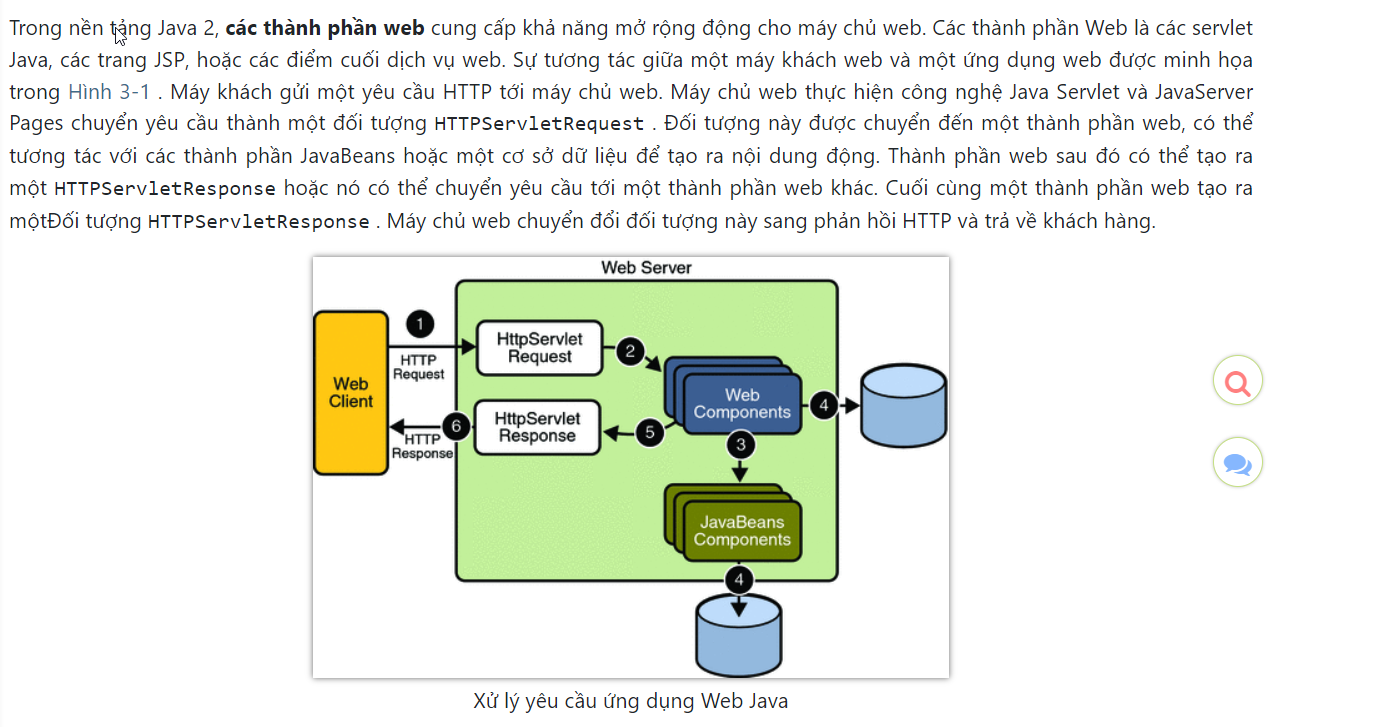
1.Cách hoạt động của Java web application



Các Web Application thường được mã hóa bằng ngôn ngữ được trình duyệt hỗ trợ. Chẳng hạn như [JavaScript](https://vietnix.vn/javascript-la-gi/) và [HTML](https://vietnix.vn/html-la-gi/) vì các ngôn ngữ này dựa vào trình duyệt để render. Một số ứng dụng động sẽ yêu cầu xử lý từ máy chủ. Phần còn lại thì không cần xử lý ở máy chủ. Dưới đây là quy trình xử lý một request của **ứng dụng web**:

* User yêu cầu (request) đến[Web Server](https://vietnix.vn/web-server-la-gi/) qua Internet thông qua trình duyệt web hoặc user interface của ứng dụng.
* Web server chuyển tiếp yêu cầu này đến **Web Application Server**.
* Web Application Server thực hiện các yêu cầu – chẳng hạn như truy vấn database hoặc xử lý dữ liệu. Sau đó tạo kết quả của dữ liệu được yêu cầu.
* Web application server gửi kết quả đến web server với thông tin yêu cầu hoặc dữ liệu đã được xử lý.

**Khác biệt webapp và website**

**Web Application:**Web Application là gì? Là một phần của phần mềm có thể được truy cập bởi trình duyệt. Trình duyệt là một ứng dụng để truy cập Internet. Web App sử dụng kết hợp các tập lệnh phía Server và các tập lệnh phía Client để trình bày thông tin. Nó yêu cầu một server để quản lý các yêu cầu của user.

**Ví dụ:** Google Apps,…

**Website:**Là một tập hợp các trang web có liên quan chứa hình ảnh, văn bản, âm thanh, video, … Nó có thể bao gồm một, hai hoặc nhiều trang. Một website cung cấp nội dung trực quan và văn bản mà người dùng có thể xem và đọc. Để xem một website cần có trình duyệt web (Chrome, Firefox,…). Có nhiều loại website như Archive website, Blog, Website cộng đồng, website bán hàng,…

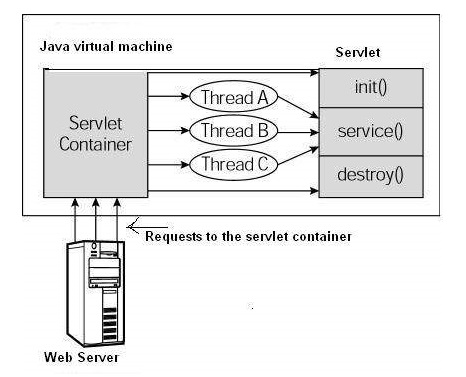
* Web server phản hồi lại máy khách (Client) với thông tin được yêu cầu. Sau đó nó sẽ xuất hiện trên màn hình của user.

2. Cách hoạt động của Servlet

Sơ đồ kiến trúc

Hình dưới đây mô tả vòng đởi của servlet điển hình:

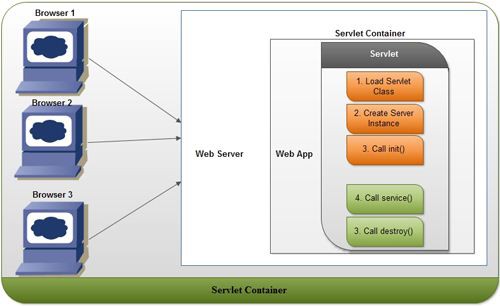
* Đầu tiên các yêu cầu HTTP đến máy chủ được ủy quyền cho bộ chứa servlet.
* Bộ chứa servlet tải servlet trước khi gọi phương thức service().
* Sau đó, servlet container xử lý nhiều yêu cầu bằng cách sinh ra nhiều luồng, mỗi luồng thực hiện service() của một thể hiện duy nhất của servlet.



**Servlet** là một giao diện được định nghĩa trong package javax.servlet. Nó thiết lập 3 method cần thiết cho vòng đời của một servlet - init (), service () và destroy (). Chúng được thực hiện bởi mỗi servlet và được gọi vào các thời điểm cụ thể bởi máy chủ.

Các lớp Servlet (class) cài đặt giao diện javax.servlet.Servlet chạy trong một web container thuộc một web server, đáp ứng nhiều yêu cầu của khách hàng trong cùng 1 thời điểm, được chuyển tiếp thông qua server và web server. Trình duyệt và Servlet giao tiếp với nhau bằng giao thức HTTP (1 giao thức stateless). Trình duyệt web thiết lập kết nối socket đến máy chủ lưu trữ trong URL và gửi yêu cầu HTTP. Một servlet nhận 2 đối tượng khi được gọi từ client. Một "ServletRequest", bao gồm yêu cầu của máy khách từ máy khách và "ServletResponse", gói thông tin liên lạc từ Servlet và trả lại máy khách.

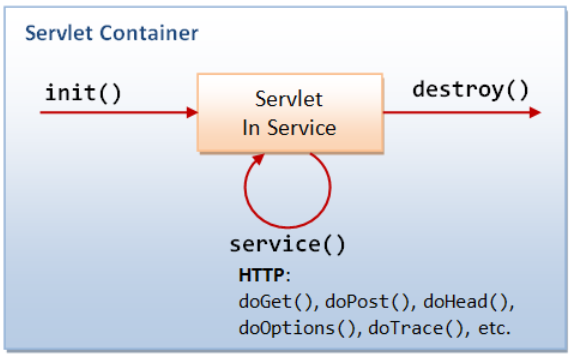
Để hiểu một cách đơn giản thì khi nhận request từ máy khách, sẽ xác định thông tin request để biết được cần response gì. Và trong lập trình java, chúng ta sẽ được cung cấp một class gọi là servlet để tính toán logic quá trình tương tác HTTP (request và response). Mỗi khi có request từ client đến, web server sẽ phân tách url, http method để xác định xem nên giao request cho servlet nào trong servlet container. Và servlet tương ứng khi được giao request sẽ tiến hành logic.



