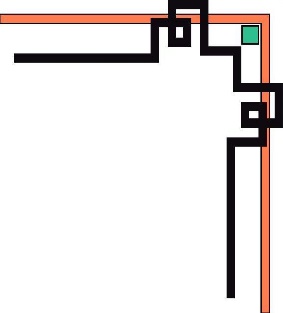
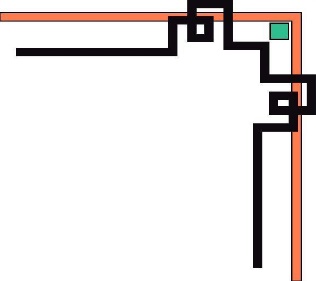
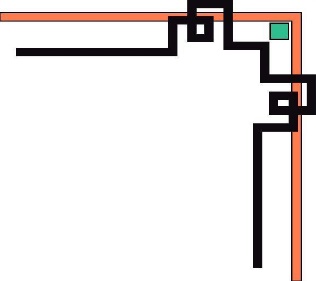
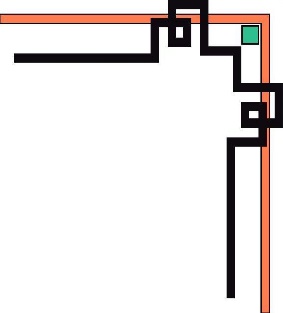
**SPRING FRAMEWORK**





**Thực hiện :**

* Lê Trọng Nghĩa
* Lê Hoàng Phú Tiên
* Tô Trần Minh Nhựt

TP. Hồ Chí Minh

**MỤC LỤC**

[1. Tổng quan. 1](#_Toc462838792)

[2. Giới thiệu. 2](#_Toc462838793)

[**2.1.** **Dependency Injection và Inversion of Control.** 2](#_Toc462838794)

[**2.2.** **Modules.** 5](#_Toc462838795)

[3. Những công nghệ cốt lõi 9](#_Toc462838796)

[**3.1.** **IoC container – Giới thiệu về Spring IoC container và beans** 9](#_Toc462838797)

[**3.2.** **IoC container – Tổng quan Bean, Container** 10](#_Toc462838798)

[**3.2.1.** **Container** 10](#_Toc462838799)

[**3.2.2.** **Bean** 13](#_Toc462838800)

[**3.3.** **IoC container - Dependencies.** 15](#_Toc462838801)

[**3.4.** **IoC container - Bean Scopes** 18](#_Toc462838802)

[**3.5.** **IoC container - Annotation-based container configuration** 21](#_Toc462838803)

[**3.6.** **IoC container – Quét Classpath và quản lý components.** 23](#_Toc462838804)

[**3.7.** **Data Access – Giới thiệu** 24](#_Toc462838805)

[**3.8.** **Data Access - DAO Support** 26](#_Toc462838806)

[**3.9.** **Data Access - ORM Data Access(Especially JPA)** 28](#_Toc462838807)

[4. Spring MVC framework. 34](#_Toc462838808)

[**4.1.** **Giới thiệu Spring MVC Framework** 36](#_Toc462838809)

[**4.2.** **Architecture - Controllers** 37](#_Toc462838810)

[**4.3.** **Architecture - Controllers Responsibilities** 39](#_Toc462838811)

[**4.4.** **Architecture - Controllers Annotations** 40](#_Toc462838812)

[**4.5.** **Architecture - Parameter Passing** 45](#_Toc462838813)

[**4.6.** **Architecture – thuộc tính Model** 47](#_Toc462838814)

[**4.7.** **View – Giới thiệu** 49](#_Toc462838815)

[**4.8.** **View – Các loại View** 50](#_Toc462838816)

[**4.9.** **View - View conventions** 54](#_Toc462838817)

[**4.10.** **View - Resolving a View** 56](#_Toc462838818)

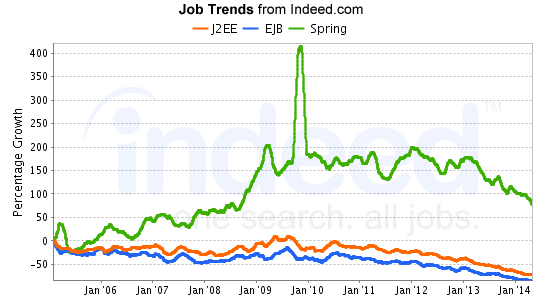
[**4.11.** **View - JSP & JSTL** 59](#_Toc462838819)

[5. Tài liệu tham khảo 73](#_Toc462838820)

# Tổng quan.

Spring là ban đầu được viết bởi Rod Johnson và lần đầu tiên được phát hành theo giấy phép Apache 2.0 trong tháng 6 năm 2003. Spring cho phép bạn chọn lựa các module bạn cần để cài đặt mà không cài đặt tất cả các module của nó.

Spring Framework là một chọn lựa, thay thế cho mô hình chuẩn EJB trong việc phát triển ứng dụng JavaEE đã đánh dấu một bước ngoặt trong lịch sử phát triển của Enterprise Java. Spring không ngừng phát triển và ngày càng phổ biến trong cộng đồng Java. Theo thống kê của [trang web tìm kiếm việc làm Indeed](http://www.indeed.com/) đến đầu năm 2014, tỉ lệ gia tăng việc làm trên Spring luôn cao hơn hẳn EJB.



# Giới thiệu.

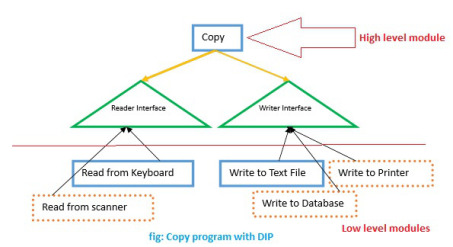
Spring framework là một nền tảng java mã nguồn mở cung cấp cơ sở hạ tầng một cách toàn diện (APIs) để hỗ trợ việc xây dựng các ứng dụng khác nhau: web, mobile, test, application,... Do đó, người lập trình có thể tập trung vào việc xây dựng ứng dụng.

Spring framework được xây dựng dựa trên POJOs (Plain old java objects). Là những Object java bình thường - chúng không có gì đặc biệt, không theo bất kỳ một mô hình, quy ước nào cả chỉ đơn giản bao gồm các thuộc tính và các phương thức. Chỉ phụ thuộc vào người viết code và những gì Java chỉ định (kiểu như nếu Java không cho viết thì coder không thể viết).

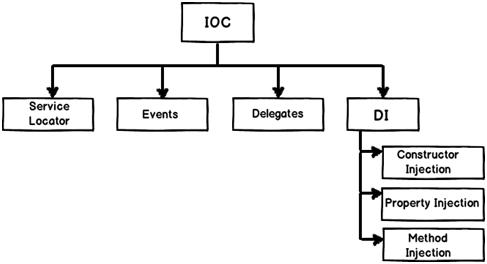
* 1. **Dependency Injection và Inversion of Control.**

Hai thành phần quan trọng của Spring chính là DI (Dependency Injection) và IoC (Inversion of Control). Để làm rõ DI, các bạn cần phân biệt 3 khái niệm sau:

* **Đảo ngược phụ thuộc (Dependency Inversion)**: Đây là nguyên lý cuối cùng trong 5 nguyên lý cơ bản trong thiết kế hướng đối tượng SOLID. Nguyên lý này chỉ ra rằng các lớp high-level không được phụ thuộc vào các lớp low-level. Thay vì để các lớp high-level sẽ sử dụng các interface do các lớp low-level định nghĩa và thực thi, thì nguyên lý này chỉ ra rằng các lớp high-level sẽ định nghĩa ra các interface, sau đó các lớp low-level sẽ thực thi các interface đó.



* **Inversion of Control:** Đây là một [design pattern](https://toidicodedao.com/2016/03/01/nhap-mon-design-pattern-phong-cach-kiem-hiep/) được tạo ra để code có thể tuân thủ nguyên lý Dependency Inversion. Có nhiều cách hiện thực pattern này: ServiceLocator, Event, [Delegate](https://toidicodedao.com/2015/02/10/series-c-hay-ho-callback-trong-c-delegate-action-predicate-func/), … Dependency Injection là một trong các cách đó.

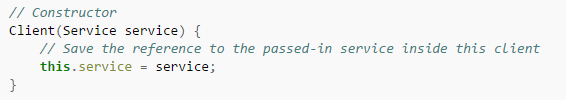


* **Dependency Injection:** Các module không giao tiếp trực tiếp với nhau, mà thông qua interface. Module cấp thấp sẽ implement interface, module cấp cao sẽ gọi module cấp thấp. Ví dụ: Để giao tiếp với database, ta có interface IDatabase, các module cấp thấp là XMLDatabase, SQLDatabase. Module cấp cao là CustomerBusiness sẽ sử dụng interface IDatabase.

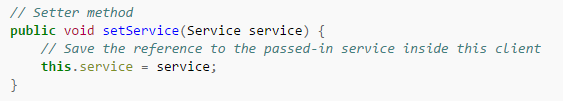
Việc khởi tạo các module cấp thấp sẽ do DI Container thực hiện. Ví dụ: Trong module CustomerBusiness, ta sẽ không khởi tạo IDatabase db = new XMLDatabase(), việc này sẽ do DI Container thực hiện. Module CustomerBusiness sẽ không biết gì về module XMLDatabase hay SQLDatabase. Việc Module nào gắn với interface nào sẽ được config trong code hoặc trong file XML.

Có 3 dạng Dependency Injection:

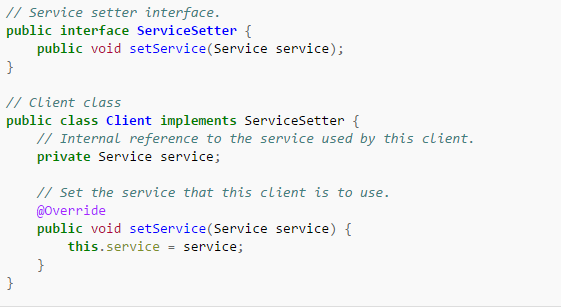
* Constructor Injection: Các dependency sẽ được container truyền vào (inject vào) 1 class thông qua constructor của class đó. Đây là cách thông dụng nhất.



