Java means DURGA SOFT..

JAVA FRAMEWORKS

Syllabus:

MODULE-1(SPRING - LOC)

Why sprint ?

1. What about spring framework
2. Spring Modules:
3. Spring Core:
4. Spring J2EE module:
5. Spring DAO module:
6. Spring ORM:
7. Spring web MVC:
8. SPRING CORE MODULE (SPRING IOC)
9. Type of Dependency Injection:
10. Spring Framework Installation:
11. Drawback of inner Beans
12. Dependency in the form of collections
13. Set Collection
14. List Collection
15. Constructor injection
16. Circular Dependency
17. Bean autowriting
18. Applicationl context interface
19. Bean initializing and destruction
20. BeanScops in springFw
21. Bean instantiation

MODULE-2(SPRING-JDBC (DAO))

1. DAO DESIGN PATTERN:
2. Methods of jdbc template:

MODULE-3(SPRING-ORM (Business layer))

MODULE-4 (SPRING-AOP(Aspect Oriented Programming))

MODULE - 5, SPRING WEBMVC MODULE

1. Spring MVC InternalResourceViewResolver Configuration Example
2. Spring MVC 4.0 RESTFul Web Services Simple Example
3. Spring Restful Web Service Example with JSON, Jackson and Client Program

SPRINGS

Why Spring?

➔ In j2ee application development, we got some component technologies and servlet technologies.

➔ By using the component and service technologies of j2ee, we can develop a good enterprise business app's.

➔ Component technologies are

1) Servlet 2) Jsp 3) EJB

➔ Service technologies are

1) JNDI (java naming and directory interface)
2) JMS (java messaging service)

3) Java Mail
4) JTS
5) JAAS (java authentication and authorization service)
6) JCA (java EE connector architecture)

➔ We use servlets and JSP technologies of J2EE for the development of web application.

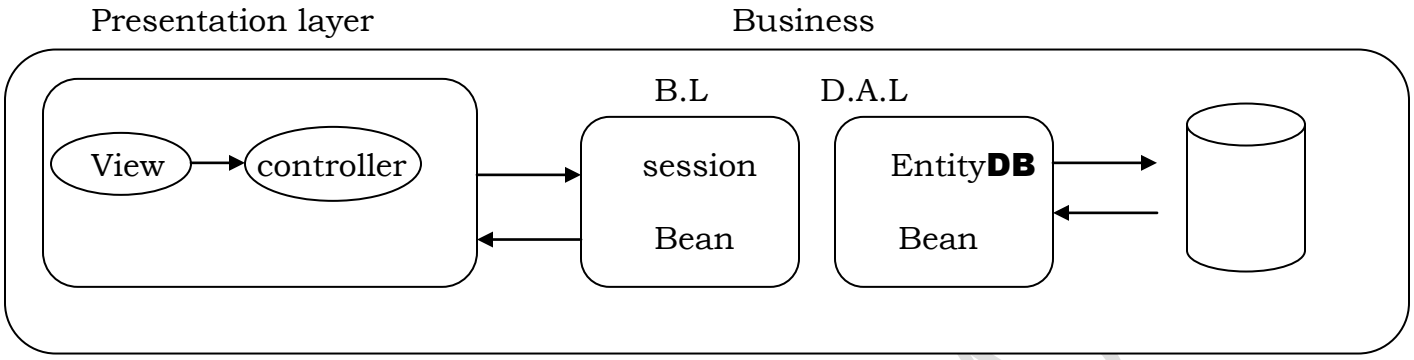
➔ We use EJB technology for the development and constructing of Enterprise applications.

➔ While developing Enterprise application through EJB, we got the following problems.

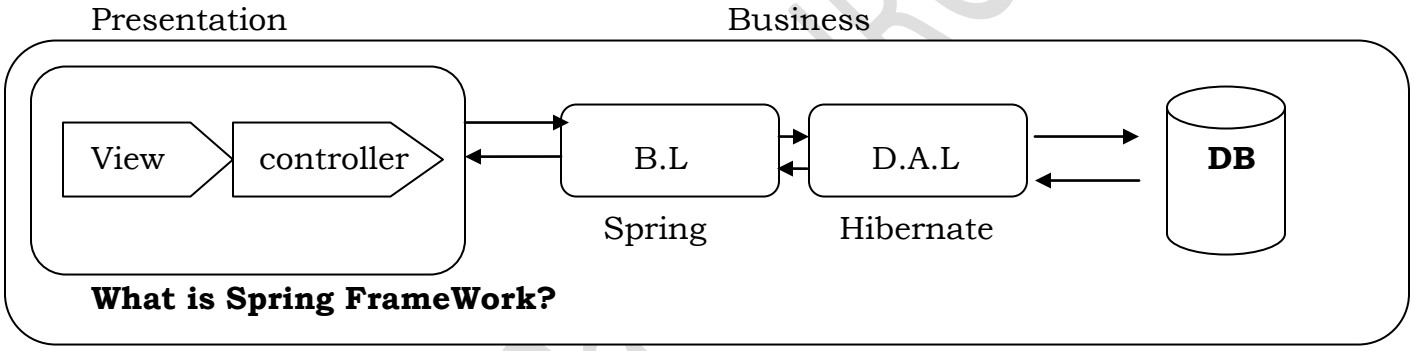
1) EJB component development requires multiple java files and xml files. So, it makes burden on the programmer (or) developer.
2) EJB's cannot run without out a container/ server. So, to test an EJB a server is mandatory.
3) While testing an EJB, if it fails, then we need to shutdown the server and after modification and once again we need to restart the server and we need to test application. It is a time consuming process.
4) EJB's are heavy weight components and having a lot of dependency with the server.

➔ In order to overcome the above problems occurred at EJB, we got an alternative for EJB technology as 'Spring Frame Work'.
Note: Spring Frame work is not a replacement for EJB and it is just alternative.

Before Spring:



After Spring:



What is Spring FrameWork?

➔ Spring is a lightweight and an open source frame work created by **Rod Johnson** in 2003.

➔ Spring is a complete and a modular Frame work. It means spring frame work can be used for all layers implementation for a real time application (or) spring can be used for the development of a particular layer of a real time application.

➔ Spring framework is said to be a non-intrusive framework. It means spring framework does not force a programmer to extend (or) implement their classes from any predefined class (or) interface given by Spring API.

➔ In case of struts f/w. It will force the programmer that the programmer class must extend from the base class provided by struts API. So, struts we call them as invasive or intrusive frame work Spring is called light weight component because of POJO model.

- Spring framework simplified J2ee application development by introducing POJO model.

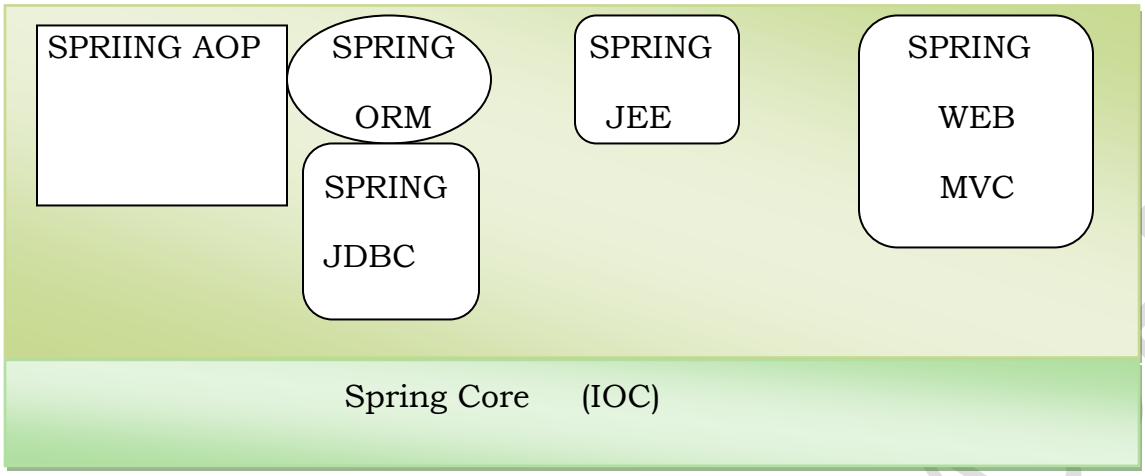
- Spring is important because of following 3 reasons.
  1) Simplicity,
  2) 2) feasibility,
  3) 3) loose coupling.
- Spring provides simplicity , because of POJI/POJO interface Model.
- Spring is easy to test, because. Spring can be executed within a server and even without a server.
- Spring provides loose coupling because of IOC (Inversion of control) mechanism.
- Spring framework can be used to develop any kind of java application. It means we can develop starting from a console application up to a enterprise level application.
- Unlike other framework, spring framework has its own container. So, spring works without server also.

Spring Modules:

In spring 1.x, the framework has divided into 7 well defined modules, but in spring 2.x the framework is divided into 6 modules only.

i. Spring Core Module
ii. Spring Context (spring jee module)
iii. Spring DAO (spring jdbc) module
iv. Spring ORM module.
v. Spring AOP module.
vi. Spring web mvc module.
vii.

Spring Modules Architecture:



Spring Core:

Struts framework highlevel object is created→ActionServlet

Hibernate highlevel object is created→ SessionFactory

Springframe work highlevel object is created→ BeanFactor

- This module provides the basic functionality required by a spring application.
- Spring core module generates an high level object of the spring framework called BeanFactory.
- Spring core module applies IOC/ dependency injection, for generating for spring IOC container called as BeanFactory.

- All framework will follow a commonality called, an high level object is created first and it will take care about low level objects required for the application.

  Struts Framework -> ActionServlet
  Hibernate Framework -> Session Factory
  Spring Framework -> BeanFactory.

Spring J2EE module: spring jee module is an extension to core module, where core module creates container and jee module makes it as framework.

- Spring jee module acts real time services like e-mailing, jms, timer, jndi, remoting, j18n, etc.., to this spring application.

Spring DAO module: This module is providing an abstraction layer on top of existing Jdbc technology and it avoids the repeated Jdbc code from an application (boilerplate code).

- The another advantage of this module is that we need to handle Exception while working with the db because in spring all exceptions are **Unchecked Exceptions**.

Spring ORM: This module is also to work with Database, spring ORM module provides an abstraction layer on top of existing ORM tools.

When compared with spring DAO, Spring ORM module has two benefits.

a. Data transfer will be done in the form of objects.
b. ORM tools always throws unchecked exceptions.

- Even in ORM tools also, boilerplate code (or) repeated code is required. So, in spring ORM module, spring framework has provided an abstraction layer and this layer avoids the use of boiler plate code.
- By without using ORM module, spring can directly communicate with iBATIS, JDO …etc.

- In a real time application, Business Logic need additional services.
- If we combine implement business logic along with services then we will get the following problems.
  a) The size of the application increases, so that complexity increases.
  b) If any modification is required in a service then in each business method we need to do modifications separately.

- To get out of the above problems , we got accept oriented programming in spring.
- With AOP, Business logic and the required services both are separated. Whenever service is required then it will be injected to the logic at run time.
- Spring AOP meaning is separation of business logic with its services is required.

Spring web MVC:

- In this module, we do the following two operations.
  a) We integrate our spring framework with outside mvc frameworks like struts, JSF, …etc.,.
  b) We can develop complete mvc application by without using any outside framework.

SPRING CORE MODULE (SPRING IOC)

Tight coupling and loose coupling between objects:

- When developing java applications, generally one object will collaborate with another object, for providing services to the clients.

- If one object is communicating with another object, i.e., if one object is calling the business method of another object then we can say there is a coupling between two objects.
- In object to object collaboration, one object becomes dependent object and the other object becomes caller object.
- For eg: if Traveler and Car are the two classes and Traveler object is called dependent object and Car object is called caller object.
- When two classes are collaborating, if the first class is getting all the required objects (dependencies) by itself then we call there is tight coupling between the two classes (objects).

For example:

```
Class Traveller          class car              class whell
{                        {                      {
Car c.newCar()           wheel w.new wheel()    void rotate()
Void startjourny()       void move()            {
{                        {                      }
c.move();                logic w.rotate()
}                        }
}
```



```
    }
```

In the above example, Traveler Object is depending on Car Object. So, Traveler class creating an object of Car class inside it.

- If directly the object is created in the dependent then there exists tight coupling.
- According to the above example there is a tight coupling between Traveler and Car object.
- If the method in Car class is changed from move() to run() then, in Traveler class also the changes are required that is in startJournery() instead of calling move(), we need to call run().

Eg:

```
class Traveller
{
Car c=new car()
Void starJourney()
{
C.run()
}
}
```

```
Class car

{

Void run()

{

}

}
```

- Tight coupling between object means, when two objects are collaborating, if any changes are done on one object then the changes are required on other object also.
- Loose coupling: In order to overcome tight coupling between objects spring framework, uses dependency Injection mechanism.
- With the help of POJI/ POJO model and dependency Injection, it is possible to achieve loose coupling.
- In the above example, Traveler and Car are tight coupled. If we want to achieve loose coupling between the objects, Traveler and Car, we need to rewrite the application like the following.

Eg

```
Class   Traveller
{
Vehicle v;
Public void setV(Vehicle v)
{
This.v=v;
}
Public void startJourny()
{
v.move();
```



```
}
}
Interface Vehicle
{
Void move();
}
Public class Car implements Vehicle
```

```
{
Public void move()
{
System.out.println("…….");
}
}
Public class MotarBike implements Vehicle
{
Public void move()
{
System.out.println("……");
}
}
```

In the above example, through dependency Injection mechanism, Spring container will initiate either Car object or MotorBike object into Traveler by calling setter(). So, if Car object is replaced with MotorBike object then no changes are required in Traveler class. It means there is a loose coupling between Traveler Object and Vehicle Object.

Type of IOC:

Inversion of control means, separating the implementation of an object from how the object is constructed.

- In case of spring framework, the framework will take care about object creation and _____ the dependent of object required. As a programmer we are responsible for only implementation of the business logic.
- There are two types of IOC a) Dependency lookup b) Dependency Injection.

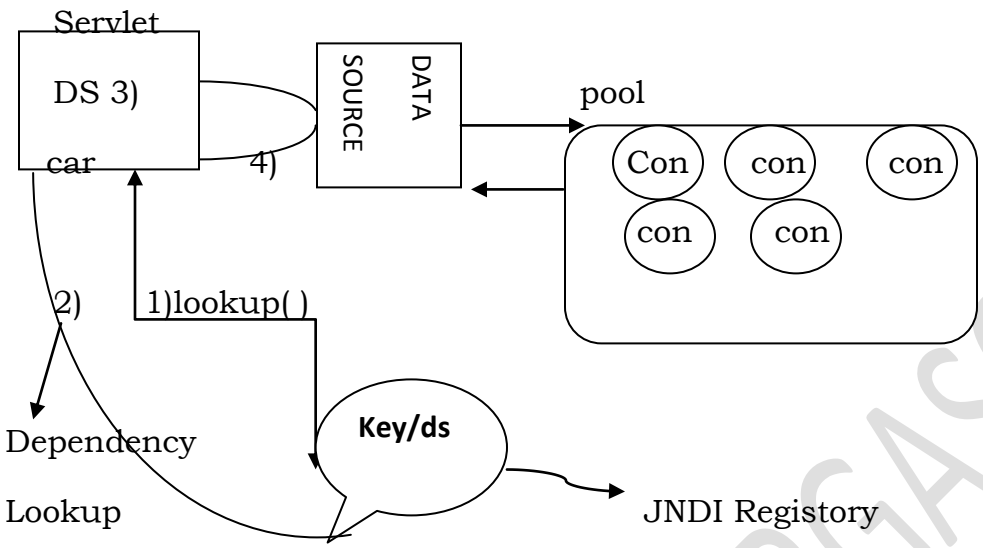
Note: taking is called lookup and giving is called injection.

Eg: connection pooling in Servlet.

The servlet will go in to the pool and take the connection, this is called Dependency lookup mechanism.

Whereas in spring, spring container will provide the object that means giving the objects automatically this is called Dependency Injection.

→as developer concentrating business logic but creating the objects.



Dependency lookup:

- In this type of IOC, when one object is collaborating with another object, the first object has to get all the dependencies required explicitly by itself.
- In dependency lookup, an external person does not provide collaboration objects automatically to the first object.
- In dependency lookup of IOC, the first object is responsible for obtaining its collaborating objects from the container (or) repository etc.,

Eg: In servlet with connection pooling, servlet object depends on datasource object. In this case nobody automatically gives data source object to servlet object.

- Servlet itself goes to individual registry and takes DataSource object from the registry by performing lookup operation as shown in above diagram.

Eg: In servlet to EJB communication servlet depends on EJB object for calling the business logic implemented in an EJB. In this communication nobody will provide EJB object to servlet object automatically.

- Servlet itself goes to EJB container and gets the EJB object from it. It means there is a dependency lookup.

Dependency Injection:

- In this type of, when one object is collaborating with another object, some external person will be provide, the collaborate object to dependent object automatically.
- In this type of IOC, the first object is not responsible to explicit take the collaborating object required. It means the first object (dependent) does not perform any lookup operation.
- In case of spring framework, the external person who performs dependency Injection is called Spring IOC container.
- EJB3.x technology and spring framework both are supporting dependency Injection type of IOC.


LEARN FROM EXPERTS ...
COMPLETE JAVA
CORE JAVA, ADV. JAVA, ORACLE, STRUTS, HIBERNATE, SPRING, WEB SERVICES,...
COMPLETE .NET
C#.NET, ASP.NET, SQL SERVER, MVC 5 & WCF
TESTING TOOLS
MANUAL + SELENIUM
ORACLE
D2K
MSBI
SHARE POINT
HADOOP
ANDROID
C, C++, DS, UNIX
CRT & APTITUDE TRAINING
AN ISO 9001:2008 CERTIFIED
DURGA
Software Solutions®
202, 2nd Floor, HUDA Maitrivanam,
Ameerpet, Hyd. Ph: 040-64512786,
9246212143, 8096969696
www.durgasoft.com

```
Class A {
B ab;
public  A(B ab){}          // constructor
public void setOb(B ab)       //setter
{
This.ab = ab;
}
public void m1()
{
    do.m2();
}
}
```



```
Class B {


}
```

Type of Dependency Injection:

1. Setter Injection
2. Constructor Injection
3. Interface Injection

- In spring framework, we use only setter and constructor injections, but not interface injection.
- In struts 2.x, we have interface injection.

Setter Injection:

In this type of dependency Injection, the spring container, uses setter() in the dependent class for injecting its dependencies (collaboration).

- Spring container knows whether to perform setter or constructor injection, by reading the information from an external file called as spring configuration file.
- In case of setter injection, the class must contain a setter () to get the dependencies, otherwise spring container does not inject the dependencies to the dependent object.

Eg: class A {

```
B  ob;

public void setOb(B ob){

this.ob = ob;
```

```
        }
        public void m1() {
        -----
        ------
        }
}

        Class B {
        public void m2(){
        ------
        }
}
```

- In the above class A is called dependent and class B is called _____ or collaborator.
- In the dependent class there is a setter() for getting collaborate object. So we called as setter injection.
- In spring framework, we call each class as a springBean. This spring bean no way related with java bean.

- *** SpringBean and JavaBean are not same, because a java bean need a public default constructor, but in spring bean sometimes we include default constructor and sometimes we do not.
- In SpringBean classes, there is exist the following 3 types of dependency values.
  I. dependency in the form of primitive value. Eg: int k;
  II. dependency in the form of object. Eg: FormBean fb;
  III. dependency in the form of collection. Eg: List as;

Eg:
```
public class TestBean{

private int x;          //primitive

private SampleBean sb;          //object

private List al;        //collection

----

----

}
```


Dependency in the form of Primitives:

- The spring container understands whether dependency in the form of primitive or not, whether setter or constructor injection is required..,etc., information by reading spring configuration file.

- Spring configuration file identified with any .xml
- If the dependency is in the form of primitive then we can directly provide in the xml file for that property.

Eg: Class DemoBean {

int k;

public void setK(int k) {

this.k = k;

}

public void display() {

System.out.println(k);

}

}

Sprconfig.xml

<! DTD --- >

<bean>

<bean id="id1" class="DemoBean">

<property name="k">

<value> 100 </value>

</property>

</bean>

- In spring configuration file, we used <property> element, so spring container understands that there is a setter injection in the bean class.
- Inside <property> element, we have used <value> element. So spring container understands that the dependency is in the form of primitives.
- In spring configuration file, we can use value as a sub element of property tag or value, as an attribute of the property tag.

Eg: <property name = "k" value="100" />

Note: If we pass a value from an xml file, then we can change that value whenever it is required by without reopening our java code. It means we can directly modify the value in xml file and we can run the java code by without any recompilation of code.

Xml are used to change the value instantly.

Steps to write a spring cline application:

Step1:

a. spring environment start by loading spring configuration file into Resource object.
b. We call this step1 as a bootstrapping of springframework.

c. Resource is an interface. ClasspathResource is an implementation class given by springframework.
d. Both Resource interface and classpathResource are given org.springframework.core.io package.
   Syntax:
   Resource res = new ClasspathResource(sfconfig.xml)

Step 2:

a. Spring IOC container object will be created by reading Bean definitions from the Resource object.
b. Spring IOC container is called BeanFactory and this container is responsible for creating the Bean object and its assigning its dependencies.
c. BeanFactory is an interface and XmlBeanFactory is an implementation class of it.
d. BeanFactory is an interface given in org.springframework.beans.factory and XmlBeanFactory is a class given in org.springframework.bean.factory.xml package

Syntax:

BeanFactory factory = new XmlBeanFactory(res);

Step 3:

a. Get the Bean object from the spring container by calling getBean()
b. While calling getBean(), we need to pass the Bean Id as a parameter.

