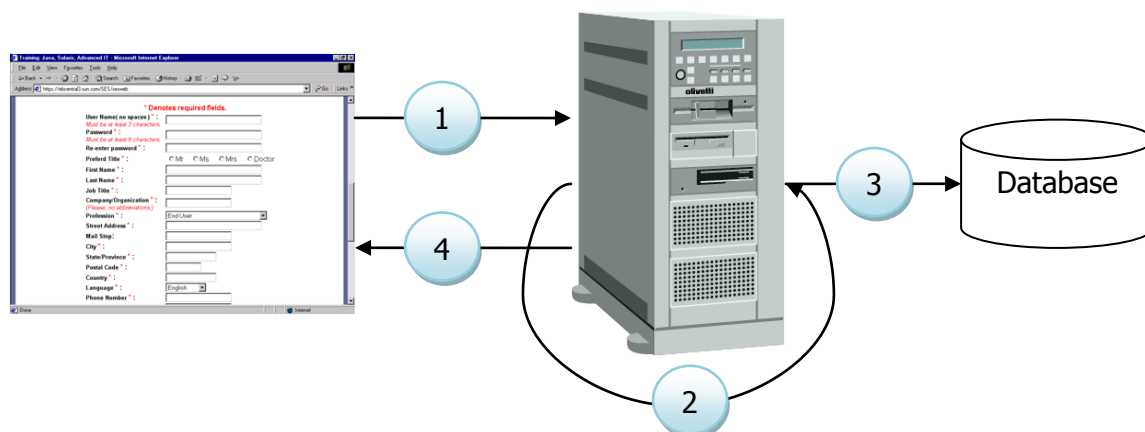
CHƯƠNG 4: SERVLET & JSP

4.1 Khai quát về lập trình web với Servlet/JSP

4.1.1 Kiến trúc công nghệ của ứng dụng web

Báo chí, email, tìm kiếm, các chương trình quản lý nghiệp vụ... là các ứng dụng web. Thể loại ứng dụng này ngày càng đa dạng, phong phú và được ứng dụng rộng rãi khắp nơi. Lợi thế của chúng là người sử dụng có thể làm việc với ứng dụng web mọi lúc mọi nơi, dễ nâng cấp – bảo trì bảo dưỡng.

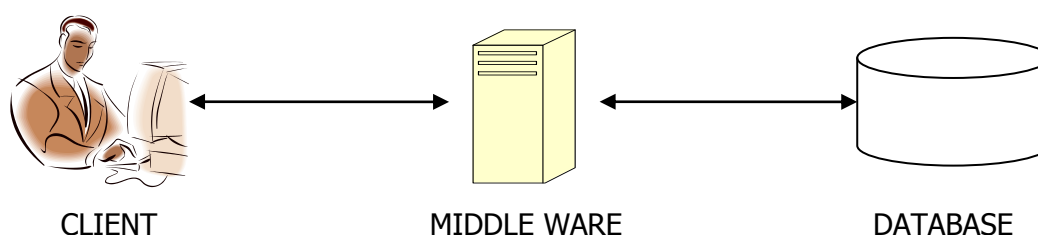
Tất cả các ứng dụng web đều được đặt ở máy chủ có cài web server. Trình duyệt web giúp người sử dụng giao tiếp với chúng. Ứng dụng web sẽ tiếp nhận và xử lý các yêu cầu của người sử dụng. Nếu ứng dụng web cần truy vấn hay lưu trữ thông tin, nó sẽ kết nối với CSDL để được trợ giúp bởi các hệ quản trị CSDL. Sơ đồ sau cho bạn biết tổng quan về qui trình truyền thông giữa trình duyệt và ứng dụng web.



Qui trình hoạt động của ứng dụng web

1. Thông tin trên form chuyển đến Web Server thông qua request
2. Server thực hiện các xử lý được cài đặt (server script – Server Side)
3. Kết nối và thao tác CSDL
4. Kết quả xử lý được trả về qua response

Đứng ở góc độ phân tán, ứng dụng web có kiến trúc 3 tầng (3-tier)



Kiến trúc ứng dụng 3 tầng

- ✓ CLIENT: trình duyệt web, như Internet Explorer, FireFox, Chrome, Opera...
- ✓ MIDDLE WARE: web server và các công nghệ server như JSP/Servlet, PHP, ASP.NET...

✓ DATABASE: hệ quản trị cơ sở dữ liệu như SQL Server, Oracle, MySQL...

4.1.2 Công nghệ lập trình web Servlet/JSP

Servlet là công nghệ lập trình web của Sun Microsystem được đưa ra để khắc phục các nhược điểm của CGI. Servlet sử dụng kỹ thuật đa tiến trình (thread) trình để phục vụ các yêu cầu thay vì đa tiến trình (process) như CGI. Một thread được sinh ra để phục vụ cho một yêu cầu, hơn nữa servlet được tải vào một lần sau đó cư trú trong bộ nhớ để phục vụ nhiều yêu cầu tiếp theo mà không kết thúc như cách xử lý của CGI. Môi trường server (Servlet engine) đủ thông minh để giải phóng các servlet vô dụng (theo tiêu chuẩn ưu tiên nào đó) khi cần dành tài nguyên phục vụ cho các hoạt động cần thiết hơn.

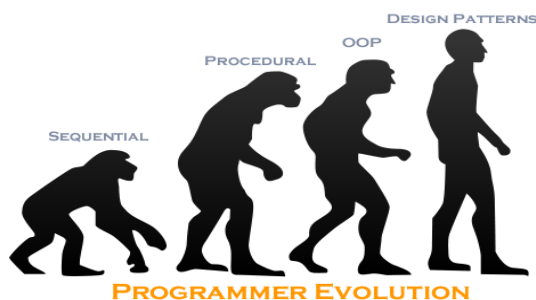
Chạy được trên mọi môi trường (phần cứng, phần mềm) chỉ cần ở đó có máy ảo java. Nhờ ưu điểm này mà ứng dụng Java thường được chọn cho những doanh nghiệp lớn với hệ thống cơ sở hạ tầng phức tạp như Unix, Linux, Window...

Virus không thể lây qua tập tin class vì trước khi chạy, class còn được kiểm duyệt.

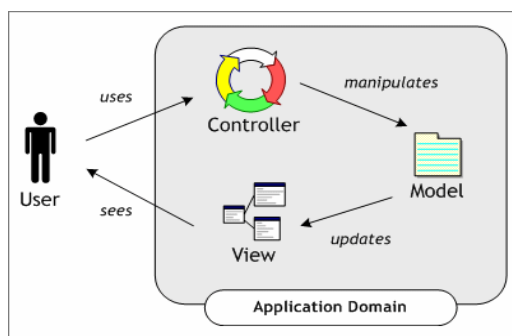
Kết hợp lập trình servlet và jsp là một thể mạnh của lập trình web với Java. Khi đó Servlet đóng vai trò tiếp nhận và xử lý yêu cầu còn JSP sẽ đảm nhận trách nhiệm sinh giao diện để hiển thị cho người dùng. JavaBean, EL và JSTL làm tăng thêm sức mạnh cho JSP. Ứng dụng web với Servlet và JSP trở nên hoàn hảo hơn khi bạn triển khai theo mô hình MVC.

4.1.3 Mô hình lập trình MVC với Servlet/JSP

Người ta ví sự tiến triển tự nhiên của một lập trình viên từ lúc mới biết viết mã bừa bải (sequential) -> tổ chức thủ tục (procedural) -> hướng đối tượng (OOP) -> khung mẫu (Design Pattern) như lịch sử tiến hóa của loài người.



MVC là một Design Pattern, nó là một mô hình lập trình chuẩn được sử dụng rộng rãi trên thế giới và mang lại hiệu quả cao cả trong quản lý lẫn hiệu năng ứng dụng. Các framework nổi tiếng như Struts, Spring, JSF, Zend, MS MVC... đều sử dụng mô hình lập trình MVC.

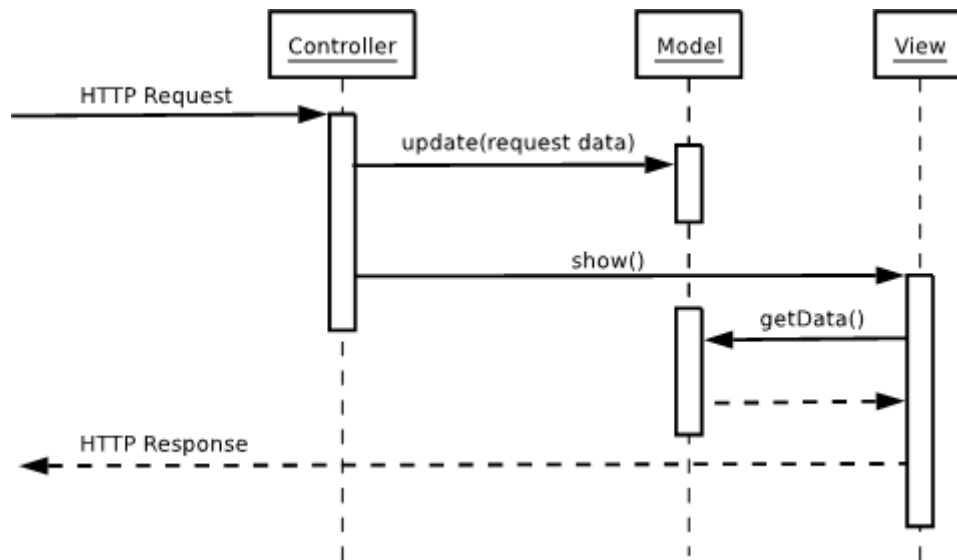


Yêu cầu được gửi đến Controller (Servlet). Controller tiếp nhận, xử lý và làm việc với dữ liệu thông qua Model. Cuối cùng chuyển tiếp đến View (JSP) để sản sinh giao diện cho người dùng.

View lấy dữ liệu từ Model (đã được tạo ra từ Controller trước đó) để sản sinh giao diện theo yêu cầu.

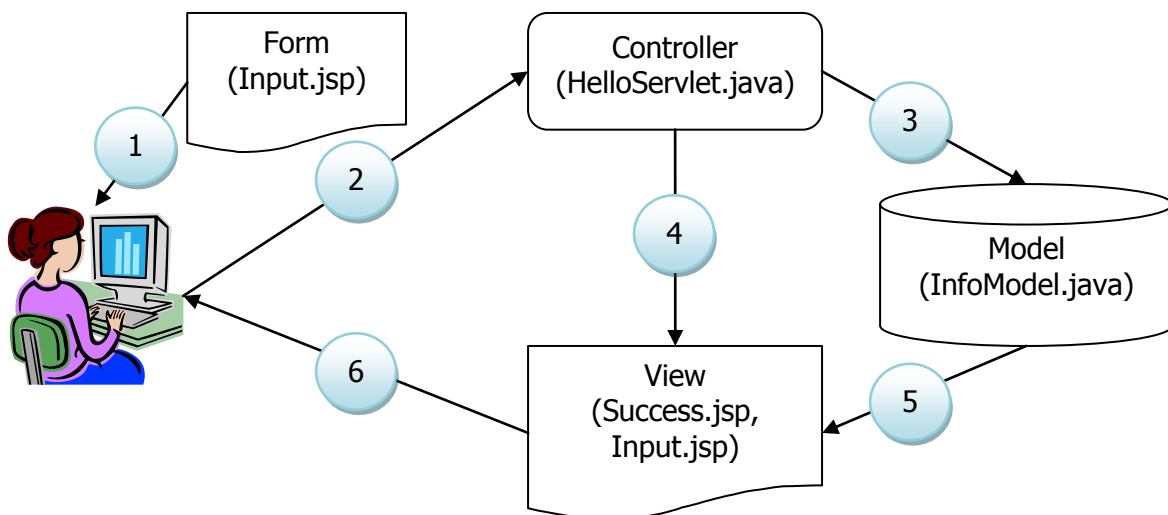
Model đóng vai trò cung cấp, cập nhật dữ liệu và chia sẻ giữa Controller và View.

Hoạt động của mô hình MVC được mô tả như sơ đồ sau



4.1.4 Triển khai mô hình MVC với Servlet/JSP

Ví dụ sau đây triển khai chức năng đăng nhập theo mô hình MVC để bạn hiểu một cách cụ thể hơn về mô hình này.



(1) Truy cập form nhập (Input.jsp)

(2) Submit dữ liệu đến Controller (hello.do - HelloServlet.java)

(3) Controller tiếp nhận, kiểm tra tính hợp lệ của dữ liệu.

+ Nếu hợp lệ: Khởi tạo, chia sẻ InfoModel và (4) chuyển tiếp sang Success.jsp

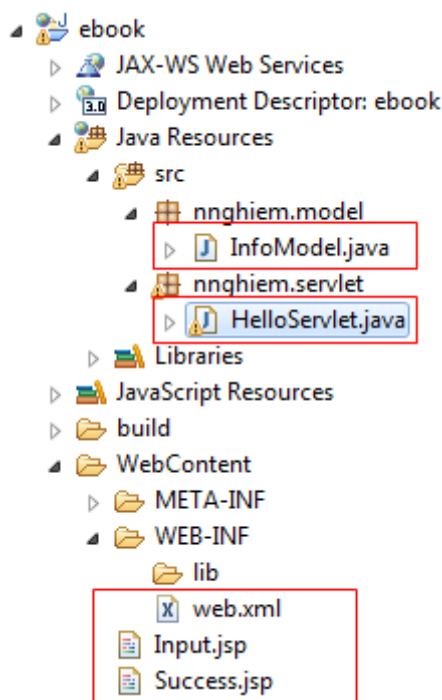
+ Ngược lại: chia sẻ một thông báo lỗi và (4) chuyển trở lại View Input.jsp

(5) View (Success.jsp) sử dụng InfoModel được khởi tạo bởi Controller để (6) hiển thị thông tin lời chào. Hoặc View (Input.jsp) sử dụng thông báo lỗi được khởi tạo bởi Controller để (6) hiển thị cho người sử dụng.

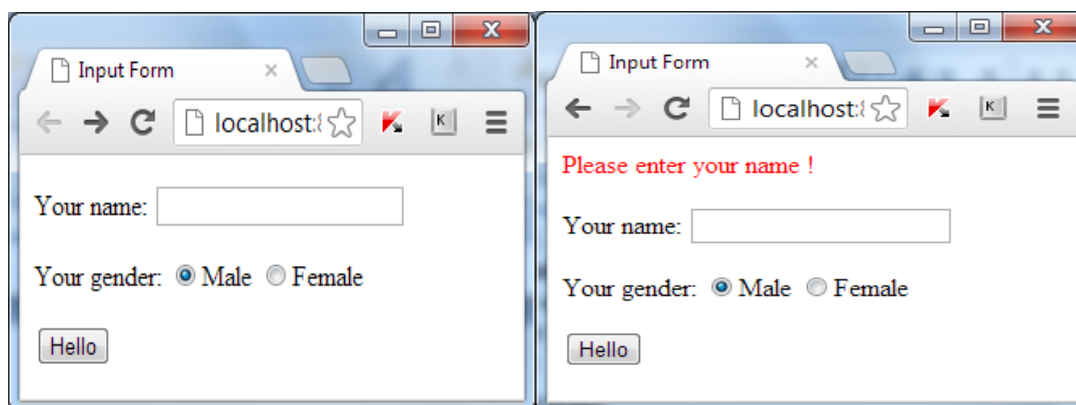
Để triển khai theo mô hình MVC, bạn cần thực hiện các công việc sau đây

- ☐ Tạo form nhập thông tin - Input (Input.jsp)
- ☐ Tạo servlet tiếp nhận và xử lý yêu cầu - **Controller** (HelloServlet.java)
- ☐ Tạo trang jsp hiển thị sau khi yêu cầu được xử lý - **View** (Success.jsp)
- ☐ Tạo model chứa thông tin chia sẻ giữa servlet và jsp - **Model** (InfoModel)
- ☐ Ánh xạ servlet bằng annotation hoặc khai báo trong file cấu hình - Web.xml

Sau đây là tổ chức và mã nguồn của các thành phần nêu trên



☐ Input (Input.jsp)



```
<%@ page contentType="text/html; charset=utf8" pageEncoding="utf8"%>
<!DOCTYPE html>
```

```
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf8">
    <title>Input Form</title>
</head>
<body>
<label style="color:red">${error}</label>
<form method="post" action="hello.do">
    <p>Your name:
    <input name="txtFullName">

    <p>Your gender:
    <input type="radio" name="rdoGender" value="Nam" checked>Male
    <input type="radio" name="rdoGender" value="Nu">Female

    <p>
    <input type="submit" name="btnHello" value="Hello">
</form>
</body>
</html>
```

Form sẽ hiển thị thông báo `${error}` nếu có và form sẽ submit dữ liệu (`txtFullName` và `rdoGender`) đến `login.do` theo phương thức POST khi bạn nhấp chuột vào nút `btnHello`.

❑ Controller (HelloServlet.java)

```
package nnghiem.servlet;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;

import nnghiem.model.InfoModel;

public class HelloServlet extends HttpServlet{

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        req.getRequestDispatcher("Input.jsp").forward(req, resp);
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        //1. Nhận tham số
        String name = req.getParameter("txtFullName");
        String gender = req.getParameter("rdoGender");

        //2. Xử lý và chia sẻ
        if(name.trim().length() > 0){
            //3. Tạo InfoModel và chia sẻ với View
            InfoModel model = new InfoModel(name, gender);
```



```

        req.setAttribute("info", model);

        //4. Chuyển tiếp đến View (Success.jsp) để hiển thị info
        req.getRequestDispatcher("Success.jsp").forward(req, resp);
    }
    else{
        //3. Chia sẻ thông báo lỗi với View
        req.setAttribute("error", "Please enter your name !");

        //4. Chuyển tiếp đến View (Input.jsp)
        req.getRequestDispatcher("Input.jsp").forward(req, resp);
    }
}
}

```

Khi bạn nhấn cầu servlet này theo phương thức GET thì phương thức `doGet()` của servlet sẽ chạy. Công việc của nó là chuyển tiếp đến Input.jsp (`req.getRequestDispatcher().forward()`) và vì vậy bạn nhận được form nhập Input.jsp. Ngược lại nếu bạn nhấn cầu theo phương thức POST thì phương thức `doPost()` của servlet sẽ chạy để xử lý yêu cầu của bạn.

Trước tiên nó gọi phương thức `req.getParameter()` để nhận tham số từ req (gồm tên và giới tính trên form nhập).

Nếu bạn không nhập tên vào thì servlet sẽ chia sẻ một thông báo lỗi (`setAttribute("error")`) và chuyển tiếp đến Input.jsp để hiển thị thông báo lỗi và yêu cầu đăng nhập lại.

Ngược lại (bạn nhập thông tin hợp lệ) thì servlet sẽ khởi tạo đối tượng từ InfoModel, chia sẻ đối tượng này (`setAttribute("info")`) và chuyển tiếp đến Success.jsp để hiển thị.

❑ Model (InfoModel)

```

package nhgkiem.model;

public class InfoModel {
    String name, gender;

    public InfoModel(){

    }

    public InfoModel(String name, String gender){
        this.name = name;
        this.gender = gender;
    }

    public String getGreeting() {
        String info = "Hello %s %s";
        if(gender.equals("Nam")){
            return String.format(info, "Mr.", name);
        }
        return String.format(info, "Ms.", name);
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {

```

```

        this.name = name;
    }

    public String getGender() {
        return gender;
    }
    public void setGender(String gender) {
        this.gender = gender;
    }
}

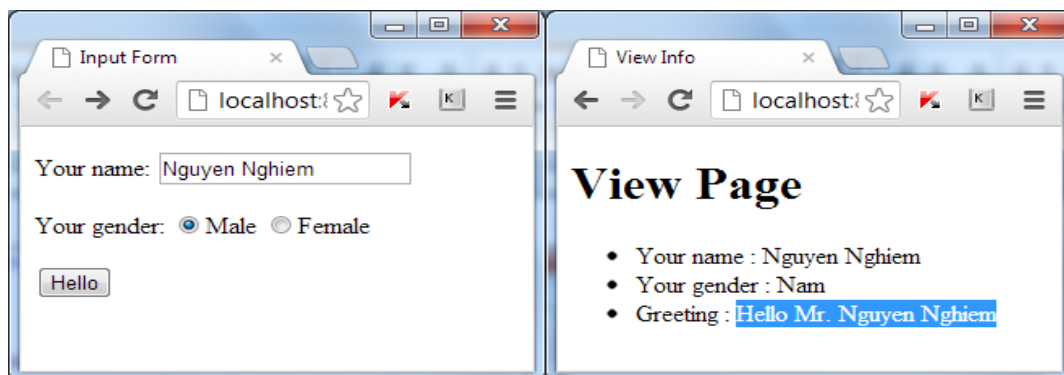
```

InfoModel được viết theo qui ước của JavaBean nghĩa là

- ✓ Có constructor không tham số
- ✓ Các phương thức getter và setter được sử dụng để đọc ghi thuộc tính

Tên thuộc tính được xác định là bỏ get hay set sau đó đổi ký tự đầu của phần còn lại. Ví dụ trong InfoModel chúng ta có 3 thuộc tính là name (từ getName), gender (từ getGender) và greeting (từ getGreeting). Trong đó name và gender là các thuộc tính cho đọc/ghi còn greeting là thuộc tính chỉ đọc (do chỉ cho getGreeting()).

❑ View (Success.jsp)



```

<%@ page contentType="text/html; charset=utf8" pageEncoding="utf8"%>
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf8">
    <title>View Info</title>
</head>
<body>
<h1>View Page</h1>
<ul>
    <li>Your name      : ${info.name}</li>
    <li>Your gender    : ${info.gender}</li>
    <li>Greeting       : ${info.greeting}</li>
</ul>
</body>
</html>

```

Thông tin chia sẻ (info) là đối tượng của InfoModel có 3 thuộc tính name, gender và greeting vì vậy việc truy xuất và hiển thị các thuộc tính rất đơn giản trong JSP 2.0 nhờ sự hỗ trợ của biểu thức EL như ở trên trang Success.jsp.

❑ Ánh xạ servlet

Để có thể khăn cầu được servlet HelloWorld bạn phải thực hiện việc ánh xạ servlet với một cái tên để giao dịch trên web theo 2 cách:

Cách 1: sử dụng annotation. Cách này rất đơn giản nhưng đòi hỏi bạn phải chạy với Servlet 3+, bạn chỉ cần thêm @WebServlet("/tên giao dịch") ngay trên định nghĩa lớp servlet.

```
@WebServlet("/hello.do")
public class HelloWorld extends HttpServlet{...}
```

Cách 2: khai báo trong WEB-INF/web.xml. Cách này phức tạp hơn đôi chút nhưng chạy được trên mọi phiên bản và hết sức mềm dẻo nếu bạn muốn thay đổi cấu hình sau này.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID"
version="3.0">
  <display-name>Java Technology Suite</display-name>
  <!-- Khai báo HelloWorld -->
  <servlet>
    <servlet-name>HelloServlet</servlet-name>
    <servlet-class>nnghiem.servlet.HelloServlet</servlet-class>
  </servlet>
  <!-- Ánh xạ HelloWorld với tên hello.do -->
  <servlet-mapping>
    <servlet-name>HelloServlet</servlet-name>
    <url-pattern>/hello.do</url-pattern>
  </servlet-mapping>
  <!-- Khai báo danh trang web mặc định -->
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

4.2 Servlet

Từ trước cho đến nay, bạn chưa một lần được nhìn thấy một servlet có cấu trúc đầy đủ mà chỉ biết viết servlet là phải kế thừa HttpServlet và cài đặt mã cho phương thức phục vụ (service(), doPost() hay doGet()).

Bạn chưa biết rõ là servlet hoạt động ra sao – được sinh ra, phục vụ và bị giải phóng khi nào. Trong phần này bạn sẽ được trang bị một cách đầy đủ và servlet. Trước tiên bạn cần biết cấu trúc của một Servlet như sau. Trong phần tiếp sẽ làm bạn hiểu rõ hơn về hoạt động của nó.


```
@WebServlet("/<tên giao dịch>")
public class <tên lớp> extends HttpServlet
{
    @Override
    protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        ...code chạy mỗi khi yêu cầu servlet theo hình thức GET...
    }

    @Override
    protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        ...code chạy mỗi khi yêu cầu servlet theo hình thức POST...
    }

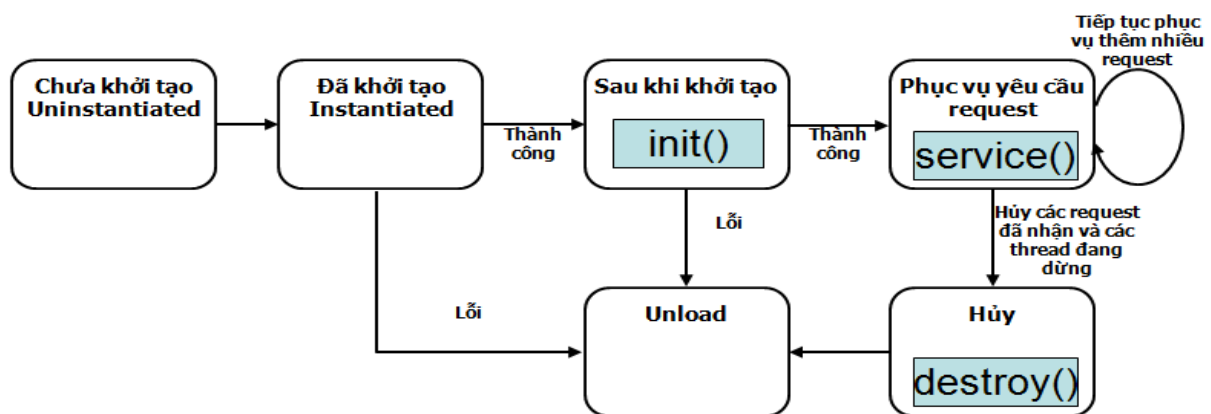
    @Override
    public void init(ServletConfig config) throws ServletException {
        ...code chạy sau khi servlet được tải vào và khởi động...
    }

    @Override
    public void destroy() {
        ...code chạy trước khi servlet bị giải phóng khỏi bộ nhớ...
    }
}
```

4.2.1 Vòng đời

Vòng đời của servlet là quá trình nạp vào bộ nhớ, khởi tạo đối tượng servlet, phục vụ yêu cầu người sử dụng, kết thúc phục vụ và giải phóng khỏi bộ nhớ.