Vòng đời của Servlet

Web container chịu trách nhiệm quản lý vòng đời của servlet. Web container tạo một phiên bản (instance) của servlet và sau đó gọi ra init () method. Khi hoàn thành init () method thì servlet đang ở trạng thái sẵn sàng cho các yêu cầu dịch vụ từ clients. Container gọi service () method của servlet để xử lý từng yêu cầu bằng cách sinh ra 1 thread mới cho mỗi yêu cầu từ thread pool của Web Container. Trước khi phá hủy instance, container sẽ gọi destroy () method. Sau khi destroy (), servlet trở thành rác chờ thu gom.

Giống như bất kỳ chương trình Java nào, servlet chạy trong một JVM. Servlet container sẽ tham gia để xử lý sự phức tạp của các HTTP rerquest. Servlet container chịu trách nhiệm cho việc tạo, thực hiện và hủy của servlet.



3.Cách hoạt động của JSP

**Vòng đời của JSP** được định nghĩa là quá trình từ khi nó được tạo ra cho đến khi nó bị hủy. Vòng đời của JSP cũng tương tự như một vòng đời của servlet với việc bổ sung thêm một bước biên dịch một JSP thành servlet.

Việc hiểu về vòng đời của JSP chính là chìa khóa để hiểu về bản chất của JSP hoạt động như thế nào.

Vòng đời của JSP bao gồm **4 pha** sau:

* Biên dịch (Compilation).
* Khởi tạo (Initialization).
* Thực thi (Execution).
* Hủy (Cleanup).

Các pha này được mô tả trong ảnh sau:



**Nội dung chính**

* [1. Biên dịch JSP](https://viettuts.vn/jsp/vong-doi-cua-jsp#goto-h2-1)
* [2. Khởi tạo JSP](https://viettuts.vn/jsp/vong-doi-cua-jsp#goto-h2-2)
* [3. Thực thi JSP](https://viettuts.vn/jsp/vong-doi-cua-jsp#goto-h2-3)
* [4. Hủy JSP](https://viettuts.vn/jsp/vong-doi-cua-jsp#goto-h2-4)

1. Biên dịch JSP

Khi trình duyệt yêu cầu một JSP, công cụ JSP đầu tiên sẽ kiểm tra xem liệu nó cần phải biên dịch trang này hay không. Nếu trang chưa bao giờ được biên soạn, hoặc nếu JSP đã được sửa đổi kể từ khi nó được biên dịch lần cuối, công cụ JSP biên dịch trang.

Quá trình biên dịch bao gồm ba bước:

* Phân tích cú pháp JSP.
* Chuyển JSP thành một servlet.
* Biên dịch servlet.

2. Khởi tạo JSP

Khi một container tải một JSP nó gọi phương thức **jspInit()** trước khi phục vụ các yêu cầu. Nếu bạn cần thực hiện khởi tạo JSP theo cách của bạn, hãy ghi đè phương thức jspInit().

[?](https://viettuts.vn/jsp/vong-doi-cua-jsp)

|  |  |
| --- | --- |
| 1  2  3 | public void jspInit() {     // Initialization code...  } |

Thông thường, khởi tạo chỉ được thực hiện một lần và giống như phương thức init servlet, thường khởi tạo các kết nối cơ sở dữ liệu, các mở các file và tạo bảng tra cứu trong phương thức jspInit.

3. Thực thi JSP

Giai đoạn này của vòng đời JSP đại diện cho tất cả các tương tác với các yêu cầu cho đến khi JSP bị hủy.

Bất cứ khi nào trình duyệt yêu cầu một JSP và trang đã được nạp và khởi tạo, công cụ JSP sẽ gọi phương thức **\_jspService()**.

Phương thức \_jspService() có một **HttpServletRequest**và một **HttpServletResponse**làm các tham số của nó như sau:

[?](https://viettuts.vn/jsp/vong-doi-cua-jsp)

|  |  |
| --- | --- |
| 1  2  3 | void \_jspService(HttpServletRequest request, HttpServletResponse response) {     // Service handling code...  } |

Phương thức **\_jspService()** của một JSP được gọi ra theo **request**. Nó có trách nhiệm tạo ra response cho request đó và phương thức này cũng có trách nhiệm tạo ra response cho tất cả bảy phương thức HTTP như GET, POST, DELETE, UPDATE vv ...

4. Hủy JSP

Giai đoạn hủy của chu kỳ sống JSP biểu thị khi một JSP không được tiếp tục sử dụng bởi một container.

Phương thức **jspDestroy()** tương đương phương thức hủy đối với servlet. Ghi đè phương thức jspDestroy() khi bạn cần thực hiện bất kỳ công việc dọn dẹp nào, chẳng hạn như giải phóng kết nối cơ sở dữ liệu hoặc đóng các tệp.

Phương thức jspDestroy() có dạng sau:

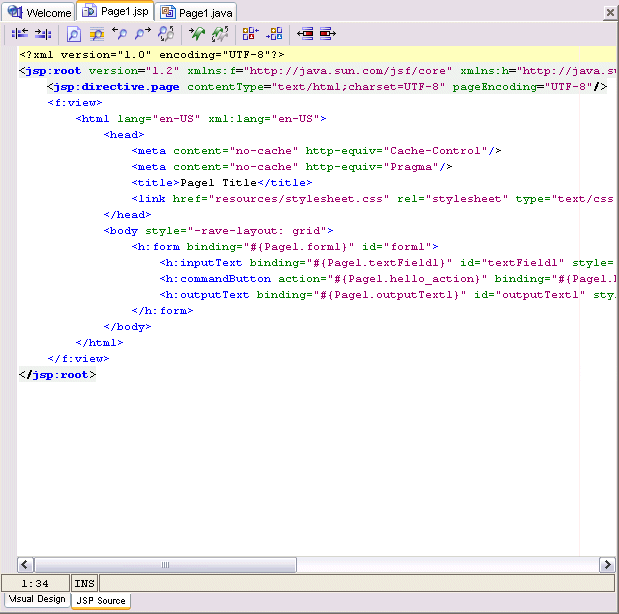
[?](https://viettuts.vn/jsp/vong-doi-cua-jsp)

|  |  |
| --- | --- |
| 1  2  3 | public void jspDestroy() {     // Your cleanup code goes here.  } |

[JSP](https://vi.wikipedia.org/wiki/JSP) (**Java server page** hay **Java Scripting Preprocessor** – tạm dịch là “Bộ tiền xử lý văn lệnh Java”)  là một công nghệ Java cho phép các nhà phát triển tạo nội dung HTML, XML hay một số định dạng khác khiến cho trang web sinh động hớn.

Các JSP tag đặc biệt, hầu hết bắt đầu với <% và kết thúc với %>.

**GIF**

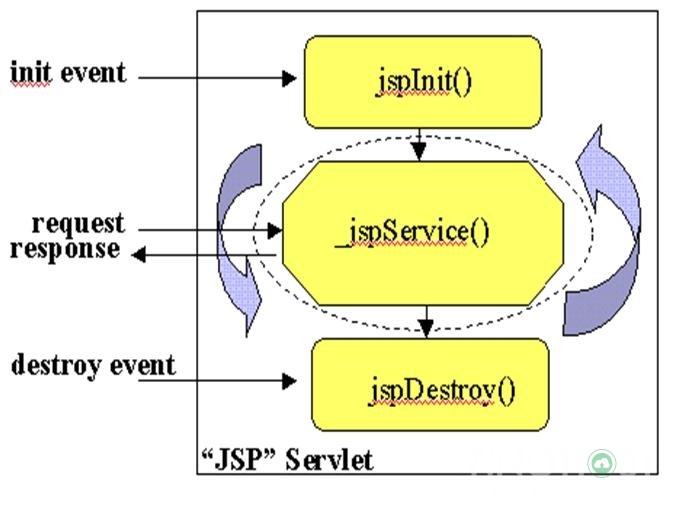


JSP thường được **làm view trong ứng dụng mvc pattern**. Thực ra, JSP vẫn có thể  đáp ứng những yêu cầu khác nhưng để thuận tiện cho việc debug hay tái sử dụng các đoạn mã thì người ta thường dùng làm view còn servlet sẽ làm controller.)

### **Một trang JSP có những thành phần gì?**

* **Thẻ Root:** Thẻ này sẽ chứa các thuộc tính, thông tin của trang JSP.
* **Comment:**  Cũng như trang HTML , trong JSP, bạn cũng có thể comment với kí hiệu này: <! your comment –>
* **Declaration:** khai báo biến hoặc phương thức của java ngay trong trang JSP. Nhưng nếu như khai báo quá nhiều trong trang thì sẽ bị nhầm lẫn giữa code JSP và code java. Cú pháp là <%! your code %>
* **Expression:** được sử dụng để chèn một giá trị vào trong trang một cách trực tiếp.
* **Thẻ biểu thức JSP:**được sử dụng để đánh giá một biểu thức và định hướng các output đến một trình duyệt web phù hợp. Cú pháp khai báo là: <%= your code %>
* **Scriptlet Tag:** cho phép bạn viết mã java trong trang JSP. Cú pháp như sau : <% <i> mã java </ i>%>
* **Directive Elements**

### **“Chu kỳ sống” JSP**



### **Ưu điểm của JSP:**

* Hỗ trợ cho việc **thiết kế giao diện web** dễ dàng hơn.
* Có vai trò lớn trong việc cho phép thiết kế web tạo nên những **trang web động**.
* Có thể viết một nơi và chạy **bất cứ nơi nào**.