* Setter Injection: Các dependency sẽ được truyền vào 1 class thông qua các hàm Setter.



* Interface Injection: Class cần inject sẽ implement 1 interface. Interface này chứa 1 hàm tên Inject. Container sẽ injection dependency vào 1 class thông qua việc gọi hàm Inject của interface đó. Đây là cách rườm rà và ít được sử dụng nhất.



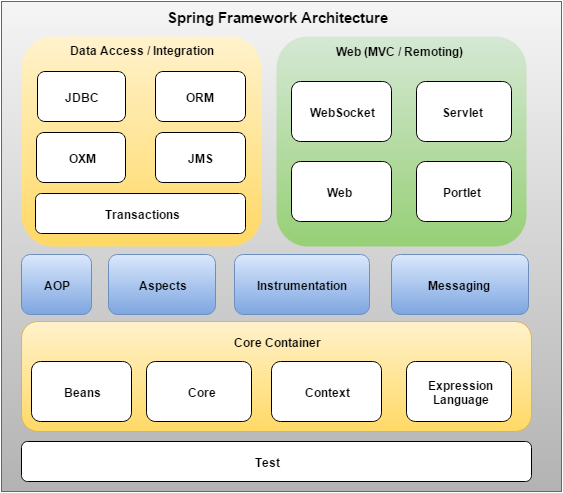
Ưu điểm:

* Giảm sự kết dính giữa các module.
* Code dễ bảo trì và thay thế các module.
* Rất dễ test và viết Unit test.
* Dễ dàng thấy được các quan hệ của module thông qua các constructor.

Nhược điểm:

* Khó debug do sử dụng interface, không biết được chính xác module nào được gọi.
* Các đối tượng được khởi tạo ngay từ đầu, làm giảm hiệu năng hệ thống.
* Làm tăng độ phức tạp của nó.
  1. **Modules.**

Spring bao gồm nhiều tính năng được tổ chức thành các module. Các module được chia thành các nhóm như mô hình sau:



Core container bao gồm 4 module: core, bean, context, expression language.

* Core module: module này cung cấp các thành phần cơ bản của spring framework, bao gồm IoC và Dependency Injection.
* Bean module: module này cung cấp BeanFactory – là một sự triển khai của Facetory pattern.
* Context module: module này xây dựng trên một nền tảng vững chắc được cung cấp bởi Core và Bean module. Nó cũng là một môi trường để truy cập vào bất kỳ đối tượng nào được định nghĩa và cấu hình. Interface Application context là thành phần quan trọng của Context module.
* Expression language module: module này cung cấp một dạng ngôn ngữ rất tiện ích cho việc truy vấn và thao tác các đối tượng tại thời điểm runtime.

Data access / intergration bao gồm các module: JDBC, ORM, OXM, JMS và transaction.

* JDBC module: module này cung cấp một lớp JDBC – abstraction hỗ trợ các công việc liên quan đến database.
* ORM module: module này cung cấp một lớp abstraction – hỗ trợ việc thực hiện mapping Object/XML cho JAXB, CASTOR, XMLBeans, JiBX và XStream.
* JMS (java message service) module cung cấp các chức năng cho việc gửi và nhận message.
* Transaction module: module này hỗ trợ lập trình và quản lý transaction.

Web chứa đựng các module: web, web – MVC, web – Socket, web – Portlet.

Web module cung cấp các chức năng tích hợp Web – oriented như là chức năng upload và khởi tạo IoC container – sử dụng servlet và application context web.

* Web –MVC module chứa các thành phần của một MVC pattern để xây dựng ứng dụng web.
* Web – Porlet module cung cấp việc thực hiện MVC để sử dụng trong môi trường portlet và phản ánh chức năng của module web – servlet.

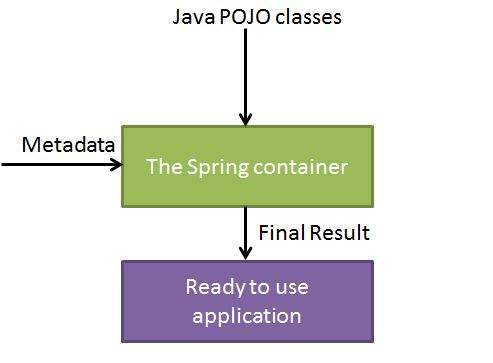
Một vài module quan trọng khác như: AOP, Aspect, Instrumentation, Messaging và Test.

* AOP module cung cấp phương thức lập trình aspect-oriented cho phép người lập trình định nghĩa các method-interceptor và phân chia các block code thực hiện các chức năng.
* Aspect module cung cấp sự tích hợp với AspectJ hỗ trợ rất nhiều cho AOP.
* Instrumentation module cung cấp các class hỗ trợ và loader để sử dụng trong ứng dụng nhất định.
* Messaging module cung cấp việc hỗ trợ cho STOM như WebSocket sub - protool để sử dụng trọng ứng dụng. Nó cũng hỗ trợ cho việc lập tình sử dụng annotation cho việc routing và xử lí các message STOMP từ Websocket client.
* Test module hỗ trợ việc test cho các thành phần Spring với Junit hoặc TestNG framework.

# Những công nghệ cốt lõi

* 1. **IoC container – Giới thiệu về Spring IoC container và beans**

IoC container là cốt lỗi của Spring framework và được xây dựng dựa trên nguyên lý Inversion of Control. Nó sẽ tạo ra các đối tượng, gom nhóm các đối tượng, cấu hình các đối tượng, và quản lý toàn bộ chu kì sống của chúng từ lúc khởi tạo cho đến lúc hủy. IoC container dùng những POJO class và những dữ liệu cấu hình để tạo ra một hệ thống hoặc ứng dụng.

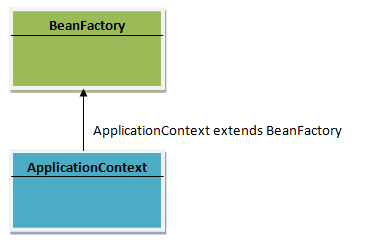


Nền tảng cho IoC container bao gồm các package:

* org.springframework.beans
* org.springframework.context là

BeanFactory interface trong package ‘.bean’ cung cấp cấu hình framework và chức năng cơ bản. Nhưng nó chỉ được sử dụng trong việc tích hợp với các frameworks thứ ba khác. Beanfactory được khuyến khích cho các lightweight framework như mobile, applet.

ApplicationContext interface là một sub-interface của BeanFactory. Nó có thêm các chức năng khác: AOP, xử lý dữ liệu nguồn cho việc sử dụng đa ngôn ngữ, truyển tải các sự kiện (event propagation), tầng ứng dụng cụ thể có các ứng dụng web ví dụ WebApplicationContext.

[](https://techythought.files.wordpress.com/2013/01/con1.png)

Trong Spring, các đối tượng được tạo ra và quản lý bởi IoC container được gọi là Beans.

* 1. **IoC container – Tổng quan Bean, Container** 
     1. **Container**

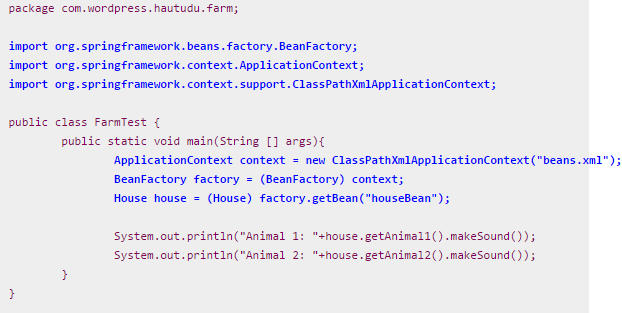
Interface org.springframework.context.ApplicationContext biểu diễn spring IoC container và chịu trách nhiệm cho việc khởi tạo, cấu hình và ghép nối các beans được tạo ra. Để thực hiện các việc khởi tạo, cấu hình và ghép nối các đối beans, Container sẽ đọc configuration metadata.

Những configuration metadata được định nghĩa trong XML. Configuration metadata định nghĩa làm thế nào mà Spring container khởi tạo. cấu hình và liên kết các đối tượng trong ứng dụng của bạn. Dưới đây là một file xml cấu trúc đơn giản của configuration metadata:



Ta có thể thấy cách mà các thành phần được khởi tạo và liên kết với nhau trong file cấu hình trên. Tag <bean> được sử dụng để biểu diễn một đối tượng của một lớp nào đó. Mỗi đối tượng sẽ có một id và kiểu được thể hiện lần lượt qua các thuộc tính id và class bên trong tag <bean>. Ở file trên, ta có 3 tag <bean> để khởi tạo 3 đối tượng của ba lớp tương ứng là Dog, Cat, và House. Đó là cách để khởi tạo một thành phần, ngoài ra đối tượng của lớp House có hai tham chiếu đến hai đối tượng của hai lớp Dog và Cat bằng việc sử dụng thuộc tính ref trong thẻ <property>, đó là cách các thành phần được liên kết với nhau trong Spring.

Tạo và sử dụng container, xét ví dụ:



Một đối tượng của ApplicationContext (container) sẽ được khởi tạo và được cung cấp về vị trí của file cấu hình. Trong ví dụ này, đối tượng context trên sẽ là một đối tượng của lớp ClassPathXmlApplicationContext và file cấu hình chính là beans.xml.

Dùng getBean() để gọi ra các beans từ file cấu hình, Ở đây từ dùng getBean() để gọi ra các bean trong file bean.xml.

ApplicationContext được cài đặt cụ thể bởi nhưng lớp sau:

* ClassPathXmlApplicationContext: Được dùng để load các thông tin từ file cấu hình nằm trong classpath của ứng dụng. Chính vì thế mà chúng ta thấy tại sao file beans.xml trong ví dụ trên được đặt nằm bên trong thư mục src của project
* FileSystemXmlApplicationContext: Được dùng để load các thông tin từ file cấu hình nằm trong hệ thống file trên máy của chúng ta. Ví dụ nếu file cấu hình beans.xml nằm trong ổ C thay vì nằm trong project.
* XmlWebApplicationContext: Được dùng để load các thông tin từ file cấu hình nằm trong ứng dụng web. Mặc định, thông tin sẽ được load từ file cấu hình /WEB-INF/applicationContext.xml
  + 1. **Bean**

Spring IoC có thể quản lý 1 hay nhiều beans. Những beans này được IoC container tạo ra dựa vào configuration metadata mà bạn cung cấp cho container.