Sau đây là sơ đồ mô tả vòng đời của servlet.



Hình: Vòng đời servlet

- ✓ Uninstantiated: là trạng thái của servlet khi chưa được nạp vào bộ nhớ và khởi tạo
- ✓ Instantiated: là trạng thái sau khi đã nạp vào bộ nhớ và khởi tạo đối tượng. Servlet thường được khởi tạo khi một người sử dụng truy xuất đến mà servlet chưa được khởi tạo.

- ✓ Sau khi khởi tạo thành công, máy servlet (servlet engine) sẽ gọi phương thức `init()` của servlet vừa được khởi tạo.
- ✓ Trong trường hợp không thể khởi tạo hoặc phương thức `init()` thực hiện có lỗi thì servlet sẽ bị giải phóng khỏi bộ nhớ. Ngược lại nếu `init()` thực hiện thành công, máy servlet sẽ tiếp tục gọi phương thức `service` của servlet đang thực hiện một tiểu trình riêng sau đó cư trú lại trong bộ nhớ để sẵn sàng phục vụ yêu cầu khác của những người sử dụng khác.
- ✓ Để tăng tình hiệu quả của sự phục vụ yêu cầu, servlet không bị giải phóng ngay mà cư trú lại để sẵn sàng phục vụ yêu cầu khác. Mỗi yêu cầu được phục vụ trên một tiểu trình riêng và các tiểu trình này có thể thực hiện song song hoặc tuần tự tùy thuộc vào cấu hình của lập trình viên. Vì một lý do nào đó (lâu không truy xuất hay thiếu bộ nhớ...) servlet sẽ bị máy servlet giải phóng. Trước khi giải phóng servlet, máy servlet không quên gọi phương thức `destroy()` của servlet.
- ✓ `init()` và `destroy()` được dành riêng cho lập trình viên viết mã trong việc quản lý vòng đời của servlet. Chúng ta thường lợi dụng vào các sự kiện này để chuẩn bị tài nguyên cho servlet khi được khởi đầu cũng như giải phóng chúng trước khi servlet bị giải phóng.

4.2.1.1 Phương thức `init()`

`Init()` được gọi ngay sau khi khởi tạo cho servlet. Thực hiện duy nhất trong lần gọi đầu tiên và không được thực hiện lại khi có request từ client.

Với cú pháp không có tham số `ServletConfig` tức bạn không cần đọc các tham số khởi đầu từ tập tin cấu hình `web.xml`.

```
public void init() throws ServletException
```

Với cú pháp có tham số `ServletConfig`, bạn có thể đọc các thông tin liên của các tham số khởi đầu dành riêng cho servlet chẳng hạn `username`, `password`, `company` ...

```
public void init(ServletConfig config) throws ServletException
```

4.2.1.2 Phương thức `destroy()`

`Destroy()` được gọi ngay trước khi servlet bị giải phóng (unload) ra khỏi bộ nhớ. Lợi dụng sự kiện này bạn có thể giải phóng các tài nguyên liên quan đến servlet như xóa tập tin rác, đóng kết nối...

```
public void destroy()
```

4.2.1.3 Phương thức `service()`, `doPost()`, `doGet()`

Đây là các phương thức phục vụ yêu cầu. Tức khi có request từ client thì các phương thức này được gọi tùy thuộc vào hình thức đệ trình của client. Nếu bạn override `service` thì `doPost()` và `doGet()` sẽ không có hiệu lực. Phương thức `service` kiểm tra hình thức đệ trình từ client là `GET`, `POST`, `DELETE` ... để thỉnh cầu các gọi phương thức phục vụ tương ứng là `doGet()`, `doPost()` hay `doDelete()`.

Như các bạn đã thấy trong ví dụ servlet, chúng ta override phương thức `doPost()` và `doGet()` để xử lý các hình thức đệ trình tương ứng. Bạn có thể làm tương tự với phương thức

service() (cùng cú pháp) để thực hiện cho tất cả các hình thức đệ trình nếu bạn không muốn phân biệt.

```
public void service(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
```

4.2.1.4 Ứng dụng HitCounter

Ví dụ sau sẽ giúp bạn triển khai ứng dụng đếm số lần truy xuất servlet HitCounterServlet. Để thực hiện điều này một tập tin logs.txt được tạo ra để lưu số đếm và nó sẽ được tải vào ngay sau khi HitCounterServlet được khởi động thành công (viết mã tại **init()**). Sau đó tăng số đếm lên mỗi khi có người truy xuất trang (viết mã tại **doGet()** và **doPost()**) và cuối cùng lưu trở lại vào file ngay trước khi servlet bị giải phóng khỏi bộ nhớ (viết mã tại **destroy()**).

Sau đây là mã nguồn đầy đủ của HitCounterServlet.java

```
@WebServlet("/hitCounter.do")
public class HitCounterServlet extends HttpServlet {
    // chứa số lần truy xuất servlet
    Properties logs = new Properties();

    @Override
    protected void doGet(HttpServletRequest req,
        HttpServletResponse resp) throws ServletException, IOException {
        // gọi lại doPost()
        this.doPost(req, resp);
    }

    @Override
    protected void doPost(HttpServletRequest req,
        HttpServletResponse resp) throws ServletException, IOException {
        // lấy số đếm cũ
        int hitCounter = Integer.parseInt(logs.getProperty("hitCounter"));
        // tăng số đếm và cập nhật
        logs.setProperty("MyCounter", String.valueOf(++hitCounter));

        // chuyển tiếp
        request.getRequestDispatcher("HitCounter.jsp").forward(req, resp);
    }

    @Override
    public void init(ServletConfig config) throws ServletException {
        // đường dẫn vật lý của tập tin chứa số lần truy xuất đặt tại website
        String path = getServletContext().getRealPath("logs.txt");

        try {
            // tải số lần truy xuất từ logs.txt
            logs.load(new FileInputStream(path));
            // chia sẻ để các JSP của thể truy xuất mà hiển thị
            getServletContext().setAttribute("logs", logs);
        }
        catch(IOException ex) {
```

```

        System.out.println("Lỗi tải file logs.txt!");
    }
}

@Override
public void destroy() {
    // đường dẫn vật lý của tập tin chứa số lần truy xuất đặt tại website
    String path = getServletContext().getRealPath("logs.txt");
    try {
        // ghi số đếm vào file HitCounter.txt
        logs.store(new FileOutputStream(path), "Log info");
    }
    catch(IOException ex) {
        System.out.println("Lỗi lưu file logs.txt!");
    }
}
}

```

Chạy và kiểm tra kết quả

- ✓ Chúng ta có thể truy xuất HitCounterServlet thông qua địa chỉ sau (tùy vào cài đặt của bạn): <http://localhost:8080/Test/hitCounter.do>
- ✓ Và kiểm tra kết quả bằng cách kết thúc (shutdown) ứng dụng của bạn và mở tập tin HitCounter.txt đặt tại thư mục của ứng dụng web đang chạy

4.2.2 Tham số

4.2.2.1 Tham số là gì?

Tham số (parameter) là dữ liệu của các trường trên form nhập hoặc chuỗi truy vấn (QueryString – phần sau dấu ? đính kèm theo địa chỉ URL của trang web). Tham số được gửi đến server để xử lý khi người dùng tạo ra một yêu cầu. Để hiểu rõ tham số chúng ta xét trang web có giao diện như sau:



[Xin chào](#)

Hình: Form nhập

Và mã HTML của nó như sau

```

<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>Tham số</title>
</head>

<body>
<form method="post" action="MyServlet1.do">
    <input type="text" name="txtThamSo1" />
    <input type="checkbox" name="chkThamSo2" />
    <select name="cboThamSo3">

```

```

        <option value="A">Mục 1</option>
        <option value="B">Mục 2</option>
    </select>
    <input type="submit" name="btnSubmit" value="Submit" />
</form>
<HR>
<a href="MyServlet2.do?ThamSo5=123&ThamSo6=Hello World">Xin chào</a>
</body>
</html>

```

Chúng ta nhận thấy MyServlet1.do được gọi theo 2 cách - thông qua form và liên kết.

- ✓ Form nhập trên đây chứa 4 tham số, được thiết kế để chuyển các tham số này đến server để xử lý bởi chương trình MyServlet1.do khi click vào nút submit
 - **txtThamSo1**: Ô nhập textfield
 - **chkThamSo2**: hộp chọn checkbox
 - **cboThamSo3**: menu chọn combobox
 - **btnSubmit**: nút submit
- ✓ Liên kết trên được thiết kế để chuyển 2 tham số (**?ThamSo5=123&ThamSo6=Hello World**) đến chương trình server để xử lý bởi chương trình MyServlet2.do khi bạn click vào liên kết Xin chào.

4.2.2.2 Nhận tham số

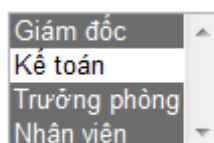
Khi muốn nhận giá trị của tham số gửi đến từ client trong servlet chúng ta sử dụng các phương thức được xây dựng trong HttpServletRequest gồm:

✓ **String getParameter(String name)**

Nhận giá trị của một tham số, nếu tham số không tồn tại sẽ trả về giá trị null. Một checkbox, nhóm radio hay select không được chọn cũng sẽ cho giá trị null (nghĩa là tham số đó không được truyền đi). Chú ý: một ô nhập không nhập vào sẽ cho giá trị rỗng chứ không phải là null.

✓ **String[] getParameterValues(String name)**

Nhận mảng giá trị của tham số. Phương thức này thường được dùng để nhận các tham số có nhiều giá trị như select-multiple



Hình: Listbox

```

<select name="LstChucDanh" multiple size="3">
    <option value="GD">Giám đốc</option>
    <option value="KT">Kế toán</option>
    <option value="TP">Trưởng phòng</option>
    <option value="NV">Nhân viên</option>
</select>

```

hay nhóm các checkbox cùng tên

☒ Đọc sách ☐ Du lịch ☒ Thể thao ☐ Âm nhạc

Hình: Nhóm checkbox

```
<input type="checkbox" name="chkSoThich" />Đọc sách
<input type="checkbox" name="chkSoThich" />Du lịch
<input type="checkbox" name="chkSoThich" />Thể thao
<input type="checkbox" name="chkSoThich" />Âm nhạc
```

✓ Enumeration <String> getParameterNames()

Nhận tất cả các tên tham số gửi đến từ client.

4.2.2.2.1 Tham số đơn giản

Sử dụng phương thức getParameter() để xử lý tham số đơn giản truyền từ client. Trong ví dụ này, bạn sẽ tìm thấy các thành phần sau:

- ✓ ThamSo1.html: form nhập
- ✓ ThamSo1Servlet.java: servlet nhận tham số
- ✓ Web.xml: khai báo và ánh xạ servlet
- ✓ Chạy và phân tích kết quả

ThamSo1.html: chứa form nhập dữ liệu

```
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <title>Tham số</title>
</head>

<body>
<form method="post" action="ThamSo1.do">
  <p>Họ và tên:
  <input type="text" name="txtHoTen" />
  <p>Giới tính:
    <input type="radio" name="rdoGioiTinh" value="Male" /> Nam
    <input type="radio" name="rdoGioiTinh" value="Female" /> Nữ
  <p>Tình trạng hôn nhân:
    <input type="checkbox" name="chkTinhTrang" value="1" /> Còn độc thân?
  <p>Quốc tịch:
  <select name="cboQuocTich">
    <option value="VN">Việt Nam</option>
    <option value="TQ">Trung Quốc</option>
  </select>
  <p>Ghi chú:
  <textarea name="txtGhiChu" rows="3" cols="44"></textarea>
  <p>
  <input type="submit" name="btnXuLy" value="Xử lý tham số đơn giản" />
</form>
</body>
</html>
```

ThamSo1Servlet.java : nhận tham số và xuất ra màn hình

```
@WebServlet("/ThamSo1.do")
```

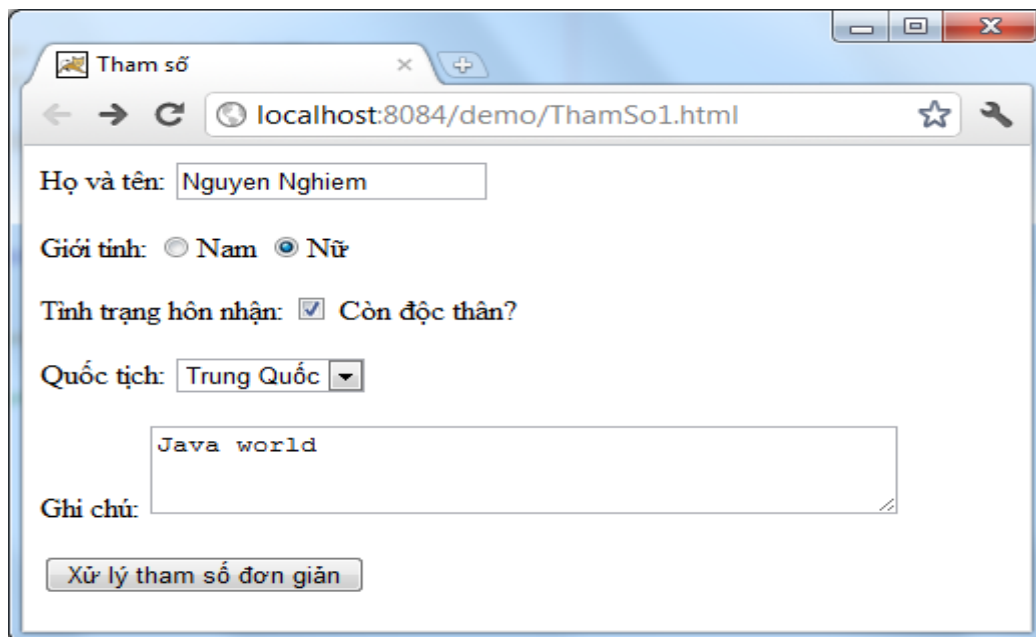
```
public class MyServlet extends HttpServlet
{
    @Override
    protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

        String txtHoTen = request.getParameter("txtHoTen");
        String rdoGioiTinh = request.getParameter("rdoGioiTinh");
        String chkTinhTrang = request.getParameter("chkTinhTrang");
        String cboQuocTich = request.getParameter("cboQuocTich");
        String txtGhiChu = request.getParameter("txtGhiChu");
        String btnXuLy = request.getParameter("btnXuLy");

        if(btnXuLy != null)
        {
            out.printf("<LI>Họ và tên: %s</LI>", txtHoTen);
            out.printf("<LI>Giới tính: %s</LI>", rdoGioiTinh);
            out.printf("<LI>Trình trạng: %s</LI>", chkTinhTrang);
            out.printf("<LI>Quốc tịch: %s</LI>", cboQuocTich);
            out.printf("<LI>Ghi chú: %s</LI>", txtGhiChu);
        }
    }
}
```

Chạy và kiểm tra kết quả

- Chạy ThamSo1.html, nhập đủ thông tin và bấm nút, servlet sẽ chạy và cho kết quả như sau



Tham số

Họ và tên:

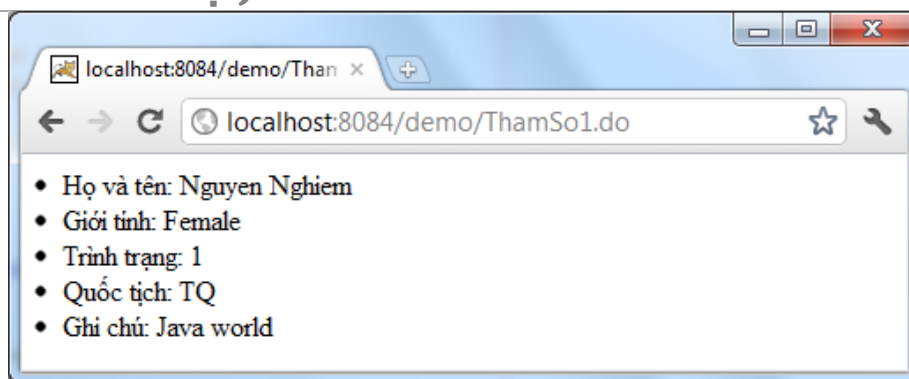
Giới tính: ☐ Nam ☒ Nữ

Tình trạng hôn nhân: ☒ Còn độc thân?

Quốc tịch:

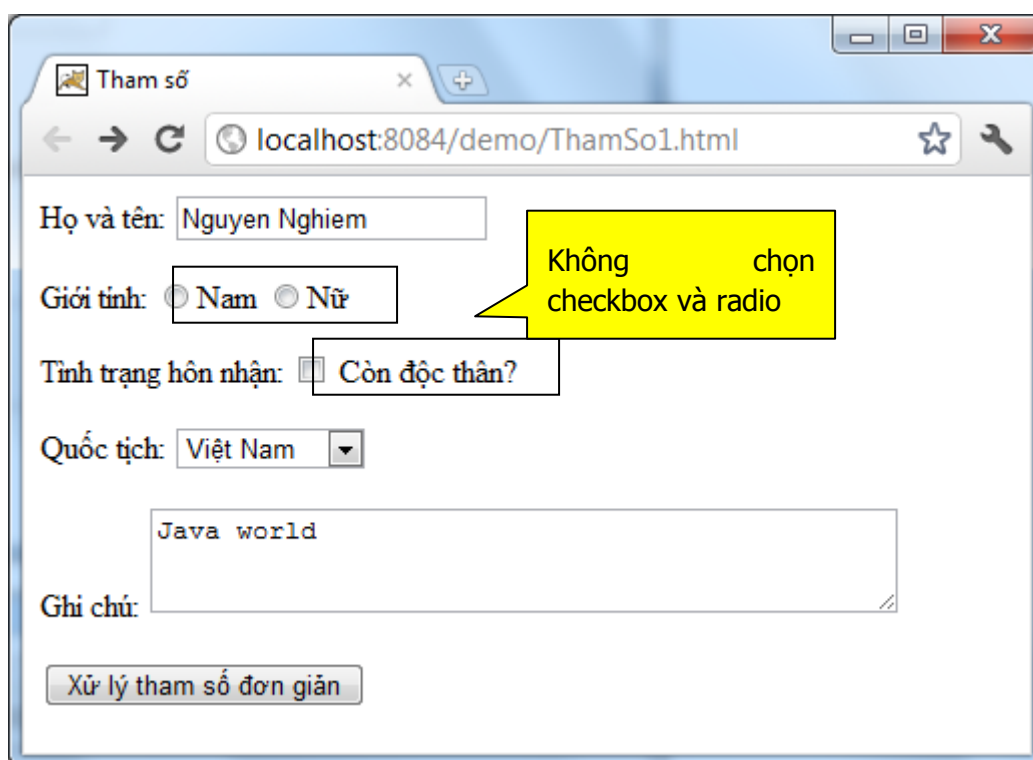
Ghi chú:

Hình: Tham số form



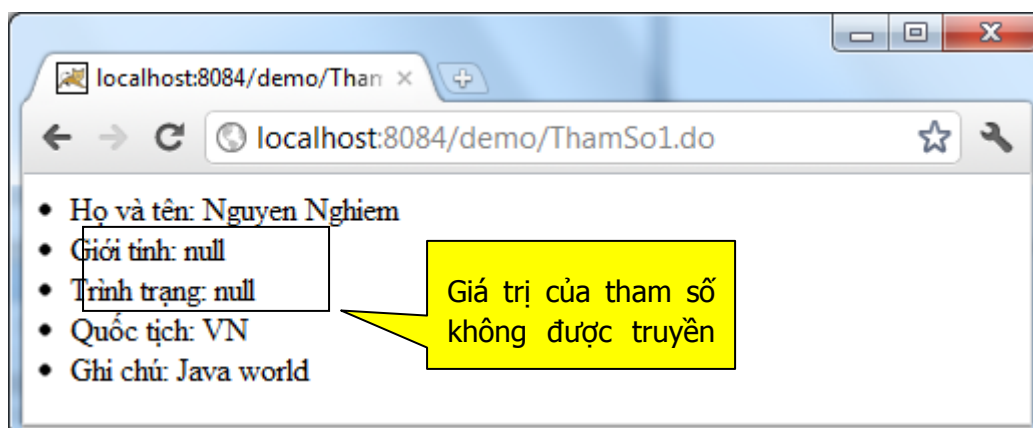
Hình: Hiển thị giá trị tham số

- Chạy ThamSo1.html, không chọn radio và checkbox sau đó bấm nút, servlet sẽ chạy và cho kết quả như sau



Hình: Tham số checkbox và radio

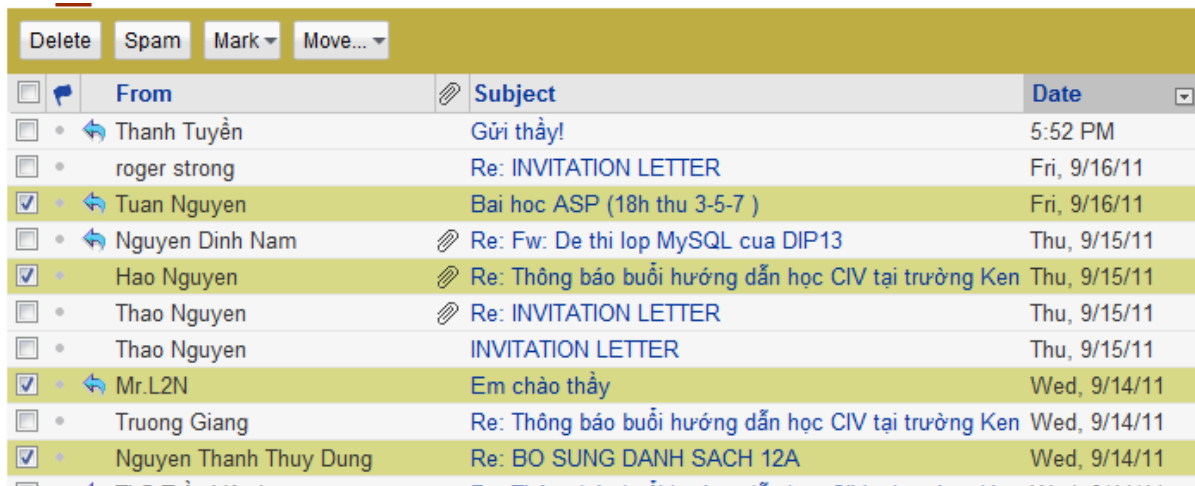
Kết quả sẽ không nhận được giới tính và trình trạng hôn nhân



Hình: Xử lý checkbox và radio

4.2.2.2.2 Tham số nhiều giá trị

Một ví dụ bạn gặp hàng ngày là xóa các email rác trong hộp mail yahoo hay gmail của bạn. Thông thường bạn chọn các email không cần nữa bằng cách tích lên các checkbox sau đó bấm chọn nút Delete. Vấn đề đặt ra ở đây là nếu bạn viết một servlet để xử lý việc xóa đó bạn sẽ phải làm thế nào.



	From	Subject	Date
<input type="checkbox"/>	Thanh Tuyền	Gửi thầy!	5:52 PM
<input type="checkbox"/>	roger strong	Re: INVITATION LETTER	Fri, 9/16/11
<input checked="" type="checkbox"/>	Tuan Nguyen	Bai học ASP (18h thu 3-5-7)	Fri, 9/16/11
<input type="checkbox"/>	Nguyen Dinh Nam	Re: Fw: De thi lop MySQL cua DIP13	Thu, 9/15/11
<input checked="" type="checkbox"/>	Hao Nguyen	Re: Thông báo buổi hướng dẫn học CIV tại trường Ken	Thu, 9/15/11
<input type="checkbox"/>	Thao Nguyen	Re: INVITATION LETTER	Thu, 9/15/11
<input type="checkbox"/>	Thao Nguyen	INVITATION LETTER	Thu, 9/15/11
<input checked="" type="checkbox"/>	Mr.L2N	Em chào thầy	Wed, 9/14/11
<input type="checkbox"/>	Truong Giang	Re: Thông báo buổi hướng dẫn học CIV tại trường Ken	Wed, 9/14/11
<input checked="" type="checkbox"/>	Nguyen Thanh Thuy Dung	Re: BỎ SUNG DANH SÁCH 12A	Wed, 9/14/11

Hình: Nhóm checkbox

Xem mã nguồn của trang thì tất cả các checkbox được đặt cùng một tên là mid (`<input type="checkbox" name="mid" value="<giá trị khác nhau> ">`). Để xóa được các mục chọn, trước hết bạn phải biết người dùng chọn những mục nào. Điều đó thật là dễ đối với servlet, bạn chỉ cần sử dụng phương thức `getParameterValues()` của `HttpServletRequest`.

Ví dụ sau sử dụng phương thức `getParameterValues()` để nhận giá trị của các tham số nhiều giá trị. Qua ví dụ này các bạn sẽ tự tin khi xử lý loại tham số này trong servlet.