### **Hạn chế:**

* Tiêu tốn **dung lượng** lưu trữ phía server**gấp đôi**.
* Lần đầu tiên truy cập vào trang JSP sẽ **mất nhiều thời gian chờ**.

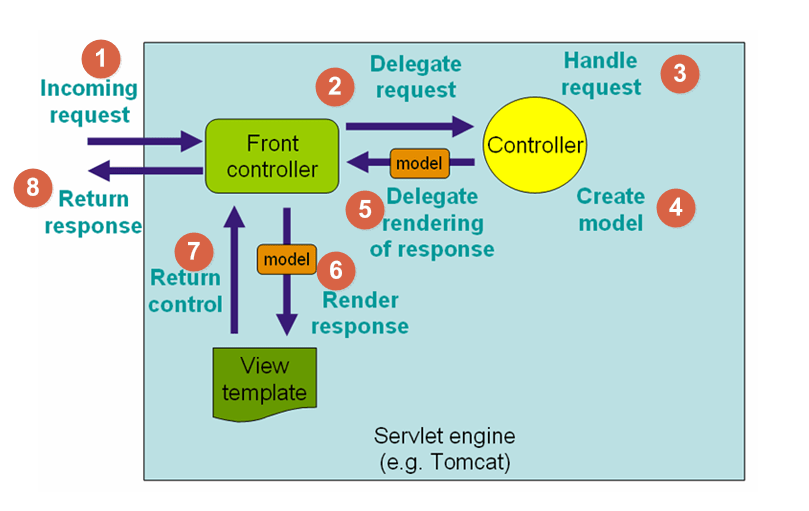
**4.Kiến trúc spring lập trình với spring web mvc**

# 1. Spring MVC (Spring Web MVC)

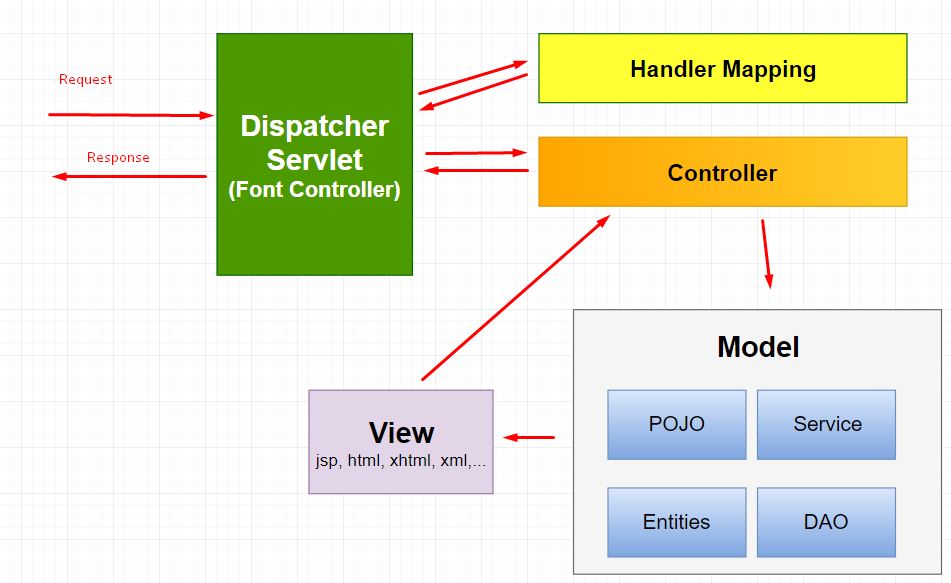
Spring MVC là một Framework / 1 Project mã nguồn mở của Spring.

Spring MVC Framewrok cung cấp kiến truc MVC (Model-View-Controller) và các component được sử dụng để phát triển các ứng dụng web một cách linh hoạt và giảm sư

# 2. Flow trong Spring MVC.



* Bất kỳ request nào tới ứng dụng web đều sẽ được gửi tới Front Controller (Dispatcher Servlet)
* Front Controller sẽ sử dụng Handler Mapping để biết được controller nào sẽ xử lý request đó
* Controller nhận request, gọi tới các class service thích hợp để xử lý yêu cầu.
* Sau khi xử lý xong, Controller sẽ nhận được model từ tầng Service hoặc tầng DAO.
* Controller gửi model vừa nhận được tới Front Controller (Dispatcher Servlet)
* Dispatcher Servlet sẽ tìm các mẫu view, sử dụng view resolver và truyền model vào nó.
* View template, model, view page được build và gửi trả lại Front Controller
* Front Controller gửi một page view tới trình duyệt để hiển thị nó cho người dùng.



Trong Mô hình MVC thì:

* **Model**: là các file POJO, Service, DAO thực hiện truy cập database, xử lý business
* **View**: là các file JSP, html…
* **Control**: là Dispatcher Controller, Handler Mapping, Controller – thực hiện điều hướn các request.

**5. inversion of control ( denpendency injection) trong Spring**

**2. Inversion of Control là gì?**

**IoC** còn có tên gọi đầy đủ là **Inversion of Control**, được hiểu là một nguyên lý thiết kế ứng dụng trong công nghệ phần mềm. Kiến trúc phần mềm khi áp dụng nguyên lý thiết kế này sẽ đảo ngược quyền điều khiển so với kiểu lập trình hướng thủ tục. Nếu như trong lập trình hướng thủ tục, các đoạn mã được thêm vào sẽ gọi các thư viện thì ở IoC lại hoàn toàn khác. Những IoC container sẽ chích những dependencies khi khởi tạo bean.

**3. Dependency Injection là gì?**

Đây là một pattern dùng để triển khai thực hiện IoC, các dependencies sẽ được đưa vào module trong quá trình khởi tạo.

Việc kết nối giữa các đối tượng với các đối tượng khác, hoặc inject các đối tượng vào đối tượng khác được thực hiện bằng quá trình lắp ráp chứ không phải bởi chính các đối tượng.

một đối tượng dependency trong lập trình truyền thống:

**public** **class** **Store** {

**private** Item item;

**public** Store() {

item = **new** ItemImpl1();

}

}

Code language: PHP (php)

Còn đây là sau khi sử dụng DI:

**public** **class** **Store** {

**private** Item item;

**public** Store(Item item) {

this.item = item;

}

}

**Spring IoC Container**

IoC container là một trong những khái niệm dùng chung cho các framework triển khai Inversion of Control – IoC.

Trong Spring, IoC container được đại diện bởi *ApplicationContext*interface. Spring container chịu trách nhiệm khởi tạo, cấu hình và lắp ráp các object được biết đến là các [Bean trong Spring](https://shareprogramming.net/tai-sao-spring-bean-duoc-goi-la-xuong-song-cua-ung-dung/), nó cũng quản lý luôn vòng đời của chúng.

Spring cung cấp một số cách triển khai *ApplicationContext*interface như thông qua *ClassPathXmlApplicationContext*và *FileSystemXmlApplicationContext*cho các ứng dụng độc lập và *WebApplicationContext*cho các ứng dụng web.

Để lắp ráp các Bean trong ứng dụng, container sẽ sử dụng các metadata được cấu hình ở dạng XML hoặc annotation.

Dưới đây là một cách thủ công để tạo một Spring container

ApplicationContext context

= **new** ClassPathXmlApplicationContext**(**"applicationContext.xml"**)**;

Để đặt giá trị cho một Item implementation ở trên thì chúng ta có thể chỉ định nó trong metadata. Và sau đó container sẽ đọc thông tin này và sử dụng nó để tạo ra một Bean tương ứng với Item interface.

Dependency Injection trong Spring có thể thực hiện thông qua constructor, setter, fields mà chúng ta sẽ tìm hiểu ở những phần sau.