Step 4:

a. GetBean() always returns object. We need to typecast the object into our spring bean type.

```
Object o = factory.getBean("id");
DemoBean db = (DemoBean)o;
```

Step 5:

Call the business method of the springBean by using the object.

```
db.display().
```

Spring Framework Installation:

- Working with a framework software is nothing but working with set of java files given by that framework.
- If we want to use spring framework in a java client application then we need to set the framework jar files into classpath.
- Spring framework is distributed as a combined jar file for all modules and also individual jar files for each module.
- Spring framework is initially released by interface 2.1 and now it is renamed as springsource.org
- Visit www.springsource.org and download spring framework 3.0.5 zip and then extract it to get the jar files.
- The main jar file of spring is called spring.jar and it depends on commons.logging.jar
- Spring.jar file exits e:\spring-framework-3.0.5\dist folder
- we can set individual jar files modules also into the classpath. These modules jar file are available in

e:\ Spring-frameworks-3.0.5\dist\modules folder.

- Commons.logging.jar does not come along with spring framework and we need to get separately.
- To work will all the modules of spring framework, we need to set the following two jar files in the classpath.
  1.spring.jar;
  2.common-logging.jar

Strutsframework—craig r mcdanahan
Hibernate--- cravin king
Springframework – Rod Johnson

SpringTest1
welcomeBean.java
spconfig.xml
client.java
*.class


```
//welcome Bean.java
Public class WelcomeBean
{
    Private string message;
    public void setMessage(String message)
    {
    this.message = message;
    }
```

```
        public void show( )
        {
        System.out.println("message");
        }
}
```

//springconfig.xml

<!DocType bean public -//SPRING//DTD BEAN 2.0 //EN"

http://www.springframework.org/dtd/spring-beans-2.0 dtd>

<beans>

<bean id="id" class = "welcomeBean">

<property name="message">

<value> welcome to spring framework </value>

</property>

</bean>

</beans>

//client.java

```
import org.springframework.core.io.*;
import org.springframework.beans.factory.*;
import org.springframework.beans.factory.xml.*;
class Client
```


```
{
Main ( )
{
Step-1
Resource res = new classpathResource("spconfig.xml");
Step-2
BeanFactory factory = new XmlBeanFactory(res);
Step-3
Object o = factory.getBean("id");
WelcomeBean wb = (WelcomeBean)o;
Step-4
wb.show();
}
```

}

1. Javac WelcomBean.java
2. set classpath = c:\spring.jar;c:\commons-logging-1.0.4.jar;%classpath%
3. client java
   o/p w to SFW

Points to remember:

- In the above client application , when we called getBean("id"), then internally the spring framework or container executes the following statements.
  WelcomeBean obj = new WelcomeBean()
  obj.setMessage("welcome to springFramework")
- By default each springBean (pojo) is a singleton class. Spring IOC container will make a spring bean as singleton automatically.
- In the above client application, if we call getBean() by passing the same id for multiple times, then multiple objects for the bean class are not created. Instantiated the same object will be redeemed for multiple times.

  Object o = factory.getBean("id1");
  Object o1 = factory.getBean("id1");
  Object o2 = factory.getBean("id1");

➢ In the above case, only one object of the WelcomeBean class created and it is given for 3 times.
  - The return type of getBean() is Object. Because getBean() is creating all spring bean object. So, it returns the spring Bean object is returns

superclass reference as spring. Client programmer is type caste into required Bean class type.

What is the difference between beanFactory and other Factory?

Beanfactory and other factory follows design pattern. It means the factory produces objects. The difference is BeanFactory produces different Bean objects, other factory always produces same type of objects.

Eg: In hibernate sessionFactory always produces Session Objects, where as BeanFactory produces different Bean Class objects of the spring application.

Dependency in the form of objects:

- While constructing spring beans (pojo classes), one spring bean class depends on another spring bean class, for performing some business operation.
- If one bean class is depending on another bean class object, then we call it as a object dependency.
- If one bean class is depending on another bean class object, then in spring framework, the spring IOC container is responsible for creating and initiating the dependencies.
- In spring configuration file, we have two ways to inform the container about this object dependency.
  a. By using inner beans.
  b. By using <required> element.

- **By using Inner Beans:**
  Inner bean needs a bean which is added for a property, by without an id in the xml file.
  - In case of Inner bean definition with setter injection. We should add the <bean> elements inside <property> in the xml.

For eg:

```
public Class DemoBean {
private SampleBean sb;
public SetSb(sampleBean sb)
{
this.sb = b;
}
public void m1( )
{
sb.m2();
}
}
```

SPconfig.xml:

```
<beans>
<bean id="id1" class = "DemoBean" >
<property name = "sb">
<bean class = "sampleBean"/>
</property>
```


```
</bean>
</beans>
```

- In the client application, if we call factory.getBean("id1") then internally the springframework will do the following operations.
  SampleBean obj1 = new sampleBean();

```
DemoBean obj2 = new DemoBean();
        this.setSb(obj1);
```

- In the above example, DemoBean obj is depending on SampleBean object. So, the spring container first created the SampleBean object and after that the container created DemoBean object.

Drawback of inner Beans:

1. An inner bean doesn't have any id. so, it is not possible to get that bean object individually from the container.
2. If another bean class also depending on the same bean then in xml file, again we need to add the inner bean definition.

SPconfig.xml

```
<beans>
<bean id="id1" class = "DemoBean">
<property name = "sb">
<bean class = "Sample Bean"/>
</property>
</bean>
<bean id="id2" class = "Example Bean">
<property name = "sb">
<bean class = "SampleBean"/>
</property>
</bean>

</beans>
```


- In order to overcome the above two problems of innerbeans, we need to use <ref> element in the spring configuration file.

<ref> Element:

- When dependencies are in the form of objects then to inform the spring IOC container, in spring-configuration xml file, we need to configure <ref> element.
- <ref> element is associated with either local or parent or bean attributes.
- When we add <ref> element then we need to pass id of collaborator bean to its attribute, because the dependency is in the form of object.

Syntax:

```
<ref local/parent/bean = "id of collaborator bean"/>
= equal to
<ref local = "id of collaborator bean"/>
<ref local = "                    "/>
or
<ref local = "                    "/>
= is not equal to
<ref local="id of collaborated bean" parent="id of " bean= " id of collab "/>
```

- local: if local attribute is used with the <ref> element then the spring ioc container will verify for the collaborator bean within the same container (factory).
- In generally we try to configure all spring beans into a simple spring configuration file, but it is not mandatory. Because, we can create multiple spring configuration files also.

Eg:

```
public class DemoBean
{
        private SampleBean sb;

        public void setSB(SampleBean sb)
        {
        this.sb = sb;
        }

        public void m1()
        {
        sb.m2();
        }
}
```

Spconfig.xml

```
<beans>
<bean id = "id1" class = "DemoBean" >
<property name = "sb">
<ref local = "id2" />
</property>
</bean>
<bean id = "id2" class "SampleBean">

</beans>
```

- In the above xml file, both the dependent bean and collaborator bean are configured into the same xml file. It means into the same IOC container. So we can use local attribute with <ref> tag.
- By loading this above xml file, we will get the spring IOC container object called BeanFactory.

  [Resource res = new ClasspathResource("spconfig.xml"),
  BeanFactory factory = new xmlBeanFactory (res) ; or ……..]

- The above BeanFactory statement can also be written like the following.
  BeanFactory factory = new XmlBeanFactory(res, nutt);
- The above xml configuration can also be created like the following:
  spconfig.xml

```
<beans>
<bean id = "id1" class = "DemoBean" >
<property name = "sb">
<ref local = "id2" />
</property>
</bean>
</beans>
------------
-----------
```

test.xml

```
<beans>
<bean id = "id2" class "SampleBean">

</beans>
```

- In the above xml files, dependentBean is configured in spconfig.xml and sampleBean is configured in the test.xml
- In the client application, we can load both xml files to get BeanFactorie.
- In spring IOC it is possible to create parent and child factories.
  For Eg:
  [Resource res1 = new ClasspathResource(“test.xml”);
  BeanFactory factory = new xmlBeanFactory (res1);]

Resource res2 = new ClasspathResource("spconfig.xml");
BeanFactory factory2 = new xmlBeanFactory(res2, factory1);

- In the above factory2 is a child container of factory.
- Dependent bean is available in factory2 (child container and collaborator bean is available in factory(parent container)). It means the both dependent bean and its collaborator bean are not available in same container, so we can't use local attribute with, <ref> element.
- **<parent> element:**
- If <parent> attribute is used with <ref> element then the spring IOC container will search for the collaborator bean all way at <parent> container but not in the same container.

  Eg:
  config.xml

```
<beans>
<bean id = "id1" class = "DemoBean" >
<property name = "sb">
<ref parent = "id2" />
</property>
</bean>
</beans>
------------
-----------
```

  test.xml

<beans>
<bean id = "id2" class "SampleBean">

</beans>

In the above xml files, ____ bean is available is available in factory2 and collaborate bean is available .

- So <parent> attribute gettable with <ref> element.
- In parent and child factories, the names are given like parent and child, but it is not possible to get parent factory bean object through child factory.
  Eg: Ojbect o = factory.getBean("id2");
- The above statement is wrong because id2 is related factory (parent).

<bean>

- If this attribute is used with <ref> element then the spring IOC container first verifies for the collaborator bean in the same factory. If not available then it will search in the parent factory.
- bean is the combination of both local and parent.
- Bean first works like local and otherwise it works like parent.

springconfig.xml

```
<beans>
<bean id = "id1" class = "DemoBean" >
<property name = "sb">
<ref parent = "id2" />
</property>
</bean>
</beans>
```

test.xml

<beans>
<bean id = "id2" class "SampleBean">

</beans>

According to xml <bean> attribute working like parent.

Parent

Note: 1. While using <ref> element, if the given id is not found then null will be assigned to the object, but an exception is not thrown.
2. If id is found, but the class is not a soutable for the required type then an exception will be thrown by spring IOC container. The exception name is org.spring-framework.beans.unsatisfiedDependencyInjectionException.

- The following spring application is to achieve lose coupling b/w objects. B/w traveler and vehicle.
- In this example we are creating spring beans in the form of interface and implementation class i.e poji/pojo model.

// Journey.java
public interface Journey

```
void startJourney();
}

//Traveller.java
public class Traveller implements Journey
{
        private Vehicle v;
        public void setV(Vehicle v)
        {
        this.v = v;
        }
        public void startJourney()
        {
        System.out.println("Journey started:");
        v.move();
        }
};

// Vehicle.java
public interface Vehicle
{
void move();
}
```

```
//car.java
public class car implements Vehicle
{
        private String fuelType;
        private int maxSpeed;

        public void setFuelType(String FuelType)
        {
        this.fuelType = fuelType;
        }

        public void setMaxSpeed(int maxSpeed)
        {
        this.maxSpeed = masSpeed;
        }

        public void move()
        {
        System.out.println("fuelType: "+fuelType);
        System.out.println("maxspeed: "+ maxSpeed);
        System.out.println("car started happy journey");
        }
};
```

```
//MotorBike.java
public class MotorBike implements Vehicle
{
        private int maxSpeed;

        public void setMaxSpeed(int maxSpeed)
        {
        this.maxSpeed = maxSpeed;
        }

        public void move()
        {
        System.out.println("maxSpeed: "+maxSpeed);
        System.out.println("Fuel Type: petrol");
        System.out.println("MotorBike started happy journey");

        }
};
```

springconfig.xml

```
<beans>
<bean id = "id1" class = "Car" >
<property name = "fuelType">
<value>Diesel</value>
</property>
<property name = "maxSpeed" value = "100"/>
</bean>
<bean id = "id2" class "MotorBike">
<property name = "maxSpeed" value = "100"/>
</bean>
</beans>
```

```
//springconfig.xml

<beans>
<bean id = "id1" class = "Car" >
<property name = "u">
<ref bean = "id1"/>
</property>
</bean>
</beans>
```

In the above xml file, only classes are configured but not an interface.

In springframework, interfaces are not allowed to configure into spring config file, only classes are allowed.


```
//client.java
import org.springframework.core.io.*;
import org.springframework.beans-factory.*;
import org.springframework.beans-factory.xml.*;

class Client
{
    Resource res1 = new ClasspathResource("spconfig.xml");
    BeanFactory bf1 = new XmlBeanFactory(res1);
    // parent container
    Resource res2 = new ClasspathResource("spconfig.xml");
    BeanFactory bf2 = new XmlBeanFactory(res2,bf1);
    // child container
    Object o = bf2.getBean("id3");
    //getting a traveller object from container
    Journey jo = (Journey)o ;
    jo.startJourney();

};

D:\SpringTest2> javac *.java
D:\SpringTest2> java Client
    o/p ; journey started
            fuel type: diesel
            maxSpeed : 100
            car started : happy journey.
```

In the spconfig.xml, if we change fuel type property of car to petrol and if we run the client application then we will get the following output.

Journey started, fuel type, petrol, maxspeed =100
Car started happy journey.

- In spconfig.xml, change <ref> from id1 to id2 and if we run the client application again then we will get the following o/p:

  Journey started
  Fuel type: petrol
  maxSpeed: 80
  MotorBike started happy journey.

In spconfig.xml, if we change id1 to id then springIOC container is injecting MotorBike object into Traveller by replacing Car object. In this process the programmer is not doing any change in the java code and not recompiling any files, because of loose coupling between the objects, this runtime injection of different objects into a class is possible.

In the above client application, we can add the following code to get the Car class object throw child factory.

```
Object o1 = factory.getBean("id1");
Car c = (Car)o1;
c.move( ) ;
```

- If we run the above client application then we will get the following o/p.

  Journey started
  Fuel type: petrol
  Maxspeed: 80
  Car started.

Jdbc does not support: i) collections ii) Inheritance

- In the client application we can typecast the given bean object into either its class type or its interface type.
- In the above client application we can directly typecast the objects into Journey interface type using a single statement.

Journey j = (journey) factory.getBean("id3");

Dependency in the form of collections:

While creating a spring bean (pojo), the bean class can use any of the following four types of collections as a dependency.

i) <set> Set (unordered, unique)
ii) <list> List (ordered, unique)
iii) <map> Map
iv) <properties> Properties

- Except the above four type of collections, if the spring bean class uses any other type of collection as a dependency then the spring container does not inject that collection object to the spring bean. In this case spring programmer is responsible for injecting the collection object manually.

Set Collection:

- If a spring bean has a property of collection type Set then in the spring config (xml file), we need to use <set>, to inform the SpringIOC container.
- In spring configuration file, we can use <value> and <ref> sub elements of <set>
- While configuring <set> in the xml file, it doesn't allow duplicate values, because Set Collection is a unique collection.

- In spring framework, if one bean class is collaborating with another bean class then SpringIOC container first creates collaboration bean object and after that dependent bean object.

```
public class SampleBean
{
    private Set data;

    public setDate(Set data)
    {
    this.data = data;
    }

    public void m1()
    {
    ----------------------
    ----------------------
    }
};
```

spconfig.xml

```
<beans>
<bean id="id1" class = "SampleBean">
<property name = "data">
```

```
<set>
<value>100</value>
<value>sathya</value>
<value>1025</value>
<ref bean = "id2" />
</set>
</property>
</bean>
<bean id="id2" class = "restBean">
</beans>
```

In client application, whenever we call factory.getBean("id1") internally spring container framework executes the following code.

```
TestBean tb = new TestBean();
Set s = new HashSet();
s.add(100); s.add("sathya");s.add(10.20);

SampleBean  ob = new SampleBean();
ob.setData(s);
```


List Collection:

- If a springBean is depending on a collection of type list them in spring config file, we need to configure <list>
- We can use <value> and <ref> as sub elements of <list>. The difference between set and list collections are

Set	List
1. Set is an unordered collection. 2. Set doesn't allow duplicate values 3. Set doesn't allow index based accessing. 4. Set doesn't support List Iterator.	1. List is an ordered collection. 2. List allows duplicate values. 3. List allows index based accessing. 4. List supports ListIterator

Eg:

```
public class SampleBean
{
        private List data;
        public setData(List data)
        {
        this.data = data;
        }
        public void m1()
        {
        }

};
```

SPConfig.xml

```
<beans>
```

```
<bean id="id1" class = "SampleBean" >
<property name = "data">
<list>
<value> sathya </value>
<value>100</value>
<value>10.25</value>
<ref bean= "id2" />
<value>100 </value>
</list></property></bean>
<bean id="id2" class="TestBean"/>
</beans>
```

- In client application, whenever we call factory.getBean("id1") then internally springcontainer framework executes the following code.
  TestBean tb = new TestBean();
  List l = new ArrayList();
  l.add("sathya"); l.add(100); l.add(10.25);l.add(tb);l.add(100);
  SampleBean ob = new SampleBean();
  ob.setData(l);
- If a Map collection contains 3 key,value pairs then internally it means 3 objects of MapEntry class.

  Map Collection:

  In a springbean, if we take collection type as Map then in spconfig file we should configure the <Map>. In spconfig file, we need to use the sub element of <Map> as <entry> in a Map collection one entry represents (key, value)

  We use sub element of <entry> as either <value> or <ref>

Eg:

```
public class SampleBean
{
    private Map data;
    public setData(Map data)
    {
    this.data = data;
    }
    public void m1()
    {
    }
};
```

SPConfig.xml

```
<beans>
<bean id="id1" class = "SampleBean" >
<property name = "data">
<map>
<entry key = "k1">
<value> sathya </value></entry>
<entry key="10">
<value>100</value></entry>
<entry key="k2">
<ref bean= "id2" /></entry>
<value>100 </value>
</map></property></bean>
<bean id="id2" class="TestBean"/>
</beans>
```

Ex2:

The following example has storing by typed Map collection.

```
public class SampleBean
{
    private Map<String, Float> data;
    public setData(Map<String, Float> data)
    {
    this.data = data;
    }
    public void m1()
    {
    }
};

<beans>
<bean id="id1" class = "SampleBean" >
<property name = "data">
<map>
<entry key = "k1">
<value> 10.20 </value></entry>
<entry key="10">

<value>15.20</value></entry>
<entry key="k2">
<ref bean= "id2" /></entry>
<value>18.92 </value>
</map></property></bean>
<bean id="id2" class="TestBean"/>
</beans>
```

Collection	Key	Value
Hashtable	String	Object
Properties	String	String
Map	Object	Object

Map:
"k1" – 100 //entity1 (one key,value is know as one entity)
"k2" – 10.25 //entity2
"k3" – Durga //entity3

a) What is the difference between an inner class and a nested class?

Ans: A non-static inner class is called inner class and a static inner class is called nested class.

```
//inner class
        class A
        {
                class B       //inner class
                {
                        ----------
                        ----------
                };
        };

//nested class
        class A
        {
                static class B // nested class
                {
                        ------------
                        ------------
                };
        };
```

Q) What is Map.Entity in java?
Ans: MapEntity is a class. Here Map is an Interface and Entity is a static class of Map interface.
A Map stores data in the form of (key, values and we call each pair as one entity).
In the statement Map Entity, the meaning is we can enter static classes inside an interface. We call those classes as nested classes.

```
public interface Map
{
-----------------
 Static class Entity
{
----------// getKey();
----------// getValue();
}
}
```

In spring: upto jdk 1.4 key=String, value= any object

From jdk 1.5 key=any object, value= any object

Properties Collection: Properties collection also stores data in form of (key, value). But both key and value are considered Strings.

- If we take properties collection in the spring bean then in the spring configuration file, we need to use <propo>.
- The sub element of <propo> is <prop> and <prop> doesn't have any sub element.

```
public class SampleBean
{
   private Properties data;

   public void setData(Properties data)
   {
   this.data = data;
   }
   ------------
   ------------
};
```



```
<beans>
<bean id="id1" class = "SampleBean" >
<property name = "data">
<propo>
<prop key = "k1">Durga</prop>
```

```
<prop key = "k2">100</prop>
</propo>
</property>
</beans>
```

- The following app is for injecting collection dependencies into a spring bean

E:\ -> Durga -> spconfig.xml
C:\ -> SpringTest3 ->SampleBean.java, client.java, *.class

```
//SampleBean.java
import java.util.*;
public class SampleBean
{
    private Map<String, Integer> Students;
    private List data;

    public void setStudents(Map<String,Integer>Students)
    {
    this.Students = Students;
    }

    public void setData(List data)
    {
    this.data = data;
    }
```


```
    public void printMap()
    {
```

```
    Set s = Students.entrySet();
    Iterator it = s.iterator();
    while(it.hasNext())
    {
            Map.Entry me = (Map.Entry)it.next();
            System.out.println(me.getKey()+" "+me.getValue());

    }
    }

    pubic void printList()
    {
    Iterator it = data.iterator();
    while(it.hasNext())
    {
    Object o = it.next();
    System.out.println(o.toString());
    }
    }

};
```



```
//SPconfig.xml
<bean>
<bean id="id1" class = "SampleBean">
<property name = "Students">
<map>
<entry key = "Ram">
<value> 500 </value></entry>
```

```
<entry key = "java">
<value> 999 </value></entry>
</map></property>
<property name = "data">
<list>
<value> Durga </value>
<value> 120 </value>
<value> 30.67 </value>
</list>
</property>
</bean>
</beans>
```

```
//Client.java
import org.springframework.core.io.*;
import org.springframework.beans.factory.*;
import org.springframework.beans.factory.xml.*;
```



```
public class client
{
    public static void main()
    {
    Resource res = new FileSystemResource("E:/durga/spconfig.xml");
    BeanFactory bf = new XmlBeanFactory(res);
    Obect o = bf.getBean("id1");
    SampleBean sb = (SampleBean)o;
    sb.printMap();
    sb.printList();
    }
};
```

IMP points: In spring framework, for Resource interface, we have two implementation classes, 1) Classpath Resource 2)FileSystemResource

- In ClassPathResource, the SpringFramwork verifies for the spring config xml file in the local classpath, otherwise in the jar files added in the classpath.
- If the spring config file is not available in the classpath then spring framework throws an Exception.
- While creating an object of ClassPathResource, we can't pass path of the springconfig file. It only accepts xml file name.