Trong file cấu hình xml ở ví dụ trên, một bean được định nghĩa bằng một thẻ </bean>. Một bean sẽ được cấu hình bởi những thông tin sau: Cách tạo ra bean, vòng đời của bean, sự phụ thuộc của bean. Tất cả những thông tin này được thể hiện bởi một tập các thuộc tính sau:

|  |  |
| --- | --- |
| **thuộc tính** | **Mô tả** |
| class | Đây là thuộc tính bắt buộc, và nó dùng để chỉ class dùng được dùng để tạo ra bean. |
| name / id | Đây là thuộc tính dùng để chỉ định danh của bean, và nó là thuộc tính duy nhất. Nghĩa là không được có một bean này khác cùng name / id. |
| scope | Thuộc tính xác định phạm vi của một đối tượng trong ứng được tạo ra từ bean trong file cấu hình. (Chi tiết ở bên dưới phần Bean scope). |
| constructor-arg | Thuộc tính dùng để truyền dependencies. |
| properties | Thuộc tính dùng để truyền dependencies. |
| autowiring mode | Thuộc tính dùng để truyền dependencies. |
| lazy-initialization mode | Thuộc tính dùng gọi IoC container tạo một bean instance. |
| initialization method | Thuộc tính chỉ một callback được gọi sau khi tất cả thuộc tính cần thiết của một bean được thiết lập bởi container |
| destruction method | Thuộc tính chỉ một callback được gọi khi container chứa bean bị hủy. |

Có 3 cách cung cấp configuration metadata cho container:

* Thông qua xml based configuration file
* Thông qua annotation-based configuration (Chi tiết ở phần sau)
* Thông qua Java-based configuration (Chi tiết ở phần sau).

Ví dụ về xml based configuration file



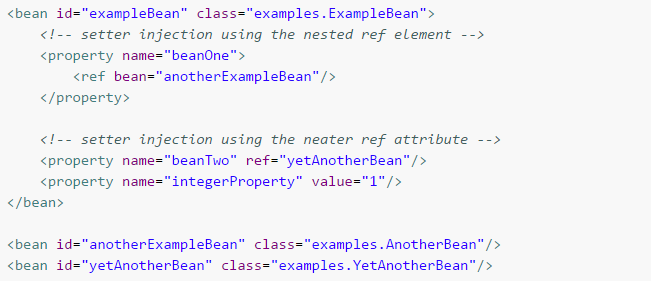
* 1. **IoC container - Dependencies.**

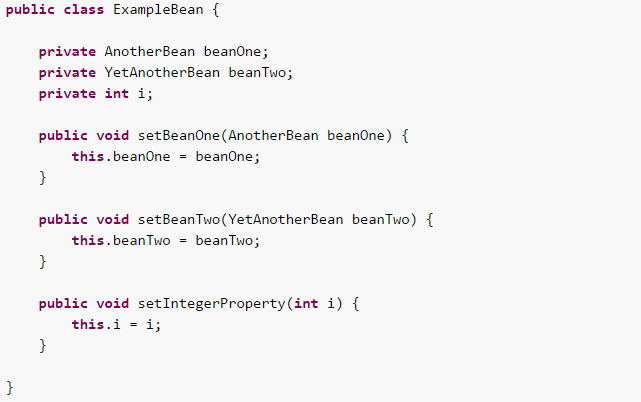
Trong spring framework. quá trình phân rã phụ thuộc như sau:

* Các ApplicationContext được tạo ra và khởi tạo bởi cấu hình siêu dữ liệu (configuration metadata) - mô tả các bean. Cấu hình siêu dữ liệu (configuration metadata) được mô tả bằng xml, code hoặc chú thích.
* Với mỗi bean, phụ thuộc của nó được thể hiện dưới hình thức các thuộc tính (property), các đối số hàm khởi tạo (constructor), hoặc đối số cho phương thức static-factory nếu bạn sử dụng thay cho việc sử dụng hàm khởi tạo (constructor) bình thường. Những phụ thuộc được cung cấp cho bean khi bean thật sự được tạo ra.
* Mỗi thuộc tính (property) hoặc đối số hàm constructor để thiết lập giá trị hoặc để tham chiếu đến bean khác trong container.
* Mỗi thuộc tính (property) hoặc đối số hàm constructor là một giá trị từ định dạng cụ thể của nó với kiểu thực tế của thuộc tính (property) hoặc đối số hàm constructor. Mặc định trong spring có thể chuyển đổi một giá trị được cung cấp bởi định dạng chuỗi để xây dựng các kiểu như int, long, String, boolean,...

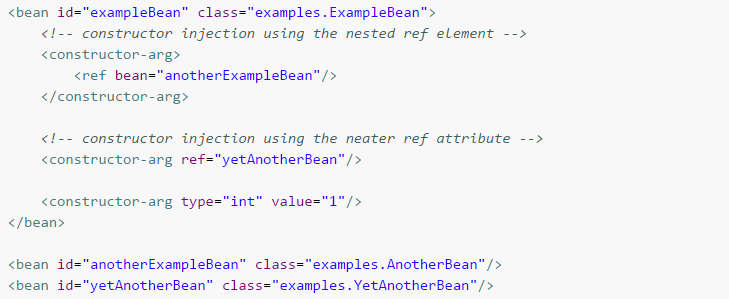
Các container spring xác nhận cấu hình của mỗi bean khi container được tạo ra. Tuy nhiên, các thuộc tính của bean không được thiết lập cho đến khi bean được tạo ra. Bean là singleton - scoped và được thiết lập trước khi được container tạo ta. Phạm vi được gọi là bean scopes. Nếu không có bean scopes, các bean chỉ được tạo ra khi nó được yêu cầu. Trong spring, DI bao gồm 2 loại:

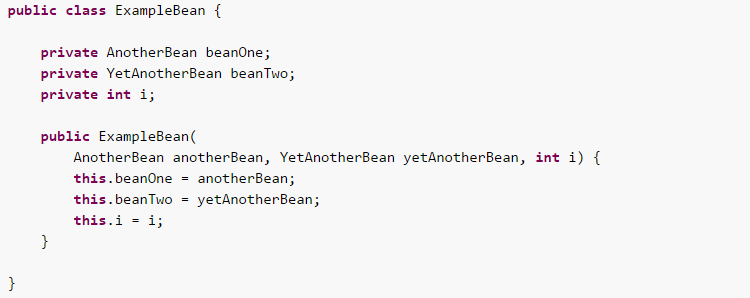
* Đây là phương pháp DI phổ biến nhất và đơn giản, nó sẽ tiêm phụ thuộc thông qua một phương thức setter.





* Tiêm phụ thuộc thông qua constructor.





* 1. **IoC container - Bean Scopes**

Khi định nghĩa một bean trong Spring, chúng ta còn phải định nghĩa scope của bean. Việc định nghĩa scope có thể thực hiện thông qua việc sử dụng thuộc tính tên là “scope” khi định nghĩa. Lấy ví dụ, khi bean phải tạo mới mỗi lần cần sử dụng, thuộc tính scope sẽ là “prototype”. Mặt khác, khi bean luôn luôn trả về một instance giống nhau khi sử dụng, thuộc tính scope sẽ là “singleton”.

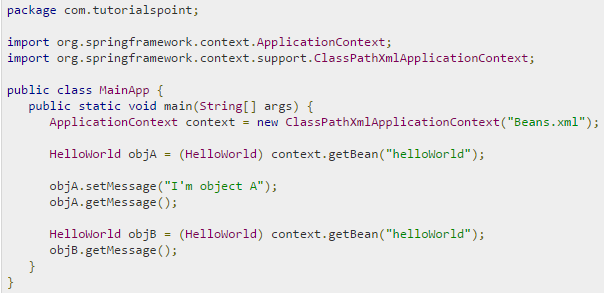
|  |  |
| --- | --- |
| Phạm vi | Miêu tả |
| singleton | Cho biết bean đó có một instance duy nhất trong Sping IoC container. |
| prototype | Cho biết bean đó được định nghĩa là có nhiều object instances, mỗi lần muốn sử dụng sẽ tạo mới. |
| request | Cho biết bean được định nghĩa với một HTTP request. Scope này chỉ hợp lệ khi chúng ta sử dụng Web Application Context. |
| Session | Cho biết bean được định nghĩa với một HTTP session. Scope này cũng chỉ hợp lệ khi chúng ta sử dụng Web Application Context. |
| global - session | Cho biết bean được định nghĩa với một global HTTP session. Scope này cũng chỉ hợp lệ khi chúng ta sử dụng Web Application Context. |

Hai scope quan trọng của bean là: singleton và prototype. Sau đây là 2 ví dụ cho singleton và prototype.

