\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

***Spring***

*Spring là framework phát triển ứng dụng phổ biến nhất cho doanh nghiệp. Hàng triệu nhà phát triển trên thế giới sử dụng Spring framework để tạo code có hiệu suất cao, dễ dàng kiểm tra và tái sử dụng.*

*Spring framework là nền tảng Java mã nguồn mở. Nó được viết bởi Rod Johnson và được phát hành lần đầu dưới giấy phép Apache 2.0 vào tháng 6 2003.*

*Spring là nhẹ và phiên bản cơ bản của Spring framework chỉ khoảng 2MB.*

*Các tính năng core của Spring Framework có thể sử dụng để phát triển bất kì ứng dụng Java nào, và nhiều extentions để phát triển web app*

***Lợi ích khi sử dụng Spring Framework***

* **POJO BASED:** Spring cho phép nhà phát triển ứng dụng cấp doanh nghiệp sử dụng POJOs. Lợi ích chỉ sử dụng POJOs là bạn không cần đến các sản phẩm EJB container như một máy chủ ứng dụng nhưng bạn có lựa chọn sử dụng các container mạnh mẽ như Tomcat hoặc vài sản phẩm khác.
* **MODULAR:** Spring tổ chức theo kiểu module. Măc dù số class và package là đáng kể, bạn chỉ cần quan tâm đến module bạn cần mà bỏ qua những cái khác.
* **INTEGRATION WITH EXISTING FRAMEWORKS -** Spring không phát minh ra công nghệ, thực tế nó sử dụng nhiều công nghệ có sẵn như ORM frameworks, logging frameworks, JEE, Quartz và JDK timers và nhiều công nghệ khác.
* **TESTABLITY:** Test app được viết bởi Spring khá đơn giản vì tính độc lập của code được áp dụng với framework này. Hơn nữa, bằng việc sử dụng JavaBeanstyel POJOs, code trở nên dễ để sử dụng dependency injection để injecting test dữ liệu.
* **WEB MVC:** Web framework của Spring là MVC framework được thiết kế tốt, cung cấp sự thay thế các framework web như Struts và 1 vài framework ít được phổ biến hơn.
* **CENTRAL EXCEPTION HANDLING**Spring cung cấp API tiện lợi để dich các ngoại lệ chi tiết về công nghệ (thrown by JDBC, Hibernate, or JDO, for example) thành các ngoại lệ nhất quán, không được kiểm tra
* **LIGHTWEIGHT:** Các container IoC nhẹ có xu hướng nhẹ, đặc biệt là khi so sánh với container EJB chẳng hạn. Điều này có lợi cho việc phát triển và triển khai các ứng dụng trên máy tính có tài nguyên CPU và bộ nhớ hạn chế.
* **TRANSACTION MANAGEMENT:** Spring cung cấp giao diện quản lí giao dịch nhất quán mà có thể thu nhỏ thành giao dịch có quy mô cục bộ (vd: sử dụng 1 db) và mở rộng quy mô giao dịch toàn cầu (vd: sử dụng JTA)

## *Audience*

Hướng dẫn này được thiết kế cho các lập trình viên Java có nhu cầu hiểu chi tiết về khung công tác Spring cùng với kiến ​​trúc và cách sử dụng thực tế của nó. Hướng dẫn này sẽ đưa bạn đến một trình độ chuyên môn trung cấp, từ đó bạn có thể đưa mình đến trình độ chuyên môn cao hơn.

## *Prerequisites*

Trước khi tiếp tục với hướng dẫn này, bạn nên hiểu rõ về ngôn ngữ lập trình Java. Một sự hiểu biết cơ bản về IDE Eclipse cũng được yêu cầu vì tất cả các ví dụ đã được biên dịch bằng IDE Eclipse.

***Mục lục***

*Architecture \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_4*

*Hello World Example \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_7*

*IOC Containers \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_10*

*Bean Definitation \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_12*

*Bean Scope \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_15*

*Bean Life Circle \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_19*

*Bean Post Processors \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 23*

*Bean Definition Inheritance\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 26*

*Dependency Injection\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_30*

*Injecting Inner Bean \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 33*

*Injecting Collection \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_53*

*Bean Autowiring \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_40*

*Annotation Bassed Configuration \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 42*

*Java Based Configuration \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 46*

*Event Handling \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 51*

*AOP with Spring Framework \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_55*

*Web MVC Framework \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_64*

*Logging with Log4J \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_70*

***Spirng Architecture***

Spring có khả năng cho tất cả các ứng dụng doanh nghiệp của bạn. Tuy nhiên, Spring là mô-đun, cho phép bạn tìm và chọn mô-đun nào phù hợp với bạn mà không cần phải mang theo phần còn lại. Phần sau đây cung cấp chi tiết về tất cả các mô-đun có sẵn trong Spring Framework.

Spring Framework cung cấp khoảng 20 mô-đun có thể được sử dụng dựa trên yêu cầu ứng dụng.



**Core Container**

Core Container bao gồm các mô-đun Core, Bean, Context và Expression Language, có chi tiết như sau

* Các **core** mô-đun cung cấp những phần cơ bản của framework này, bao gồm các tính năng IoC và Dependency injection.
* Các **Bean** mô-đun cung cấp BeanFactory là một thực hiện phức tạp của model factory.
* Các **Context** mô-đun được xây dựng trên nền tảng vững chắc được cung cấp bởi các module Core và Bean và nó là một phương tiện để truy cập bất kỳ đối tượng được xác định và cấu hình. ApplicationContext interface là tiêu điểm của mô-đun Context.
* Các **SpEL** mô-đun cung cấp một ngôn ngữ biểu hiện mạnh mẽ cho các truy vấn và thao tác một đồ thị đối tượng khi chạy.

## *Data Access/Integration*

Lớp Tích hợp / Truy cập dữ liệu bao gồm các mô đun JDBC, ORM, OXM, JMS và Transaction có chi tiết như sau

* Các **JDBC** mô-đun cung cấp một lớp JDBC-trừu tượng mà loại bỏ sự cần thiết của JDBC tẻ nhạt
* Các **ORM** mô-đun cung cấp các lớp tích hợp cho các API ánh xạ đối tượng-quan hệ phổ biến, bao gồm JPA, JDO, Hibernate, và iBatis.
* Các **OXM** mô-đun cung cấp một lớp trừu tượng mà hỗ trợ Object triển khai lập bản đồ XML / cho JAXB, Castor, XMLBeans, JiBX và XStream.
* Mô-đun **JMS của** Dịch vụ nhắn tin Java chứa các tính năng để tạo và tiêu thụ thư.
* Các **Transaction** mô-đun hỗ trợ quản lý giao dịch theo chương trình và khai báo cho các lớp mà thực hiện các giao diện đặc biệt và cho tất cả các POJO của bạn.

***Web***

Lớp Web bao gồm các mô đun Web, Web-MVC, Web-Socket và Web-Portlet, các chi tiết như sau -

* Các **Web** mô-đun cung cấp tích hợp web theo định hướng cơ bản tính năng như chức năng tập tin tải lên nhiều phần dữ liệu và khởi tạo các container IoC sử dụng servlet listener và một App Context web theo định hướng.
* Mô **-** đun **Web-MVC** chứa triển khai Model-View-Controller (MVC) của Spring cho các ứng dụng web.
* Mô **-** đun **Web-Socket** cung cấp hỗ trợ cho giao tiếp hai chiều dựa trên WebSocket giữa máy khách và máy chủ trong các ứng dụng web.
* Mô **-** đun **Web-Portlet** cung cấp triển khai MVC được sử dụng trong môi trường portlet và phản ánh chức năng của mô-đun Web-Servlet.

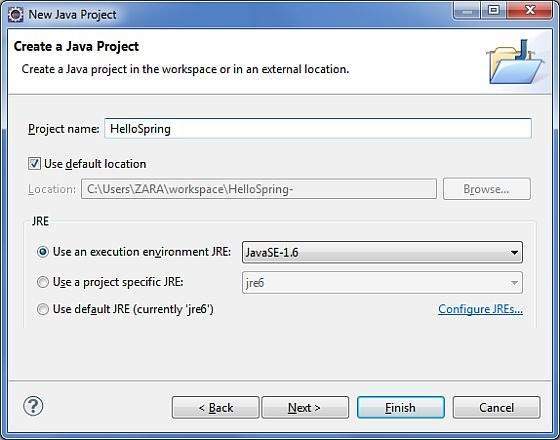
## *Miscellaneous*

Có một vài mô-đun quan trọng khác như các mô-đun AOP, Aspects, Instrumentation, Web và Test có chi tiết như sau -

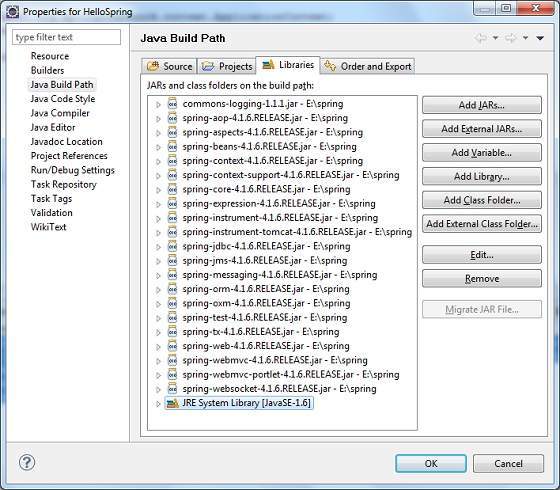
* Các **AOP** module cung cấp một thực hiện lập trình hướng khía cạnh cho phép bạn xác định method-interceptors và pointcuts để các chức năng được phân chia rõ ràng theo mong muốn.
* Các **Aspect** mô-đun cung cấp tích hợp với AspectJ, mà lại là một framework AOP mạnh mẽ và trưởng thành.
* Các **Instrumentation** mô-đun cung cấp hỗ trợ thiết bị đo đạc và lớp loader triển khai lớp sẽ được sử dụng trong các máy chủ ứng dụng nhất định.
* Các **Messaging** mô-đun cung cấp hỗ trợ cho STOMP như WebSocket tiểu giao thức để sử dụng trong các ứng dụng. Nó cũng hỗ trợ một mô hình lập trình chú thích để định tuyến và xử lý các thông báo STOMP từ các máy khách WebSocket.
* Các **Test** mô-đun hỗ trợ việc kiểm tra các thành phần Spring với JUnit hoặc TestNG khuôn khổ.

***HelloWorld Example***

***Bước 1: Tạo dự án Java và thêm các thư viện cần thiết***

******

Thêm các thư viện cần thiết và Configure Build Path:



***Bước 2: Tạo các file java:***

Tạo 2 file **HelloWorld.java** and **MainApp.java**

File HelloWorld.java

package com.tutorialspoint;

public class HelloWorld {

private String message;

public void setMessage(String message){

this.message = message;

}

public void getMessage(){

System.out.println("Your Message : " + message);

}

}

File MainApp.java

package com.tutorialspoint;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

HelloWorld obj = (HelloWorld) context.getBean("helloWorld");

obj.getMessage();

}

}

1. Bước 1, tạo application context và sử dụng framework API **ClassPathXmlApplicationContext().** API này load các Bean từ Config file và dựa trên API cung cấp, nó đảm nhiệm nhiệm vụ tạo và khởi chạy các obj-bean trong file cấu hình
2. Bước 2, lấy Bean cần thiết bằng cách sử dụng method **getBean()** và tạo bối cảnh. Phương thức này sử dụng bean ID trả về một obj chung, mà cuối cùng có thể chuyển thành đối tượng thực tế. Một khi bạn có đối tượng, bạn có thể sử dụng đối tượng này để gọi các method của nó.