- ✓ ThamSo2.html: form nhập
- ✓ ThamSo2Servlet.java: servlet nhận tham số
- ✓ Web.xml: khai báo và ánh xạ servlet
- ✓ Chạy và phân tích kết quả

ThamSo2.html

```
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <title>Tham số</title>
</head>

<body>
<form method="post" action="ThamSo2.do">
  <p>Sở thích:<br>
    <input type="checkbox" name="chkSoThich" value="DS" />Độc sách
    <input type="checkbox" name="chkSoThich" value="DL" />Du lịch
    <input type="checkbox" name="chkSoThich" value="TT" />Thể thao
    <input type="checkbox" name="chkSoThich" value="AL" />Âm nhạc
  <p>Chức danh:<br>
```

```
<select name="lstChucDanh" size="3" multiple>
    <option value="GD">Giám đốc</option>
    <option value="KT">Kế toán</option>
    <option value="TP">Trưởng phòng</option>
    <option value="TB">Trưởng ban</option>
    <option value="NV">Nhân viên</option>
</select>
<p>
    <input type="submit" name="btnXuLy" value="Xử lý tham số nhiều giá trị"/>
</form>
</body>
</html>
```

ThamSo2Servlet.java

```
package nnghiem.servlet;

import java.io.*;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;

@WebServlet("/ThamSo2.do")
public class MyServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

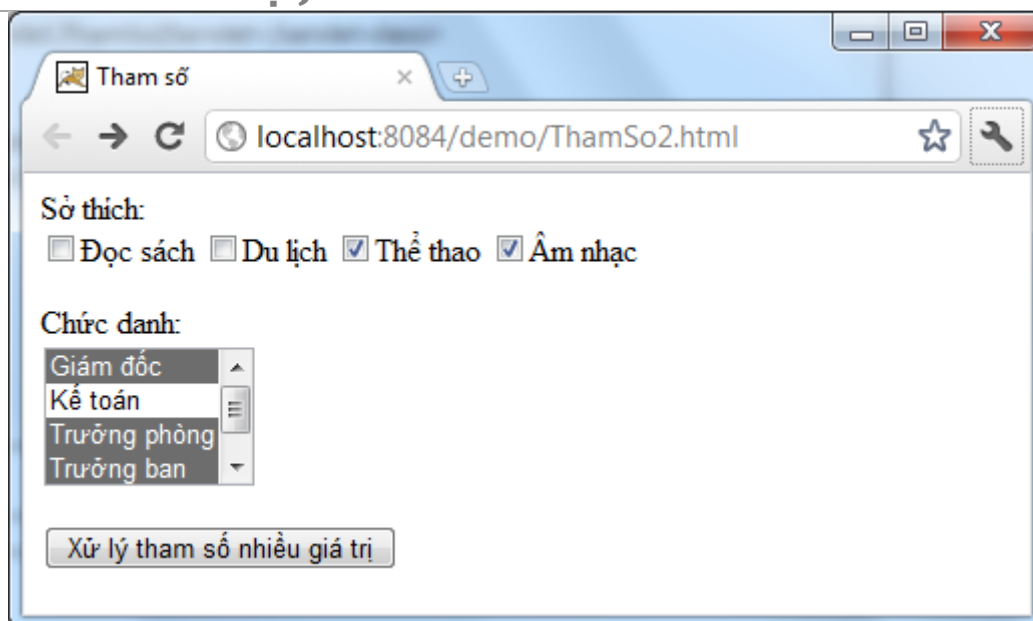
        String[] soThich = request.getParameterValues("chkSoThich");
        String[] chucDanh = request.getParameterValues("lstChucDanh");

        out.printf("<h2>SỞ THÍCH</h1>");
        for(int i=0; soThich!=null && i<soThich.length; i++)
        {
            out.printf("<LI>%s</LI>", soThich[i]);
        }

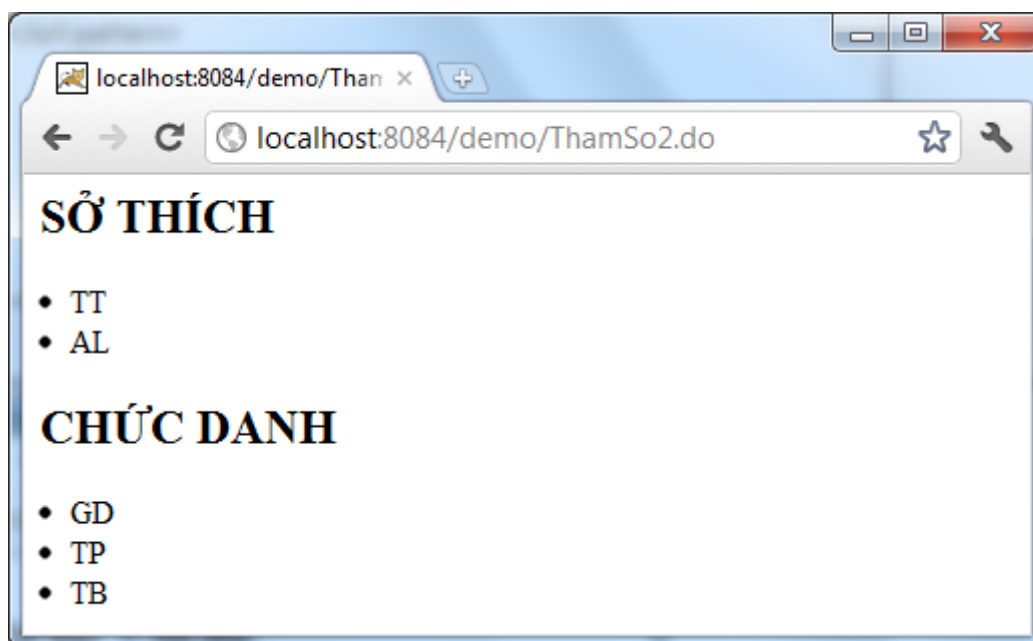
        out.printf("<h2>CHỨC DANH</h1>");
        for(int i=0; chucDanh!=null && i<chucDanh.length; i++)
        {
            out.printf("<LI>%s</LI>", chucDanh[i]);
        }
    }
}
```

Chạy và phân tích kết quả trong 2 trường hợp

- Chọn các mục
- Không chọn mục nào



Hình: form chứa tham số phức tạp



Hình: Kết quả xử lý tham số phức tạp

4.2.2.2.3 Duyệt tất cả tham số

Sử dụng phương thức `getParameterNames()` lấy tất cả danh sách tham số trong request.

- ✓ ThamSo3.html: form nhập chứa nhiều loại tham số
- ✓ ThamSo3Servlet.java: servlet nhận và xuất toàn bộ tham số ra màn hình
- ✓ Web.xml: khai báo và ánh xạ servlet
- ✓ Chạy và phân tích kết quả

ThamSo3.html: form nhập

```
<html>
```

```
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>Tham số</title>
</head>

<body>
<form method="post" action="ThamSo3.do">
    <p>Họ và tên:
    <input type="text" name="txtHoTen" />
    <p>Giới tính:
        <input type="radio" name="rdoGioiTinh" value="Male" /> Nam
        <input type="radio" name="rdoGioiTinh" value="Female" /> Nữ
    <p>Tình trạng hôn nhân:
        <input type="checkbox" name="chkTinhTrang" value="1" /> Còn độc thân?
    <p>Quốc tịch:
    <select name="cboQuocTich">
        <option value="VN">Việt Nam</option>
        <option value="TQ">Trung Quốc</option>
    </select>
    <p>Ghi chú:
    <textarea name="txtGhiChu" rows="3" cols="44"></textarea>
    <p>Sở thích:
        <input type="checkbox" name="chkSoThich" value="DS" /> Đọc sách
        <input type="checkbox" name="chkSoThich" value="DL" /> Du lịch
        <input type="checkbox" name="chkSoThich" value="TT" /> Thể thao
        <input type="checkbox" name="chkSoThich" value="AN" /> Âm nhạc
    <p>Chức danh:
    <select name="lstChucDanh" size="3" multiple>
        <option value="GD">Giám đốc</option>
        <option value="KT">Kế toán</option>
        <option value="TP">Trưởng phòng</option>
        <option value="TB">Trưởng ban</option>
        <option value="NV">Nhân viên</option>
    </select>
    <p>
    <input type="submit" name="btnXuLy" value="Quét tất cả các tham số" />
</form>
</body>
</html>
```

ThamSo3Servlet.java: servlet nhận tham số

```
@WebServlet("/ThamSo3.do")
public class MyServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html; charset=UTF-8");
        PrintWriter out = response.getWriter();

        Enumeration<String> names = request.getParameterNames();
        while(names.hasMoreElements()) {
            String name = names.nextElement();
            String[] values = request.getParameterValues(name);

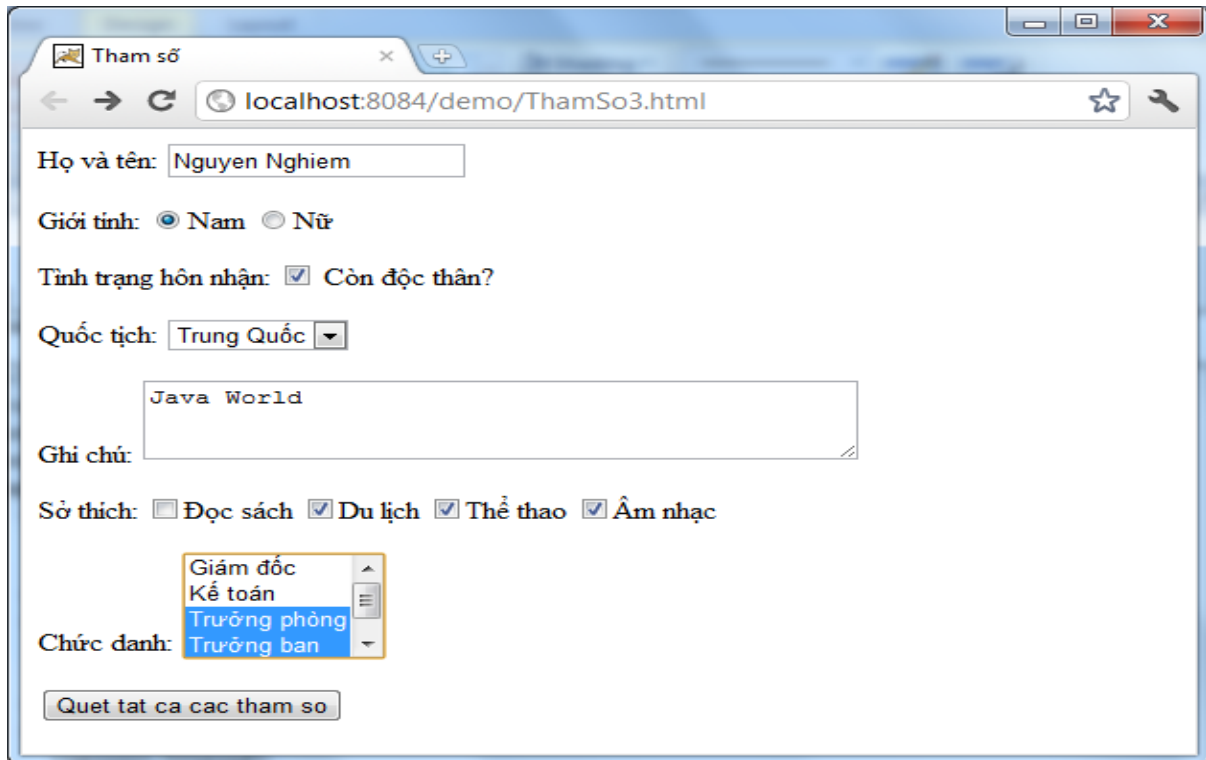
            out.printf("<b>%s</b>", name);
            for(int i=0; values != null && i<values.length; i++) {
                out.printf("<LI>%s</LI>", values[i]);
            }
        }
    }
}
```

```

    }
  }
}

```

Chạy và phân tích kết quả



Tham số

localhost:8084/demo/ThamSo3.html

Họ và tên:

Giới tính: ☒ Nam ☐ Nữ

Tình trạng hôn nhân: ☒ Còn độc thân?

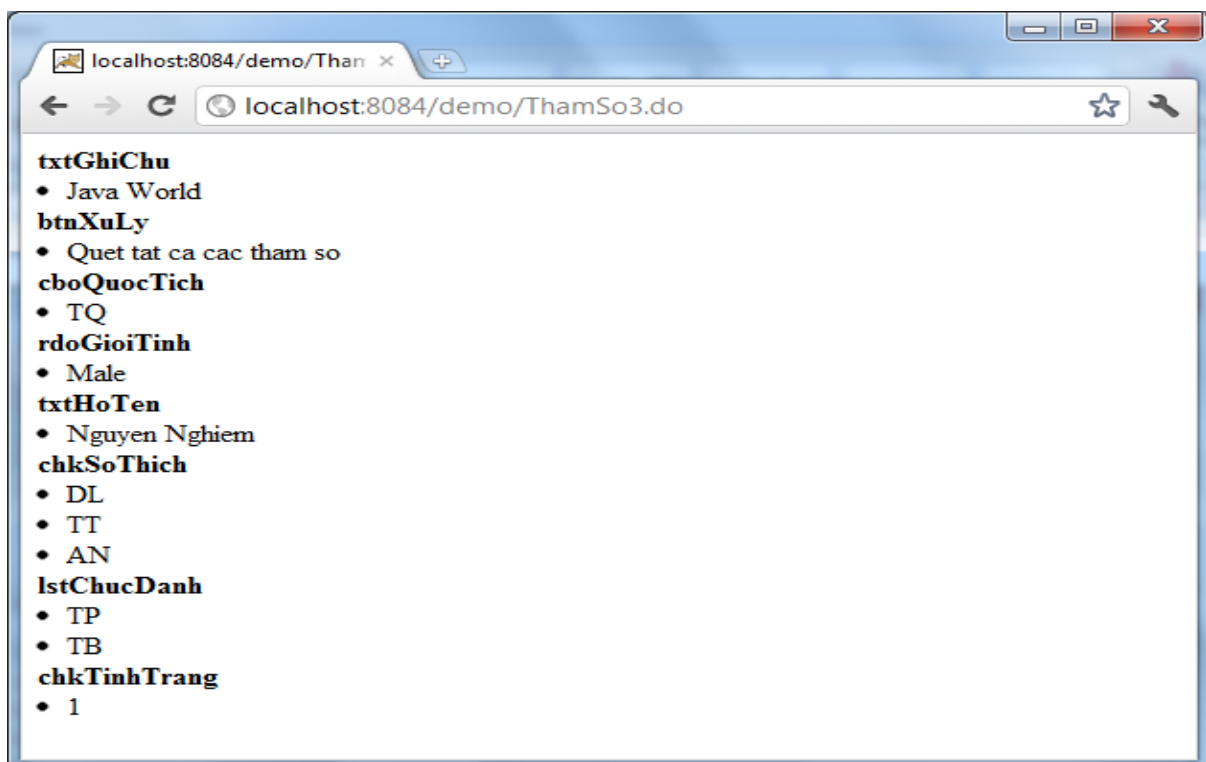
Quốc tịch:

Ghi chú:

Sở thích: ☐ Đọc sách ☒ Du lịch ☒ Thể thao ☒ Âm nhạc

Chức danh:

Hình: form chứa tham số đủ loại



localhost:8084/demo/Tham x

localhost:8084/demo/ThamSo3.do

txtGhiChu

- Java World

btnXuLy

- Quet tat ca cac tham so

cboQuocTich

- TQ

rdoGioiTinh

- Male

txtHoTen

- Nguyen Nghiem

chkSoThich

- DL
- TT
- AN

lstChucDanh

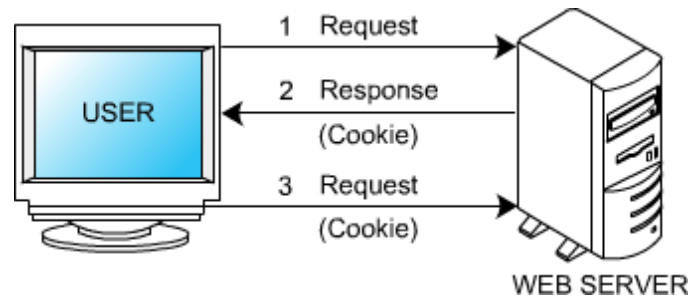
- TP
- TB

chkTinhTrang

- 1

4.2.3 Cookie

Cookies là các thông tin text đơn giản được lưu ngay trên máy của người sử dụng (máy khách). Chúng ta có thể tạo ra cookie bằng cách viết mã JavaScript (chạy ngay trên máy khách) hoặc tạo và gửi về từ servlet. Các request của máy khách sẽ đính kèm cookies phù hợp đến server và chương trình có thể tiếp nhận và xử lý.



Hình: Mô hình truyền nhận cookie

- ✓ Server dùng cookies để sử dụng các thông tin của liên quan đến người sử dụng máy đó.
Ví dụ
 - Ghi nhớ tài khoản đăng nhập
 - Ghi nhớ các mặt hàng, bài báo... mà mình yêu thích
 - Ghi nhớ mã phiên làm việc (session id) hiện tại
- ✓ Lưu số lần truy xuất của máy đó đến website hay từng sản phẩm, bài báo cụ thể
- ✓ Mỗi browser có thể chứa 20 cookies/site và tối đa là 300 cookies.
- ✓ Kích thước tối đa của cookies là 4Kb
- ✓ Ưu điểm
 - Xác nhận user trong suốt quá trình giao dịch điện tử
 - Hỗ trợ việc checkLogin (low security)
 - Hỗ trợ việc cung cấp thông tin ưa thích cho người dùng
 - Hỗ trợ cho việc quảng cáo trên trang web
- ✓ Nhược điểm
 - Xâm phạm tự do riêng tư
 - Browser phải hỗ trợ hay cho phép cookies.
 - Kích thước giới hạn.

4.2.3.1 Tạo và gửi cookie về client

Thông tin của mỗi cookie gồm name (tên), value (giá trị), maxAge (thời gian tồn tại, path (đường dẫn) và domain được phép truyền nhận.

Bước 1: Tạo cookie:

```
Cookie newCookie = new Cookie(name, value);
```

Bước 2: Thiết lập thuộc tính:

```
newCookie.setMaxAge(maxAge);
```

```
newCookie.setPath(path);
```

Bước 3: Gửi cookie về client:

```
response.addCookie(newCookie);
```

Ví dụ sau tạo một cookie có tên là MyCookie và giá trị là "My Cookie Value" và thời gian tồn tại là 1 năm, sau đó gửi cookie này về client để lưu trên máy khách.

```
package nnghiem.servlet;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;

@WebServlet("/createCookie.do")
public class CreateCookieServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {

        //1. Tạo cookie
        String name = "MyCookie";
        String value = "My Cookie Value";
        Cookie newCookie = new Cookie(name, value);

        //2. Thiết lập thuộc tính
        int maxAge = 60*60*24*365; // 1 năm
        newCookie.setMaxAge(maxAge);

        //3. Gửi về client để lưu
        response.addCookie(newCookie);
    }
}
```

4.2.3.2 Nhận cookies từ client

Các cookie được gửi đến từ client ở trong request. Chúng ta có thể nhận nó thông qua phương thức **getCookies()** của request. Phương thức này sẽ cho chúng ta một mảng chứa các cookie. Sau đây là ví dụ nhận và xử lý cookie có tên là MyCookie.

```
@WebServlet("/showCookie.do")
public class ShowCookieServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException{

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        // lấy danh sách cookie gửi đến từ client
        Cookie[] cookies = request.getCookies();
        // tìm và hiển thị thông tin cookie có tên MyCookie
    }
}
```

```

for(int i=0; cookies != null && i<cookies.length; i++) {
    Cookie cookie = cookies[i];
    String name = cookie.getName();
    String value = cookie.getValue();
    if(name.equals("MyCookie")) {
        out.println("<ul>");
        out.println("<li>Name: " + name);
        out.println("<li>Value: " + value);
        out.println("</ul>");
        break;
    }
}
}
}

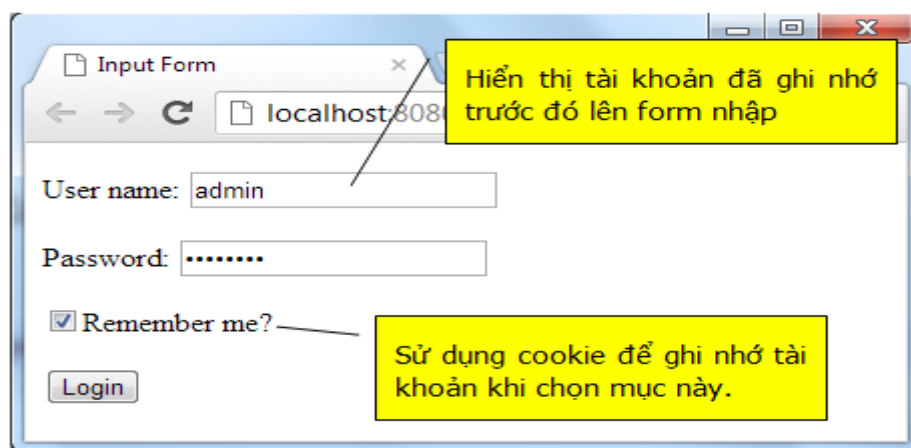
```

Chú ý: cookie mới được tạo ra không thể đọc ngay mà phải đợi đến yêu cầu sau mới có thể đọc và sử dụng nó.

4.2.3.3 Đăng nhập có lưu tài khoản

Ứng dụng đăng nhập có ghi nhớ tài khoản để lần sau khởi phải nhập lại thông tin tài khoản tại một máy cụ thể nào đó không còn xa lạ đối với mỗi người dùng Internet ngày nay. Có lẽ bạn rất muốn biết ứng dụng đó được triển khai như thế nào trong Servlet và JSP đúng không ? Vài đây là thời khắc mà bạn chờ đợi.

Bước 1: giao diện (Login.jsp)



Giao diện đăng nhập có ghi nhớ tài khoản

Mã JSP tương ứng của form

```

<%@ page contentType="text/html; charset=utf8" pageEncoding="utf8"%>
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf8">
    <title>Input Form</title>
</head>
<body>
<label style="color:red">${error}</label>
<form method="post" action="Login.do">
    <p>User name:

```




```

<input name="txtUserName" value="${cookie.ckiId.value}">
<p>Password:
<input name="txtPassword" type="password" value="${cookie.ckiPw.value}">
<p>
<input name="chkRemember" type="checkbox">Remember me?
<p>
<input type="submit" name="btnLogin" value="Login">
</form>
</body>
</html>

```

Tại các điểm nổi bật trên trang JSP được sử dụng để hiển thị thông tin của tài khoản đã ghi nhớ trước đó lên trên các ô nhập để người dùng không phải nhập lại tài khoản của mình.

Bạn cũng thấy action của form đăng nhập là login.do. Và mã nguồn của servlet này như sau.

Bước 2: mã điều khiển (LoginServlet)

```

package nnghiem.servlet;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;

@WebServlet("/login.do")
public class LoginServlet extends HttpServlet{

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        req.getRequestDispatcher("Login.jsp").forward(req, resp);
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        //1. Nhận tham số
        String uid = req.getParameter("txtUserName");
        String pwd = req.getParameter("txtPassword");
        String rem = req.getParameter("chkRemember");

        //2. Kiểm tra sự tồn tại
        if(uid.equalsIgnoreCase("admin") && pwd.length() > 6){

            // tạo các cookie
            Cookie ckiId = new Cookie("ckiId", uid);
            Cookie ckiPw = new Cookie("ckiPw", pwd);
            // thiết lập thuộc tính
            if(rem != null){ // có chọn -> ghi nhớ cookie trong 1 tháng
                ckiId.setMaxAge(30 * 24 * 60 * 60);
                ckiPw.setMaxAge(30 * 24 * 60 * 60);
            }
            else{ //không chọn -> xóa cookie

```

```
        ckiId.setMaxAge(0);
        ckiPw.setMaxAge(0);
    }
    // gửi về client
    resp.addCookie(ckiId);
    resp.addCookie(ckiPw);

    //3. Chia sẻ thông báo lỗi với View
    req.setAttribute("error", "Đăng nhập thành công!");
}
else{
    //3. Chia sẻ thông báo lỗi với View
    req.setAttribute("error", "Sai thông tin đăng nhập!");
}
//4. Chuyển tiếp đến View (Login.jsp) để hiển thị thông báo
req.getRequestDispatcher("Login.jsp").forward(req, resp);
}
```

Bạn chú ý điều kiện đăng nhập hợp lệ trong trường hợp này là UserName=admin và Password có số ký tự lớn hơn 6.

Bạn cần tập trung vào những vùng nổi bật trong mã nguồn và ghi chú đính kèm từng đoạn mã lệnh để hiểu hơn.

4.2.4 Header

Header được sử dụng để gửi các thông tin điều khiển về trình duyệt để điều khiển nó một số hình thức hoạt động nào đó.

- ✓ Refresh trang định kỳ
- ✓ Không cho cache trang
- ✓ Gửi dữ liệu nén
- ✓ Kiểu dữ liệu được gửi về

Sau đây là các ứng dụng hữu ích của các response header.

4.2.4.1 Refresh & Cache-Control

Để thực hiện công việc này, bạn chỉ cần gửi thời lượng làm tươi trang web sử dụng header **Refresh** và đặt giá trị cho header **Cache-Control** để báo cho trình duyệt biết là không lưu trang này vào history (lịch sử) của nó. Với cách làm này thì sau một thời gian định kỳ (60 giây) trang web sẽ được làm tươi.

```
package nngiem.servlet;

import java.io.*;
import java.util.*;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;

@WebServlet("/refresh.do")
```

```
public class RefreshServlet extends HttpServlet
{
    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        resp.setContentType("text/html");
        PrintWriter out = resp.getWriter();
        out.printf("<H2>THOI GIAN HIEN TAI TREN SERVER</H2>");
        out.printf("<B>%s</B>", new Date());

        // yêu cầu trình duyệt refresh trang web sau 60 giây
        resp.addIntHeader("Refresh", 60);

        // yêu cầu trang web không xếp trang này vào history
        resp.addHeader("Cache-Control", "no-cache");
    }
}
```

Kết quả thực hiện



Hình: Điều khiển trình duyệt bằng Response Header (Refresh)

Trang web trên khi chạy sẽ hiển thị thời gian của server. Thời gian này được cập nhật 60 giây một lần.

