Series Học Spring core

Contents

Spring – IoC Containers

The Spring-IoC container là core của spring framwork. Container sẽ tạo những object, gắn kết chúng với nhau, cấu hình chúng và quản lí chu trình sống từ lúc khởi tạo cho đến lúc hủy. The Spring container sử dụng dependency injection (DI) để quản lí các component để làm nên 1 ứng dụng .

Spring có 2 kiểu của containers:

1 BeanFactory container

2 ApplicationContext container

1 BeanFactory

Về bản của của một BeanFactory không khác gì chỉ là một interface có khả năng duy trì các phiên bản đăng kí khác nhau của các bean và dependecy của chúng. The BeanFactory cho phép chúng ta read định nghĩa (file xml chẳn hạn) và truy cập bean thông qua sử dụng the bean factory.

1.1 Cách tạo ra một BeanFactory

Khi sử dụng một BeanFactory chúng ta có thể tạo 1 bean và đọc các bean đã định nghĩa trong 1 file có dạng xml:

```
1 bean trong file.xml

<bean id="e" class="com.Group.Employee">
<constructor-arg value="10" type="int"></constructor-arg>
</bean>
```

```
InputStream is = new FileInputStream("beans.xml");
BeanFactory factory = new XmlBeanFactory(is);

//Get bean
HelloWorld obj = (HelloWorld) factory.getBean("helloWorld");
```

Cũng có thể tạo bean factory bằng cách khác như sau :

```
ClassPathResource resource = new ClassPathResource("beans.xml");
BeanFactory factory = new XmlBeanFactory(resource);
//Get bean
HelloWorld obj = (HelloWorld) factory.getBean("helloWorld");
```

Về cơ bản chỉ có thế sử dụng getBean(String) , bạn có thể lấy instance của một bean.

1.2 BeanFactory Method

BeanFactory interfaces chỉ có 6 methods cho client call:

1 boolean containsBean(String) : trả về true nếu beanfactory chứa một bean đã được định nghĩa hoặc 1 instance bean khớp với name string input đã truyền vào .

2 Object getBean(String): trả về 1 instance của 1 bean đã được đăng kí với name truyền vào. Dự vào cách bean đã được cấu hình bằng BeanFactory là 1 singleTon shared instance hoặc 1 bean mới được tạo và trả về . Một BeansException sẽ ném ra trong khi the bean không thể tìm thấy (NoSuchBeanDefinitionException), hoặc một exception xuất hiện trong khi khởi tạo và chuẩn bị bean.

3 Object getBean(String, Class) : trả về 1 bean , đã được đăng kí với name truyền vào. Bean trả về sẽ được chuyển (cast) sang kiểu Class. Nếu bean không thể case 1 exception tương ứng sẽ ném ra (BeanNotOfRequiredTypeException). Hơn nữa tất cả các quy tắc của getBean(String) cũng được ứng dụng vào đây.

4 Class getType(String name) : trả về The Class của the bean với tên đã truyền vào, nếu không có bean nào tương ứng với tên truyền vào một exception NoSuchBeanDefinitionException sẽ được ném ra .

5 boolean isSingleTon(String name): xác định liệu có hay không 1 bean đã được định nghĩa hoặc bean instance được đăng kí với name truyền vào là 1 singleTon. Nếu không có bean tương ứng nào được tìm thấy tương ứng với name sẽ exception ném ra NoSuchBeanDefinitionException .

6 String[] getAliases(String) : Trả về bí danh cho name bean truyền vào nếu tìm thấy bất kỳ bean nào được định nghĩa tương ứng với name truyền vào.

2 ApplicationContext

ApplicationConext container có thêm nhiều chức năng enterprise-specific như là có khả năng giải quyết message từ một file properties và có khả năng lắng nghe nhiều events. Container này được định nghĩa bởi *org.springframework.context.ApplicationContext* interface

The ApplicationContext chứa tất cả các chức năng của BeanFactory container, nên nó được khuyên dùng hơn là BeanFactory. BeanFactory có thể vẫn được sử dụng cho App nhỏ nhẹ giống như mobile devices or

taplet dựa trên ứng dụng mà ở đó quan tâm đến dữ liệu lưu trữ và tốc độ.