***Bước 3: Tạo file Bean Configuration:***

Tạo file xml đặt dưới đường dẫn src như sau:

<?xml version = "1.0" encoding = "UTF-8"?>

<beans xmlns = "http://www.springframework.org/schema/beans"

xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation = "http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id = "helloWorld" class = "com.tutorialspoint.HelloWorld">

<property name = "message" value = "Hello World!"/>

</bean>

</beans>

Khi ứng dụng Spring được tải vào bộ nhớ, Framework sẽ sử dụng tệp cấu hình ở trên để tạo tất cả các bean được xác định và gán cho chúng một ID duy nhất như được định nghĩa trong **<bean>**. Bạn có thể sử dụng **thẻ <property>** để chuyển các giá trị của các biến khác nhau được sử dụng tại thời điểm tạo đối tượng.

Kết quả:

Your Message : Hello World!

Bạn có thể thấy tính linh hoạt của ứng dụng Spring ở trên bằng cách thay đổi giá trị của thuộc tính "message" và giữ cả hai tệp nguồn không thay đổi.

***IOC Container***

**IoC Container trong Spring** chính là lõi của Spring Framework. IoC Container sẽ tạo ra các đối tượng, nối chúng lại với nhau, cấu hình chúng, và quản lý vòng đời của chúng. IoC Container sử dụng DI ([Dependency Injection](https://viettuts.vn/spring/dependency-injection-trong-spring)) để quản lý các thành phần tạo nên một ứng dụng.



Spring cung cấp 2 loại container khác nhau:

|  |  |
| --- | --- |
| **Sr.Không.** | **Container & Mô tả** |
| 1 | [Spring BeanFactory Container](https://www.tutorialspoint.com/spring/spring_beanfactory_container.htm)  Đây là bộ chứa đơn giản nhất cung cấp hỗ trợ cơ bản cho DI và được xác định bởi giao diện *org.springframework.beans.factory.BeanFactory* . BeanFactory và các giao diện liên quan, chẳng hạn như BeanFactoryAware, InitializingBean, DisposableBean, vẫn hiện diện trong Spring với mục đích tương thích ngược với một số lượng lớn các khung công tác của bên thứ ba tích hợp với Spring.  <https://www.tutorialspoint.com/spring/spring_beanfactory_container.htm> |
| 2 | [Spring ApplicationContext Container](https://www.tutorialspoint.com/spring/spring_applicationcontext_container.htm)  Container này bổ sung thêm chức năng dành riêng cho doanh nghiệp như khả năng phân giải tin nhắn văn bản từ tệp thuộc tính và khả năng xuất bản các sự kiện ứng dụng cho người nghe sự kiện quan tâm. Container này được xác định bởi giao diện *org.springframework.context.ApplicationContext* .  <https://www.tutorialspoint.com/spring/spring_applicationcontext_container.htm> |

*ApplicationContext* bao gồm tất cả các chức năng của *BeanFactory* , do đó, nó thường được khuyên dùng hơn *BeanFactory* . BeanFactory vẫn có thể được sử dụng cho các ứng dụng nhẹ như thiết bị di động hoặc ứng dụng dựa trên applet trong đó dung lượng và tốc độ dữ liệu là đáng kể.

***Bean Definition***

Các đối tượng là cốt lõi của ứng dụng của bạn và được quản lý bởi bộ chứa Spring IoC được gọi là các **bean** . Bean là một đối tượng được khởi tạo, lắp ráp và được quản lý bởi bộ chứa Spring IoC. Những bean này được tạo với siêu dữ liệu cấu hình mà bạn cung cấp cho container. Ví dụ, ở dạng định nghĩa <bean /> XML mà bạn đã thấy trong các chương trước.

Bean định nghĩa chứa thông tin được gọi là **siêu dữ liệu cấu hình(metadata configuration)** , cần thiết cho container để biết những điều sau đây

* Cách tạo Bean
* Vòng đời của Bean
* Phụ thuộc của Bean

Dưới đây là tập thuộc tính mô tả các Bean:

|  |  |
| --- | --- |
| **Sr.No.** | **Properties & Description** |
| 1 | **class**  This attribute is mandatory and specifies the bean class to be used to create the bean. |
| 2 | **name**  This attribute specifies the bean identifier uniquely. In XMLbased configuration metadata, you use the id and/or name attributes to specify the bean identifier(s). |
| 3 | **scope**  This attribute specifies the scope of the objects created from a particular bean definition and it will be discussed in bean scopes chapter. |
| 4 | **constructor-arg**  This is used to inject the dependencies and will be discussed in subsequent chapters. |
| 5 | **properties**  This is used to inject the dependencies and will be discussed in subsequent chapters. |
| 6 | **autowiring mode**  This is used to inject the dependencies and will be discussed in subsequent chapters. |
| 7 | **lazy-initialization mode**  A lazy-initialized bean tells the IoC container to create a bean instance when it is first requested, rather than at the startup. |
| 8 | **initialization method**  A callback to be called just after all necessary properties on the bean have been set by the container. It will be discussed in bean life cycle chapter. |
| 9 | **destruction method**  A callback to be used when the container containing the bean is destroyed. It will be discussed in bean life cycle chapter. |

***Spring Configuration Metadata***

Có ba phương pháp quan trọng để cung cấp siêu dữ liệu cấu hình cho Spring Container:

* XML based configuration file.
* Annotation-based configuration(ở chương sau)
* Java-based configuration(ở chương sau)

Bạn đã thấy cách siêu dữ liệu cấu hình dựa trên XML được cung cấp cho container, nhưng chúng ta hãy xem một tệp cấu hình dựa trên XML khác với các định nghĩa bean khác nhau bao gồm *lazy-init*, *init-method* và *destroy-method*

<?xml version = "1.0" encoding = "UTF-8"?>

<beans xmlns = "http://www.springframework.org/schema/beans"

xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation = "http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<!-- A simple bean definition -->

<bean id = "..." class = "...">

<!-- collaborators and configuration for this bean go here -->

</bean>

<!-- A bean definition with lazy init set on -->

<bean id = "..." class = "..." lazy-init = "true">

<!-- collaborators and configuration for this bean go here -->

</bean>

<!-- A bean definition with initialization method -->

<bean id = "..." class = "..." init-method = "...">

<!-- collaborators and configuration for this bean go here -->

</bean>

<!-- A bean definition with destruction method -->

<bean id = "..." class = "..." destroy-method = "...">

<!-- collaborators and configuration for this bean go here -->

</bean>

<!-- more bean definitions go here -->

</beans>

***Spring Bean Scope***

Khi xác định <bean>, bạn có tùy chọn khai báo một phạm vi cho bean đó. Ví dụ, để buộc Spring tạo ra một thể hiện bean mới mỗi lần cần, bạn nên khai báo thuộc tính scope của bean là **prototype**. Tương tự, nếu bạn muốn Spring trả về cùng một thể hiện bean mỗi lần cần, bạn nên khai báo thuộc tính scope của bean là **singleton** .

Spring Framework hỗ trợ năm phạm vi sau, ba phạm vi chỉ khả dụng nếu bạn sử dụng ApplicationContext nhận biết web:

|  |  |
| --- | --- |
| **r.No.** | **Scope & Description** |
| 1 | **singleton**  This scopes the bean definition to a single instance per Spring IoC container (default). |
| 2 | **prototype**  This scopes a single bean definition to have any number of object instances. |
| 3 | **request**  This scopes a bean definition to an HTTP request. Only valid in the context of a web-aware Spring ApplicationContext. |
| 4 | **session**  This scopes a bean definition to an HTTP session. Only valid in the context of a web-aware Spring ApplicationContext. |
| 5 | **global-session**  This scopes a bean definition to a global HTTP session. Only valid in the context of a web-aware Spring ApplicationContext. |

## *The singleton scope*

Nếu một phạm vi được đặt thành **singleton**, bộ chứa Spring IoC sẽ tạo chính xác một thể hiện của đối tượng được xác định bởi định nghĩa bean đó. Trường hợp duy nhất này được lưu trữ trong bộ đệm của các hạt đơn lẻ như vậy và tất cả các yêu cầu và tham chiếu tiếp theo cho hạt đậu có tên đó trả về đối tượng được lưu trữ.

## Phạm vi mặc định luôn là singleton. Nhưng để rõ ràng bạn có thể cấu hình như sau:

<!-- A bean definition with singleton scope -->

<bean id = "..." class = "..." scope = "singleton">

<!-- collaborators and configuration for this bean go here -->

</bean>

## *Ví dụ:*

## *Nội dung tệp Helloworl.java*

package com.tutorialspoint;

public class HelloWorld {

private String message;

public void setMessage(String message){

this.message = message;

}

public void getMessage(){

System.out.println("Your Message : " + message);

}

}

## *Nội dung tệp MainApp.java*

package com.tutorialspoint;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

HelloWorld objA = (HelloWorld) context.getBean("helloWorld");

objA.setMessage("I'm object A");

objA.getMessage();

HelloWorld objB = (HelloWorld) context.getBean("helloWorld");

objB.getMessage();

}

## *Nội dung tệp Beans.xml*

<?xml version = "1.0" encoding = "UTF-8"?>

<beans xmlns = "http://www.springframework.org/schema/beans"

xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation = "http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id = "helloWorld" class = "com.tutorialspoint.HelloWorld" scope = "singleton">

</bean>

</beans>

***Kết quả:***

Your Message : I'm object A

Your Message : I'm object A

***The Prototype scope***

Nếu phạm vi được đặt thành prototype, Spring IoC sẽ tạo một phiên bản bean mới của đối tượng mỗi khi yêu cầu cho bean cụ thể đó được thực hiện. Theo quy định, sử dụng phạm vi prototype cho tất cả các loại bean full-state và phạm vi singleton cho các loại beans stateless.

<!-- A bean definition with prototype scope -->

<bean id = "..." class = "..." scope = "prototype">

<!-- collaborators and configuration for this bean go here -->

</bean>

## *Ví dụ:*

## *Tương tự như trên nhưng ta đổi 1 chút ở file Beans.xml*

<?xml version = "1.0" encoding = "UTF-8"?>

<beans xmlns = "http://www.springframework.org/schema/beans"

xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation = "http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id = "helloWorld" class = "com.tutorialspoint.HelloWorld" scope = "prototype">

</bean>

</beans>

## *Kết quả:*

Your Message : I'm object A

Your Message : null

***Spring Bean Life Circle***

Vòng đời của Bean trong Spring bao gồm khởi tạo, sử dụng và kết thúc. Mặc dù, có một danh sách các hoạt động xảy ra đằng sau thời điểm bắt đầu và trước khi bean bị hủy.

Trong phần này, ta chỉ bàn luận về 2 phương thức liên quan đến việc gọi lại (callback) vòng đời của bean được yêu cầu tại thời điểm khởi tạo (init) và hủy (destroy).

## *Initialization callbacks*

Giao diện org.springframework.beans.factory.InitializingBean chỉ định một phương thức duy nhất -

void afterPropertiesSet() throws Exception;

Do đó, bạn có thể thực hiện đơn giản giao diện trên và công việc khởi tạo có thể được thực hiện bên trong phương thức afterProperIESset () như sau -

public class ExampleBean implements InitializingBean {

public void afterPropertiesSet() {

// do some initialization work

}

}

Trong trường hợp siêu dữ liệu cấu hình dựa trên XML, bạn có thể sử dụng thuộc tính **init-method** để chỉ định tên của phương thức có chữ ký không có đối số void. Ví dụ:

<bean id = "exampleBean" class = "examples.ExampleBean" init-method = "init"/>

Sau đây là định nghĩa lớp ExampleBean

public class ExampleBean {

public void init() {

// do some initialization work

}

}

## *Destruction callbacks*

Các *org.springframework.beans.factory.DisposableBean* giao diện xác định một method duy nhất

void destroy() throws Exception;

Do đó, bạn chỉ cần implement giao diện trên và công việc có thể được thực hiện bên trong phương thức hủy () như sau

public class ExampleBean implements DisposableBean {

public void destroy() {

// do some destruction work

}

}

Trong trường hợp siêu dữ liệu cấu hình dựa trên XML, bạn có thể sử dụng thuộc tính **destroy-method** để chỉ định tên của phương thức có chữ ký không có đối số trống. Ví dụ:

<bean id = "exampleBean" class = "examples.ExampleBean" destroy-method = "destroy"/>

Sau đây là định nghĩa lớp -

public class ExampleBean {

public void destroy() {

// do some destruction work

}

}

## Khuyến cáo không nên sử dụng InitializingBean hoặc DisposableBean callbacks, vì XML config cho ta sự linh hoạt hơn trong việc đặt tên method.