  Resource res = new ClassPathResource("E:\sathya\spconfig.xml")
  Resource res = new ClassPathResource("SPConfig.xml");
- In FileSystemResource, spring framework verifies for the spconfig file in the given path. If not available then spring framework throws an exception.
- In FileSystemResource, xml file can be loaded from anywhere in system.
- The difference between ClassPathResource and FileSystemResource is, in ClassPathResource xml file is loaded only from the classpath, but in FileSystemResource it can be loaded from any place in the system.
- In the above client application we have used FileSystemResource……

- If we want to use ClassPathResource instead of FileSystemResource then the following changes are required.
- In the client application, add the following statement
  Resource res = new ClassPathResource("SPConfig.xml");
  a) Create a jar file for the xml file
     E:\durga\jar –cvf me.jar
  b) Set me.jar into the classpath.
     C:\springTest3>set classpath = %classpaht%;E:\durga\me.jar
  c) Compile and run the client application
     javac Client.java
     java Client

Constructor injection:

- In this type of injection, spring framework or springContainer uses constructor of the bean class for assigning the dependencies.

- In spconfig file, we need to inform the springIOC container about constructor injection by using <constructor-arg>
- In the springbean class, if both constructor and setter injection applied for the same property then constructor injection will be overridden by setter injection.

For ex:

```
public class DemoBean
{
        private String message;

        public DemoBean(String message)
        {
        this.message = message;
        }

        public void setMessage(String message)
        {
        this.message = message;
        }

        public void show()
        {
        System.out.println(message); //hello
        }
}
```

```
<beans>
<bean id="id1" class="DemoBean">
<constructor-arg>...constructor injection
<value>welcome </value>
</constructor-arg>
<property name="message">
<value>hello</value>
</property>
</bean>
</beans>
```

- In the client application, we call factory.getBean("id1") then ---nally spring framework executes the following statements.

  DemoBean ob = new DemoBean("welcome");
  ob.setMessage("hello");
- First constructor injection executed and after that setter injection executed. So welcome is overridden with hello.
- In constructor injection, if argument types are different them at time of configuring at xml file, we can use type attribute.

```
<beans>
<bean id="id1" class="DemoBean">
<constructor-arg type="java.lang.String">
<value>10</value>
</constructor-arg>
```

```
<constructor-arg>
<value>100</value>
</constructor-arg>
</bean>
</beans>
```


```
public class DemoBean
{
        private int id;
        private String sname;
        public DemoBean(int id, String sname)
        {
        this.id = id;
        this.name = sname;
        }
        ----------
        ----------
        }
};
```

- According to the above xml, DemoBean obj is created with 100 as id and 10 as sname.

```
public class DemoBean
{
    public String uname,pwd;
    public DemoBean(String uname, String pwd)
    {
```

```
    this.uname = uname;
    this.pwd = pwd;
    }
    ------------
    ------------
    }
};
```



```
<beans>
<bean id="id1" class="DemoBean">
<constructor-arg index="1" value="sathya"/>
<constructor-arg index="0" value="ten" />
</bean>
</beans>
```

- According to the above xml DemoBean object is created with ten as username and sathya as password.

```
public class DemoBean
{
    private sampleBean sb;
    public DemoBean(SampleBean sb)
    {
    this.sb = sb;
    }
    ------------
    ------------
    }};
```


LEARN FROM EXPERTS ...
COMPLETE JAVA
CORE JAVA, ADV. JAVA, ORACLE, STRUTS, HIBERNATE, SPRING, WEB SERVICES,...
COMPLETE .NET
C#.NET, ASP.NET, SQL SERVER, MVC 5 & WCF
TESTING TOOLS
MANUAL + SELENIUM
ORACLE
D2K
MSBI
SHARE POINT
HADOOP
ANDROID
C, C++, DS, UNIX
CRT & APTITUDE TRAINING
AN ISO 9001:2008 CERTIFIED
DURGA
Software Solutions®
202, 2nd Floor, HUDA Maitrivanam,
Ameerpet, Hyd. Ph: 040-64512786,
9246212143, 8096969696
www.durgasoft.com

```
<beans>
<bean id="id1" class="DemoBean">
<constructor-arg ref = "id2"/>
</bean>
<bean id = "id2" class="sampleBean"/>
</beans>
```

- In the above xml file ref is used as an attribute, it is equal to ref bean. It is not equal to ref local and ref parent.

Circular Dependency:

Q. what is circular dependency?
Ans: If A and B are two classes and if A depends on B and B depends on A then we will get circular dependency.

- Whenever circular dependency is occurred then we can't solve this problem with the help of constructor injection in this case instead of constructor injection we need to use setter injection.
  For ex:

```
class A
{
    B b;
    A(B b)
    {
    this.b = b;
    }
};

class B
{
    A a;
    B(A a)
    {
    this.a=a;
    }
};
```

```
<beans>
<bean id="id1" class="A">
<constructor-arg ref = "id2"/>
</bean>
<bean id = "id2" class="B">
<constructor-arg ref = "id1"/>
</bean>
</beans>
```


- In a client application, whenever we call factory.getBean("id1"), then springIOC container is trying to execute class A object, but A needs B object, so the container tries to create B class Object. But B class needs A object. So A and B objects are not created and spring container throws bean currently in creation Exception.
- In order to solve the circular dependency problem b/w classes..... We need to change/ replace constructor injection with setter injection at classA.

  Ex:

```
class A
{
    B b;
    public void setB(B b)
    {
    this.b = b;
    }
};
```

```
//spconfig.xml
<beans>
<bean id="id1" class="A">
<property name="b" ref = "id2"/>
</bean>
<bean id = "id2" class="B">
<constructor-arg ref = "id1"/>
</bean>
</beans>
```

- In the client application, whenever we call factory.getBean("id1") internally the following steps are executed.
  Step1: spring container creates a mock (empty) object of class A using default constructor.
  Step2: Spring container creates class B object by passing the mock obj of class A into its constructor.
  Step3: Spring container injects class B obj into class A by calling setter method.
  Step4: finally class A obj is given back to the client application.

Q. What are the differences between constructor, setter injections?

Setter injection	Constructor injection
1. Setter injection makes bean objects mutable. 2. In setter injection, partial injection of dependencies is possible. 3. Setter injection can solve circular dependency problem. 4. Setter injection overrides the constructor injected value.	1. Constructor injection makes bean object as immutable. 2. In constructor injection part injection of dependency is not possible 3. Constructor injection doesn't solve circular dependency problem. 4. Constructor injection doesn't override setter injected value.

Note: If a class contains more number of properties. Constructor injection is recommended to use (because we can reduce length of the code of class).

If a bean class contains multiple constructors then we need to configure the bean class multiple times with different id's into spring config file.

For eg:

```
class A
{
    private int id;
    private String sname;

    A(int id, String sname)
```

```
    {
    this.id = id;
    this.sname = sname;
    }

    A(int a, int b, int c)
    {
    this.a = a;
    this.b = b;
    this.c = c;
    }
```



```
    A(String sname, int id)
    {
    this.sname = sname;
    this.id = id;
    }
};
//spconfig.xml
<beans>
<bean id="id1" class="A">
<constructor-arg value="10"/>
<constructor-arg type="java.lang.String" value="100"/>
</bean>
<bean id="id2" class="A">
<constructor-arg value = "10">
<constructor-arg value = "20">
<constructor-arg value = "30">
</bean>
</beans>
```

- In the above example, whenever we call factory.getBean("id1") from the client application then spring framework creates an object of class A by calling first constructor.

Bean autowriting:-

Writing a bean means configuring a bean along with it's dependency into an xml like.

By default were the bean properties into an xml file

If auto writing is enabled then springfw will take care about injecting the dependencies and program is no need to configure into an .xml file explicitly

Bean auto writing is only supported, if the dependencies are in the form of objects.

To enable auto writing , we should add autowrite attribute to the <bean>

1)dynamic 2) bytype 3) constructor 4) autodetect 5) name

1) **Byname**:-in this case, springfw attempts to bindout a bean in the confi file, whose id is matching with the property name to be wired.
2) if a bean found, with the id as property name then that class obj with be injected into that property by calling setter injection.
3) If no id found that property remains unwide.

Example:- public class exampleBean1

```
{
Privat ExampleBean2
Public void setEb2(ExampleBeaan2 eb2)
{
This.eb2=eb2;
}
}
```

Spconfig.xml:-

```
<beans>
<bean id="id1 class="ExampleBean1"
Autowrite="byname">
</bean>
<bean id="id2" class="ExampleBean2"/>
</beans>
```

2)Bytype:in the case the springfw attempts to bind out the class in xml file whose name Is matching with the property type to be wired (or) not.
if found then injection that class obj by calling setter injection
if no class found in the xml with the same name then that property remains unwired.
If a class is found in xml for more than once then springfw throws unsatisfied dependency injection exception.

Example:- public class ExampleBean1
{
Private ExampleBean2 eb2;
Public void setEb2(ExampleBean2 eb2)
{
This.eb2=eb2;
}
}
Spconfig.xml:-
<beans>
<bean id=”id1” class =”ExampleBean1”
Autowire=”bytype”/>
<bean id=”id2” class=”ExampleBean2”
</beans>

3)constructor:- This autowrite type is equal to bytype but here constructor injection will be encountered.

```
Example:- public class Ex1
{
private  Ex2 eb2
public Ex1(Ex2 eb2)
{
This.eb2=eb2;
}
}
```

<beans>
<bean id="id1" class="Ex1" autowire="constructor"/>
<bean id="id2 class="Ex2"/>
</beans>

4)**autodetect**:-the type of autowiring first work like a constructor and if not then works like bytype.

Validating the dependency:- in springfw, either in explicit wiring or in autowiring if all properties are not configured in xml, but still spring continer creates a n object of the bean class.

By defaultspringcontainer doesn't verify whether call properties are set in xml (or) not.

If we enable the dependency validation then spring container verifies whether all dependencies are set (or) not if not then container doesn't create an obj and throws an exception.

To enable dependency validation, we need to add dependency-check attribute for the <bean>

The different values of dependency-check attributes are

1) No(default) 2) simple 3) objects 4) all

Simple:-in this case spring container verifies for primitives and collections are set (or) not. If not set then exception will be thrown.

Object:-in this case, spring container verifies whether objects are set (or) not, if not the container throws an exception.

All:-in this case the container verifies for both primitives collections and objects.

Example:- public class TestBean
{
Private int a;
Private int b;
Public void seta(int a)
{
This.a=a;
}
Public void setB(int b)
{
This=b;
}
}
<beans>
<bean id=”id1” class =”TestBean”
Dependency-check=”simple”>
<property name=”a” value=”10”/>
</beans>
</beans>
In the above example property b is not in the xml. So simple dependency validation is failed. Then an excepting will be occurred.

Example:-public class TestBean
{
Private DemoBeandb;
Private int a;
Private int b;

```
Public void seta(int a)
{
This.a=a;
}
Public void setB(int b)
{
This.b=b;
}
Public void setDb(DemoBeandb)
{
This.db=db
}
}
<bean id=”id1” class=”Testbean”
Dependency-check=”objects”>
<property name=”a” value=”10”/>
</bean>
```


In the above example, because dependency-check=”objects” it only verifies for object dependency and spring container throws an exception, because the object is not injected.

The following spring application is for validating the dependencies for objects This spring application uses second spring IOC container called application context.

```
//car.java
Public class car
{
Private void setW(wheel w)
```

[
This.w=w;
}
Public void display()
{
Sop(“done”);
}
}

springTest3
car.java
wheel.java
client.java
*.classes
Spconfig.xml

.wheel.java
Public class wheel
{

};

It is not a empty class because in java every class (empty)
Contain public, default constructor it is nothing but a method
In java it is not possible to write empty class.

Spconfig.xml
<beans>
<bean id=”id1” class=”car” dependency-check =”objects”/>
<bean id=”id2” class=”wheel”/>

```
</beans>
```

//client.java:-

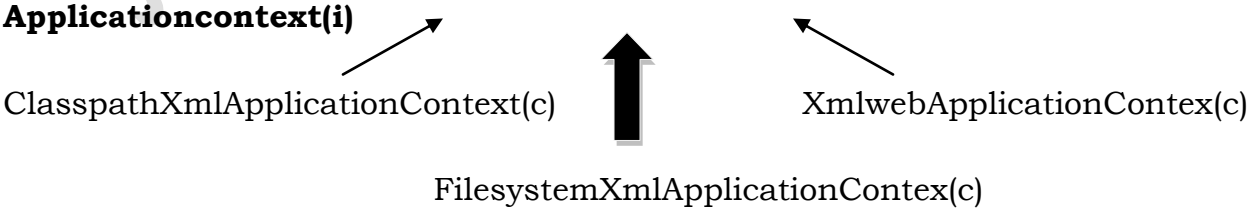
```
Import org.springframework.context.*;
Import org.springframework.support.*;
Class client
{
Public svm()
ApplicationContext ctx=new classpathXmlApplicationContext("spconfig.xml")
Object o=ctx.getBean(id1");
Car c=(Car)o;
c.display( );
}
}
```

In the above client application, we have used the second spring IOC container called ApplicationContext.
In springfw, we have two IOC container.
1)BeanFactory
2) Applicationcontext

BeanFactory is called as basic IOC container and Application context is called an advanced IOC container

Applicationl context interface:-This spring IOC container is given in org.springframework.context.* package and it's implementation classes are given in org.springframework.context.support package.
Applicationcontext interface has three implementation classes.
In Frameworks, interface and their implementation classes, both are given by framework only.

BeanFactory-(i)
Applicationcontext(i)



ClasspathXmlApplicationContext loads(Reads) the spcofig file either from the local classpath(or) from a jar file which is added in the classpath.
Filesystem XmlApplicationcontext always loads from filesystem of the system.
XmlWebApplication.Context is used, when spring beans are a part of web application
ApplicationContext ctx=new
ClassPathXmlApplicationContext("E:/sathya/spconfig.xml");
The xml file path is not allowed.
ApplicationContext ctx= new
FileSystemXmlApplicationContext("E:/sathya/spconfig.xml")
ApplicationContext ctx= new
FileSystemXmlApplicationContext("spconfig.xml");→allowed
In this case, the xml file is secured in the local folder

Bean initializing and destruction :-
While creating our bean class is spring, apart from injection we can also create methods for performing initialization and decimalization (clean-up) code required for the bean class.
In a springbean if we want to add some initialization code and cleanup code then we need to implement our spring bean class from initializing and DisposableBean interfaces.

If we implement the above two interfaces then in our spring bean class we have to override the following methods.

1)after PropertieSet() from Initializing Bean interface
2) destroy() from DisPosableBean interface.

For example:-

```
Public class TestBean implements InitializingBean, DisposableBean
{
Public Void afterPropertiesSet()
{
//initialization logic
}
Public void destroy()
{
Deinitiaalization (clean-up)logic
}
}
```


It Is not a pojo class.
In the above bean class, whenever client request is given for getting the bean object then internally the following steps are executed.

1) Object is created.
2) Properties are injected
3) After PropertiesSet() called.
4) Now obj is given to client application,

Before container is removing (destroying) the object, container first calls destroy () and then removes that obj from memory .
In the above example, the drawback is our class is not a POJO class . Because it is implementing from predefined interface given by the framework.
So we can rewrite the above bean class by adding our own methods for initializing and cleanup.

For example:- public class TestBean

```
{
Public void init()
{
//initialization code
}
Public void teardown()
{
//destruction code(cleanup)
}
}
```

While configuring the above into xml, we have to add init-method, destroy-method attributes for the <bean>

```
<bean id=”id1” class=”TestBean” init-method=”init”
Destroy-method=”teardown”>
--
</bean>
```

In this case container will do the following, before giving the object to a client.

1) Object is created
2) Properties are injected.
3) Innit() will be called.
4) Object is given to client.

Whenever container is giving to remove the first it calls teardown() and after that the object is removed.

BeanScops in springFw:- By default every springbean is singleton. It means even though we call getBean() from client for multiple times but only one object is created and it will be given for multiple times.

If we want a separate bean obj for each time whenever we call getBean() then we need to change the beanscope. From singleton to prototype.
If we want to change the bean scope then we need scope attribute.
<bean id="id1" class="TestBean" singleton="true/false"/>
<bean id="id1" class="clasname" scope="prototype"/>→spring2.x
In spring2.x, we have the following five scopes added.
For a bean

1) Singleton(default) 2) prototype 3) request 4) session 5) global session application

Request.session and global session are used in springMVc application development (web applications).

Bean instantiation:-

Spring framework instantiates/gets a spring bean object, by using the following three ways.

1) By calling constructor.
2) By calling static factory method.
3) By calling an instance factory method.

1)**By calling constructor:-** whenever we call getBean("id") from the client application then the springIOC container uses new operator and calls constructor of the class and gets an objects of the bean class.
After getting the bean obj, it will apply injections and after that other initializations and then returns that bean object to the client application.
If we change the bean scope to prototype then the IOC container gets/creates a new object for each call to the getBean("id")

2)**By calling static factory method:-** by default, spring framework makes each bean as singleton .But whenever the scope of a bean is changed to prototype then the bean obj becomes non.singleton
If spring programmer doesn't want to make a springbean as prototype bean then the programmer has to explicitly make the bean class as singleton
To make a springbean class as a singleton, the following rules are need to be followed.

1) Create a private static obj of the class, inside the class
2) Create a private constructor.
3) Crete a public static factory method.

For example:- public class SampleBean
{
Private static SampleBean sb=new SampleBean();
Private SampleBean()
{
…
}
Public static SampleBean getSampleBean()
{
return sb;
}
}

While constructing the above bean class into spring config file we need to inform to the springIOC container that, call static factory method of the bean class to get an obj of the bean class.
To inform the springIOC container, we should add an attribute called factory-method to the <bean>
<bean id="id1" class="sampleBean"
*factory-method="getSampleBean"

…
..
</bean>

Whenever getBean() is called from the client application then the IOC container will get the bean object by using the following statement internally.
(SampleBean sb=SampleBeangetSampleBean();)
After getting the obj. the IOC container applies all injections and initialization required and then finally returns that bean obj to the client application.

Note:- if we change scope of the bean from singleton to prototype then, still the bean is a singleton bean only. It means that bean never out as prototype bean.

Question:-how to create synchronized singleton class in java?
If we add synchronized wayward for the static factory method then the class becomes synchronized system class.

3)**By calling an instance factory method:-** in this approach springIOC container calls the factory method defined in one class to get an obj of another bean class.
In java we have two types of factory methods
1)static factory method.
2) instance factory method.
A factory method may(or) may not returns same class object but a factory method must return an object

```
Public class SampleBean1
{
……
……
}
Public class SampleBean2
{
Public SampleBean1 getInstance()
{
SampleBean1 sb1=new SampleBean()
Return sb1;
}
}
<bean id=”id1” factory.bean=”id2” factory-method=”getInstance”/>
```

<bean id=”id2” class=”SampleBean2”/>

In the above xml, for the bean id1 we have removed class attributes and we have added factory-bean.
In a client applicaton, whenever we call factory.getBean(“id1”),internally the following staments are executed by the container.
SampleBean2 ob2=new SampleBean();
SampleBean ob1=ob2=getInstance();

Question:- how to make a java class as immutable?
We can make a java class as immutable by following the below rules.

1) Instance variables of the class must be private and final.
2) Class must containe a public parameterized constructor.
3) Define getter methods for the variables and avoid setter methods.

For example:- public class Sample
{
Private final int x,y;
Public Sample(intx,inty)
{
This.x=x;
This.y=y;
}
Public int getx()
{
Returns x;
Public int gety()
{

Returns y;

```
}
}
```

Question:-can we assign value for private final variable using setter methods (or) not.

Not possible.

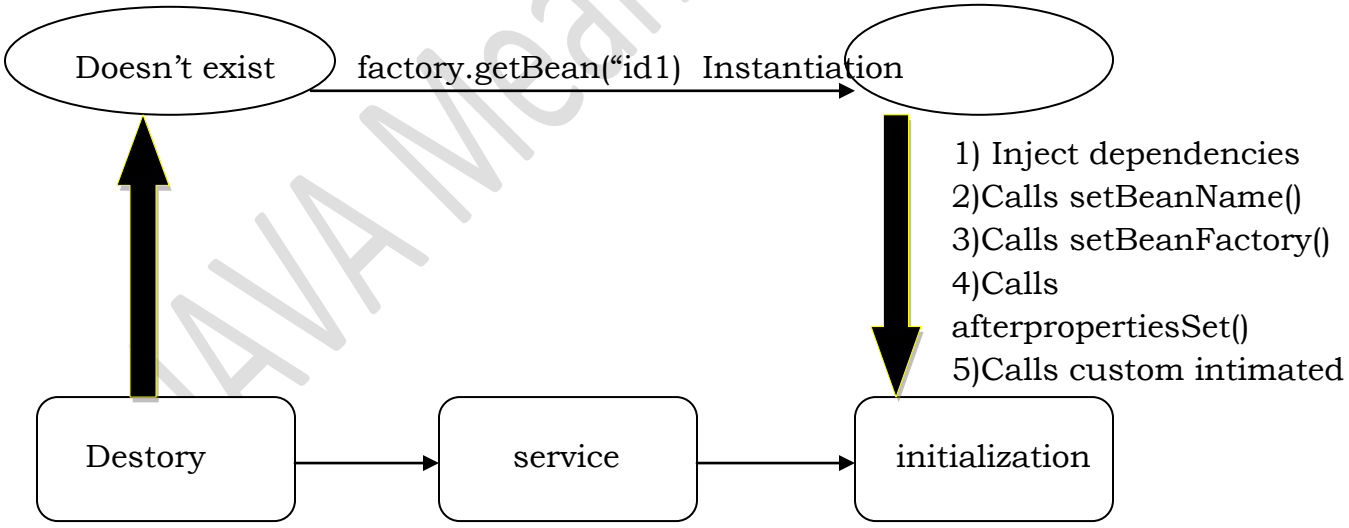
For final instance variables of the class, initialization must be done through constructor but not through setter.