2.1 Những kiểu của ApplicationContext

Những implement thường được sử dụng cho ApplicatonContext như sau:

1 FileSystemXmlApplicationContext – this container này loads các định nghĩa của beans từ file xml. ở đó bạn cần phải cung cấp đầy đủ path của file xml để cấu hình contructor.

2 ClassPathXmlApplicationContex – this container này load các định nghĩa của beans từ file.xml . ở đó bạn không cần phải cung cấp đầy đủ đường dẫn của file.xml nhưng bạn cần set Classpath properly bởi vì container sẽ tìm cấu hình bean file xml trong classpath.

3 WebXmlApplicationContext - this container này load file xml với định nghĩa của tất cả các bean từ bên trong một web application.

2.2 Các tạo ra 1 ApplicationContext

Một ví dụ code cho applicationcontext được khởi tạo :

```
ApplicationContext context = new
FileSystemXmlApplicationContext("beans.xml");
HelloWorld obj = (HelloWorld) context.getBean("helloWorld");
```

2 Spring – Inversion of Control vs Dependency Injection

2.1 Inversion Of Control (IoC) là gì

Trong lập trình truyên thống, the flow của bussiness logic được xác định bởi các objects nó được gán tĩnh cho nhau. Với inversion of control , the flow dựa vào đồ thị object nó được tạo ra bởi assember và được thực hiện bởi sự tương tác các object được xác định thông qua sự trừu tượng hóa. Tiến trình binding đạt được thông qua denpendency injection.

Inversion of control phục vụ các mục đích sau :

1 Tách rời việc thực hiện 1 nhiệm vụ nhất định ra khỏi việc triển khai.

2 Mỗi module có thể focus trên những gì có được thiết kế.

3 Những modules không đưa ra giả định về những gì hệ thống khác làm mà chỉ dựa vào sự tương tác của chúng.

4 Thay thế những module không có tác dụng phụ trên các module khác.

2.2 Depency Injection (DI) là gì

IoC là một mô hình thiết kế với mục tiêu là cho nhiều quyền kiểm soát cho các component đích trên ứng dụng của bạn, những thành phần đó sẽ hoàn thành công việc. Trong khi Dependency injection là một pattern được sử dụng để tạo instance của object mà các đối tượng khác dựa vào mà không cần biết tại thời điểm biên dịch class nào sẽ được sử dụng để cung cấp cho chức năng đó. IoC đựa trên depedency injection bởi vì cần có 1 cơ chế để kích hoạt các components cung cấp chức năng cụ thể.

Hai khái niệm hoạt động cùng nhau theo cách này nó sẽ chấp nhận cho việc linh hoạt, tái sử dụng, và đóng gói code, như thế chúng khái niệm qua trọng để thiết kế giải pháp hướng đối tượng

2.3 Cách implement IoC

Trong lập trình hướng đối tượng, có 1 vài cách cơ bản để inplement inversion of control.

1 sử dụng một Factory pattern

2 sử dụng một service locator pattern

3 sử dụng dependency injection của bất kỳ kiểu nào bên dưới:

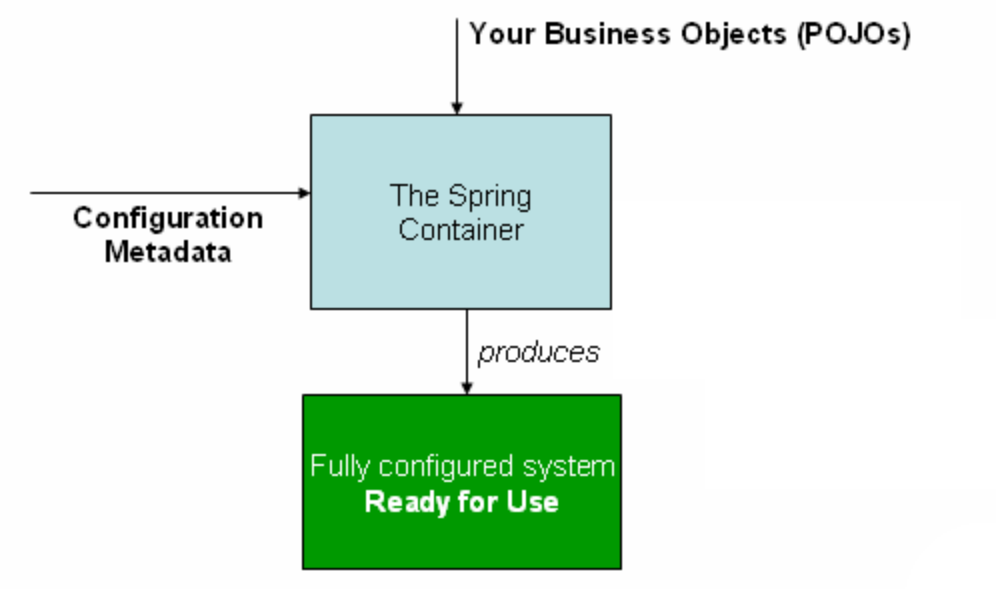
- Một constructor injection
- Một setter injection
- Một interface Injection