## Ví dụ:

## File HelloWorld.java

package com.tutorialspoint;

public class HelloWorld {

private String message;

public void setMessage(String message){

this.message = message;

}

public void getMessage(){

System.out.println("Your Message : " + message);

}

public void init(){

System.out.println("Bean is going through init.");

}

public void destroy() {

System.out.println("Bean will destroy now.");

}

}

## File MainApp.java: bạn cần đăng kí một shutdown hook registerShutdownHook() được khai báo trong lớp AbstractApplicationContext. Điều này khiến context sẽ được đóng và gọi ra các hàm hủy cần thiết

package com.tutorialspoint;

import org.springframework.context.support.AbstractApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

AbstractApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

HelloWorld obj = (HelloWorld) context.getBean("helloWorld");

obj.getMessage();

context.registerShutdownHook();

}

}

## File Beans.xml

<?xml version = "1.0" encoding = "UTF-8"?>

<beans xmlns = "http://www.springframework.org/schema/beans"

xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation = "http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id = "helloWorld" class = "com.tutorialspoint.HelloWorld" init-method = "init"

destroy-method = "destroy">

<property name = "message" value = "Hello World!"/>

</bean>

</beans>

## Kết quả:

Bean is going through init.

Your Message : Hello World!

Bean will destroy now.

## *Default initialization and destroy methods*

## Nếu bạn có quá nhiều bean có hàm tạo-hủy cùng tên, không cần phải khai báo init-method và destroy-method ở mỗi bean. Thay vào đó hãy sử dụng default-init-method và default-destroy-method khai báo dưới thẻ <beans> như dưới đây:

<beans xmlns = "http://www.springframework.org/schema/beans"

xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation = "http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"

default-init-method = "init"

default-destroy-method = "destroy">

<bean id = "..." class = "...">

<!-- collaborators and configuration for this bean go here -->

</bean>

</beans>

***Bean Post Processors***

Các **BeanPostProcessor** interface định nghĩa gọi lại hàm mà bạn có thể thực hiện để cung cấp riêng instantiation logic, dependency-resolution logic, vv Bạn cũng có thể thực hiện một số logic tùy chỉnh sau Spring container kết thúc cài đặt, cấu hình, và khởi tạo một bean bằng cách cắm vào một hoặc nhiều BeanPostProcessor interface.

Bạn có thể định cấu hình nhiều interface BeanPostProcessor và bạn có thể kiểm soát thứ tự các giao diện BeanPostProcessor này thực thi bằng cách đặt thuộc tính **order** được cung cấp mà BeanPostProcessor thực hiện giao diện **Ordered** .

BeanPostProcessors hoạt động trên các bean (hoặc đối tượng), điều đó có nghĩa là bộ chứa Spring IoC khởi tạo một thể hiện bean và sau đó các giao diện BeanPostProcessor thực hiện công việc của chúng.

Một **ApplicationContext** tự động phát hiện bất kỳ đậu được định nghĩa với việc thực hiện các **BeanPostProcessor** và đăng ký những hạt beans như postprocessors, được sau đó được gọi một cách thích hợp bởi các container khi tạo bean.

Ví dụ:

File **HelloWorld.java**

package com.tutorialspoint;

public class HelloWorld {

private String message;

public void setMessage(String message){

this.message = message;

}

public void getMessage(){

System.out.println("Your Message : " + message);

}

public void init(){

System.out.println("Bean is going through init.");

}

public void destroy(){

System.out.println("Bean will destroy now.");

}

}

*Đây là một ví dụ rất cơ bản về việc triển khai BeanPostProcessor, nó in tên bean trước và sau khi khởi tạo bất kỳ bean nào. Bạn có thể triển khai logic phức tạp hơn trước và sau khi xác định một bean vì bạn có quyền truy cập vào đối tượng bean bên trong cả hai phương thức xử lý bài.*

Dưới đây là nội dung của tệp **initHelloWorld.java**

package com.tutorialspoint;

import org.springframework.beans.factory.config.BeanPostProcessor;

import org.springframework.beans.BeansException;

public class InitHelloWorld implements BeanPostProcessor {

public Object postProcessBeforeInitialization(Object bean, String beanName)

throws BeansException {

System.out.println("BeforeInitialization : " + beanName);

return bean; // you can return any other object as well

}

public Object postProcessAfterInitialization(Object bean, String beanName)

throws BeansException {

System.out.println("AfterInitialization : " + beanName);

return bean; // you can return any other object as well

}

}

Nội dung class **MainApp.java**

package com.tutorialspoint;

import org.springframework.context.support.AbstractApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

AbstractApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

HelloWorld obj = (HelloWorld) context.getBean("helloWorld");

obj.getMessage();

context.registerShutdownHook();

}

}

Tệp ***Beans.xml***

<?xml version = "1.0" encoding = "UTF-8"?>

<beans xmlns = "http://www.springframework.org/schema/beans"

xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation = "http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id = "helloWorld" class = "com.tutorialspoint.HelloWorld"

init-method = "init" destroy-method = "destroy">

<property name = "message" value = "Hello World!"/>

</bean>

<bean class = "com.tutorialspoint.InitHelloWorld" />

</beans>

Nếu mọi thứ đều ổn với ứng dụng của bạn, nó sẽ in thông báo sau

BeforeInitialization : helloWorld

Bean is going through init.

AfterInitialization : helloWorld

Your Message : Hello World!

Bean will destroy now.

***Spring Bean Definition Inheritance***

Một định nghĩa bean có thể chứa nhiều thông tin cấu hình, bao gồm các giá trị hàm tạo, thuộc tính và thông tin dành riêng cho container như phương thức khởi tạo, tên phương thức static factory v.v.

Bean con kế thừa dữ liệu cấu hình từ định nghĩa cha. Định nghĩa con có thể ghi đè một số giá trị hoặc thêm các giá trị khác nếu cần.

Kế thừa định nghĩa Spring Bean không liên quan gì đến kế thừa lớp Java nhưng khái niệm thừa kế là như nhau. Bạn có thể định nghĩa một định nghĩa bean cha là một mẫu và các bean con khác có thể kế thừa cấu hình cần thiết từ bean cha bằng thuộc tính **parent**

Ví dụ:

File **Beans.xml**

<?xml version = "1.0" encoding = "UTF-8"?>

<beans xmlns = "http://www.springframework.org/schema/beans"

xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation = "http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id = "helloWorld" class = "com.tutorialspoint.HelloWorld">

<property name = "message1" value = "Hello World!"/>

<property name = "message2" value = "Hello Second World!"/>

</bean>

<bean id ="helloIndia" class = "com.tutorialspoint.HelloIndia" parent = "helloWorld">

<property name = "message1" value = "Hello India!"/>

<property name = "message3" value = "Namaste India!"/>

</bean>

</beans>

File **HelloWorld.java**

package com.tutorialspoint;

public class HelloWorld {

private String message1;

private String message2;

public void setMessage1(String message){

this.message1 = message;

}

public void setMessage2(String message){

this.message2 = message;

}

public void getMessage1(){

System.out.println("World Message1 : " + message1);

}

public void getMessage2(){

System.out.println("World Message2 : " + message2);

}

}

File **HelloIndia.java**

package com.tutorialspoint;

public class HelloIndia {

private String message1;

private String message2;

private String message3;

public void setMessage1(String message){

this.message1 = message;

}

public void setMessage2(String message){

this.message2 = message;

}

public void setMessage3(String message){

this.message3 = message;

}

public void getMessage1(){

System.out.println("India Message1 : " + message1);

}

public void getMessage2(){

System.out.println("India Message2 : " + message2);

}

public void getMessage3()007B

System.out.println("India Message3 : " + message3);

}

}

File **MainApp.java**

package com.tutorialspoint;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

HelloWorld objA = (HelloWorld) context.getBean("helloWorld");

objA.getMessage1();

objA.getMessage2();

HelloIndia objB = (HelloIndia) context.getBean("helloIndia");

objB.getMessage1();

objB.getMessage2();

objB.getMessage3();

}

}

**Kết quả:**

World Message1 : Hello World!

World Message2 : Hello Second World!

India Message1 : Hello India!

India Message2 : Hello Second World!

India Message3 : Namaste India!

## *Bean Definition Template*

Bạn có thể tạo một template định nghĩa Bean, có thể được sử dụng bởi các định nghĩa bean con khác. Trong khi xác định Template định nghĩa Bean, bạn không nên chỉ định thuộc tính **class** và nên chỉ định thuộc tính **abstract** và nên chỉ định thuộc tính trừu tượng có giá trị là **true** như sau:

<?xml version = "1.0" encoding = "UTF-8"?>

<beans xmlns = "http://www.springframework.org/schema/beans"

xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation = "http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id = "beanTeamplate" abstract = "true">

<property name = "message1" value = "Hello World!"/>

<property name = "message2" value = "Hello Second World!"/>

<property name = "message3" value = "Namaste India!"/>

</bean>

<bean id = "helloIndia" class = "com.tutorialspoint.HelloIndia" parent = "beanTeamplate">

<property name = "message1" value = "Hello India!"/>

<property name = "message3" value = "Namaste India!"/>

</bean>

</beans>

***Dependency Injection***

Mỗi ứng dụng dựa trên Java có một vài đối tượng hoạt động cùng nhau để trình bày những gì người dùng cuối xem. Khi viết một ứng dụng Java phức tạp, các lớp ứng dụng phải độc lập nhất có thể với các lớp Java khác để tăng khả năng sử dụng lại các lớp này và kiểm tra tính độc lập với các lớp khác trong unit test/ . Dependency Injection (hay đôi khi được gọi là wiring) giúp gắn các lớp này lại với nhau và đồng thời giữ chúng độc lập.

public class TextEditor {

private SpellChecker spellChecker;

public TextEditor() {

spellChecker = new SpellChecker();

}

}

Những gì chúng tôi đã làm ở đây là, tạo ra sự phụ thuộc giữa TextEditor và SpellChecker. Ngược lại với kịch bản điều khiển, thay vào đó chúng ta sẽ làm như thế này

public class TextEditor {

private SpellChecker spellChecker;

public TextEditor(SpellChecker spellChecker) {

this.spellChecker = spellChecker;

}

}

Ở đây, TextEditor không cần lo lắng về việc triển khai SpellChecker. SpellChecker sẽ được triển khai độc lập và sẽ được cung cấp cho TextEditor tại thời điểm khởi tạo TextEditor. Toàn bộ quy trình này được kiểm soát bởi Spring Framework.

Ở đây, chúng tôi đã loại bỏ toàn bộ quyền kiểm soát khỏi TextEditor và giữ nó ở một nơi khác (ví dụ: tệp cấu hình XML) và phần phụ thuộc (tức là lớp SpellChecker) đang được đưa vào lớp TextEditor thông qua **contructor**. Do đó, luồng kiểm soát đã bị "đảo ngược" bởi Dependency Injection (DI) vì bạn đã ủy thác hiệu quả các phụ thuộc vào một số hệ thống bên ngoài.

