SPRING FRAME WORK

Mail = real_time_programmer03@gmail.com
(if have any query mail I try to solve it)

Contents =

- **What is framework ?**

Framework is a semi developed software which provides some common logics which are required for several applications development.

Ex : Connection opening & closing, query execution, capture form data, sending email, generating excel report, generating pdf report, sending msg to mobile, copy data from one object to another object etc......

Frameworks will be released in the form of jar files

- **Advantages of frameworks**

1) Code Re-Usability

2) We can avoid boiler plate code

3) Less time for development

4) Less of no.of issues in project.

- **Dis-Advantages**

===============
1) When we use frameworks in project development, performance of application will be bit slow.

Note: When we compare advantages and dis-advantages of frameworks , advantages are more hence frameworks are having lot of demand in companies.

- **Types of Frameworks**

1) Web frameworks : Used to develop only web components

   Ex : Struts 1.x/2.x, JSF etc...

2) ORM frameworks : Used to developer Persistent components

   Ex : Hibernate, JPA, IBatis, MyBatis, TopLink etc...

3) Application Development Framework : Used to develop end to end project (web,service,persistent components).

   Ex : Spring

- **Layered architecture**

- Now a days every project is getting developed using layered architecture.
- Below are the layers that we can see in real time projects.

1) Presentation Layer
2) Web Layer
3) Service Layer
4) Persistence Layer

<u>1) Presentation Layer :</u>
Presentation layer contains user interface (UI)
Ex : Login form, registration, forgot password, search etc...

<u>2) Web Layer :</u>
a) In this layer we will develop web components.
b) Web components are also called as Request Handlers.
c) Request Handlers are also called as Controllers in frameworks.

3) Service or Business or Helper Layer:

As part of this layer we will write business logics like below

a) Server side validations
b) Password encryption and de-cryption
c) Sending Emails using java mail api
d) Sending OTP using twillo api
e) Generating Excel report using apache poi api
f) Genering pdf report using itext or aspose api.
g) Copying data from model obj to entity obj and vice versa.
h) Integration tier logics like soap clients and rest clients.

4) Persistence layer:

a) This layer contains persistent components
b) Persistent components are responsible for only Database communication.
c) Persistent components are called as dao s or repositories.

- **<u>Spring Framework</u>**

Author: Rod Johnson
Year : 2003
Initial Version : Spring 1.x
.Spring 3.x (So Many changes in Spring)
Current Version : Spring 5.x
Pivotal team

web frameworks : web components ----- struts

orm frameworks : persistence layer ---- hibernate

app development frameworks : Everything ---- spring

Spring Author : Rod Johnson

- **<u>Advantages of Spring Framework</u>**

1) Using spring we can develop both standalone and web apps.

2) Spring is not a single framework. It is collection of modules.

3) Spring will not force you to use all modules. Whatever is required for project take only that module from spring.

4) Spring is a Versatile framework. Spring can be integrated with any framework which is related to java.

5) Spring is a non-invasive framework. Spring will not force programmers to extend or implement any predefined classes or interfaces.

Note: Struts will force programmers to extend properties from Action class hence Struts is called as Invasive framework.

- **<u>Spring Architecture</u>**

Spring architecture changed from version to version.

Intial Version of Spring is 1.x. It contains total 7 modules.

Spring 2.x contains 6 modules.

Spring 3.x contains almost 20 modules. (So many annotations introduced) in 3.x version.

In Spring 4.x "Messaging" module got added

In Spring 5.x Reactive programming is added.

Current Version of Spring is 5.x. It contains approxmately 20-30 modules.

Spring 1.x Modules
----
1) Spring Core
2) Spring Context
3) Spring DAO
4) Spring ORM
5) Spring AOP
6) Spring Web and
7) Spring Web MVC

Note : In Spring 2.x version Spring Web and Spring Web MVC are combined as Single module hence we have total 6 modules in Spring 2.x version.

- **Spring Core**

-> This is base module for spring framework.

-> It is providing fundamental concepts IOC & DI in Spring framework.

IOC : Inversion of control

DI : Dependency Injection.