2.4 Inversion of Control trong Spring

The org.springframwork.beans và org.springframword.context packges cung cấp cơ bản cho Spring Framwork's IoC container. The BeanFactory interface cung cấp một cơ chế cấu hình nâng cao có thể quản lí các đối tượng có bản chất bất kỳ nào. The ApplicationContext interface xây dựng trên BeanFactory (it là con đó) và có thêm các tính năng khác như tích hợp dễ hơn với Spring' AOP , message resource (sử dung i18n) , event lan truyền, và Tầng ứng dụng cụ thể là WebApplicationContext được sử dụng cho web Application.

The BeanFactory là đai điện thật sự của Spring IoC container no có nhiệm cụ cho việc chứa đựng và nếu không thì quản lí các bean đã nói ở trên

The BeanFactory interface là trung tâm IoC container interface trong Spring .



Ở đó có một số implement của Beanfactory interafce. Implement được sử dụng rỗng rãi của BeanFactory là XmlbeanFactory class. Class khác cũng được sử dụng rỗng rãi là XmlWebApplicationContext. Dựa vào định nghĩa bean, the factory sẽ trả về một instance độc lập của một contained object (The Prototype design pattern) hoặc 1 singleton shared instance (singleton desing pattern). Kiểu của instance sẽ được trả về dựa vào the bean factory configuration.

Trước khi chia kiểu của Denpendency , trước hết hãy xác định cách tạo ra 1 bean trong spring.

2.5 Cách tạo bean trong spring

Một định nghĩa bean có thể xem như là một công thức cho việc tạo một hoặc nhiều object. The container xem xét công thức cho một name bean when được yêu cầu và sử dụng cấu hình metadata được gói bởi đinh nghĩa của bean để create (hoặc yêu cầu) môt object thật sự.

Sử dụng constructor

Khi khởi tạo 1 bean bằng cách sử dụng cách tiếp cận contructor , tất cả các class thông thường đều có thể dc sử dụng và tương thích với Spring. Đó là 1 class được khởi tạo không cần phải implement bất kì 1 interface nào hoặc được viết code 1 cách rõ ràng . Chỉ cần chỉ định class bean là đủ, Khi sử dụng XML-based configuration metadata bạn có thể chỉ định bean giống như thế này

```
<bean id="exampleBean"/>
```

Sử dụng static factory method (static)

Khi định nghĩa 1 bean nó được tạo bằng cách sử dụng 1 static factory method, Cần phải xác định method factory-method trong bean. (type 2 bacsic)

```
<bean id="exampleBean" factory-method="createInstance"/>
```

Spring mong đợi có thể gọi phương thức này và lấy lại 1 object trực tiếp , từ thời điểm đó trở đi coi như thể nó đã được tạo bình thường thông qua 1 hàm contructor.

