Elías Calderón - Kevin León - Josué León

Análisis del Application Domain: Contenedor de Inversion of Control NAIoCC

Meta:

Se desarrollará un contenedor simple que maneje el control de inversión, como Spring, con todas las funcionalidades principales que tiene.

Ofrecerá la inyección de dependencias mediante métodos “set” o vía un constructor.

El alcance de los objetos del contenedor podrá ser Singleton o Prototype.

Se manejará el ciclo de vida de los objetos mediante métodos de inicialización y destrucción.

Además, tendrá la posibilidad de inyectar dependencias automáticamente mediante el nombre o tipo de los atributos o el constructor del objeto (autowiring).

Finalmente, será disponible configurar el contenedor mediante tres distintos formatos: XML, Annotations y programáticamente.

Usuarios:

Los usuarios del contenedor pueden ser programadores con conocimiento en Java, específicamente que tengan alguna idea acerca del uso de archivos XML y el uso de anotaciones, ya que son necesarios para utilizar el contenedor.

Para utilizar el contenedor el usuario debe definir los POJO’s (Plain Old Java Objects) y la información que permita la generación de los mismos (metadata). Los POJO’s los puede definir mediante el uso de objetos de Java y la metadata puede estar conformada por el uso del XML, anotaciones y formato programático. Una vez establecidos estos elementos, debe crear una instancia de la fábrica para su funcionamiento.

Formato del XML:

Bean engloba la estructura del árbol para los nodos con tag bean.

-Obligatorio.

-Contiene los atributos **id**, **class**, **scope**, **init**, **destroy**, **lazy-generation**, **autowire**.

Id es el identificador único para cada bean.

-Obligatorio.

-Su valor no se puede repetir en otros beans.

Class identifica la ruta para la clase del bean

-Obligatorio.

-Su valor se puede repetir en distintos beans.

Scope identifica el alcance del bean

-Puede ser "**Singleton**" (Paralelo a Singleton) o "**Prototype**" (Paralelo a Prototype).

-No es obligatorio, y por defecto es "**Singleton**".

-Su valor se puede repetir en distintos beans.

Init determina el método de inicialización a llamar cuando se crea el **bean**.

-Análogo a Init-Method.

-No es obligatorio.

-Su valor se puede repetir en distintos beans.

-En el XML se puede definir un método de **init** default para todos los beans poniendo un atributo **init** en **beans**, pero si se especifica en un bean el atributo init entonces se le hace Override al default.

Destroy determina el método de destrucción a llamar cuando se destruye el bean.

-Análogo a Destroy-Method.

-No es obligatorio.

-Su valor se puede repetir en distintos beans.

-En el XML se puede definir globalmente en **beans** para todos, se comporta igual que init.

Lazy-generation determina si el bean se genera a la hora de recuperar el bean o cuando se crea el contenedor.

-No es obligatorio, su valor por defecto es **false**.

-Su valor se puede repetir en distintos beans.

-Si el valor es **true** se genera hasta que se recupera el bean.

-Si el valor es **false** se genera cuando se instancia el contenedor.

Autowire especifica la manera automática de inyectar las dependencias de un bean.

-No es obligatorio.

-Su valor se puede repetir en distintos beans.

-El valor puede ser "**byName**", "**byType**", "**constructor**" o "**none**". El valor por defecto es "**none**", que equivale a no especificar el autowire del todo.

-Se puede haver Override, al específicar un **attribute** o un **param** del constructor.

Attribute especifica los atributos del objeto, para que sea posible la inyección de dependencias mediante los métodos de "set".

-Análogo a property.

-No es obligatorio.

-Puede haber más de un attribute en un mismo bean.

-Tiene **name**.

-Tiene **value** o **ref**

Name identifica el nombre del atributo a inyectar

-Es obligatorio.

-Su valor no se puede repetir en atributos del mismo bean

Constructor en XML engloba las definiciones de parámetros del constructor designado para la inyección.

-No es obligatorio.

-Es único para un bean, solo puede haber un tag de constructor en un bean.

Param identifica un argumento de un constructor.

-Análogo a constructor-arg

-No es obligatorio

-Tiene **type** o **index**.

-Tiene **value** o **ref**.

Type identifica el tipo de un parámetro del constructor a hacer inyección.

-Es obligatorio, pero se puede poner **index** en cambio, o los dos.

-El valor es único.

Index identifica el índice de un parámetro del constructor a hacer inyección.

-Es obligatorio, pero se puede poner type en cambio, o los dos.

-El valor es único.

Value identifica el valor de un atributo o un argumento.

-Es obligatorio, pero se puede poner **ref** en cambio.

-El valor no es único.

Value-type identifica el tipo de un atributo.

-Es obligatorio si se usa value en un attribute.