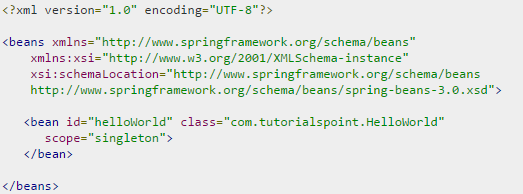
Dưới đây là nội dung của **HelloWorld.java**:



Sau đây là nội dung của **MainApp.java** file:



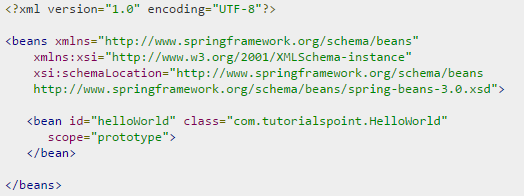
* Sau đây là tập tin cấu hình **Beans.xml** cần thiết cho phạm vi singleton:



Kết quả:



* Sau đây là tập tin cấu hình **Beans.xml** cần thiết cho phạm vi protype:

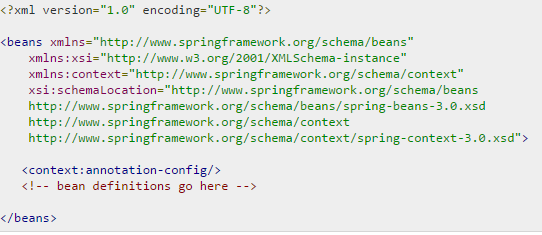


Kết quả:



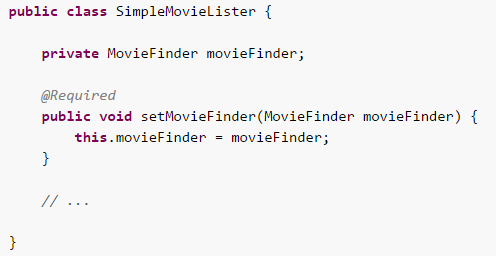
* 1. **IoC container - Annotation-based container configuration**

Annotation-based configuration là thông tin cấu hình được cung cấp đến spring container để tạo ra beans. Thay vì dùng xml để mô tả bean, ta sẽ đưa cấu hình vào trong các class thành phần của chính nó bằng cách sử dụng annotation trên class, method, hoặc các trường khai báo. Annotation mặc định không được hỗ trợ trong spring container, do đó nếu muốn sử dụng thì phải vào file cấu hình của spring chỉnh sửa để container có thể hỗ trợ.

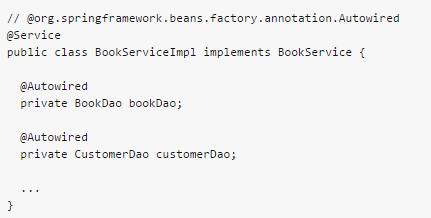


Thẻ <context: annotation-config/> để start annotation. Các annotation cần nắm:

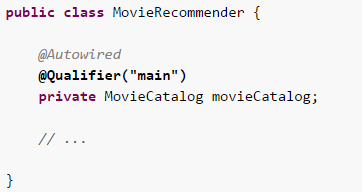
* **@Required:** Annotation này đơn giản chỉ ra rằng thuộc tính nào của bean phải được cài đặt tại thời điểm config, thông qua khai báo tường minh hay thông qua autowiring. Nếu những thuộc tính của bean không thể được cài đặt thì container sẽ throw **BeanInitializationException**



* **@Autowired**: Tự động liên kết các bean được sử dụng trong các class với các bean được Spring Container sinh ra và quản lý.



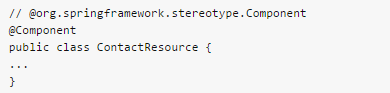
* **@Qualifier:** Khi có nhiều hơn một bean với cùng một loại và chỉ có một bean trong số đó cần được wire với một property nào đó, **@Qualifier** sẽ được sử dụng với **@Autowired** để giảm thiểu sự nhầm lẫn bằng cách định danh chính xác bean nào được wire.



* 1. **IoC container – Quét Classpath và quản lý components.**

Loại bỏ sự cần thiết phải sử dụng XML để thực hiện định nghĩa bean thay vào đó bạn có thể sử dụng các chú thích (ví dụ @Component), biểu thức AspectJ loại, hoặc các tiêu chí lọc riêng của bạn để chọn các lớp sẽ có các định nghĩa bean với các container.

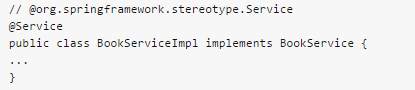
* **@Component:** dùng để *scan* các Spring Components.



* **@Repository:** dùng để đánh dấu các lớp DAO của Spring. Mọi truy cập dữ liệu logic tới cơ sở dữ liệu cần phải đặt trong các lớp DAO.



* **@Service:** dùng để đánh dấu các lớp Service. Tất cả xử lý nghiệp vụ logic cần phải đặt trong các lớpService.

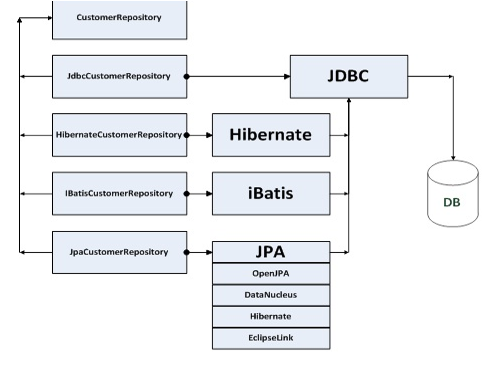


* 1. **Data Access –** **Giới thiệu**

Một trong những khía cạnh rất quan trọng trong bất kỳ ứng dụng Enterprise là khả năng truy xuất và ghi dư liệu của ứng dụng của bạn một cách dễ dàng .

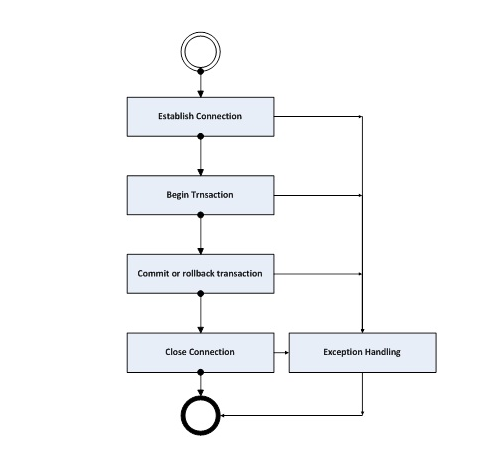
Spring làm cho hiệu quả truy cập dữ liệu dễ dàng hơn bằng cách quản lý các nguồn dữ liệu của bạn thông qua việc trừu tượng hóa và cung câp các APIs mà không làm người xây dựng phụ thuộc vào bất cứ một công nghệ dữ liệu cụ thể nào đó.

Spring kết hợp hoàn hảo hầu hết các công nghệ dữ liệu như là : JDBC, Hibernate, iBatis, JPA or JDO . Chúng ta có thể tự do chuyển đổi qua lại giửa các công nghệ dữ liệu mà không cần lo lắng.



Một số tính năng quan trọng của Spring framework cung cấp trong việc quản lý data access:

* Transaction Management
* Connection Management
* Exception Handling



* 1. **Data Access - DAO Support**

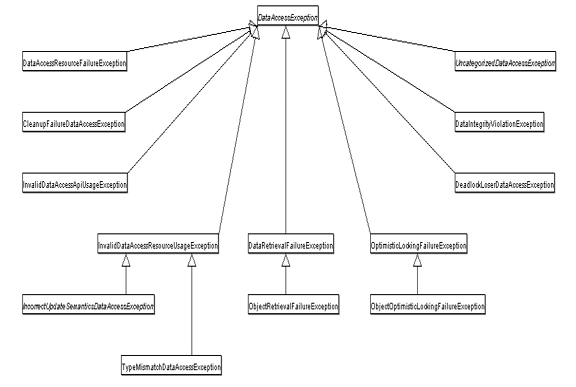
Data access object(DAO) hỗ trợ làm việc môt cách nhất quán đối với các công nghệ truy cập dữ liệu như JDBC, Hibernate, JPA hay JDO. DAO là một abstract class có các hàm dành cho việc thiết lập và sử dụng các công nghệ cơ sở dữ liệu.

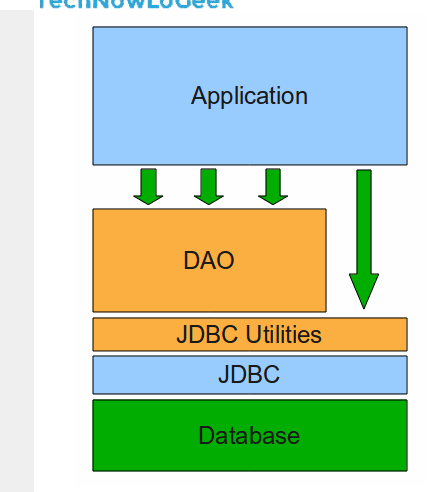
DAO cho phép sự chuyển đổi dữ liệu giữa các công nghệ truy cập dữ liệu một cách dễ dàng. Và chúng ta không cần về việc bắt ngoại lệ trên mỗi công nghệ cụ thể.

DAO cung cấp môt bộ chuyển đổi thuận lợi từ công nghệ bắt lỗi đặc biệt giống như SQLException. Và sẽ phân cấp các exception riêng với DataAccessException là exception gốc. Những ngoại lệ trong DataAccessException luôn bảo đảm rằng sẽ không xãy ra sai sót và thiếu cũng như các Exception trong từng công nghệ cụ thể.

Trừ những exception cua JDBC, Spring có thể bọc các ngoại lệ của Hibernate, JPA, JDO lại và chỉ tập trụng vào ngoại lệ ở runtime. Điều này cho phép xử lý hầu hết các ngoại lệ không thể phục hồi nằm trong duy nhất trong một lớp thích hợp. Mà sẽ không gây rắc rối trong khối try catch và throw.

Dưới đây là mô hình ngoại lệ của DAO:

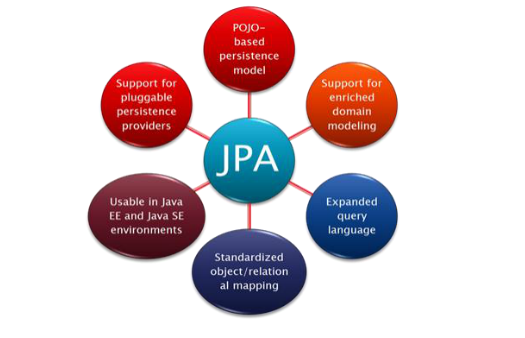




* 1. **Data Access - ORM Data Access(Especially JPA)**

JPA (Java persistence API) :

* Là đặc tả chuẩn cho các tác vụ ở tầng dữ liệu.
* JPA sử dụng các lớp POJO.
* Có nhiều công nghệ ORM hổ trợ miễn phí.
* Hổ trợ cấu hình triển khai annotation và xml
* Ứng dụng có cấu trúc tách biệt rõ ràng



Tại sao nên dùng JPA:

* Viết ít code hơn
* Performance tốt
* Độc lập về database
* Không phải làm việc với SQL

Một số tính năng của JPA

* JPA hỗ trợ pluggable, tức là có thể sử dụng nhiều nhà hãng cung cấp thứ ba như Hibernate hay Toplink.
* Hỗ trợ annotation
* Giảm bớt số lớp yêu cầu cho việc phát triển persistence.
* Không cần phải viết các mô tả triển khai trong xml. Annotation dựa trên metadata đã hỗ trợ trong các ứng dụng JPA.
* Đã chuẩn hóa ORM và dễ dàng phát triển hơn
* JPA hỗ trợ truy vấn động và tĩnh.
* Nhiều IDE hỗ trợ phát triển ứng dụng JPA và có thể tự động sinh code ánh xạ từ cơ sở dữ liệu thành các entity và ngược lại.