Phương thức tiêm phụ thuộc thứ hai là thông qua các **Phương thức Setter** của lớp TextEditor nơi chúng ta sẽ tạo một cá thể SpellChecker.

|  |  |
| --- | --- |
| **Sr.No.** | **Dependency Injection Type & Description** |
| 1 | [Constructor-based dependency injection](https://www.tutorialspoint.com/spring/constructor_based_dependency_injection.htm)  Constructor-based DI is accomplished when the container invokes a class constructor with a number of arguments, each representing a dependency on the other class.  <https://www.tutorialspoint.com/spring/constructor_based_dependency_injection.htm> |
| 2 | [Setter-based dependency injection](https://www.tutorialspoint.com/spring/setter_based_dependency_injection.htm)  Setter-based DI is accomplished by the container calling setter methods on your beans after invoking a no-argument constructor or no-argument static factory method to instantiate your bean.  <https://www.tutorialspoint.com/spring/setter_based_dependency_injection.htm> |

Cú pháp inject qua contructor(ở file Beans.xml)

<?xml version = "1.0" encoding = "UTF-8"?>

<beans xmlns = "http://www.springframework.org/schema/beans"

xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation = "http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<!-- Definition for textEditor bean -->

<bean id = "textEditor" class = "com.tutorialspoint.TextEditor">

<constructor-arg ref = "spellChecker"/>

</bean>

<!-- Definition for spellChecker bean -->

<bean id = "spellChecker" class = "com.tutorialspoint.SpellChecker"></bean>

</beans>

Cú pháp inject qua setter(ở file Beans.xml)

<?xml version = "1.0" encoding = "UTF-8"?>

<beans xmlns = "http://www.springframework.org/schema/beans"

xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation = "http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<!-- Definition for textEditor bean -->

<bean id = "textEditor" class = "com.tutorialspoint.TextEditor">

<property name = "spellChecker" ref = "spellChecker"/>

</bean>

<!-- Definition for spellChecker bean -->

<bean id = "spellChecker" class = "com.tutorialspoint.SpellChecker"></bean>

</beans>

Bạn có thể kết hợp cả hai, dựa trên contructor và setter DI nhưng đó là một quy tắc tốt để sử dụng contructor-arg cho các phụ thuộc bắt buộc và setters cho các phụ thuộc tùy chọn.

Code sẽ rõ ràng hơn với nguyên tắc DI và tách riêng hiệu quả hơn khi các đối tượng được cung cấp với các phụ thuộc của chúng. Đối tượng không tra cứu các phụ thuộc của nó và không biết vị trí hoặc lớp của các phụ thuộc, thay vào đó mọi thứ đều được Spring Framework chăm sóc.

***Injecting Inner Beans***

Như bạn đã biết các lớp bên trong Java được định nghĩa trong phạm vi của các lớp khác, tương tự, **inner bean** là các bean được định nghĩa trong phạm vi của các bean khác. Do đó, một phần tử <bean /> bên trong các phần tử <property /> hoặc <constructor-arg /> được gọi là bean bên trong và nó được hiển thị bên dưới.

<?xml version = "1.0" encoding = "UTF-8"?>

<beans xmlns = "http://www.springframework.org/schema/beans"

xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation = "http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id = "outerBean" class = "...">

<property name = "target">

<bean id = "innerBean" class = "..."/>

</property>

</bean>

</beans>

Ví dụ:

Class **TextEditer.java**

package com.tutorialspoint;

public class TextEditor {

private SpellChecker spellChecker;

// a setter method to inject the dependency.

public void setSpellChecker(SpellChecker spellChecker) {

System.out.println("Inside setSpellChecker." );

this.spellChecker = spellChecker;

}

// a getter method to return spellChecker

public SpellChecker getSpellChecker() {

return spellChecker;

}

public void spellCheck() {

spellChecker.checkSpelling();

}

}

Class **SpellChecker.java**

package com.tutorialspoint;

public class SpellChecker {

public SpellChecker(){

System.out.println("Inside SpellChecker constructor." );

}

public void checkSpelling(){

System.out.println("Inside checkSpelling." );

}

}

Class **MainApp.java**

package com.tutorialspoint;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

TextEditor te = (TextEditor) context.getBean("textEditor");

te.spellCheck();

}

}

Nội dung tệp **Beans.xml**

<?xml version = "1.0" encoding = "UTF-8"?>

<beans xmlns = "http://www.springframework.org/schema/beans"

xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation = "http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<!-- Definition for textEditor bean using inner bean -->

<bean id = "textEditor" class = "com.tutorialspoint.TextEditor">

<property name = "spellChecker">

<bean id = "spellChecker" class = "com.tutorialspoint.SpellChecker"/>

</property>

</bean>

</beans>

Kết quả:

Inside SpellChecker constructor.

Inside setSpellChecker.

Inside checkSpelling.

***Injecting Collection***

Bạn đã thấy cách định cấu hình kiểu dữ liệu nguyên thủy bằng cách sử dụng thuộc tính **giá trị** và tham chiếu đối tượng bằng thuộc tính **ref** của thẻ <property> trong tệp cấu hình Bean của bạn. Cả hai trường hợp xử lý chuyển giá trị số ít cho một hạt.

Bây giờ nếu bạn muốn chuyển các giá trị số nhiều như các Java Collection: List, Set, Map và Prop. Để xử lý tình huống này, Spring cung cấp bốn loại cấu hình collection như sau

|  |  |
| --- | --- |
| **Sr.No** | **Element & Description** |
| 1 | **<list>**  This helps in wiring ie injecting a list of values, allowing duplicates. |
| 2 | **<set>**  This helps in wiring a set of values but without any duplicates. |
| 3 | **<map>**  This can be used to inject a collection of name-value pairs where name and value can be of any type. |
| 4 | **<props>**  This can be used to inject a collection of name-value pairs where the name and value are both Strings. |

Ví dụ:

Nội dung của JavaCollections:

package com.tutorialspoint;

import java.util.\*;

public class JavaCollection {

List addressList;

Set addressSet;

Map addressMap;

Properties addressProp;

// a setter method to set List

public void setAddressList(List addressList) {

this.addressList = addressList;

}

// prints and returns all the elements of the list.

public List getAddressList() {

System.out.println("List Elements :" + addressList);

return addressList;

}

// a setter method to set Set

public void setAddressSet(Set addressSet) {

this.addressSet = addressSet;

}

// prints and returns all the elements of the Set.

public Set getAddressSet() {

System.out.println("Set Elements :" + addressSet);

return addressSet;

}

// a setter method to set Map

public void setAddressMap(Map addressMap) {

this.addressMap = addressMap;

}

// prints and returns all the elements of the Map.

public Map getAddressMap() {

System.out.println("Map Elements :" + addressMap);

return addressMap;

}

// a setter method to set Property

public void setAddressProp(Properties addressProp) {

this.addressProp = addressProp;

}

// prints and returns all the elements of the Property.

public Properties getAddressProp() {

System.out.println("Property Elements :" + addressProp);

return addressProp;

}

}

Nội dung của MainApp.java:

package com.tutorialspoint;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

JavaCollection jc=(JavaCollection)context.getBean("javaCollection");

jc.getAddressList();

jc.getAddressSet();

jc.getAddressMap();

jc.getAddressProp();

}

}

Nội dung file Beans.xml – cấu hình các collection:

<?xml version = "1.0" encoding = "UTF-8"?>

<beans xmlns = "http://www.springframework.org/schema/beans"

xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation = "http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<!-- Definition for javaCollection -->

<bean id = "javaCollection" class = "com.tutorialspoint.JavaCollection">

<!-- results in a setAddressList(java.util.List) call -->

<property name = "addressList">

<list>

<value>INDIA</value>

<value>Pakistan</value>

<value>USA</value>

<value>USA</value>

</list>

</property>

<!-- results in a setAddressSet(java.util.Set) call -->

<property name = "addressSet">

<set>

<value>INDIA</value>

<value>Pakistan</value>

<value>USA</value>

<value>USA</value>

</set>

</property>

<!-- results in a setAddressMap(java.util.Map) call -->

<property name = "addressMap">

<map>

<entry key = "1" value = "INDIA"/>

<entry key = "2" value = "Pakistan"/>

<entry key = "3" value = "USA"/>

<entry key = "4" value = "USA"/>

</map>

</property>

<!-- results in a setAddressProp(java.util.Properties) call -->

<property name = "addressProp">

<props>

<prop key = "one">INDIA</prop>

<prop key = "one">INDIA</prop>

<prop key = "two">Pakistan</prop>

<prop key = "three">USA</prop>

<prop key = "four">USA</prop>

</props>

</property>

</bean>

</beans>

Kết quả:

List Elements :[INDIA, Pakistan, USA, USA]

Set Elements :[INDIA, Pakistan, USA]

ap Elements :{1 = INDIA, 2 = Pakistan, 3 = USA, 4 = USA}

Property Elements :{two = Pakistan, one = INDIA, three = USA, four = USA}

## *Injecting Bean References*

Định nghĩa Bean sau đây sẽ giúp bạn hiểu cách tiêm tham chiếu bean như một trong các thành phần của Collection. Thậm chí bạn có thể trộn tất cả các tham chiếu và các giá trị lại với nhau như được hiển thị trong đoạn mã sau

<?xml version = "1.0" encoding = "UTF-8"?>

<beans xmlns = "http://www.springframework.org/schema/beans"

xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation = "http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<!-- Bean Definition to handle references and values -->

<bean id = "..." class = "...">

<!-- Passing bean reference for java.util.List -->

<property name = "addressList">

<list>

<ref bean = "address1"/>

<ref bean = "address2"/>

<value>Pakistan</value>

</list>

</property>

<!-- Passing bean reference for java.util.Set -->

<property name = "addressSet">

<set>

<ref bean = "address1"/>

<ref bean = "address2"/>

<value>Pakistan</value>

</set>

</property>

<!-- Passing bean reference for java.util.Map -->

<property name = "addressMap">

<map>

<entry key = "one" value = "INDIA"/>

<entry key = "two" value-ref = "address1"/>

<entry key = "three" value-ref = "address2"/>

</map>

</property>

</bean>

</beans>

***Beans Auto-Wiring***

Bạn đã học cách định nghĩa các bean bằng phần tử <bean> và inject <bean> bằng cách sử dụng các phần tử <constructor-arg> và <property> trong tệp cấu hình XML.

Spring container có thể tự động xác định mối quan hệ giữa các bean cộng tác mà không cần sử dụng các phần tử <constructor-arg> và <property>, giúp cắt giảm số dòng cấu hình XML mà bạn viết cho một ứng dụng lớn dựa trên Spring.