For example:- public class DemoBean
{
Private final int m;
Public DemoBean (int m);
{
This.m=m;
}
}

Spconfig.xml
<bean id="id1" class="DemoBean">
<constructor-arg value="100"/>
</bean>

Bean life cycle:-

1) Doesn't exist
2) Instantiation
3) Initialization
4) Service
5) Destroy



1) Custom destroy method
2) Calls destroy()

A Bean object life cycle is a circular process, which means the life cycle starts at “doesn’t exit” and finally ends at “doesn’t exit” only.

Whenever we call getBean() by passing id then the IOC container creates an object of the bean class. It means the bean object is moved from doesn’t exist stage into instantiation.

After a bean obj is created, the IOC container initializes the object through the following verifications.

a) Container verifies whether dependencies are exist (or) not.if exist then it injects the dependencies.
b) The container verifies whether the bean class is implemented from BeanNameAware interface (or) not if yes then container calls SetBeanName(), by passing id of that bean.
c) Container verifies whether bean class is implemented from BeanFactoryAware interface (or) not . if yes then calls SetBeanFactory method by passing the current container (this) object.
d) Container verifies whether bean class is implemented initializing bean interface (or) not . if yes then calls afterPropertiesSet()
e) Container verifies any user defined init() is added (or) not if yes then container calls that custom init()
f) By completing the above operations, the IOC container returns the initialized bean object to the client application
g) In the client application service methods(Business method) of the bean are called.

h) Whenever container is going to shutdown, it removes the bean object. It means first container verifies whether any user defined destroy methods is added (or) not. If yes then executes it. Implemented DisPosableBean interface (or) not . if yes then container calls destroy() and finally makes the bean obj as destroyed

i) When the bean obj is destroyed then again it goes back to “doesn’t exist” stats.

j) 

Question:- what is the diff b/w the sprigIOC containers BeanFactory and applicationContext?

BeanFactory is a Basic IOC container and Application context is an advanced container which is extended from Bean Factory.

BeanFactory doesn’t support real time services added to a bean like mailing messaging , i18n ….etc But Application context supports the services added to the springbean.

For real time application ApplicationContext is soutable then BeanFactory.

MODULE-2

SPRING-JDBC (DAO)

DAO DESIGN PATTERN:

While developing the business logic of an application generally we include data access code as combinely this leado to high coupling between business logic and Data access logic .

Tighi coupling means if we went to change the data access code from one technology to another technology then it effection on the bussines logic also.

DAO is a Design pattern , which is used to get a coupling between business logic and data access logic of an application.

Dao provides an interface which list of operl to the business logic , by hiding the implementation from the business logic

The advantage of DAO interface is , into implementation can be changed from one technology to another technology by with out making any effect on the By.

Yer an hibernate is a DAO which hider the implementation from the java business logic by

In DAO Design pattern , the business logic gets a reference of DAO interface throw a Dao factory .

DAO factory is a factory design pattern which provides Dao objects . the following example is a Dao pattern to separate user business logic from the persistence operations required on user.

USER DAO (I)

Void insert()

```
Void update()

Void find()

User DAO Impl1 (0)


P user DAO Impl1 implements userDAO

{

P   v  insert ()

{

 //Insert logic with jdbc

}
```


```
P v update()

{

// update logic with jdbc }

P v find()
```

```
{
//search logic with jdbc
}
}
```

DTO or VO =Data Transfer Object or Value of Object .

User DAO impl2

P c user DAOimpl2 implement userDAO

{

P v insert ()

{

//insert logic with hibernate

}

P v update ()

{

// update logic with hibernate

}

P v find ()

{

```
// search logic with hibernate
}
}
//userDAO factory
P c user DAO factory
{
Public static user DAO getuserDAO (string s)
{

If (s.equals("jdbc")
Return new userDAO impl1();
Else
Return new userDAOImpl2();
}
}
// user business
P   c userBusiness
{
P  v  bm1()
{
```

```
userDAO ud = user DAOfactory .get userDAO("jdbc", "hibernate");

ud.insert();

}
```

}

In the above example we have created a DAO with set ofoperations , which are required for the business logic .

We have done implementation for those operations using jdbc technology and hibernate .

We have created a DAO factory , which returns the Dao reference to the business logic , when ever the factory method is called

In the business method of the business class , we are getting DAO interface reference through DAO Factory by calling its factory method .

If we pass jdbc as a parameter to the factory method then the factory returns an implementation obj done throw jdbc technology to the B.H.

If we B.H want hibernate as a persistence technology then in the business method we need to pass hibernate as a parameter to the factory method.

By a little (or) no modifications in the B.H , we can shift from one persistence technology to another persistence technology . this is achived through DAO interface.

The DAO design pattern is one of the most important pattern of J2EE and it provides loose coupling between B.L and its persistence logic.

SPRING—JDBC

Data source is diff implementation.

Eg: d manager--------etc.

Spring given 1. DriverManagerDataSource. (spfw)

2. Basic DataSource

Apache given (commands-dbcp)

1. JDBc – technology is required either directly or indirectly for getting a connection with database.
2. Without using jdbc technology we cant able to connect with db using java.
3. If java programmer is a directly working with jdbc technology , then there are some problems faceing by the java programmer.

Jdbc – technology exceptions are checked exceptions . so we must put try and catch block in the code at various places and it increases the complexcity of the application .

In a java program repeated code is required , to perform operation on database from the various client application . the repeated code like loading the driver , connection open or close creating a connection ……etc . of jdbc . this kind of repeted code we call bailarplate code.

In jdbc if we open the connection with database then we are responsible to close it . if the connection are not close , then later at some point of time , we many get out of connection problems.

By a little or no modification in the B.H, we can shift from one persistence technology to another persistence technology . this is achived through DAO interface.

The DAO design pattern is one of the most important pattern of j2ee and it provides loose coupling between B.H and its persistence logic.

Jdbc throws error codes of the db , when an exception is occurred . some times the error codes are unknown to the java programmer and error codes are diff from one database to another database . so developing jdbc application using those error codes will made our application db dependent .

In order over come the above problems obtained when using the jdbc technology directly , spring framework as provided an abstraction layers on top of the existing jdbc. Technology, we call this layer as spring jdbc layer.

In spring jdbc layer , the spring programmer was work with obstraction layer and the obstraction layer internally uses the jdbc technology .

Spring jdbc layer concentrate on avoiding connection management coading and error management coading the spring programmer concentrate on the construction and execution of sql operation.

Spring frame work as provided an exception &later and it translate the checked exception as spring type and finally the un cheeked exception are throw to the java programmer . while creating the spring –jdbc , the programmer no need open and ---------- by spring frame work.

A java application can get a connection while the data base using the following 2 ways:

1 . java . sql . DriverManager class.

2 . java.sql.DataSource interface.

Spring framework always user data source interface to obtain a connection with the database internally .

Spring framework classes data source implementation classes to obtain a connection with the database.

Data source interface also having diff types of implementations . 1. Basic Implementation , it is equal to Driver Manager .

Connection poling implementation,

Transaction implementation,

While using dataSource , it is always does not means that we are working with connection pool . if the internal implementation class is connection pool is enabled then only the datasource will be provide a logical connection from the pool .

In spring framework , we use any one of the following two implementation classes of datasource interface.

Org.spring framework.jdbc.datasource.drivermanager datasource class

Org.apache.commans.dbcp.Basic DataSourse also.

The two class are soutable whenever our spring app is under development.

In production mode the spring application under connection poling provided by the application.server.

In the above two classes ,DriverManager DataSource is given by spring fw and it is equal to drivermanager cllly open a new connection and classes the connection for each operations done the data base.

Basic Datasource is given apache- commons- dbcp project. And it is better than DriverManager DataSourse. Because Basic DataSouirce has inbuild connection pool implementation . In spring confriction file, we need to configenafion the following 4 properties to obtain the connection with

1 . driverclassname

2 . orl

3 . username

4. Password

Eg: <bean id =”id”classes=”org.spfw.jdbc.datasource.DMDS”

<properly name=”DriverClassName” value=oracle.jdbc.oracledrivers

<properly name=”url”value= jdbc:oracle:thin:@localhost:1521

<properly name=”username” value:”scotts”

<properly name=”password” value:”tiggers”

</beam>

In spring fw we can configure in xml file only classes but not

In spring –jdbc communication ,we need to configure the data source properties into spring configuration xml file.

The advantage of configuring , the datasource properties in to any xml file, is , we can easily shift from one database to another database. Or from one jdbc driver to another jdbc driver, by just changing data source properties at xml file , by with out changing our java application .

Central classes IN SPRING-jdbc:

1 . JDBC-Template

2 . Stored procedure

Procedure

Functions

Can be called taking the input and given to the output time is called template pattern . in middle what happen we don't know

JdbcTemplateClasses : This class is given in org.springframework.jdbc.core package and this clas will provides methods for executing the sql commands or operations on a database.

Jdbc template classes followes design pattern , where a template class will accept input from the user and output to the user ,by hiding the internal details.

Jdbc termplate classes has provided the following 3 type of method to execute the sql operations on the database.

Execute()

Update()

Query methods.

Execute () and update() are for non select operations on db and query methods are select operations on dbase.

Jdbc template id = new jdbc template(dc)

Jdbc template class dependent on DataSource Object, because jdbc template classes open connection internally throw datasource object only

In jdbc template classes , we have both setter and construction for inserting datasource object

jdbcTemplate class object .created in the following 2 ways.

1 . jdbctemplate sd = new jdbctemplate()

2 . jdbctemplate jt = new JdbcTemplate(DataSource)

In first case , dataSource Object is inserted to jdbc template by calling setter injection.

It.SetDatasource(DataSource obj);

Spring IOC container will inject DataSource Object in jdbcTemplate. But we need to configure then into configuration file .

Spconfig1.xml:

<beans>

<bean id>"id1" cl;ass ="org.spfw jdbc.datasource .driverManager datasource">

</bean>

,bean id ="id2" class "org: spfw jdbc .core . jdbctemplate">

<constructor- org ref ."id1"1>

</bean>

</beans>

In the above xml datasource object injected into jdbc Template throws constructor injection by the container .

The DataSource object can be injected throw throw setter injection also into jdbc template . for this we need to configure record beam like the followings.

<beam id= “id2”class=” orgspfw.jdbc.core.jdbctemplate”>

<property name=”datasource” ref”ids

</bean>.

While constructing our spring beam class , our beam also is depending on jdbc template class object.

Ex: public class DemoBean

{

Private jdbctemplate it;

Public void set it (jdbc Template it)

{

this.it= it;

}

}

Our DemoBeam class having setter injection to get the collaborate object jdbc template . so in sp config file we need to configure DemoBean class like following.

< bean id = “id3” class=”DemoBean”>

<properly name = “it” ref =”id2”/>

</beran>

Ex: the following example is to create a table and insert a row using jdbc template.

Springjdbc1

Demobean.java

Spconfing.xml

Client.java

//demobean

Import org . spfw.jdbc.core.*;

Public class demobean

{

Private jdbctemplate jt;

Public demobeam(jdbctemplate jt)

{

This.it=if;

}

Public void create table()

{

jt.execute(“create table spring (sno number(3), sname varchar(2));

sop(“table created”);

}

Public void insert row ()

{

Int k = jt .update(“insert into spring values (1,sathya);

Sop(k+”row inserted”);

}

}

Spconfig.xml:

<! DTD >

<beans>

<bean id=”id1” class =”org. spfw.jdbc.datasource.driverManager>Datasource

<property name =”driver class name”valvu=”sun.jdbc.odbc.jdbcdriver>

<property name = “url” value > “jdbc.odbc.oradjn”/>

<property name = “user name “ value = “scott/>

<property name = “password” value = “tiger”/>

</bean>

<bean id = “id2” class = org.spfw.jdbc.core.jdbctemplate”>;

<constructor – org ref=”id1”);

<bean>

<bean id= “id3” class =”DFemoBean”>

<constructor-org =”id2”/>

</bean>

</beans>

Client. Java:

Import org.spfw.content.*;

Import org.spfw.client.support.*;

Class client

{

Main()

{

Application Content ctx = new class path xml application content (“spconfig.xml”);

Object o =ctx.getBean(“id3”);

DFemo bean db = (DemoBean)o;

Db.create table ()

Db.insertrow();

```
}

}
```

o/p 1 table created

1 row created

Methods of jdbc template:

1 **. <u>void execute ("sql cmd")-</u>**

This method is used to executed both DDL and DML (non-select) operation on the database.

This is method allowed only static SQL commands. It means the SQL command should not contain (?) symbol.

This Is suitable DDL operations on the database because for DML operations , this method does not return the count value back to the programmer.

Example:- jt.execute("update emp set sal=2000 where empno=7000");

In the above method execute() does not inform whether the salary is update(or) not so for DML operations this execute is not suitable.

2) **int update(SQL command"); int update("SQL cmd,object[]):-**

This method is used to execute only DML operation on the db it means either insert (or) update(or) delete.

If we use a single parameter then we need to pass static SQL command and if we use two parameters then we need to use dynamic SQL command.

If we use dynamic sql command then the value into each question mark, should be set in the form of an OBJECT

In case of dynamic command, first wee need to store all objects into an object array and then we need to pass that object array as aparameter.

EXAMPLE:- Object parameter[]={new Integer(103),"sathya"};

Int k=it.update("insert into spring values(??)",params);

The second example can be written also like the following int k= it.update("insert into spring value(??)", new object[]{new Integer(103),"sathya"});

Query methods of jdbc template:-

1) **Int queryForInt("sqlcmd")& query ForInt("sqlcmd",Objcets[]):-**

This method is used for executing a select operation on db, which returns an interger.

We can pass either static (or) dynamic SQL command to this method.

EXAMPLE:-int c=jt.queryForInt("select count(*) from emp");

Int c=jt.queryForInt("select count(*) from emp where deptno=? , new Object[]{new Integer(20)});

2)**long queryForLong("sql comd") & queryForLong("sqlcomnd", object[]);-**

If our query command is going to return a long to value then we use this method

We can pass both static and dynamic sql commands to this method.

Example: long l=jt.queryForLong("select sam(sal) from emp");

2) Long l= jt.queryForLong("select sum(sal) from emp where deptno=9, new object[]{new Integer(20)});

3)**object queryFor object("sql comd",class c):-**

This method is used to get the result of the select command, in the form of required object.

We have to pass the class type (or) required class object, as a second parameter

Example:- Object o=id.queryForObject("select sydate from dual", Date.Clas);

Date d=(Date)o;

Example:- Object o=it.queryForObject("select avg(sal)) from emp", Integer.class),

Integeri=(Integer)o;

Int x=i.intValue();

4) **list queryForList("sql comd") & queryForList("sql comd",Object[]):-**

This method is used for selecting one (or) more records from the database table.

Internally jdbcTemplate stores all the run into a ResultsetObject and creates the objects[] array for an each row and finally stores all these objects into a collection of type Array List and the List object is given back to the programmer.

While iterating the collection the programmer as to type cast the result into an object array.

Employee DAO(i)

{

InjectEmployee (Empolyeeob)

liistAllEmployee()

updateEmpolyee(int eno,double sal) DAO pattern

}

Public class DemoBean

{

EmployeDAO .edo;

Public void insert(Employee ob)

{

2)edo.insertEmployee(ob)

3)

}

DTO pattern:-

Client

{

--

Db.insert(e);

}

employeeDAOImpl©

insertEmployee()

{

--

}

List AllEmployee()

{

--

}

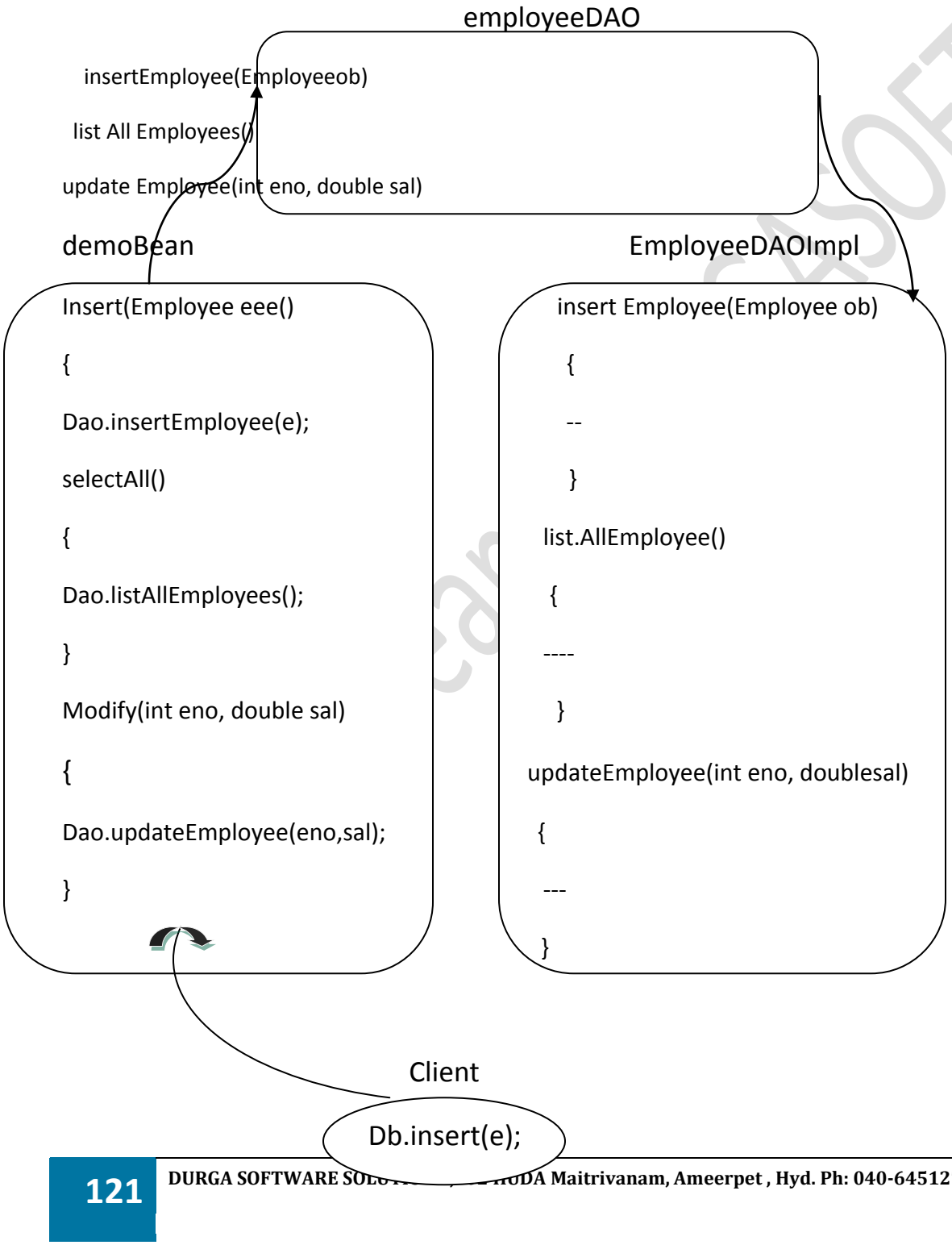
updateEmploy()

{

--

}

The following application is a DTO and DAO pattern implementation to perform the dbase operation required for a springBean.



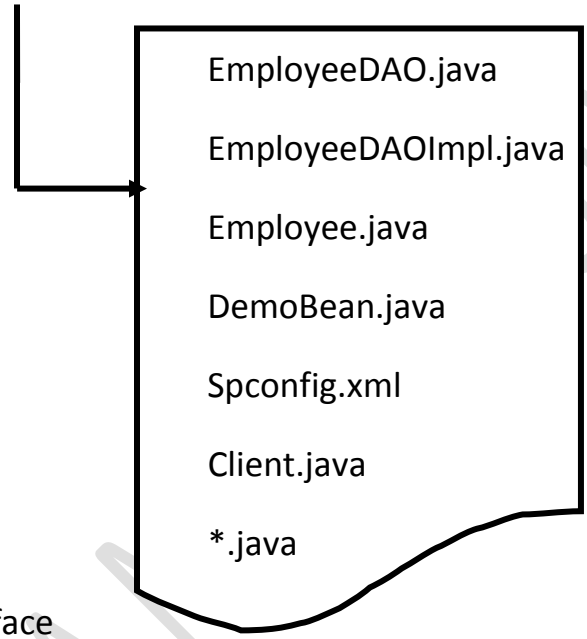
Here two designe patterns are covered.

1)DAO

2) DTO are designed patterns.

