SPRING FRAMEWORK

Spring framework is the open-source framework for developing java applications. It makes it easy to create java enterprise applications. Spring frameworks core features are used by the java application. It has extensions for building web applications.

INVERSION OF CONTROL (IOC):

The IOC is a design principle which is used to manage components which are inverted to external entity. Rather than instantiating objects by the application is overridden, the container will manage creating, managing and, destroying of the beans.

- IOC is also known as *Dependency Injection(DI)*. The container will inject dependencies while creating the bean, this process is inverse hence the name *Inversion of Control*.
- The ***org.springframework.beans*** and ***org.springframework.context*** pakages are the basis for the Spring Frameworks IoC container.
- *BeanFactory* interface provides the configuration mechanism for managing any type of objects. *ApplicationContext* is a sub-interface of *BeanFactory*.

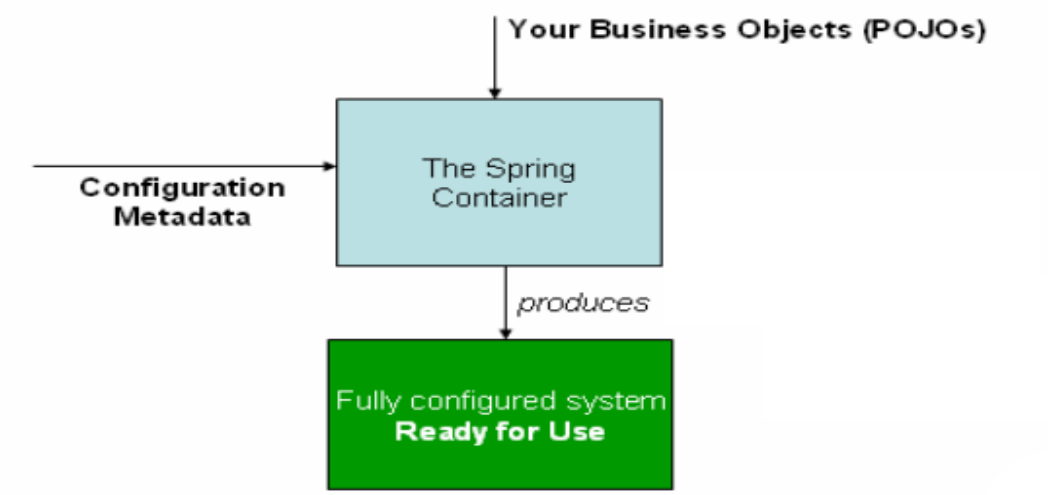

Figure 1. The Spring IoC container

Why *ApplicationContext* interface different from *BeanFactory*:

ApplicationContext provides more enterprise specific functionality whereas *BeanFactory* provides configuration framework and basic functionality.

- Easier interaction with **AOP** feature.
- Message resource handling.

- Event publication.
- Application layer specific context such as *WebApplicationContext* for use in web applications.