Sử dụng instance factory method (non-static)

Cách này cũng giống như khởi tạo trực tiếp giống như factory-method nhưng nó sẽ sử dụng bean của 1 thằng đã tồn tại (myFactoryBean) chứ không sử dụng (exampleBean), factory-method này createInstance có thể non-static (type3 trong demo)

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

<bean id="exampleBean" factory-bean="myFactoryBean" factory-
method="createInstance"></bean>
```

2.6 Dependency Injection in Spring

Nguyên tắc cơ bản đằng sau của Dependency Injection (DI) là Những Object đó định nghĩa các dependency của chúng chỉ thông qua constructor arguments, arguments to a faction-method, hoặc các properties Nó được set trên object instance sau khi nó được khởi tạo hoặc trả về từ factory-method. Sau đó nó là công việc của container thật sự inject những dependency đó khi đó nó được khởi tạo đưa vào bean. Về cơ bản đó là nghịch đảo (Inverse), Vì thế nó có tên là inversion of Control

Setter Injection

Setter-based DI được nhận biết bởi cách gọi setter methods trên beans sau khi kêu gọi 1 constructor không đối số hoặc không có đối số static factory-method trong lúc khởi tạo bean của bạn thì sẽ tự gọi setter injection.

```
	public ContructPOJODependency(String name, String nameClass, int maso, int
maso2) {
		super();
		this.name = name;
		this.nameClass = nameClass;
		this.maso = maso;
		this.maso2 = maso2;
	}
```

XML config

```
35
36          <!-- Dependency setter String name, String nameClass, int maso, int maso2 -->
37      <bean id="setterarg" class="basicioc.pojo.ContructPOJODependency">
38          <property name="name">
39              <value>setter name</value>
40          </property>
41          <property name="nameClass">
42              <value>"setter nameClass"</value>
43          </property>
44          <property name="maso">
45              <value>355</value>
46          </property>
47          <property name="maso2">
48              <value>3553</value>
49          </property>
50      </bean>
```

Constructor Injection

Constructor-based DI được nhận biết bởi gọi 1 constructor với một số argument, mỗi cái đại diện cho một Collaborator. Ngoài ra, gọi 1 static factory method với đối số xác định vào contructor của bean, cũng có thể xác định như là tương đương.

```
      public ContructPOJODependency(String name, String nameClass, int maso, int
maso2) {
            super();
            this.name = name;
            this.nameClass = nameClass;
            this.maso = maso;
            this.maso2 = maso2;
      }
```

Cấu hình file xml

```
27
28      <!-- Dependency contruct String name, String nameClass, int maso, int maso2 -->
29      <bean id="constructorarg" class="basicioc.pojo.ContructPOJODependency">
30          <constructor-arg index="0" value="name 0"></constructor-arg>
31          <constructor-arg index="1" value="nameClass 1"></constructor-arg>
32          <constructor-arg index="2" value="500" ></constructor-arg>
33          <constructor-arg index="3" value="300"></constructor-arg>
34      </bean>
35
36  </beans>
```

Interface Injection

Trong các này chúng ta implement một interface từ IOC framwork. IOC framwork sẽ sử dụng interface method để inject the object trong hàm main. Nó thích hợp hơn nhiều để sử dụng cách này khi bạn cần có 1 số

logic ko thể áp dụng để đặt 1 số thuộc tính. Giống như logging support.

```
 public void SetLogger(ILogger logger)

{
  _notificationService.SetLogger(logger);
  _productService.SetLogger(logger);
}
```

3 Spring 5 – Bean scopes

Trong Spring framwork, chúng ta có thể tạo bean trong 6 bean có sẵn spring-bean-scopes và bạn cũng có thể đinhk nghĩa bean của bạn. Trong số 6 spoces trên, chỉ có 4 cái có giái trị nếu bạn sử dụng web-aware ApplicationContext. Singleton và prototype spcopes có giá trị trong bất kì kiểu của IOC container nào.

3.1 Spring Bean Scope Type

Trong Spring, scope có thể được xác định sử dụng @Scope annotation. Đi nhanh xem list 6 bean có sẵn scope của chúng. Phạm vi cũng có thể áp dụng tốt cho spring-boot

Singleton: 1 đối tượng được tao ra trong 1 spring IOC Container (Default)

Prototype: Trái ngược với singleton, nó cung cấp 1 instance mới mỗi khi mỗi lúc một bean được yêu cầu.

Request: Một single instance sẽ được tạo ra và có giá trị trong lúc complete lifecycle của một HTTP request. Chỉ có giá trị trong web-awake Spring ApplicationContext.

Session: Một Single instance sẽ được tạo ra và có giá trị trong lúc complete lifecycle của một HTTP session . chỉ có giá trị trong web-awake Spring ApplicationContext.

Application: Một Single Instance sẽ được tạo ra và có giá trị trong lúc complete lifecycle của ServerletContext. Chỉ áp dụng cho web

Websocket: Một single instance sẽ được tạo ra và có giá trị trong lúc complete lifecycle của Websocket.

3.2 Singleton scope

Singleton là bean scope mặc định trong spring container. Nó sẽ gọi The Container để create và quản lí chỉ 1 instance của bean class trên 1

container. This single instance được lưu trữ trong một cache của singleton beans và tất cả các yêu cầu và reference đến tên bean đó sẽ được lấy instance đó từ cached này.

Example của singleton scope bean using java config

