Spring IOC

Dependency injection is a pattern we can use to implement IoC, where the control being inverted is setting an object's dependencies.

Connecting objects with other objects, or “injecting” objects into other objects, is done by an assembler rather than by the objects themselves.

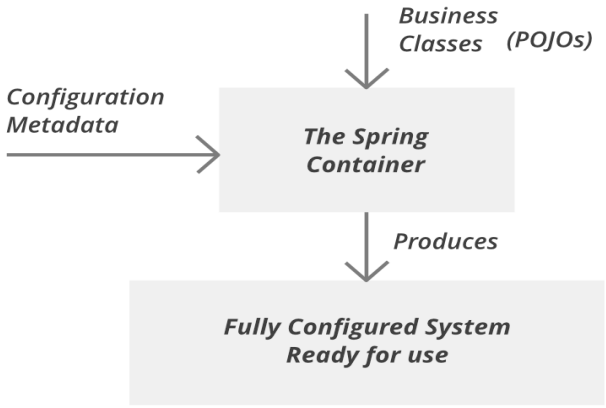
Spring IoC is the mechanism to achieve loose-coupling between Objects dependencies. To achieve loose coupling and dynamic binding of the objects at runtime, objects dependencies are injected by other assembler objects.

Spring IoC container is the program that **injects** dependencies into an object and make it ready for our use.

IoC container is responsible to instantiate, configure and assemble the objects. The IoC container gets informations from the XML file and works accordingly.

The main tasks performed by IoC container are:

- to instantiate the application class
- to configure the object
- to assemble the dependencies between the objects



Spring IoC container provides us different ways to decouple the object dependencies.

BeanFactory is the root interface of Spring IoC container. ApplicationContext is the child interface of BeanFactory interface that provide Spring AOP features, i18n etc.

Some of the useful child-interfaces of ApplicationContext are **ConfigurableApplicationContext** and **WebApplicationContext**.

Spring Framework provides a number of useful ApplicationContext implementation classes that we can use to get the spring context and then the Spring Bean.

Some of the useful ApplicationContext implementations that we use are;

- **AnnotationConfigApplicationContext**: If we are using Spring in standalone java applications and using annotations for Configuration, then we can use this to initialize the container and get the bean objects.
- **ClassPathXmlApplicationContext**: If we have spring bean configuration xml file in standalone application, then we can use this class to load the file and get the container object.
- **FileSystemXmlApplicationContext**: This is similar to ClassPathXmlApplicationContext except that the xml configuration file can be loaded from anywhere in the file system.
- **AnnotationConfigWebApplicationContext** and **XmlWebApplicationContext** for web applications.

Spring provides two types of Containers namely as follows:

1. BeanFactory Container
2. ApplicationContext Container

Spring Bean Configuration

Spring Framework provides three ways to configure beans to be used in the application.

- **Annotation Based Configuration** - By using @Service or @Component annotations. Scope details can be provided with @Scope annotation.
- **XML Based Configuration** - By creating Spring Configuration XML file to configure the beans. If you are using Spring MVC framework, the xml based configuration can be loaded automatically by writing some boiler plate code in web.xml file.

- **Java Based Configuration** - Starting from Spring 3.0, we can configure Spring beans using java programs. Some important annotations used for java based configuration are @Configuration, @ComponentScan and @Bean.

Spring Bean Scopes

There are five scopes defined for Spring Beans.

singleton - Only one instance of the bean will be created for each container. This is the default scope for the spring beans. While using this scope, make sure bean doesn't have shared instance variables otherwise it might lead to data inconsistency issues.

prototype - A new instance will be created every time the bean is requested.

request - This is same as prototype scope, however it's meant to be used for web applications. A new instance of the bean will be created for each HTTP request.

session - A new bean will be created for each HTTP session by the container.

global-session - This is used to create global session beans for Portlet applications.