SPRING IOC CONTAINER HIERARCHY

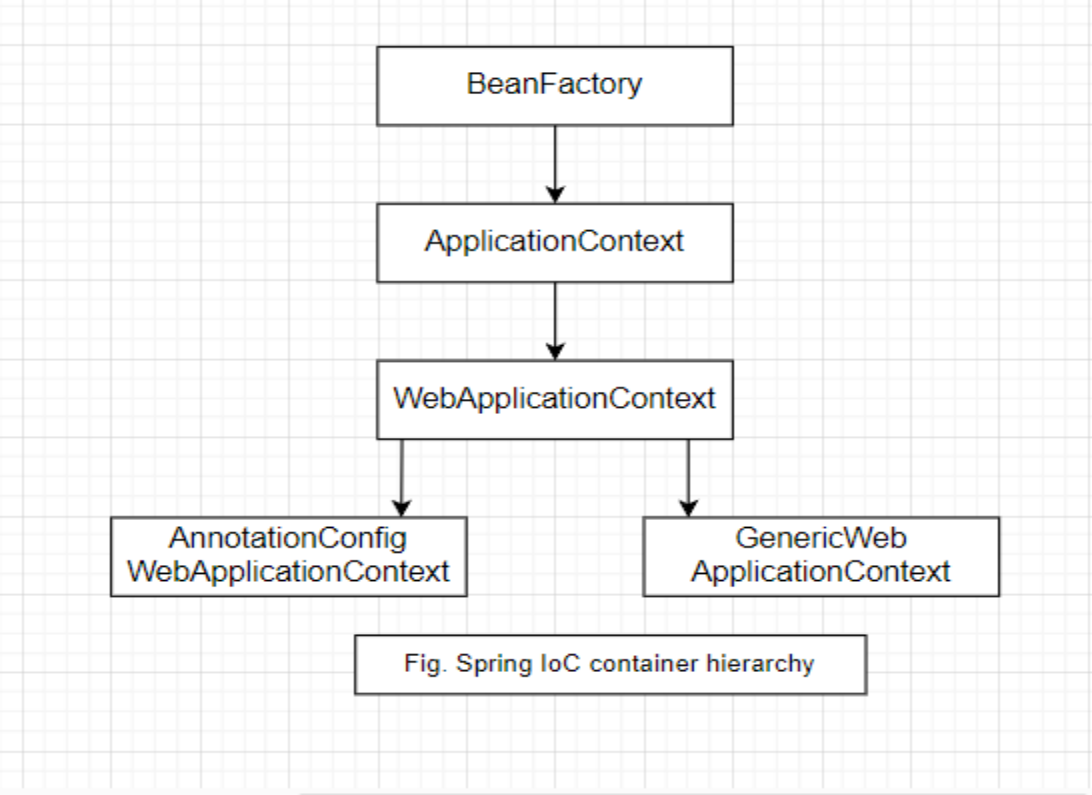

Fig. Spring IoC container hierarchy

The objects that are needed for the application are managed by the *Spring IoC container* i.e., Spring will *create, initializes, and register* those objects are called ***beans***.

- ***org.springframework.context.ApplicationContext*** (***ApplicationContext***) or interface represents Spring IoC container is responsible for instantiating, configuring, and assembling of beans.
- ***ApllicationContext*** is also known as Spring IoC container.
- IoC container gets instructions on what objects to instantiate, configure and assemble by reading *configuration metadata* that is supplied to the container.
- Spring IoC container can mange one or more beans.
- *ApplicationContext* (or Container) has several implementations and one of them is *WebApplicationContext* sub-interface, is also a container.

- The *ApplicationContext* implementations also permit the registration of existing objects that are created outside the container. Means it can manage beans that are created outside by reading configuration metadata.
- Configuration metadata may be in XML, Java Annotations or Java code.
- ApplicationContext has several implementations, in stand-alone applications we create instance of ClassPathXmlApplicationContext or FileSystemXmlApplicationContext.

CONFIGURATION METADATA:

Configuration metadata is an instructions providing to the container to create bean objects. Metadata can be supplied in multiple ways, XML based configuration, Java code and Java Annotations.

XML BASED CONFIGURATION:

- In this configuration the bean is defined inside the <bean> which comes under the super <beans> tag.
- The <bean> contains id and class attributes.
- Id attribute identifies the individual bean definition and class attribute defines the type of the bean and uses the fully qualifies name.
- The property name element defines the bean is dependent on another object to complete its work, name defines the name of the JavaBean name and ref element refers to another bean object.
- We retrieve the instance of the bean by using the ***getBean()*** method.

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

    <!-- Define a bean -->
    <bean id="myService" class="com.example.MyService">
        <property name="dependency" ref="myDependency"/>
    </bean>

    <bean id="myDependency" class="com.example.MyDependency"/>
</beans>
```

BEAN:

The Spring IoC container manages one or more beans. These are created with configuration metadata that is supplied to the container. The beans are created inside the IoC container. The bean definitions are represented as BeanDefinition objects, which contains metadata.

- A package (fully qualified name) class name.
- Behavior of the bean, which states how the bean should behave in the container.
- Reference to the other beans to do their work.
- Contains information on how to create a specific bean.
- The ApplicationContext implementations also permit the registration of existing objects that are created outside the container.

DEPENDENCIES:

DEPENDENCY INJECTION:

Dependency injection(DI) is a process whereby objects define their dependencies that they work with other objects only through constructor arguments, arguments to factory methods or properties that are set on the object instance after it has constructed or returned from the factory.

DI can be applied in two ways; constructor based injection and setter based injection.

Constructor based DI:

The constructor-based dependency injection is done by the container invoking a constructor with a few arguments, each representing a dependency. It is like methods with specific arguments to construct the bean.