```
 @Component

//This statement is redundant - singleton is default scope
@Scope("singleton")  //This statement is redundant
public class BeanClass {

}
```

```
<!-- To specify singleton scope is redundant -->
<bean id="beanId" class="com.howtodoinjava.BeanClass" scope="singleton" />
//or
<bean id="beanId" class="com.howtodoinjava.BeanClass" />

}
```

3.3 Prototype scope

Prototype scope có kết quả trả về 1 instance mới mỗi khi có yêu cầu bean từ application.

Bạn phải biết hủy khởi tạo không được gọi trong phạm vi prototype scope, chỉ có hàm khởi tạo được gọi nên các developer phải có nhiệm vụ cleanup prototype scope instance và bất kì resouce nào nó nắm giữ.

Java config:

```
 @Component

//This statement is redundant - singleton is default scope
@Scope("prototype")
public class BeanClass {

}
```

XML Config

```
<bean id="beanId" class="com.howtodoinjava.BeanClass" scope="prototype" />
```

Để sử dụng được request, session, application và websocket scope thì phải đăng kí : RequestContextListener or RequestContextFilter

3.4 Request scope

Trong request scope, container sẽ tạo ra 1 instance mới cho mỗi http request, nên nếu server hiện tại có 50 request, thì container có thể có ít nhất 50 instance cá nhân của bean class này. Bất kì trạng thái thay đổi đến từ 1 instance sẽ không visible đến các instance khác. Những instance đó sẽ sớm bị hủy lúc request được complete.

Java config

```
 @Component

@Scope("request")
public class BeanClass {
}

//or

@Component
@RequestScope
public class BeanClass {
}
```

```
<bean id="beanId" class="com.howtodoinjava.BeanClass" scope="request" />
```

3.5 Session Scope

Trong session scope, container sẽ tạo ra 1 instance cho mỗi HTTP session. Nên nếu server có 20 active sessions, thì container có có thể có ít nhất 20 instance riêng lẽ của bean class session. Tất cả các http request trong cùng 1 sesion lifetime sẽ có thể truy cập cùng 1 single bean trong 1 session scope.

Bất kì thay đổi trạng thái nào của 1 instance cũng ko thể ảnh hưởng đến các instacen khác. Những instance này sẽ bị hủy nếu session bị destroy/end trên server.

Java config

```
 @Component

@Scope("session")
public class BeanClass {
}

//or

@Component
@SessionScope
public class BeanClass {
}
```

Xml Config

```
<bean id="beanId" class="com.howtodoinjava.BeanClass" scope="session" />
```

3.6 Websocket scope

WebSocket protocol cho phép nói chuyện 2 đầu giữa client và remote host cũng như là server với client . Websocket protocol provides a single TCP connection cho 2 hướng. Đều đó hữu ít cho ứng dụng nhiều user realtime multi-user game

Trong kiểu này ứng dụng web, http chỉ được sử dụng để bắt tay chào hỏi giữa server và client và server có thể respond với http status 101 (switching protocol) nếu đồng ý thì gửi 1 request bắt tay, nếu thành công thì trò chuyện giữa client và server ok .

Java config

```
 @Component

@Scope("websocket")
public class BeanClass {
}
```

Xml config

```
<bean id="beanId" class="com.howtodoinjava.BeanClass" scope="websocket" />
```

Hãy note lại là websocket là kiểu singleton và có live longer than any websocket session .

3.7 Custome thread scope

Spring cũng cung cấp một non-default thread scope using class SimpleThreadScope. Để sử dụng thí scope, bạn phải use register it container using CustomeScopeConfigurer class.