-> Using this core module we can develop stand-alone applications.
-> All other modules in spring are depeloped on top of spring core module.

Spring Context:

-> Spring context module deals with Configuration in Spring applications.
-> It is used for packages scanning and classes loading.

Spring DAO:

-> DAO means data access object.
-> It is used to develop persistence layer using spring framework.
-> This is also called as Spring JDBC.
-> This Spring DAO module is developed on top of java jdbc.
-> This module resolved boiler plate code writing.

Q) Can we call Spring JDBC as replacement for java jdbc ?

Ans) No, Spring jdbc is complementing java jdbc.

-> This is base module of Spring framework

-> This is providing IOC & DI.

Projects are used to reduce human efforts.

Project contains several classes.

All the classes will not play same role in project.

Based on role the classes are playing they can be divided into below 3 types

1) POJO
2) Java bean
3) Component

<u>POJO : Plain old java object</u>
A java class which can be compiled with only jdk software then we can call that class as a POJO.

POJO classes shouldn't extend or implement any api or framework related classes and interfaces.

Ordinary java class without any special effects.

These classes are used to transfer data from one layer to another layer in object format.

We can write business logic also in POJO classes.

Ex:

```
public class Demo{

}

public class Demo1{
   int i;
   int j;
   Double d;
   String name;
}
```

```
public class Demo2 implements Callable{
	//call()
}

public class Demo3 extends Thread{

}

public class Demo4 implements Serializable{

}

public class Demo5 implements Servlet{

}
```

Java Bean

The class which follows bean specification rules is called as java bean class.

Rules
-----
1) Class should be public
2) Class should contain public 0-param constructor.
3) Every variable should be private
4) Every variable should contain public setter & getter methods
5) Recommended to implement java.io.Serializable interface.

- > Java Bean classes are used to exchange data among multiple layers.

```
public class User implements java.io.Serializable{

	public User(){
	}

	private Integer id;

	public void setId(Integer id){
```

```
            this.id = id;
        }
        public Integer getId(){
            return id;
        }
}
```

Component

The classes which contains business logic are called as Component classes.

Ex : Controller, Service and DAO etc..

When we have several classes like above, all the classes are dependent on some other classes to complete execution.

Q) How one class can talk to another class in java ?

A) We have 2 ways to communicate from one class to another class.

1) IS-A (Inheritence)
2) HAS-A (Composition)

Inheritence

- Extending properties from one class to another class is called inheritence
- The main goal of Inheritence is re-usability

Dis-Advantages

- Classes will be tightly coupled
- Our class can't extend properties from any other class in future
- Code is not easily testable.

To solve Inheritence problems, we can go for Composition

Composition
The process of creating object for a class.

Composition Dis-Advantages
1) Classes are tightly coupled
2) Code is not easily testable.
3) For all classes we can't use new operator to create object.
Ex : abstract classes and singleton classes
4) Some classes will contain complex instantiation process
5) Sometimes we don't know class name to instantiate.

Factory Design Pattern
Design patterns are used to provide solutions for common problems which are occuring in several projects.
Calendar c = new Calendar() ; // invalid bcz it is abstract

Calendar.newInstance() ; //

- **Dependency Injection**

The process of injecting dependent object into target object is called Dependency Injection (DI).

We can achieve dependency injection in 2 ways.

1) Setter Injection (SI)

2) Constructor Injection (CI)

Setter Injection :
Injecting dependent object into target object by calling target class setter method.

```
Car c = new Car();
c.setEng(eng);
```

Constructor Injection :

Injecting Dependent Object into target object by calling target class constructor.

```
Car c = new Car(eng);
```

- **IOC : Inversion of control**

IOC is a principle which is used to manage and colloborate dependencies among objects in application.

Q) can ioc manage and colloborate any two class objects ?

Ans) No - If your classes fololIow some rules and guidelines then only ioc can manage and colloborate your objects.

If we need to use IOC for Dependency Injection for our classes then our classes should follow Strategy Design Pattern.

Strategy Design Pattern Rules
----

1) Favour composition over inheritence

2) Always code to interfaces instead of concrete implementations

3) Code should be open for extension and should be closed for modification.