## *Autowiring Modes*

## Sau đây là các chế độ inject tự động, có thể được sử dụng để hướng dẫn Spring container sử dụng chế độ tự động nào để tiêm sự phụ thuộc. Bạn sử dụng thuộc tính autowire của phần tử <bean /> để chỉ định chế độ autowire cho định nghĩa bean.Các chế độ autowire Spring hỗ trợ như sau:

|  |  |
| --- | --- |
| **Sr.No** | **Mode & Description** |
| 1 | **no**  This is default setting which means no autowiring and you should use explicit bean reference for wiring. You have nothing to do special for this wiring. This is what you already have seen in Dependency Injection chapter. |
| 2 | [byName](https://www.tutorialspoint.com/spring/spring_autowiring_byname.htm)  Autowiring by property name. Spring container looks at the properties of the beans on which *autowire* attribute is set to *byName* in the XML configuration file. It then tries to match and wire its properties with the beans defined by the same names in the configuration file. |
| 3 | [byType](https://www.tutorialspoint.com/spring/spring_autowiring_bytype.htm)  Autowiring by property datatype. Spring container looks at the properties of the beans on which *autowire* attribute is set to *byType* in the XML configuration file. It then tries to match and wire a property if its **type** matches with exactly one of the beans name in configuration file. If more than one such beans exists, a fatal exception is thrown. |
| 4 | [constructor](https://www.tutorialspoint.com/spring/spring_autowiring_byconstructor.htm)  Similar to byType, but type applies to constructor arguments. If there is not exactly one bean of the constructor argument type in the container, a fatal error is raised. |
| 5 | **autodetect**  Spring first tries to wire using autowire by *constructor*, if it does not work, Spring tries to autowire by *byType*. |

## Autowiring hoạt động tốt nhất khi nó được sử dụng nhất quán trong một dự án. Nếu autowing không được sử dụng một cách nhất quán, các nhà phát triển có thể nhầm lẫn khi sử dụng nó để kết nối chỉ một hoặc hai bean. Mặc dù,autowiring có thể giảm đáng kể nhu cầu chỉ định các thuộc tính hoặc đối số của hàm tạo nhưng bạn nên xem xét các hạn chế và nhược điểm của nó trước khi sử dụng.

|  |  |
| --- | --- |
| **Sr.No.** | **Limitations & Description** |
| 1 | **Overriding possibility**  You can still specify dependencies using <constructor-arg> and <property> settings which will always override autowiring. |
| 2 | **Primitive data types**  You cannot autowire so-called simple properties such as primitives, Strings, and Classes. |
| 3 | **Confusing nature**  Autowiring is less exact than explicit wiring, so if possible prefer using explict wiring. |

***Spring Annotation Based Configuration***

Bắt đầu từ Spring 2.5, có thể định cấu hình dependency injection bằng cách sử dụng **Annotation** . Vì vậy, thay vì sử dụng XML để mô tả hệ bean wiring, bạn có thể chuyển cấu hình bean vào chính lớp thành phần bằng cách sử dụng các chú thích trên lớp, phương thức hoặc khai báo trường có liên quan.

Annotation Injection được thực hiện trước khi XML injection. Do đó, cấu hình sau sẽ ghi đè lên cấu hình trước cho các thuộc tính nếu wiring thông qua cả hai phương pháp.

Theo mặc định, Annotation wiring không được bật trong bộ chứa Spring. Vì vậy, trước khi chúng ta có thể sử dụng hệ thống wiring dựa trên annotation, chúng ta sẽ cần kích hoạt nó trong tệp cấu hình Spring. Vì vậy, hãy xem xét tệp cấu hình sau trong trường hợp bạn muốn sử dụng bất kỳ chú thích nào trong ứng dụng Spring của mình.

<?xml version = "1.0" encoding = "UTF-8"?>

<beans xmlns = "http://www.springframework.org/schema/beans"

xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

xmlns:context = "http://www.springframework.org/schema/context"

xsi:schemaLocation = "http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<context:annotation-config/>

<!-- bean definitions go here -->

</beans>

Khi <context: annotation-config /> được định nghĩa, bạn có thể bắt đầu chú thích code của mình để cho biết Spring sẽ tự động inject các giá trị vào các thuộc tính, phương thức và hàm tạo. Chúng ta hãy xem một vài chú thích quan trọng để hiểu cách chúng hoạt động

|  |  |
| --- | --- |
| **Sr.No.** | **Annotation & Description** |
| 1 | [@Required](https://www.tutorialspoint.com/spring/spring_required_annotation.htm)  The @Required annotation applies to bean property setter methods.  <https://www.tutorialspoint.com/spring/spring_required_annotation.htm> |
| 2 | [@Autowired](https://www.tutorialspoint.com/spring/spring_autowired_annotation.htm)  The @Autowired annotation can apply to bean property setter methods, non-setter methods, constructor and properties. |
| 3 | [@Qualifier](https://www.tutorialspoint.com/spring/spring_qualifier_annotation.htm)  The @Qualifier annotation along with @Autowired can be used to remove the confusion by specifiying which exact bean will be wired.  <https://www.tutorialspoint.com/spring/spring_qualifier_annotation.htm> |
| 4 | [JSR-250 Annotations](https://www.tutorialspoint.com/spring/spring_jsr250_annotations.htm)  Spring supports JSR-250 based annotations which include @Resource, @PostConstruct and @PreDestroy annotations.  <https://www.tutorialspoint.com/spring/spring_jsr250_annotations.htm> |

Các **@Autowired** chú thích cung cấp cách điều khiển Beans vị trí và cách thức autowiring nên được thực hiện. Autowired Annotation có thể sử dụng để autowire bean qua thuộc tính, contructor và setter.

## *@Autowired on Setter Methods*

Bạn có thể sử dụng chú thích **@Autowired** trên các phương thức setter để loại bỏ phần tử <property> trong tệp cấu hình XML. Khi Spring tìm thấy một chú thích @Autowired được sử dụng với các phương thức setter, nó sẽ cố gắng thực hiện autowired **byType**  trên phương thức.

package com.tutorialspoint;

import org.springframework.beans.factory.annotation.Autowired;

public class TextEditor {

private SpellChecker spellChecker;

@Autowired

public void setSpellChecker( SpellChecker spellChecker ){

this.spellChecker = spellChecker;

}

public SpellChecker getSpellChecker( ) {

return spellChecker;

}

public void spellCheck() {

spellChecker.checkSpelling();

}

}

## *@Autowired on Properties*

## Bạn có thể sử dụng chú thích @Autowired trên các thuộc. Khi bạn sẽ chuyển các giá trị của các thuộc tính tự động bằng cách sử dụng <property> Spring sẽ tự động gán các thuộc tính đó với các giá trị hoặc tham chiếu được truyền. Vì vậy, với việc sử dụng @Autowired trên các thuộc tính, tệp TextEditor.java của bạn sẽ trở thành như sau

package com.tutorialspoint;

import org.springframework.beans.factory.annotation.Autowired;

public class TextEditor {

@Autowired

private SpellChecker spellChecker;

public TextEditor() {

System.out.println("Inside TextEditor constructor." );

}

public SpellChecker getSpellChecker( ){

return spellChecker;

}

public void spellCheck(){

spellChecker.checkSpelling();

}

}

## *@Autowired with (required = false) option*

Mặc định, chú thích @Autowired cho thấy sự phụ thuộc được yêu cầu tương tự như chú thích @Required, tuy nhiên, bạn có thể tắt hành vi mặc định bằng cách sử dụng tùy chọn **(required = false)** với @Autowired.

Ví dụ sau sẽ hoạt động ngay cả khi bạn không chuyển bất kỳ giá trị nào cho thuộc tính age nhưng vẫn sẽ yêu cầu thuộc tính name. Bạn có thể tự thử ví dụ này vì nó tương tự như ví dụ chú thích @Required ngoại trừ chỉ có tệp **Student.java** đã được thay đổi.

package com.tutorialspoint;

import org.springframework.beans.factory.annotation.Autowired;

public class Student {

private Integer age;

private String name;

@Autowired(required=false)

public void setAge(Integer age) {

this.age = age;

}

public Integer getAge() {

return age;

}

@Autowired

public void setName(String name) {

this.name = name;

}

public String getName() {

return name;

}

}

## *Java Based Configuration*

Cho đến nay bạn đã thấy cách chúng tôi định cấu hình Spring bean bằng tệp cấu hình XML. Nếu bạn cảm thấy thoải mái với cấu hình XML, thì thực sự không cần phải học cách tiến hành cấu hình dựa trên Java vì bạn sẽ đạt được kết quả tương tự bằng cách sử dụng một trong hai cấu hình có sẵn.

Tùy chọn cấu hình dựa trên Java cho phép bạn viết hầu hết cấu hình Spring mà không cần XML nhưng với sự trợ giúp của một số chú thích dựa trên Java được giải thích trong chương này.

## *@Configuration & @Bean annotation*

## Chú thích một lớp với @Configuration chỉ ra rằng lớp có thể được sử dụng bởi Spring IoC container làm nguồn định nghĩa bean. Các @Bean nói Spring là method với chú thích @Bean sẽ trả về một đối tượng cần được đăng ký như là một bean trong Application context Spring. Lớp @Configuration đơn giản nhất có thể như sau

package com.tutorialspoint;

import org.springframework.context.annotation.\*;

@Configuration

public class HelloWorldConfig {

@Bean

public HelloWorld helloWorld(){

return new HelloWorld();

}

}

## Đoạn code trên được sử dụng tương tự với đoạn này ở trong file xml:

<beans>

<bean id = "helloWorld" class = "com.tutorialspoint.HelloWorld" />

</beans>

## Ở đây, tên phương thức được chú thích với @Bean hoạt động như bean ID và nó tạo và trả về bean thực sự. Lớp cấu hình của bạn có thể có một khai báo cho nhiều hơn một @Bean. Khi các lớp cấu hình của bạn được xác định, bạn có thể tải và cung cấp chúng cho bộ chứa Spring bằng cách dùng *AnnotationConfigApplicationContext* như sau:

public static void main(String[] args) {

ApplicationContext ctx = new AnnotationConfigApplicationContext(HelloWorldConfig.class);

HelloWorld helloWorld = ctx.getBean(HelloWorld.class);

helloWorld.setMessage("Hello World!");

helloWorld.getMessage();

}

## Bạn có thể dùng nhiều lớp cấu hình như sau:

public static void main(String[] args) {

AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext();

ctx.register(AppConfig.class, OtherConfig.class);

ctx.register(AdditionalConfig.class);

ctx.refresh();

MyService myService = ctx.getBean(MyService.class);

myService.doStuff();

}

## Ví dụ:

## Đây là nội dung của tệp HelloWorldConfig.java

package com.tutorialspoint;

import org.springframework.context.annotation.\*;

@Configuration

public class HelloWorldConfig {

@Bean

public HelloWorld helloWorld(){

return new HelloWorld();

}

}

Đây là nội dung của tệp **HelloWorld.java**

package com.tutorialspoint;

public class HelloWorld {

private String message;

public void setMessage(String message){

this.message = message;

}

public void getMessage(){

System.out.println("Your Message : " + message);

}

}

Sau đây là nội dun của tệp **MainApp.java**

package com.tutorialspoint;

import org.springframework.context.ApplicationContext;

import org.springframework.context.annotation.\*;

public class MainApp {

public static void main(String[] args) {

ApplicationContext ctx =

new AnnotationConfigApplicationContext(HelloWorldConfig.class);

HelloWorld helloWorld = ctx.getBean(HelloWorld.class);

helloWorld.setMessage("Hello World!");

helloWorld.getMessage();

}

}

**Kết quả:**

Your Message : Hello World!

## *Injecting Bean Dependencies*

## Khi @Beans có sự phụ thuộc lẫn nhau, biểu thị rằng sự phụ thuộc đơn giản như có một phương thức bean gọi phương thức khác như sau

package com.tutorialspoint;

import org.springframework.context.annotation.\*;

@Configuration

public class AppConfig {

@Bean

public Foo foo() {

return new Foo(bar());

}

@Bean

public Bar bar() {

return new Bar();

}

}

## Ở đây, Bean foo nhận được một tham chiếu đến bar thông qua phép tiêm constructor.

## *The @Import Annotation*

Các **@import** cho phép tải các định nghĩa @Bean từ một lớp cấu hình. Hãy xem xét một lớp ConfigA như sau

@Configuration

public class ConfigA {

@Bean

public A a() {

return new A();

}

}

Bạn có thể import khai báo Bean trên khai báo Bean khác như sau

@Configuration

@Import(ConfigA.class)

public class ConfigB {

@Bean

public B b() {

return new B();

}

}

Bây giờ, thay vì cần chỉ định cả ConfigA.Class và ConfigB.Class khi khởi tạo context, chỉ cần cung cấp ConfigB như sau -

public static void main(String[] args) {

ApplicationContext ctx = new AnnotationConfigApplicationContext(ConfigB.class);

// now both beans A and B will be available...