Một số ORM framework hỗ trợ JPA:

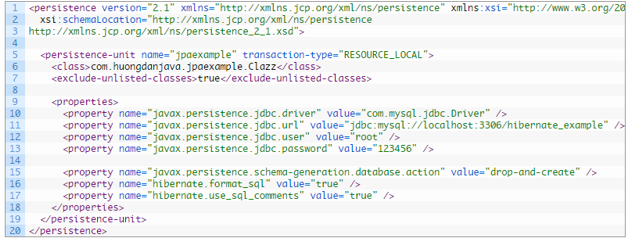
* Hibernate
* EclipseLink
* Open JPA

Một số lợi ích của JPA:

* Đơn giản hóa công nghệ cho tầng persistence (tầng dữ liệu).
* Không phụ thuộc vào các framework ORM.
* Có nhiều cung cấp hỗ trợ cài đặt JPA.
* Dữ liệu được lưu trữ thông qua ORM.

Khái niệm JPA bao gồm ba thành phần chính là Entity, EntityManager, và EntityManagerFactory

* **Entity** là các đối tượng persistence thể hiện một mẫu tin trong cơ sở dữ liệu. Entity chỉ là các lớp POJO đơn giản, dễ dàng tạo. Dưới đây là một số đặc điểm của một Entity:
* Entity có thể tương tác với cơ sở dữ liệu quan hệ.
* Entity được xác định thông qua một định danh( tương đương với khóa chình trong table của cơ sở dữ liệu quan hệ)
* Entity hỗ trợ giao tác (transaction)
* Entity hỗ trợ kế thừa giống như những lớp thường khác.
* **EntityManager** là một giao diện (interface) cung cấp các API cho việc tương tác với các Entity. Một số chức năng cơ bản của EntityManager như:
* Persist: phương thức này dùng để lưu một thực thể mới tạo vào cơ sở dữ liệu
* Merge: dùng để cập nhật trạng thái của entity vào cơ sở dữ liệu.
* Remove: xóa một thể hiện của entity.
* **EntityManagerFactory** được dùng để tạo ra một thể hiện của EntityManager

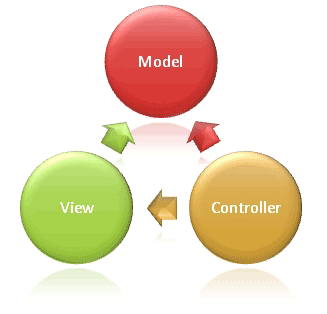




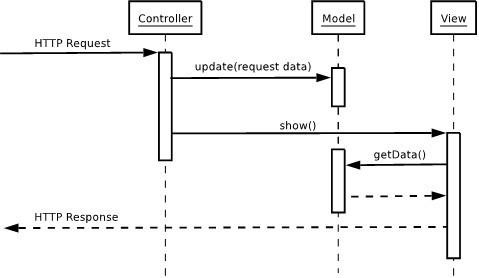
Tag

# Spring MVC framework.

Trong kiến trúc MVC, một đối tượng đồ họa người dùng (GUI Component) bao gồm 3 thành phần cơ bản: Model, View và Controller. Model có trách nhiệm đối với toàn bộ dữ liệu cũng như trạng thái của đối tượng đồ họa. View chính là thể hiện trực quan của Model, hay nói cách khác chính là giao diện của đối tượng đồ họa. Controller điều khiển việc tương tác giữa đối tượng đồ họa với người sử dụng cũng như đối tượng khác.



Khi người sử dụng hoặc những đối tượng khác cần thay đổi trạng thái của đối tượng đồ họa, nó sẽ tương tác thông qua Controller. Controller sẽ thực hiện việc thay đổi trên Model. Khi có bất kỳ sự thay đổi nào xảy ra ở Model, nó sẽ phát thông điệp (broadcast message) thông báo cho View và Controller biết. Nhận được thông điệp từ Model, View sẽ cập nhật lại thể hiện của mình, đảm bảo rằng nó luôn là thể hiện trực quan chính xác của Model. Còn Controller, khi nhận được thông điệp từ Model, sẽ có những tương tác cần thiết phản hồi lại người sử dụng hoặc các đối tượng khác.



Lấy ví dụ một GUI Component đơn giản là Checkbox. Checkbox có thành phần Model để quản lý trạng thái của nó là check hoặc uncheck, thành phần Vew sẽ thể hiện nó ứng với trạng thái của Model lên màn hình, thành phần Controller sẽ xử lý sự kiện khi có sự tương tác của người sử dụng hoặc đối tượng khác lên checkbox.

Khi người sử dụng nhấn chuột vào checkbox, thành phần Controller của checkbox sẽ xử lý sự kiện này, yêu cầu thành phần Model thay đổi dữ liệu trạng thái. Sau khi thay đổi trạng thái, thành phần Model phát thông điệp đến thành phần View và Controller. Thành phần View của checkbox nhận được thông điệp sẽ cập nhật lại thể hiện của checkbox, phản ánh chính xác trạng thái của checkbox do Model lưu giữ. Thành phần Controller nhận được thông điệp do Model gửi tới sẽ có những tương tác phản hồi với người sử dụng nếu cần thiết.

Ưu điểm:

* MVC làm cho ứng dụng trở nên trong sáng, giúp lập trình viên phân tách ứng dụng thành ba lớp một cách rõ ràng. Điều này sẽ rất giúp ích cho việc phát triển những ứng dụng xét về mặt lâu dài cho việc bảo trì và nâng cấp hệ thống.
* MVC hiện đang là mô hình lập trình tiên tiến bậc nhất hiện nay, điều mà các framework vẫn đang nổ lực để hướng tới sự đơn giản và yếu tố lâu dài cho người sử dụng.

Nhược điểm:

* Mặc dù, MVC tỏ ra lợi thế hơn nhiều so với cách lập trình thông thường. Nhưng MVC luôn phải nạp, load những thư viện đồ sộ để xử lý dữ liệu. Chính điều này làm cho mô hình trở nên chậm chạp hơn nhiều so với việc code tay thuần túy.
* MVC đòi hỏi người tiếp cận phải biết qua OOP, có kinh nghiệm tương đối cho việc thiết lập và xây dựng một ứng dụng hoàn chỉnh. Sẽ rất khó khăn nếu OOP của người sử dụng còn yếu.
  1. **Giới thiệu Spring MVC Framework**

Spring's Web MVC framework được thiết kế xung quanh một DispatcherServlet, nó gửi các request đến các handler, với việc cho phép cấu hình các handler mapping, view resolution, locale and theme resolution như là sự hỗ trợ tốt nhất cho việc upload file. Handler mặc định rất đơn giản “Controller interface”, chỉ đưa ra một phương thức ModelAndView handleRequest (request,response). Cái này đã có thể được sử dụng cho các controller của ứng dụng, nhưng sẽ thích hơn khi bao gồm kiến trúc thực thi có thứ bậc, sự nhất quán, ví dụ AbstractController, AbstractCommandController and SimpleFormController. Các controller của ứng dụng sẽ là các lớp con tiêu biểu của chúng. Chú ý rằng chúng ta có thể chọn một lớp cơ sở nếu chúng ta không có một form, chúng ta không cần một form controller. Đây là điều khác biệt chính so với Strusts.

Hình dưới đây mô tả luồng xử lý của một request



* DispatcherServlet nhận request và dựa vào HandlerMapping để tìm và invoke Controller tương ứng.
* Controller xử lý Request và return một ModelAndView object. ModelAndView object này chứa model data và ViewName.
* ViewResolver nhận ViewName từ DispatcherServlet để tìm View tương ứng và return View về cho DispatcherServlet.
* DispatcherServlet sẽ pass Model vào View và response về user.
  1. **Architecture - Controllers**

Controllers là các thành phần được DispatcherServlet gọi để xử lý các nghiệp vụ business. Tất cả các Controller trong Spring đều implement từ Controller interface trong package org.springframework.web.servlet.mvc. Interface controllers định nghĩa 1 phương thức chịu tránh nhiệm xứ lí request và trả về model cùng với view thích hợp.

Một số lớp controllers cơ bản được sử dụng trọng Spring:

* AbstractController:
* ParameterizableViewController:
* SimpleFormController: một form controller cung cấp sự hỗ trợ thậm chí nhiều hơn khi tạo một form với một đối tượng corresponding command. SimpleFormController để cho bạn đặc tả một command object, một viewname cho form, một viewname cho trang bạn muốn hiện thị user khi form submission đã thành công, và thêm nữa.
* AbstractCommandController - một command controller, bạn có thể sử dụng tạo ra command controller của riêng bạn, có khả năng liên kết các tham số request vào đối tượng dữ liệu bạn đặc tả. Class này không đưa ra hình dạng của chức năng; nó làm bằng cách nào đó đưa ra tính năng kiểm chứng (validation) và để bạn đặc tả trong controller của chính nó những gì làm với đối tượng command mà đã được xác định với giá trị tham số của request.
* AbstractCommandController: một command controller, bạn có thể sử dụng tạo ra command controller của riêng bạn, có khả năng liên kết các tham số request vào đối tượng dữ liệu bạn đặc tả. Class này không đưa ra hình dạng của chức năng; nó làm bằng cách nào đó đưa ra tính năng kiểm chứng (validation) và để bạn đặc tả trong controller của chính nó những gì làm với đối tượng command mà đã được xác định với giá trị tham số của request.
* MultiActionController: Spring đưa ra một multi-action controller với nó bạn tập hợp nhiều action vào trong một controller, như vậy nhóm các chức năng với nhau. Multi-action controller trong org.springframework.web.servlet.mvc.multiaction – và là khả năng ánh xạ các request đến tên phuong thức và gọi đúng tên phương thức. Sử dụng controller multi-action thì đặc biệt thuận tiện khi bạn có nhiều chức năng phổ biến trong một controller, nhưng muốn có nhiều mục (multiple entry) trỏ vào controller,
* UrlFilenameViewController: UrlFilenameViewController kiểm tra URL và lấy tên file của request và sử dụng nó như một tên view. Ví dụ, tên file của http://www.springframework.org/index.html request là index.