Every project compulsory singleton and DTO pattern are used.

springJdbc2



//DAO interface

//EmpolyeeDAO.java

Public interface EmployeeDAO

{

Int insert Employee(Empolyee e);

Void listAllEmployees();

Int updateEmployee(int eno, double sal);

```
}

//DAO implementation

//EmployeeDAOImpl.java

Import java.util.*;

Import org.spfw.jdbc.core.*;

Public class EmployeeDAOImpl implements EmployeeDAO

{

Private JdbcTemplate it;

Public void setIt(jdbc Template jt)

{

This.it=it;
```


```
}

Public int insertEmployee(Employee e)

{
```

```
Int eno=e.getEmpno();

String ename=e.getEname();

Int sal=e.getSal();

Int dno=e.getDeptno()

Object params[]={eno,ename,sal,dno)

Int k= it.update("insert into Employee Values(????)",params);

Retrun k;

}
```


```
Public void listAllEmployees()

{

List l=it.querForList("select * from Employee");

Iterator it=l.iterator()

While(it.hasNext())

{

Map m=(Map) it.next();
```

```
Public int updateEmployee(int eno,double sal)

{

Object params[]={sal,eno};

Int k=it.update(“update employee set sal=? Where empno=?,param);

Return k;

}

}// class.
```

//Employee.java

//DTO (or) Vo(DataaTransfer object (or) ValueObject)

Public class Employee

{

Private int empno;

Private String ename;

Privat int sal;

Private int deptno;

Setter and getter

}

//**DemoBean.java**

//**Business component.**

Public class DemoBean

{

Private EmployeeDAO dao;

```
Public void setDao(EmployeeDAO dao)
{
This.dao=dao;
}
Public void insert (Employee e)
{
```


```
Int x=dao.insertEmployee (e);
Sop(x=”Row inserted”);
]
Public void selectaAll()
{
Dao.ListAllEmployees();
}
```

Public void modify(int eno, double sal)

{

Int x= dao.updateeEmployee(eno,sal)

Sop(x+"row updated");

}

}

Spconfig.xml:-

Previous application unlist jdbcTemplate java

<beans>

<bean id="id3" class="EmployeeDAOImpl">

<property name="it" ref="id2"/>

</bean>

<bean id ="id4" class="DemoBean">

<property name="dao" ref="id3"/>

</bean>

</beans>

//client.java

Import org.spfw.context.*;

Import org.spfw.context.support.*;

Class client

{

Main()

{

ApplicationContext ctx=new classPathXmlApplicationContext("spconfig.xml");

Object o=ctx.getBean("id4");

DemoBean d=(DemoBean)o;

Employee e=new Employee()

e.setEmpno(7898);

e.setEname("RAM");

e.setDeptno(10);

db.insert(e);

sop("……");

db.DelectAll();

sop("……");

db.modify(7788,9999);

}}

Loading datasource Properties frama resource bundle:-

While configuring a datasource implementation class into sprig-configuration xml file, we are placing directly the connection properties into the xml file.

Instated of directly placing the values into the xml file, we can load the values at run time for the datasource properties from a resource bundle.

If we want get the data source properties at run time from a resource bundle to then while configuring the bean into xml file, we should put the bundle keys with expression language into am xml file.

For example:-- <bean id="id1" class="org.spfw.jdbc.datasource.DMDS">

<property name="driverclassName">

<value>${jdbc.driver}</value>

</property>

<property name=”url”>

<Value>${jdb.orl}</value>

</property>

</property name=”username”>

<value>${jdbc.user}</value>

</property>

<property name=password>

<value>${jdbc.pass}</value>

</property>

</bean>

In to the above bean definition, the values are passed at run time from the resource bundle. The bundle is like the following.

Abcd.txt

Jdbc.driver=sun.jdbc.odbc.jdbcodbcDriver

Jdbc.url=jdbc.odbc.sathya

Jdbc.user=scott

Jdbc.pass=layer

Property Palce holder configure:-

is a class given by spring framework the purpose of his class is take the values from resource bundle and given to xml file.

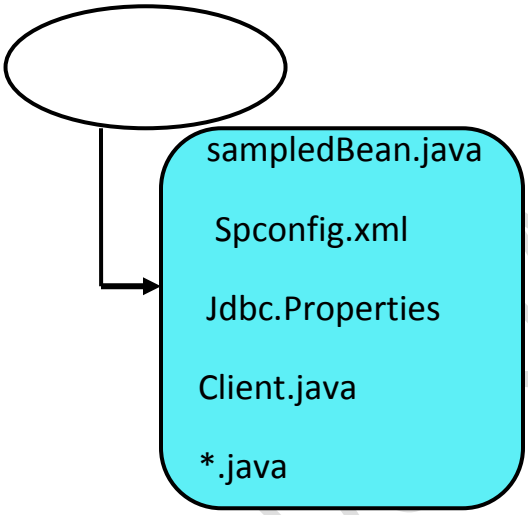
In springframework , we have predefined class given called propertyPlaceholderConfigure . and this class will load the data from the bundle and writes the values from bean definition of xml.

propertyPlaceholderConfigure is a class given org.spfw.beans.factory.config.package

question:-the following spring applicaton is to load the data source properties from a resource bundles

.

Springjdbcs



//sampleBean.java

```
Import org.spfw.jdbc.core.*;

Public class SampleBean

{
```



```
Private JdbcTemplate jt;
```

Public void setjt(JdbcTemplate jt)

{

This.jt=jt;

}

Public void update(double sal, int eno)

{

Object p[]={sal,eno}

Int k=jt.update("update emp set sal=? Where empno=?"p);

Sop("Towupdated");

}

}

Spconfig.xml:-<beans>

<bean id="id1">

Class=DMDS">

<Property name=”driverClassName”>

<value>${jdbc.driver}</value>

<property>

<property name=”url”>

<…..

…….

…..

</property

</bean>

<bean id=”id2” class=”org.spfw.jdbc.core.jdbcTemplate”>

<constructor-arg>

<ref bean=”id1”>

</constructor-arg>

</bean>

</bean idd=”id3” class=”SampleBean”>

</property name=”jt”>

<ref bean=”id2”/>

</property>

</bean>

</beans>

#jdbc.properties

Jdbc.driver=sum.jdbc.odbc.jdbcodbcDriver

Jdbc.url=jdbc.odbc.oradsn

Jdbc.user=scott

Jdbc.pass=tiger

Client.java

Import org.spfw.beans.factory.*;

Import org.spfw.beans.factory.xml.*;

Import org.spfw.beans.factory.config.*;

Import org.spfw.core.io.*;

Public class client

{

Main()

{

Resource new ClassPathResource(“Spconfig.xml”);

XmlBeanFactory factory=new XmlBeanBeanFactory(res);

```
PropertyPlaceholderConfigurer ppc=new PropertyPlacehodConfigurer()
//reading properties from bundle.
PPc setLocation(new classPathResource("jdbc.properties");
//writing datasource properties into bean definition of xml
PPc.postProcessBeanFactory(factory);
SampleBean ob=(SampleBean)factory.getBean("id3");
Ob.update(2356,7900);
}
```



```
}
```

Calble stmts especially used for calling procedure and functions.

HHPServlet abstract class but there is abstract methods.

Calling procedure /function:-

```
Public class Mybean extends Storedprocedure
{
Public mybean(DataSource ds)
```

{

Super(ds,"pro1")

Declareparameter(new Sqlparameter("no",Types.INTEGER));

Declareparameter(new SqlOutParameter("s",Types.DECIMAL));

Declareparameter(new Sqloutparameter("d",Types.INTEGER));

}

Procedure contain input,output,inout

Parameter

Calling procedure and function:-to call a procedure (or)function from a db, we have two approaches in spring, that is extending by StoredProcedure class by with out extended it.

If we want call procedure (or) function then the simple approach is by extending our bean class from StoredProcedure.

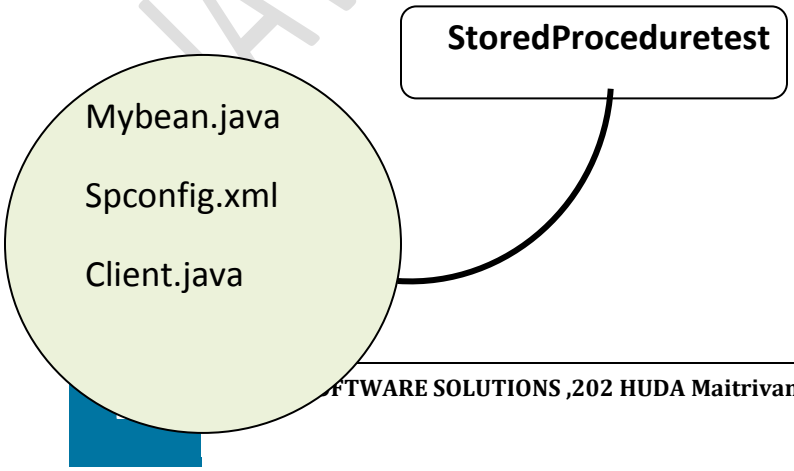
If we want call procedure (or) functions of the dbase then the following steps are need to be followed.

1) Extend the bean class from StoredProcedure.

2) Call the base class constructor from our bean class by passing DataSource object, name of the procedure (or) function as parameter.
3) Declare the keys in the form of sql Parameter objects, that are used as a keys for Map objects while passing input and while getting output.
4) Call execute() from the bm() of the bean by passing the input values in the form of a Map object.
5) Read the output values returned into the Map object from the procedure (or) function.

NOTE:- StoredProcedure is an abstract class and we can get it's featers our class, by only extending our class from that class.

Question:-the following example is for calling a procedure of oracle from the spring bean the procedure takes input as an employee number and return name and experience of output.



*.class

Procedure code:-

SQL> ed demo1

Create (or) replace Procedure Pro1(eno in number, name out vaarchar2, e out number)

Is

D1 date;

D2 date;

Begin

Select ename into name from emp where empno=eno;

Select hiredate into d1 from emp where empno=eno;

Select sysdaate into d2 from dual;

e:=(d2-d1)/365;

```
end;

SQL>@demo1

/

Procedure is created
```

//Mybean.java:-


```
Import org.spfw.jdbc.object.*;

Import org.spfw.jdbc.core.*;

Import java1.sql.*;

Import java.sql.*;

Import java.util.*;

Public class Mybean extend StoredProcedure

{

Public Mybean(DataSource db)

{

Super(ds,”Pro1”);
```

declareParameter(new Sqlparameter(“no”,Types.INTEGER));

declareParameter(new SqlOutparameter(“name”,Types.VARCHAR));

declareParameter(neew SqlOutparameter(“exp”,Types.INTEGER));

}

Public void execute Pro(int empno)

{

Map imp=new HashMap()

Map Omp=execute(imp)

Object o1=omp.get(“name”);

Sop(o1.toString());

Object o2=omp.get(“exp”);

Integer i=(Integer)o2;

Sop(i.intValue());

Sop(“….”);

}

}

Spconfig.xml:-

<beans>

<bean id=”id1” class=”DMDS”>

…

…

</bean>

<bean id=”id2” class=”MyBean”>

<constructor-arg ref=”id1”/>

</bean>

</beans>

Client.java:-

Import org.spfw.context.*;

Import org.spfw.contex.support.*;

Public class client

```
{
Main()
{
ApplicationContext context=new classPathApplication.context(“spconfig.xml”);
Object o=context.getBean(“id2”);
MyBean ob=(Mybean)o;
Ob.executePro(7788)
Ob.executePro(7900)
}
}
```


In the above example we have to used declare Parameter() for creating SQL parameter objects for the keys used in the Moup objects.

Instated of calling declare Parameter() for multiple times we can store all sql parameter objects into an array and then we can set an array by calling setParameter().

We can add the following code inplace of declare parameter calling in the constructed of our became to class.

Sqlparameter Params[]={new SqlParameter{“no”, Types.INTEGER));

New Sqloutparameter (“name”,Types.VARCHAR),new SqloutParameter(“exp”,Types.INTEGER)}

This.setParameter(params);

NOTE:-I don’t want get the connection DMDS and DDS, with out using this class we can get the connection by using connector

Using connection pool in spring:-

When we configure BasicDataSource in to spring configuration xml file then implicitly, BDS,alas has inbuilt connection pool service.

The inbuilt connection pool is only soutable for small scale application, but not for production level application

If a spring application (or) any other java app's want to get a connection from external connection pool, then the application need the mediator object called Data Source, which is configure at server side.

When the DS configure at admistrator side, then with indiregictory with same indicator.

If any java app's want's to get connection pool service of a server then the application as to get the DataSource object from the registory

If an app's want to get Data Source object then it need indi name. see the above diagram

In case of spfw the programmer is not responsible for opening and sdosing the connection . it is done by jdbcTemplate .

jdbcTemplate need indiname of the DataSource Object.

In order to pass indiname fo the DataSourceObject, we need to configure indiObjectFactoryBean

jdbcTemplate is a class which depends on jndiobject FactoryBean for getting a connection from the connection pool.

In the spring Configuration file we need to configure jndiObjectFactoryBean and jdbcTempalte like the following.”>

<bean id=”id1” class=”jndiobjectFaactoryBean

<property name=”indiname” value=”sathyaindi”/>

</bean>

<bean id=”id2” class=”org.spfw.jdbc.core.jdbcTemplate”>

<constructor-arg>

<ref bean=”id”/>

<constructor –arg>

</bean>

Configuring a connectionpool in weblogic server:-

Start →programs→Beanproducts→weblogicSerer9.0→ this is step1

Open the browser and Type the following url

http:// localhost.7001/consale

username-→weblogic

password→weblogic

log

this is step2

step3:-

at leftside expandservice→ expandjdbc→select Data Source→

click on look and edit button→ new→ enter the following details.

Name→ testpool

Indiname→sathyaindi

Databasetype→ oracle

Driver→oracle driver his

Next

Step:-4

Click on next→ enter the following connection property

DatabaseName → sathya

Host→ localhase

Port→ 1521

User→scott

Pass →tiger

NEXT

Step:-5

click on TestConfiguraation button→ next→ select exampleserver check box→ funish→ Activate changes.

Logout

Undil here admin rule is over.

//Mybean.java

Import org.spfw.jdbc.core.*;

Import java.util.*;

```
Public class MyBean
{
Private jdbc Template it;
Public void setit(jdbcTemplate it)
{
This.it=it;
}
Public void loadall()
{
```


```
List l=jt.queryForList("select * from employee")
Iterator it=l.iterator();
While(it.hasNext())
{
Sop(it.next().toString());
```

}

}

}

Spconfig.xml:-

<DTD>

</beans>

<bean id=”id1 class=”org.spfw.jdndw.IndiOectFactoryBean”>

<property name=”indiName” value=”sathyajndi”>

</bean>

<bean

<bean id=”id3” class=”MyBean”/>

<property name=”it” ref=”id2”/>

</bean>

</beans>

In the client application, we have to add the jndi properties to system properties. Because jdbcTemplate class reads jndi properties from system properties, for connecting with the jndiRegistory

In order to add jndi properties to the system properties , we should follow the below steps.

1) Get all

MODULE-3

While developing real time application, spring frame work frequently used with ORM tools because spring bussineslayer need data access layer implemented in spring frame work .

Spring framework an interacted with an orm tools like hibernate, dao,etc.

While integrating spring framework with orm tools we have fallow approaches.

We can directly use the orm api into a spring bean for performing data access layer logic.

We can use the abstraction layer given by spring framework on lop as existing orm tools. For performing data access layer operation.

Orm is an translator or an engine, it acts as a bridge b/w a java application and observer internally translate object into text format and vise versa.

Spring-ORM module provides DAO (data access object) pattern implementation.

The advantage of spring framework is, spring framework as given a single exception hierarchy ,irrespective the persistence technology used.

1) Local session factory :-factory object.
2) Hibernate template :-template design pattern it given API (or) method.
3) My bean:- when the dependency is there we use reject.

NOTE:- while integrating spring framework within hibernate, we use an abstraction layer which io given by spring framework and it internally uses hibernate ORM framework.

Spring framework has provided a central class to perform database operation in the form of objects called hibernate template.

While integrating the spring framework with the hibernate,the hibernate setup required need tobe configure with spring framework provided class called

local session factory bean.

While integrating spring with hibernate we do not required construct hibernate configuration file, we need to configure local session factory bean.

LSFB construct session factory of hibernate internally.

Hibernate template is a class which internally used session API of hibernate

As part of spring hibernates integrating, we configure the fallowing two classes given by spring framework.

1) Local session factory bean.
2) Hibernate template.

The above two classes given in org.spring framework orm. Hibernate 3.x package.

LOCAL SESSION FACTORY BEAN:-

Data source.

Mapping resources.

Hibernate properties.

HIBERNATE .CFOF.XML CONTAIN :-

Connecting properties.

Hibernate properties.

Mapping resources.

Public class local session factory bean.

{

Private datasources datasource,

Private list mapping sources,

Private properties hibernate properties,

}

NOTE:-while configuring Local Session Factory Bean class int spring configuration file, then we need to configuration the fallowing 3 properties.

1) Data source
2) Mapping Resources.
3) Hibernate Properties.

LocalSessionFactoryBean depends on Data Sources object for opening connection with the database.

Mapping Resources is a collection of hibernate mapping files and it is a property of collection type list.

Hibernate properties is a collection of type properties.

LocalSessionFactoryBean clas is containing setter injections for all the properties so at the time of configuring into spring configuration file, we need to use propert elements.

Example:-

<bean id=”id2” class=”org.spring framework.orm.hibernate3.LocalSessionFactoryBean>

<property name=”datasource” ref=”id1”/>

<property name=”mapping Resources”>

<list>

<property>

<property name=”hibernate properties”>

<props>

<prop.key=”dialect”>org.hibernate.dialect.oracllee.dialect<\prop>

<prop key=”hbm2ddl.auto”>update<\prop>

<prop key=”show sal”>true </prop>

</props>

</property>

</bean>

Hibernate template is a class which provide conviect method for the spring bean developer for transfer to and fro data b/w application and db in the form of objects.

Hibernate template depends on LocalSessionFactoryBean in hibernate template we have a property called session factory and it io type of localsessionfactoybean. So we need configure hibernate template like the fallowing.

<beanid=”id3” class=”hibernate template”>

<property name=”sessionFactory” ref=”id2”/>

</bean>

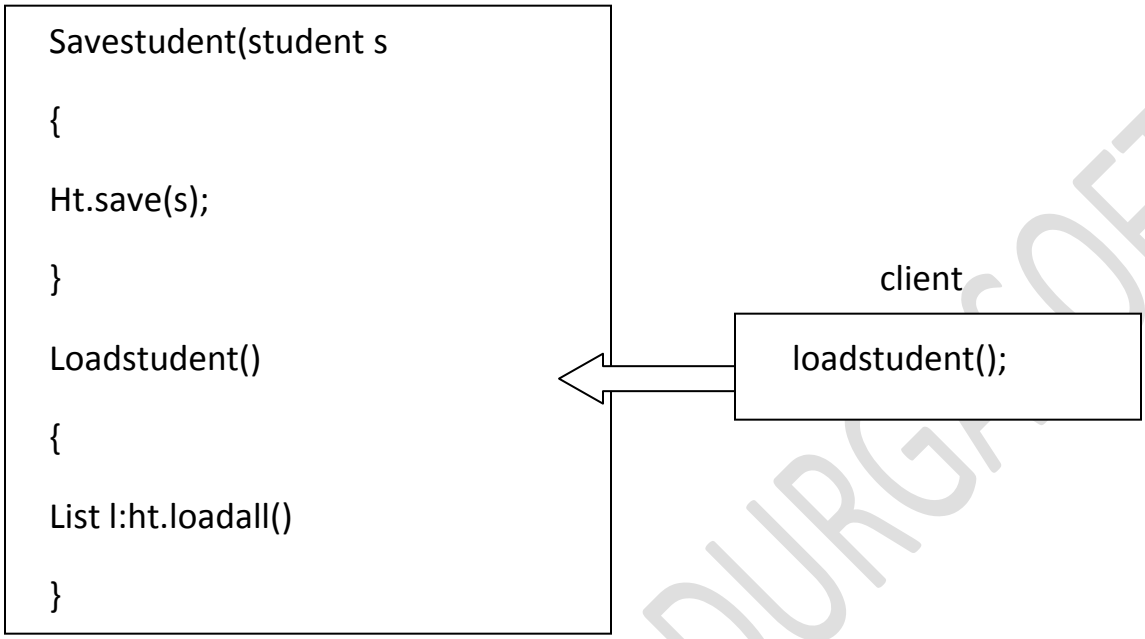
While constructing our spring bean class then it depends on hibernate template .so we need to configure the spring bean class like the fallowing.

<bean id=”id4” class=”my Bean”>

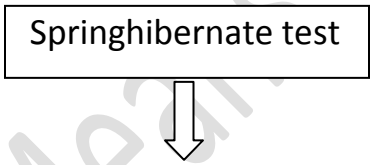
<property name=”hibernate” ref=”id3”\>

<\bean>

Mybean



The following examples is for integrating springbean with hibernate.



Mybean.java

Spconfig.xml

Student.java

Student.hbm.xml

Classeclient.java

Mybean.java

```
Import java.util.*;

Import org.jpfw.orm.hibernate.*;
```

Public class mybean

{

Private hibernate template ht

Public void setHt (hibernate template ht)

{

This.ht=ht;

}

Public void save student(student s)

}

Ht.save(s);-int ht take

Sop("student object java");

}

Public void load student ()

{

```
List l=ht.loadall(student.class)

Iterator it=l.iterator();

While(it.hasnext())

{

Student s=(student)it.next();

Sop(s.gettstudentid()+””+s.getstudentname()+””+s.getmarks());

Sop(“……”);

}

}
```



```
}

Student.java

Public class student

{

Private int studentid;

Private string studentname;
```

```
Private int marks;

Public void setstudentid(int studentid)

{

This.studentid=studentid;

}

Public int getstudentid()

{

Return studentid;

}

}
```


Student.hbm.xml

```
<!DOCTYPE

Hibernate.3.0.dtd>

<hibernate-mapping>

<class name=”student” table=”STUDENT”)

<id ame =”studentid” column=”sid”\>
```

<property name=”studenutname “ column=”sname”\>

<property name=”marks”\>

<class><\hibernate.mapping>

Spconnfig.xml

<beans>

<bean id=”id1”class=”org.spfw.jdbc.datasource.DMDS”>

<property name---driverclass

…….url

……..username

……password

<\bean>

<bean id=”id2” class =”org.spfw.orm.hibernate.LocalSessionFactoryBean”>

<property name=”data source”>

<ref bean=”id1”\>

<\property>

<property name=”mapping Resources”>

<list>

<Value>student.hbm.xm<value>

<List>

<Property>

<property name=”hibernate properties”>

<Props>

<Prop key=”hibernate.dialect”>org.hibernate.dialect.oracle9dialect<\prop>

<prop key=”show .sql>true<\prop>

</props>

</property>

</bean>

<bean id=”id3” class=”org.spfw.orm.hibernate3.HibernatTemplate”>

<property name=”session Factory”>

<ref bean=”id2”\>

</property>

</bean>

<bean id=”id4” class=”my bean”>

<property name=”ht”>

<ref bean=”id3”\>

</property>

</bean>

</beans>

Spring framework is internally uses the hibernate frame work.