```
public class SimpleMovieLister {

    // the SimpleMovieLister has a dependency on a MovieFinder
    private final MovieFinder movieFinder;

    // a constructor so that the Spring container can inject a MovieFinder
    public SimpleMovieLister(MovieFinder movieFinder) {
        this.movieFinder = movieFinder;
    }

    // business logic that actually uses the injected MovieFinder is omitted...
}
```

- To avoid the ambiguity, it is allowed to provide the arguments explicitly by using the annotation @ConstructorProperties. On top of the constructor of the class.
- The ambiguity can avoid in other ways also by mentioning the type attribute, it takes values of datatypes of the property.
- Another way is available to avoiding the ambiguity of the bean by using the name attribute inside the <constructor-arg name="….">.

Setter based DI:

Setter based dependency injection is accomplished by the container calling setter methods after invoking no-argument constructor or no-argument static factory method to instantiate bean.

- Setter based DI is also used to calling circular based DI i.e. two objects are dependent on each other to do their work.

```
public class SimpleMovieLister {

    // the SimpleMovieLister has a dependency on the MovieFinder
    private MovieFinder movieFinder;

    // a setter method so that the Spring container can inject a MovieFinder
    public void setMovieFinder(MovieFinder movieFinder) {
        this.movieFinder = movieFinder;
    }