A a = ctx.getBean(A.class);

B b = ctx.getBean(B.class);

}

## *Lifecycle Callbacks*

Chú thích @Bean hỗ trợ chỉ định các phương thức gọi lại hủy và khởi tạo tùy ý, giống như các thuộc tính init-method và destroy-method của Spring XML trên phần tử bean

public class Foo {

public void init() {

// initialization logic

}

public void cleanup() {

// destruction logic

}

}

@Configuration

public class AppConfig {

@Bean(initMethod = "init", destroyMethod = "cleanup" )

public Foo foo() {

return new Foo();

}

}

## *Specifying Bean Scope*

Phạm vi mặc định là singleton, nhưng bạn có thể ghi đè lên điều này bằng chú thích @Scope như sau -

@Configuration

public class AppConfig {

@Bean

@Scope("prototype")

public Foo foo() {

return new Foo();

}

}

# *Event Handling in Spring*

Bạn đã thấy trong tất cả các chương rằng cốt lõi của Spring là **ApplicationContext** , quản lý vòng đời hoàn chỉnh của các Bean. ApplicationContext xuất bản một số loại sự kiện khi tải các bean. Ví dụ: *ContextStartedEvent* được xuất bản khi bối cảnh được bắt đầu và *ContextStoppedEvent* được xuất bản khi bối cảnh bị dừng.

Xử lý trong trường hợp *ApplicationContext* được cung cấp thông qua các Class *ApplicationEvent*  và Interface *ApplicationListener*. Do đó, nếu một bean implement *ApplicationListener* , thì mỗi khi *ApplicationEvent* được xuất bản lên ApplicationContext, bean đó sẽ được thông báo.

|  |  |
| --- | --- |
| **Sr.No.** | **Spring Built-in Events & Description** |
| 1 | **ContextRefreshedEvent**  This event is published when the *ApplicationContext* is either initialized or refreshed. This can also be raised using the refresh() method on the *ConfigurableApplicationContext* interface. |
| 2 | **ContextStartedEvent**  This event is published when the *ApplicationContext* is started using the start() method on the *ConfigurableApplicationContext* interface. You can poll your database or you can restart any stopped application after receiving this event. |
| 3 | **ContextStoppedEvent**  This event is published when the *ApplicationContext* is stopped using the stop() method on the *ConfigurableApplicationContext* interface. You can do required housekeep work after receiving this event. |
| 4 | **ContextClosedEvent**  This event is published when the *ApplicationContext* is closed using the close() method on the *ConfigurableApplicationContext* interface. A closed context reaches its end of life; it cannot be refreshed or restarted. |
| 5 | **RequestHandledEvent**  This is a web-specific event telling all beans that an HTTP request has been serviced. |

Event handling của Spring là một luồng đơn vì vậy nếu event được publish, trừ và đến khi tất cả người nhận nhận được thông báo, các process sẽ bị chặn và luồng sẽ không tiếp tục. Do đó, cần thận trọng khi thiết kế ứng dụng khi dùng event handling .

## *Listening to Context Events*

Để bắt một sự kiện của context, một bean nên implement interface *ApplicationListener* chỉ có một phương thức **trênApplicationEvent()**. Vì vậy, hãy để chúng tôi viết một ví dụ để xem cách các sự kiện lan truyền và cách bạn có thể đặt mã của mình để thực hiện nhiệm vụ được yêu cầu dựa trên các sự kiện nhất định.

Đây là nội dung của tệp **HelloWorld.java**

package com.tutorialspoint;

public class HelloWorld {

private String message;

public void setMessage(String message){

this.message = message;

}

public void getMessage(){

System.out.println("Your Message : " + message);

}

}

Sau đây là nội dung của tệp **CStartEventHandler.java**

package com.tutorialspoint;

import org.springframework.context.ApplicationListener;

import org.springframework.context.event.ContextStartedEvent;

public class CStartEventHandler

implements ApplicationListener<ContextStartedEvent>{

public void onApplicationEvent(ContextStartedEvent event) {

System.out.println("ContextStartedEvent Received");

}

}

Sau đây là nội dung của tệp **CStopEventHandler.java**

package com.tutorialspoint;

import org.springframework.context.ApplicationListener;

import org.springframework.context.event.ContextStoppedEvent;

public class CStopEventHandler

implements ApplicationListener<ContextStoppedEvent>{

public void onApplicationEvent(ContextStoppedEvent event) {

System.out.println("ContextStoppedEvent Received");

}

}

Sau đây là nội dung của tệp **MainApp.java**

package com.tutorialspoint;

import org.springframework.context.ConfigurableApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

ConfigurableApplicationContext context =

new ClassPathXmlApplicationContext("Beans.xml");

// Let us raise a start event.

context.start();

HelloWorld obj = (HelloWorld) context.getBean("helloWorld");

obj.getMessage();

// Let us raise a stop event.

context.stop();

}

}

Sau đây là tệp cấu hình **Beans.xml**

<?xml version = "1.0" encoding = "UTF-8"?>

<beans xmlns = "http://www.springframework.org/schema/beans"

xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation = "http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id = "helloWorld" class = "com.tutorialspoint.HelloWorld">

<property name = "message" value = "Hello World!"/>

</bean>

<bean id = "cStartEventHandler" class = "com.tutorialspoint.CStartEventHandler"/>

<bean id = "cStopEventHandler" class = "com.tutorialspoint.CStopEventHandler"/>

</beans>

Khi bạn đã hoàn tất việc tạo các tệp cấu hình nguồn và bean, hãy để chúng tôi chạy ứng dụng. Nếu mọi thứ đều ổn với ứng dụng của bạn, nó sẽ in thông báo sau -

ContextStartedEvent Received

Your Message : Hello World!

ContextStoppedEvent Received

Nếu bạn thích, bạn có thể xuất bản các sự kiện tùy chỉnh của riêng mình và sau đó bạn có thể nắm bắt tương tự để thực hiện bất kỳ hành động nào chống lại các sự kiện tùy chỉnh đó. Nếu bạn quan tâm đến việc viết các sự kiện tùy chỉnh của riêng bạn, bạn có thể kiểm tra các [sự kiện tùy chỉnh trong mùa xuân.](https://www.tutorialspoint.com/spring/custom_events_in_spring.htm)

<https://www.tutorialspoint.com/spring/custom_events_in_spring.htm>

# *AOP với Spring Framework*

Một trong những thành phần chính của Spring Framework là **lập trình hướng khía cạnh (AOP)** . Lập trình hướng theo khía cạnh đòi hỏi phải phá vỡ logic chương trình thành các phần riêng biệt được gọi là *concerns*. Các chức năng trải rộng trên nhiều điểm của một ứng dụng được gọi là **cross-cutting concerns** và những mối quan tâm xuyên suốt này tách biệt về mặt khái niệm với business logic của ứng dụng. Có nhiều ví dụ điển hình tốt khác nhau của lập trình các khía cạnh như logging, auditing, transaction, security, caching, v.v.

Đơn vị chính của mô đun hóa trong OOP là lớp,khi đó trong AOP, đơn vị của mô đun là khía cạnh. Dependency Injection giúp bạn tách rời các đối tượng ứng dụng của bạn với nhau và AOP giúp bạn tách rời các mối quan tâm xuyên suốt(cross-cutting concern) từ các đối tượng mà chúng ảnh hưởng. AOP giống như các trình kích hoạt trong các ngôn ngữ lập trình như Perl, .NET, Java và các ngôn ngữ khác.

Mô-đun Spring AOP cung cấp các thiết bị chặn để chặn ứng dụng. Ví dụ, khi một phương thức được thực thi, bạn có thể thêm chức năng bổ sung trước hoặc sau khi thực hiện phương thức.

## *AOP Terminologies(thuật ngữ trong AOP)*

Trước khi bắt đầu làm việc với AOP, chúng ta hãy làm quen với các khái niệm và thuật ngữ AOP. Các thuật ngữ này không dành riêng cho Spring, thay vào đó chúng có liên quan đến AOP.

|  |  |
| --- | --- |
| **Sr.No** | **Terms & Description** |
| 1 | **Aspect**  This is a module which has a set of APIs providing cross-cutting requirements. For example, a logging module would be called AOP aspect for logging. An application can have any number of aspects depending on the requirement. |
| 2 | **Join point**  This represents a point in your application where you can plug-in the AOP aspect. You can also say, it is the actual place in the application where an action will be taken using Spring AOP framework. |
| 3 | **Advice**  This is the actual action to be taken either before or after the method execution. This is an actual piece of code that is invoked during the program execution by Spring AOP framework. |
| 4 | **Pointcut**  This is a set of one or more join points where an advice should be executed. You can specify pointcuts using expressions or patterns as we will see in our AOP examples. |
| 5 | **Introduction**  An introduction allows you to add new methods or attributes to the existing classes. |
| 6 | **Target object**  The object being advised by one or more aspects. This object will always be a proxied object, also referred to as the advised object. |
| 7 | **Weaving**  Weaving is the process of linking aspects with other application types or objects to create an advised object. This can be done at compile time, load time, or at runtime. |

## *Types of Advice*

|  |  |
| --- | --- |
| **Sr.No** | **Advice & Description** |
| 1 | **Before**  Run advice before the a method execution. |
| 2 | **After**  Run advice after the method execution, regardless of its outcome. |
| 3 | **after-returning**  Run advice after the a method execution only if method completes successfully. |
| 4 | **after-throwing**  Run advice after the a method execution only if method exits by throwing an exception. |
| 5 | **around**  Run advice before and after the advised method is invoked. |

## *Custom Aspects Implementation*

Spring hỗ trợ cách sử dụng **@AspectJ annotation style**  và cách sử dụng dựa trên **schema-based** để thực hiện các khía cạnh tùy chỉnh . These two approaches have been explained in detail in the following sections.

|  |  |
| --- | --- |
| **Sr.No** | **Approach & Description** |
| 1 | [XML Schema based](https://www.tutorialspoint.com/spring/schema_based_aop_appoach.htm)  Aspects are implemented using the regular classes along with XML based configuration.  <https://www.tutorialspoint.com/spring/schema_based_aop_appoach.htm> |
| 2 | [@AspectJ based](https://www.tutorialspoint.com/spring/aspectj_based_aop_appoach.htm)  @AspectJ refers to a style of declaring aspects as regular Java classes annotated with Java 5 annotations.  <https://www.tutorialspoint.com/spring/aspectj_based_aop_appoach.htm> |

## *@AspectJ dựa vào AOP với Spring*

@AspectJ đề cập đến một kiểu khai báo các aspect như các lớp Java thông thường được chú thích bằng các chú thích Java 5. Hỗ trợ @AspectJ được kích hoạt bằng cách bao gồm phần tử sau trong tệp cấu hình dựa trên file XML của bạn.

*<aop:aspectj-autoproxy/>*

Bạn cũng sẽ cần các thư viện AspectJ cần trên đường dẫn ứng dụng của bạn

***Declaring an aspect***

Các Class Aspect giống như bất kỳ bean thông thường khác và có thể có các phương thức và trường giống như bất kỳ lớp nào khác, ngoại trừ việc chúng sẽ được chú thích bằng @Aspect như sau -

package org.xyz;

import org.aspectj.lang.annotation.Aspect;

@Aspect

public class AspectModule {

}

Chúng sẽ được cấu hình trong XML giống như bất kỳ bean nào khác như sau -

<bean id = "myAspect" class = "org.xyz.AspectModule">

<!-- configure properties of aspect here as normal -->

</bean>

## *Declaring a pointcut*

Một **Pointcut**(**điểm cắt)** giúp xác định các **joint point(điểm tham gia)** (tức là các phương thức) quan tâm sẽ được thực hiện với các advice khác nhau. Trong khi làm việc với cấu hình dựa trên @ AspectJ, khai báo pointcut có hai phần -

* **Pointcut expression** xác định chính xác phương thức thực thi mà chúng ta quan tâm.
* **Pointcut signeture** bao gồm một tên và bất kỳ số lượng tham số. Phần thân của method là không liên quan và trên thực tế nên trống rỗng.