**6.Mô tả các đối tượng được khởi tạo sẵn trong JSP , request , response,session.Cách truyền tham số qua request**

|  |  |
| --- | --- |
| 1 | **request**  Đây là đối tượng **HttpServletRequest** được liên kết với request. |
| 2 | **response**  Đây là đối tượng **HttpServletResponse** được liên kết với response tới client. |
| 3 | **out**  Đây là đối tượng **PrintWriter** được sử dụng để gửi dữ liệu tới client. |
| 4 | **session**  Đây là đối tượng **HttpSession** được liên kết với request. |
| 5 | **application**  Đây là đối tượng **ServletContext** được liên kết với application context. |
| 6 | **config**  Đây là đối tượng **ServletConfig** được liên kết với page. |
| 7 | **pageContext**  Điều này đóng gói việc sử dụng các tính năng cụ thể của server như **JspWriters** với hiệu suất cao hơn. |
| 8 | **page**  Đơn giản là một từ đồng nghĩa với **this**, được sử dụng để gọi các phương thức được định nghĩa bởi lớp servlet được biên dịch. |
| 9 | **Exception**  Đối tượng **Exception** cho phép các dữ liệu ngoại lệ được truy cập bằng JSP được chỉ định. |

1. Đối tượng request

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

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

2. Đối tượng response

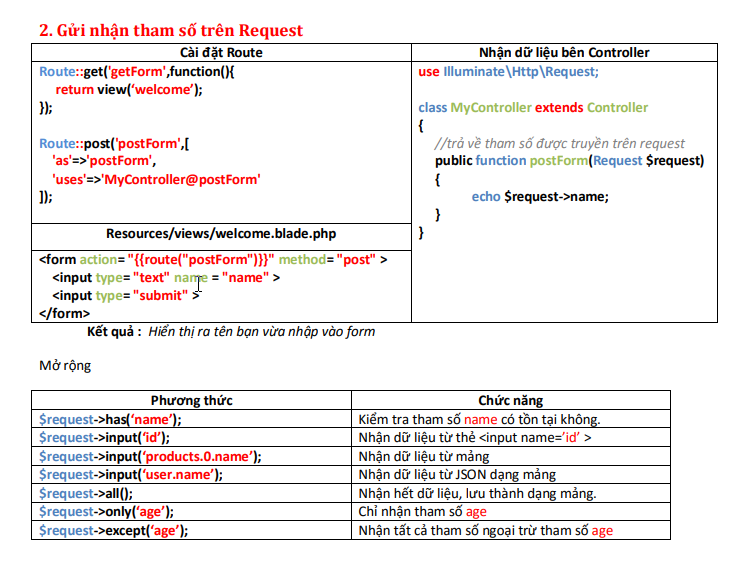
Đối tượng response là một thể hiện của **javax.servlet.http.HttpServletResponse**. Cũng như việc server tạo ra các đối tượng resquest, nó cũng tạo ra một đối tượng response để đại diện cho các phản hồi cho client.

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

## Đối tượng session

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

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



**7.Java Beans và cách sử dụng**

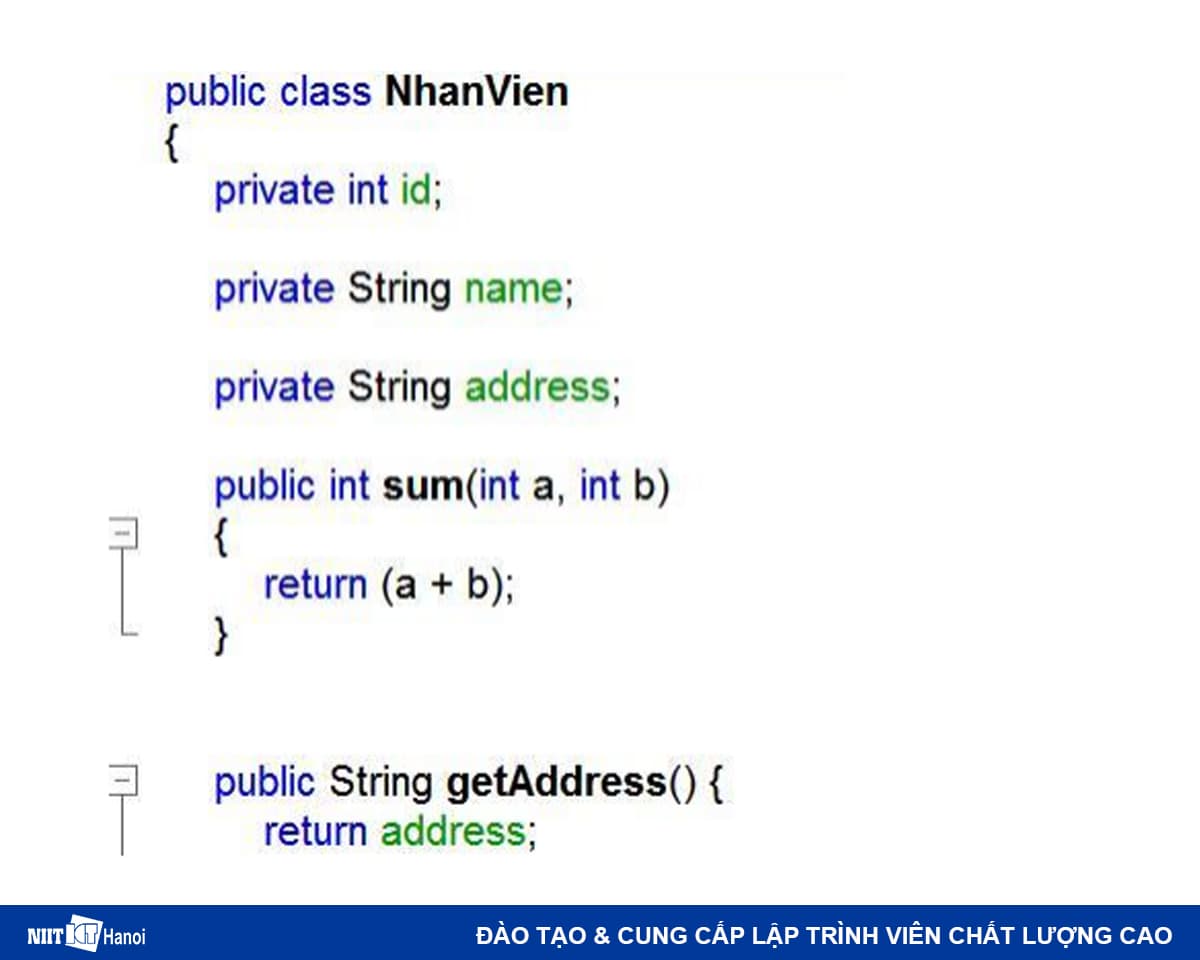
**JavaBean** là các lớp Java thuần, dùng để thể hiện các thực thể trong các chương trình Java.

Một lớp JavaBean sẽ có 3 đặc điểm như sau về mặt cú pháp:

1. Là một public class  
2. Có các thuộc tính private  
3. Ứng với mỗi thuộc tính, lớp sẽ cung cấp một cặp phương thức setter / getter để truy cập và thao tác với giá trị của từng thuộc tính.

+ setter: Là các phương thức cho phép thiết lập giá trị của thuộc tính.  
+ getter: Là các phương thức cho phép truy cập và lấy về giá trị của thuộc tính.

Tạo ra một Java class dùng để thể hiện JavaBean. Ở đây ta định nghĩa lớp có tên là NhanVien.  
  
Khai báo các properties, sau đó tạo ra các cặp method setter/getter.



Ta tạo một trang **JSP** để viết mã giao diện trong đó có sử dụng JavaBean. Sau đó ta gọi các method của JavaBean.  
  
Có thể sử dụng 1 trong 2 cách:

* Sử dụng [**JSP Scriptlets**](https://docs.oracle.com/javaee/5/tutorial/doc/bnaou.html): <% %>
* Hoặc sử dụng action <jsp:useBean>

Sau đó hiển thị kết quả trên trang JSP. Dưới đây là hình minh họa mình đã truy cập đến JavaBean theo 2 cách.

  
  
*2 Cách truy cập JavaBean*

Ý nghĩa của các thẻ:

* <jsp:useBean>: Dùng để định vị hoặc tạo ra đối tượng JavaBean.
* <jsp:setProperty>: Dùng để thiết lập giá trị cho thuộc tính của JavaBean.
* <jsp:getProperty>: Dùng để get (lấy) giá trị thuộc tính của JavaBean.

Action useBean khai báo JavaBean để sử dụng trong một tệp JSP.  
  
Sau khi được khai báo, bean trở thành một biến script có thể được truy cập bởi cả các phần tử script và các thẻ tùy chỉnh khác được sử dụng trong JSP.

**8.Cách hoạt động của Servlet Fillter.Cách sử dụng.**

Servlet Filter là các lớp trong Java có thể được sử dụng trong lập trình Servlet cho các mục đích sau:

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

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

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

Các Filter được triển khai trong **web.xml** file và sau đó map tới hoặc các tên Servlet hoặc các URL pattern.

Các phương thức Servlet Filter

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

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

Ví dụ về Servlet Filter

Ví dụ sau về Servlet Filter sẽ in địa chỉ IP và thời gian hiện tại của Client. Ví dụ này sẽ giúp bạn hiểu cơ bản về Servlet Filter, nhưng bạn cũng có thể viết các ứng dụng Filter phức tạp hơn bởi sử dụng cùng khái niệm này.

// Import required java libraries

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

import java.util.\*;

// Implements Filter class

public class LogFilter implements Filter {

public void init(FilterConfig config)

throws ServletException{

// Get init parameter

String testParam = config.getInitParameter("test-param");

//Print the init parameter

System.out.println("Test Param: " + testParam);

}

public void doFilter(ServletRequest request,

ServletResponse response,

FilterChain chain)

throws java.io.IOException, ServletException {

// Get the IP address of client machine.

