**Bean 的生命周期**

项目地址：https://github.com/xuegang20/mystudyrecode/tree/main/code/SpringExample

理解 Spring bean 的生命周期很容易。当一个 bean 被实例化时，它可能需要执行一些初始化使它转换成可用状态。同样，当 bean 不再需要，并且从容器中移除时，可能需要做一些清除工作。

尽管还有一些在 Bean 实例化和销毁之间发生的活动，但是本章将只讨论两个重要的生命周期回调方法，它们在 bean 的初始化和销毁的时候是必需的。

为了定义安装和拆卸一个 bean，我们只要声明带有 **init-method** 和/或 **destroy-method** 参数的 。init-method 属性指定一个方法，在实例化 bean 时，立即调用该方法。同样，destroy-method 指定一个方法，只有从容器中移除 bean 之后，才能调用该方法。

Bean的生命周期可以表达为：Bean的定义——Bean的初始化——Bean的使用——Bean的销毁

**初始化回调**

*org.springframework.beans.factory.InitializingBean* 接口指定一个单一的方法：

void afterPropertiesSet() throws Exception;

因此，你可以简单地实现上述接口和初始化工作可以在 afterPropertiesSet() 方法中执行，如下所示：

public class ExampleBean implements InitializingBean {

public void afterPropertiesSet() {

// do some initialization work

}

}

在基于 XML 的配置元数据的情况下，你可以使用 **init-method** 属性来指定带有 void 无参数方法的名称。例如：

<bean id="exampleBean"

class="examples.ExampleBean" init-method="init"/>

下面是类的定义：

public class ExampleBean {

public void init() {

// do some initialization work

}

}

**销毁回调**

*org.springframework.beans.factory.DisposableBean* 接口指定一个单一的方法：

void destroy() throws Exception;

因此，你可以简单地实现上述接口并且结束工作可以在 destroy() 方法中执行，如下所示：

public class ExampleBean implements DisposableBean {

public void destroy() {

// do some destruction work

}

}

在基于 XML 的配置元数据的情况下，你可以使用 **destroy-method** 属性来指定带有 void 无参数方法的名称。例如：

<bean id="exampleBean"

class="examples.ExampleBean" destroy-method="destroy"/>

下面是类的定义：

public class ExampleBean {

public void destroy() {

// do some destruction work

}

}

如果你在非 web 应用程序环境中使用 Spring 的 IoC 容器；例如在丰富的客户端桌面环境中；那么在 JVM 中你要注册关闭 hook。这样做可以确保正常关闭，为了让所有的资源都被释放，可以在单个 beans 上调用 destroy 方法。

建议你不要使用 InitializingBean 或者 DisposableBean 的回调方法，因为 XML 配置在命名方法上提供了极大的灵活性。

**例子**

这里是 **HelloWorld.java** 的文件的内容：

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.");

}

}

下面是 **MainApp.java** 文件的内容。在这里，你需要注册一个在 AbstractApplicationContext 类中声明的关闭 hook 的 **registerShutdownHook()** 方法。它将确保正常关闭，并且调用相关的 destroy 方法。

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();

}

}

下面是 init 和 destroy 方法必需的配置文件 **Beans.xml** 文件：

<?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>

一旦你创建源代码和 bean 配置文件完成后，我们就可以运行该应用程序。如果你的应用程序一切都正常，将输出以下信息：

Bean is going through init.

Your Message : Hello World!

Bean will destroy now.

**默认的初始化和销毁方法**

如果你有太多具有相同名称的初始化或者销毁方法的 Bean，那么你不需要在每一个 bean 上声明**初始化方法**和**销毁方法**。框架使用 元素中的 **default-init-method** 和 **default-destroy-method** 属性提供了灵活地配置这种情况，如下所示：

<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>