    // business logic that actually uses the injected MovieFinder is omitted...
}
```

- In DI if one object needs the reference of the other object then it can be used through the <ref bean="…"> tag.
- Another way is by using ref attribute <constructor-arg ref=".." />
- We can define one bean inside the other outer bean, but inner bean doesn't have the id attribute or name attribute. If specified, the id or name attributes then container will ignore them.
- It is not possible to create object of the inner bean.

Autowiring Collaborators:

The Spring container can autowire relationship between collaborating beans. Spring container will resolve the collaborating by automatically inspecting the contents of the ApplicationContext.

- Autowiring reduces the need to specify the property or constructor arguments.
- If we want to modify the dependency of the class, then it will automatically have satisfied by the dependency without modifying anything.
- When using XML based configuration is done by autowire attribute inside the <bean> element.
- There are two ways to apply autowire i.e. XML based and Annotation based.
- In XML based there are different modes are present:
    - No mode
    - Byname
    - byType
    - constructor
- In Annotation based @Autowired is used.
- In XML based configuration if we used autowire attribute to specify the dependency then no need to use reference attribute.
- If more than one beans are present, the container will get confuse about creating beans for that we use another annotation @Qualifier along with @Autowire.
- The @Qualifier annotation takes one argument i.e. name of the bean which is going to create by the container.

BEAN LIFE CYCLE CALLBACKS:

When every bean is created by the IoC container it has two life cycle methods they are initialization method and destroy method.

- Every time when the container created bean definition it will call the initialization method before setting the properties to the bean.
- After all the transactions done, then it will call destroy method.
- There are three ways for initialization and destroy the bean.
- One is by define the standard or custom methods and then pass their names to the init-method and destroy-method of the bean.
    - <bean init-method=”init” destroy-method=”destroy” />
- Second way is by implementing the InitializingBean and DisposableBean interfaces.
- The InitializingBean interface provides afterPropertiesSet() method and DisposableBean interface provides the destroy() method.
- The third easy and simplest way of initializing and destroying the bean is by using the Annotations.

- For initialization method we are using the @PostConstruct annotation and for the destroy method @PreDestroy annotation.

BEAN SCOPE:

Spring framework given the flexibility to manage beans, not only creation of beans it also allows us to control the scope of the bean that is created through the configuration metadata.

- We can control bean scope in two ways using XML configuration or @Scope annotation.
- The @Scope annotation is always existing with @Component annotation.
- The Spring framework supports six scopes, from that four are web aware they are used only in web applications. We Can also create custom scopes also.
- They are:
    - Singleton
    - Prototype
    - Session
    - Request
    - Application
    - Websocket

1) Singleton: it is a default scope. The single bean definition to single object instance of every Spring IoC container. Every time the container will call only one bean definition for creation of the object, all the objects have the same hashCode value.

2) Prototype: The bean instance is created every time when request from the Spring IoC container. In prototype scope every beans are different.

The other scopes like request scope, session scope, application scope are only runs in web application not get executed in stand-alone applications.

3) Request: A single instance of the bean is created for each HTTP request.

4) Session: A single instance of the bean created for each HTTP session.

ANNOTATION BASED CONFIGURATION:

Annotation based configuration is the another way to create bean. Container will create, manage and assemble the beans.

- The annotation based configuration provides more flexibility to write concise code for configuration.
- The container will create beans based on the metadata i.e. provided using the annotations.
- It is also possible to integrate both XML and Annotations to create beans.
- To use annotations, we must declare context name space
    - <context:annotation-config /> in XML.

@Autowired Annotation:

The @Autowired annotation is used for automatic dependency injection.

- It allows the Spring container to provide an instance of required dependency when the bean is created.
- It can be applied to fields, constructors and methods.
- When using on fields Spring container will automatically inject bean of required type into a field.
- When used on constructor Spring container will automatically inject bean into the constructor parameter.
- When used on methods Spring container will automatically inject the bean into method parameter.
- The @Inject annotation can be used in place of @Autowired annotation.

@Qualifier Annotation:

The @Qualifier annotation is used to differentiate a bean among the same type of bean objects. If we have more than one bean of same type and want to wire one of them then we need to use @Qualifier along with @Autowired annotation.

The @Qualifier annotation is usually used to tell @Autowired to, which bean it has injected.

@Value Annotation:

The @Value annotation is used to provide the values to property using the annotation.

To provide the value to Collection fields first we have to declare util collection type in XML configuration file along with id attribute, after that value of the id attribute is specified inside the @Value annotation using Expression Language.

@Primary Annotation:

The @Autowire annotation leads to many candidates, to control over those we use @Primary annotation. The @Primary indicates that a particular bean should be given more preference when multiple beans are present.

It can also be used with XML bean using primary attribute

<bean class="…" primary="true" />

@Resource Annotation:

The @Resource annotation is used to inject resources into beans or object they needed. @Resource annotation can be used with fields, property methods(setter), and constructor. It takes two attributes name and type attribute.

The name attribute specifies name of the resource to be injected.

The type attribute specifies that type of the resource to be injected.

It works same has a @Autowired annotation.

@Component Annotation and Stereotype Annotations:

The @Component Annotation is used to create the bean by using the annotation configuration.

- It overrides the explicitly defining the bean in the XML file.
- To create an object of the class we just need to declare @Component annotation on top of the class.
- And if we are using the XML configuration then we need to declare the package in which the class exist using context name space.
    - <context:component-scan base-package="…." />

- The component scan will scan the whole package i.e. sub packages and classes exist inside the package and creates object.
- The classes that are annotated with @Component are managed by the Spring IoC container.
- In the place of @Component we can use @Repository, @Service, @Controller annotation. All these annotated classes ae managed by container.
- These annotations provide for semantic specifications for their particular usage.
- To create a beans of these annotated classes @ComponentScan annotation has there on top of the configuration class.
- @ComponentScan takes basePackages attribute to specify or gives metadata to the Spring IoC container to search and create classes that are annotated with the proper annotations.
- In XML based configuration also we have to provide the basePackage in the context name space.
    - <context:component-scan basePackage="…" />

@Inject is equivalent to @Autowired

@Named is equivalent to @Qualifier