//client.java

Import org.spfw.conntext.*;

Import org.spfw.context.support.*;

Public class client

```
{
Public static void main(string[]org)
{
Context=new  CPXAC("spconfig.xml");
Object o=context.getBean("id4");
My Bean ob=(MyBean )o;
Student s=new student();
s.setstudentId(101);
s.setstudentname("aaa");
s.setmarks(300);
ob.savestudent(s);
ob.loadstudent();
}
```



```
}
```

DTO PATTERN:-

Dto pattern says that store the all values in one object and send the object in called DTO designe pattern.

Before run the above application we can set the jar file and create table.

Create table student (sid number(3).....);

The fallowing list of jar files are required to set In the class path.

Spring.jar. 2)common.logging.jar 3)ojdbc14.jar 4)hibernates.jar 5)asm.jar 6)commons-collection.jar 7)cglib.jar 8)domuj.jar 9)jta.jar.

Note:-

Direct integration from spring frame hibernate.

it is also possible interface the spring and hibernate by directly importing hibernate API into spring Bean.

While using direct hibernate API into spring bean then the session Factory of hibernate.should be made as single ton by the programmers.

In order to make the sessionFactory as singleton in a spring bean then we add nit and destroy to the spring bean class.

Then we can implement our bean class From initializing and disposable interface (or) we can include one method for initialization after methods for destroy()

If we add our own methods for inilization and destroy we have to inform the spring.joc container by adding init.method and destroy.method attribute for the bean element

Approach:-

Public class MyBean implements inilializingBean,dioposaleBean

{

Public void after Properties Set()

{

//create sessionFactory of hibernate

}

Public void destroy()

{

//close sessionFactory

}

Public void bm()

{

Open session

Do operation

Close session

}

}

Approach-2

Public class my Bean

{

Public void setup ()

{

//Create session Factory of hibernate

{

Public void fear Down ()

{

```
//close sessionFactory
}
Public void bm()
{
//open session
//do operation
//close session
}
}
```

Xml:--

<bean id=”id1” class=”My Bean” init-method=”setup” destroy-method=”tear Down”>

……

……

Either we can use any thing(orm,jdbc…) are common exception.

Whenever we are communicated with db common for all exception either we can use jdbc (or) ORM tools.

EXCEPTION

↑

Runtime Exception

↑

NestedRuntime Exception

↑

DataAccesssException

i) Bad sql Grammer Exception
ii) Data Integrity Valation Exception
iii) Concurrency Failure Exception
iv) Data Access Resource Failure Exception

MODULE-4

SPRING-AOP(Aspect Oriented Programming)

```
Public class Account
{Public class withdraw()
{
//authentication
//logging
//transaction
//Business logic
}
Public void deposit()
{
//authentication
//logging
//transaction
//exception
}
}
```

cross-cutting functionality

AOP

```
public class Account
{
public void withdraw()
{
//authentication
//logging
//transaction
//business logic
}
Public void deposit
{
```

```
//Business logic

}}
```

NOTE:-

While implementing Business logic for the real time application we need not only Business Logic but also some service to it, to make it as enterprise logic.

While implementing the Business logic, in the business methods, we can combinely implement both the logic and the required services in the bm() method.

According to Spring Framework the services that overlap on the Business Logic are called as cross-cutting concern (or) cross-cutting functionality

If we combinely implement the Business logic and the required services then there are some drawbacks (or) problems.

1) The services and the Business logic in each method, increases the site of a java class. So that complexity increases
2) If the same services are required in another bm() then we cannot reuse those services and in each method separate implementation are required.

3) If any change in some service is require; if we do change and method, then it doesn't effect on the same service implemented in other method. So we need to implement the changes (or) modify individually in each method.
4) Selecting services dynamically at runtime is not possible.
5) In order to overcome the above problems, we need to separate the business logic and the services which are need for the logic as separately we call this separation of Business Logic and the cross-cutting functionality as Aspect-Oriented Programming(AOP)
6) Using AOP the Business Logic and the cross-cutting functionalities are implemented separately and executing at runtime as combinely
7) 

AOP Terminology:--

1) **Aspect**:- An Aspect represents a cross-cutting functionality.
   One real time service require for a b.logic is called Aspect.
   An Aspect denotes only the cross-cutting functionally soul at its implementation
   Ex:- logging Aspect, Transaction aspect
2) **Advice**:- the implementation of Aspect is called advice(logic)
   An advice provides the code for implementation of a service

For example logging is an Aspect and logging is an advice which means logging aspect denotes service name and logging advice denotes implementation of logic

An Advice advices a new behavior for Business logic

3) **Join Point**:- while executing the business logic of a business method, the additional services are need to be injected at diff places are points. We call such points as join points.

At a join point, a new service will be added into the normal flow of business method.

In the execution of business method, the services are required at the following 3 places we call then as the 3 join points.

i) Before Business logic of a methods starts.

ii) After Business logic of a method is completed.

iii) If the Business logic throws an exception at runtime.

In to a join point, an Aspect in injected .If is nothing but, an implementation of an aspect that is an advice

4) **Point cut:-** public class Account

{

withdraw() Authentication

deposit() logging

get Balance () Transaction

}

A point cut defines what advices are required at what join points.

All business methods of a class does not require All services, so a point cut informs the IOC container that we have two types of point cuts in AOP

i) Static point cut.

ii) Dynamic point cut

5) **Introduction**:-an introduction adds new properties and new methods to the already existing class at run time by with out changes to that class.
an introduction introduces new behavior and a new state to the existing classes at run time.

6) **Target**:- public class Account
{
withdraw();
deposit();
getBalance()
}

A target is a class which is going to be advice

A target class is nothing but a class which is not at added with any AOP feature. It means a target is a pre-AOP object .

7) **Proxy**:-a proxy denotes an object that is applied with AOP feature

An object aat pre-AOP Stage is called target and an object at post-AOP Stage is called proxy.

8) **Weaving** :-weaving is a process of applying advices to a target object, to get (or) generating a new proxy object

Through weaving process a target object will be converted to a proxy object by adding advices.

9) **Advisor**:- an advisor what join point what advisor advising advices to various join points

We call an advisor also as pointcut advisor.

We have two types advisor.

i) Static advisor.
ii) Dynamic advisor.

iii)

TYPES OF ADVICES:-AOP means separate logic and separate service and at run time combined both.

i) Before advice
ii) After advice
iii) Throws advice
iv) Around advice

i)Before advice:- this advice containces implementation of a service (or) service, which are need to be apply at before the Business logic of method is going to start.

During compilation time, the services are not apply to the logic and the services are apply at only at run time.

In order to create a before advice our class should implement Method Before Advice interface.

Method Before Advice is an interface given in org.spfw.aop package.

If we implement method Before Advice then we need to override a method called before ()

The services which are implemented in the before () are executed at before the b.l.m()

For example:- public class wellcomeAdvice implements MethodBeforeAdvice.

{

Public void before(Method m, Object ars[] object target)throws Exception

{

```
// implements before services(s);

}

}
```

The first parameter method is a class of java.lang.reflect package and this parameter is used to access the name of the bm () ,thrown get Name()

The second parameter object [ ] is used to access the parameters of bm() the parameters are stored In an object[ ]and those parameter are given to the before method by the container in the form of an object array.

The third parameter Is an object, to whome this services is going to apply.

The services implemented in before advice will be called from a proxy class which generated by the container for a target class.

Example:- public class Demo…..target class created by programmer

```
{

Public void bm()

{

// business logic
```

}

}

Public class Demo Proxy….Proxy class created by container

{

Public void bm()

{

Before(……);

// b.logic

}

}

ii)**After advice**:- This advice contain services, which are apply to b.m() at after b.l.m() is executed.

In order to create an after returning advice in spring, our class should implement an implement called AfterReturningAdvice.

AfterReturningAdvice is an interface given in org.spfw.aop package and we need override method given by this interface called after returning ()

Example:-public class GoodByeAdvice implements AfterReturningAdvice

{

Public void afterReturning(object returnsvalueMethodm,Object arg[ ], object target)

{

// implement after service(s);

}

}

The first parameter contains return value of the b.m() (or) next

If ab.m() return type is void then the return value contains null.

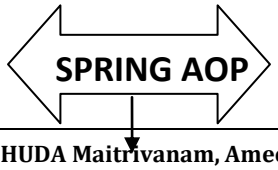
In this afterReturningAdvice, we can access the return value of the business method, and we can modify the return value.

But we cannot return this modified valued back to the client because the return type of afterReturning () is void

***the following examples is to apply both before and after advices to our spring bean business methods.

//TestInter.java

Public void interface TestInter

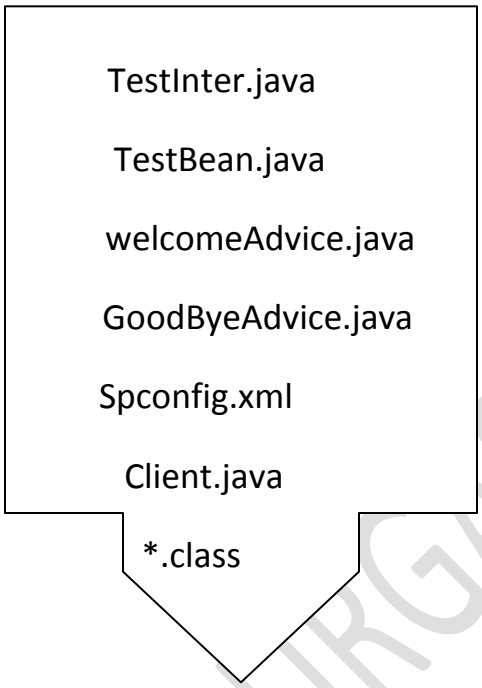


{

Void m1();

Void m2();

}



//TestBean.java

Public class TestBean implements TestInter

{

Public void m1()

{

Sop("I am m1 business logic");

}

```
Public void m2()
{
Sop("I am m2 business logic");
}
}
//welcome Advice.java(before advice)
Import org.sppfw.aop.*;
Import java.lang.reflect.*;
Public class welcome Advice implements MethodBeforeAdvice
{
```


```
Public void before(Method m, object args[ ], object target) throws Exception.
{
Sop(" I am before advice to "+m.getname());
}
}
```

```
//GoodByeAdvicee.java(after advice)

Import java.lang.reflect.*;

Import org.spfw.aop.*;

Public class GoodByeAdvice implements AfterReturningAdvice.

{

Public void afteReturning(object returnvalue,Method m, object args[ ],object
target)throw exception

{

Sop(" I am afterAdvice to "+m.getname());

}

}
```


Spconfig.xml:-

```
<DTD…>

<bean>
```

<bean id=”id1” class=”TestBean”\>

<bean id =”id2” class=”WelcomeAdvice”\>

<bean id=”id3” class=”GoodByeAdvice”**\>**

<bean id=”id4” class=”org.spfw.ao.framework.ProxyFactoryBean”>

<property name=”proxyEnterfaces”>

<value>TestInter<\value>

<property>

<property name=”interfere Names”>

<list>

<Value> id2<\value>

<Value>id3<\value>

<\list>

<\property>

<property name=”target”>

<ref bean=”id1”>

</property>

</bean>

</beans>

Client.java:-

Import org.spfw.context.support*;

Import org.spfw.context.*;

Public class client

{

Main()

{

A c ctrl=CPXA("spconfig.xml");

Object o=ctx.getBean("id4"); ⟶ if we are passing id1we are not getting services

TestInterr ti=(TestInter)o;

Ti.m1();

Sop("……."); ⟶ If we are passing id4 we are getting services.

```
Ti.m2();

}

}
```

V.V.IMP.POINTS:-

1) in spfw,while creating a springbean,we need to follow POJI/POJO Model, it means a spring bean is ex combination f an interface ad its implementation class.

2) In AOP, all ways we need create a spring bean in the form of interface and implementation class only because in AOP, the ioc container internally creates proxy class by implementing the interface.

3) in spconfig file, we need to configure a predefined bean class name called proxyFactoryBean. This class is used to configure our Bean class and it's required advices as a group (or) as a unit.

4)ProxyFactoryBean is internally used by spfw (or) container and generates a proxy class for the target class.

5) in the client application, we are passing id of ProxyFactoryBean to get the object because we need proxy object, but not larget object.

6) in client application, we can not type cast the object into our bean class type. Because the object is created for proxy class internally but not for our bean class.

So we have type cast into interface type because the proxy class also implement the same interface internally.

7)in the above alient app, if we write the following statement then we will get class cast Exception.

TestBean to=(TestBean)o;

8)In the above client app, if we want only target object without services then we need to pass id1 instated of id4 like the following.

Object o=ctx.getBean("id1");

TestInter i=(TestInter)o;

ti.m1();

ti.m2();

o/p:- I am m1 business logic

I am m2 business logic

3)**Thrown Advice**:-In this type of advice, we implement services which are executed whenever the business logic of a method thrown an exception for creating a throws advice, our class must Implement Throws Advice interface.

ThrowsAdvice is a marker interface given in org.spfw.aop package and there are no methods in this interface to provide implementation.

While creating a throwsadvice class In spring AOP, we should implement the services in a method called afterThrowing (), either with single param (or) four params

Actively afterThrowing() is not given by throws advice but we should the implement the services afterThrowing only. Because whenever an exception is occurred in b.l then the ioc container class afterThrowing() to apply the services.

Example:- public class RollBackAdvice implement ThrowsAdvice

{

Public void afterThrowing(Exception e)

{

//services

}

(or)

Public void afterThrowing(Method m, object args[ ],object target,Exception e)

{

//services

}

}

In case of single parameter, only exception details are accessible. But incase of a parameter methods apart from exception details we can also access method name and parameters.

At the time of creating throws advice, we can write an number of times afterThrowing(). If means we can write individual afterThrowing() method separately.

If we write, multiple afterThowing() in a class. The ioc container gives the preference like the following

i) The framework (or) container first verifies whether any afterThrowing() is available with a specific exception type as a parameter (or) not if exist then the container calls afterthrowing() and executed the services implementd in it.

ii) If specific type of afterThrowing() is not available then the container verifies whether afterThrowing() within 4 params exist (or) not if exit then execute services implement in it.

iii) If 4 params afterThrowing() doesnot exit in the class then the container (or) framework verifies whether an afterThrowing() with superclassException parameter (or) not if exit the services implemented in it.

Example:-public class TestBean implement TestInter

{

Public void bm()

{

Int c=10/0;

Sop(c);

}

}

Public class RollBack intents ThrowsAdvice

{

Public void afterThrowing (Exception)

```
{
//services
}
Public void afterThrowingMethodm, object args[],object target
{
//services
}
Public void after Throwing (Arithmetic Exception e)
{
//service
}
```


```
}
```

In above bm() call AmE occurred the container going to executed the services implemented of the RollBackAdvice class.

4) Around Advice:- around advice is the combination both before and after advices

In a single advice it possible both before and after services.

Around advice is not given by spring aop and it is from an open source implementation called aopallience

Around advice can be used by any frame work which is supporting AOP.

To create around advice, our class should implement an interface called Method Interceptor .

In order advice, we implement before and after services in a single, a method called invoke()

In order to separate before and after service and execute b.l() in middle, we call procedure()

Example:- public class MyAroundAdvice implement MethodInterceptor

{

Public Object invoke(MethodInvocation mi)throws Exception

{ before service

Object o=mi.proced()

} After service

}

Note:- around advice is combination of both before and after advice. So we implement only one class enough to implement.

Around advice can access the b.m() value and it can modify the return value and it can return a diff value then original return value back to the client.

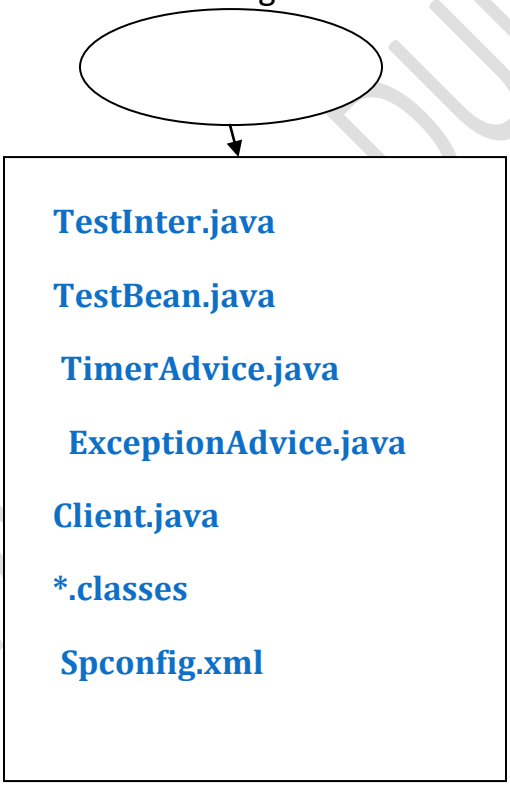
Example:- the following application is to apply both throws advice and around advice to the business method.

If an exception is occurred then throws advice is executed along with around advice.

If no excepting occurred in the b.m() then only around advice is executed, by with out throws advices

In around advice ,we are calculating the time taken to run the business logic.

SpringAOP2



//TestInterr.java

Public interface TestInter

{

```
Void doCalc(int a,int b);

}

//TestBean.java

Public class TestBean implement TestInter

{

Public void docalc(int a, int b)

{

{
```



```
Thread.sleep(5000);

Int c=a/b;

Thread.sleep(4000);

SOP("result="+c);

} catch(Exceptione)

{
```

```
}
}
}
//ExceptionAdvice.java
Import org.spfw.aop*;
Public class ExceptionAdvice implement ThrowsAdvice
{
Public void afterThrowing(Exception)
{
Sop("I am throws advice with Exception Parameter");
}
Public void after Throwing (Arithmetic  Exception)
{
```



```
Sop ("I am Arithmetic Exception);
}
```

```
}
//Timer Advice.java
Import org.aopalliance.intercept.*;
Public class TimerAdvice implements MethodInterceptor
{
Public object invoke(Method Invocation mi) throws Throw able.
{
```


```
Long x=0,y=0,z=0;
Try{
X=system.currentTimeMillis();
Object o=mi.proceed;
Return o;
}
Finally {
Y=system.currentTimeMillis();
```

```
Z=(y-x)/1000;//convert seconds
Sop("time taken="+z+" seconds);
}
}
}
```

Spconfig.xml:-<beans>

```
<bean id="id1" class="TestBean"/>
<bean id="id2" class="ExcceptionAdvice"/>
<bean id="id3" class="TimerAdvice"/>
<bean id="id4" class="org.spfw.aop.framework.ProxyFactoryBean">
<Property name="ProxyInterfaces">
<value>TestInter</value>
</property>
<property name="interceptornames">
<list>
<value>id2</value>
```

<value>id3</value>

</list>

</property>

<property name=”target”>

<ref bean=”id1”/>

</property>

</bean>

</beans>

//client.java:-

Previous

TestInter ti=(TestInter)o;

ti.docalc(10,5);

Sop("…….");

Ti.docalc(10,0);

}

}

Pointcut:-1) while creating b.m() is a spring bean class, all business methods of the class doesn't need advices and all methods of business class doesn't need all advices.

2) in order to find which b.m() of class eligible for getting advices, we need to define a point cut.

3) A point cut verifies whether a particular methods of a particular class is matching with a particular criteria (or) not if matches then that method is identified as eligible for getting advices.

4) in spring AOP, we have two types of point cuts.

i) static point cut ii) dynamic point cut.

STATIC:-In a static point cut, a point cut verifies whether a particular method of a particular class is eligible for getting an advice (or) not .It means the point cut verifies for class name and method, but not run time parameter of the method.

In spring AOP, we have two static point cut classes.

1)NameMatchMethod Point cut

2) RegexpMethod Point cut

The above two classes are predefined static point cut classes given by spring AOP frame work.

The above two classes are just going to verify whether a method name is matching with the given conducted (or) not.

But these two classes doesn't check the class names are matching (or) not.

According to the predefined static point cut, they don't verify class name is matching (or) not and makes all classes are eligible to get advices by default.

As a programmer we can also create our own static point cut, and in that we can verify a class name is matching (or) not and method name is matching (or) not to get an advice.

NAME MATCH METHOD POINT CUT:-

These point cut class is going to verify whether the method names of a spring Bean class are matching with given criteria (or) not

While configure the class into xmal file, we use mapped name (or) Mapped Names property of class into xml file.

This class is a predefined class, so we need to configure the directly into the xml file like the following

Example:- <bean id=”id1
class=”org.spfw.aop.support.NameMatchMethodPointCut”>

<property name=”mapped Name”>

<value>set*</value>

</property>

</bean>

According to the above configuration the point cut class identifies that setter methods of the bean class are eligible for getting advices.

If there is no commonality in the method names then individual methods provided by configuring mapped names property.

Mapped name is a collection of type list. So we need <list> while configuring the collection .

Example:- <bean id=”id1” class=”previous”>

<property name=”mapped names”>

<list>

<value>x</value>

<value>m</value>

<value>z</value>

</list>

</property>

</bean>

According to above configuration only the methods whose name is x, m(or)z are eligible to get advices.

