Spring framework 5 Life-cycle

Aware interfaces

Permiten recibir eventos de Spring framework, las interfaces disponibles son las siguientes:

- -BeanNameAware
- -BeanFactoryAware
- -ApplicationContextAware
- -MessageSourceAware
- -ApplicationeventPublisherAware
- -ResourceLoaderAware
- -EnvironmentAware

BeanNameAware

A continuación se presenta un ejemplo sobre el uso de la interfaz **BeanNameAware:**

```
@Component
public class MyBeanNameAware implements
BeanNameAware {

private static final Logger log =
LoggerFactory.getLogger(MyBeanNameAware.class);

    @Override
    public void setBeanName(String name) {
        log.info("Bean name {}", name);
    }

}
```

@PostConstruct / @PreDestroy callbacks

Las callbacks **@PostConstruct** y **@PreDestroy** se ejecutan una vez que se construyó el bean y antes de que éste sea destruido, siguen las siguientes reglas:

- -Los métodos a los que se aplica pueden tener cualquier modificador de acceso pero no deben de recibir argumentos y su valor de retorno debe ser **void**.
- -Éstas anotaciones no son propias de Spring, están definidas en el **JSR-250** también utilizado en EJB.
- -Si es un bean creado de forma explícita se pueden definir del siguiente modo **@Bean(initMethod**="init", **destroyMethod**="destroy").
- -Los métodos anotados con **@PostConstruct** se ejecutarán después de la inyección de dependencias.
- -Los métodos anotados con **@PreDestroy** se ejecutarán antes de que el bean sea destruido.
- -Los métodos anotados con **@PreDestroy** no se ejecutarán para beans **prototype**.
- -Los métodos anotados con **@PreDestroy** solo se ejecutan si se termina la JVM de forma normal.

InitializingBean / DisposableBean

Las interfaces **InitializingBean** y **DisposableBean** funcionan del mismo modo que **@PostConstruct** y **@PreDestroy,** permitiendo ejecutar algunas tareas de inicialización y liberación de recursos, están definidas del siguiente modo:

```
public interface InitializingBean {
    void afterPropertiesSet() throws Exception;
}
public interface DisposableBean {
    void destroy() throws Exception;
}
```

Inicialización

Existen 2 formas de inicializar un bean por Spring:

- -**Eager**: Los beans de tipo eager son inicializados sin importar si se utilizarán o no dentro de la aplicación.
- -**Lazy**: Los beans de tipo lazy se crean hasta el momento en el que se utilizarán por primera vez.

Eager / Lazy

Los beans **singleton** son por default **eager**, los beans **prototype** son por default **lazy**.

Si se desea definir un bean singleton como lazy se utiliza la anotación **@Lazy.**

NOTA: Si un bean singleton es **lazy** pero otro bean que depende de el no lo es, ambos serán considerados **eager**.

Definición de callbacks con beans declarados de forma explícita

Para los beans declarados de forma explícita, es posible definir las callbacks del siguiente modo:

```
@Configuration
public class BeanConfig {
    @Bean(initMethod="init",destroyMethod="destroy")
    public Circle getCircle() {
        return new Bean();
    }
}
```

La anotación **@Bean** define los atributos **initMethod** y **destroyMethod** los cuales representan el nombre de la callback de inicialización y de destrucción.

BeanPostProcessor

La interfaz **BeanPostProcessor** define los siguientes métodos:

```
public interface BeanPostProcessor {
    @Nullable
    default Object postProcessBeforeInitialization(Object
bean, String beanName) throws BeansException {
        return bean;
    }
    @Nullable
    default Object postProcessAfterInitialization(Object
bean, String beanName) throws BeansException {
        return bean;
    }
}
```

Es posible implementarlos para definir una forma de inicialización propia, a continuación se describe su comportamiento:

- **postProcessBeforeInitialization** : Permite ejecutar lógica de inicialización **antes** de que se inicialice el bean por el contenedor de spring.
- **postProcessAfterInitialization** : Permite ejecutar lógica de inicialización **después** de que el bean fue inicializado por el contenedor de spring.

DEVS4J
Hecho por Devs para Devs

BeanFactoryPostProcessor

La interfaz **BeanFactoryPostProcessor** define el siguiente método:

```
public interface BeanFactoryPostProcessor {

    void postProcessBeanFactory
    (ConfigurableListableBeanFactory beanFactory)
    throws BeansException;
}
```

Permite modificar la **definición** de los beans, pero nunca modificará las instancias.

Ciclo de vida

Una vez que los eventos anteriores se completaron de forma existosa, el bean se encontrará listo para ser utilizado. Es importante considerar durante el ciclo de vida, factores como los siguientes:

- -Si existen ***aware interfaces*** definidas
- -Si el bean se construirá de modo eager / lazy
- - El scope de los beans a construir(Recordemos que beans prototype no ejecutan los métodos para destrucción)

Orden de ejecución

Como vemos, es posible tener tanto beans anotados con **@PostConstruct** y **@PreDestroy,** como beans que implementan las interfaces **InitializingBean** y **DisposableBean,** a continuación se muestra el orden de ejecución de los mismos:

- -Se ejecutan las **AwareInterfaces**
- -Se ejecutan los métodos **postProcessBeforeInitialization** de los beans que implementen la interfaz **BeanPostProcessor**
- -Se ejecutan los beans anotados con **@PostConstruct**
- -Se ejecutan los métodos ***afterProperties*** de los beans que implementan la interfaz **InitializingBean.**
- -Se ejecutan los métodos **postProcessAfterInitialization** de los beans que implementan la interfaz **BeanPostProcessor**
- -Se ejecutan los métodos anotados con **@PreDestroy**
- -Se ejecutan los métodos ***destroy*** de los beans que implementan la interfaz **DisposableBean**

A lo anterior lo conoceremos como el ciclo de vida de la fase de inicialización.