Ví dụ sau định nghĩa một điểm cắt có tên 'businessService' sẽ khớp với việc thực thi mọi phương thức có sẵn trong các lớp theo gói com.xyz.myapp.service -

import org.aspectj.lang.annotation.Pointcut;

@Pointcut("execution(\* com.xyz.myapp.service.\*.\*(..))") // expression

private void businessService() {} // signature

Ví dụ sau định nghĩa một điểm cắt có tên 'getname' sẽ khớp với việc thực thi phương thức getName () có sẵn trong lớp Sinh viên theo gói com.tutorialspoint

import org.aspectj.lang.annotation.Pointcut;

@Pointcut("execution(\* com.tutorialspoint.Student.getName(..))")

private void getname() {}

## *Declaring advices*

Bạn có thể khai báo bất kỳ advice nào trong số 5 advice bằng cách sử dụng các chú thích @{ADVICE-NAME} như được đưa ra trong đoạn mã sau. Điều này giả định rằng bạn đã xác định một pointcut businessService()

@Before("businessService()")

public void doBeforeTask(){

...

}

@After("businessService()")

public void doAfterTask(){

...

}

@AfterReturning(pointcut = "businessService()", returning = "retVal")

public void doAfterReturnningTask(Object retVal) {

// you can intercept retVal here.

...

}

@AfterThrowing(pointcut = "businessService()", throwing = "ex")

public void doAfterThrowingTask(Exception ex) {

// you can intercept thrown exception here.

...

}

@Around("businessService()")

public void doAroundTask(){

...

}

Bạn có thể định nghĩa một pointcut cho bất kì advice nào. Sau đây là một ví dụ để xác định point cut cho @Before advice

@Before("execution(\* com.xyz.myapp.service.\*.\*(..))")

public doBeforeTask(){

...

}

## *@AspectJ Based AOP Example*

|  |  |
| --- | --- |
| **Steps** | **Description** |
| 1 | Create a project with a name *SpringExample* and create a package *com.tutorialspoint* under the **src** folder in the created project. |
| 2 | Add required Spring libraries using *Add External JARs* option as explained in the *Spring Hello World Example* chapter. |
| 3 | Add Spring AOP specific libraries **aspectjrt.jar, aspectjweaver.jar** and **aspectj.jar** in the project. |
| 4 | Create Java classes **Logging**, *Student* and *MainApp* under the *com.tutorialspoint* package. |
| 5 | Create Beans configuration file *Beans.xml* under the **src** folder. |
| 6 | The final step is to create the content of all the Java files and Bean Configuration file and run the application as explained below. |

Nôị dung file **Logging.java**

package com.tutorialspoint;

import org.aspectj.lang.annotation.Aspect;

import org.aspectj.lang.annotation.Pointcut;

import org.aspectj.lang.annotation.Before;

import org.aspectj.lang.annotation.After;

import org.aspectj.lang.annotation.AfterThrowing;

import org.aspectj.lang.annotation.AfterReturning;

import org.aspectj.lang.annotation.Around;

@Aspect

public class Logging {

/\*\* Following is the definition for a pointcut to select

\* all the methods available. So advice will be called

\* for all the methods.

\*/

@Pointcut("execution(\* com.tutorialspoint.\*.\*(..))")

private void selectAll(){}

/\*\*

\* This is the method which I would like to execute

\* before a selected method execution.

\*/

@Before("selectAll()")

public void beforeAdvice(){

System.out.println("Going to setup student profile.");

}

/\*\*

\* This is the method which I would like to execute

\* after a selected method execution.

\*/

@After("selectAll()")

public void afterAdvice(){

System.out.println("Student profile has been setup.");

}

/\*\*

\* This is the method which I would like to execute

\* when any method returns.

\*/

@AfterReturning(pointcut = "selectAll()", returning = "retVal")

public void afterReturningAdvice(Object retVal){

System.out.println("Returning:" + retVal.toString() );

}

/\*\*

\* This is the method which I would like to execute

\* if there is an exception raised by any method.

\*/

@AfterThrowing(pointcut = "selectAll()", throwing = "ex")

public void AfterThrowingAdvice(IllegalArgumentException ex){

System.out.println("There has been an exception: " + ex.toString());

}

}

Nội dung file **Student.java**

package com.tutorialspoint;

public class Student {

private Integer age;

private String name;

public void setAge(Integer age) {

this.age = age;

}

public Integer getAge() {

System.out.println("Age : " + age );

return age;

}

public void setName(String name) {

this.name = name;

}

public String getName() {

System.out.println("Name : " + name );

return name;

}

public void printThrowException(){

System.out.println("Exception raised");

throw new IllegalArgumentException();

}

}

Nội dung **MainApp.java** file

package com.tutorialspoint;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

Student student = (Student) context.getBean("student");

student.getName();

student.getAge();

student.printThrowException();

}

}

Nội dung file **Beans.xml**

<?xml version = "1.0" encoding = "UTF-8"?>

<beans xmlns = "http://www.springframework.org/schema/beans"

xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

xmlns:aop = "http://www.springframework.org/schema/aop"

xsi:schemaLocation = "http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop-3.0.xsd ">

<aop:aspectj-autoproxy/>

<!-- Definition for student bean -->

<bean id = "student" class = "com.tutorialspoint.Student">

<property name = "name" value = "Zara" />

<property name = "age" value = "11"/>

</bean>

<!-- Definition for logging aspect -->

<bean id = "logging" class = "com.tutorialspoint.Logging"/>

</beans>

Khi bạn đã hoàn tất việc tạo các tệp cấu hình nguồn và bean, hãy để chúng tôi chạy ứng dụng. Nếu mọi thứ đều ổn với ứng dụng của bạn, nó sẽ in thông báo sau

Going to setup student profile.

Name : Zara

Student profile has been setup.

Returning:Zara

Going to setup student profile.

Age : 11

Student profile has been setup.

Returning:11

Going to setup student profile.

Exception raised

Student profile has been setup.

There has been an exception: java.lang.IllegalArgumentException

.....

other exception content

## *Spring Web MVC Framework*

Spring Web MVC framework cung cấp kiến ​​trúc Model-View-Controller (MVC) và các thành phần sẵn sàng có thể được sử dụng để phát triển các ứng dụng web linh hoạt và ít bị phụ thuộc. MVC parttern cho kết quả tách biệt các khía cạnh khác nhau của ứng dụng (logic đầu vào, logic nghiệp vụ và logic UI), đồng thời cung cấp một khớp nối lỏng lẻo giữa các yếu tố này.

* Các **Model** đóng gói dữ liệu ứng dụng và nói chung nó sẽ bao gồm POJO.
* Các **View** có trách nhiệm để hiển thị các dữ liệu model và nói chung nó tạo ra HTML mà trình duyệt của khách hàng có thể giải thích.
* Bộ **Controller** chịu trách nhiệm xử lý các yêu cầu của người dùng và xây dựng một mô hình phù hợp và chuyển nó đến dạng xem để hiển thị.

## The DispatcherServlet

The Spring Web model-view-controller (MVC) framework được thiết kế xung quanh một ***DispatcherServlet*** xử lý tất cả các yêu cầu và phản hồi HTTP. Quy trình xử lý yêu cầu của Spring Web MVC *DispatcherServlet* được minh họa trong sơ đồ sau

## *C:\Users\ad\Desktop\spring_dispatcherservlet.png*

Sau đây là chuỗi các sự kiện tương ứng với yêu cầu HTTP đến với *DispatcherServlet* -

* Sau khi nhận được một yêu cầu HTTP, *DispatcherServlet* tư vấn các *HandlerMapping* để gọi *Controller* thích hợp .
* Controller nhận yêu cầu và gọi các phương thức dịch vụ phù hợp dựa trên phương thức GET hoặc POST đã sử dụng. Phương thức dịch vụ sẽ đặt dữ liệu mô hình dựa trên logic nghiệp vụ đã xác định và trả về tên view cho *DispatcherServlet* .
* Các *DispatcherServlet* sẽ nhận sự giúp đỡ từ *ViewResolver*  lấy các view được xác định cho yêu cầu.
* Khi view được hoàn tất, *DispatcherServlet* chuyển dữ liệu model đến chế độ view mà cuối cùng được hiển thị trên trình duyệt.

Tất cả các thành phần được đề cập ở trên, ví dụ HandlerMapping, Trình điều khiển và ViewResolver là các phần của *WebApplicationContext* w, một phần mở rộng của *ApplicationContext* đơn giản với một số tính năng bổ sung cần thiết cho các ứng dụng web.

## *Required Configuration*

## Bạn cần ánh xạ các yêu cầu mà bạn muốn *DispatcherServlet* xử lý, bằng cách sử dụng ánh xạ URL trong tệp web.xml . Sau đây là một ví dụ để hiển thị khai báo và ánh xạ cho ví dụ HelloWeb *DispatcherServlet* -

<web-app id = "WebApp\_ID" version = "2.4"

xmlns = "http://java.sun.com/xml/ns/j2ee"

xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation = "http://java.sun.com/xml/ns/j2ee

http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd">

<display-name>Spring MVC Application</display-name>

<servlet>

<servlet-name>HelloWeb</servlet-name>

<servlet-class>

org.springframework.web.servlet.DispatcherServlet

</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>HelloWeb</servlet-name>

<url-pattern>\*.jsp</url-pattern>

</servlet-mapping>

</web-app>

Tệp **web.xml** sẽ được lưu trong thư mục WebContent / WEB-INF của ứng dụng web của bạn. Khi khởi tạo **HelloWeb** DispatcherServlet, framework sẽ tải Application Context từ một tệp có tên **[ servelet-name]-servlet.xml** nằm trong WebContent / WEB-INFdirectory của ứng dụng. Trong trường hợp này, tệp của chúng tôi sẽ là **HelloWebservlet.xml** .

Tiếp theo, thẻ <servlet-maps> cho biết URL nào sẽ được xử lý bởi DispatcherServlet. Tại đây, tất cả các yêu cầu HTTP kết thúc bằng **.jsp** sẽ được xử lý bởi **HelloWeb** DispatcherServlet.

Nếu bạn không muốn đi với tên tệp mặc định là *[tên dịch vụ] -servlet.xml* và vị trí mặc định là *WebContent / WEB-INF* , bạn có thể tùy chỉnh tên tệp và vị trí này bằng cách thêm trình nghe của trình dịch vụ *ContextLoaderListener* trong tệp web.xml của bạn như sau -

<web-app...>

<!-------- *DispatcherServlet* definition goes here----->

....

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/HelloWeb-servlet.xml</param-value>

</context-param>

<listener>

<listener-class>

org.springframework.web.context.ContextLoaderListener

</listener-class>

</listener>

</web-app>

Bây giờ, hãy để chúng tôi kiểm tra cấu hình bắt buộc cho tệp **HelloWeb-servlet.xml** , được đặt trong thư mục *WebContent / WEB-INF* của ứng dụng web của bạn -

<beans xmlns = "http://www.springframework.org/schema/beans"

xmlns:context = "http://www.springframework.org/schema/context"

xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation = "http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<context:component-scan base-package = "com.tutorialspoint" />

<bean class = "org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name = "prefix" value = "/WEB-INF/jsp/" />

<property name = "suffix" value = ".jsp" />

</bean>

</beans>

Sau đây là những điểm quan trọng về tệp **HelloWeb-servlet.xml** -

* Tệp *[servlet-name] -servlet.xml* sẽ được sử dụng để tạo các bean được xác định, ghi đè các định nghĩa của bất kỳ bean nào được xác định có cùng tên trong phạm vi toàn cục.
* Thẻ *<context:component-scan...>* sẽ được sử dụng để kích hoạt khả năng quét chú thích Spring MVC cho phép sử dụng các chú thích như @Controll và @RequestMapping, v.v.
* Các *InternalResourceViewResolver* sẽ có những quy định để giải quyết các tên view. Theo quy tắc được xác định ở trên, tên view là **hello** được ủy quyền cho việc triển khai chế độ view được đặt tại */CHB-INF/jsp/hello.jsp* .