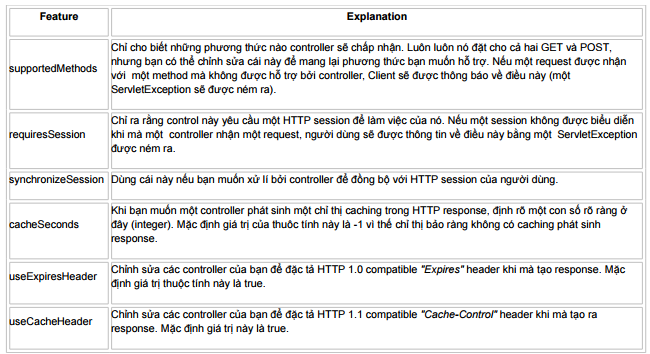
Để tạo một class controller thì ta phải extends một class hoặc implements một interface phía trên.



Ta vẫn còn một cách để tạo controllers đó là sử dụng annotation, cách được dùng phổ biến hơn với phiên bản Spring hiện tại.

* 1. **Architecture - Controllers Responsibilities**

Để cung cấp một cơ sở hạ tâng, tất cả những Controller khác nhau của Spring kế thừa từ AbstractController.



* 1. **Architecture - Controllers Annotations**

Từ phiên bản 2.5 Spring đã giới thiệu và đưa mô hình lập trình cho các MVC controllers dựa trên các annotations như: @Controller, @RequestMapping, @RequestParam, @ModelAttribute,...làm cho việc tạo một controller trở nên đơn giản. Khi tạo một controller thì ta không phải extends một lớp hoặc implements một interface nào.



@controller annotation dùng để định nghĩa lớp controller

@ ResquestMapping annotation để định phương thức, với thuộc tinh value định nghĩa requesturl và method định nghĩa HTTP request method.

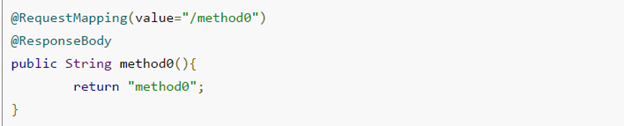
Ở bản 4.3 spring đã giới thiệu @RequestMapping annotation và các loại của nó dùng để định nghĩa các loại của HTTP request method như: @GetMapping, @PutMapping, @PostMapping, @DeleteMapping, @PatchMapping.

* RequestMapping với Class



Bây giờ “/home” sẽ là đường dẫn đến controller của HomeController.

* RequestMapping với hàm



Bây giời đường dẫn sẽ là tên đường dẫn của controller + “/method0”

* RequestMapping với nhiều lựa chọn đường dẫn



Bây giời đường dẫn sẽ là tên đường dẫn của controller + “/method1” hoặc “/method1/second”

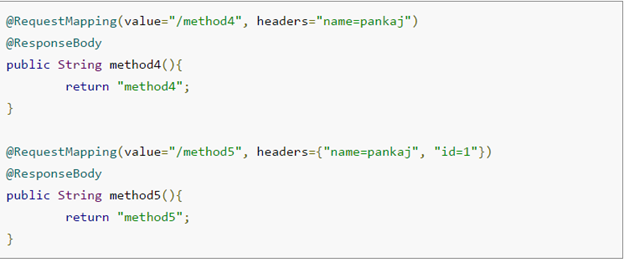
* RequestMapping với HTTP Method



* RequestMapping vơi @pathvariable



* RequestMapping với Header



* RequestMapping với Produces và consumes

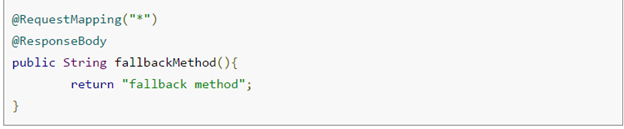




* RequestMapping với @RequestParam : sử dụng requestmapping này khi ta cần lấy tham từ từ request



* RequestMapping với fallbackMethod : Đảm bảo rằng nếu đường dẫn không đúng thay vì trả về một trang web HTTP 404 thì chúng ta sẽ tra về một trang web thông báo lỗi do chúng ta xây dựng.



* 1. **Architecture - Parameter Passing**

**@RequestParam** được dùng để liên kết một tham số request với một tham số method trong controller. Sử dụng @RequestParam: xét url http: cubearticle.com/users/profiles?user=Ashutosh

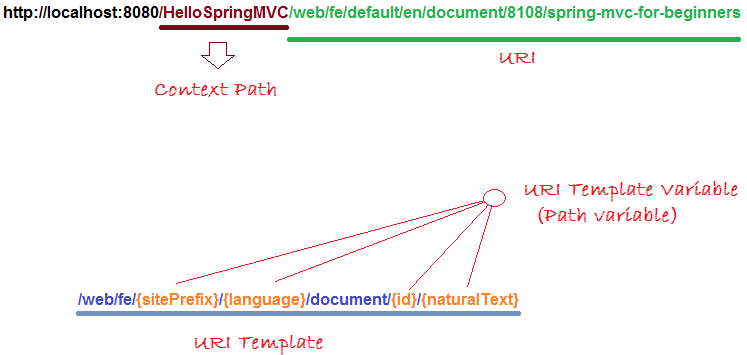


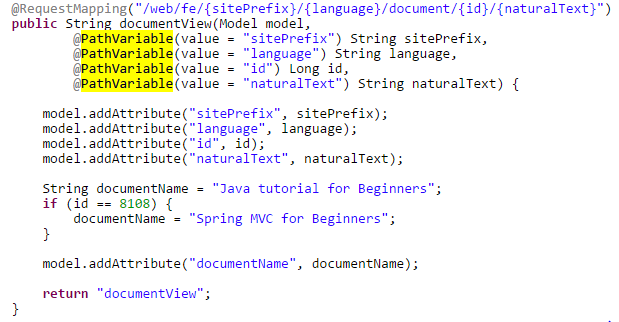
**@pathvariable** được dùng để liên kết một tham số method với một biến template URL (URL template variable). Ví dụ một mẫu URL:

/web/fe/ {sitePrefix}/ {language}/document/ {id}/ {naturalText}

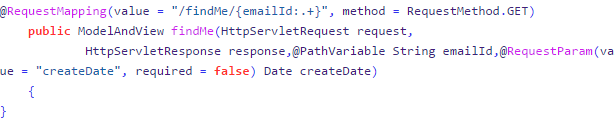
Các URL cụ thể theo mẫu:

/web/fe/default/en/document/8108/spring-mvc-for-beginners



Ví dụ về sử dụng @Pathvariable 

Xét ví dụ để phân biệt @RequestParam và @PathVariable

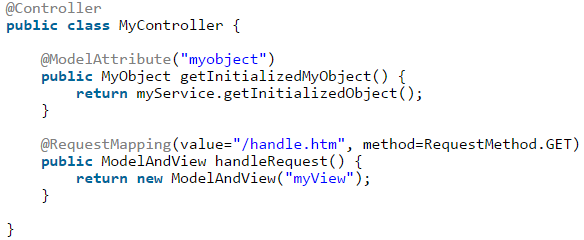
url: localhost:8080/app/findMe/**user@gmail.com**?**createDate**=09-18-2015

Ở đây, email là Pathvariable, còn createDate là RequestParam.

* 1. **Architecture – Thuộc tính Model**

**@ModelAttribute** có thể được dùng như một tham số truyền vào method hay ở ví trí trước method. Mục tiêu chính của annotation này là để liên kết (bind) những request parameters hoặc form với một model object. @ModelAttribute chỉ làm việc khi nó nằm trong class mà class đó là Controller.

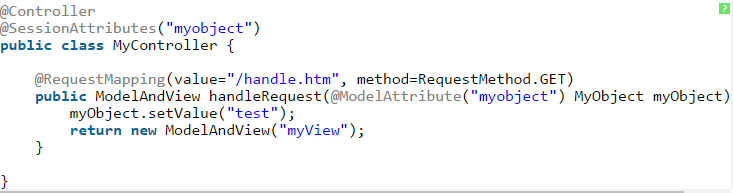
Xét các ví dụ để hiểu hơn về @ModelAttribute.



Ở ví dụ này, Giá trị trả về của method getInitializedMyObject() sẽ được thêm vào Model. View sẽ có thể nhìn thấy đối tượng này thông qua từ khóa ‘myobject’ from request attribute.



Ở đây, method getInitializeMyObject() sẽ được thực thi đầu tiên và kết quả của method này sẽ được lưu trữ trên một Map tạm. Giá trị trả về này sẽ được truyền như một tham số vào phương thức xử lý (handler method). Và cuối cùng đối tượng myObject sẽ được thêm vào model để view có thể nhìn thấy nó.



Trong trường hợp này, Spring tìm ‘myobject’ trong session truyền nó như một tham số vào phương thức xử lý (handler method). Nếu ‘myobject’ không được tìm thấy trong session, thì HttpSessionRequiredException sẽ được đưa ra.

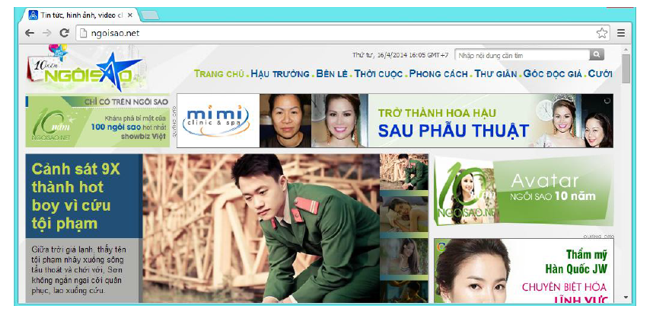


Trong ví dụ trên, một thực thể của MyObject được tạo ra và truyền vào phương thức xử lý (handler method). Nếu MyObject là interface hay abstract class thì BeanInstantiationException sẽ được đề xuất.