4.2.4.2 Gửi dữ liệu nén

Dữ liệu sẽ được nén trước khi gửi về client sẽ làm giảm lượng dữ liệu truyền thông và do đó sẽ tăng tốc độ truy cập. Bạn chỉ cần điều khiển header **Content-Encoding** có giá trị là gzip khi đó dữ liệu sẽ được nén trước khi gửi về cho client.

```
package nnghiem.servlet;

import java.io.*;
import java.util.Enumeraion;
import java.util.zip.GZIPOutputStream;
import javax.servlet.ServletException;
import javax.servlet.http.*;

public class GZipServlet extends HttpServlet {
    @Override
```

```
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
    resp.setContentType("text/html");

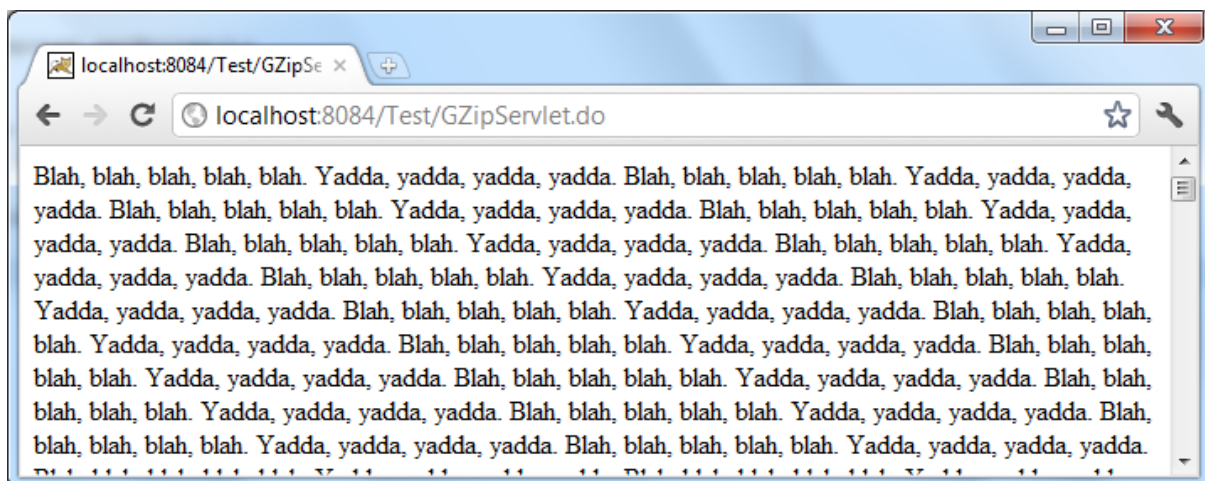
    // Báo để trình duyệt biết dữ liệu gửi về được nén dạng gzip
    resp.setHeader("Content-Encoding", "gzip");

    OutputStream os = resp.getOutputStream();
    PrintWriter gout = new PrintWriter(new GZIPOutputStream(os), false);
    // Xuất một lượng dữ liệu cực lớn về client
    String line = "Blah, blah, blah, blah, blah. Yadda, yadda.";
    for(int i=0; i<10000; i++) {
        gout.println(line);
    }

    // đóng để tổng dữ liệu về client
    outg.close();
}
}
```

Kết quả thực thi

- ✓ Không nén (28.8K modem): > 50 seconds
- ✓ Nén (28.8K modem): < 5 seconds



Hình: nhân dữ liệu nén

4.2.4.3 Gửi các dạng dữ liệu khác

Bạn có thể gửi các dạng dữ liệu khác nhau về client bằng cách điều khiển các header **Content-Disposition** và **Content-Type**. Đoạn mã java sau giúp bạn có thể gửi bất kỳ loại dữ liệu nào về client

```
// chỉ định kiểu dữ liệu được gửi về client
resp.setContentType(mime);
// chỉ định kiểu gửi dạng đính kèm (download) với tên file cụ thể
resp.setHeader("Content-Disposition", "attachment; filename=<tên tập tin>");
```

```
// Gửi dữ liệu hình ảnh về
```

```
resp.getOutputStream().write(data);
```

Sau đây là một số MIME (loại tập tin) chúng ta thường gặp

File	MIME type	Extension
XML	text/xml	.xml
HTML	text/html	.html
Plaintext file	text/plain	.txt
PDF	application/pdf	.pdf
Graphics Interchange Format (GIF) image	image/gif	.gif
JPEG image	image/jpeg	.jpeg
PNG image	image/x-png	.png
MP3 music file	audio/mpeg	.mp3
Shockwave Flash animation	application/futuresplash or application/x-shockwave-flash	.swf
Microsoft Word document	application/msword	.doc
Excel worksheet	application/vnd.ms-excel	.xls
PowerPoint document	application/vnd.ms-powerpoint	.ppt

4.2.4.3.1 Gửi hình về client

Ví dụ sau đây cho phép bạn gửi hình ảnh (gif hoặc jpeg) về client.

```
package nnghiem.servlet;

import java.io.*;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;

@WebServlet("/image.gif")
public class ImageServlet extends HttpServlet
{
    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        String name = "Dragon.gif";
        // lấy đường dẫn vật lý của tập tin "Dragon.gif"
        String path = getServletContext().getRealPath("/res/" + name);

        // Đọc nội dung tập tin hình ảnh vào mảng data
        FileInputStream fis = new FileInputStream(path);
        byte[] data = new byte[fis.available()];
```

```

fis.read(data);
fis.close();

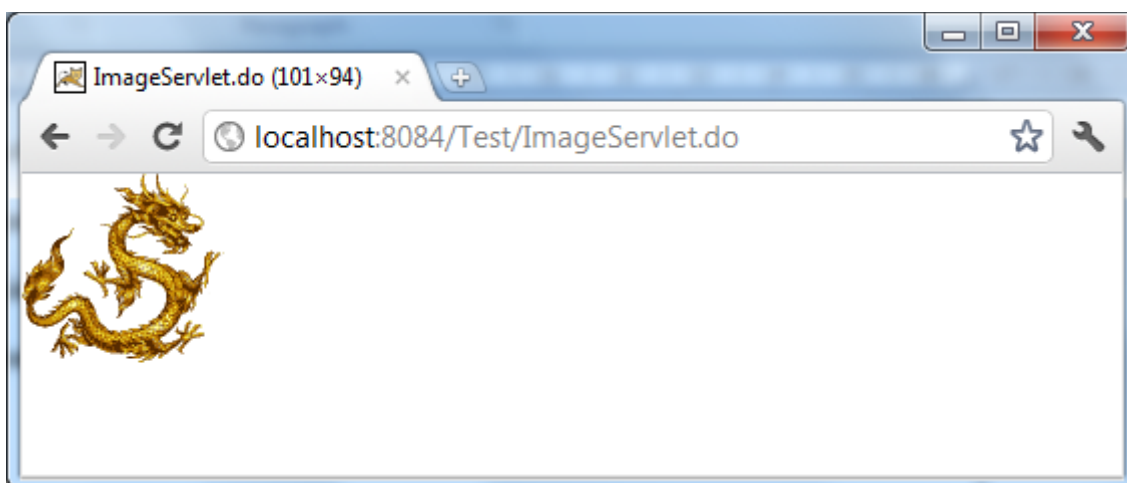
// Chỉ định kiểu dữ liệu trả về
resp.setContentType("image/gif");

// download với tên file trong biến "name"
resp.setHeader("Content-Disposition", "attachment; filename="+name);

// Gửi dữ liệu hình ảnh về
resp.getOutputStream().write(data);
}
}

```

Kết quả thực thi ta nhận được một hình ảnh được sinh từ servlet



Hình: Trả dữ liệu hình từ servlet

4.2.4.3.2 Gửi captcha

Ví dụ sau cho phép bạn gửi captcha về client. Bạn có thể mở rộng ví dụ này để vẽ các hình ảnh dựa vào dữ liệu trong CSDL như các biểu đồ thống kê số liệu dạng cột, bánh...

```

package nnghiem.servlet;

import ...

@WebServlet("/captcha.jpg")
public class CaptchaImageServlet extends HttpServlet {
    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        HttpSession session = req.getSession();

        // Sinh số ngẫu nhiên từ 10000 đến 99999 và lưu vào session
        long captcha = 10000 + Math.round(89999 * Math.random());
        session.setAttribute("captcha", captcha);

        // tạo ảnh ảo
        int w = 100, h = 40;
        BufferedImage image =

```

```

        new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);
// lấy môi trường đồ họa của ảnh để chuẩn bị vẽ lên ảnh
Graphics2D g = image.createGraphics();
// đặt màu nền cho ảnh
g.setColor(Color.WHITE);
g.fillRect(0, 0, w, h);

// đặt màu và font chữ và vẽ chuỗi captcha
g.setColor(Color.BLUE);
g.setFont(Font.decode("Impact-30"));
g.drawString(String.valueOf(captcha), 10, h - 5);

// gửi ảnh về client
resp.setContentType("image/jpeg");
ImageIO.write(image, "jpeg", resp.getOutputStream());
g.dispose();
    }
}

```

Để sử dụng captcha này bạn chỉ cần nhúng thẻ `` vào trang web nào bạn cần. Sau khi người sử dụng nhập mã vài ô nhập bạn chỉ so sánh nó với mã đã sinh ra đặt trong session có tên là captcha.

4.2.4.3.3 Gửi MS Word về client

Chương trình sau sẽ sinh dữ liệu dạng MS Word từ servlet. Để đơn giản, bài này đọc dữ liệu từ 1 tập tin msword, bạn có thể đọc nội dung msword từ CSDL.

```

package nnghiem.servlet;

import ...

@WebServlet("/word.do")
public class MSWordServlet extends HttpServlet{
    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        String name = "jsp.doc";
// Lấy đường dẫn vật lý của tập tin jsp.doc
        String path = getServletContext().getRealPath("/res/" + name);

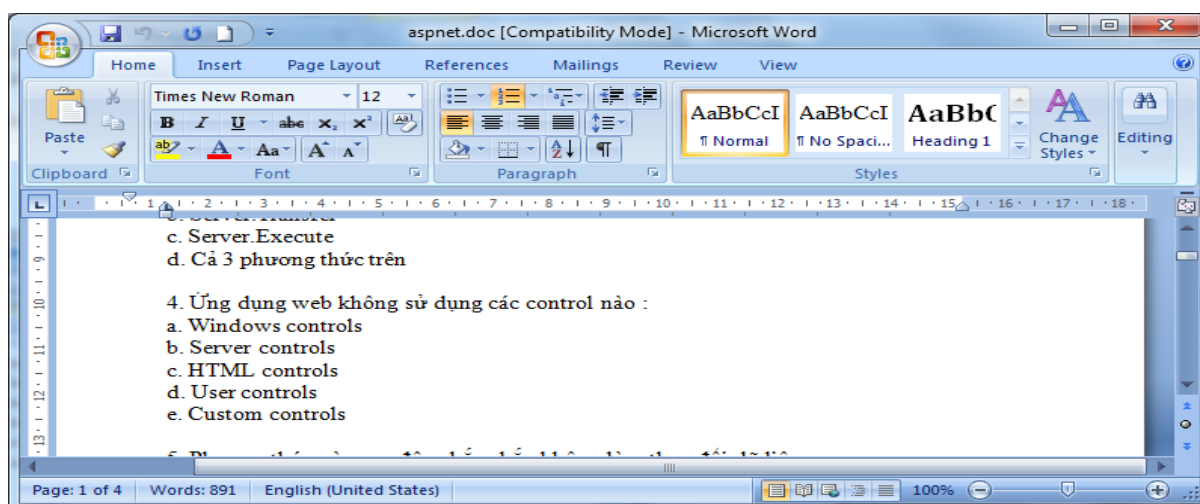
// Đọc nội dung file vào mảng data[]
        FileInputStream fis = new FileInputStream(path);
        byte[] data = new byte[fis.available()];
        fis.read(data);
        fis.close();

// Chỉ định kiểu dữ liệu và gửi về client
        resp.setContentType("application/msword");

// download với tên file "jsp.doc"
        resp.setHeader("Content-Disposition", "attachment;filename=jsp.doc");
        resp.getOutputStream().write(data);
    }
}

```

Kết quả chạy <http://localhost:8080/Test/word.do> với IE



Hình: Trả MS Word từ servlet

4.2.4.3.4 Sinh bảng tính excel

Servlet sinh bảng tính Excel sau đây có 3 đặc điểm cần lưu ý sau đây

- ✓ Sử dụng **response.setContentType("application/vnd.ms-excel")** để đặt kiểu tài liệu excel
- ✓ Sử dụng dấu tab (\t) để chuyển đến ô tiếp theo trên cùng hàng của bảng tính
- ✓ Sử dụng các hàng của excel (=SUM(B2:E2)) như thường dùng

```
package nhghiem.servlet;
```

```
import java.io.*;
```

```
import javax.servlet.ServletException;
```

```
import javax.servlet.annotation.WebServlet;
```

```
import javax.servlet.http.*;
```

```
@WebServlet("/excel.xls")
```

```
public class MExcelServlet extends HttpServlet {
```

```
    @Override
```

```
    public void service(HttpServletRequest req, HttpServletResponse resp)
```

```
        throws ServletException, IOException {
```

```
            resp.setContentType("application/vnd.ms-excel");
```

```
            PrintWriter out = resp.getWriter();
```

```
            out.println("\tQ1\tQ2\tQ3\tQ4\tTotal");
```

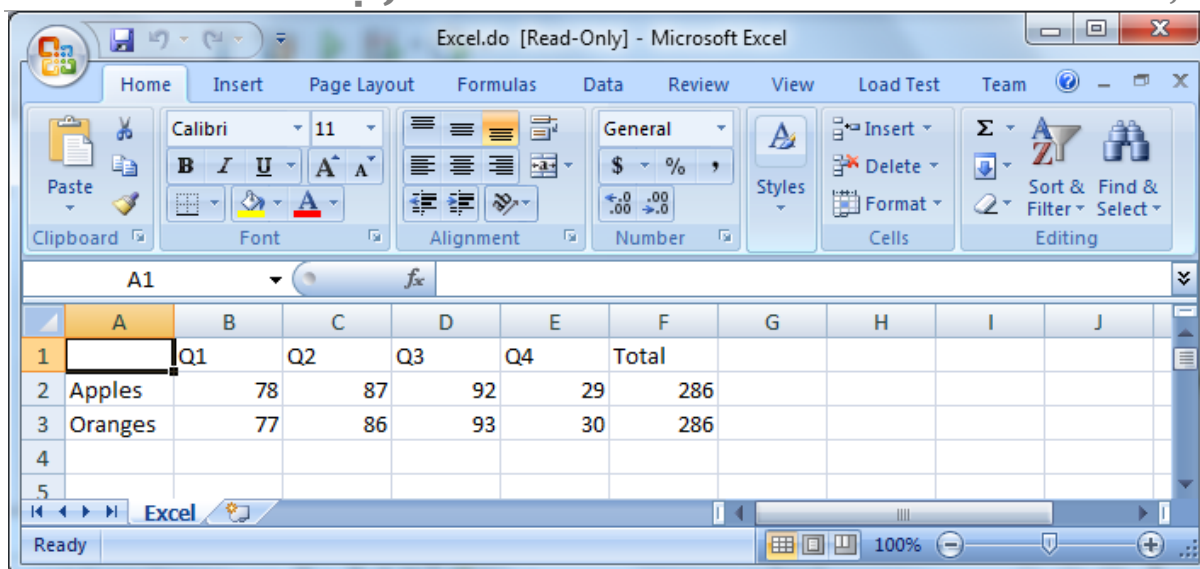
```
            out.println("Apples\t78\t87\t92\t29\t=SUM(B2:E2)");
```

```
            out.println("Oranges\t77\t86\t93\t30\t=SUM(B3:E3)");
```

```
        }
```

```
}
```

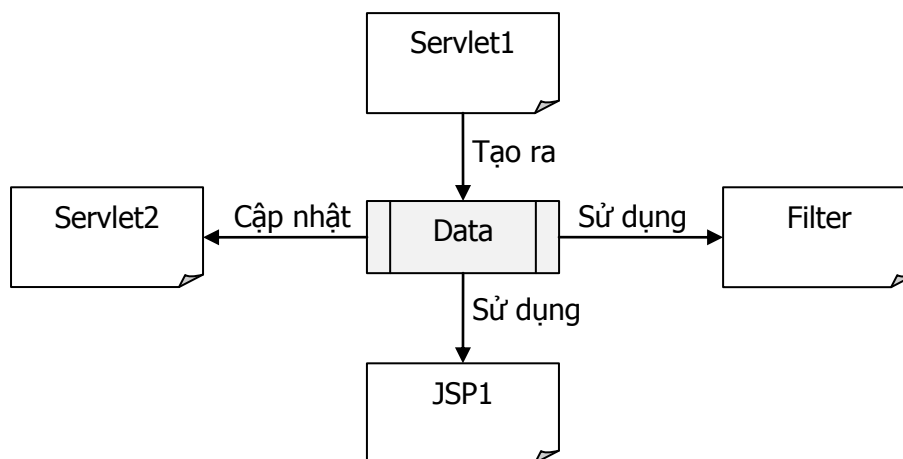
Chạy và phân tích kết quả: <http://localhost:8084/demo/excel.xls>



	A	B	C	D	E	F	G	H	I	J
1		Q1	Q2	Q3	Q4	Total				
2	Apples	78	87	92	29	286				
3	Oranges	77	86	93	30	286				
4										
5										

4.2.5 Chia sẻ dữ liệu

Chia sẻ dữ liệu là việc tạo dữ liệu ở thành phần web (servlet, jsp, filter, listener...) này có thể được sử dụng hoặc cập nhật ở một thành phần web khác. Vì vậy nắm vững kỹ thuật chia sẻ dữ liệu trong lập trình web với java sẽ giúp bạn tổ chức và phối hợp tốt giữa các thành phần web trong ứng dụng web của bạn.



Hình: Chia sẻ dữ liệu giữa các thành phần web

Một ví dụ chia sẻ dữ liệu giữa servlet và JSP mà ta thường dùng là servlet sử dụng lệnh `request.setAttribute("tên", "giá trị")` để chia sẻ dữ liệu với JSP trên phạm vi request, sau đó JSP sử dụng chúng trong việc xây dựng giao diện `${tên}`.

Việc chia sẻ dữ liệu trong lập trình web với Java không chỉ có thế mà còn nhiều phạm vi chia sẻ khác. Cụ thể trong servlet có 3 phạm vi chia sẻ dữ liệu là request, session và application.

- ✓ Phạm vi request: dành cho các thành phần web hoạt động trên cùng một yêu cầu có thể chia sẻ dữ liệu với nhau. Dữ liệu chia sẻ sẽ bị giải phóng ngay sau khi yêu cầu kết thúc. Tất cả các servlet/jsp được include hoặc forward bởi một servlet/jsp khác sẽ chia sẻ với nhau trên phạm vi request.

Trong servlet và filter, request là tham số của các phương thức dịch vụ (service, doPost hay doGet). Vì vậy bạn đã có sẵn mà không cần phải tạo ra nó.

- ✓ Phạm vi session: dành cho các thành phần web được truy xuất trong cùng một phiên làm việc, tức là các thành phần web được yêu cầu từ khi mở trình duyệt cho đến khi đóng trình duyệt hoặc quá hạn sử dụng (session timeout).

Đối tượng session trong servlet được tạo ra từ việc gọi phương thức getSession() của đối tượng request.

HttpSession session = request.getSession();

- ✓ Phạm vi application: dành cho tất cả các thành phần web của ứng dụng được yêu cầu trong suốt quá trình từ lúc ứng dụng bắt đầu cho đến khi ứng dụng kết thúc.

Trong servlet bạn có thể tham chiếu đến đối tượng application bằng việc gọi phương thức getServletContext().

ServletContext application = getServletContext();

Trong JSP request, session và application là đối tượng ngầm định (được định nghĩa sẵn) vì vậy bạn cứ thể dùng mà không cần phải tạo dựng.

Tất cả các phạm vi chia sẻ (request kiểu HttpServletRequest, session kiểu HttpSession và application kiểu ServletContext) đều có các phương thức làm việc với thuộc tính (dữ liệu chia sẻ) giống nhau.

- ✓ void setAttribute(String name, Object value): thiết lập giá trị thuộc tính trong scope. Tạo mới nếu chưa tồn tại thuộc tính và ghi đè nếu thuộc tính đã tồn tại.

Ví dụ:

```
request.setAttribute("HoTen", "Nguyễn Nghiêm");  
request.getServletContext().setAttribute("visitors", 2012);
```

- ✓ Object getAttribute(String name): lấy giá trị thuộc tính. Nếu chưa có trong scope thì trả về null.

Ví dụ:

```
String s = request.getAttribute("HoTen");  
int visitors = (Integer) getServletContext().getAttribute("visitors");
```

- ✓ void removeAttribute(String name): xóa thuộc tính chia sẻ khỏi scope.

Ví dụ:

```
Request.removeAttribute("HoTen");
```

- ✓ Enumeration<String> getAttributeNames(): lấy danh sách tất cả các thuộc tính trong scope

Ví dụ:

```
HttpSession session = request.getSession();  
Enumeration<String> names = session.getAttributeNames();
```

```
while(names.hasMoreElements()){  
    String name = names.nextElement();  
    Object value = session.getAttribute(name);  
}
```

4.3 JSP

4.3.1 Các thành phần cú pháp

4.3.2 Chỉ thị

Trong jsp có 3 loại chỉ thị thường sử dụng là @page, @taglib và @include:

- ✓ Chỉ thị page <%@page...%> được sử dụng để khai báo cho trang jsp một số thông số cho trang jsp của bạn.

Ví dụ sau sẽ cho biết trang web sản sinh mã HTML với chế độ mã hóa ký tự là UTF-8, tức có thể hiển thị tiếng Việt.

```
<%@ page contentType="text/html; charset=utf-8" pageEncoding="utf-8"%>
```

- ✓ Chỉ thị taglib <%@taglib...%> được sử dụng để khai báo thư viện thẻ mà bạn sử dụng trong trang jsp.

Ví dụ sau cho biết trang jsp của bạn có sử dụng thư viện thẻ (phần lỗi, định dạng và thư viện hàm cho EL) của JSTL

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

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

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

- ✓ Chỉ thị include <%@include%> sẽ bao hàm nội dung một file đặt tại vị trí của chỉ thị
- ```
<%@ include file="module.jsp"%>
```

### 4.3.3 Mã kịch bản

Mã java có thể được viết trên trang JSP theo một số cách tùy thuộc vào nhu cầu của bạn.

#### 4.3.3.1 Scriptlet

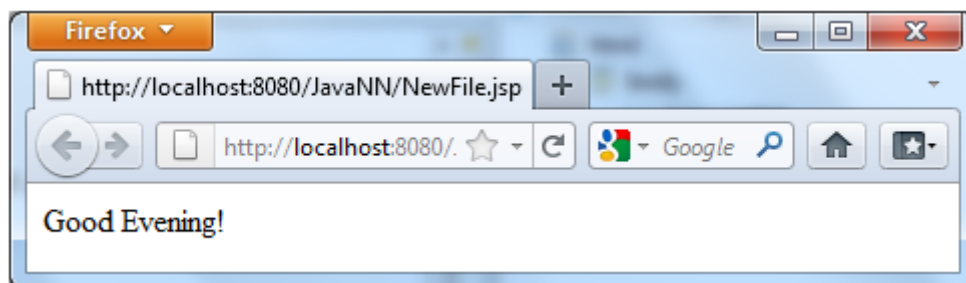
Scriptlet là cách viết mã java thường được sử dụng nhất. Bạn viết mã java đặt giữa cặp dấu <%...java...%>. Mã java đặt ở đây sẽ như đang viết trong phương thức dịch vụ của servlet.

Ví dụ Greeting.jsp sau đây sẽ hiển thị lời chào tiếng anh phù hợp với thời điểm trong ngày

```
<%@ page contentType="text/html; charset=utf-8" pageEncoding="utf-8"%>
<!DOCTYPE html>
<html>
<head>
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
 <title>Học JSP</title>
</head>
```

```
<body>
Good
<%
 java.util.Calendar currTime = new java.util.GregorianCalendar();

 if (currTime.get(currTime.HOUR_OF_DAY) < 12) {
%>
 Morning!
<%
 }
 else if (currTime.get(currTime.HOUR_OF_DAY) < 18) {
%>
 Afternoon!
<%
 }
 else {
%>
 Evening!
<%
 }
%>
</body>
</html>
```



Kết quả truy xuất trang vào buổi tối

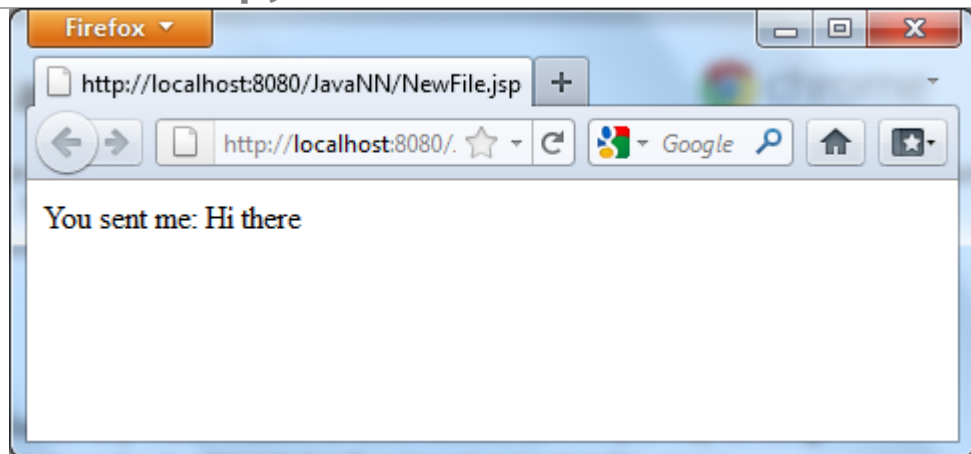
#### 4.3.3.2 Khai báo thành viên

Trong JSP bạn cũng có thể khai báo biến và hàm thành viên. Bạn sử dụng `<%!...khai báo...%>` để khai báo các thành viên và sử dụng trong trang jsp.