We can start IOC Container in 2 ways

1)BeanFactory

2)ApplicationContext

```
BeanFactory factory = new XmlBeanFactory(Resource res);
```

ClassPathResource
FileSystemResource

Both are interfaces only.

```
BeanFactory  factory = new XmlBeanFactory(Resource resource);

String fileName="Beans.xml";
Resource resource = new ClassPathResource(fileName);
BeanFactory factory = new XmlBeanFactory(resource);
```

```
//loading xml file from classpath
Resource resource  = new ClassPathResource("Beans.xml");

BeanFactory factory = new XmlBeanFactory(resource);
```

```
//loading xml file from file system
Resource resource =
          new FileSystemResource ("D://Config/Beans.xml);
BeanFactory factory = new XmlBeanFactory(resource);
```

<id>100</id> ------> Simple Element

```
<person>
        <name>John</name>
</person>
```

----
name ------ > simple element
person -----> compound element

```
<?xml version="1.0" encoding="UTF-8"?>
<Book>
   <name>Spring</name>
   <author>Rod Johnson</author>
</Book>
```

----
name & author -----> Simple Elements
Book -------------->Compound Elements

1) Wellformness : Will talk about can we read that xml or not?
2) Validness : Will talk about can we use that xml or not?

1) Wellformness : can we read it or not ?

a) XMl Should contain only one root element
b) Every start tag should contain end tag.

2) Validness : can we use it or not ?

XML validitity will be decided by dtd or xsd.

DTD : Only structure validation

XSD : Structure + Data type validation

Parser : A program which reads xml file data in xml style.

xml style : ignoring meta data and reading actual data.

--------JAX-P API-------------------
SAX
DOM
STAX

Beans.xml

```
<beans>

   <bean   id="a"   class=""> --- Bean Definition

   <bean id="b"  class="" >  -- - Bean Definition

</beans>
```

Resource

ClassPathResource

FileSystemResource

1) BeanFactory

```
          Resource resource =
                    new ClassPathResource("com/maruti/cfg/Beans.xml);

          BeanFactory factory =
                              new XmlBeanFactory(resource);
```

1) Loads Beans.xml file from given location

2) Give that xml file as input to XmlBeanFactory

3) Parser check will for welformness and validness

4) BeanFactory will load all beans defintions from xml file and will store BeansMetaData (in memory).

Code :

```
Resource res = new ClassPathResource("com/cfg/Beans.xml");
BeanFactory factory = new XmlBeanFactory(res);
```

Constructor Injection:

INjecting dependent class object into target class by calling target class constructor is called as Constructor Injection(CI).

```
public class Car {

   private IEngine  eng=null;

   public Car(DieselEng eng){
        this.eng = eng;
   }
}

new Car(); //invalid
new Car(new PetrolEngine());
```

Bean=

```
<bean id="car"  class="pkg.Car">
  <constructor-arg name="eng" ref="petrolEng" />
</bean>
```

Difference between setter method and constructor injection :

1) **setter injection**

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

name - variable name
ref - bean id

target obj will be created first in SI

partial dependency injection is possible in SI

2) **constructor of target**

<constructor-arg name="" ref="" />

dependent obj will be created first in CI

Partial Dependency is not possible in CI.

- **Collection Injection**

List
Set
Map and
Properties

LIST =
private List<String> places; //java.util.ArrayList

```
<property name="places">
   <list>
          <value>Hyd</value>
          <value>Pune</value>
   </list>
</property>
```

SET=

```
private Set<Long> phnos; //java.util.LinkedHashSet

<property name="phnos">
  <set>
          <value>0808080</value>
          <value>9797979</value>
  </set>
</property>
```

MAP =

```
private Map<String,Long> projectCodes; //java.util.LinkedHashMap

<property name="projectCodes">
          <map key-type="" value-type="">
                    <entry key="" value="" />
                    <entry key="" value""/>
          </map>
</property>
```

```
private Properties emails;

<property name="emails">
 <props>
          <prop key="personal">iyiy</prop>
          <prop key="ofc"></prop>
 </props>
</property>
```

Enumeration
Iterator
ListIterator
SplitIterator (jdk 1.8)

Collections are mainly used for storing the data
Streams are used for performing operations on collection data

Note: In Constructor Injection, its mandatory that we need to inject values for all the variables.

