# BeanFactory

Es la clase base sobre la que se asienta la [inversión de control](http://campus.dicotraining.com/mod/book/view.php?id=15) (IoC) en Spring. Los objetos gestionados por Spring son llamados beans y el BeanFactory permite la configuración de dichos objetos usando [inyección de dependencias](http://campus.dicotraining.com/mod/book/view.php?id=26).

Normalmente poco cantidad de código se enlazará con el contenedor de Spring, incluso tales dependencias con el código de spring se pueden eliminar cargando las factorías de forma declarativa.

Existen diferentes implementaciones que soportan diferentes niveles de funcionalidad, dependiendo del propósito.

Hasta la versión 2.5 de Spring el mecanismo de configuración es normalmente XML, pero incluso en la versión 3.x se puede usar sin problemas, a partir de la versión 3.0 la configuración, si se desea, puede ser desplazada completamente por anotaciones en Java, pero para poder usar una versión 3.x de Spring será necesario utilizar un JDK de Java versión 1.5 o superior, puesto que a partir de esta versión es cuando se pueden usar características como los genéricos o las anotaciones.

Todas las factorías implementan el interfaz org.springframework.beans.factory.BeanFactory y los subinterfaces aportan funcionalidad extra, por ejemplo:

* BeanFactory es el interfaz básico y permite operaciones como extraer un bean del contexto de diversas maneras:
  + getBean(String name): recuperará un bean por el nombre encontrado en la configuración.
  + getBean(String name, Class requiredType) permiten especificar la clase devuelta por el bean o lanzar un excepción si no existe.
* HierarchicalBeanFactory: La mayoría de las factorías pueden ser creadas como partes de una jerarquía.Si se solicita una factoría y no existe, la factoría padre puede ser consultada.

Todos los beans declarados forman parte de una jerarquía.

Otras factorías de conveniencia son:

* ListableBeanFactory: Permiten listar o enumerar factorías de beans Incluyendo los nombres de todos los beans, los nombres de todos los beans de cierto tipo y el número de beans de la factoría. Funciona sólo para un bean factory.
* BeanFactoryUtils dispone de los mismos métodos pero son aplicables a una jerarquía.
* AutowireCapableBeanFactory: Permite mediante los métodos autowireBeanProperties() y applyBeanPropertyValues() configurar objetos externos, Permite trabajar con código de terceros que no es instanciado por Spring, pero aún así admiten [inyección de dependencias](http://campus.dicotraining.com/mod/book/view.php?id=26).
* ConfigurableBeanFactory: permite opciones de configuración adicional en un bean básico, aplicadas en la estapa de inicialización Todos los interfaces pueden lanzar excepciones que son subclase de BeansException

1.1 ApplicationContext

Se trata de una factoría avanzada: org.springframework.context.ApplicationContext hereda de BeanFactory, pero lo amplía con bastante información.

Esta factoría utiliza un estilo orientado al framework: ciertas operaciones manejadas de forma programática con el bean Factory pueden ser manejadas de manera declarativa.

Ofrece un reconocimiento automático y uso de post-procesadores y una serie de utilidades y capas de abstracción de recursos, como MessageResource que se trata de un Interfaz usado para localizar mensajes localizados.

Entre algunas de las capas de abstracción que puede usarse se encuentra:

Integraciones con frameworks de persistencia como:

* JDBC
* JDO
* JPA
* Hibernate
* Toplink
* EclipseLink
* iBatis

Integraciones con frameworks Web como:

* Java Server Faces
* Struts
* Tapestry
* SpringMVC

Capas de abstracción para conexiones remotas como RMI, JNDI, Hessian, Burlap. Integración con lenguajes de Scripting y Servicios Web y REST. Así como capas de recursos para manejar conversiones ente datos de forma automática y comprobación de tipos de manera dinámica.

También da soporte a eventos de aplicación y del framework, en donde  el contexto es capaz de disparar estos eventos hacia listeners registrados, o el ResourceLoader una capa de  Abstracción para manejar recursos de bajo nivel y poder cargar los recursos desde distintas fuentes como sitema de ficheros, URLs, sockets, classpath, etc.

En general es mejor siempre usar Application Context vs. Bean Factory dejando este último para aplicaciones con requisitos muy concretos sobre el consumo de recursos y memoria.

1.2 Cargar el contexto

Spring proporciona una serie de factorías de conveniencia para poder cargar recursos e instanciar el contexto. El primer paso antes de poder recuperar un bean declarado en un fichero de configuración (por ejemplo en XML) será cargar dicho fichero de configuración, bien en el BeanFactory, bien en el ApplicationContext, con preferencia sobre este último por lo explicado en el punto anterior.

Por ejemplo, podemos declarar un bean dentro de un fichero xml junto con su declaración de XMLSchema:

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://www.springframework.org/schema/context" xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<beans>

<bean id="prediccionDao" class="es.adama.daos.PrediccionDaoImpl">

</bean>

</beans>

En este caso estamos declarando una clase es.adama.daos.PrediccionDaoImpl.class para que sea encontrada por Spring y le asigne un nombre "prediccionDao". Esto es lo mismo que si hicieramos desde dentro de un programa en Java:

import es.adama.daos.\*;

PrediccionDaoImpl prediccionDao = new PrediccionDaoImpl();

Llegado este punto podemos cargar el fichero XML de varias formas.

