**Spring version 5.3.6**

**1.1. Introducción a Spring IoC Container and Beans**

Aca cubrimos la implementación de Spring Framework del principio de Inversión de Control (IoC). IoC también se conoce como inyección de dependencia (DI).

**Inyección de dependencia (DI)**: Es un proceso mediante el cual los objetos definen sus dependencias (es decir, los otros objetos con los que trabajan) solo a través de argumentos de constructor, argumentos para un método de fábrica o propiedades que se establecen en la instancia del objeto después de que se construye o devuelve desde un método de fábrica.

*Nota: Entiendo que cuando se crean las* ***beans****, es a través de la inyección de las dependencias, ahí se crean las* ***beans****. Entonces en Spring los objetos son denominados* ***beans.*** *Un bean es un objeto que es instanciado, ensamblado y administrado por el contenedor Spring IoC.*

### 1.2. Descripción general del contenedor

los org.springframework.context.ApplicationContext representa al contenedor IoC y es el responsable de crear instancias, configurar y ensamblar los beans. El contenedor obtiene sus instrucciones sobre que objetos instanciar, configurar y ensamblar leyendo los metadatos (configurado XML). Estos XML permite ver los objetos de mi app y las interdependencias de los mismos.

El siguiente diagrama muestra una vista de alto nivel de cómo funciona Spring. Sus clases de aplicación se combinan con metadatos de configuración para que, después de la ApplicationContext se crea e inicializa, tiene un sistema o aplicación completamente configurado y ejecutable.

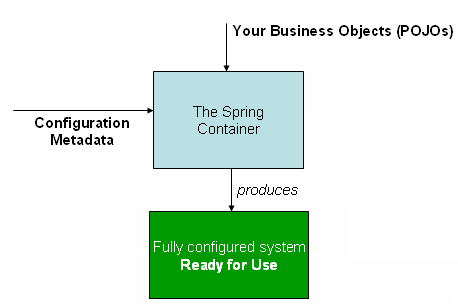


Figura del Contenedor Spring (IOC)

**1.2.1. Metadatos de configuración**

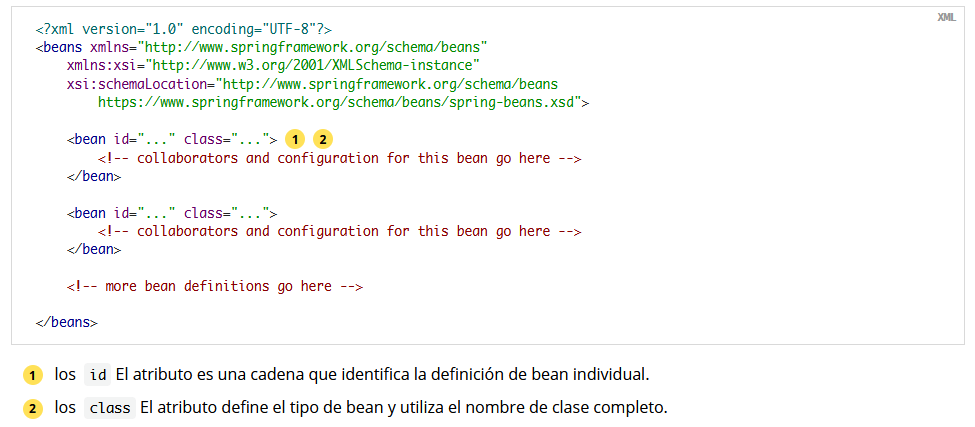
Dado el diagrama anterior que representa como funciona el contenedor. Entonces estos metadatos representan como yo le digo al contenedor Spring como debe configurar, crear instancias y ensamblar objetos de mi app.

**Metadatos en XML:**

La configuración consta de al meno una, pero normalmente más de una definición de bean que debe administrar el contenedor. Los metadatos de esta configuración basado en XML configuran los bean como elementos dentro de un nivel superior

Etiqueta: <bean> </bean>

En los beans, por lo general, define los objetos de la capa de servicio, los objetos de acceso a datos (DAO), los objetos de presentación como Struts Action instancias, objetos de infraestructura como Hibernate SessionFactories, JMS Queues, Etcétera.



**1.2.2. Crear una instancia de un contenedor**

La ruta de ubicación o rutas proporcionadas a un constructor de tipo ApplicationContext son cadenas de recursos que permiten que el contenedor cargue los metadatos de configuración.