```
<bean
class="org.springframework.beans.factory.config.CustomScopeConfigurer">
    <property name="scopes">
        <map>
            <entry key="thread">
                <bean
class="org.springframework.context.support.SimpleThreadScope"/>
            </entry>
        </map>
    </property>
</bean>
```

Mỗi request cho một bean sẽ return 1 instance bên trong 1 thread cùng tên

Java Config

```
@Component
@Scope("thread")
public class BeanClass {
}
```

Xml Config

```
<bean id="beanId" class="com.howtodoinjava.BeanClass" scope="thread" />
```

4 Spring – Bean Life Cycle

Trong phần này, học về spring bean life cycle. Chúng ta sẽ học về life cycle gia đoạn , khởi tạo và destroy bằng cách gọi hàm. Học cách control bean life cycle events sử dụng bằng cấu hình xml cũng như là annotaion.

4.1 Bean lifecycle

Khi container được start – một spring bean cần phải được khởi tạo, dựa vào Java hoặc xml định nghĩa bean. Nó cũng có thể được yêu cầu thực hiện 1 số bước khởi tạo để đưa nó vào trạng thái có thể dc sử dụng. Life cycle cũng áp dụng tốt cho spring boot.

Sau đó, khi bean không cần thiết nữa , nó sẽ remove ra khỏi container .

Spring bean factory có nhiệm vụ để quản lí life cycle của các bean được khởi tạo thông qua sping container.

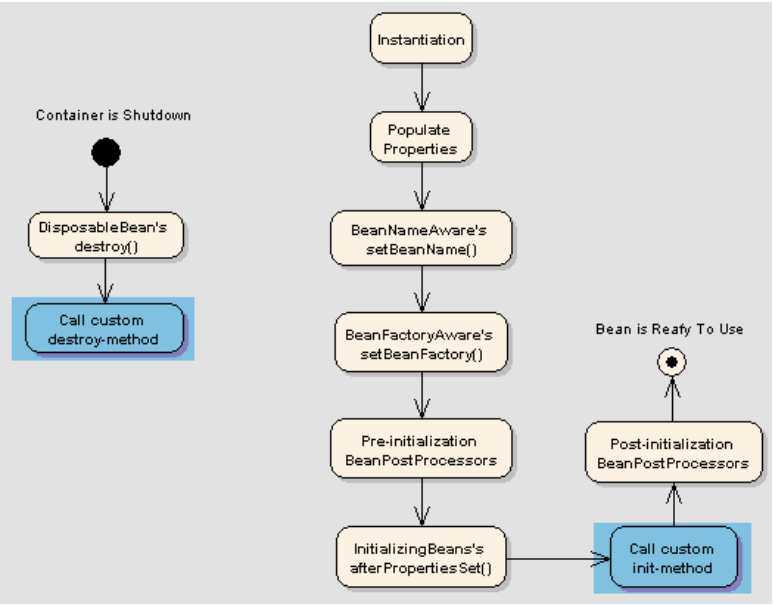
4.1.2 Life cycle callbacks

Spring bean factory điều kiển khởi tạo và hủy của các bean. Để thực thi 1 vài custom code, nó cung cấp the callback method có thể được phân loại rỗng rãi trong ra 2 nhóm :

Post-initialization callback method

Per-destructor callback method

4.1.3 Life cycle in Diagram



4.2 Life cycle callback method

Spring famework cung cấp 4 cách cho việc điều kiển life cycle events của bean:

1 InitialzingBean và DisposableBean callback interface

2 *Aware interface cho các hành vi cụ thể

3 Custom init() và destroy() methods trong bean cấu hình file

4 @PostContruct và @PreDestroy annotations.

4.2.1 InitializingBean và DisposableBean

The org.springframework.beans.factory.initialzingBean interface chấp nhận 1 bean thực hiện việc khởi tạo sau khi tất cả các properties cần thiết được set bởi the container

The InitializtingBean interfaces chỉ có 1 method thôi.

Đây không phải là cách thích hợp cho việc khởi tạo bean bởi vì nó kết hợp chặc chẽ bean của bạn với spring container. Cách tiếp cận tốt hơn là sử dụng "init-method" attribute trong bean định nghĩa trong file xml configuration.

Tương tự, triển khai org.springframwork.beans.factory.DisposableBean linterface chấp nhận 1 bean gọi 1 callback khi the container chứa nó bị hủy.

The DisposableBean interface chỉ có 1 method destroy()