La primera podría ser cargarlo como un recurso con una de las factorías explicadas anteriormente, como es ResourceLoader y el Interfaz Resource, por ejemplo:

Resource recurso = new FileSystemResource("beans.xml");

BeanFactory contexto = new XmlBeanFactory(recurso);

En este ejemplo estaríamos cargando un fichero xml (beans.xml) como el de arriba con la definición del prediccionDao desde un sistema de ficheros mediante FileSystemResource y posteriormente invocamos un contexto de Spring indicando que el tipo de fichero de configuración es XML, mediante XMLBeanFactory.

Por supuesto, podemos cambiar el BeanFactory por ApplicationContext y el resultado sería el mismo. En ambos casos se cargaría el contexto de Spring con todos los beans asociados.

Merece la pena ver la [documentación del interfaz Resource](http://static.springsource.org/spring/docs/3.0.x/javadoc-api/org/springframework/core/io/Resource.html), donde se puede apreciar las factorías de carga de recursos tan amplia de las que dispone. Por ejemplo podríamos cargar recursos de distintos tipos como ficheros, objetos serializados y desde distintas fuentes, como un contexto Servlet, un Portet, desde un sistema de ficheros, un socket o desde el propio classpath de la aplicación.

Otra de las cosas que Spring proporciona son una serie de clases de utilidad para mejorar el desarrollo y factilitar las tareas. Por ejemplo si queremos cargar un fichero xml para crear el contexto de Spring y el fichero reside dentro del classpath de la aplicación podemos hacerlo en una única línea sin necesidad de cargarlo primero como un Resource:

ApplicationContext contexto = new XmlClasspathApplicationContext("beans.xml");

Y ya estaría, el nombre de la clase lo dice todo. Esta clase buscaría el fichero en el classpath de la aplicación, algo que por otra parte es lo común en la mayoría de las aplicaciones. Sin embargo, las posibilidades que nos da spring de poder cargar recursos desde distintas fuentes de datos nos permite pensar en arquitecturas descentralizadas con repositorios de configuración que pueden ser accedidos en remoto y así configurar entornos de trabajo bajo demanda, para producción, o testing, por ejemplo.

## Cargar varios ficheros de configuración

En proyectos de cierta envergadura, no es común disponer de todos los beans configurados en el mismo XML, por lo que se parte en varios documentos XML en función de la funcionalidad, o los servicios, o incluso en función de los equipos de desarrollo.

En estos casos hay varias estrategias a seguir siendo la más sencilla y usada el cargarlos mediante un array de Strings de esta manera:

ApplicationContext appContext = new ClassPathXmlApplicationContext( new String[] {"beans-servicios.xml", "beans-dao.xml"});

## Otras opciones

Spring permite usar un prefijo para especificar todos los recursos de un nombre en particular que se encuentren en el classpath: classpath\*:

ApplicationContext appContext = new ClassPathXmlApplicationContext("classpath\*:ApplicationContext.xml");

Cargar factorías configuradas mediante XML es secillo:

ClassPathResource res = new ClassPathResource("es/adama/spring/beans.xml");

XmlBeanFactory factory = new XmlBeanFactory(res);

O bien

FilesystemResource res = new FilesystemResource("/ruta/algun/fichero/beans.xml");

XmlBeanFactory factory = new XmlBeanFactory(res);

También es posible usar un InputStream:

InputStream is = new FileInputStream("/ruta/algun/fichero/beans.xml");

XmlBeanFactory factory = new XmlBeanFactory(is);

En el caso de que estemos trabajando con anotaciones en lugar de XML debemos usar la clase [AnnotationConfigApplicationContext](http://static.springsource.org/spring/docs/3.0.x/javadoc-api/org/springframework/context/annotation/AnnotationConfigApplicationContext.html). También disponemos de un modelo mixto en el que parte de la configuración pueda estar en XML y parte en anotaciones. Spring nos permite jugar con todo ello.

1.3 Inversión de Control

Una vez cargado el contexto de Spring es hora de trabajar con el mismo.

Sobre el contexto podemos alcanzar cualquier bean gestionado por el propio spring y que aparezca en alguna de sus jerarquías de Beans.

Existen varias formas de acceder a recuperar un bean, mediante los métodos .getBean() en sus diversas formas ([ver documentación](http://static.springsource.org/spring/docs/3.0.x/javadoc-api/org/springframework/beans/factory/BeanFactory.html)), por ejemplo, imagina que en el documento XML anterior la clase PrediccionDaoImpl fuera un servicio de predicción metereológica:

PrediccionDao prediccion = (PrediccionDao) contexto.getBean("prediccionDao");

En este caso estamos pidiendo a Spring que nos devuelva un objeto cuyo ID en el fichero XML anterior se llama "prediccionDao". Es Spring quien nos devuelve ese objeto, no hay otra clase que llama a este objeto y hace un new de dicho objeto. Esto es [inversión de control](http://campus.dicotraining.com/mod/book/view.php?id=15).

Podemos recuperar objetos gestionados por spring de distintas maneras, por ejemplo:

* getBean(NombreDeClase.class) recuperará una instancia de un objeto que corresponda a dicha clase.
* getBeansOfType(NombreDeClase.class) recuperará una colección (un Map) con todas las instancias que correspondan a esa clase.

En la propia documentación del API de BeanFactory puedes encontrar varios métodos de conveniencia para instanciar objetos, en función del ID, del nombre de la clase, de la clase o un subtipo de la misma, etc.