**SPRING FRAMEWORK**

Spring is a *lightweight* framework. It can be thought of as a *framework of frameworks* because it provides support to various frameworks such as Struts, Hibernate, Tapestry, EJB, JSF, etc. The framework, in broader sense, can be defined as a structure where we find solution of the various technical problems.

The Spring framework comprises several modules such as IOC, AOP, DAO, Context, ORM, WEB MVC etc. We will learn these modules in next page. Let's understand the IOC and Dependency Injection first.

Spring Core

IOC & DI, AutoWiring, AOP, JDBC, Context

Spring MVC

Web MVC, ORM

Sprint Boot

RestFul Services, MicroServices

Graphical user interface

Description automatically generated

### **Test**

This layer provides support of testing with JUnit and TestNG.

### **Spring Core Container**

The Spring Core container contains core, beans, context and expression language (EL) modules.

#### **Core and Beans**

These modules provide IOC and Dependency Injection features.

#### **Context**

This module supports internationalization (I18N), EJB, JMS, Basic Remoting.

#### **Expression Language**

It is an extension to the EL defined in JSP. It provides support to setting and getting property values, method invocation, accessing collections and indexers, named variables, logical and arithmetic operators, retrieval of objects by name etc.

### **AOP, Aspects and Instrumentation**

These modules support aspect oriented programming implementation where you can use Advices, Pointcuts etc. to decouple the code(Service Oriented Programming).

The aspects module provides support to integration with AspectJ.

The instrumentation module provides support to class instrumentation and classloader implementations.

### **Data Access / Integration**

This group comprises of JDBC, ORM, OXM, JMS and Transaction modules. These modules basically provide support to interact with the database.

### **Web**

This group comprises of Web, Web-Servlet, Web-Struts and Web-Portlet. These modules provide support to create web application.

### **Inversion of Control (IOC) and Dependency Injection**

Inversion of control is a design principle which helps to invert the control of object creation. Inversion of control is the principle where the control flow of a program is inverted: instead of the programmer controlling the flow of a program, the external sources (framework, services, other components) take control of it. It's like we plug something into something else. an example about EJB 2.0.

**Inversion of control as a design guideline** serves the following purposes:

1. There is a decoupling of the execution of a certain task from implementation.
2. Every module can focus on what it is designed for.
3. Modules make no assumptions about what other systems do but rely on their contracts.
4. Replacing modules has no side effect on other modules.

**Dependency Injection** is a design pattern which implements IOC principle. DI provides objects that an object needs. Let’s say, class X is dependent on Y. So rather than creating object of Y within the class “X”, we can inject the dependencies via a **constructor or setter injection.**

**How to implement IoC**

In object-oriented programming, there are several basic techniques to implement inversion of control. These are:

1. using a factory pattern
2. using a service locator pattern
3. using a **dependency injection** of any given below type:
   * a constructor injection
   * a setter injection
   * an interface injection

| **Sr. No.** | **Key** | **IOC** | **Dependency Injection** |
| --- | --- | --- | --- |
| 1 | Design Principle | It is design principle where the control flow of the program is inverted | It is one of the subtypes of the IOC principle |
| 2 | Implementation | It is a term which is implemented by multiple design patterns  service locator , events , delegates and dependency Injection | DI is design pattern which can be achieved by constructor and setter injection |

They make the code easier to test and maintain. Let's understand this with the following code:

1. **class** Employee{
2. Address address;
3. Employee(){
4. address=**new** Address();
5. }
6. }

In such case, there is dependency between the Employee and Address (tight coupling). In the Inversion of Control scenario, we do this something like this:

1. **class** Employee{
2. Address address;
3. Employee(Address address){
4. **this**.address=address;
5. }
6. }

Thus, IOC makes the code loosely coupled. In such case, there is no need to modify the code if our logic is moved to new environment.

In Spring framework, IOC container is responsible to inject the dependency. We provide metadata to the IOC container either by XML file or annotation.

#### **Advantage of Dependency Injection**

* makes the code loosely coupled so easy to maintain.
* makes the code easy to test.

### **Advantages of Spring Framework**

There are many advantages of Spring Framework. They are as follows:

#### **1) Predefined Templates**

Spring framework provides templates for JDBC, Hibernate, JPA etc. technologies. So there is no need to write too much code. It hides the basic steps of these technologies.