If we don't know value for a variable, then we shuld use null injection.

```
<constructor-arg name="movieName">
        <null />
</constructor-arg>
```

- **Bean Scopes**

Bean Scope will decide how many objects should be created for a bean class in IOC container.

```
//obj will be created
Movie m1 = factory.getBean("movie",Movie.class);

//obj creation depends on bean scope
Movie m2 = factory.getBean("movie",Movie.class);
```

Spring supports below 4 types of scopes

1) singleton (by default)
2) prototype
3) request
4) session

syntax :

```
<bean id="car"
      class="pkg.Car"
      scope="singleton|prototype|request|session" />
```

Singleton :
Only one object will be created.

```
<bean id="car" class="pkg.Car" />
<bean id="car" class="pkg.Car" scope="singleton"/>
```

Prototype :
For every call of getBean() method new object will be created.
```
<bean id="car" class="pkg.Car" scope="prototype"/>
```

```
BeanFactory factory = new XmlBeanFactory(resource);
factory.getBean("movie",Movie.class); //obj will be created
factory.getBean("movie",Movie.class); //
```

```
<bean id="" class="" />
```

- **BeanFactory**

It is an interface
It is used to start IOC Container
IT is Lazy Container

Why it is called as Lazy ? ::

- Until unless we call getBean( ) method bean object will not be created.
- how many objects should be created for a bean depends on bean scope.

It doesn't support for I18N.
It doesn't support for Event Handling

It doesn't support for WebApplication Development.

- **<u>ApplicationContext</u>**

It is an interface
It is also used to Start IOC Container
It is Eager container

Why it is Eager Container ? ::
When IOC starts immediatley it will create bean objects if scope is singleton and lazy-init=false.

It supports I18N
It supports Event Handling
It supports Web Application Development

```
ApplicationContext ctx =
                   new ClasspathXmlApplicationContext(String filePath)
```

Note : scope attribute default value is singleton
       lazy-init attribute default value is false.

```
<bean id=""  class=""  scope="" lazy-init="true" />
```

- **<u>Injecting Inner Beans</u>**

One bean object can be injected to another bean in 2 ways

1) Style-1 (Re-usability will be available)

```
<bean id="a" class="pkg.Demo" />

<bean id="b" class="pkg.Demo1">
  <property name="a" ref="a"/>
</bean>
```

2) Stype-1 (No Re-Usability)

```
<bean id="b" class="pkg.Demo1">
  <property name="a">
        <bean class="pkg.Demo" />
  </property>
</bean>
```

Note : We can't create object for inner beans directly.

- **<u>Bean Inheritence</u>**

What is Inheritence?

- Extending Properties from one class to another class is called Inheritence.
- The main goal is re-usability.
- It is also called as IS-A relationship.

Bean Inheritence
To copy properties from one bean to another bean we can use Bean Inheritence concept in Spring.

Note: If property is not declared in child bean definition then only IOC will copy parent bean property value to child bean property.

```
public class Car {
   int id;
   String name;
   Double price;
   String color;
   String regNum;
   //setters
   //toString()
}
-------------------------------------------------------------------
<bean id="c1" class="pkg.Car">
  <property name="id" value="101"/>
  <property name="name" value="Swift" />
  <property name="price" value="850000"/>
  <property name="color" value="Red"/>
  <property name="regNum" value="TG 27 ES 6868"/>
</bean>


<bean id="c2" class="pkg.Car" parent="c1">
  <property name="id" value="102"/>
 <property name="regNum" value="TG 27 ES 6862"/>
</bean>

<bean id="baseCar" class="com.maruthi.beans.Car" abstract="true">
          <property name="name" value="Swift" />
          <property name="price" value="850000" />
          <property name="color" value="Red" />
</bean>

<bean id="c1" class="com.maruthi.beans.Car" parent="baseCar">
          <property name="id" value="101" />
          <property name="regNum" value="TG27 ES 6861" />
</bean>
-------------------------------------------------------------------
```
-> It is recomended to configure base bean with common properties
-> if we give abstract=true in bean definition , that bean definition will be abstract hence object will not be created.
-> If we give abstract=true, class will not become abstract.