Ví dụ: DeclareMethod.jsp

```
<%@ page contentType="text/html; charset=utf-8" pageEncoding="utf-8"%>
<html>
<body>
<%!
 public String myMethod(String someParameter) {
 return "You sent me: " + someParameter;
 }
%>
<%= myMethod("Hi there") %>
</body>
</html>
```

Hình sau cho thấy kết quả hiển thị của ví dụ trên



Kết quả hiển thị của DeclareMethod.jsp

### 4.3.3.3 Biểu thức

Biểu thức trong JSP được viết `<%=biểu thức%>` để thu gọn cho cách viết `<%out.print(biểu thức)%>`, tức là để tính và xuất giá trị của biểu thức.

Sau đây là ví dụ hoàn chỉnh được sử dụng từ phần scriptlet

Mã nguồn cho Greeting1.jsp

```
<%@ page contentType="text/html; charset=utf-8" pageEncoding="utf-8"%>
<html>
<body>
<%
 java.util.Calendar currTime = new java.util.GregorianCalendar();
 String timeOfDay = "";

 if (currTime.get(currTime.HOUR_OF_DAY) < 12) {
 timeOfDay = "Morning!";
 }
 else if (currTime.get(currTime.HOUR_OF_DAY) < 18) {
 timeOfDay = "Afternoon!";
 }
 else {
 timeOfDay = "Evening!";
 }
%>
Good <%=timeOfDay%>
</body>
</html>
```

### 4.3.3.4 Ghi chú

Một số cách ghi chú thường dùng trên trang JSP

Ghi chú HTML

```
<-- Đây là ghi chú HTML. -->
```

Ghi chú JSP - 1 dòng

```
<% / / Đây là một dòng ghi chú JSP. %>
```

Ghi chú JSP - N dòng

```
<% / *%>
```

Ghi chú nhiều dòng đặt ở đây

```
<% * /%>
```

Ghi chú JSP - N dòng

```
<% -- Ghi chú này sẽ không xuất hiện trong mã nguồn servlet. --%>
```

Ghi chú JSP - N dòng

```
<% --
```

```
 <%
```

```
 out.println ("Bạn sẽ không bao giờ nhìn thấy điều này.");
```

```
 %>
```

```
--%>
```

### 4.3.4 Các đối tượng ngầm định

Các đối tượng ngầm định là các đối tượng được định nghĩa sẵn để JSP sử dụng mà không cần khai báo hay tạo dựng ra chúng. Để thuận tiện cho người lập trình JSP, JSP engine cung cấp một danh sách các đối tượng ngầm định thường dùng như sau

Đối tượng	Lớp thực hiện	Mô tả
request	HttpServletRequest	Là request trong servlet
response	HttpServletResponse	Là response trong servlet
out	JspWriter	Giống response.getWriter() của servlet
session	HttpSession	request.getSession() trong servlet
application	ServletContext	getServletContext() trong servlet
config	ServletConfig	getServletConfig() trong servlet
pageContext	PageContext	Container của các đối tượng implicit object và phạm vi chia sẻ page
page	Object	Đối tượng this (trang hiện tại)
exception	Throwable	Ngoại lệ chứa lỗi của trang

Trong số các đối tượng được trình bày trên, hầu như bạn đã quen dùng trong servlet duy chỉ có pageContext là đối tượng mới. Đối tượng này đóng vai trò như một bộ chứa (Container) để từ đó truy xuất ra các thành phần khác.

### 4.3.5 Đối tượng pageContext

Đối tượng pageContext hoạt động như một kho lưu trữ thông tin trung tâm của trang JSP. PageContext có liên quan đến "thực hiện" của một trang, có nghĩa là nó được tạo ra vào lúc bắt đầu của servlet cài đặt của trang JSP và bị giải phóng ở cuối trang. Thông thường, hệ thống khởi tạo các biến ngầm định bằng cách gọi một phương thức trong đối tượng pageContext. Ví dụ, nếu JSP của bạn sử dụng đối tượng session, các servlet được tạo ra thường có chứa một dòng như thế này:

```
HttpSession session = pageContext.getSession ();
```

Ngoài việc cung cấp truy cập vào các đối tượng ngầm định, đối tượng pageContext có khả năng quét các thuộc tính và tên thuộc tính trong tất cả các phạm vi có thể (page, request, session, và application) bằng phương thức

```
String findAttribute(String attributeName)
```

Cuối cùng, đối tượng pageContext chứa các phương thức tiện ích làm cho nó dễ dàng để bao gồm hoặc chuyển tiếp đến một servlet hoặc JSP khác.

```
Include(String path) hay forward(String path)
```

Là một nhà phát triển JSP, bạn nên thường sử dụng pageContext để lưu trữ hoặc lấy thuộc tính hoặc để chuyển tiếp hoặc bao gồm các trang JSP.

#### 4.3.5.1 Truy cập các đối tượng ngầm định

Các đối tượng được kết hợp với các biến ngầm định (request, session, out...) có sẵn từ đối tượng pageContext.

Các đối tượng ngầm định được khởi tạo từ các phương thức sau đây trong pageContext:

Phương thức	Công dụng
public ServletRequest getRequest()	Tham chiếu đến đối tượng request
public ServletResponse getResponse()	Tham chiếu đến đối tượng response
public JspWriter getOut()	Tham chiếu đến đối tượng out
public HttpSession getSession()	Tham chiếu đến đối tượng session
public ServletContext getServletContext()	Tham chiếu đến đối tượng application
public ServletConfig getServletConfig()	Tham chiếu đến đối tượng config
public Object getPage()	Tham chiếu đến đối tượng page
public Throwable getException()	Tham chiếu đến đối tượng exception

### 4.3.5.2 Truy cập các thuộc tính từ lớp PageContext

Lớp PageContext thực hiện nhiệm vụ tăng gấp đôi khi nó đi kèm để lưu trữ và lấy các thuộc tính. Trước tiên, bạn có thể lưu trữ các thuộc tính trong thể hiện của PageContext một cách rõ ràng. Những thuộc tính này có phạm vi page, có nghĩa là các thuộc tính chỉ sẵn để sử dụng trong vòng đời của một trang. Mỗi thể hiện của mỗi trang JSP có PageContext riêng cho các thuộc tính lưu trữ. Ngoài ra, bất kỳ JavaBeans bạn tạo ra với một phạm vi page cũng được lưu trữ như các thuộc tính trong PageContext.

Các phương thức để lưu trữ, thu hồi, và loại bỏ các thuộc tính trong PageContext tương tự như các đối tác của nó là ServletRequest, HttpSession và ServletContext:

Phương thức	Công dụng
Object getAttribute(String attributeName)	Lấy thuộc tính trong phạm vi trang
void setAttribute(String attributeName, Object attributeValue)	Lưu thuộc tính vào phạm vi trang
Enumeration<String> getAttributeNames()	Lấy tất cả các tên thuộc tính trong phạm vi trang
<b>Object findAttribute(String attributeName)</b>	<b>Tìm thuộc tính trong tất cả các phạm vi theo thứ tự page-&gt;request-&gt;session-&gt;application. Nếu tìm thấy rồi thì dừng lại.</b>

Khía cạnh thú vị nhất của PageContext mà nó mang lại cho bạn truy cập vào các thuộc tính trong phạm vi cụ thể nào đó bằng cách thêm một tham số phụ cho các phương thức getAttribute, setAttribute và removeAttribute:

Phương thức	Công dụng
Object getAttribute(String attributeName, int scope)	Lấy thuộc tính trong phạm vi chỉ định bởi tham số thứ 2
void setAttribute(String attributeName, Object attributeValue, int scope)	Lưu thuộc tính vào phạm vi chỉ định bởi tham số thứ 2
Enumeration<String> getAttributeNamesInScope(int scope)	Lấy tất cả các tên thuộc tính trong phạm vi chỉ định bởi tham số thứ 2

Đặc điểm mới trong JSP2.0: Đặc tả của JSP2.0 mang lại nhiều tính năng mới hơn cho JSP. Nó cũng tách JSP từ servlet. JSP có khả năng hoạt động riêng của chúng, và các tác giả của bản đặc tả JSP có thể thấy rằng tương lai JSP container có thể lựa chọn để làm việc với JSP trong môi trường riêng của họ. Do đó, có một số thay đổi thiết kế. Ví dụ, PageContext bây giờ được mở rộng thành JspContext một lớp mới tóm tắt thông tin đó không phụ thuộc vào servlet. Nó hỗ trợ SimpleTags (bạn sẽ tìm hiểu điều này sau) mở rộng JSP bằng cách định nghĩa ra các thẻ mới. Cùng với các phương thức khác, JspContext cũng cung cấp các phương thức để làm việc với các thuộc tính, ở đây chúng tôi chỉ giới thiệu bao quát.



Giá trị phạm vi được xác định bởi các hằng số `PageContext` là `PAGE_SCOPE`, `REQUEST_SCOPE`, `SESSION_SCOPE`, và `APPLICATION_SCOPE`. Ví dụ, để lấy một đối tượng từ một session, bạn có thể thực hiện lời gọi phương thức sau đây:

```
Object ob = pageContext.getAttribute("myObject", PageContext.SESSION_SCOPE);
```

Bạn có thể xác định vị trí tất cả các tên thuộc tính trong một phạm vi nhất định bằng cách gọi `getAttributeNamesInScope()`:

```
public Enumeration getAttributeNamesInScope(int scope)
```

Bạn có thể tìm kiếm thông qua tất cả các phạm vi đối với một đối tượng cụ thể bằng cách gọi `findAttribute()`:

```
public Object findAttribute(String name)
```

`PageContext` tìm kiếm đối tượng đầu tiên trong phạm vi page, sau đó phạm vi request, phạm vi session, và cuối cùng là phạm vi application. Tìm kiếm dừng lại ở phạm vi đầu tiên mà nó tìm thấy đối tượng. Nếu bạn có một thuộc tính tên là `myObject` được lưu trữ trong đối tượng request và cũng trong đối tượng application thì `findAttribute("myObject")` trả về đối tượng được lưu trữ trong đối tượng request.

Phương thức `getAttributeScope()` trả về phạm vi mà nó tìm thấy thuộc tính:

```
public int getAttributeScope(String name)
```

Ví dụ, nếu phương thức `findAttribute` tìm thấy `myObject` trong đối tượng request, `getAttributeScope()` sẽ trả về giá trị là `PageContext.REQUEST_SCOPE`.

Ví dụ sau cho thấy một JSP xuất ra tất cả thuộc tính có trong phạm vi request

```
<%@ page import="java.util.*" %>
<html>
<body>
<%
 // Lấy tên tất cả các thuộc tính trong phạm vi REQUEST
 int scope = PageContext.REQUEST_SCOPE;
 Enumeration<String> e = pageContext.getAttributeNamesInScope(scope);
 while (e.hasMoreElements()) {
 // Lấy tên thuộc tính.
 String name = e.nextElement();
 // Lấy giá trị thuộc tính.
 Object value = pageContext.getAttribute(name, scope);
 out.print("<p>" + name + " : " + value);
 }
%>
</body>
</html>
```

Phương thức `getAttributeNamesInScope()` rất hữu ích để khám phá các đối tượng phạm vi khác nhau.

#### 4.3.5.3 Chuyển tiếp (forward) và bao gồm (include)

Bạn đã được sử dụng `include` và `forward` trong servlet nhờ vào `RequestDispatcher`

✓ `req.getRequestDispatcher("page.jsp").include(req, resp)`

- ✓ `req.getRequestDispatcher("page.jsp").forward(req, resp)`

Trong JSP bạn có thể viết như trên. Tuy nhiên có cách viết gọn hơn tương đương như sau

- ✓ `pageContext.include("page.jsp")`
- ✓ `pageContext.forward("page.jsp")`

Bạn cũng có cách viết khác thuần túy jsp là

- ✓ `<jsp:include page="page.jsp" flush="true"/>`
- ✓ `<jsp:forward page="page.jsp"/>`

Lưu ý quan trọng: *trước forward bạn không được xuất bất kỳ cái gì cả, nếu không bạn sẽ gặp lỗi.*

## 4.4 Ngôn ngữ biểu thức - EL

EL (Expression Language) là ngôn ngữ biểu thức được sử dụng trong JSP để đơn giản hóa việc truy xuất, tính toán và hiển thị các attribute, parameter, cookie, header... trong JSP.

Bạn đã từng sử dụng **`${xyz}`** và hiểu là hiển thị giá trị của thuộc tính (attribute) có tên là xyz trong phạm vi nào đó (page, request, session, application). Đó là biểu thức EL. Dù sao bạn vẫn chưa thể hiểu được một cách kỹ lưỡng về hoạt động của biểu thức EL này. Trong phần này bạn tìm hiểu sâu hơn về biểu thức EL.

### 4.4.1 Truy xuất thuộc tính xyz trong phạm vi

Để truy xuất attribute (tên là xyz) được chia sẻ trong scope (page, request, session hay application) cụ thể nào đó, bạn sử dụng cách sau đây

- ✓ Page: `${pageScope.xyz}` hay `${ pageScope['xyz']}`
- ✓ Request: `${requestScope.xyz}` hay `${ requestScope['xyz']}`
- ✓ Session: `${sessionScope.xyz}` hay `${ sessionScope['xyz']}`
- ✓ Application: `${applicationScope.xyz}` hay `${ applicationScope['xyz']}`

Khi bạn truy xuất attribute không chỉ rõ scope (**`${xyz}`**) thì JSP sẽ truy tìm xyz trong tất cả các phạm vi theo thứ tự (page -> request -> session -> application). Nếu tìm thấy sẽ dừng lại mà không tìm tiếp.

### 4.4.2 Truy xuất tham số

- ✓ Truy xuất tham số một giá trị xyz: `${param.xyz}` hay `${param[xyz]}`
- ✓ Truy xuất tham số nhiều giá trị xyz: `${paramValues.xyz[0]}` hay `${paramValues[xyz][0]}`

### 4.4.3 Truy xuất cookie có tên xyz

- ✓ `${cookie.xyz.value}` hay `${cookie[xyz].value}`

### 4.4.4 Làm việc với các kiểu attribute đặc biệt

❖ **Truy xuất tập hợp:**

Sử dụng chỉ số để truy xuất các phần tử trong tập hợp `${collection[0]}`

❖ **Truy xuất map:**

Sử dụng khóa để truy xuất các phần tử trong map `${map[key]}`

❖ **Truy xuất thuộc tính javabeen:**

Sử dụng tên thuộc tính (loại bỏ get và set sau đó đổi ký tự đầu sang ký tự thường) truy xuất giá trị thuộc tính `${bean.propertyName}`. Nghĩa là trong lớp JavaBean có phương thức getter là `getPropertyName()`.

Ví dụ sau hiển thị giá trị thuộc tính **error** được chia sẻ trong phạm vi nào đó. Trong ví dụ cũng hiển thị giá trị của cookie có tên **ckiId** và **ckiPw** trên các ô nhập txtUserName và txtPassword

```
<%@ page contentType="text/html; charset=utf8" pageEncoding="utf8"%>
<!DOCTYPE html>
<html>
<head>
 <meta http-equiv="Content-Type" content="text/html; charset=utf8">
 <title>Input Form</title>
</head>
<body>
<label style="color:red">${error}</label>
<form method="post" action="Login.do">
 <p>User name:
 <input name="txtUserName" value="${cookie['ckiId'].value}">

 <p>Password:
 <input name="txtPassword" type="password" value="${cookie['ckiPw'].value}">

 <p>
 <input name="chkRemember" type="checkbox">Remember me?

 <p>
 <input type="submit" name="btnLogin" value="Login">
</form>
</body>
</html>
```

### 4.4.5 Thư viện hàm hỗ trợ EL

Từ JSTL 1.1 EL được hỗ trợ thêm sức mạnh của các hàm xử lý để tăng cường hơn nữa khả năng của EL. Đa phần các hàm trong bảng sau tập trung vào việc xử lý chuỗi, riêng hàm `fn:length()` được sử dụng để đếm số phần tử trong Collection, Array, Map... hay số ký tự trong chuỗi.

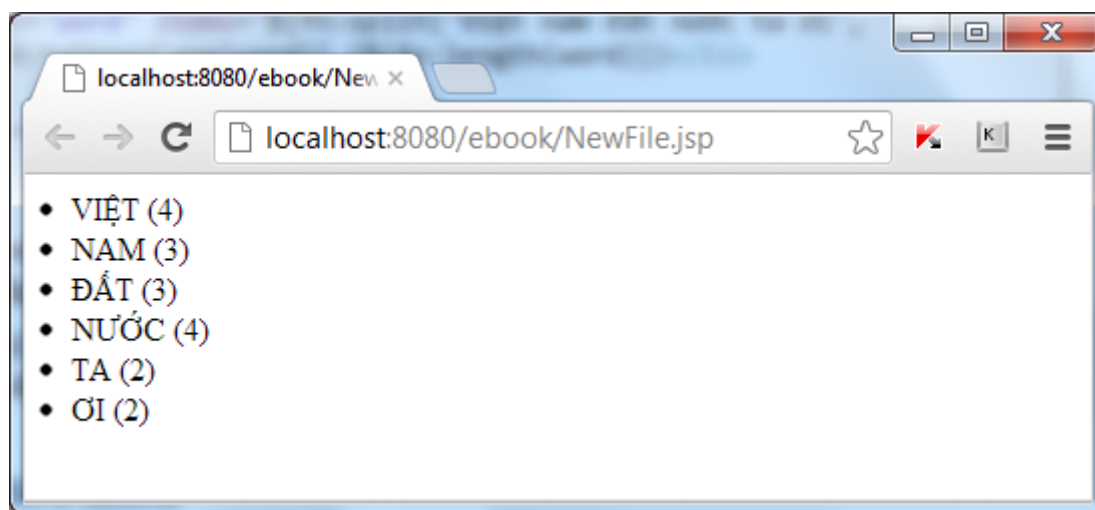
Tên hàm	Mô tả mục đích
<code>boolean fn:contains(String, String)</code>	Chuỗi (1) có chứa chuỗi (2) hay không
<code>boolean fn:containsIgnoreCase(String, String)</code>	Chuỗi (1) có chứa chuỗi (2) hay không (không phân biệt hoa thường)
<code>boolean fn:startsWith(String, String)</code>	Chuỗi đối số thứ nhất có bắt đầu bởi chuỗi đối số thứ hai hay không

boolean fn:endsWith(String, String)	Chuỗi (1) có kết thúc bởi (2) hay không
String fn:escapeXML(String)	Mã hóa thành thực thể các ký tự phạm cú pháp XML
int fn:indexOf(String, String)	Tìm vị trí xuất hiện đầu tiên của chuỗi (2) trong chuỗi (1)
String[] fn:split(String, String)	Tách chuỗi (1) thành mảng sử dụng chuỗi (2) như chuỗi phân cách
String fn:join(String[], String)	Gia nhập các phần tử trong mảng (1) thành chuỗi sử dụng chuỗi(2) như là chuỗi phân cách.
int fn:length(Map; array; Collection; Iterator; Enumeration; or String)	Tìm độ dài của chuỗi hay số lượng các phần tử trong tập hợp.
String fn:replace(String, String, String)	Thay thế chuỗi (1) bởi chuỗi (3) trong chuỗi (1)
String fn:substring(String, int, int)	Lấy chuỗi trong chuỗi (1) tính từ vị trí (1) cho đến vị trí (3)
String fn:substringAfter(String, String)	Lấy chuỗi con trong chuỗi (1) đứng sau chuỗi (2)
String fn:substringBefore(String, String)	Lấy chuỗi con trong chuỗi (1) đứng trước chuỗi (2)
String fn:toLowerCase(String)	Đổi chuỗi sang chữ thường
String fn:toUpperCase(String)	Đổi chuỗi sang chữ HOA
String fn:trim(String)	Cắt bỏ khoảng trắng 2 đầu chuỗi

Khi sử dụng các hàm này, bạn cần khai báo thư viện thẻ đầu trang JSP

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

Ví dụ sau đây sẽ cho bạn thấy cách tách chuỗi "Việt nam đất nước ta ơi" thành các từ bỏie ký tự trắng. Sau đó đổi thành chữ hoa, đếm số ký tự của mỗi từ và hiển thị.



Mã nguồn JSP của trang

```
<%@ page contentType="text/html; charset=utf-8" pageEncoding="utf-8"%>
<%@ taglib uri="http://java.sun.com/jstl/core_rt" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
```

```
<c:forEach var="word" items="${fn:split('Việt nam đất nước ta ơi', ' ')}">
 ${fn:toUpperCase(word)} (${fn:length(word)})
</c:forEach>
```

## 4.5 JSTL

JSTL (Java Standard Tag Library) là bộ thư viện thẻ chuẩn được sử dụng để tăng cường cho sức mạnh của lập trình JSP.

Trong số các bộ thư viện thẻ được giới thiệu, các thẻ điều khiển và định dạng được sử dụng nhiều nhất. Vì vậy trong phần này, bạn sẽ được tìm hiểu về các thẻ làm việc trong hai phần này.

### 4.5.1 Thẻ điều khiển

Khai báo thư viện thẻ lỗi:

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

Các thẻ điều khiển thuộc thư viện thẻ lỗi gồm

- ✓ **<c:forEach>** lệnh lặp trên Map, Collection và Array
- ✓ **<c:if>** lệnh if
- ✓ **<c:choose>...<c:when>**: lệnh if...elseif...else

#### 4.5.1.1 Thẻ <c:if>

Thực thi phần thân của thẻ nếu giá trị của thuộc tính test có giá trị true

```
<c:if test="điều kiện">
 Thân thẻ sẽ được thực hiện nếu điều kiện ở thuộc tính @test đúng
</c:if>
```

Ví dụ: nếu trong phạm vi session có tồn tại thuộc tính có tên là user thì hiển thị dòng  
<h1>Welcome <tên của user trong session></h1>

```
<%@ page contentType="text/html; charset=utf-8" pageEncoding="utf-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<!DOCTYPE html>
<html>
 <head>
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
 <title>JSP Page</title>
 </head>
 <body>
 <c:if test="${!empty sessionScope.user}">
 <h1>Welcome ${sessionScope.user.fullName}</h1>
 </c:if>
 </body>
</html>
```

### 4.5.1.2 Thẻ <c:choose>

Bao bọc các điều kiện loại trừ lẫn nhau. Tương tự if...else if....else trong java. Nếu điều kiện của thuộc tính @test của thẻ <c:when> nào đúng thì thân của thẻ đó được thực hiện. Ngoài ra thân của <c:otherwise> sẽ được thực hiện.

Cú pháp

```
<c:choose>
 <c:when test="điều kiện 1"></c:when>
 <c:when test="điều kiện N"></c:when>
 ...
 <c:otherwise> </c:otherwise>
</c:choose>
```

Ví dụ: hiển thị lời chào tùy thuộc vào thời điểm truy xuất trong ngày

```
<%@ page contentType="text/html; charset=utf-8" pageEncoding="utf-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<jsp:useBean id="date" class="java.util.Date" scope="page"/>

<c:choose>
 <c:when test="${date.hours < 12}">
 <c:set var="greeting" value="Good morning !"/>
 </c:when>
 <c:when test="${date.hours > 17}">
 <c:set var="greeting" value="Good evening !"/>
 </c:when>
 <c:otherwise>
 <c:set var="greeting" value="Hello"/>
 </c:otherwise>
</c:choose>

<!DOCTYPE html>
<html>
 <head>
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
 <title>JSP Page</title>
 </head>
 <body>
 <h1>${greeting}</h1>
 </body>
</html>
```

### 4.5.1.3 Thẻ <c:forEach>

Bạn có thể sử dụng hành động <c:forEach> để lặp qua một cấu trúc dữ liệu, chẳng hạn như một mảng, Map, hoặc tập hợp nếu bạn chỉ định cấu trúc dữ liệu với thuộc tính items.

Bạn cũng có thể sử dụng `<c:forEach>` để lặp qua các giá trị số nguyên nếu bạn không chỉ định thuộc tính `items`.