Phần sau đây sẽ cho bạn thấy cách tạo các thành phần thực tế của mình, tức là Controller, Model, View

## *Defining a Controller*

DispatcherServlet ủy quyền yêu cầu cho controller để thực thi chức năng dành riêng cho nó. Các **@Controller** chú thích chỉ ra rằng một lớp đặc biệt phục vụ vai trò của một bộ điều khiển. Các **@RequestMapping** chú thích được sử dụng để ánh xạ một địa chỉ URL hoặc là toàn bộ một lớp hoặc một phương thức xử lý cụ thể.

@Controller

@RequestMapping("/hello")

public class HelloController {

@RequestMapping(method = RequestMethod.GET)

public String printHello(ModelMap model) {

model.addAttribute("message", "Hello Spring MVC Framework!");

return "hello";

}

}

Các **@Controller** chú thích định nghĩa lớp như một bộ điều khiển Spring MVC. Ở đây, cách sử dụng đầu tiên của **@RequestMapping** chỉ ra rằng tất cả các phương thức xử lý trên bộ điều khiển này đều liên quan đến đường dẫn **/hello** . Chú thích tiếp theo **@RequestMapping (method = RequestMethod.GET)** được sử dụng để khai báo **phương thức printHello (** ) làm phương thức dịch vụ mặc định của bộ điều khiển để xử lý yêu cầu HTTP GET. Bạn có thể định nghĩa một phương thức khác để xử lý bất kỳ yêu cầu POST nào tại cùng một URL.

Bạn có thể viết trình điều khiển trên ở dạng khác trong đó bạn có thể thêm các thuộc tính bổ sung trong *@RequestMapping* như sau

@Controller

public class HelloController {

@RequestMapping(value = "/hello", method = RequestMethod.GET)

public String printHello(ModelMap model) {

model.addAttribute("message", "Hello Spring MVC Framework!");

return "hello";

}

}

Thuộc tính **value**  chỉ URL mà method xử lý được ánh xạ và các **method** định nghĩa phương thức để xử lý yêu cầu HTTP GET. Những điểm quan trọng sau đây cần được lưu ý về Controller được xác định ở trên -

* Bạn sẽ xác định logic nghiệp vụ cần thiết bên trong một phương thức dịch vụ. Bạn có thể gọi một phương thức khác bên trong phương thức này theo yêu cầu.
* Dựa trên logic nghiệp vụ được xác định, bạn sẽ tạo một model trong phương thức này. Bạn có thể sử dụng để set các thuộc tính model khác nhau và các thuộc tính này sẽ được truy cập bởi để trình biểu diễn đến view cuối cùng. Ví dụ này tạo ra một model với thuộc tính "message"
* Một phương thức dịch vụ được xác định có thể trả về một Chuỗi chứa tên của **view** sẽ được sử dụng để render ra model. Ví dụ này trả về "hello" dưới dạng tên xem logic.

## *Create JSP view*

Spring MVC hỗ trợ nhiều kiểu xem cho các công nghệ trình bày khác nhau. Chúng bao gồm - Các tệp tin JSP, HTML, PDF, Excel, XML, Velocity, XSLT, JSON, Atom và RSS, JasperReports, v.v. Nhưng thông thường nhất chúng ta sử dụng các mẫu JSP được viết bằng JSTL.

Chúng ta hãy viết một view đơn giản: **Hello** view trong /WEB-INF/hello/hello.jsp

<html>

<head>

<title>Hello Spring MVC</title>

</head>

<body>

<h2>${message}</h2>

</body>

</html>

Ở đây **$ {message}** là thuộc tính mà chúng ta đã thiết lập bên trong Controller. Bạn có thể có nhiều thuộc tính được hiển thị trong chế độ xem của bạn.

## *Ví dụ về Spring Web MVC Framework*

Dựa trên các khái niệm trên, hãy để chúng tôi kiểm tra một vài ví dụ quan trọng sẽ giúp bạn xây dựng các ứng dụng web với Spring

|  |  |
| --- | --- |
| **Sr.No.** | **Example & Description** |
| 1 | [Spring MVC Hello World Example](https://www.tutorialspoint.com/spring/spring_mvc_hello_world_example.htm)  This example will explain how to write a simple Spring Web Hello World application.  <https://www.tutorialspoint.com/spring/spring_mvc_hello_world_example.htm> |
| 2 | [Spring MVC Form Handling Example](https://www.tutorialspoint.com/spring/spring_mvc_form_handling_example.htm)  This example will explain how to write a Spring Web application using HTML forms to submit the data to the controller and display a processed result.  <https://www.tutorialspoint.com/spring/spring_mvc_form_handling_example.htm> |
| 3 | [Spring Page Redirection Example](https://www.tutorialspoint.com/spring/spring_page_redirection_example.htm)  Learn how to use page redirection functionality in Spring MVC Framework.  <https://www.tutorialspoint.com/spring/spring_page_redirection_example.htm> |
| 4 | [Spring Static Pages Example](https://www.tutorialspoint.com/spring/spring_static_pages_example.htm)  Learn how to access static pages along with dynamic pages in Spring MVC Framework.  <https://www.tutorialspoint.com/spring/spring_static_pages_example.htm> |
| 5 | [Spring Exception Handling Example](https://www.tutorialspoint.com/spring/spring_exception_handling_example.htm)  Learn how to handle exceptions in Spring MVC Framework.  <https://www.tutorialspoint.com/spring/spring_exception_handling_example.htm> |

## *Logging with Log4j*

Đây là một chức năng Log4J rất dễ sử dụng trong các ứng dụng Spring. Ví dụ sau đây sẽ đưa bạn qua các bước đơn giản để giải thích sự tích hợp đơn giản giữa Log4J và Spring.

Chúng tôi giả sử bạn đã cài đặt **log4J** trên máy của mình. Nếu bạn không có thì bạn có thể tải xuống từ <https://logging.apache.org/> và chỉ cần giải nén tệp nén trong bất kỳ thư mục nào. Chúng tôi sẽ chỉ sử dụng **log4j-xyzjar** trong dự án của chúng tôi.

Tiếp theo, chúng ta hãy có một IDE Eclipse hoạt động và thực hiện các bước sau để phát triển Ứng dụng web dựa trên biểu mẫu động bằng Spring Web Framework

|  |  |
| --- | --- |
| **Steps** | **Description** |
| 1 | Create a project with a name *SpringExample* and create a package *com.tutorialspoint* under the **src** folder in the created project. |
| 2 | Add required Spring libraries using *Add External JARs* option as explained in the *Spring Hello World Example* chapter. |
| 3 | Add log4j library *log4j-x.y.z.jar* as well in your project using using *Add External JARs*. |
| 4 | Create Java classes *HelloWorld* and *MainApp* under the *com.tutorialspoint* package. |
| 5 | Create Beans configuration file *Beans.xml* under the **src** folder. |
| 6 | Create log4J configuration file *log4j.properties* under the **src** folder. |
| 7 | The final step is to create the content of all the Java files and Bean Configuration file and run the application as explained below. |

Đây là nội dung của tệp **HelloWorld.java**

package com.tutorialspoint;

public class HelloWorld {

private String message;

public void setMessage(String message){

this.message = message;

}

public void getMessage() {

System.out.println("Your Message : " + message);

}

}

Sau đây là nội dung của tệp thứ hai **MainApp.java**

package com.tutorialspoint;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import org.apache.log4j.Logger;

public class MainApp {

static Logger log = Logger.getLogger(MainApp.class.getName());

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

log.info("Going to create HelloWord Obj");

HelloWorld obj = (HelloWorld) context.getBean("helloWorld");

obj.getMessage();

log.info("Exiting the program");

}

}

Bạn có thể tạo thông báo **debug** và **error** theo cách tương tự như chúng tôi đã tạo thông báo thông tin. Bây giờ chúng ta hãy xem nội dung của tệp **Beans.xml**

<?xml version = "1.0" encoding = "UTF-8"?>

<beans xmlns = "http://www.springframework.org/schema/beans"

xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation = "http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id = "helloWorld" class = "com.tutorialspoint.HelloWorld">

<property name = "message" value = "Hello World!"/>

</bean>

</beans>

Sau đây là nội dung của **log4j.properies**  để xác định các quy tắc chuẩn cần thiết cho Log4J để tạo log message

# Define the root logger with appender file

log4j.rootLogger = DEBUG, FILE

# Define the file appender

log4j.appender.FILE=org.apache.log4j.FileAppender

# Set the name of the file

log4j.appender.FILE.File=C:\\log.out

# Set the immediate flush to true (default)

log4j.appender.FILE.ImmediateFlush=true

# Set the threshold to debug mode

log4j.appender.FILE.Threshold=debug

# Set the append to false, overwrite

log4j.appender.FILE.Append=false

# Define the layout for file appender

log4j.appender.FILE.layout=org.apache.log4j.PatternLayout

log4j.appender.FILE.layout.conversionPattern=%m%n

Khi bạn đã hoàn tất việc tạo tệp cấu hình nguồn và bean, hãy để chúng tôi chạy ứng dụng. Nếu mọi thứ đều ổn với ứng dụng của bạn, điều này sẽ in thông báo sau trong bảng điều khiển Eclipse -

Your Message : Hello World!

Nếu bạn kiểm tra ổ C: \\, thì bạn sẽ tìm thấy tệp nhật ký **log.out của mình** với nhiều thông điệp **tường trình** khác nhau, như một cái gì đó như sau -

<!-- initialization log messages -->

Going to create HelloWord Obj

Returning cached instance of singleton bean 'helloWorld'

Exiting the program

## *API đăng nhập Jakarta Commons (JCL)*

Ngoài ra, bạn có thể sử dụng API đăng nhập **Jakarta Commons (JCL)** để tạo nhật ký trong ứng dụng Spring của bạn. JCL có thể được tải xuống từ <https://jakarta.apache.org/commons/logging/> . Tệp duy nhất mà chúng tôi cần về mặt kỹ thuật của gói này là tệp *commons-log-xyzjar* , tệp này cần được đặt trong đường dẫn lớp của bạn theo cách tương tự như bạn đã đặt *log4j-xyzjar* trong ví dụ trên.

Để sử dụng chức năng ghi nhật ký, bạn cần một đối tượng *org.apache.commons.logging.Log* và sau đó bạn có thể gọi một trong các phương thức sau theo yêu cầu của bạn -

* fatal (Thông điệp đối tượng)
* error (thông báo đối tượng)
* warn (Thông điệp đối tượng)
* info (tin nhắn đối tượng)
* debug (Thông điệp đối tượng)
* trace (thông điệp đối tượng)

Sau đây là sự thay thế của MainApp.java, sử dụng API JCL

package com.tutorialspoint;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import org.apache.commons.logging. Log;

import org.apache.commons.logging. LogFactory;

public class MainApp {

static Log log = LogFactory.getLog(MainApp.class.getName());

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

log.info("Going to create HelloWord Obj");

HelloWorld obj = (HelloWorld) context.getBean("helloWorld");

obj.getMessage();

log.info("Exiting the program");

}

}

Bây giờ giữ phần còn lại của cấu hình và nội dung không thay đổi trong ví dụ trên, nếu bạn biên dịch và chạy ứng dụng của mình, bạn sẽ nhận được kết quả tương tự như những gì bạn đã sử dụng API Log4J.