- **<u>Collection Merging</u>**

If we want to copy collection properties from one bean to another bean then we can use Collection Merging.

Note: Collection Merging will work only in Bean Inheritence scenario.

Meeting :

id = 101
name = Triage Meeting
purpose = To discuss defects occured in UAT
participants = john, Ram, Sita, Gita

- Scrum Meetings
- Status Meetings
- Triage Meetings
- All Hands Meeting

- **<u>Manual Wiring</u>**

Injecting dependent object into target object using ref attribute.

- **<u>AutoWiring</u>**

IOC container can perform auto wiring.

IOC can identify dependent object and it can inject into target object.

By Default Autowiring will be in disable mode. If we want to use Autowiring, we should enable it.

Autowiring should be enabled on Target Bean.

syntax : <bean id="" class="" autowire="byName|byType|constructor" />

byName:

- With target class user-defined variable name there should be a bean in bean configuration file. Then that bean will be considered as dependent bean and it will be injected to target bean.

- If we don't have any bean defintion which is matching with target class user defined type variable then autowiring can't be performed.

Note : IOC will check varibale name with bean id or bean name.

byType:

- IOC will check user-defined varible data type with bean class in bean configuration file. If variable data type and Bean class is same then ioc consider that bean as dependent bean and it will inject that bean to target.

Note : There is a possibility that one class can be configured with multiple bean definitions using unique ids. In this sitatuion IOC can't identify dependent bean hence it will throw Exception (NoUniqueBeanDefinitionFound).

- To resolve this we can use 'primary' attribute in bean defition. The bean which contains primary=true is called as auto-wire candidate.

byName - variable name should match with bean name

byType - variable data type should match with bean class

<u>constructor</u>

no ------------- > no auto-wiring

```
<bean id="" class="" autowire="byName | byType | constructor" />
```

constructor
IOC
DI (SI & CI)

```
<bean id=""
      class=""
      scope=""
      parent=""
      abstract=""
      lazy-init=""
      autowire="" >

          <property />
          <constructor-arg />

</bean>
```

- **What is Spring Bean?**

The class whose life cycle is managing by IOC then that class is called as Spring Bean.

```
<bean id=""
      class=""
      scope=""
      parent=""
      abstract=""
      lazy-init=""
      autowire="" >

          <property name="" value or ref />
          <constructor-arg />

</bean>
```

- **P & C space :**

```
class Contact {

  int contactId;
  String contactName;
  Long contactNumber;
  Address addr;

  //Setter methods

  //toString( )

}
----------------old approach---------------------------
<beans <xsd linking> >
<bean id="c" class="pkg.Contact">
          <property name="contactId" value="101" />
          <property name="contactName" value="Raj"/>
          <property name="contactNumber" value="797979979"/>
          <property name="addr" ref="a"/>
</bean>
</beans>
---------------------using p-namespace-----------
<beans xsd-linking  p-namespace-url >

   <bean id="c" class="pkg.Contact"
                    p:contactId="101"
                    p:contactName="Raj"
                    p:contactNumber="7797979"
                    p:addr-ref = "a" />
<bean>
=================================================================
===============
<bean id="contact" class="com.pc.beans.Contact"
          p:id="101" p:name="Raj"
          p:number="9797979799" p:addr-ref="address" />
```

```
<bean id="address" class="com.pc.beans.Address"
c:city="hyd" c:state="TG" c:country="India" />
```

- **Depends-On**
  If one beam is indirectly depend on another bean then we go for depends on

```
class Car {

   private Engine eng;

}

class Robot {

  private Chip chip;

}
------------------------------------------------------------
<bean id="c" class="pkg.Chip" />

<bean id="r" class="pkg.Robot">
   <property name="chip" ref="c"/>
</bean>
------------------------------------------------



<bean id="emiCalc" class="pkg.EmiCalculator"
          depends-on="cacheManager"/>
```

- **Bean Aliasing**

<bean name="c1,c2 c3" class="com.maruthi.Car" />

Car c1 = ctxt.getBean("c1",Car.class);

Q)what is the difference between id and name attributes in bean tag?