ApplicationContext context = new ClassPathXmlApplicationContext

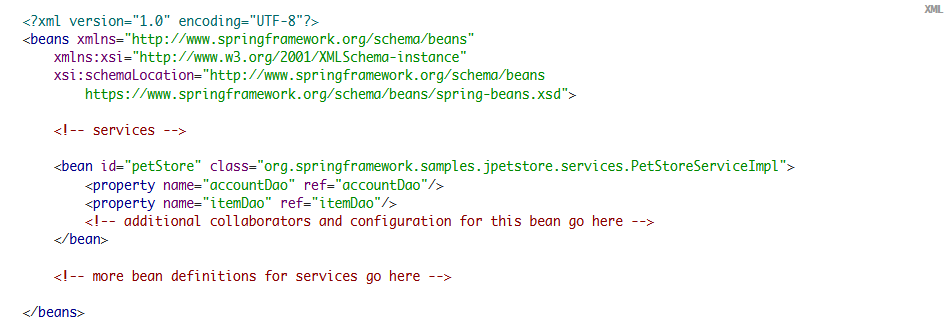
(

"services.xml",

"daos.xml"

);

Esto muestra los objetos de la capa de servicio services.xml



Y este ejemplo muestra los objetos de acceso a datos

**

##### **Redacción de metadatos de configuración basados ​​en XML.**

Puede resultar de utilidad, que las definiciones de beans abarquen diferentes archivos XML.

Cada archivo de configuración XML individual representa una capa lógica o un modulo en la arquitectura.

Se pueden cargar definiciones de beans de otros XML, utilizando el tag <import/> con el parámetro Resources como ruta de otro archivo de configuración.

<beans>

<import resource="services.xml"/>

<import resource="resources/messageSource.xml"/>

<import resource="/resources/themeSource.xml"/>

<bean id="bean1" class="..."/>

<bean id="bean2" class="..."/>

</beans>

Las definiciones de beans deben de ser validas de acuerdo con Spring Schema.

#### **1.2.3. Usando el contenedor**

ApplicationContext es la interfaz de una fábrica avanzada capaz de mantener un registro de diferentes beans y sus dependencias. Usando el método T getBean(String name, Class<T> requiredType), puede recuperar instancias de sus beans.

Permite leer definiciones de beans y acceder a ellas.

*ApplicationContext* context = new ClassPathXmlApplicationContext("service.xml");  
  
Message msg = context.getBean("helloWorld", Message.class);System.*out*.println(msg.getMessage());

La variante más flexible es GenericApplicationContext en combinación con lectores delegados, por ejemplo, con XmlBeanDefinitionReader para archivos XML, como muestra el siguiente ejemplo:

GenericApplicationContext context = new GenericApplicationContext();

new XmlBeanDefinitionReader(context).loadBeanDefinitions("services.xml", "daos.xml");

context.refresh();

Se puede utilizar getBeans()para recuperar instancias de las beans, la ApplicationContext (interfaz) tiene otros métodos para recuperar beans, pero preferentemente el código de mi app no debería usarlos, es decir que no debo tener llamadas de getBeans().

### 1.3. Descripción general de Bean

<https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-definition>

### 1.3.1 Nombrar Beans

El nombramiento de beans es sencillo, simplemente usando como referencia dos cosas, ya sea name o id Lo recomendable para definir estos, es primero letra en minúscula y luego mayúscula, es decir: accountManager, accountService, userDao, loginController.

Si no defino ninguno de estos, Spring lo hace de manera automática, pero sería difícil acceder a estos ya que no tengo una referencia alguna para acceder.

#### **1.3.2. Instanciar frijoles**

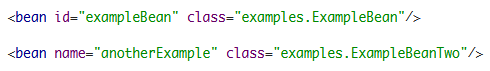
Para instanciar beans lo necesario es usar la keyword class suele ser obligatorio.

**Nombres de clases anidados**

Si desea configurar una definición de bean para una clase anidada, puede utilizar el nombre binario o el nombre de origen de la clase anidada.

Por ejemplo, si tienes una clase llamada SomeThing en el com.example paquete, y este SomeThing la clase tiene un static clase anidada llamada OtherThing, pueden estar separados por un signo de dólar ( $) o un punto ( .). Entonces el valor de la class atributo en una definición de frijol sería com.example.SomeThing$OtherThing o com.example.SomeThing.OtherThing.

##### **Instanciación con un constructor**



Inversión de Control (Inversion of Control)

Se dice que hay inversión de control porque es el framework el que se encarga de instanciar y controlar los objetos. Esta externalizado el manejo de los objetos.

Esto nos da ventajas como: Modularidad – ampliar funcionalidad de nuestras aplicaciones sin modificar clases – evitar dependencia entre clases – flexibiliza mi app siendo más configurable.

**1.4. Dependencias**

Es una relación existente entre dos clases, una clase depende de…Es decir, que una clase depende de otra para poder funcionar.

**1.4.1. Inyección de Dependencias**

Este es un proceso donde los objetos definen sus dependencias solo con argumentos de un constructor. O sea, se le está inyectando la dependencia en un constructor, ver ejemplo:

// Clase con dependencia definida

class Moto {

private String nombre;

private Freno freno;

public Moto (String nombre) {

this.nombre = nombre;

this.freno = new Freno(“XSwagger”);

}

}

// Clase con proceso de Inyeccion

// Se ve como se inyecta la dependencia con el objeto pasado por

// parametro

class Moto {

private String nombre;

public Moto (String nombre, **Freno freno**) {

this.nombre = nombre;

**this.freno = freno;**

}

}

Para esto, por lo general se usa la Abstracción o las Interfaces para que el uso sea más fácil para probar.