String ipAddress = request.getRemoteAddr();

// Log the IP address and current timestamp.

System.out.println("IP "+ ipAddress + ", Time "

+ new Date().toString());

// Pass request back down the filter chain

chain.doFilter(request,response);

}

public void destroy( ){

/\* Called before the Filter instance is removed

from service by the web container\*/

}

}

**9.Mô hình MVC.RequestDispatcher**

**Mô hình MVC là gì? Các thành phần của MVC**

**MVC** là từ viết tắt bởi 3 từ **Model – View – Controller**. Đây là mô hình thiết kế sử dụng trong kỹ thuật phần mềm. Mô hình source code thành 3 phần, tương ứng mỗi từ. Mỗi từ tương ứng với một hoạt động tách biệt trong một mô hình.

Để hiểu rõ hơn, sau đây chúng ta sẽ cùng đi phân tích từng thành phần:



Mô hình MVC và các thành phần bên trong của MVC.

**Model (M):**

Là bộ phận có chức năng lưu trữ toàn bộ dữ liệu của ứng dụng. Bộ phận này là một cầu nối giữa 2 thành phần bên dưới là View và Controller. Model thể hiện dưới hình thức là một cơ sở dữ liệu hoặc có khi chỉ đơn giản là một [file XML](https://monamedia.co/file-xml-la-gi/) bình thường. Model thể hiện rõ các thao tác với cơ sở dữ liệu như cho phép xem, truy xuất, xử lý dữ liệu,…

**View (V):**

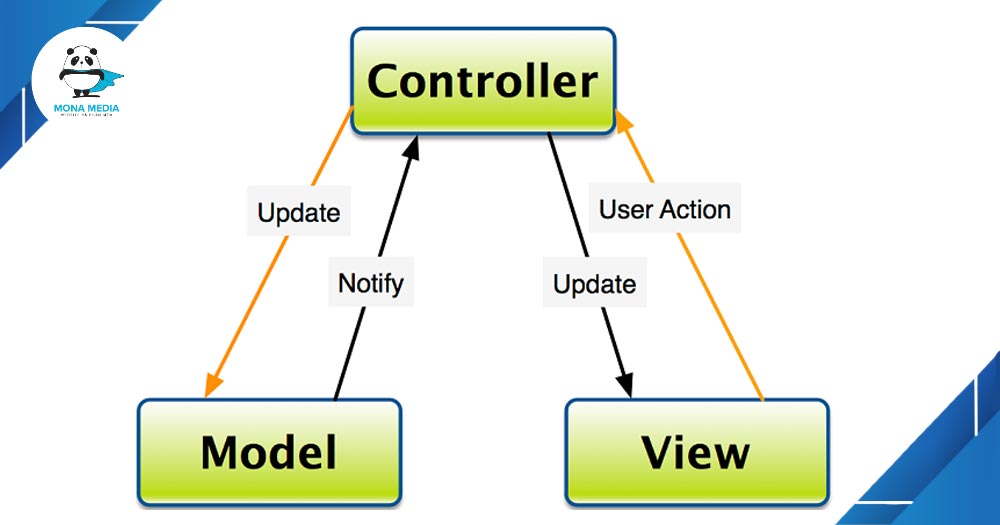
Đây là phần giao diện (theme) dành cho người sử dụng. Nơi mà người dùng có thể lấy được thông tin dữ liệu của MVC thông qua các thao tác truy vấn như tìm kiếm hoặc sử dụng thông qua các website.

Thông thường, các ứng dụng web sử dụng MVC View như một phần của  hệ thống, nơi các thành phần [HTML](https://monamedia.co/html-va-css-la-gi/) được tạo ra. Bên cạnh đó, View cũng có chức năng ghi nhận hoạt động của  người dùng để tương tác với Controller. Tuy nhiên, View không có mối quan hệ trực tiếp với Controller, cũng không được lấy dữ liệu từ Controller mà chỉ hiển thị yêu cầu chuyển cho Controller mà thôi.

Ví dụ: Nút “delete” được tạo bởi View khi người dùng nhấn vào nút đó sẽ có một hành động trong Controller.

**Controller (C):**

Bộ phận có nhiệm vụ xử lý các yêu cầu người dùng đưa đến thông qua view. Từ đó, C đưa ra dữ liệu phù hợp với người dùng. Bên cạnh đó, Controller còn có chức năng kết nối với model.



Luồng tương tác giữa các thành phần trong MVC.

Ví dụ: Chức năng thêm bài viết mới trong trang quản trị website. Nơi đây có 2 trường nhập về tiêu đề và nội dung bài viết, thì trong đó:

* View sẽ hiển thị ra phần nhập form tiêu đề và nội dung.
* Controller lấy dữ liệu từ 2 trường và gửi tới Model.
* Model lúc này sẽ nhận dữ liệu từ Controller để lưu trữ trong cơ sở dữ liệu.

Mỗi bộ phận thực hiện chức năng nhất định, nhưng chúng có sự thống nhất, liên kết với nhau tạo nên **mô hình MVC**. Mô hình này tương đối nhẹ. Nó có thể tích hợp được nhiều tính năng có trong ASP.NET hiện giờ. Ví dụ như authentication (quá trình xác thực).

**Luồng đi trong mô hình MVC như thế nào?**

Bạn có thể hình dung, khi  một yêu cầu từ máy client gửi tới server, Controller sẽ thực hiện nhiệm vụ của mình đó là tiếp nhận và xử lý yêu cầu. Trong trường hợp cần thiết, nó có thể liên hệ Model – bộ phận làm việc với database để hỗ trợ.

Khi xử lý xong yêu cầu, kết quả sẽ được trả về View. Tại View sẽ tạo mã HTML thành giao diện và trả về hiển thị trên trình duyệt.

**10.Kiến trúc của strust2**

Struts framework

Struts framework ban đầu được tạo ra bởi **Craig McClanahan** và tặng cho Apache Foundation vào tháng 5 năm 2000 và Struts 1.0 đã được phát hành vào tháng 6 năm 2001.

Phiên bản ổn định hiện tại của Struts là 2.3.33 được phát hành vào tháng 7 năm 2017.

Các bài hướng dẫn Struts2 framework bao gồm lý thuyết các ví dụ đơn giản cho người mới bắt đầu và những người có kinh nghiệm.

Struts2 framework

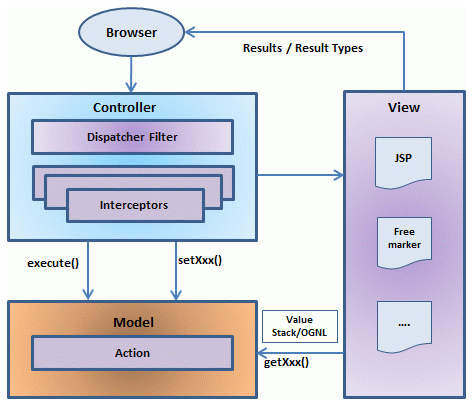
**Struts2** là framework được sử dụng để tạo ứng dụng web dựa trên mẫu thiết kế MVC. Struts2 không chỉ là phiên bản tiếp theo của Struts1, mà nó là bản nâng cấp hoàn chỉnh của kiến trúc Struts.

Struts2 framework là sự kết hợp hoàn hảo từ WebWork framework và Struts framework **để làm cho việc phát triển web trở nên dễ dàng và đơn giản hơn cho các nhà phát triển**.

Struts2 là một pull-MVC(hoặc MVC2). Mô hình Model-View-Controller trong Struts2 được thực hiện với 5 thành phần cốt lõi sau đây:

* **Actions**
* **Interceptors**
* **Value Stack / OGNL**
* **Results / Result types**
* **View technologies**

Struts2 hơi khác so với mô hình MVC truyền thống trong đó Action chiếm vai trò của Model thay vì Controller.



Biểu đồ trên mô tả các **M** odel, **V** iew và **C** ontroller cho kiến trúc mức cao Struts2. Controller được thực hiện với bộ lọc servlet Struts2 cũng như các interceptor, Model được thực hiện với các hành động, và View là sự kết hợp của các kiểu kết quả và các kết quả. Giá trị Stack và OGNL cung cấp thread, liên kết chung và cho phép tích hợp với các thành phần khác.

Ngoài các thành phần trên, sẽ có rất nhiều thông tin liên quan đến cấu hình. Cấu hình cho các ứng dụng web, cũng như cấu hình cho các action, interceptor, result, vv

Đây là kiến trúc tổng quan của mô hình Struts2 MVC. Chi tiết về từng thành phần sẽ được đề cập đến trong các bài tiếp theo.