* 1. **View – Giới thiệu**

View là nơi hiển thị nội dung của trang web cho phía người dùng. View được viết dựa theo ngôn ngữ HTML.

* View của người dùng:

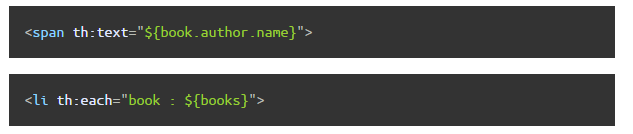


* View của bên server được viết bằng ngôn ngữ HTML:



* 1. **View – Các loại View**

Thymeleaf : là một kiểu view rất linh hoạt . Với view này bạn có thể tự mình định nghĩa các thuộc tính cho các thẻ tag

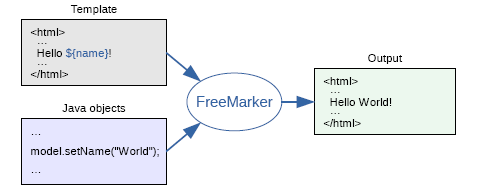


Groovy markup template: là một template dùng để tạo ra văn bản tự động. Thông qua các hàm có sẵn được hô trợ bởi spring để tao ra cai file xml như (XML, XHTML, HTML5…)

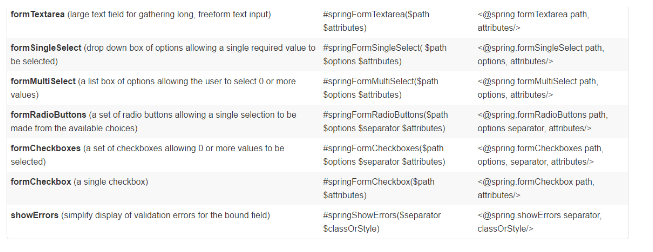




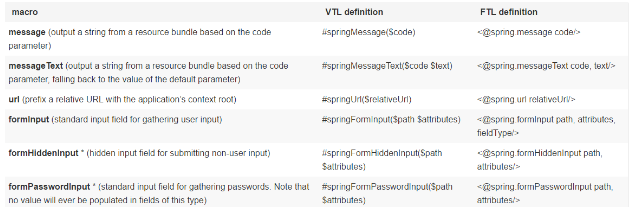
Velocity & FreeMarker: là một mẫu view dựa vào một template có sẵn và một java object để tạo ra một trang HTML











JSP và JSTL: JSP là trang web bổ sung các đặc điểm kỹ thuật của servlet và rất hữu ích trong việc phát triển giao diện web. JSP là công nghệ kết hợp ngôn ngữ HTML/XML và các yếu tố ngôn ngữ của Java để trả về nội dung động của cho người dùng. JSTL thì hổ trợ cho JSP.



* 1. **View - View convention**

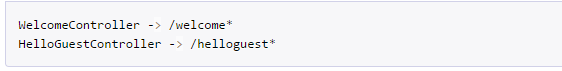
ControllerClassName HandlerMapping: Trong Spring MVC, ControllerClassName HandlerMapping sẽ dùng quy ước để ánh xạ URL từ người dùng tới Controller. Tên đường dẫn sẽ là tên class controller bỏ đi chữ “Controller” và biến thành chữ thường cộng với “/” trước tên đường dẫn.

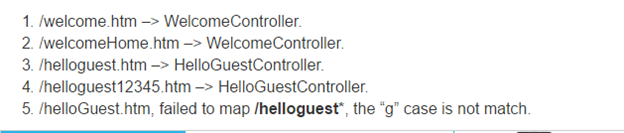
Mặc định, Spring MVC sẽ dùng BeanNameURLHandlerMapping



Muốn sữ dụng ControllerClassName HandlerMapping thì ta cần khai báo trong file cấu hình







* Case Sensitive





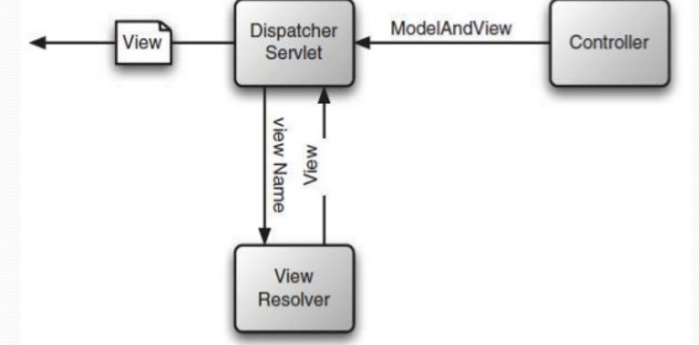
* Path Prefix



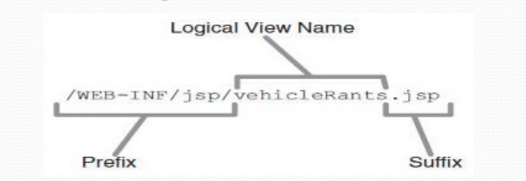
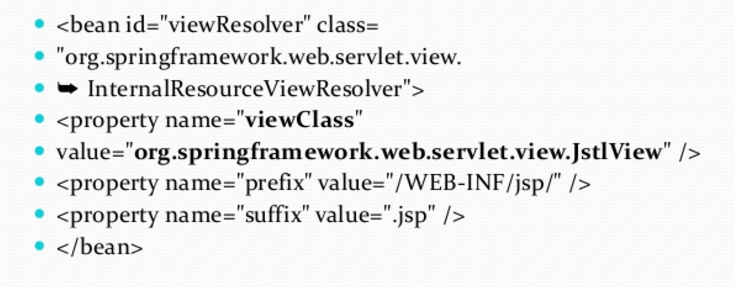


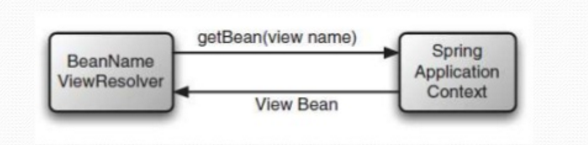
* 1. **View - Resolving a View**

**View resolver** được dùng để phân giải từ view name từ dispatcher servlet và trả lại view cho người dùng . View name là kết quả được trả vê từ đối tượng Model and View do controller gửi tới dispatcher servlet.

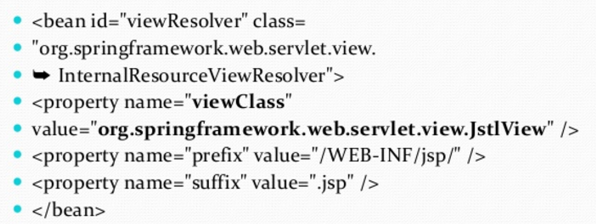


Các loại view resolver trong Spring web:

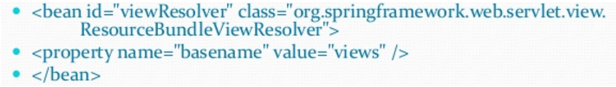
* Internal Resource view resolver: sẽ phân giải từ một view name thành môt view Object với một template cụ thể(ví dụ: JSPs hoặc Velocity templates). Kết quả trả về sẻ có môt tiền tố và hậu tố và view name nhận được.
  + 
* 
* Bean name view resolver: thấy là bộ phân giải sẽ trả về bean name từ một view name mà dispatcher servlet gửi tới. Bean name sẽ được lấy từ application context.



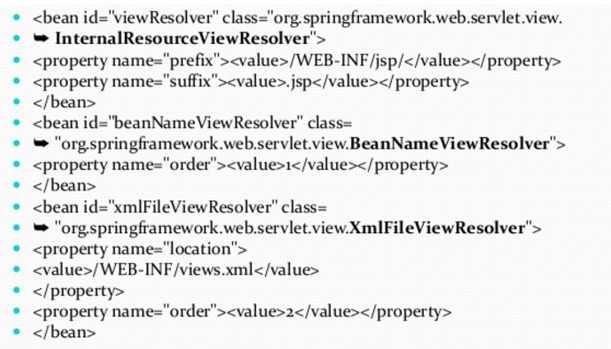
* XML file view resolver: bộ phân giải này cũng giống như bộ phân giải bean name nhưng sẽ trả về môt file xml



* Resource Bundle View Resolver : tra về view được lấy từ file resource bundle (ví dụ : a properties file)



* Ban đầu Internal resolver sẽ là mặc định nhưng với những lựa chọn View implementations (ví dụ: RSS, PDF, Image, ect…) thì Bean name resolver và XML resolver và Resource bundle sẽ được dùng tới.
* Khi mi cùng một view name, thì view name đó sẽ được thiết lập à có nhiều hơn view resovler được sử dụng với từng resolver. Và internal resolver sẽ là thực hiện sau cùng



* 1. **View - JSP & JSTL**

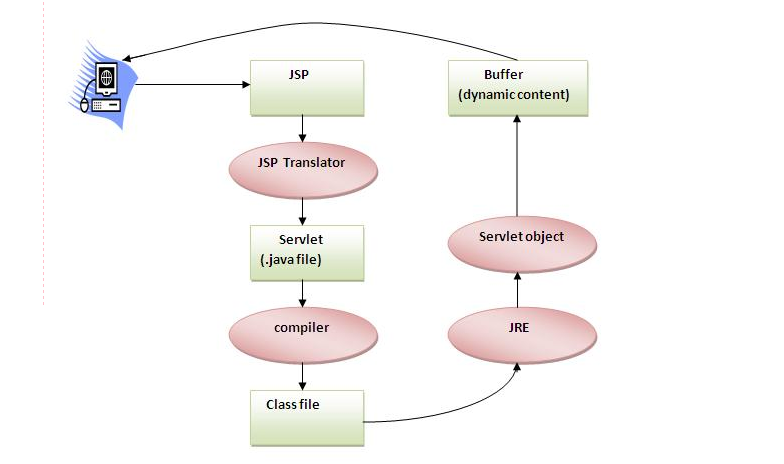
Giới thiệu: JSP (Java Server Page) là trang web bổ sung các đặc điểm kỹ thuật của servlet và rất hữu ích trong việc phát triển giao diện web.