Let's take the example of JdbcTemplate, you don't need to write the code for exception handling, creating connection, creating statement, committing transaction, closing connection etc. You need to write the code of executing query only. Thus, it save a lot of JDBC code.

#### **2) Loose Coupling**

The Spring applications are loosely coupled because of dependency injection.

#### **3) Easy to test**

The Dependency Injection makes easier to test the application. The EJB or Struts application require server to run the application but Spring framework doesn't require server.

#### **4) Lightweight**

Spring framework is lightweight because of its POJO implementation. The Spring Framework doesn't force the programmer to inherit any class or implement any interface. That is why it is said non-invasive.

#### **5) Fast Development**

The Dependency Injection feature of Spring Framework and it support to various frameworks makes the easy development of JavaEE application.

#### **6) Powerful abstraction**

It provides powerful abstraction to JavaEE specifications such as [JMS](https://www.javatpoint.com/jms-tutorial), [JDBC](https://www.javatpoint.com/java-jdbc), JPA and JTA.

#### **7) Declarative support**

It provides declarative support for caching, validation, transactions and formatting.

Maven Build Tool:- ( springs has to work on Maven Build Tool)

We need these jars :- spring-core, spring-context, spring-aop, spring-bean, spring-language, common-loggings

Coding :-

The main intention of the IOC Container is to read bean configurations from configuration file, creating bean objects and providing bean objects to spring applications.

A bean is a class which consist of private variables and it’s constructor as well as getter and setter methods.

Two types of Context for executing bean objects

1. BeanFactory
2. ApplicationContext

Bean Factory

It is the fundamental or base container provided by spring framework in order to manage bean objects.

BeanFactory IOC Container will provide basic functionalities to the spring fw by creating maintaining bean objects as per the beans configuration details which we provide in spring bean configuration file.

To represent beanfactory IOC Container, Sp Fw will be provided an interface in the form of “org.springframework.beans.factory.BeanFactory” and it is need implementation class in the form of “org.springframework.bean.factory.xml.XmlBeanFactory”

If we want to use beanfactory IOC Container, we have to follow below given steps

1. Create Resource Object
2. Create BeanFactory Object
3. Get Bean and access Business Method

Resource is an object in SpringFramework, it able to represent all bean configuration details which we provided in beans config details.

“org.springframework.core.io.Resource” (interface)

It is having class implementations : org.springframework.core.io.ClassPathResource

Ex:- Resource rs = new ClassPathResource(“beansinfo.xml”);

To get Bean

BeanFactory factory = new XmlBeanFactory(resourceObj)

BeanClassName Obj = (BeanClassName)factory.getBean(“bean id”);

ApplicationContext:-

It is also an IOC Container is an extension of BeanFactory, it able to provide some advanced features like Internationalization, Event handling, …. Along with fundamental functionalities what beanfactory is providing.

“org.springframework.context.ApplicationContext”

ClassPathXmlApplicationContext class

ApplicationContext <obj> = new ClassPathXmlApplicaitonContext(“config xml filename”);

There two ways to achieve dependency injection,

* 1. Using Constructor
  2. Using Setter Method

Using Constructor: -

If we inject dependent values to the Bean using constructor, then it is called as CDI.

To do this process, we have to create parameterized constructor in Bean Class then config file has to set as follows

<beans>

<bean id=”….” Class=”-----“>

<constructor-arg value=”----” />

</bean>

</beans>

Using Setter Method :-

If we inject dependent values to the Bean using setter methods(setXXX()), then it is called SDI

To inject primitive values and string value to the bean object then we have to use “value” attribute in <property> tag in config file.

If we want to inject User Defined Data types, that is Object reference values then we must use “ref” attribute in <property>

Class Account

Class Employee

{

Private Account accinfo;

}

Beanconfigfile.xml

<beans>

<bean id=”accid” class=”…..Account”>

</bean>

<bean id=”empid” class=”….Employee”>

<property name=”accinfo” ref=”accid” />

</bean> </beans>

Different types of Elements Injection

1. If we want to inject user defined data types then we have to use “ref” either in <property> or <constructor-tag> / we can also use <ref> nested tag under <property> or <constructor-tag> tag.