- Using id we can configure only one value
- Using name we can configure more than one name

Note : When we configure multiple names for a bean using name attribute space and comma will be considered as delimter.

If you want write a bean name including comma or space then we should use alias tag.

<alias name="res1" alias="res5," />

Amazon - E-Commerce
BlueDart
Cache memory and cache manager

Bean Aliasing

id vs name

<bean id="car" class="" />

<bean name="c1,c2,c3 c4" class="" />

<alias name="c1" alias="c5," />

ApplicationContext

Alias tag in config file

- **<u>Aware Interfaces</u>**

- This concept is also called as Interface Injection.

- Dependency Injection is performing through interface hence it is called as Interface iNjection.

Ex : BeanFactoryAware and ApplicationContextAware etc...

- If we don't know which dependent object should be injected into target at compilation time then we can inject IOC to our target.

- If IOC is injected to our target class, then target class can decide which dependent object it should load in runtime based on conditions.

- **<u>BeanFactoryAware</u>**

```
class AmazonService implements BeanFactoryAware {

private BeanFactory factory = null;

public setBeanFactory(BeanFactory factory){
        this.factory = factory;
}

        //logic
}
```

- **<u>Factory Methods</u>**

The method which is responsible to creat same or different class object is called as factory method.

Factory methods can be classified into 2 types

1) Static factory method

```
        Connection con = DriverManager.getConnection(url,uname,pwd);

        Calendar c = Calendar.getInstance();
```

2) Instance Factory Method

```
                String msg = "hello";
                String sub = msg.substr(0,3);

                char ch = msg.charAt(0);
```

```
class Car {

}

<bean id="c" class="pkg.Car" />
-------------------------------------------

<bean id="al" class="java.util.ArrayList" />

-------------------------------------------

<bean id="calc" class="java.util.Calendar"
                factory-method="getInstance" />


Calendar c = Calendar.getInstance( ) ;
```

================Static Factory method Instantiation================

```
<bean id="cal"
     class="java.util.Calendar"
     factory-method="getInstance" />
```

12-May-2019 + 57 = ?

```
addDays(Date d, int days) {
        cal.setTime(d);
        cal.add(Calendar.DAY,days);
        Date finalDate = cal.getTime();
}
```

new Date()

String str = "10-May-2019" ;

SimpleDateFormat

Date parse(String str) -----> String to Date conversion

String format(Date d) ------> Date to String Conversion

- **Method Injections**

1) **LookUp method Injection :**
   When we want to inject prototype bean into singleton bean

2) **Method Replacement :**
   If we want to replace one method with another method in runtime.

```
<bean id="t" class="pkg.Token" scope="prototype" />

<bean id="tokenGenerator" class="pkg.TokenGenerator">
        <property name="token" ref="t"/>
</bean>
---------------------------------------------------------
TokenGenerator tg =
        ctxt.getBean("tokenGenerator",TokenGenerator.class);
```

tg.getToken(); // 1234

tg.getToken( ); //1234

- **<u>I18N (INTERNATIONALIZATION)</u>**

- When we want to develop a application which is support the all types of language or all types of the region we go for internationalization .
- **Location** : location is a geographical location or area
- we specify language object by the languageCode_CountryCode
  like hindi = HI_IN

- **<u>Property editor</u>**

- When we use steer injection that time we just specify the value attribute or ref attribute then how the IOC know what is the data type of that value. That can manage by the property editor .
- Property editor convert the value we specify by setter injection to there corresponding data type.
- We also create our own property editor .

- **<u>How the garbage collector work internally</u>**

- Garbage collector use the following 3 stapes .

1. Mark
2. Sweep
3. Compaction

   Mark =
   mark means the garbage collector first check the objects or fields which doesn't have an any reference address and mark that field or object.

   Sweep =
   sweep means that after marking the object garbage collector delete or remove that from memory.

Compaction =
 it means after removing that field there is holes are create and there is more space is between two objects. After sweeping the garbage collector go for compaction means it adjust all object remove the holes and make space is free for next objects.