RegoxpMethodPointcut:- This pointcut class is going to verify whether the method name of class is matching with the given Regular expression (or) not. If matches then those methods are eligible to get advices.

While configure this class into xml files we are going to configure any one of the following two properties.

i) Pattern ii) patterns.

Example:- <bean id=”id1” class =”org.spfw.aop.support .RegexpMethodspointcut”>

<property name=”pattern”>

<valu>get.*n.+[0-9]</value>

</property>

</bean>

According to above configuration a method name should start with get and b/w and letter n there should be zero (or) more characters and after N there should be one (or) more character and finally ending must be digit. If a method is satisfies this pattern then it is eligible getting advice.

While constructing the regular expression we use the following symbols.

.- Matchess for a single character.

.*Matches for zero (or) more character.

.+Matches for one (or) more character.

.9 Matches for zero (or) one character.

[A-Z] Matches for an uppercase alphabet from A to Z

[abc] Matches for either a (or) b(or) c

[0-9] Matches for a digit.

If we want configure multiple regular expression.

Then we need to configure patterns property.

Example:- <bean id="id1" class="previous">

<property name="patterns">

<list>

<value>get.*n.+[0-9]</value>

<value>[A-Z].*[0-9]</value>

<values>,*names</valu>

<list>

</property>

</beans>

Point cut advice:-

1) A point cut contain information about what methods are eligible to get advices. But a point cut doesn't know what advice is required for the methods.
2) This information about a point cut needed advices with be stored with an advisor we also called as a point cut advisor.
3) A point cut advisor is the combination of both a point cut and an advisor.

4) In spring AOP frame work, we have totally the following 3 advisor classes for both static and dynamic point cuts.

i) NameMatchedMethodPointcutAdisor.

ii) RegexpMethodPoint CutAdvisor.

iii) Default Point cut Advisor.

5) An advisor while configuring into spring configuration file, we need to pass id of the point cut class and id of the advice class as a reference.

Example:- <bean id=”id1” class=”WelcomeAdvice”/>

<bean id=”id2” class =”org.spfw.aop.support.nameMatchMethodpointcut”>

<property name=”mapped name”>

<value>set*</value>

</property>

</bean>

<bean id=”id3”
class=”org.spfw.aop.support.NameMatchMethodPointcutAdvisor”>

<property name=”point cut’ ref=”id2”/>

<property name=”advice” ref=”id1”/>

The following example is related to a static point cut using both nameMatchMethodpoint cut and RegularMapped point cut.

Spring Aop4

TestInter.java

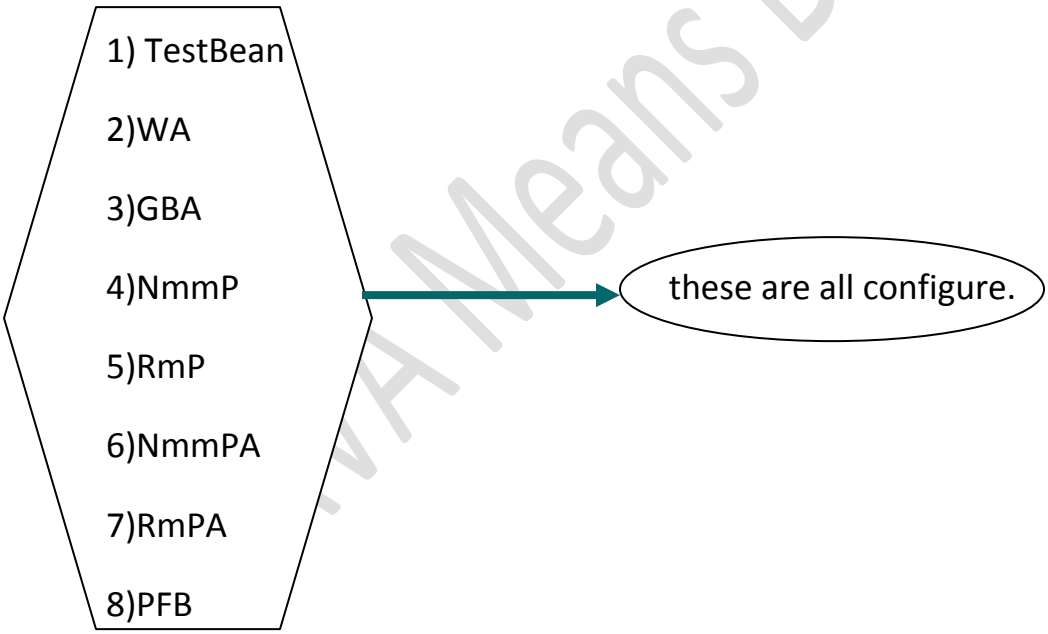
TestBean.java

WelcomeAdvice.java

GoodByeAdvice.xml

Client.java

*.class



//**TestInter.java**:-

Public interface TestInter

{

Void sayHello();

Void sayBye();

Void getData();

Void getMyData();

}

//**TestBean.java**:-

Public class TestBean implement TestInter

{

Public Void sayHello()

{

Sop("I am B.L of say Hello");

}

Public Void sayBye()

{

Sop("I am B.L of sayBye()");

}

Public void getData()

{

Sop("I am B.L of getData()");

}

Public void getMyData()

{

Sop("I am B.L of getMyData()");

}

}

//spconfig.xml:-

<beans>

<bean id="id1" class ="TestBean"/>

```
<bean id="id2" class="WelcomeAdvice"/>
<bean id="id3" class="GoodByeAdvice"/>
<bean id="id4" class="org.spfw.aop.support.NameMatchedMethodPointcut">
<property name="mapped Name">
<value>say*</value>
</property>
</bean>
<bean id="id5" class="org.spfw.aop.support.NameMatchMethodPointcutAdvisor ">
<property name="point cut" ref="id4"/>
<property name="advice" ref="id2"/>
</bean>
```


```
<bean id="id2" class="org.spfw.aop.support.RegexpMethodPointcutAdvisor">
<property name="point cut" ref="id5"/>
<property name="advice" ref="id3"/>
</bean>
```

<bean id=”id8” class=”org.spfw.aop.framework.proxyFactoryBean”>

<property name=”proxyInterfaces”>

<value>TestInter</value>

</Property>

<property name=”interceptorNames”>

<list>

<value>id6</value>

<value>id7</value>

<list>

</property>

<property name=”target”>

<ref ban=”id1”/>

</property>

</bean>

</beans>

Client:-

Object o=ctx.getBean("id8)

TestInter ti=(TestInter)o;

Ti.sayHello()

Sop(".....");

Ti.sayBye();

Sop(".....");

Ti.getData();

Sop("……");

Ti.getMyData();

Sop("…….");

}

}

DynamicPointCut:-

In dynamic pointcut, the business method is eligible to get advise (or) not will be decided. By depending on the run time values like who is calling the b.m() from where the b.m() is called and run time parameters given is the method.

In spring AOP, we have a single pointcut class related to dynamic pointcut is control low Pointcut.

controlFlowPointcut verifies whether the control is coming to a b.m() from a particular method of particular class or not. If the control flow is coming from the given method of given class to the b.m() then the method become eligible to get advice.

While configuring the controlFlowPointcut in the spconfig file we need to configure two construct args class name and method name.

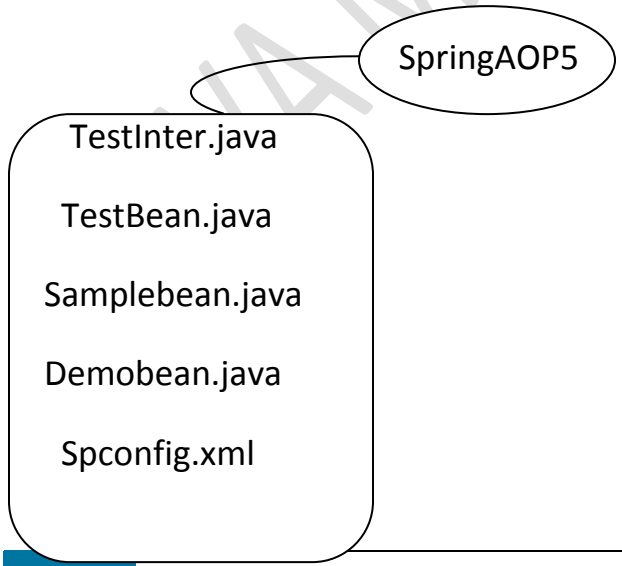
Example:- <bean id=”id1” class=”org.spfw.aop.support.controlFlowPointcut”>

<constructor-arg type=”java.langclass value=”SampleBean”/>

<constructor-arg type=”java.Lang.stringValue=”x”/>

</bean>

** the following example is for controlFlowpointcut , to apply a before advice to the b.m(),if it is called from x() as sample Beans



Client.java

Copy previous code…….welcomeAdvice.java x class.

//testInter.java:-

```
Public Interface TestInter

{

Void bm()

}
```

TestBean.java:-


```
Public class testBean implement TestInter

{

Public void bm()

{

Sop(I am b.l of bm());

}

}
```

//sampleBean.java:-

```
Public class saampleBean

{

Public void x(TestInterti)

}

+I bm();

}

Public void y(TestInter+i)

{

Ti bm();

}
```



```
}
```

//demoBean.java:-

```
Public class DemoBean

{
```

```
{
Public void x(TestInter ti)
{
Ti.bm();
}
}
```

//spconfig.xml:-

```
<beans>
<bean id=”id1” class=”TestBean”/>
<bean id=”id2” class=”SampleBean”/>
<bean id=”id3” class=”DemoBean”/>
<bean id=”id4” class=”WelcomeAdvice”/>
<bean id=”id5” class=”org.spfw.aop.support.controlFlowPointcut”/>
<constructor-arg type=”java.lang.class” value=”sampleBean”/>
</bean>
<bean id=”id6 “ class=”org.spfw.aop.support.DefaultPointcutAdvisor”>
```

<property name=”pointcut ref=”id5”/>

<property name=”advice” ref=”id4”/>

</bean>

<bean id=”id7” class=”org.spfw.ao.framework.proxyFactoryBean”>

<property name=”proxyInterfaces”>

<value>TestInter</value>

</property>

<property name=”inntercepterNames”>

<list>

<value>id6</value>

<list>

</property>

<property name=”target”

<ref bean=”id1”/>

</property>

</bean>

</beans>

Client:-

Object o=ctx.getBen(“id7”);

TestInter ti=(TestInter)o;

Object o=ctx.getBean(“id2”);

sampleBean sb=(sampleBean)o;

object o2=ctx.getBean(“id3”);

DemoBean dd=(DemoBean)o2;

Sb.x(ti);

Sop(“…….”);

Sb.y(ti);

Sop(“……..”);

Db.x(ti);

Sop(“……..”);

Ti.bm();

}

}

Userdefined pointcut:-- apart from existing (or) predefined static and Dynamic point cut class, as a programmer we can also create out own static and dynamic point cut classes.

If we want to create a user defined static pointcut class then our class should extend a base class called staticMethodMatcherPointcut.

If we want to create a dynamic point cut our class should extend base class called DynamicMethodMatcherPointcut.

If we want create a dynamic point cut class of type user define then we need to implement the functionality require in matched() by overriding it.

Matches() returns Boolean Value and by depending on the return Value, a method is decided whether a method is eligible (or) not.

For example:- public class userDynamicPointcut extends dynamicMethodMatcherPointcut.

{

Public Boolean matches(Method m, class c, object args[])

{

Boolean flag=”false”

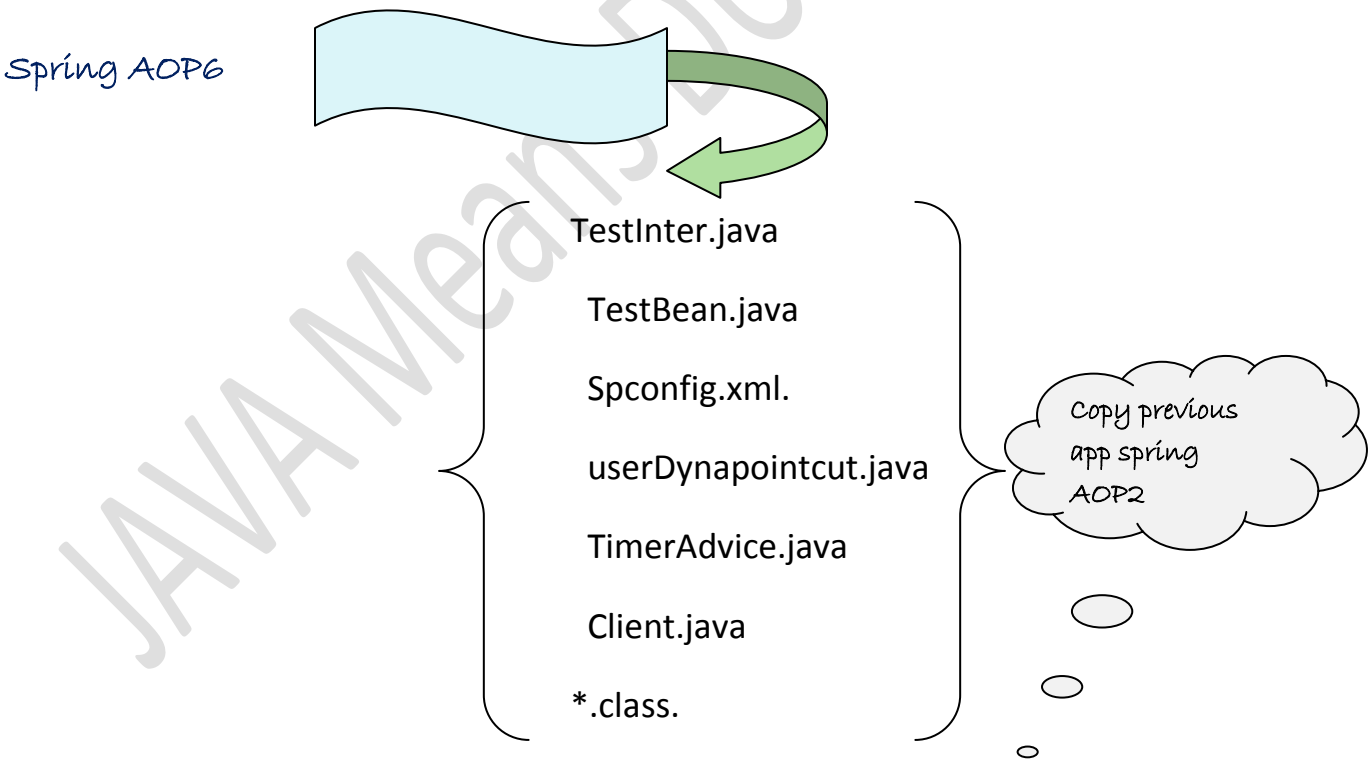
…..

}

}

For userdefined pointcut classes, we need to use default pointcut advisor, to combine pointcut and advise.

Example:- the following application is to apply around advise to findfact() of the bean class, if the run time parameter value given to findfact() is grafer than (or) and 1 to 10.



//TestInter.java:-

Public interface TestInter

{

Void findFact(int k);

Void xyz(int k);

}

//testBean.java:-

Public class TestBean implement TestInter

{

Public void findFact(int k)

{

Int f=1;

For(int i=1,i<=k,i++)

{

F=f*I;

}

Sop("factorial="+f);

```
}
Public void xyz(int k)
{
Sop(" I am B.L of xyz");
}
}
```

//userDynaPointcut.java:-

```
Import java.lang.reflect.*;
Import org.spfw.aop.support.*;
Public class userDynaPoint cut extends DynamicMethodMatchePointcut
{
```



```
Public Boolean matches(Method m,class c object args[])
{
Boolean flag="false";
String str=m.getname();
```

```
Object o=args[0];

Integer i=(Integer)o;

Int x=i.intValue();

If(str.equals(“findFact”)x>=10)

Flag=”true”;

Return flag;

}}
```

//spconfig.xml:-


```
<beans>

<bean id=”id1” class=”TestBean”/>

<bean id=”id2” class=”TimerAdvice”/>

<bean id=”id3” class=”userDynaPointcut”/>

<bean id=”id4” class=”org.spfw.aop.support.DefaultPointcutAdvisor”>
```

<property name=”pointcut” ref=”id3”/>

<property name=”advice” ref=”id2”/>

</bean>

<bean id=id5 class =”org.spfw.aop.framework.proxyFactoryBean”>

<property name=”proxy Interfaces”>

<value>TestInter</value>

</property>

<property name=”interceptor names”>

<list>

<value> id4</value>

</list>

</property>

<property name=”target”>

<ref bean=”id1”/>

</property>

</bean>

```
</bean>

//client.java

Object o=ctx.getBean(“id5);

TestInter ti=(TestInter)o;

Ti.findFact(5);

Sop(“……”);

Ti.xyz(15);

Sop(“……”);

ti.findFact(12);

sop(“……”);

}

}
```

MODULE-5

SPRING WEBMVC MODULE

→when request is sent from a browser then a front controller servlet of springMVC called Dispatcher Servlet traps the given request

→Dispatcher servlet takes the help of HandlerMappingBean,to find a suitable controller for Handling the given request.

→Dispatcher servlet deligates the given request to the ControllerBean

→The Method in which either “business” or intermediate logic defined for a request is executed.

→The method of controller bean returns a ModelAndView(mav) object.

→DispatcherServlet calles a ViewResolverBean to find the appropriate view for sending the response.

→The DispatcherServlet Forwards the request a view.

→Finally the response generated by the view will be a displayed on browser.

Request Flow Diagram:

Note:-DispatcherServlet is a FrontController of SpringMVC it takes the help of Controller bean to process a given request.So a controller is a helper to the Front controller called DispatcherServlet.

Configuring Dispatcher Servlet

→DispatcherServlet is a pre-defined Servlet class which acts as FrontController.

→A request flow of a MVC application in spring will be taken care by DispatcherServlet.

→This servlet class we need to configure in web.xml file of a web application.

→In an MVC application, a spring configuration file will be read by the DispatcherServlet only.

→A DispatcherServlet will automatically load/read a spring configuration file in the following 2 conditions are satisfied.

a)The file name should be <servlet-name>-servlet.xml.

b)The file must be placed in WEB-INF folder.

→If the above Two conditions are not satisfied then we need to configure a context parameter called “contextConfigLocation” and a listener called “ContextLoaderListener” in web.xml file.

Ex:

Web.xml

<web-app>

<servlet>

<servlet-name>spring<servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>spring</servlet-name>

<url-pattern>/*</url-pattern>

</servlet-mapping>

</web-app>

If the spring configuration file name is not following the naming principle or not placed in WEB-INF folder.then we need to add the following two tags in web.xml file.

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/applicationContext</param-value>

</context-param>

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

Note:-if spring 3.x,if spring configuration file is not fallowing a naming rule,instead

Of a Context Parameter and listener, we can also configure an init parameter to DispatcherServlet.

<init-param>

<param-name>contextConfiguration</param-name>

<param-value>/WEB-INF/applicationContext.xml</param-value>

</init-param>

Creating a Controller :

→A controller in SpringMVC is not a frontController it means not C in MVC.

→A Controller in Spring is a helper class to Front Controller in a controller,we defined either business or intermediate logic.

→intermediate logic is a logic to call business logic.

→A controller Bean is like an action class of Struts MVC.

→To create a ControllerBean in Spring 3.x,The following are the 2 rules.

1)we should use @Controller and @Requestmapping annotations.

2)we should place the Controller bean in a package.

Spring MVC InternalResourceViewResolver Configuration Example

In Spring MVC based application, the last step of request processing is to return the logical view name. Here DispatcherServlet has to delegate control to a view template

so the information is rendered. This view template decides that which view should be rendered based on returned logical view name. These view templates are one or more view resolver beans declared in the web application context. These beans have to implement the ViewResolver interface for DispatcherServlet to auto-detect them. Spring MVC comes with severalViewResolver implementations. In this example, we will look at such a view resolver template i.e. **InternalResourceViewResolver**.

In most of the applications, views are mapped to a template's name and location directly. InternalResourceViewResolver helps in mapping the logical view names to directly view files under a certain pre-configured directory. To registerInternalResourceViewResolver, you can declare a bean of this type in the web application context.