**Nội dung chính**

* [Vòng đời của một request](https://viettuts.vn/struts2/kien-truc-struts2#goto-h2-1)

Vòng đời của một request

Dựa trên digram ở trên, có thể giải thích vòng đời của một request của người dùng trong Struts2 như sau:

* Người dùng gửi request tới máy chủ để yêu cầu một số tài nguyên (ví dụ các trang).
* FilterDispatcher xem xét yêu cầu và sau đó xác định action thích hợp.
* Chức năng interceptor đã cấu hình được áp dụng như xác nhận hợp lệ, upload file, vv
* Action đã chọn được thực thi để thực hiện thao tác được yêu cầu.
* Một lần nữa, interceptor đã cấu hình được áp dụng để thực hiện bất kỳ post-processing nếu cần thiết.
* Cuối cùng kết quả được chuẩn bị bởi view và trả kết quả cho người dùng.

**11.Kiến trúc của JSF**

# JSF là gì? Giới thiệu JSF Framework

Posted on [*Tháng Mười Một 27, 2017*](https://stackjava.com/jsf/series-jsf-phan-1-jsf-la-gi-gioi-thieu-jsf.html)

Series JSF – Phần 1: JSF là gì? Giới thiệu JSF Framework.

Ở series này mình sẽ nói về phiên bản JSF mới nhất là JSF 2.2 vì nó dễ dùng và mạnh mẽ hơn các phiên bản trước.

Những bài đầu trong series này sẽ khá nhàm chán và hơi khó hiểu, mình sẽ cố gắng viết và giải thích chi tiết nhất có thể.

Kinh nghiệm của mình là những bài lý thuyết đọc qua để hiểu được tổng quan rồi tiến luôn tới làm ví dụ. Sau đó quay lại đọc lý thuyết sẽ hiệu quả hơn.

# 1. JSF là gì?

[**JavaServer Faces**](https://javaee.github.io/javaserverfaces-spec/) (JSF) là một web framework MVC. Nó tập  trung vào việc đơn giản hóa xây dựng giao diện người dùng(User Interface -UI) (với hơn 100 tags UI) cho ứng dụng web và làm cho phép việc sử dụng lại các thành phần UI được thực hiện một cách dễ dàng.

# 2. Lợi ích của JSF

JSF giảm chi phí trong việc tạo và bảo trì ứng dụng, nó sẽ chạy trên một máy chủ ứng dụng Java và hiển thị các thành phần UI cho khách hàng. JSF tạo điều kiện cho phát triển ứng dụng web nhờ các tính năng sau:

* Cung cấp các thành phần giao diện (UI) sử dụng lại được
* Chuyển dữ liệu dễ dàng giữa các thành phần UI
* Quản lý trạng thái UI giữa các request từ server
* Cho phép cài đặt các thành phầ UI tùy chỉnh
* Lấy sự kiện ở phía client tới phía server

# 3. Các tính năng của JSF

JSF cung cấp các tính năng sau:

## Các thành phần dựa trên framework (Compnent Based Framework):

JSF là 1 framework server-side dựa trên các component. Nó cung cấp các component sẵn có để xây dựng ứng dụng web. Bạn có thể sử dụng HTML5, các Facelets tag để tạo trang web.

## Công nghệ Facelets (Facelets Technology):

Facelets là một một mã nguồn mở về hệ thống web mẫu (Web template system). Nó là công nghệ xử lý view mặc định cho JSF, yêu cầu đầu vào XML hợp lệ để làm việc. Facelets hỗ trợ tất cả các thành phần UI của JSF và tập trung hoàn toàn vào xây dựng view cho ứng dụng JSF

## Expression Language (EL)

Expression languate cung cấp một cơ chế quan trọng cho việc tạo giao diện người dùng (web pages) để giao tiếp với ứng dụng (các beans).

## HTML5:

HTML5 là một chuẩn mới cho viết trang web. JSF đưa ra một cách dễ dàng để sử dụng các thuộc tính của HTML 5 vào các thành phần JSF.

## Phát triển web dễ dàng và nhanh chóng (Ease and Rapid web Development):

JSF cung cấp các các tool và các thư viện sẵn có và nhiều tiện ích để bạn có thể phát triển ứng dụng web dễ dàng và nhanh chóng.

## Hỗ trợ đa ngôn ngữ (Support Internationalization):

JSF hỗ trợ đa ngôn ngữ cho việc tạo ứng dụng web. Bạn có thể thạo một ứng dụng cho nhiều ngôn ngữ khác nhau.

Bean Annotations: JSF cung cấp các annotaions tiện ích, thứ giúp bạn có thể thực hiện kiểm tra tính hợp lệ của data.

## Xử lý exception (Exception Handling):

JSF cung cấp cách xử lý exception mặc định.

Templating: trong ứng dụng JSF bạn có thể tạo các template mới, sử dụng lại chúng và coi nó như một component.

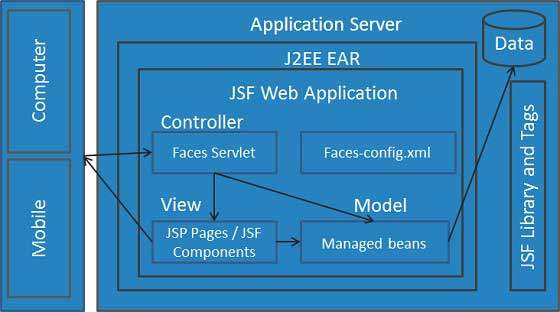
## Hỗ trợ AJAX (AJAX Support):

Bạn có thể hiển thị các request tới server side mà không cần tải lại trang. JSF cũng hỗ trợ việc hiển thị lại từng thành phần sử dụng AJAX. (Ví dụ: bạn gõ vào ô search, nó sẽ hiển thị ra kết quả mà không cần tải lại trang… JSF khá mạnh trong làm việc với AJAX, gần như bạn không cần phải quan tâm tới việc gửi nhận request, javascript trong JSF)

## Bảo mật (Security):

JSF lưu trữ trạng thái của các thành phần view, khi gửi request tới server, các thành phần này phải có các tham số hợp lệ (ví dụ field name của thẻ input với có giá trị giống với giá trị server đang lưu). Các tham số này được tạo ra một cách ngẫu nhiên.

# 4. Kiến trúc của JSF



Faces Servlet đóng vai trò là Controller: nhận các request từ client gửi đến và điều hương tới  View hoặc Model

Các Managed Beans đóng vai trò là Model: thực hiện các chức năng business và giao tiếp với data

Phần View gồm JSP Pages và các component JSF.

**12.Lập trình csdl trong web application transaction**

# Transaction là gì?

Transaction là 1 giao dịch (hay còn gọi là 1 giao tác) bao gồm 1 loạt các hành động được phải được thực hiện thành công cùng nhau, nếu 1 hành động thất bại thì tất cả các hành động trong loạt hành động đó sẽ trở về trạng thái ban đầu.

Ví dụ bạn có 2 tài khoản ngân hàng A và B, bạn chuyển 1.000.000đ từ tài khoản A vào tài khoản B.

Transaction ở đây sẽ gồm 2 hành động:

* Tài khoản A bị trừ 1.000.000đ
* Tài khoản B cộng thêm 1.000.000đ

Bây giờ A bị trừ 1.000.000đ  nhưng vì 1 lý do nào đấy B bị lỗi không nhận được 1.000.000đ thì transaction sẽ bị hủy, số tiền của 2 tài khoản A và B sẽ không bị thay đổi.

Chính vì ràng buộc phải hoàn thành cùng nhau nên transaction được áp dụng nhiều trong các hành động yêu cầu độ tin cậy, đảm bảo tính chính xác dữ liệu…

A bị trừ 50.000.000đ còn B được cộng thêm 50.000.000đ nên sẽ update amount của A thành 40.000.000đ còn B là 140.000.000đ

* **UpdateWithTransaction.java**
* package stackjava.com.demojdbc.statement;
* import java.sql.Connection;
* import java.sql.DriverManager;
* import java.sql.SQLException;
* import java.sql.Statement;
* public class UpdateWithTransaction {
* public static void main(String[] args) throws SQLException, ClassNotFoundException {
* Class.forName("com.mysql.jdbc.Driver");
* Connection dbConnection = DriverManager.getConnection("jdbc:mysql://localhost:3306/demo-jdbc-transaction", "root", "admin1234");
* Statement stmt = dbConnection.createStatement();
* dbConnection.setAutoCommit(false);
* try {
* stmt.executeUpdate("UPDATE account\_banking SET amount = 40000000.0 WHERE name = 'A'");
* stmt.executeUpdate("UPDATE account\_banking SET amount = 140000000.0 WHERE name = 'B'");
* dbConnection.commit();
* } catch (Exception e) {
* e.printStackTrace();
* dbConnection.rollback();
* }
* dbConnection.close();
* }
* }

Giải thích:

* **dbConnection.setAutoCommit(false)**: dữ liệu sẽ chỉ update vào database khi gọi lệnh **commit()**
* **dbConnection.rollback()**: lấy lại dữ liệu ban đầu.

**13.Các scope application, page,request , session khi khai báo sử dụng Java Bean**

# 2. Application / context scope

* Application scope hay context scope bắt đầu từ khi ứng dụng web được chạy cho tới khi shutdown. Các tham số, giá trị (Parameters/attributes) bên trong application scope sẵn dùng với tất cả các request và session.
* **Application** scope được định nghĩa bởi javax.servlet.**ServletContext** interface.
* Application object sẵn dùng trong 1 JSP page giống như một object ẩn được gọi bởi application.
* Trong một servlet, bạn có theer lấy các đối tượng application bằng cách gọi getServletContext() bên trong servlets code một cách trực tiếp hoặc gián tiếp quagetServletConfig().getServletContext().
* Web container cung cấp một đối tượngServletContext cho mỗi ứng dụng web.