##### **Inyección de dependencia basada en constructor**

##### <https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-constructor-injection> Resolución de Argumentos de Constructor **(Sin entender)**

class Moto {

private String nombre;

public Moto (String nombre, **Freno freno**) {

this.nombre = nombre;

**this.freno = freno;**

}

}

Dada la siguiente clase, veamos como se tratan los constructores definidos en la metadata, siendo que este POJO está en el paquete com.Models

**public SomeService {**

**private String data;**

**public SomeService (String data) {this.data = data;}**

**}**

**Coincidencia de tipo de argumento de constructor**

Vemos claramente que se setea información desde el tipo de dato que tiene el constructor

<bean id="miServicio-I"  
 class="com.Models.SomeService">  
 <constructor-arg  
 type="java.lang.String"  
 value="METADATA {service.xml}"/>  
</bean>

**Índice de argumentos del constructor**

Vemos claramente que se setea información desde la posición de donde esta el argumento posicionado en el constructor

<bean id="miServicio-I"  
 class="com.Models.SomeService">  
 <constructor-arg  
 index="0"  
 value="METADATA {service.xml}"/>  
</bean>

**Nombre del argumento del constructor**

Acá se puede ver que lo hace mediante el nombre, en este caso es ‘data’

<bean id="miServicio-I"  
 class="com.Models.SomeService">  
 <constructor-arg  
 name="data"  
 value="METADATA {service.xml}"/>  
</bean>

Para que funcione, el código debe compilarse con la depuración habilitada para que Spring pueda buscar el nombre del parámetro en el constructor. Si no quiero o no deseo compilar con el indicador de depuración, se puede usar la anotación JDK [@ConstructorPropiedades](https://download.oracle.com/javase/8/docs/api/java/beans/ConstructorProperties.html)  para nombrar explícitamente los argumentos de su constructor. Ejemplo:

@ConstructorProperties({"data"})  
public SomeService (String *data*) {  
 *this*.data = *data*;  
}

##### **Inyección de dependencia basada en setter**

Es prácticamente un setter que inyecta la dependencia. Ejemplo sencillo del método:

**class Moto {**

**private String nombre;**

**public Moto (String nombre, Freno freno) {**

**this.nombre = nombre;**

**this.freno = freno;**

**}**

**public void setFreno(Freno freno){**

**this.freno = freno;**

**}**

**}**

¿DI basado en constructor o basado en setter?

Es buena práctica usar la inyección desde el constructor como argumento, ya que no permite que la dependencia sea null en cualquiera de los casos. También, se usa desde el constructor para dependencias obligatorias o requeridas. Sin embargo, para DI en setter, este solamente es recomendado usarlo para dependencias opcionales.

##### **Proceso de resolución de dependencia**

El contenedor realiza la resolución de la dependencia del frijol de la siguiente manera:

* La ApplicationContext se crea e inicializa con metadatos de configuración que describe todos los frijoles. Los metadatos de configuración se pueden especificar mediante XML, código Java o anotaciones.
* Para cada bean, sus dependencias se expresan en forma de propiedades, constructor argumentos, o argumentos para el método static-factory (si usa eso en lugar de un constructor normal). Estas dependencias se proporcionan al bean, cuando el bean es realmente creado.
* Cada propiedad o argumento del constructor es una definición real del valor a establecer, o una referencia a otro frijol en el contenedor.
* Cada propiedad o argumento de constructor que es un valor se convierte de su especificado formato al tipo real de esa propiedad o argumento de constructor. Por defecto, Spring puede convertir un valor proporcionado en formato de cadena a todos los tipos integrados, como int, long, String, boolean, Etcétera.

El contenedor Spring valida la configuración de cada bean a medida que se crea el contenedor. Sin embargo, las propiedades del bean en sí no se establecen hasta que se crea realmente el bean. Se crean los beans que tienen un alcance único y se configuran para ser instanciados previamente (el valor predeterminado) cuando se crea el contenedor. Los ámbitos se definen en [Bean Scopes](https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-factory-scopes) . De lo contrario, el bean se crea solo cuando se solicita. La creación de un frijol potencialmente causa un gráfico de beans que se crearán, como las dependencias del bean y sus dependencias ' las dependencias (y así sucesivamente) se crean y asignan. Tenga en cuenta que las discrepancias de resolución entre esas dependencias pueden aparecer tarde, es decir, en la primera creación del bean afectado.

**Dependencias circulares**

Si utiliza predominantemente la inyección de constructor, es posible crear un escenario de dependencia circular.

Por ejemplo: la clase A requiere una instancia de la clase B a través de la inyección del constructor, y la clase B requiere una instancia de la clase A a través de la inyección del constructor. Si configura beans para que las clases A y B se inyecten entre sí, el contenedor Spring IoC detecta esta referencia circular en tiempo de ejecución y lanza una BeanCurrentlyInCreationException.