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

***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***

* 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.
* 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.
* 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.
* 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 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.
* 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
* 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.
* 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)

***Mục lục***

*Architecture \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 10*

*Hello World Example \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 13*

*IOC Containers \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 18*

*Bean Definitation \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 22*

*Bean Scope \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 24*

*Bean Life Circle \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 27*

*Bean Post Processors \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 30*

*Bean Definition Inheritance\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 36*

*Injecting Inner Bean \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 39*

*Injecting Collection \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 45*

*Bean Autowiring \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 51*

*Annotation Bassed Configuration \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 58*

*Java Based Configuration \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 64*

*Event Handling \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 71*

*AOP with Spring Framework \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 64*

*JDBC Framework \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 71*

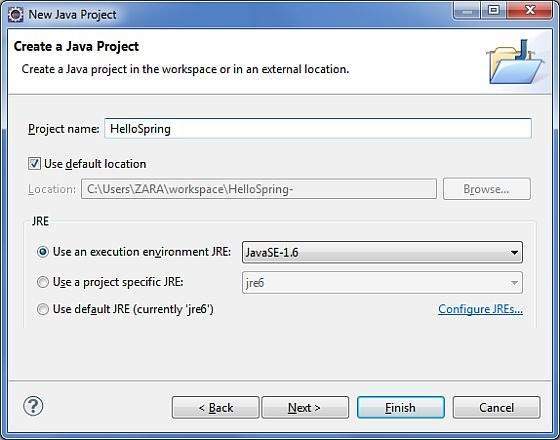
*Transaction Management\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 58*

*Web MVC Framework \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 64*

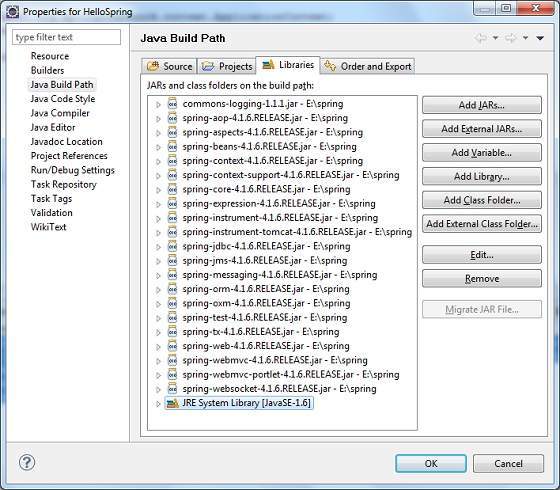
*Logging with Log4J \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 71*

***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>