Cách 1: lặp theo cấu trúc dữ liệu

```
<c:forEach items [begin] [end] [step] [var] [varStatus]>
 body content
</c:forEach>
```

Cách 2: duyệt các giá trị số nguyên

```
<c:forEach begin end [step] [var] [varStatus]>
 body content
</c:forEach>
```

Ý nghĩa các thuộc tính:

Thuộc tính	Kiểu	Mô tả
items	String, Array, Collection, Iterator, Enumeration, Map	Chỉ định cấu trúc dữ liệu. Nếu là chuỗi thì các phần tử phải cách nhau dấu phẩy.
begin	int	Vị trí hoặc số bắt đầu duyệt
end	int	Vị trí số kết thúc duyệt
step	int	Bước nhảy
var	String	Biến nắm giữ phần tử hiện tại của tập cấu trúc
varStatus	String	Biến nắm giữ thông tin trạng thái

Ví dụ: duyệt tập hợp `items`, mỗi item sẽ truy xuất và hiển thị thuộc tính `name` và `price`.

```
<%@ page contentType="text/html; charset=utf-8" pageEncoding="utf-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<!DOCTYPE html>
<html>
<head>
 <title>Học JSTL</title>
</head>
<body>

 <c:forEach var="item" items="${items}">

 ${item.name}
 ${item.price}

 </c:forEach>
```

```
</body>
</html>
```

### 4.5.2 Thẻ định dạng

Với JSTL bạn có thể định dạng ngay và số rất dễ dàng trên JSP nhờ bộ thẻ định dạng. Bạn cần khai báo thư viện thẻ định dạng trước khi sử dụng thẻ

- ✓ Khai báo: `<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>`
- ✓ Thẻ định dạng ngày: `<f:formatDate>`
- ✓ Thẻ định dạng số: `<f:formatNumber>`

Ví dụ 1: định dạng theo miền địa phương

```
<fmt:setLocale value='en-US' /> <!--chọn ngôn ngữ-->
English: <fmt:formatNumber value='1255.23' />
<fmt:setLocale value='vi-VN' />
Việt Nam: <fmt:formatNumber value='1255.23' />
```

Kết quả: English: **1,255.23** Việt Nam: **1.255,23**

Ví dụ 2: các kiểu định dạng

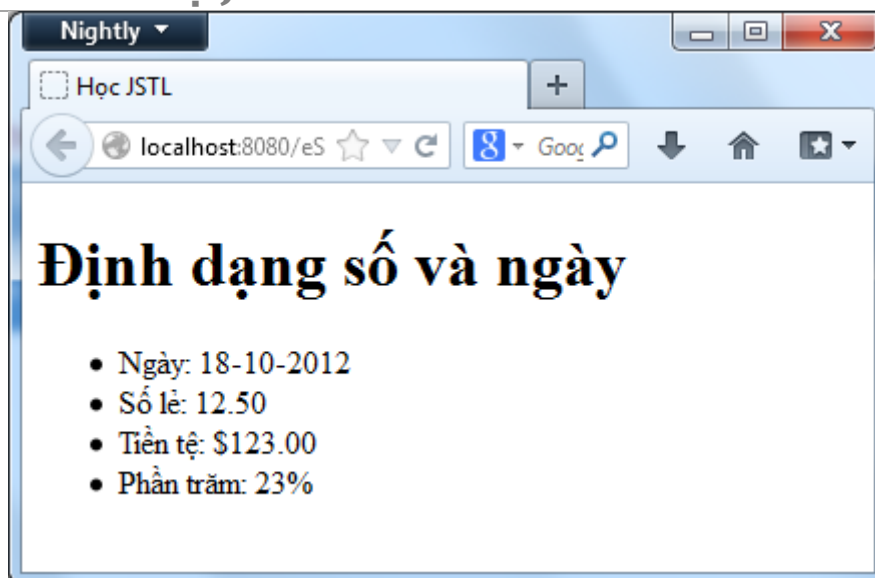
```
<%@ page contentType="text/html" pageEncoding="UTF-8"%>
<%@ taglib prefix="f" uri="http://java.sun.com/jsp/jstl/fmt" %>
<!DOCTYPE html>
<html>
<head>
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
 <title>Học JSTL</title>
</head>
<body>

<h1>Định dạng số và ngày</h1>
<jsp:useBean id="now" class="java.util.Date"/>

 Ngày: <f:formatDate value="${now}" pattern="dd-MM-yyyy"/>
 Số lẻ: <f:formatNumber value="${12.5}" minFractionDigits="2"/>
 Tiền tệ: <f:formatNumber value="${123}" type="currency"/>
 Phần trăm: <f:formatNumber value="${0.23}" type="percent"/>

</body>
</html>
```





Kết quả hiển thị

## 4.6 JavaBeans

### 4.6.1 Định nghĩa lớp JavaBean

Lớp JavaBean là lớp Java được xây tuân theo một số qui ước để thuận tiện cho việc sử dụng trong JSP. Qui ước của lớp JavaBean bao gồm

- ✓ **Class phải định nghĩa public.** Yêu cầu này là bắt buộc vì class được sử dụng bên ngoài package với bean class.

```
public class MyJavaBean{...}
```

- ✓ **Phải có constructor không tham số.** Nếu lớp của bạn không định nghĩa constructor không tham số thì lớp này không thể khởi tạo đối tượng mặt định để sử dụng với `<jsp:useBean/>` của JSP. Khi đó lớp của chỉ được khởi tạo bằng cách viết code java mà thôi. Khi bạn không định nghĩa bất kỳ constructor nào trong lớp thì constructor không tham số được sử dụng mặc định. Còn nếu bạn có định nghĩa một constructor có tham số nào đó rồi thì hãy định nghĩa thêm constructor không tham cho lớp.

*// Hoặc không có constructor nào => constructor không tham số là mặc định*

```
public class MyJavaBean{...}
```

*// Hoặc có constructor => phải khai báo constructor không tham số*

```
public class MyJavaBean{
 public MyJavaBean(){...}
 public MyJavaBean(parameters){...}
}
```

- ✓ **Nên implements interface Serializable.** Nếu bạn không implements interface này thì trong quá trình persistence (chuyển đổi trạng thái của bean Activate <-> Passivate) của servlet engine sẽ đánh mất dữ liệu của javabean.
- ✓ **Phải định nghĩa các phương thức *getter* và *setter* để đọc ghi dữ liệu thuộc tính của bean.**

```
public class MyJavaBean{
 public <type> getXxx(){...}
 public void setXxx(<type> value){...}
}
```

Cặp phương thức getXxx() và setXxx() được sử dụng để làm việc với thuộc tính xxx. Nếu kiểu của thuộc xxx tính là boolean thì nên sử dụng isXxx() thay cho getXxx().

```
public class MyJavaBean{
 public boolean isXxx(){...}
 public void setXxx(boolean value){...}
}
```

Trong trường hợp chỉ có getXxx thì thuộc tính được gọi là thuộc tính chỉ đọc (readonly). Trường hợp này thường xảy ra khi chúng ta chỉ truy xuất giá trị được tính toán từ các biến khác mà ra. Ngược lại chỉ có setXxx thì gọi là chỉ ghi (writeonly), rất hiếm xảy ra.

Nếu thuộc tính dùng để nắm giữ mảng hay List (còn được gọi là thuộc tính có chỉ số - indexed properties), thì bạn cần định nghĩa 2 cặp phương thức get/set cùng tên để truy xuất toàn mảng và các phần tử tại một vị trí nào đó bên trong mảng.

```
public class MyJavaBean{
 public <type>[] getXxx(){}
 public void setXxx(<type>[] value){}
 public <type> getXxx(int index){}
 public void setXxx(int index, <type> value){}
}
```

**Ví dụ 1:** lớp JavaBean sau đây có 4 thuộc tính (3 thuộc tính đọc/ghi – toanHangA, toanHangB, toanTu; 1 thuộc tính chỉ đọc là ketQua)

```
public class MayTinh implements Serializable{
 private double toanHangA;
 private double toanHangB;
 private char toanTu;
 /*
 * ketQua là thuộc tính readonly cho phép đọc kết quả
 * của phép tính dựa vào 2 toán hạng và toán tử hiện có
 */
 public double getKetQua() {
 double ketQua = 0;
 }
}
```

```

 switch(toanTu){
 case '+':
 ketQua = toanHangA + toanHangB;
 break;
 case '-':
 ketQua = toanHangA - toanHangB;
 break;
 case 'x':
 ketQua = toanHangA * toanHangB;
 break;
 case '/':
 ketQua = toanHangA / toanHangB;
 break;
 }
 return ketQua;
 }
 /*
 * toanHangA là thuộc tính read/write
 */
 public double getToanHangA() {
 return toanHangA;
 }
 public void setToanHangA(double toanHangA) {
 this.toanHangA = toanHangA;
 }
 /*
 * toanHangB là thuộc tính read/write
 */
 public double getToanHangB() {
 return toanHangB;
 }
 public void setToanHangB(double toanHangB) {
 this.toanHangB = toanHangB;
 }
 /*
 * toanTu là thuộc tính read/write
 */
 public char getToanTu() {
 return toanTu;
 }
 public void setToanTu(char toanTu) {
 this.toanTu = toanTu;
 }
}

```

Ví dụ 2: định nghĩa lớp JavaBean với

- ✓ **masv, hoTen, tuoi, gioiTinh, diem** là các thuộc tính **read/write** có kiểu dữ liệu khác nhau
- ✓ **hocLuc** là thuộc tính chỉ đọc (**readonly**)
- ✓ **soThich** là thuộc tính có chỉ số (**indexed property**).

```
public class SinhVien implements Serializable{
```

```
/*
 * các trường dữ liệu nội bộ
 */
private String masv;
private String hoTen;
private boolean gioiTinh;
private int tuoi;
private double diem;
private String[] soThich;
/*
 * masv là thuộc tính read/write
 */
public String getMasv() {
 return masv;
}
public void setMasv(String masv) {
 this.masv = masv;
}
/*
 * hoTen là thuộc tính read/write
 */
public String getHoTen() {
 return hoTen;
}
public void setHoTen(String hoTen) {
 this.hoTen = hoTen;
}
/*
 * gioiTinh là thuộc tính read/write
 */
public boolean isGioiTinh() {
 return gioiTinh;
}
public void setGioiTinh(boolean gioiTinh) {
 this.gioiTinh = gioiTinh;
}
/*
 * tuoi là thuộc tính read/write
 */
public int getTuoi() {
 return tuoi;
}
public void setTuoi(int tuoi) {
 this.tuoi = tuoi;
}
/*
 * diem là thuộc tính read/write
 */
public double getDiem() {
 return diem;
}
public void setDiem(double diem) {
 this.diem = diem;
}
```

```
}
/*
 * soThich là thuộc tính có chỉ số và cho phép read/write
 */
public String[] getSoThich() {
 return soThich;
}
public void setSoThich(String[] soThich) {
 this.soThich = soThich;
}
public String getSoThich(int index) {
 return soThich[index];
}
public void setSoThich(int index, String soThich) {
 this.soThich[index] = soThich;
}
/*
 * hocLuc là thuộc tính readonly
 */
public String getHocLuc() {
 if (diem < 5) {
 return "yếu";
 }
 if (diem < 6.5) {
 return "trung bình";
 }
 if (diem < 7.5) {
 return "khá";
 }
 if (diem < 9) {
 return "giỏi";
 }
 return "xuất sắc";
}
}
```

## 4.6.2 Sử dụng JavaBean

Bean được hiểu là một đối tượng được tạo ra từ một lớp JavaBean và được chia sẻ trong một phạm vi lưu trữ (page, request, session hay application) nào đó. Vì vậy để tạo đối tượng bean và chia sẻ bean trong phạm vi lưu trữ bạn có thể viết mã bằng java hay sử dụng thẻ `<jsp:useBean>` trong JSP.

### 4.6.2.1 Khởi tạo bean

```
<jsp:useBean id="?" class="?" scope="?" />
```

Thẻ trên được sử dụng để tạo ra một đối tượng mới từ lớp JavaBean @class, chia sẻ trong phạm vi @scope, với định danh @id nếu trong @scope chưa tồn tại bean này. Ngược lại thẻ chỉ tham chiếu đến bean đang tồn tại trong @scope có cùng @id.

- ✓ **@class** là thuộc tính chỉ định class dùng để tạo bean. Thông thường chúng ta dùng dạng đầy đủ để chỉ ra class này. Ví dụ: class="nnghiem.bean.MayTinh".

- ✓ **@id** dùng để đặt tên cho bean, là định danh duy nhất trong mỗi phạm vi chia sẻ. Vì vậy trong trường hợp trong phạm vi chia sẻ đó đã có bean có id này rồi thì nó tham chiếu tới bean đã tồn tại mà không phải tạo ra một đối tượng mới. Ví dụ id="cal"
- ✓ **@scope** dùng để chỉ ra phạm vi lưu trữ và chia sẻ của bean. Các giá trị hợp lệ là page, request, session và application. Ý nghĩa các phạm vi chia sẻ trên là:
  - **page**: bean chỉ tồn tại trong phạm vi trang jsp nay thôi. Phạm vi này chỉ sử dụng để tiện cho sử dụng EL mà thôi. Đây là thuộc tính mặc định.
  - **request**: bean tồn tại theo sự tồn tại của request. Điều này có nghĩa các thành phần web khác (listener, filter, servlet, custom tag) cùng chung request này có thể chia sẻ bean được tạo ra. Các JSP và servlet nằm trong dây chuyền include()/forward() sẽ cùng một request.
  - **session**: bean tồn tại trong phạm vi phiên làm việc. Nghĩa là các thành phần web khác có thể chia sẻ bean này khi được kích hoạt trong cùng phiên làm việc. Lưu ý mỗi web user làm việc với một phiên khác nhau. Của ai tạo rathì của người đó dùng. Giỏ hàng, thông tin người dùng thường được lưu ở phạm vi này.
  - **application**: bean được tồn tại trên toàn ứng dụng web. Nghĩa là tất cả các trang được truy xuất bởi bất kỳ user nào cũng có thể truy xuất. Chia sẻ chung cho toàn mọi người. Bộ đếm, khẩu hiệu, tên doanh nghiệp, logo thường lưu ở đây.

Ví dụ sau sẽ tạo ra một đối tượng từ lớp MayTinh và lưu vào phạm vi chia sẻ session với tên là cal nếu trong session chưa tồn tại attribute nào có tên cal. Ngược lại (nếu bean cal đã được tạo và lưu vào session) thì thẻ này được sử dụng để tham chiếu đến bean đã tồn tại.

```
<jsp:useBean id="cal" class="nnghiem.bean.MayTinh" scope="session"/>
```

Trong Java (servlet, jsp, listener, filter) bạn có thể thực hiện công việc tương đương thẻ <jsp:useBean> ở trên như đoạn mã sau:

```
HttpSession session = request.getSession();
MayTinh cal = (MayTinh) session.getAttribute("cal");
if(cal == null){
 bean = new MayTinh();
 session.setAttribute("cal", cal);
}
```

#### 4.6.2.2 Đọc/ghi thuộc tính của bean

**Ghi giá trị cho các thuộc tính của bean:** Bean sau khi được tạo ra cần thiết lập dữ liệu cho các thuộc tính của nó. Trong Java bạn chỉ việc gọi phương thức setXxx(value) để thiết lập giá trị cho thuộc tính xxx của bean. Ví dụ: cal.setToanHangA(5.5);

Trong JSP bạn có 3 cách để thiết lập giá trị cho các thuộc tính của bean bằng cách sử dụng thẻ <jsp:setProperty name="?" property="?" value="?" param="?"/>. Với 4 attribute, thẻ <jsp:setProperty/> được chia ra làm 3 trường hợp sử dụng sau đây:

- ✓ 

```
<jsp:setProperty name="cal" property="toanTu" value="+"/>
```

Thiết lập giá trị thuộc tính (toanTu) của bean (cal) bằng giá trị cụ thể (+)
- ✓ 

```
<jsp:setProperty name="cal" property="toanTu" param="txtToanTu"/>
```

Thiết lập giá trị thuộc tính (toanTu) của bean (cal) bằng giá trị của tham số (txtToanTu)

✓ `<jsp:setProperty name="cal" property="*" />`

Thiết lập tất cả các giá trị thuộc tính (toanHangA, toanHangB và toanTu) của bean (cal) bằng các giá trị của tham số cùng tên.

Ví dụ sau đây cho phép khởi tạo bean và thiết lập giá trị cho tất cả các thuộc tính từ các tham số cùng tên của bean cal.

```
<jsp:useBean id="cal" class="nnghiem.bean.MayTinh" scope="session" />
<jsp:setProperty name="cal" property="*" />
```

**Đọc và hiển thị giá trị các thuộc tính của bean:** Trong JSP các phương thức getter được sử dụng để đọc và hiển thị dữ liệu thuộc tính của java bean dưới 2 hình thức:

✓ Sử dụng EL `${bean.propertyName}`

✓ Sử dụng thẻ `<jsp:getProperty property="propertyName" name="bean" />`.

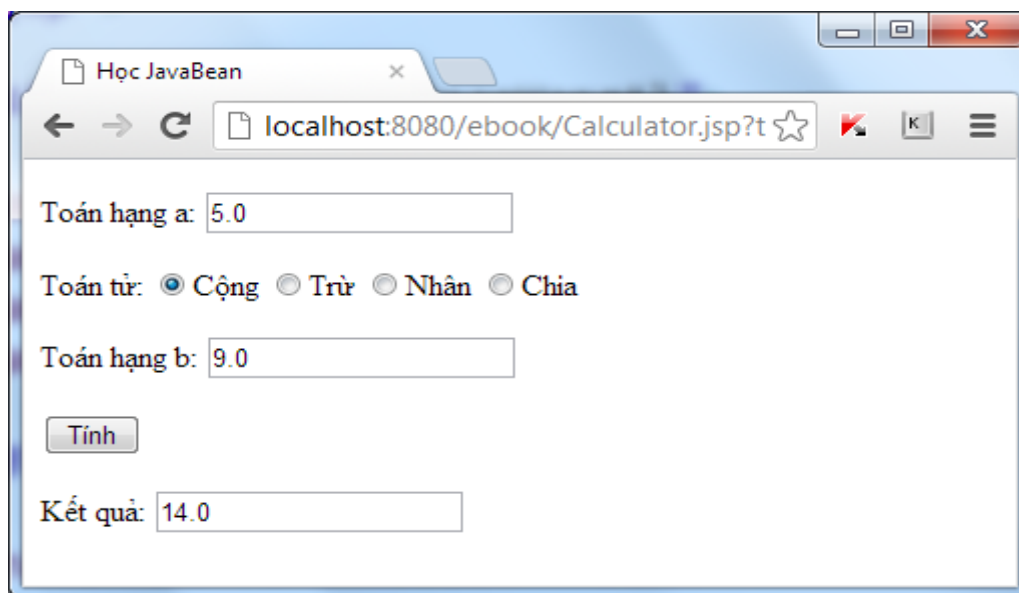
Trong đó **"bean"** là một đối tượng (object) được lưu giữ trong một scope (page, request, session hay application) nào đó của ứng dụng web. Và **"propertyName"** là phần đã loại bỏ get/set khỏi các phương thức getter và setter sau đó chuyển đổi ký tự đầu tiên của phần còn lại sang ký tự thường.

Ví dụ để hiển thị thuộc tính ketQua của bean cal, bạn có thể viết:

`${cal.ketQua}` hay `<jsp:getProperty property="ketQua" name="cal" />`.

### 4.6.3 Một số ví dụ về JavaBean

Ví dụ 1: Sử dụng lớp MayTinh trên đây để thực hiện các phép tính cơ bản là cộng, trừ, nhân và chia như mô tả trên giao diện sau đây.



Nhập các toán hạng a và b sau đó chọn toán tử và nhấp nút Tính. Kết quả sẽ được tính và hiển thị trong ô kết quả.

Sau đây là mã của trang Calculator.jsp

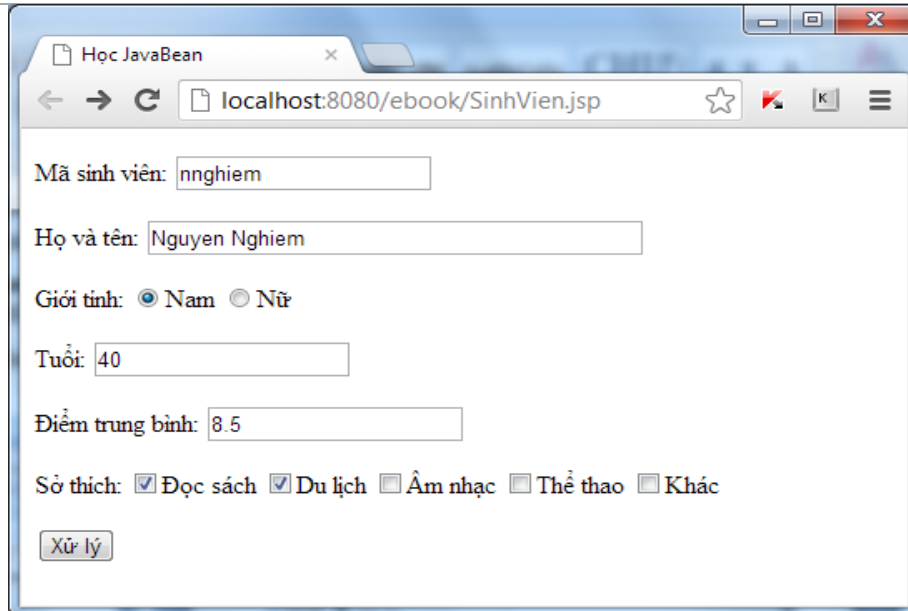
```
<%@ page contentType="text/html; charset=utf-8" pageEncoding="utf-8"%>
<!DOCTYPE html>
<html>
<head>
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
 <title>Học JavaBean</title>
</head>
<body>
<jsp:useBean id="cal" class="nnghiem.bean.MayTinh" scope="request"/>
<jsp:setProperty name="cal" property="*" />
<form action="Calculator.jsp">
 <p>Toán hạng a:
 <input name="toanHangA" value="${cal.toanHangA}" />
 <p>Toán tử:
 <input type="radio" name="toanTu" value="+" checked>Cộng
 <input type="radio" name="toanTu" value="-">Trừ
 <input type="radio" name="toanTu" value="x">Nhân
 <input type="radio" name="toanTu" value=":">Chia
 <p>Toán hạng b:
 <input name="toanHangB" value="${cal.toanHangB}" />
 <p>
 <input type="submit" name="btnTinh" value=" Tính " />
 <p>Kết quả:
 <input name="txtKetQua" value="${cal.ketQua}" />
</form>
</body>
</html>
```

Trong trang JSP trên bạn cần lưu ý các đoạn mã quan trọng sau đây:

- ✓ **`<jsp:useBean id="cal" class="nnghiem.bean.MayTinh" scope="request"/>`**: Dòng mã này được sử dụng để khởi tạo bean
- ✓ **`<jsp:setProperty name="cal" property="*" />`**: Dòng mã lệnh này được sử dụng để thiết lập giá trị cho tất cả các thuộc tính từ các tham số cùng tên
- ✓ **`${cal.toanHangA}`, `${cal.toanHangB}`, `${cal.ketQua}`**: Các dòng mã lệnh này được sử dụng để truy xuất và hiển thị các thuộc tính trong bean cal

Ví dụ 2: Sử dụng lớp JavaBean SinhVien để tiếp nhận và hiển thị thông tin từ form nhập sau đây. Ví dụ này cung cấp cho bạn cách tiếp nhận dữ liệu nhiều giá trị (sở thích) và xử lý đa dạng kiểu dữ liệu trong JavaBean.





Mã sinh viên:

Họ và tên:

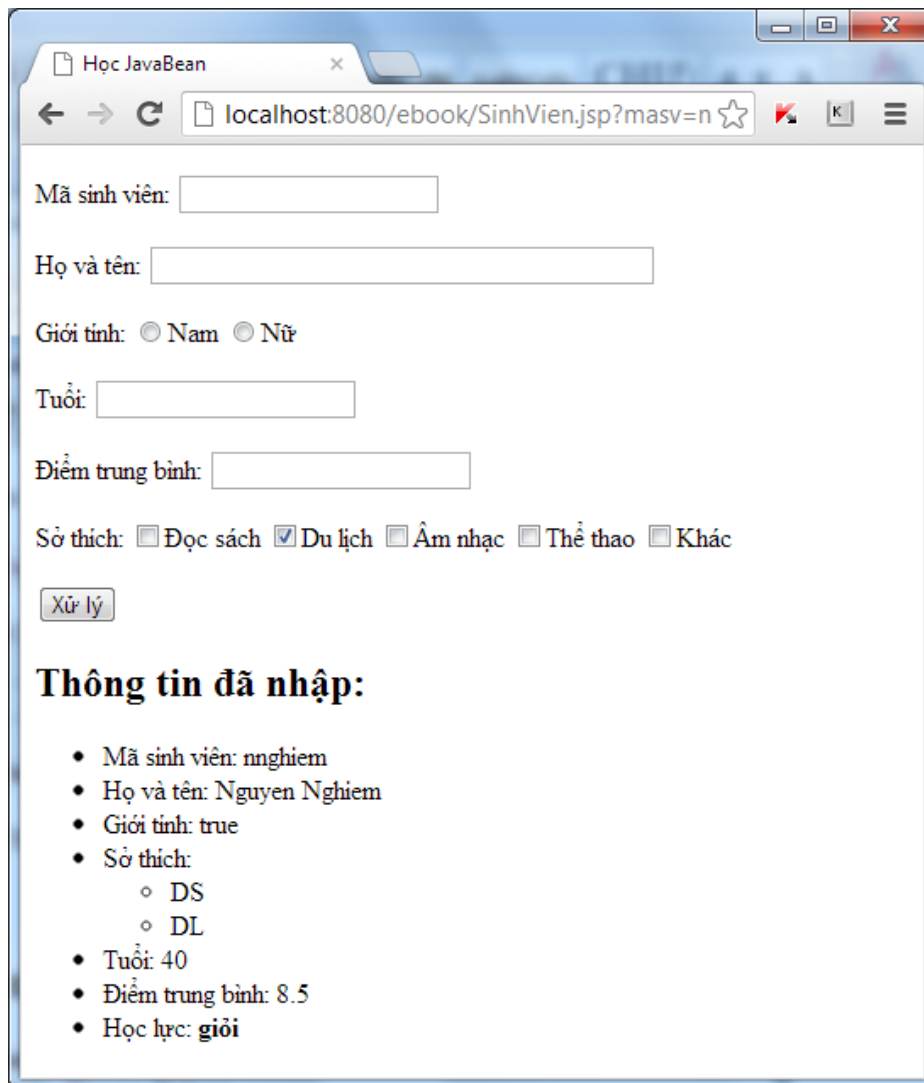
Giới tính: ☒ Nam ☐ Nữ

Tuổi:

Điểm trung bình:

Sở thích: ☒ Đọc sách ☒ Du lịch ☐ Âm nhạc ☐ Thể thao ☐ Khác

Sau khi nhập thông tin và nhấp nút Xử lý, bạn sẽ nhận được phần thông tin đã nhập ở phía dưới.