# 3. Request scope

* Request scope bắt đầu ngay khi một HTTP request được gửi tới server và kết thúc khi server trả về một HTTP response
* Các tham số/thuộc tính trong một Request scope có thể được truy cập từ các servlet hoặc jsp cùng phục vụ trong 1 request. Ví dụ bạn gọi 1 servlet/jsp sau đó các servlet/jsp này lại gọi các servlet/jsp khác rồi mới trả về reponse.
* Request object có sẵn trong JSP page như là 1 object ẩn . Bạn có thể set value cho 1 thuộc tính trong request object từ servlet và lấy nó ra ở JSP (Phải trong cùng 1 request).

# 4.Session scope

* Một Session Scope bắt đầu khi một client thành lập kết nối với ứng dụng web cho tới khi hết thời gian timeout hoặc browser bị đóng.
* Các giá trị bên trong session scope có thể được truy cập từ nhiều request khác nhau của cùng 1 client
* Một tính năng đáng chú ý của trình duyệt web (browser) là  session được chia sẻ giữa các tab khác nhau. Bạn có thể gửi request từ 1 tab khác mà không cần login lại. Ví dụ bạn login gmail ở 1 tab, sau đó bạn sang tab khác vẫn có thể tải mail, gửi mail mà không cần phải login lại. Do đó, hãy luôn mở một cửa sổ trình duyệt mới khi bạn muốn thực hiện các giao dịch bảo mật.
* Session object sẵn dùng trong JSP pages giống như một object ẩn
* Trong Servlet, bạn có thể lấy đối tượng object bằng cách gọi request.getSession().

# JSP page scope

* Page scope giới hạn bởi phạm vi và thời gian tồn tại của các thuộc tính trong cùng 1 page nơi mà nó được tạo.
* Nó sẵn dùng trong một JSP page giống như một object ẩn.

Ví dụ:

* <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
* pageEncoding="ISO-8859-1"%>
* <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
* <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
* <html>
* <head>
* <title>JSP Page Scope Example</title>
* </head>
* <body>
* <c:set var="name" value="Dinesh" scope="page" />
* Local Variable : <c:out value="${name}" />
* <a href="test.jsp">Test Page</a>
* </body>
* </html>

Kết quả:

Local Variable: Dinesh

**14.Custom tagss trong jsp**

Một **custom tag** là một phần tử JSP được định nghĩa bởi người sử dụng. Khi một JSP page chứa một custom tag được phiên dịch thành một Servlet, thẻ này được biến đổi để hoạt động trên một đối tượng đã gọi một Tag Handler. Sau đó, Web container triệu hồi các hoạt động này khi Servlet của JSP page được thực thi.

Sự kế thừa của các JSP tag giúp bạn tạo các thẻ mới mà bạn có thể chèn một cách trực tiếp vào trong một JSP page, như khi bạn thực hiện với các thẻ có sẵn (built-in) trong các chương trước đã giới thiệu. JSP 2.0 Specification giới thiệu Simple Tag Handler để viết các Custom Tag này.

Để viết một custom tag, đơn giản bạn có thể kế thừa lớp SimpleTagSupport và ghi đè phương thức **doTag()**, tại đây bạn có thể đặt code của bạn để tạo content cho thẻ đó.

## Tạo thẻ "Hello" trong JSP

Giả sử bạn muốn định nghĩa một custom tag tên là <ex:Hello> và bạn muốn sử dụng nó trong kiểu mà không có phần thân như sau:

<ex:Hello />

Để tạo một Custom Tag trong JSP, đầu tiên bạn phải tạo một lớp Java mà hoạt động như là một Tag Handler. Vì thế, chúng ta tạo HelloTag class như sau:

package com.tutorialspoint;

import javax.servlet.jsp.tagext.\*;

import javax.servlet.jsp.\*;

import java.io.\*;

public class HelloTag extends SimpleTagSupport {

public void doTag() throws JspException, IOException {

JspWriter out = getJspContext().getOut();

out.println("Hello Custom Tag!");

}

}

Code trên có mã hóa đơn giản, với phương thức doTag() nhận đối tượng JspContext hiện tại bởi sử dụng phương thức getJspContext() và sử dụng nó để gửi "Hello Custom Tag!" tới đối tượng JspWriter hiện tại.

Biên dịch lớp trên và copy nó trong một thư mục có sẵn trong biến môi trường CLASSPATH. Cuối cùng, tạo file thư viện thẻ sau: <Tomcat-Installation-Directory>webapps\ROOT\WEB-INF\custom.tld.

<taglib>

<tlib-version>1.0</tlib-version>

<jsp-version>2.0</jsp-version>

<short-name>Example TLD</short-name>

<tag>

<name>Hello</name>

<tag-class>com.tutorialspoint.HelloTag</tag-class>

<body-content>empty</body-content>

</tag>

</taglib>

Bây giờ là lúc sử dụng custom tag là **Hello** đã định nghĩa trên trong chương trình JSP như sau:

<%@ taglib prefix="ex" uri="WEB-INF/custom.tld"%>

<html>

<head>

<title>A sample custom tag</title>

</head>

<body>

<ex:Hello/>

</body>

</html>

Thử gọi JSP trên và nó sẽ cho kết quả sau:

Hello Custom Tag!

## Truy cập phần thân thẻ trong JSP

Bạn có thể bao một message trong phần thân của Custom Tags như khi bạn đã thấy với các thẻ chuẩn. Giả sử bạn muốn định nghĩa một custom tag với tên <ex:Hello> và bạn muốn sử dụng nó trong kiểu có một phần thân thẻ:

<ex:Hello>

This is message body

</ex:Hello>

Chúng ta sửa đổi code trên một chút để xử lý phần thân cho thẻ:

package com.tutorialspoint;

import javax.servlet.jsp.tagext.\*;

import javax.servlet.jsp.\*;

import java.io.\*;

public class HelloTag extends SimpleTagSupport {

StringWriter sw = new StringWriter();

public void doTag()

throws JspException, IOException

{

getJspBody().invoke(sw);

getJspContext().getOut().println(sw.toString());

}

}

Trong trường hợp này, kết quả từ sự triệu hồi, đầu tiên được bắt vào trong một StringWriter trước khi được ghi tới JspWriter mà liên kết với thẻ đó. Bây giờ, theo đó, chúng ta cần thay đổi TLD file như sau:

<taglib>

<tlib-version>1.0</tlib-version>

<jsp-version>2.0</jsp-version>

<short-name>Example TLD with Body</short-name>

<tag>

<name>Hello</name>

<tag-class>com.tutorialspoint.HelloTag</tag-class>

<body-content>scriptless</body-content>

</tag>

</taglib>

Chúng ta gọi thẻ trên với phần thân chính xác như sau:

<%@ taglib prefix="ex" uri="WEB-INF/custom.tld"%>

<html>

<head>

<title>A sample custom tag</title>

</head>

<body>

<ex:Hello>

This is message body

</ex:Hello>

</body>

</html>

Nó sẽ cho kết quả:

This is message body

## Thuộc tính của Custom Tag trong JSP

Bạn có thể sử dụng các thuộc tính đa dạng cùng với các thẻ custom tag của bạn. Để chấp nhận một giá trị thuộc tính, một lớp custom tag cần triển khai các phương thức setter, như phương thức setter trong JavaBeans như sau:

package com.tutorialspoint;

import javax.servlet.jsp.tagext.\*;

import javax.servlet.jsp.\*;

import java.io.\*;

public class HelloTag extends SimpleTagSupport {

private String message;

public void setMessage(String msg) {

this.message = msg;

}

StringWriter sw = new StringWriter();

public void doTag()

throws JspException, IOException

{

if (message != null) {

/\* Use message from attribute \*/

JspWriter out = getJspContext().getOut();

out.println( message );

}

else {

/\* use message from the body \*/

getJspBody().invoke(sw);

getJspContext().getOut().println(sw.toString());

}

}

}

Tên của thuộc tính là "message", vì thể phương thức setter là setMessage(). Bây giờ chúng ta thêm thuộc tính này trong TLD file bởi sử dụng phần tử <attribute> như sau:

<taglib>

<tlib-version>1.0</tlib-version>

<jsp-version>2.0</jsp-version>

<short-name>Example TLD with Body</short-name>

<tag>

<name>Hello</name>

<tag-class>com.tutorialspoint.HelloTag</tag-class>

<body-content>scriptless</body-content>

<attribute>

<name>message</name>

</attribute>

</tag>

</taglib>

Thử JSP trên với thuộc tính message như sau:

<%@ taglib prefix="ex" uri="WEB-INF/custom.tld"%>

<html>

<head>

<title>A sample custom tag</title>

</head>

<body>

<ex:Hello message="This is custom tag" />

</body>

</html>

Nó sẽ cho kết quả sau:

This is custom tag

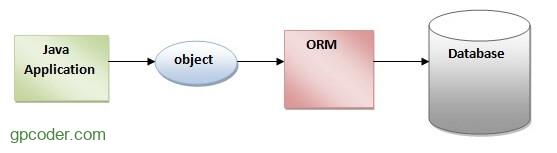
**15.** **Cách hoạt động của Hibernate.Ví dụ**

## Hibernate Framework là gì?

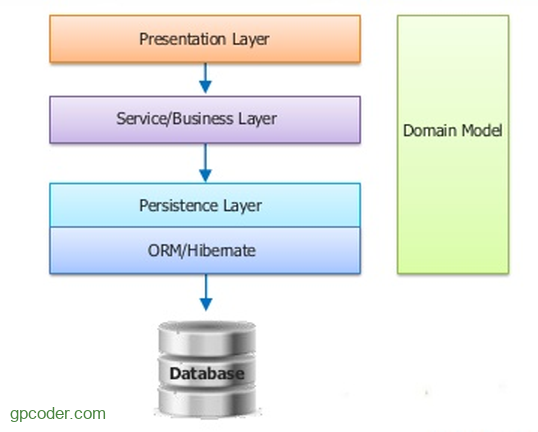
### ORM

Như chúng ta đã biết, **ORM** (Object Relational Mapping) framework là một cơ chế cho phép người lập trình thao tác với database một cách hoàn toàn tự nhiên thông qua các đối tượng. Lập trình viên hoàn toàn không quan tâm đến loại database sử dụng SQL Server, MySQL, PostgreSQL, …

ORM giúp đơn giản hoá việc tạo ra dữ liệu, thao tác dữ liệu và truy cập dữ liệu. Đó là một kỹ thuật lập trình để ánh xạ đối tượng vào dữ liệu được lưu trữ trong cơ sở dữ liệu.

[](https://gpcoder.com/wp-content/uploads/2019/10/orm.jpg)

### Peristence layer

[](https://gpcoder.com/wp-content/uploads/2019/10/persistence-layers.png)

Một ứng dụng có thể được chia làm 3 phần như sau: giao diện người dùng (**presentation layer**), phần xử lý nghiệp vụ (**business layer**) và phần chứa dữ liệu (**data layer**). Cụ thể ra, business layer có thể được chia nhỏ thành 2 layer con là business logic layer và persistence layer.

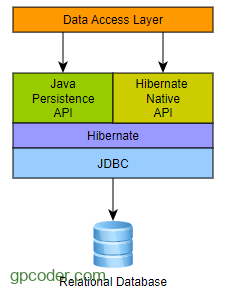
* **Business logic layer**: các tính toán logic nhằm thỏa mãn yêu cầu người dùng.
* **Persistence layer**: chịu trách nhiệm giao tiếp với data layer (thường là một hệ quản trị cơ sở dữ liệu quan hệ – Relational DBMS). Persistence sẽ đảm nhiệm các nhiệm vụ mở kết nối, truy xuất và lưu trữ dữ liệu vào các Relational DBMS.

### Hibernate

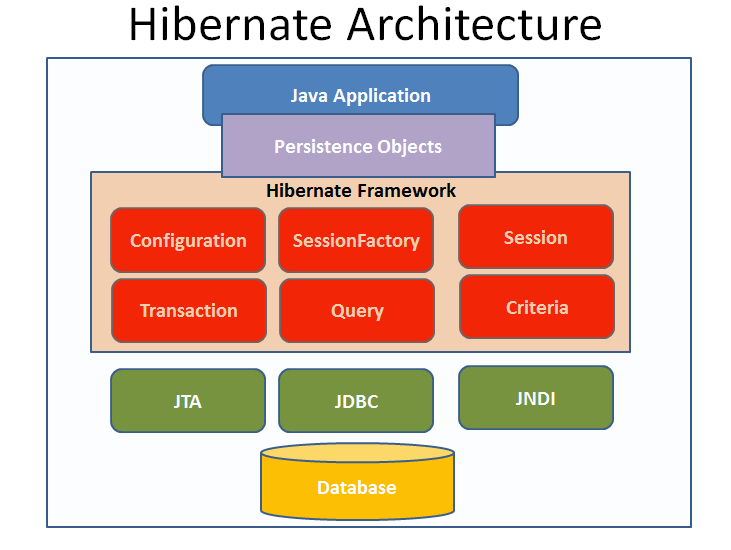
[**Hibernate**](https://hibernate.org/)là một trong những ORM Framework. Hibernate framework là một framework cho persistence layer. Như vậy, nhờ có Hibernate framework mà giờ đây khi bạn phát triển ứng dụng bạn chỉ còn chú tâm vào những layer khác mà không phải bận tâm nhiều về persistence layer nữa.

Hibernate giúp lập trình viên viết ứng dụng Java có thể map các object (POJO) với hệ quản trị cơ sở dữ liệu quan hệ (database), và hỗ trợ thực hiện các khái niệm lập trình hướng đối tượng với cơ dữ liệu quan hệ.

Hibernate giúp lưu trữ và truy vấn dữ liệu quan hệ mạnh mẽ và nhanh. Hibernate cho phép bạn truy vẫn dữ liệu thông qua Java Persistence API (JPA) hoặc bằng ngôn ngữ SQL mở rộng của Hibernate (HQL) hoặc bằng SQL thuần (Native SQL).

[](https://gpcoder.com/wp-content/uploads/2019/10/data-access-layer.png)

Hibernate vốn là một thư viện sinh ra để làm việc với mọi loại database, nó không phụ thuộc vào bạn chọn loại database nào. Nếu **Java** là “Viết 1 lần chạy mọi nơi” thì **Hibernate** là “Viết 1 lần chạy trên mọi loại database”.

Hibernate là một thư viện ORM (Object Relational Mapping) mã nguồn mở giúp lập trình viên viết ứng dụng Java có thể map các objects (pojo) với hệ quản trị cơ sở dữ liệu quan hệ,  
và hỗ trợ thực hiện các khái niệm lập trình hướng đối tượng với cớ dữ liệu quan hệ.  
**Hibernate Workflow**  
  
**Persistence object**  
Chính là các POJO object map với các table tương ứng của cơ sở dữ liệu quan hệ. Nó như là những “thùng xe” chứa dữ liệu từ ứng dụng để ghi xuống database, hay chứa dữ liệu tải lên ứng dụng từ database.  
**Session Factory**  
Là một interface giúp tạo ra session kết nối đến database bằng cách đọc các cấu hình trong Hibernate configuration. Mỗi một database phải có một session factory.

Tỉ dụ nếu ta sử dụng MySQL, và Oracle cho ứng dụng Java của mình thì ta cần có một session factory cho MySQL, và một session factory cho Oracle.

**Hibernate Session**  
Mỗi một đối tượng session được Session factory tạo ra sẽ tạo một kết nối đến database.  
**Transation**  
Là transaction đảm bảo tính toàn vẹn của phiên làm việc với cớ sở dữ liệu. Tức là nếu có một lỗi xảy ra trong transaction thì tất cả các tác vụ thực hiện sẽ thất bại.  
**Query**  
Hibernate cung cấp các câu chuy vấn HQL (Hibernate Query Language) tới database và map kết quả trả về với đối tượng tương ứng của ứng dụng Java.

Nếu bạn muốn làm việc với Hibernate thử hãy xem Tutorial hướng dẫn sử dụng[Hibernate cho người mới bắt đầu tại lcdung.top](http://lcdung.top/)

**16.PrimeFaces**

# PrimeFaces là gì?

Primefaces là một framework mã nguồn mở cho JSF (JavaServer Faces) với hơn 100 component, tối ưu cho mobile, validate phía client, hỗ trợ AJAX mạnh mẽ…

PrimeFaces cũng được coi là một trong những thư viện UI phổ biến nhất trong hệ sinh thái Java, được sử dụng rộng rãi bởi nhiều công ty, tập đoàn lớn trên thế giới như: Cisco, Ebay, HP, Ford…

Ở series này mình sẽ nói về PrimeFaces Showcase – thư viện UI dựa trên JSF  
(Ngoài PrimeFaces Showcase thì còn có PrimeFacesNG base trên Angular JS, PrimeFacesReact base trên ReactJS…)

# 2. Tại sao dùng PrimeFaces?

Ở loạt bài về JSF – [Xử lý AJAX trong JSF](https://stackjava.com/jsf/series-jsf-phan-10-xu-ly-ajax-trong-jsf-framework.html) mình có nhắc tới JSF xử lý AJAX rất mạnh, thì với PrimeFaces khả năng xử lý ấy còn mạnh mẽ hơn ở mức từng component, tùy chỉnh linh hoạt…

* Đơn giản và hiệu suất: PrimeFaces rất nhẹ, cấu hình đơn giản
* Dễ sử dụng
* Cộng đồng lớn
* Kết hợp với Spring: Được Spring khuyên dùng trong các ứng dụng Spring – JSF