Spring Bean LifeCycle

Spring Bean LifeCycle

The Spring Bean lifecycle involves several steps including bean initialization and bean destroy. These steps are managed by the Spring IOC and it lets us perform custom initializing and end up tasks.

In Spring, if we want to perform some tasks at the time of bean initialization and before destroying the bean object then we can use Spring's predefined interfaces InitializingBean and DisposableBean. These interfaces provide methods that can be used to perform tasks before and after creating the bean. Java provides annotations too to work with the Bean lifecycle.

There are two ways to perform these tasks:

- **Interfaces**
- **Annotations**

The Spring InitializingBean interface provides a method afterPropertiesSet() that can be used to perform initializing tasks while the DisposableBean interface provides a method destroy() to perform cleaning resources before destroying bean objects.

Spring Bean LifeCycle

Bean Lifecycle using Interfaces

Here, we are using these interfaces in our project. Our project is a maven project and contains the below files with code.

// Manager.java

It is our bean class that will be used to perform implementations. It implements the Employee interface and implements doWork() method.

```
import org.springframework.beans.factory.DisposableBean;
import org.springframework.beans.factory.InitializingBean;
import org.springframework.stereotype.Component;

@Component
public class Manager implements InitializingBean, DisposableBean, Employee{

	@Override
	public void afterPropertiesSet() throws Exception {

		System.out.println("Perform tasks while initializing Bean");

	}
	@Override
	public void destroy() throws Exception {

		System.out.println("Perform tasks before destroying of Bean");

	}
	@Override
	public void doWork() {
		System.out.println("Manage branch office");
	}
}
```

Spring Bean LifeCycle

// Employee.java

It is an interface that contains an abstract method doWork() which will be overridden by the implemented class.

```
package com.studytonight.community;
public interface Employee
{
        void doWork();
}
```

// BankApp.java

It is a configuration file that reads the applicationContext file and get Bean using the getBean() method and then call method based on the retrieved object.

```
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
public class BankApp
 {

        public static void main(String[] args)
        {

                AnnotationConfigApplicationContext context = new
AnnotationConfigApplicationContext(AppConfig.class);
                Employee employee = (Employee) context.getBean("manager");
                employee.doWork();
                context.close();
        }
}
```

Spring Bean LifeCycle

// AppConfig.java

```
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan("com.studytonight.community")
public class AppConfig
{

}
```

// pom.xml

This file contains all the dependencies of this project such as spring jars, servlet jars, etc. Put these dependencies into your project to run the application.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.studytonight</groupId>
  <artifactId>springApp</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <dependencies>
        <!-- https://mvnrepository.com/artifact/org.springframework/spring-web -->
                <dependency>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-core</artifactId>
                        <version>${spring.version}</version>
                </dependency>
                <dependency>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-context</artifactId>
```

Spring Bean LifeCycle

```
                <version>${spring.version}</version>
            </dependency>
            <dependency>
                <groupId>javax.annotation</groupId>
                <artifactId>javax.annotation-api</artifactId>
                <version>1.3.2</version>
            </dependency>
        </dependencies>
        <properties>
            <spring.version>5.2.8.RELEASE</spring.version>
        </properties>
        <build>
            <sourceDirectory>src</sourceDirectory>
            <plugins>
                <plugin>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <version>3.8.1</version>
                    <configuration>
                        <source>1.8</source>
                        <target>1.8</target>
                    </configuration>
                </plugin>
            </plugins>

        </build>
</project>
```

Spring Bean LifeCycle

Run the Application

After successfully completing the project and adding the dependencies run the application and you will get the output as below.

```
Perform tasks while initializing Bean

Manage branch office

Perform tasks before destroying of Bean
```

Spring Annotations: Another Approach

Spring provides two annotations: @PostConstruct and @PreDestroy to perform initialization and end up tasks. In this case, we don't need to use interfaces and their methods. This approach is pretty easy and recommended.

Note: For Java 9 and higher, We need to add some extra JARs in our project because of javax.annotation package has been removed from its default classpath. So, add the following JARs into the pm.xml file.

```
<dependency>
        <groupId>javax.annotation</groupId>
        <artifactId>javax.annotation-api</artifactId>
        <version>1.3.2</version>
</dependency>
```

Spring Bean LifeCycle

Lifecycle using Spring Annotations

While working with annotations, no methods are provided then we can use our own methods having annotations. See, the initWork() method and predestroy() method are own created and we used annotations on both to execute them with Bean lifecycle. You can replace the above example code (manager.java) with this code and it will produce the same result.

```
import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;
import org.springframework.stereotype.Component;

@Component
public class Manager implements Employee{

	public void doWork()
	 {
		System.out.println("Manage the branch office");

	}
	@PostConstruct
	public void initWork()
	 {

		System.out.println("Perform tasks while initializing Bean");

	}
	@PreDestroy
	public void Predestroy()
	{

		System.out.println("Perform tasks before destroying of Bean");
	}
}
```

Copy

Spring Bean LifeCycle

Run the Application

After successfully completing the project and adding the dependencies run the application and you will get the output as below.

```
Perform tasks while initializing Bean

Manage branch office

Perform tasks before destroying of Bean
```