```
public class DemoBean implements InitializingBean, DisposableBean
{
    //Other bean attributes and methods

    @Override
    public void afterPropertiesSet() throws Exception
    {
        //Bean initialization code
    }

    @Override
    public void destroy() throws Exception
    {
        //Bean destruction code
    }
}
```

4.2.2 Aware interfaces for specific behavior

Spring đề nghị 1 dãy interface *Aware để chấp nhận chỉ định container rằng họ yêu cầu 1 cơ sở hạ tầng dependency chắn chắn. Mỗi interface sẽ yêu cầu bạn implement một method để inject the dependency trong bean .

```
public class DemoBean implements ApplicationContextAware,
        ApplicationEventPublisherAware, BeanClassLoaderAware,
BeanFactoryAware,
        BeanNameAware, LoadTimeWeaverAware, MessageSourceAware,
        NotificationPublisherAware, ResourceLoaderAware
{
    @Override
    public void setResourceLoader(ResourceLoader arg0) {
        // TODO Auto-generated method stub
    }

    @Override
    public void setNotificationPublisher(NotificationPublisher arg0) {
        // TODO Auto-generated method stub

    }

    @Override
    public void setMessageSource(MessageSource arg0) {
        // TODO Auto-generated method stub
    }

    @Override
    public void setLoadTimeWeaver(LoadTimeWeaver arg0) {
        // TODO Auto-generated method stub
    }

    @Override
    public void setBeanName(String arg0) {
        // TODO Auto-generated method stub
    }

    @Override
    public void setBeanFactory(BeanFactory arg0) throws BeansException {
        // TODO Auto-generated method stub
    }

    @Override
    public void setBeanClassLoader(ClassLoader arg0) {
        // TODO Auto-generated method stub
    }

    @Override
    public void setApplicationEventPublisher(ApplicationEventPublisher
arg0) {
        // TODO Auto-generated method stub
    }

    @Override
    public void setApplicationContext(ApplicationContext arg0)
            throws BeansException {
        // TODO Auto-generated method stub
    }
}
```

4.2.3 Custome init() va destroy() methods

Mặc định init và destroy methods trong cấu hình bean file có thể được định nghĩa 2 cách :

Bean local definition áp dụng cho 1 single bean

Global definition áp dụng cho tất cả các bean đã định nghĩa trong beans context.

Bean local definition (riêng lẽ từng bean)

```
<beans>

    <bean id="demoBean" class="com.howtodoinjava.task.DemoBean"
                    init-method="customInit"
                    destroy-method="customDestroy"></bean>

</beans>
```

Global definition (các bean con được gọi cùng tên hàm đó)

Ở dưới là biểu của Global definition . Những method đó sẽ được gọi cho tất cả các bean definition trong tag <beans> (chứa các bean có tag <bean>). Nó hưu ích cho việc đặt tên cho các định nghĩa chung như tên hàm init() và destroy() cho tất cả các bean nhất quán. Có nghĩa là trong các bean trong tag ,<beans>

Đều có tên customInit() cho việc khởi tạo và customDestroy()ta chỉ cần custome thôi không cần phải quan tâm tên nữa nó là nhất quán .

```
<beans default-init-method="customInit" default-destroy-
method="customDestroy">

        <bean id="demoBean"
      class="com.howtodoinjava.task.DemoBean">
            </bean>

</beans>
```

Trong java

```
public class DemoBean
{
    public void customInit()
    {
        System.out.println("Method customInit() invoked...");
    }

    public void customDestroy()
    {
        System.out.println("Method customDestroy() invoked...");
    }
}
```

4.2.4 @PostContruct và @PreDestroy

Từ spting 2.5 , bạn có thể sử dụng annotation để chỉnh định life cycle sử dụng @PostContruct và @PreDestroy annotation

@PostContruct annotatied method sẽ được gọi sau khi the bean đã khởi tạo sử dụng contruct default và trước lúc instance được trả về cho yêu cầu

@PerDetroy annoted method được gọi trước khi the bean bị detroy bên trên bean container .