Mã sinh viên:

Họ và tên:

Giới tính: ☐ Nam ☐ Nữ

Tuổi:

Điểm trung bình:

Sở thích: ☐ Đọc sách ☒ Du lịch ☐ Âm nhạc ☐ Thể thao ☐ Khác

**Thông tin đã nhập:**

- Mã sinh viên: nnghiem
- Họ và tên: Nguyen Nghiem
- Giới tính: true
- Sở thích:
  - DS
  - DL
- Tuổi: 40
- Điểm trung bình: 8.5
- Học lực: **giỏi**

Mã JSP của trang SinhVien.jsp như sau

```
<%@ page contentType="text/html; charset=utf-8" pageEncoding="utf-8"%>
<%@ taglib uri="http://java.sun.com/jstl/core_rt" prefix="c" %>
<!DOCTYPE html>
<html>
<head>
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
 <title>Học JavaBean</title>
</head>
<body>
<form action="SinhVien.jsp">
 <p>Mã sinh viên:
 <input name="masv">
 <p>Họ và tên:
 <input name="hoTen" size="44">
 <p>Giới tính:
 <input type="radio" name="gioiTinh" value="true">Nam
 <input type="radio" name="gioiTinh" value="false">Nữ
 <p>Tuổi:
 <input name="tuoi">
 <p>Điểm trung bình:
 <input name="diem">
 <p>Sở thích:
 <input type="checkbox" name="soThich" value="DS">Đọc sách
 <input type="checkbox" name="soThich" value="DL" checked="">Du lịch
 <input type="checkbox" name="soThich" value="AN">Âm nhạc
 <input type="checkbox" name="soThich" value="TT">Thể thao
 <input type="checkbox" name="soThich" value="KH">Khác
 <p>
 <input type="submit" name="btnXuLy" value="Xử Lý">
 <c:if test="${! empty param['btnXuLy']}">
 <jsp:useBean id="sv" class="nnghiem.bean.SinhVien" scope="page"/>
 <jsp:setProperty name="sv" property="*" />

 <h2>Thông tin đã nhập:</h2>

 Mã sinh viên: ${sv.masv}
 Họ và tên: ${sv.hoTen}
 Giới tính: ${sv.gioiTinh}
 Sở thích:

 <c:forEach var="st" items="${sv.soThich}">
 ${st}
 </c:forEach>

 Tuổi: ${sv.tuoi}
 Điểm trung bình: ${sv.diem}
 Học lực: ${sv.hocLuc}

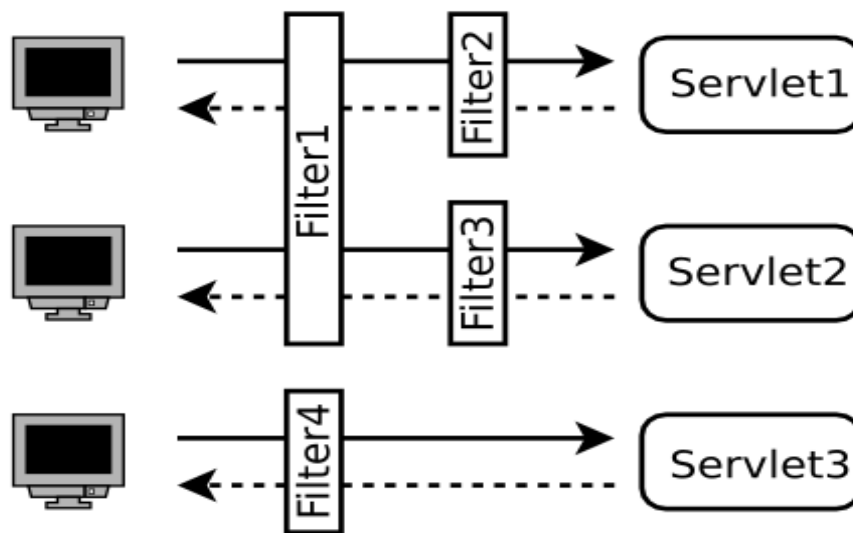
 </c:if>
</form>
</body>
</html>
```

## 4.7 Filter & Listener

### 4.7.1 Filter

Bộ lọc là thành phần quan trọng được sử dụng để xử lý các yêu cầu trước khi đến servlet/jsp và sau khi servlet đã thực hiện.

Lọc là kỹ thuật lập trình mạnh mẽ cho các nhà phát triển servlet, nó được xem là một phần tách rời của nhiều servlet/jsp để thực hiện một công việc chung giữ chúng. Việc lắp ráp và tháo rời theo kiểu Filter sẽ giúp thiết kế ứng dụng mềm dẻo hơn, linh động hơn.



Hình: Mô hình lọc yêu cầu

#### 4.7.1.1 Tạo lớp Filter

Tạo lớp Java thực thi theo giao tiếp `javax.servlet.Filter` và cài đặt mã nguồn cho các phương thức `init()`, `destroy()` và `doFilter()` để thực hiện các công việc mà bạn mong muốn.

Ví dụ sau đây định nghĩa một Filter thường được sử dụng để thiết lập chế độ mã hóa ký tự UTF-8 cho các yêu cầu và hồi đáp của các jsp/servlet.

```
package nhgkiem.filter;

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

@WebFilter(
 urlPatterns = {"//*"},
 dispatcherTypes = {DispatcherType.REQUEST}
)
public class EncodingFilter implements Filter {

 @Override
 public void doFilter(ServletRequest request, ServletResponse response,
 FilterChain chain) throws IOException, ServletException {
 request.setCharacterEncoding("utf-8");
 }
}
```

```
response.setCharacterEncoding("utf-8");
chain.doFilter(request, response);
}

@Override
public void init(FilterConfig config) throws ServletException {}

@Override
public void destroy() {}
}
```

#### 4.7.1.2 Vòng đời của Filter

✓ Phương thức **init()**

Chạy sau khi Filter được nạp vào bộ nhớ để sẵn sàng phục vụ. Mã nguồn viết ở đây thường chuẩn bị các tài nguyên cần thiết cho filter.

✓ Phương thức **destroy()**

Chạy trước khi Filter bị giải phóng khỏi bộ nhớ. Mã nguồn đặt ở đây thường mang tính thu gom, làm sạch các tài nguyên đã sử dụng trong filter.

✓ Phương thức **doFilter()**

Chạy mỗi lần có yêu cầu thỏa mãn các urlPatterns và kiểu yêu cầu đến JSP/servlet. Nếu trong phương thức này không gọi lại **chain.doFilter(request, response);** thì yêu cầu sẽ không được chuyển tiếp đến đích của yêu cầu. Mã đặt trước lời gọi **chain.doFilter(request, response);** sẽ được thực thi trước khi đến đích, còn mã đặt sau **chain.doFilter(request, response);** sẽ được thực thi sau khi đích yêu cầu đã phục vụ xong.

#### 4.7.1.3 Ánh xạ Filter

Nếu một lớp Java chỉ implements Filter thì các phương thức chưa thể tự chạy để lọc các yêu cầu theo mong muốn được mà bạn phải ánh xạ bộ lọc phù hợp với điều kiện lọc cụ thể. Việc ánh xạ này xảy ra theo hai cách là XML và annotation.

##### ❖ Ánh xạ bộ lọc bằng XML

Ví dụ sau đây ánh xạ EncodingFilter được sử dụng để lọc tất cả các yêu cầu đến website. Việc ánh xạ này được thực hiện trong tập tin cấu hình ứng dụng web WEB-INF/web.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID"
version="3.0">
 <display-name>eStore</display-name>

 <filter>
 <filter-name>utf8</filter-name>
 <filter-class>nnghiem.filter.EncodingFilter</filter-class>
```

```
</filter>

<filter-mapping>
 <filter-name>utf8</filter-name>
 <url-pattern>/*</url-pattern>
 <dispatcher>REQUEST</dispatcher>
</filter-mapping>

</web-app>
```

- ✓ Thẻ <filter> được sử dụng để khai báo lớp Filter <filter-class>EncodingFilter</filter-class> và đặt cho nó một cái tên utf8 <filter-name>utf8</filter-name>
- ✓ Thẻ <filter-mapping> được sử dụng để ánh xạ filter có tên utf8 <filter-name>utf8</filter-name> được sử dụng để lọc tất cả <url-pattern>/\*</url-pattern> các yêu cầu <dispatcher>REQUEST</dispatcher>
  - Giá trị của <dispatcher> có thể là REQUEST, INCLUDE, FORWARD tùy thuộc vào loại yêu cầu cần lọc (yêu cầu từ người dùng, bao hàm hay chuyển tiếp từ servlet khác)
  - <url-pattern> mẫu url cần lọc. Có thể là một trang jsp, servlet hay sử dụng ký tự đại diện \* để xây dựng mẫu.

#### ❖ Ánh xạ bộ lọc bằng annotation

Sau đây là ví dụ tương đương với phần cấu hình XML nhưng bằng cách sử dụng annotation. Việc cấu hình này được viết ngay trong Filter.

```
@WebFilter(
 urlPatterns = {"//*"},
 dispatcherTypes = {DispatcherType.REQUEST}
)

public class EncodingFilter implements Filter {...}
```

- ✓ @WebFilter: được sử dụng để cấu hình lớp Filter
- ✓ urlPatterns = {"//\*"}: được sử dụng để khai báo các mẫu url cần lọc
- ✓ dispatcherTypes = {DispatcherType.REQUEST}: được sử dụng để khai báo các thể loại yêu cầu cần lọc.

#### 4.7.1.4 Bộ lọc HitCounter

Sau đây là ví dụ về bộ lọc đếm số lần click của tất cả các tài nguyên được truy xuất. Trong ví dụ này sử dụng tập tin Properties để tổ chức lưu trữ số lần truy xuất cho các tài nguyên.

```
package nnghiem.filter;

import ...

@WebFilter(
 urlPatterns = {"//*"},
 dispatcherTypes = {DispatcherType.REQUEST}
)

public class HitCounterFilter implements Filter{
```

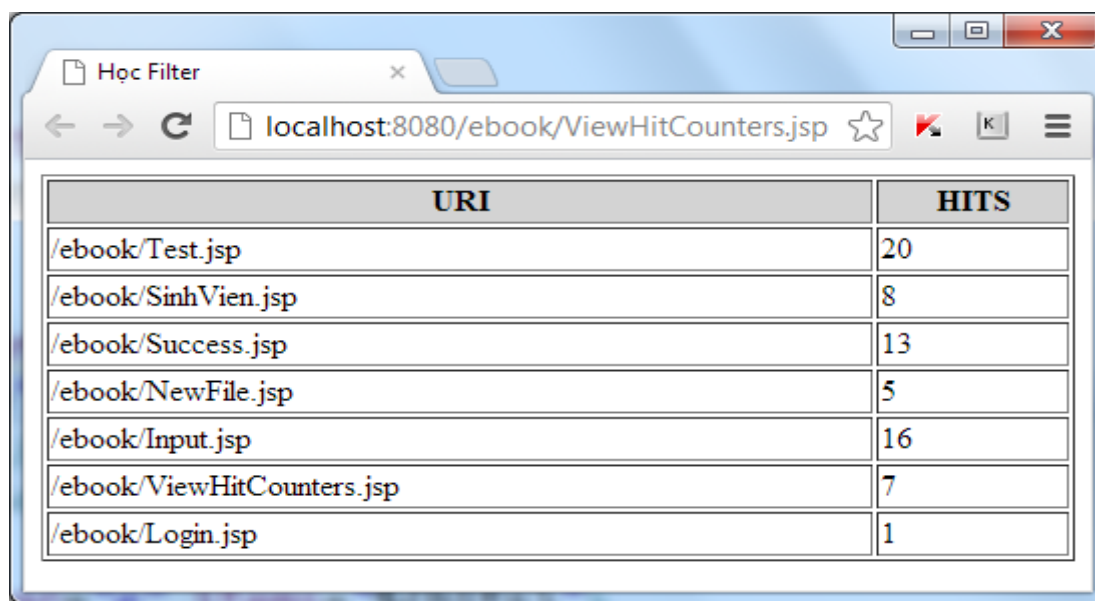
```
String fileName = "counters.properties";
Properties counters = new Properties();
/**
 * Sau khi Filter sẵn sàng -> tải danh sách số lần truy xuất
 * của các tài nguyên trong tập tin counters.properties của
 * các trang web vào Properties
 */
@Override
public void init(FilterConfig config) throws ServletException {
 // lấy đường dẫn vật lý của file counters.properties
 ServletContext context = config.getServletContext();
 fileName = context.getRealPath(fileName);
 try{
 // tải các số đếm cũ vào Properties
 counters.load(new FileInputStream(fileName));
 }
 catch (Exception e) {
 e.printStackTrace();
 }
 // chia sẻ hit counters trên phạm vi application
 context.setAttribute("hits", counters);
}
/**
 * Trước khi Filter bị giải phóng khỏi bộ nhớ -> lưu danh
 * sách số lần truy xuất các tài nguyên trong website vào
 * tập tin counters.properties
 */
@Override
public void destroy() {
 try{
 counters.store(new FileOutputStream(fileName), "counters");
 }
 catch (Exception e) {
 e.printStackTrace();
 }
}
/**
 * Cứ mỗi lần truy xuất -> Tăng số lần truy xuất
 * của tài nguyên được yêu cầu lên 1
 */
@Override
public void doFilter(ServletRequest request, ServletResponse response,
 FilterChain chain) throws IOException, ServletException {

 // lấy địa chỉ của tài nguyên bị truy xuất
 HttpServletRequest req = (HttpServletRequest) request;
 String uri = req.getRequestURI();

 // tăng số lần truy xuất của tài nguyên lên một
 int hits = Integer.parseInt(counters.getProperty(uri, "0"));
 counters.setProperty(uri, String.valueOf(hits + 1));
}
```

```
// chuyển đến thành phần tiếp theo
chain.doFilter(request, response);
}
}
```

Sau khi hoàn thành Filter, bạn có thể truy xuất đến bất cứ tài nguyên nào trong website. Filter sẽ giúp bạn ghi nhận số lượt truy xuất của các tài nguyên đó. Nếu bạn muốn xem số lần truy xuất như giao diện sau đây thì chỉ cần hiển thị bộ đếm (Properties) đã được chia sẻ với cái tên hits tại dòng mã `context.setAttribute()` cuối phương thức `init()`



URI	HITS
/ebook/Test.jsp	20
/ebook/SinhVien.jsp	8
/ebook/Success.jsp	13
/ebook/NewFile.jsp	5
/ebook/Input.jsp	16
/ebook/ViewHitCounters.jsp	7
/ebook/Login.jsp	1

Mã JSP của trang ViewHitCounters.jsp như sau

```
<%@ page contentType="text/html; charset=utf-8" pageEncoding="utf-8"%>
<%@ taglib uri="http://java.sun.com/jstl/core_rt" prefix="c" %>

<!DOCTYPE html>
<html>
<head>
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
 <title>Học Filter</title>
</head>
<body>
<table border="1" width="100%">
<tr bgcolor="lightgray">
 <th>URI</th>
 <th>HITS</th>
</tr>
<c:forEach var="e" items="${hits}">
<tr>
 <td>${e.key}</td>
 <td>${e.value}</td>
</tr>
</c:forEach>
</table>
</body>
</html>
```

## 4.7.2 Listener

Bạn có thể viết ra các thành phần để nhận biết các biến cố xảy ra trong lòng ứng dụng web với Java một cách dễ dàng chỉ việc thực thi một số interface đã được định nghĩa sẵn trong Java.

Các biến cố thường xuyên cần được theo dõi là vòng đời của ứng dụng, vòng đời của session, vòng đời của request. Tuy nhiên vòng đời của request đã được xử lý bởi Filter nên trong phần này chỉ trình bày theo dõi vòng đời của ứng dụng và session.

### 4.7.2.1 Theo dõi vòng đời của ứng dụng web

Để theo dõi vòng đời của ứng dụng (tức bạn muốn biết khi nào ứng dụng bắt đầu, khi nào ứng dụng kết thúc) bạn chỉ cần viết một lớp và thực thi theo interface `ServletContextListener` sau đó khai báo để hệ thống nhận biết lớp của bạn là một listener.

```
package nhgkiem.listener;

import javax.servlet.*;

public class ApplicationListener implements ServletContextListener {
 /**
 * Biến cố này xảy ra ngay trước khi ứng dụng kết thúc.
 */
 @Override
 public void contextDestroyed(ServletContextEvent e) {
 // lấy phạm vi chia sẻ toàn ứng dụng
 ServletContext application = e.getServletContext();
 //...mã xử lý...
 }
 /**
 * Biến cố này xảy ra ngay sau khi ứng dụng được khởi động
 */
 @Override
 public void contextInitialized(ServletContextEvent e) {
 // lấy phạm vi chia sẻ toàn ứng dụng
 ServletContext application = e.getServletContext();
 //...mã xử lý...
 }
}
```

- ✓ Dựa vào biến cố `contextDestroyed()` bạn có thể cài đặt mã để thực hiện thu gom hoặc lưu giữ các công việc cuối cùng trước khi ứng dụng kết thúc hoàn toàn.
- ✓ Dựa vào biến cố `contextInitialized()` bạn có thể cài đặt mã để thực hiện công việc chuẩn bị tài nguyên cho toàn ứng dụng. Các công việc thông thường là cấu hình hệ thống, chuẩn bị dữ liệu cache...

### 4.7.2.2 Theo dõi vòng đời phiên làm việc

Lập trình web thường xuyên cần biết khi nào một phiên làm việc được tạo ra hay kết thúc. Nắm được những biến cố này sẽ giúp bạn triển khai một số công việc liên quan đến từng người dùng. Ví dụ chuẩn bị giỏ hàng, tăng số lượt truy cập website, tăng hoặc giảm số người đang trực tuyến...



Để thực hiện được điều này, bạn chỉ cần thực thi interface `HttpSessionListener` và cài đặt mã nguồn cho các phương thức qui định bởi interface để thực hiện công việc mong muốn.

```
package nhghiem.listener;

import javax.servlet.http.*;

public class SessionListener implements HttpSessionListener{
 /**
 * Biến cố này xảy ra ngay sau khi một
 * phiên làm việc mới bắt đầu
 */
 @Override
 public void sessionCreated(HttpSessionEvent e) {
 // Lấy đối tượng session
 HttpSession session = e.getSession();
 //...công việc...
 }
 /**
 * Biến cố này xảy ra ngay trước khi một phiên
 * làm việc hết hạn sử dụng
 */
 @Override
 public void sessionDestroyed(HttpSessionEvent e) {
 // Lấy đối tượng session
 HttpSession session = e.getSession();
 //...công việc...
 }
}
```

Dựa vào biến cố `sessionCreated()` bạn có thể cài đặt mã để chuẩn bị dữ liệu cần thiết cho mỗi phiên làm việc hoặc những công việc liên quan đến số phiên làm việc

Dựa vào biến cố `sessionDestroyed()` bạn có thể cài đặt mã để thu gom, giải phóng hoặc cất giữ dữ liệu liên quan đến mỗi phiên làm việc mà trước đó đã chuẩn bị hay quá trình làm việc sinh ra.

#### 4.7.2.3 Theo dõi vòng đời phối hợp

Bạn có thể cài đặt mã nguồn theo dõi các biến cố ứng dụng và session trong cùng một lớp Java bằng cách thực thi theo 2 interface `HttpSessionListener` và `ServletContextListener`. Bằng cách này bạn sẽ thực hiện được các công việc phối hợp.

Sau đây là ví dụ xây dựng bộ đếm số lượt truy xuất website. Cứ mỗi phiên làm việc chỉ được phép đếm một lần (mỗi khi refresh hay truy xuất nhiều trang khác nhau trong phiên làm việc đó sẽ không được đếm lại).

```
package nhghiem.listener;

import ...

public class SessionListener
 implements HttpSessionListener, ServletContextListener{
```



```

String fileName = "config.properties";
Properties config = new Properties();

/**
 * Tải tập tin cấu hình để tiếp tục thực hiện công việc
 * trong suốt quá trình thực hiện của ứng dụng.
 * Chia sẻ thông tin cấu hình để các thành phần khác cùng làm việc
 */
@Override
public void contextInitialized(ServletContextEvent e) {
 // lấy đường dẫn vật lý của tập tin config.properties
 ServletContext context = e.getServletContext();
 fileName = context.getRealPath(fileName);
 try{
 // tải thông tin cấu hình từ file config.properties
 config.load(new FileInputStream(fileName));
 }
 catch (Exception ex) {
 ex.printStackTrace();
 }
 context.setAttribute("app", config);
}

/**
 * Tăng số lượt truy cập lên một mỗi khi một phiên làm việc mới
 * bắt đầu. Sau đó lưu số đếm số lượt truy xuất vào file cấu hình.
 * Giỏ hàng điện tử cũng được chuẩn bị để người dùng duy trì
 * hàng hóa trong quá trình chọn mua trong phiên làm việc
 */
@Override
public void sessionCreated(HttpSessionEvent e) {
 // tham chiếu đến đối tượng session
 HttpSession session = e.getSession();

 // tăng số lượt viếng thăm lên 1
 int visitors = Integer.parseInt(config.getProperty("visitors"));
 config.setProperty("visitors", String.valueOf(visitors + 1));

 try{
 // lưu số lượt viếng thăm vào file
 config.store(new FileOutputStream(fileName), "web.config");
 }
 catch (Exception ex) {
 ex.printStackTrace();
 }

 // khởi tạo và chia sẻ giỏ hàng để người dùng duy trì hàng hóa
 ShoppingCart cart = new ShoppingCart();
 session.setAttribute("cart", cart);
}