<property name=”….”>

<ref bean=”….” />

</property>

1. If we want to inject a **“List”** in a bean we have to set config file as follows

<property name=”….”>

<list>

<value>val-1</value>

……………………

</list>

</property>

1. If we want to inject a **“Set”** in a bean we have to set config file as follows

<property name=”….”>

<set>

<value>val-1</value>

……………………

</set>

</property>

1. If we want to inject a **Map** in a bean we have to set config file as follows

<property name=”….”>

<map>

< entry key=”key1” value=”value1” />

<entry key=”key2” value=”value2” />

……………………

</map>

</property>

1. If we want to inject a **“Properties”** in a bean we have to set config file as follows

<property name=”….”>

<props>

<prop key=”key1”>value-1</prop>

<prop key=”key2”>value-2</prop>

……………………

</props>

</property>

Class Student

{

String sid;

String sname;

Address adrs;

List<String> qualification;

Set<String> courses;

Map<String, String> course\_and\_faculty;

Properties course\_and\_cost;

getXXX();

setXXX();

void studentInfo()

{

Print “sid, sname, adrs, qualification, courses, course\_and\_faculty, course\_and\_cost”;

}

}

Class Address

{

String HouseNo, Streetname, city, country;

getXXX()

setXXX()

public String toString()

{

Return HouseNo + “\t” + Streetname + “\t” + city + “\t” + country;

}

}

1. **P-Namespace and C-Namespace**

At the time of bean creation in config file, it allows to send values to the setter methods / constructors.

**p-Namespace**

in general, in setter dependency injection, to specify values directly can send as attributes, to do this, follows steps

* 1. To use p-Namespace, in config file define “p” namespace in XSD as follows
     1. xmlns:p=[*http://www.springframework.org/schema/p*](http://www.springframework.org/schema/p)
  2. Bean attributes should be given as follows
     1. <bean id=”…” class=”….” p:prop\_name=”value” …. />
     2. <bean id=”…” class=”….” p:prop\_name-ref=”ref” …. />

**c-Namespace**

in general, in constructor dependency injection, to specify values directly can send as attributes, to do this, follows steps

* 1. To use c-Namespace, in config file define “c” namespace in XSD as follows
     1. xmlns:c=[*http://www.springframework.org/schema/*](http://www.springframework.org/schema/p)*c*
  2. Bean attributes should be given as follows
     1. <bean id=”…” class=”….” c:arg\_name=”value” …. />
     2. <bean id=”…” class=”….” c:arg\_name-ref=”ref” …. />

1. **Inner Bean: -**

In Spring framework, whenever a bean is used for only one particular property, it’s advise to declare it as an inner bean. And the inner bean is supported both in setter injection ‘property‘ and constructor injection ‘constructor-arg‘.

public class Customer

{

private Person person;

public Customer(Person person) {

this.person = person;

}

public void setPerson(Person person) {

this.person = person;

}

@Override

public String toString() {

return "Customer [person=" + person + "]";

}

}

public class Person

{

private String name;

private String address;

private int age;

//getter and setter methods

@Override

public String toString() {

return "Person [address=" + address + ",

age=" + age + ", name=" + name + "]";

}

}

Often times, you may use ‘ref‘ attribute to reference the “Person” bean into “Customer” bean, person property as following :

**Method-1**

<bean id="CustomerBean" class="com.mkyong.common.Customer">

<property name="person" ref="PersonBean" />

</bean>

<bean id="PersonBean" class="com.mkyong.common.Person">

<property name="name" value="mkyong" />

<property name="address" value="address1" />

<property name="age" value="28" />

</bean>

In general, it’s fine to reference like this, but since the ‘mkyong’ person bean is only used for Customer bean only, it’s better to declare this ‘mkyong’ person as an inner bean as following

<bean id="CustomerBean" class="com.mkyong.common.Customer">

<property name="person">

<bean class="com.mkyong.common.Person">

<property name="name" value="mkyong" />

<property name="address" value="address1" />

<property name="age" value="28" />

</bean>

</property>

</bean>

Bean Scope :- it will explain the scope of the bean created within the application.

* + 1. Singleton :- it is a default scope, Once this bean object is created IOC, that object will stored in cache memory. If we any no of times that object every time same memory will be used.
    2. Prototype
    3. Request scope
    4. Session scope
    5. Globalsession scope
    6. Application scope
    7. webSocket scope