-El valor no es único.

Ref hace referencia a un bean Id declarado en algún lugar del xml

-Es obligatorio, pero se puede poner **value** en cambio.

-La referencia no es única.

Estructura del XML:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<beans init="defaultInitMethod" destroy="defaultDestroyMethod">

	<bean  id = "beanId" class = "package.path.class"
		scope="Singleton/Prototype"
		init="methodName" destroy="methodName"
		lazy-generation="true/false"
		autowire="byName/byType/none">

		<constructor>

			<param type="package.path.class"
				index="numberIndex"
				value="valor"/ref="beanId" />

		</constructor>

		<attribute name="nombreAtr" value="valor"/ref="beanId" />

	</bean>

</beans>
```

Formato de las Annotations Nuestro:

@Bean

Le indica al contenedor que la clase anotada con el **@Bean** se registra como un bean en el contenedor. El beanId que se le pone al Bean es el nombre de la clase, pero con la primera letra en minúscula. Se pone arriba de la clase.

Estructura

```
@Bean
public class BeanClass {

	...
}
```

Seria equivalente a:

```
<beans>
	<bean id = "beanClass" class = "package.path.BeanClass" />
</beans>
```

@Scope:

Identifica el alcance del bean. Por parámetro se le debe de mandar si es **Singleton** o **Prototype**. Se pone arriba de la clase.

Estructura:

```
@Bean
@Scope("Singleton")/@Scope("Prototype")
public class BeanClass {
	...
}
```

Seria equivalente a:

```
<beans>
	<bean id = "beanId" class = "package.path.class" scope="Singleton/Prototype">
	</bean>
</beans>
```

@Init / @Destroy

-**@Init** : Se pone arriba del método, determina el método de inicialización a llamar cuando se crea el **bean**.

-**@Destroy**: Se pone arriba del método, determina el método de destrucción a llamar cuando se destruye el **bean**.

Estructura:

```
@Bean
public class BeanClass {

	@Init
	public void initMethod() {
		…
	}

	@Destroy
	public void destroyMethod() {
		…
	}
}
```

Seria equivalente a:

```
<beans>
	<bean  id = "beanId" class = "package.path.class"
		init="initMethod"
		destroy="destroyMethod" >
	</bean>
</beans>
```

@Lazy:

Determina si el bean se genera a la hora de recuperar el bean o cuando se crea el contenedor.

Estructura:

```
@Bean
@Lazy
public class BeanClass {
	...
}
```

Seria equivalente a:

```
<beans>
	<bean  id = "beanId" class = "package.path.class"
		lazy-generation="true/false"
	</bean>
</beans>
```

@Autowire:
Especifica la manera automática de inyectar las dependencias de un bean.

Estructura:

```
@Bean
@Autowire(“byType”/“byName”)
public class BeanClass {
	...
}
```

Seria equivalente a:

```
<beans>
	<bean  id = "beanId" class = "package.path.class"
		autowire="byName/byType/none">
	</bean>
</beans>
```

@Attribute:
Se pone arriba de un atributo para especificar que va a ser una propiedad del bean, debe tener un setter asociado a este.
Este annotation tiene un parámetro, en donde se le especifica el **value** o **reference**.
Si se llama a un valor explícito se pone:

-**@Attribute(“val=Valor”)**

Si se llama a una referencia de un bean se pone:

-**@Attribute(“ref=beanId”)**

Estructura:

```
@Bean
public class BeanClass {

	@Attribute(“val=2”)
	private int classInt;

	public void setClassInt() {
		...
	}
}

También puede ser

@Bean
public class BeanClass {
```

```
	@Attribute(“ref=otherBeanId”)
	private otherBean beanReference;

	public void setBeanReference() {
		...
	}
}
```

Seria equivalente a:

```
<beans>
	<bean  id = "beanId" class = "package.path.class">
		<attribute name="nombreAtr" value="valor"/ref="beanId" />
	</bean>
</beans>
```

@Constructor

Se pone arriba del constructor que se utilizará para la inyección de las dependencias del **Bean**. Los valores y referencias que se necesitan para los parámetros del constructor, se especifican enviando al Annotation por parámetro una string en el orden que el constructor los tiene que recibir. Poniendo **val** si es un valor explícito, y **ref** si es una referencia a un bean.

Estructura

@Bean

```
public class BeanClass{

	@Constructor(“val=val1,ref=beanId,..,valn”)
	Public BeanClass(. . .){
		. . .
	}
}
```

Sería equivalente a:

```
<beans>
	<bean  id = "BeanClass" class = "package.path.class">
		<constructor>
			<param type="package.path.class"
				index="numberIndex"
				value="valor"/ref="beanId" />
		</constructor>
	</bean>
</beans>
```