@Override
public void sessionDestroyed(HttpSessionEvent e) {
}

@Override

```

```
public void contextDestroyed(ServletContextEvent e) {
}
}
```

## 4.8 Bài tập

1. Nhập 2 số vào form, chương trình sẽ thực hiện phép cộng và hiển thị kết quả như hình dưới đây.

Toán hạng 1:

Toán hạng 2:

Kết quả: 6 + 3 = 9

Nhấp nút Cộng

Toán hạng 1:

Toán hạng 2:

Kết quả: 6 - 3 = 3

Nhấp nút Trừ

Hướng dẫn:

- ✓ Nhận các toán hạng từ tham số form
  - ✓ Chuyển đổi thành số thực và thực hiện phép cộng
  - ✓ Lưu kết quả vào thuộc tính request (setAttribute())
  - ✓ Chuyển về trang JSP nơi có chứa chuỗi "Kết quả: \${param.th1} + \${param.th2} = \${ketQua}" để hiển thị kết quả như trên.
  - ✓ Thêm các nút [Trừ][Nhân] và [Chia] vào giao diện và nâng cấp mã để có thể thực hiện được các phép tính đơn giản.
2. Nhập vào một chữ số, chương trình xuất chữ (tên gọi của chữ số đó)

Số:

Chữ: năm

Hướng dẫn

- ✓ Nhận chữ số từ tham số form
  - ✓ Sử dụng if...else if...else hoặc switch...case để xử lý
  - ✓ Chuyển tiếp về JSP để hiển thị
  - ✓ Nâng cấp bài để có thể đọc được số có 3 chữ số
3. Tạo một từ điển trực tuyến với số lượng các từ vừa phải và cho phép tra cứu trực tuyến.

Từ:

Nghĩa: trường học

Hướng dẫn

- ✓ Sử dụng tập tin properties để lưu từ điển

✓ Sử dụng phương thức getProperty(tu, "unknown") để tra cứu từ

✓ Nâng cấp để có thể thêm từ mới vào từ điển

4. Tính bonus cho nhân viên. Bonus cho nhân viên phụ thuộc vào tuổi. Cụ thể như sau

Tuổi	< 30	31..40	> 40
Mức bonus	5% lương	7% lương	10% lương

Nhân viên:

Tuổi:

Lương:

Bonus: Kết quả bonus hiển thị ở đây

Hướng dẫn:

✓ Nhận thông tin nhân viên từ tham số form nhập

✓ Chuyển lương sang số thực, tuổi sang số nguyên và xét độ tuổi để tính bonus

✓ Lưu kết quả vào attribute của request

✓ Chuyển tiếp về JSP để hiển thị kết quả: "Bonus: \${bonus}"

5. Giải phương trình bậc nhất  $ax + b = 0$ . Trong đó các hệ số a và b được nhập từ người dùng.

Hệ số a:

Hệ số b:

Kết quả:

Hướng dẫn:

✓ Nhận các hệ số a và b từ tham số form nhập

✓ Chuyển sang số thực, biện luận và giải phương trình để tìm nghiệm

○  $A=0$  và  $B \neq 0$ : Vô nghiệm

○  $A=0$  và  $B=0$ : Vô số nghiệm

○ 1 nghiệm duy nhất  $x = B/A$

6. Kiểm tra 3 số được nhập từ người dùng có phù hợp với độ dài của một tam giác hay không.

Độ dài cạnh thứ nhất: Độ dài cạnh thứ hai: Độ dài cạnh thứ ba: 

Kết quả:

Hướng dẫn

- ✓ Nhận 3 số từ tham số form nhập và chuyển sang số thực
  - ✓ Kiểm tra 3 số a, b, c nhận được có thỏa mãn điều kiện là tổng 2 số luôn luôn lớn hơn số còn lại hay không.
  - ✓ Lưu thông báo hợp lệ hay không hợp lệ vào attribute của request và chuyển tiếp sang JSP để hiển thị: "Kết quả:  $\${thongBao}$ "
  - ✓ Cho biết đó là tam giác gì: cân, đều, vuông hay thường
7. Giải phương trình bậc 2 ( $ax^2 + bx + c = 0$ ). Trong đó các hệ số phương trình a, b và c được nhập từ người dùng.

Hệ số a: Hệ số b: Hệ số c: 

Kết quả:

Hướng dẫn:

- ✓ Nhận các hệ số a, b và c từ tham số form và chuyển sang số thực
- ✓ Tính  $\Delta = b^2 - 4ac$
- ✓ Biện luận và giải phương trình
  - ( $\Delta < 0$ ): Vô nghiệm
  - ( $\Delta = 0$ ): Nghiệm kép
  - ( $\Delta > 0$ ): 2 Nghiệm phân biệt
    - $X1 = (-b + \text{Math.sqrt}(\Delta)) / (2 * a);$
    - $X2 = (-b - \text{Math.sqrt}(\Delta)) / (2 * a);$

8. Xếp loại học lực. Nhập vào điểm toán, lý và hóa, chương trình tính điểm trung bình và hiển thị học lực của học viên đó. Học lực được tính theo điểm trung bình theo qui luật sau:

Điểm trung bình	< 5	5 -> 6,5	6.6 -> 7.5	7.6 -> 9	> 9
Học lực	Yếu	Trung bình	Khá	Giỏi	Xuất sắc

Điểm toán:

Điểm lý:

Điểm hóa:

Điểm trung bình: ???

Xếp loại: ???

Hướng dẫn:

- ✓ Nhận điểm các môn từ tham số form nhập, chuyển sang số thực và tính điểm trung bình
- ✓ Dùng cấu trúc if...else if...else để xếp loại học lực và lưu vào attribute của request
- ✓ Chuyển tiếp về JSP để hiển thị kết quả
- ✓ Điểm trung bình: `${diemTrungBinh}`
- ✓ Xếp loại: `${xepLoai}`

Tính diện tích hình chữ nhật. Nhập chiều dài và chiều rộng. Chương trình tính và hiển thị diện tích của hình chữ nhật.

Chiều dài:

Chiều rộng:

Diện tích hình chữ nhật là ???

Hướng dẫn

- ✓ Nhận chiều dài và chiều rộng từ người dùng và chuyển sang số thực
- ✓ Tính diện tích và lưu kết quả vào thuộc tính request
- ✓ Chuyển tiếp đến JSP để hiển thị: "Diện tích hình chữ nhật là: `${dienTich}`"

9. Tính thuế thu nhập. Nhập lương, chương trình sẽ tính và hiển thị thuế thu nhập. Thuế thu nhập được tính theo luật lũy tiến như sau:

Lương	< 10 triệu	10 -> 20 triệu	> 20 triệu
Mức thuế	10%	15%	20%

Họ và tên: Lương: 

Thuế thu nhập của nhân viên ??? là ???

10. Đăng nhập đơn giản. Mã tài khoản và mật khẩu được nhập vào, chương trình kiểm tra xem mã đăng nhập và mật khẩu sau đó đưa ra thông báo phù hợp.

Mã đăng nhập: Mật khẩu: 

Thông báo đăng nhập

Hướng dẫn

- ✓ Nhận thông tin tài khoản từ tham số form nhập
- ✓ Nếu mã đăng nhập có chứa chuỗi user và độ dài của mật khẩu trên 6 ký tự thì xem như đăng nhập thành công.
- ✓ Lưu chuỗi thông báo vào thuộc tính request và chuyển tiếp đến JSP để hiển thị thông báo: `#{thongBao}`

11. Tìm số lớn nhất, nhỏ nhất của các số \$a, \$b, \$c, \$d, \$e nhập từ người dùng.

A: B: C: D: E: 

MIN = ???

MAX = ???

Hướng dẫn

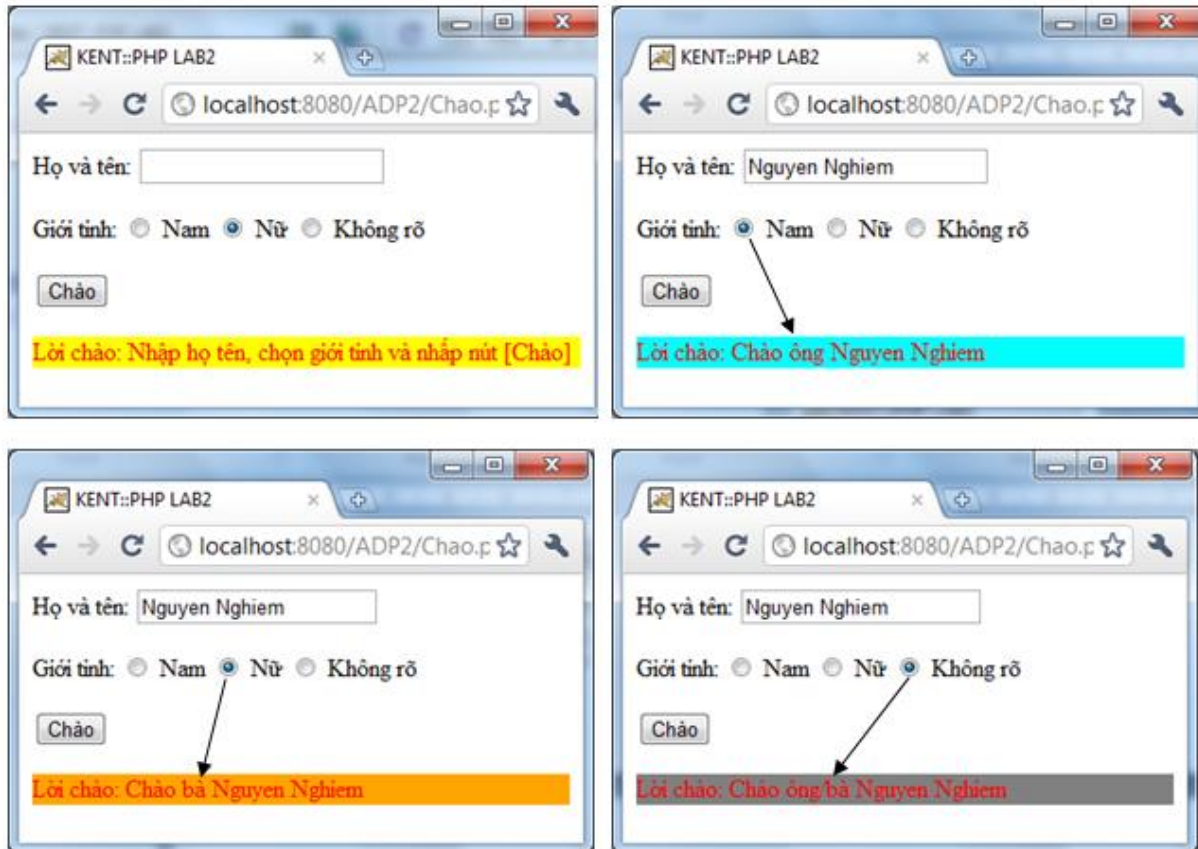
- ✓ Nhận các giá trị từ tham số người dùng và chuyển đổi sang số thực
- ✓ Tiến hành so sánh 2 số, lấy số lớn so sánh với một số khác trong nhóm số còn lại
- ✓ Lặp lại công việc cho đến khi hết các số so sánh
- ✓ Làm tương tự với số nhỏ nhất

- ✓ Lưu các số nhỏ nhất và lớn nhất vào request và chuyển tiếp sang JSP để xem kết quả

MAX = \${max}

MIN = \${min}

12. Nhập họ tên và chọn giới tính vào form nhập, chương trình sẽ đưa ra lời chào phù hợp được mô tả như các hình vẽ sau.



Chú ý sự thay đổi tên, xưng hô (ông, bà hay ông/bà) và màu nền sắc nơi hiển thị lời chào.

Hướng dẫn:

- ✓ Đọc lấy họ tên và giới tính từ tham số form
- ✓ Phân biệt 3 trường hợp chọn (dùng if...else if...else) để tạo biến chứa màu
- ✓ Sử dụng setAttribute() để lưu các biến vào thuộc tính request
- ✓ Chuyển trở lại trang JSP
- ✓ Ở trang JSP sẽ hiện lời chào: `<div style="background-color:${mau}">Chao ${xungHo} ${ten}</div>`
- ✓ Chạy và khảo sát mã nguồn: click chuột phải và chọn view page source để xem mã sinh ra

13. Nhập vào họ tên, tuổi và chọn giới tính sau đó xuất chuỗi thông báo như hình sau.



Ho ten: Tuoi: 

Gioi tinh:

☒ Nam☐ Nữ

Mr. Nguyễn Nghiệm la 21.00 tuổi

Hướng dẫn: tương tự bài trên

14. Viết và sinh bảng nhân của một số nhập từ người dùng.

N:  

8 x 1	=	8
8 x 2	=	16
8 x 3	=	24
8 x 4	=	32
8 x 5	=	40
8 x 6	=	48
8 x 7	=	56
8 x 8	=	64
8 x 9	=	72
8 x 10	=	80

Hướng dẫn:

- ✓ Nhận số cần sinh bảng nhân N từ form nhập
- ✓ Sử dụng vòng lặp for chạy từ 1 đến 10
- ✓ Nội dung vòng for là xuất một hàng của bảng "`<tr> <td> ${N}x${i} </td> <td> = </td> <td> ${i*N} </td> </tr>`". Trong đó bạn phải thay `${N}` bởi giá trị N nhận được và `${i}` là giá trị của biến chạy trên vòng lặp.
- ✓ Chạy và nhập các số từ 2 đến 9 để kiểm tra kết quả.

15. Tính trung bình cộng các số chia hết cho 3 từ Min cho đến Max. Trong đó Min, Max nhập từ người dùng.

Min:

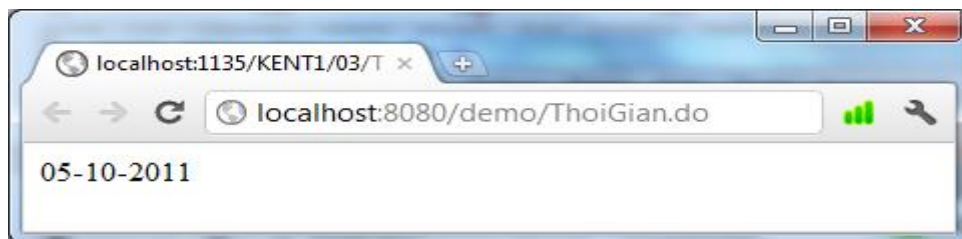
Max:

Average: 15

Hướng dẫn:

- ✓ Nhận các số min, max từ tham số form nhập
- ✓ Chuyển đổi sang kiểu số nguyên
- ✓ Khai báo biến chứa tổng và số đếm các số chia hết cho 3
- ✓ Dùng vòng lặp for có biến chạy từ min đến max
- ✓ Mỗi lần lặp cộng dồn vào biến tổng và tăng số đếm nếu biến chạy có giá trị chia hết cho 3
- ✓ Lấy tổng chia cho số đếm sẽ cho kết quả trung bình cộng
- ✓ Lưu vào attribute của request và quay trở lại trang JSP có chứa: "Average: \${ketQua}" để xem kết quả

16. Hiện thị ngày hiện tại dưới dạng



Hướng dẫn: Sử dụng phương thức format() của SimpleDateFormat và định dạng dd-MM-yyyy

17. Nhập ngày tháng năm, hiện thị ngày trong tuần (weekday) và số ngày trong tháng đó.

NGÀY THÁNG	
Ngày :	<input type="text" value="2010-04-07"/> <input type="button" value="Tính"/>
Thứ :	?
Số ngày :	?

Hướng dẫn

- ✓ Nhận chuỗi từ tham số form nhập
- ✓ Sử dụng phương thức parse của lớp SimpleDateFormat để chuyển sang Date
- ✓ Sử dụng phương thức format() của SimpleDateFormat để lấy tên thứ (ngày trong tuần) của ngày nhập vào
- ✓ Kiểm tra tháng và năm (có nhuận hay không?) để biết số ngày trong tháng

18. Hãy xử lý chuỗi theo yêu cầu sau. Nhập vào chuỗi bất kỳ, xử lý và xuất ra như yêu cầu trên form nhập.

Nhập chuỗi:

Kiểm tra

Số ký tự:  $\$ \{soKyTu\}$

Số từ:  $\$ \{soTu\}$

Chuỗi con từ ký tự thứ 5 đến 20:  $\$ \{chuoiCon\}$

Chuỗi "Việt Nam" xuất hiện tại vị trí:  $\$ \{viTriVN\}$

Chuỗi kết thúc bởi chuỗi "kết thúc" hay không?  $\$ \{ketThuc\}$

Hướng dẫn:

- ✓ Nhận chuỗi từ tham số form nhập
- ✓ Sử dụng phương thức `length()` để biết số ký tự
- ✓ Sử dụng phương thức tách chuỗi `split("[\\s,;.]+")` để tách chuỗi theo các ký tự phân cách là ký tự trống (`\s` gồm về đầu dòng, xuống dòng mới, ký tự tab, khoảng trắng), dấu phẩy (`,`), dấu chấm phẩy (`;`), dấu chấm (`.`) thành mảng. Sau đó dùng thuộc tính `length` của mảng để biết số phần tử, cũng là số từ.
- ✓ Sử dụng phương thức `substring(5, 20)` để lấy chuỗi con
- ✓ Sử dụng phương thức `indexOf("Việt Nam")` để lấy vị trí xuất hiện của chuỗi "Việt Nam"
- ✓ Sử dụng phương thức `endsWith("kết thúc")` để biết chuỗi có kết thúc bởi chuỗi "kết thúc" hay không.
- ✓ Lưu các kết quả vào các biến phạm vi request và chuyển tiếp sang JSP để xem kết quả

19. Tìm kiếm và thay thế chuỗi

Tìm:

Thay:

Nội dung:

```
Cong cha nhu nui thai song
Nghia me nhu nuoc trong nguon chay ra
```

Thay the

**Hướng dẫn**

- ✓ Nhận các chuỗi nội dung, tìm kiếm và thay thế từ tham số form
- ✓ Sử dụng phương thức `replaceAll("thai son", "Thai Son")` để thay thế chuỗi
- ✓ Lưu kết quả đã thay thế vào phạm vi request và chuyển tiếp sang JSP để hiển thị lại lên ô nội dung

**20. Nhập dãy số từ form xử lý theo yêu cầu trên form**

**Nhập dãy số:**

Xử lý

- Số lượng các phần tử mảng: `soPhanTu`
- Phần tử lớn nhất, nhỏ nhất: `soLanNhat`, `soNhoNhat`
- Trung bình cộng các số chẵn: `trungBinh`
- Phần tử ngẫu nhiên: `soNgauNhien`
- Đảo và xuất mảng: `mangDao`
- Sắp xếp tăng dần và xuất mảng: `mangGiamDan`

**Hướng dẫn:**

- ✓ Nhận chuỗi từ tham số form
- ✓ Sử dụng hàm `split("[\\s;,,]")` để tách thành mảng các chuỗi chứa số.
- ✓ Chuyển mảng chuỗi thành mảng số để thực hiện các yêu cầu
- ✓ Thuộc tính `length` của mảng cho biết số phần tử của mảng (tức của dãy số)
- ✓ Sử dụng vòng lặp `for` duyệt qua các phần tử của mảng để tìm số lớn nhất, số nhỏ nhất, tổng và số lượng các số chẵn sau đó tính trung bình cộng
- ✓ Sử dụng hàm `Math.round((số phần tử)*Math.random())` để lấy vị trí phần tử ngẫu nhiên trong mảng. Từ đó có được giá trị của phần tử trong mảng.
- ✓ Sử dụng hàm `Arrays.sort()` và `Arrays.reverse()` để sắp xếp và đảo các phần tử trong mảng.
- ✓ Lưu kết quả tính toán vào các biến phạm vi và chuyển tiếp sang JSP để hiển thị kết quả.

**21. Nhập vào chuỗi và xuất ra danh sách các từ trong chuỗi**

Nhap chuỗi:

- Cong
- cha
- nhu
- nui
- thai
- son

Hướng dẫn: Sử dụng phương thức tách chuỗi split("[ ]+") để tách chuỗi theo các ký tự trắng thành mảng chứa các từ. Dùng vòng lặp for để duyệt và xuất các phần tử của mảng.

22. Xây dựng form bầu chọn website gồm 5 mức

☐ Tuyệt vời ☒ Rất tốt ☐ Tốt ☐ Bình thường ☐ Tồi

Kết quả bình chọn

- Tuyệt vời: ???
- Rất tốt: ???
- Tốt: ???
- Bình thường: ???
- Tồi: ???

Hướng dẫn

- ✓ Sử dụng tập tin Properties để lưu kết quả bình chọn
- ✓ Nhận mục chọn và tăng số lần của mục tương ứng trong Properties

23. Kiểm lỗi dữ liệu đầu vào cho form nhập sau đây. Trong đó Ten không được để trống, tuổi phải là số từ 16 đến 65 và email phải đúng định dạng email.

Ten:  (\*)

Tuoi:  (16->65)

Email:  (@)

Tuoi khong hop le !

Hướng dẫn

- ✓ Sử dụng getParameter() để nhận dữ liệu từ tham số form nhập
- ✓ Sử dụng phương thức length() của chuỗi để kiểm tra số lượng ký tự

- ✓ Sử dụng hàm `Integer.parseInt(String)` và ngoại lệ để kiểm tra xem có nhập đúng số hay không
- ✓ Sử dụng biểu thức chính qui để kiểm tra định dạng email

24. Sử dụng biểu thức chính qui để bắt lỗi đầu vào đối với form nhập sau

Email:

CMND:

Mobile:

Moto Number:

Thông báo lỗi:

- Vui lòng nhập đúng dạng số mobile Việt Nam !
- Vui lòng nhập đúng dạng số xe máy saigon !
- Vui lòng nhập đúng dạng email !
- Vui lòng nhập đúng dạng CMND (9 chữ số) !

25. Khóa tài khoản sai 3 lần đăng nhập sai. HD: bạn hãy ghi số lần đăng nhập sai liên tiếp lên máy của người đăng truy xuất. Nếu đăng nhập sai bạn đọc số lần, tăng lên và gửi trở lại máy người dùng. Nếu đăng nhập đúng thì số số lần đăng nhập trên máy người dùng.
26. Bậc captcha sau 3 lần đăng nhập sau. HD: giống bài trên nhưng thay vì bạn khóa tài khoản thì bạn bậc captcha để người dùng nhập xác nhận mã an toàn. Phương pháp này chống mò mặt khẩu thật sự hữu hiệu.
27. Lưu trữ các mặt hàng yêu thích của khách hàng. HD: trên mỗi mặt hàng, bạn đặt một ngôi sao, khi người dùng click vào, bạn sẽ gửi mã hàng hóa đó xuống máy khách để lưu lại. Trên trang chủ hoặc một góc nào đó, bạn hiển thị các hàng hóa yêu thích (được lưu trên máy người dùng) lên.
28. Tạo một mảng 2 chiều và sinh bảng tính excel. Cột bên phải sẽ là tổng của cột trên hàng đó, hàng dưới cùng là tổng của cột tương ứng.
29. Tạo trang tài liệu pdf cho phép người dùng download. Hãy sử dụng cookie để ghi nhận số lần download mỗi tài liệu của mỗi máy.
30. Hướng dẫn: viết servlet để điều khiển download. Trước khi đọc nội dung tài liệu để trả về, hãy ghi nhận số lần download vào cookie (đọc cookie cũ và tăng số lần download lên).
31. Nâng cấp bài một lên để sau khi download đủ 3 lần thì phải đăng nhập mới được download tiếp
32. Tạo form nhập cho phép nhập thông tin sinh viên trực tuyến. Thông tin sinh viên được lưu vào tập tin Properties.

Mã sinh viên :

Họ và tên :

Khoa :

Điểm trung bình :

Địa chỉ :

Ngày sinh :

Ghi chú :

Hướng dẫn: Mỗi sinh viên được lưu vào 1 tập tin <mã sinh viên>.properties

33. Tạo máy tính có nhớ đơn giản như sau

Số:

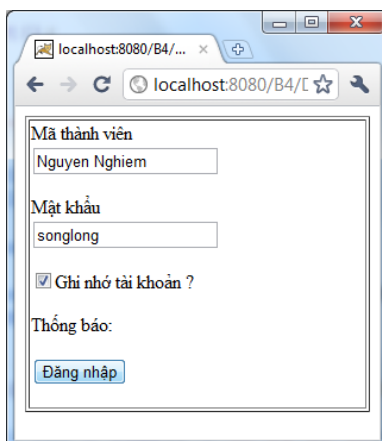
Số nhớ: 0

Kết quả: 8

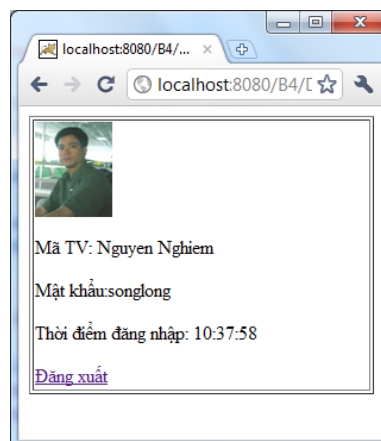
Hướng dẫn

- ✓ Sử dụng session để lưu số nhớ và khởi động là 0 khi mới đến trang này
- ✓ Cộng dồn số nhớ hay trừ bớt tùy thuộc vào hành động click chuột

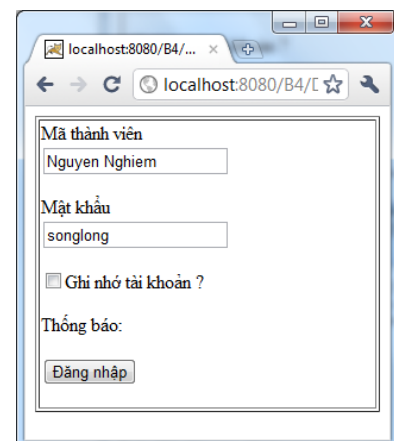
34. Tạo form đăng nhập có sử dụng cookie để ghi nhận thông tin tài khoản đăng nhập ngay trên máy người dùng. Khi người dùng truy xuất form đăng nhập và nếu cookie đã được ghi nhận trước đó thì thông tin tài khoản được đặt sẵn lên form nhập và tích vào checkbox. Form đăng nhập cũng sử dụng session để ghi nhận thông tin tài khoản sau đăng nhập. Dựa sự ghi nhận này để biết được người dùng đã đăng nhập hay chưa để hiển thị phần giao diện tương ứng với trạng thái đăng nhập. Có thể là hiển thị form đăng nhập nếu người dùng chưa đăng nhập, ngược lại hiển thị thông tin tài khoản của người dùng đã đăng nhập.



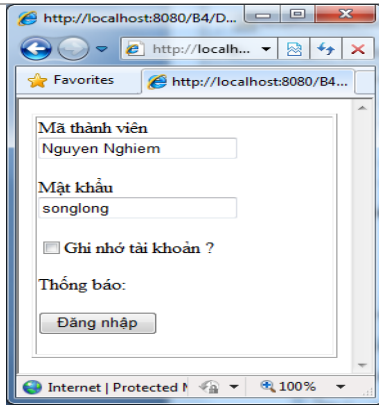
1. Đăng nhập có chọn ghi nhớ



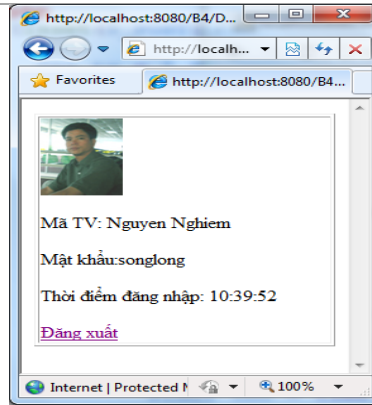
2. Sau đăng nhập



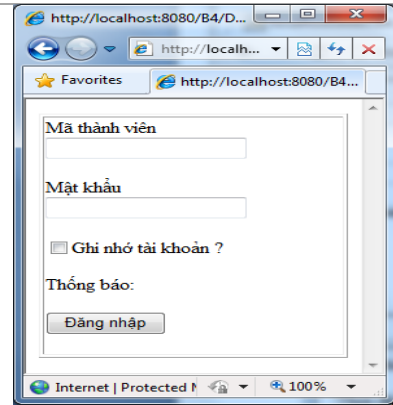
3. Chạy lại trang web



4. Đăng nhập không chọn ghi nhớ



5. Sau đăng nhập



6. Chạy lại sau đăng nhập chưa ghi nhớ

#### Hướng dẫn

- ✓ Xét xem trong session đã ghi nhận thông tin tài khoản đăng nhập hay chưa để cho hiển thị phần giao diện tương ứng.
- ✓ Hiển thị thông tin tài khoản của cookie lên ô nhập và tích vào checkbox ghi nhớ nếu đã tồn tại cookie trước đó.
- ✓ Nhận thông tin tài khoản và thông tin ghi nhớ từ form
- ✓ Thực hiện đăng nhập bình thường
- ✓ Nếu đăng nhập thành công thì lưu thông tin tài khoản vào session và xét tới việc ghi nhớ tài khoản theo các bước sau:
  - Tạo các cookie để chứa thông tin tài khoản
  - Đặt tuổi thọ phù hợp cho cookie nếu có tích vào ghi nhớ
  - Đặt tuổi thọ âm (quá khứ) nếu không tích vào ghi nhớ
  - Gửi cookie về client