```
public class DemoBean
{
    @PostConstruct
    public void customInit()
    {
        System.out.println("Method customInit() invoked...");
    }

    @PreDestroy
    public void customDestroy()
    {
        System.out.println("Method customDestroy() invoked...");
    }
}
```

5 Spring BeanPostProcessor Example

Một bean post processor chấp nhận custom modifier của new instance được tạo từ spring bean factory. Nếu bạn muốn triển khai một vài custom logic sau khi Spring container hoàn thành việc khởi tạo, cấu hình bean. Chúng ta có thể cắm 1 hoặc nhiều BeanPostProcessor implement.

Trong trường hợp nhiều BeanPostProcessor instance, chúng ta có thể sắp xếp chúng thông qua setting order property hoặc implement Ordered interface.

BeanPostProcessor interface có 2 hàm

postProcessorBeforInitialization() và
postProcessorAfterInitialization()

mỗi bean intances được tạo bởi container, the post-processor nhận được 1 gọi hàm từ container bao gồm lúc trước khi container gọi hàm khơi tạo và sau khi khơi tạo bean

5.1 How To Create BeanPostProcessor

Để khởi tạo một bean post processor trong spring:

Implement the BeanPostProcessor interface

Implemetn the callback methods

```
public class CustomBeanPostProcessor implements BeanPostProcessor
{
    public Object postProcessBeforeInitialization(Object bean, String beanName)
throws BeansException
    {
        System.out.println("Called postProcessBeforeInitialization() for :" +
beanName);
        return bean;
    }

    public Object postProcessAfterInitialization(Object bean, String beanName)
throws BeansException
    {
        System.out.println("Called postProcessAfterInitialization() for :" +
beanName);
        return bean;
    }
}
```

5.2 Cách đăng kí BeanPostProcessor

Một ApplicationContext tự động phát hiện bất kì bean nào được định nghĩa configuration metadata bằng cách implement BeanPostProcessor interface. Nó sẽ đăng kí những bean này như là post-processor và sẽ được gọi khi tạo 1 bean mới.

```
 <beans>

     <bean id="customBeanPostProcessor"
               class="com.howtodoinjava.demo.processors.CustomBeanPostProcessor" />
</beans>
```

5.3 Khi nào BeanPostProcessor methods được gọi

Thông thường Spring' DI container sẽ làm những bước sau để tạo 1 bean, khi bạn yêu cầu:

1 tạo bean instance bằng 1 hàm contructor hoặc bằng 1 factory method

2 set những value và những bean reference vào trong bean properties

3 gọi setter methods đã định nghĩa trong tất cả the aware interfaces

4 Pass the instance vào postProcessorBeforeInitialization() method cho mỗi bean post processor

5 Gọi hàm initialization callback method

6 Pass the instance to the postProcessorAfterInitialization() method cho mỗi bean post processor

7 the bean sẵn sàn dc sử dụng

8 Khi container được shutdown, call the destroy callback method

5.4 Example

Sử dụng class EmployeeImpl:

```
public class EmployeeDAOImpl implements EmployeeDAO
{
    public EmployeeDTO createNewEmployee()
    {
        EmployeeDTO e = new EmployeeDTO();
        e.setId(1);
        e.setFirstName("Lokesh");
        e.setLastName("Gupta");
        return e;
    }

    public void initBean() {
        System.out.println("Init Bean for : EmployeeDAOImpl");
    }

    public void destroyBean() {
        System.out.println("Init Bean for : EmployeeDAOImpl");
    }
}
```

Cấu hình trong file xml

```
  <bean id="customBeanPostProcessor"
class="com.howtodoinjava.demo.processors.CustomBeanPostProcessor" />


<bean id="dao" class="com.howtodoinjava.demo.dao.EmployeeDAOImpl" init-
method="initBean" destroy-method="destroyBean"/>
```

Demo

```
 ApplicationContext context = new ClassPathXmlApplicationContext("beans.xml");

---

Called postProcessBeforeInitialization() for : dao
Init Bean for : EmployeeDAOImpl
Called postProcessAfterInitialization() for : dao
```

GitHub:

https://github.com/nguyenthinhit996/sharefullcode/tree/spring/spring/core/beanproject

Referncen:

https://howtodoinjava.com/