JSP là công nghệ kết hợp ngôn ngữ HTML/XML và các yếu tố ngôn ngữ của Java để trả về nội dung động của cho người dùng.

Trang JSP là trang chưa code Java kết hợp với HTML

JSP định nghĩa một số các thẻ tag để tạo ra web page gọi là JSP tag

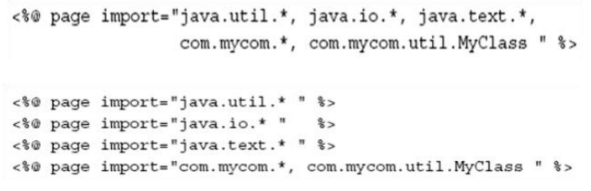
Vòng đời của JSP

* 

Các loại tag cua JSP:

* Directive: cung cấp thông tin tổng quát về các trang JSP cho JSP engine. Có 3 loại của directive:
* page: thông báo engine về tính chất tổng thể của một trang JSP
* Include: chỉ thị cho JSP engine để đưa vào nội dung của một trang khác





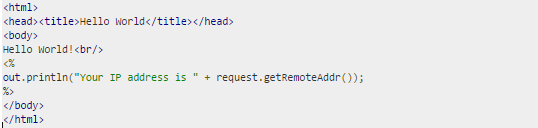
* Taglib: được sử dụng để kết hợp một tiền tố với một thẻ library.
* Luôn bắt đầu <%@ page/include/taglib…%>
* Ghi chú:
* Tên thẻ, thuộc tính, và giá trị phân biệt chữ hoa và chữ thường
* Giá trị phải được đặt trong một cặp dấu nháy đơn hoặc kép
* Không có khoảng trắng giữa dấu bằng(=) và giá trị



* Declaration: khai báo và định nghĩa cho method và biến có thể được sử dụng cho trang JSP. Cấu trúc: <%! // code here %>



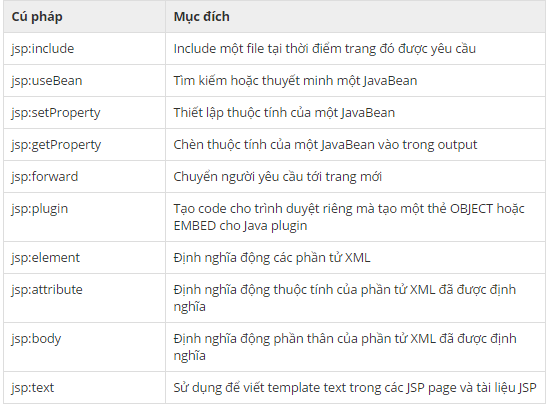
* Scriplet: là đoạn mã được nhúng vào trang JSP. Được thực thi mỗi khi trang web được truy cập. Cú pháp: <% // code here %>



* Expression: hoạt động như các placeholder cho các biểu thức ngôn ngữ Java. Cú pháp: <%= // code here %>



* Action: Action trong JSP sử dụng các chỉ lệnh trong cú pháp XML để điều khiển hành vi của Servlet engine. Bạn có thể chèn động một file, tái sử dụng các thành phần JavaBeans, chuyển người sử dụng tới trang khác, hoặc tạo HTML cho Java Plugin.

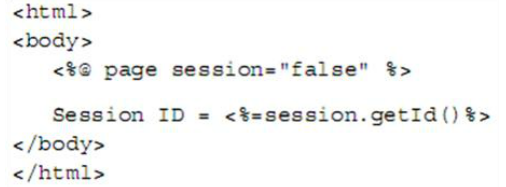




* Các thuộc tinh ẩn của JSP:



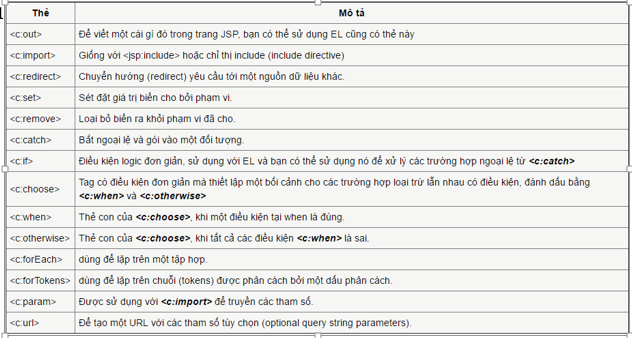
* Session: biến thuộc loại HttpSession, được khai báo khi thuộc tính session của derective page là true (mặc định).



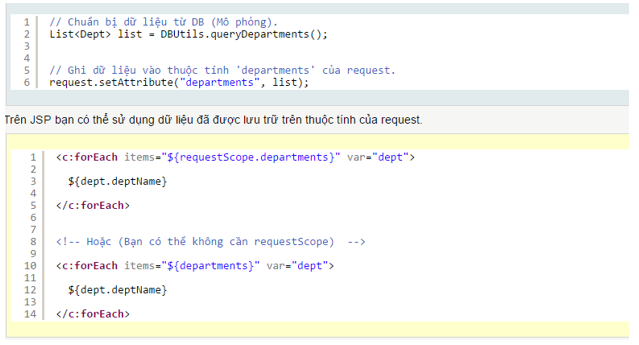
* Request & response : 2 biến tiềm ẩn thuộc HttpServletRequest và HttpServletResponse, Sử dụng hoàn toàn giống như servlet



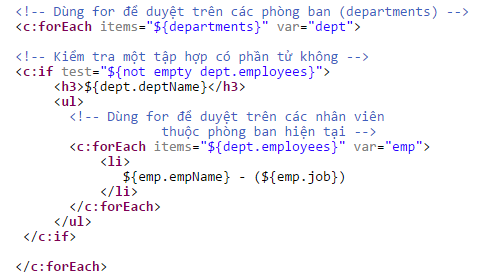
* JSTL (JSP Standard Tag Library): một tập hợp các thẻ JSP hữu ích gói gọn chức năng cốt lổi chung cho nhiều ứng dụng JSP . JSTL hỗ trợ các thao tác phổ biến, cấu trúc như: lặp và điều kiện, thẻ cho các thao tác trên xml, thẻ quốc tế, thẻ SQL.
* JSTL (Core tags):



<c:forEach>:

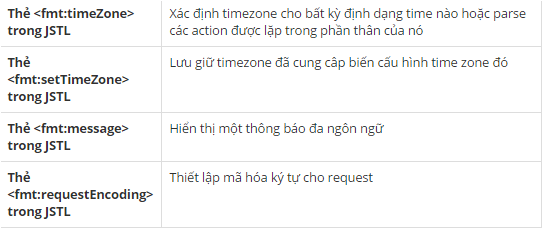


<c:if>

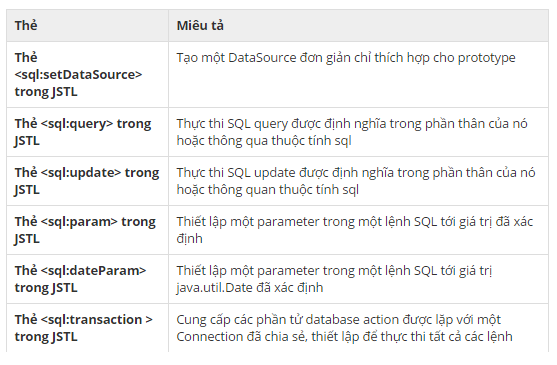


* Formatting Tags trong JSTL : được sử dụng để định dạng và hiển thị text, date, time và ngôn ngữ trong website

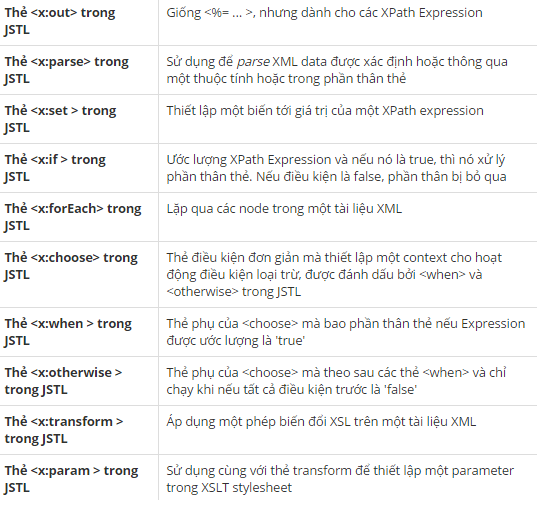




* SQL tag trong JSTL: cung cấp các thẻ tương tác với các loại database như Oracle, MySQL, SQL...



* XML Tag trong JSTL:Nhóm XML Tags trong JSTL cung cấp một cách để tạo và thao tác các tài liệu XML



* JSTL Functions trong JSTL:JSTL bao gồm một số hàm chuẩn, mà hầu hết là các hàm thao tác chuỗi phổ biến





# Tài liệu tham khảo

* <https://www.tutorialspoint.com/spring/spring_overview.html>
* <http://docs.spring.io/spring/docs/current/spring-framework-reference/html/overview.html>
* <http://docs.spring.io/spring/docs/4.2.x/spring-framework-reference/html/dao.html2>
* <http://www.dineshonjava.com/2012/12/dao-support-classes-in-spring-chapter-35.html#.V-SNsCiLTIU>
* <http://docs.spring.io/spring/docs/current/spring-framework-reference/html/jdbc.html>
* <https://en.wikipedia.org/wiki>
* <http://www.javatpoint.com/jsp-tutorial>
* <http://docs.spring.io/spring/docs/current/spring-framework-reference/html/view.html>
* <http://docs.spring.io/spring-framework/docs/2.5.x/api/org/springframework/web/servlet/mvc/support/ControllerClassNameHandlerMapping.html>
* <https://www.mkyong.com/spring-mvc/spring-mvc-controllerclassnamehandlermapping-example/>