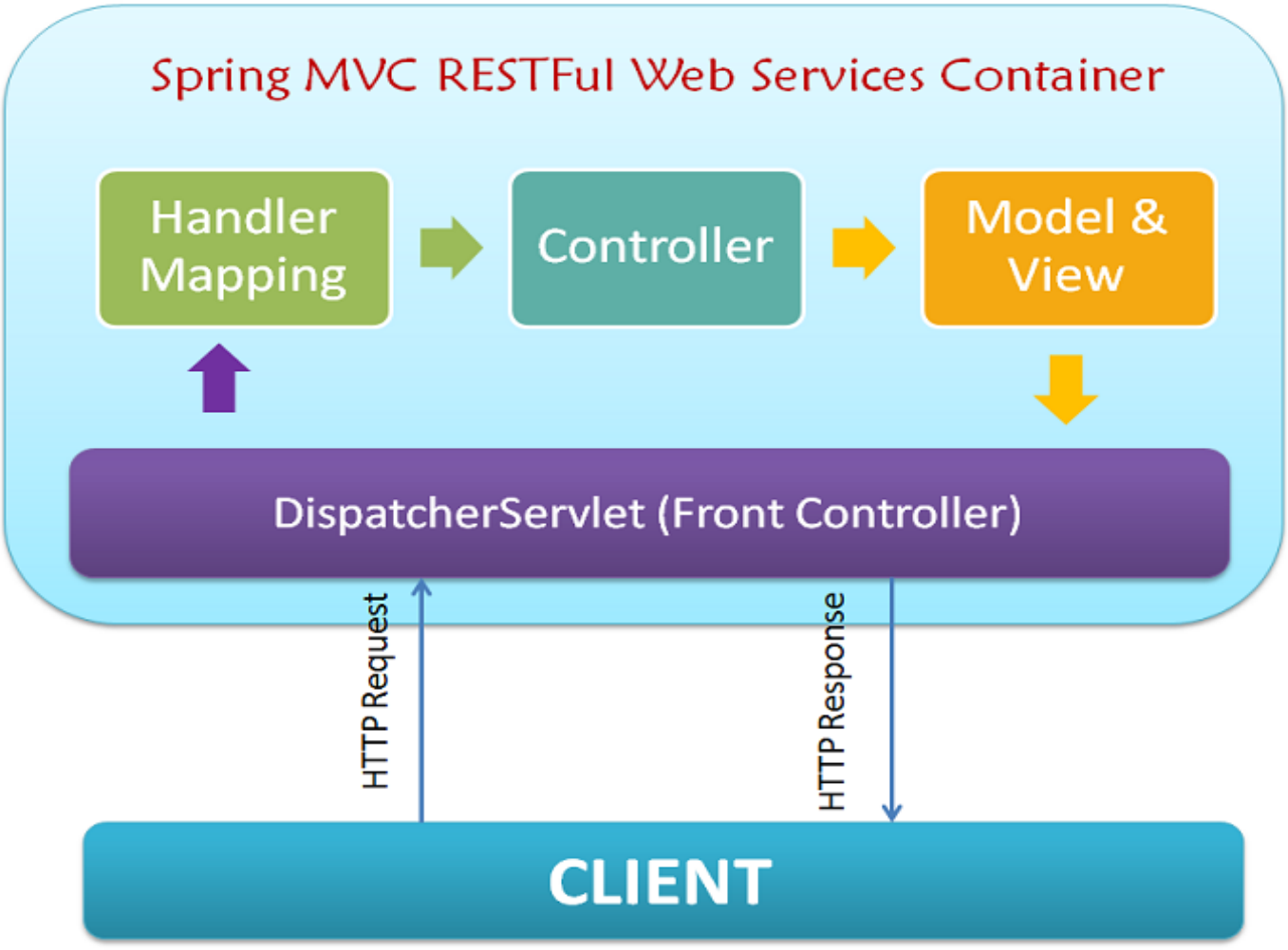

```
<beanclass="org.springframework.web.servlet.view.InternalResourceV
    <propertyname="viewClass"value="org.springframework.web.servle
    <propertyname="prefix"value="/WEB-INF/jsp/"/>
    <propertyname="suffix"value=".jsp"/>
</bean>
```

After above configuration, InternalResourceViewResolver will resolves the view names 'home' and 'admin/home' etc. in the following way.

LOGICAL VIEW NAME	ACTUAL VIEW FILE
home	/WEB-INF/jsp/home.jsp
admin/home	/WEB-INF/jsp/admin/home.jsp
report/main	/WEB-INF/jsp/report/main.jsp

By default, InternalResourceViewResolver resolves view names into view objects of type JstlView if the JSTL library (i.e., jstl.jar) is present in the classpath. So, you can omit the **viewClass** property if your views are JSP templates with JSTL tags. Otherwise you will need to provide a matching **viewClass** i.e. org.springframework.web.servlet.view.tiles2.TilesView if your view is based on tiles.

Spring MVC 4.0 RESTFul Web Services Simple Example



REST(Representational State Transfer) is an architectural style with which Web Services can be designed that serves resources based on the request from client. A Web Service is a unit of managed code, that can be invoked using HTTP requests. Let me put it simple for those who are new to Web Service.You develop the core functionality of your application, deploy it in a server and expose to the network. Once it is exposed, it can be accessed using URI's through HTTP requests from a variety of client applications. Instead of repeating the same functionality in multiple client (web, desktop and mobile) applications, you write it once and access it in all the applications.

Articles on Spring MVSpring MVC & REST

Spring MVC supports REST from version 3.0. It is easier to build restful web services with spring with it's annotation based MVC Framework. In this post, I am going to explain how to build a simple RESTFul web service using Spring MVC 4.0, that would return plain text.

Now, take a look at the architecture diagram above to understand how spring mvc restful web service handles requests from client. The request process flow is as follows,

1. Client application issues request to web service in the form of URI's.

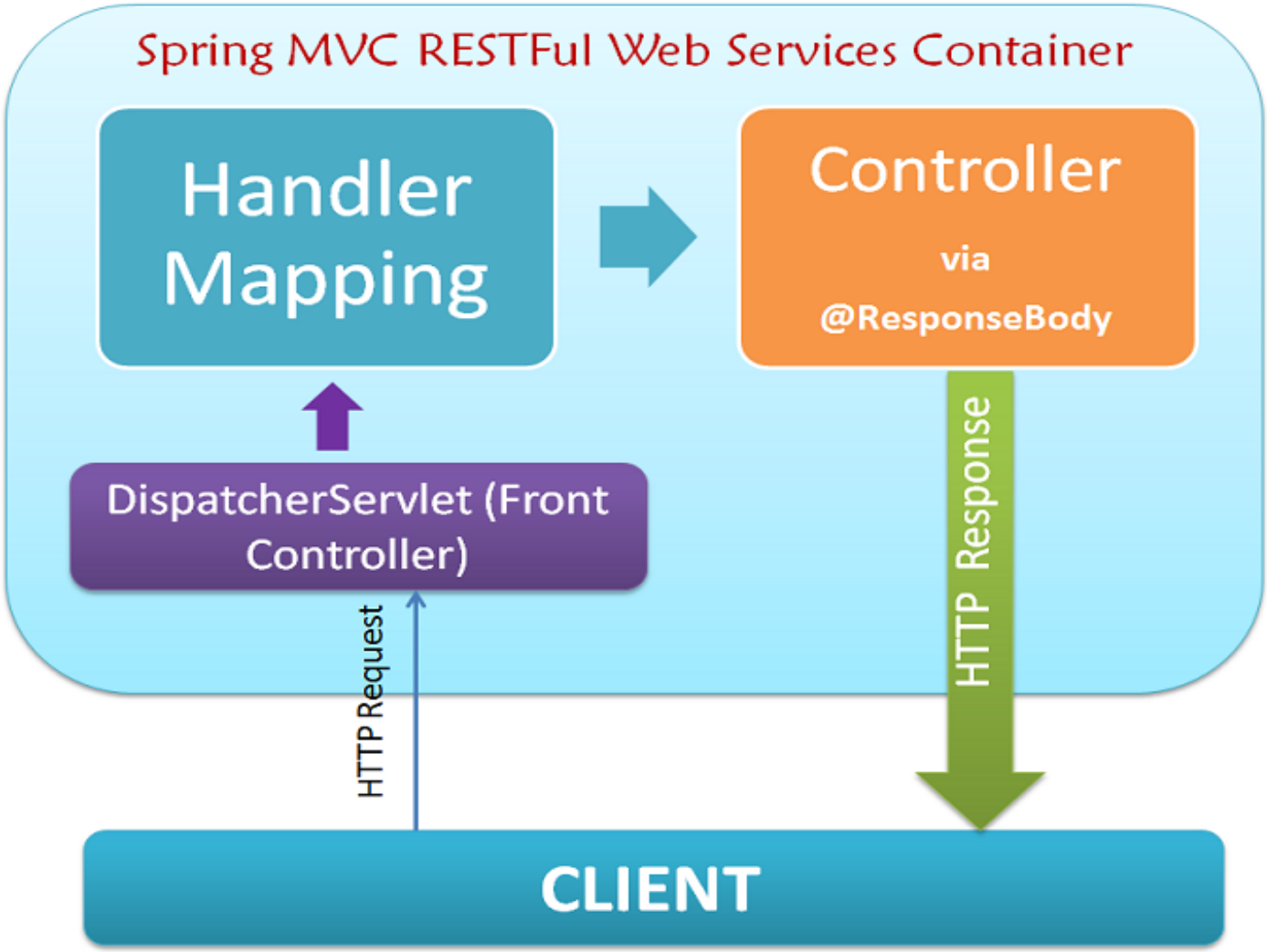
Example: http://example.com/service/greetings/DurgaSoft

2. All HTTP Requests are intercepted by DispatcherServlet (Front End Controller). - This is defined in the web.xml file.

3. DispatcherServlet looks for Handler Mappings. Spring MVC supports three different ways of mapping request URI's to controllers : annotation, name conventions and explicit mappings. Handler Mappings section defined in the application context file, tells DispatcherServlet which strategy to use to find controllers based on the incoming request. More information on Handler Mappings can be found here.

4. Requests are now processed by the Controller and response is returned to DispatcherServlet. DispatcherServlet looks for View Resolver section in the application context file. For RESTFul web services, where your web controller returns ModelAndView object or view names, 'ContentNegotiatingResolver' is used to find the correct data representation format.

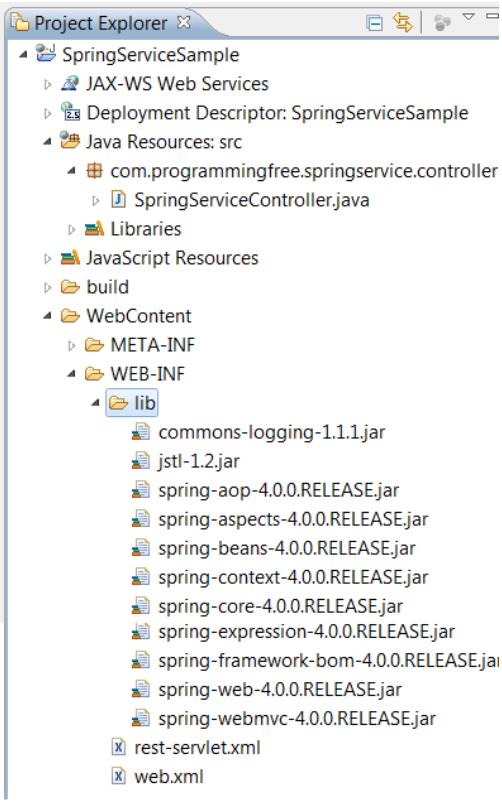
5. There is also an alternate option to the above step. Instead of forwarding ModelAndView object from Controller, you can directly return data from Controller using @ResponseBody annotation as depicted below. You can find more information on using ContentNegotiatingResolver or ResponseBody annotation to return response here.



Its time to get our hands dirty with some real coding. Follow the steps below one by one to get your first Spring MVC REST web service up and running.

1. Download Spring MVC 4.0 jar files from this maven repository here.

2. Create Dynamic Web Project and add the libraries you downloaded to WEB-INF/lib folder.



3. Open /WEB-INF/web.xml file and add below configuration before </webapp> tag.

```
?
1 <servlet>
2  <servlet-name>rest</servlet-name>
3  <servlet-class>
```

```
4    org.springframework.web.servlet.DispatcherServlet
5   </servlet-class>
6   <load-on-startup>1</load-on-startup>
7 </servlet>
8
9 <servlet-mapping>
10  <servlet-name>rest</servlet-name>
11  <url-pattern>/*</url-pattern>
12</servlet-mapping>
```

Note that in the above code,we have named Spring Dispatcher servlet class as "rest" and the url pattern is given as "/*" which means any uri with the root of this web application will call DispatcherServlet. So what's next? DispatcherServlet will look for configuration files following this naming convention - [servlet-name]-servlet.xml. In this example, I have named dispatcher servlet class as "rest" and hence it will look for file named 'rest-servlet.xml'.

4. Now create an xml file under WEB-INF folder and name it 'rest-servlet.xml'. Copy the below in it.

```
?
1 <?xmlversion="1.0"encoding="UTF-8"?>
2 <beansxmlns="http://www.springframework.org/schema/beans"
3   xmlns:context="http://www.springframework.org/schema/context"
4   xmlns:mvc="http://www.springframework.org/schema/mvc"xmlns:xsi=
5 instance"
6   xmlns:p="http://www.springframework.org/schema/p"
```

```
7  xsi:schemaLocation="
8          http://www.springframework.org/schema/beans
9          http://www.springframework.org/schema/beans/spring-beans
10         http://www.springframework.org/schema/context
11         http://www.springframework.org/schema/context/spring-con
12         http://www.springframework.org/schema/mvc
13         http://www.springframework.org/schema/mvc/spring-mvc-4.0
14 <context:component-scanbase-package="com.programmingfree.spring
15 <mvc:annotation-driven/>
    </beans>
```


With ComponentScan tag, Spring auto scans all elements in the provided base package and all its child packages for Controller servlet. Also, we have used <mvc:annotation-driven> tag instead of ViewResolver, with which we can directly send response data from the controller.

5. Let us create a controller servlet in the package mentioned in the component-scan tag. Copy and paste the below code in it.

```
?
1 package com.programmingfree.springservice.controller;
2
3
4 import
5 org.springframework.web.bind.annotation.PathVariable;
6 import
7 org.springframework.web.bind.annotation.RequestMapping;
```

```
8 import
9 org.springframework.web.bind.annotation.RequestMethod;
10import
11org.springframework.web.bind.annotation.RestController;
12
13
14@RestController
15@RequestMapping("/service/greeting")
16public class SpringServiceController {
17 @RequestMapping(value = "/{name}", method =
18RequestMethod.GET)
   public String getGreeting(@PathVariable String name) {
    String result="Hello "+name;
    return result;
   }
  }
```


'@RequestMapping' annotation is used for defining incoming request urls for class and method levels. In this case all uri's in the format `<root-url>/service/greeting/` will be routed to this Controller class. With *@RequestMapping* annotation, we can only define generic uri mappings. For dynamic uri mappings in case of passing variables along with the uri, *@PathVariable* is used. Here in this case, we pass a variable 'name' along with the uri such as, `<root-url>/service/greeting/Priya`. Here the last parameter (Priya) in the uri is retrieved using *@PathVariable.*

I explained that while using <mvc:annotation-config> tag instead of view resolver, we use *'@ResponseBody'*annotation to return response data directly from controller.

But in the above code, I have not used *'@ResponseBody'*. This is because, in Spring MVC 4.0, they have introduced *'@RestController'* such that we need not use *'@ResponseBody'* tag in each and every method. *'@RestController'* will handle all of that at the type level.

This annotation simplifies the controller and it has '*@Controller*' and '*@ResponseBody*' annotated within itself.

Let us take a look at the same controller but with Spring MVC 3.0,

?

```
  @Controller
1 @RequestMapping("/service/greeting")
2 public class SpringServiceController {
3  @RequestMapping(value = "/{name}", method =
4 RequestMethod.GET)
5  public @ResponseBody String
6 getGreeting(@PathVariable String name) {
7   String result="Hello "+name;
8   return result;
9  }
  }
```

Note that in Spring MVC 3.0 we had to expicitly use *@Controller* annotation to specify controller servlet and *@ResponseBody* annotation in each and every method.

With the introduction of *'@RestController'*annotation in Spring MVC 4.0, we can use it in place of *@Controller* and *@ResponseBody* annotation.

That is all! Simple RESTFul Web Service returning greeting message based on the name passed in the URI, is built using Spring MVC 4.0.

So now run this application in Apache Tomcat Web Server.

Open Web Browser, and hit this url, 'http://localhost:8080/<your-application-name>/service/greeting/<anyname> . If you have downloaded this sample application, then use, http://localhost:8080/SpringServiceSample/service/greeting/priya . You will be seeing greeting message sent as response from server,

Spring Restful Web Service Example with JSON, Jackson and Client Program

Spring is one of the most widely used Java EE framework. We have earlier seen how to use Spring MVC to create Java based web applications. Today we will use Spring MVC to create Restful web application and then test it out with the Rest client. At the end, we will also look into how to invoke Spring Restful web service using **Spring RestTemplate API**.

We will use Spring latest version **4.0.0.RELEASE** and utilize Spring Jackson JSON API integration to send JSON response in the rest call response. The tutorial is developed in Spring STS IDE for creating Spring MVC skeleton code easily and then extended to implement Restful architecture.

Create a new Spring MVC Project in the STS, our final project will look like below image. We will look into each of the components one by one.

Configuration XML Files

Our pom.xml file looks like below.

pom.xml

```
1  <?xmlversion="1.0"encoding="UTF-8"?>
2  <projectxmlns="http://maven.apache.org/POM/4.0.0"xmlns:xsi="http://www.w3.c
```

```
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apac
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.journaldev</groupId>
    <artifactId>SpringRestExample</artifactId>
    <name>SpringRestExample</name>
    <packaging>war</packaging>
    <version>1.0.0-BUILD-SNAPSHOT</version>
    <properties>
```



```
        <java-version>1.6</java-version>
        <org.springframework-version>4.0.0.RELEASE</org.springframework-ver
        <org.aspectj-version>1.7.4</org.aspectj-version>
        <org.slf4j-version>1.7.5</org.slf4j-version>
        <jackson.databind-version>2.2.3</jackson.databind-version>
    </properties>
    <dependencies>
        <!-- Jackson -->
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
```

```
        <artifactId>jackson-databind</artifactId>
        <version>${jackson.databind-version}</version>
    </dependency>
    <!-- Spring -->
    <dependency>
```



```
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>${org.springframework-version}</version>
        <exclusions>
            <!-- Exclude Commons Logging in favor of SLF4j -->
            <exclusion>
                <groupId>commons-logging</groupId>
                <artifactId>commons-logging</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
    <dependency>
```

```
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>${org.springframework-version}</version>
    </dependency>
```


```
    <!-- AspectJ -->
    <dependency>
        <groupId>org.aspectj</groupId>
        <artifactId>aspectjrt</artifactId>
        <version>${org.aspectj-version}</version>
    </dependency>

    <!-- Logging -->
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
        <version>${org.slf4j-version}</version>
```

```
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>jcl-over-slf4j</artifactId>
            <version>${org.slf4j-version}</version>
            <scope>runtime</scope>
        </dependency>
        <dependency>
```



```
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-log4j12</artifactId>
            <version>${org.slf4j-version}</version>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
            <version>1.2.15</version>
            <exclusions>
```

```
95                  <exclusion>
96                      <groupId>javax.mail</groupId>
97                      <artifactId>mail</artifactId>
98                  </exclusion>
99                  <exclusion>
100                     <groupId>javax.jms</groupId>
101                     <artifactId>jms</artifactId>
102                 </exclusion>
103
104
105
106
107
108
109                 <exclusion>
110                     <groupId>com.sun.jdmk</groupId>
111                     <artifactId>jmxtools</artifactId>
112                 </exclusion>
113                 <exclusion>
114                     <groupId>com.sun.jmx</groupId>
115                     <artifactId>jmxri</artifactId>
116                 </exclusion>
117
```

```
118             </exclusions>
119             <scope>runtime</scope>
120         </dependency>
121
122         <!-- @Inject -->
123         <dependency>
124             <groupId>javax.inject</groupId>
125             <artifactId>javax.inject</artifactId>
126             <version>1</version>
127         </dependency>
```


```
134         <!-- Servlet -->
135         <dependency>
136             <groupId>javax.servlet</groupId>
137             <artifactId>servlet-api</artifactId>
138             <version>2.5</version>
139             <scope>provided</scope>
140
```

```
141         </dependency>
142         <dependency>
143             <groupId>javax.servlet.jsp</groupId>
144             <artifactId>jsp-api</artifactId>
145             <version>2.1</version>
146             <scope>provided</scope>
147         </dependency>
148         <dependency>
149             <groupId>javax.servlet</groupId>
150             <artifactId>jstl</artifactId>
151             <version>1.2</version>
152         </dependency>
```



```
158         <!-- Test -->
159         <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.7</version>
            <scope>test</scope>
```

```
    </dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
            <artifactId>maven-eclipse-plugin</artifactId>
            <version>2.9</version>
            <configuration>
```



```
                <additionalProjectnatures>
                    <projectnature>org.springframework.ide.eclipse.core
                </additionalProjectnatures>
                <additionalBuildcommands>
                    <buildcommand>org.springframework.ide.eclipse.core
                </additionalBuildcommands>
                <downloadSources>true</downloadSources>
                <downloadJavadocs>true</downloadJavadocs>
            </configuration>
```

```
</plugin>
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>2.5.1</version>
    <configuration>
        <source>1.6</source>
        <target>1.6</target>
        <compilerArgument>-Xlint:all</compilerArgument>
        <showWarnings>true</showWarnings>
        <showDeprecation>true</showDeprecation>
    </configuration>
</plugin>
```



```
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>exec-maven-plugin</artifactId>
    <version>1.2.1</version>
    <configuration>
```

```
                    <mainClass>org.test.int1.Main</mainClass>
                </configuration>
            </plugin>
        </plugins>

    </build>
</project>
```

.

web.xml

```
1  <?xmlversion="1.0"encoding="UTF-8"?>
2  <web-appversion="2.5"xmlns="http://java.sun.com/xml/ns/javaee"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
   http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
5
6      <!-- The definition of the Root Spring Container shared by all Servlet
```

```
and Filters -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring/root-context.xml</param-value>
    </context-param>
```


```
    <!-- Creates the Spring Container shared by all Servlets and Filters -
    <listener>
        <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-clas
    </listener>

    <!-- Processes application requests -->
    <servlet>
        <servlet-name>appServlet</servlet-name>
        <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
```

```
30              <param-name>contextConfigLocation</param-name>
31              <param-value>/WEB-INF/spring/appServlet/servlet-
    context.xml</param-value>
32          </init-param>
33          <load-on-startup>1</load-on-startup>
        </servlet>


        <servlet-mapping>
            <servlet-name>appServlet</servlet-name>
            <url-pattern>/</url-pattern>
        </servlet-mapping>


    </web-app>
```


root-context.xml

```
1   <?xmlversion="1.0"encoding="UTF-8"?>
2   <beansxmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd">
```

```
5
6       <!-- Root Context: defines shared resources visible to all other web c
7
8   </beans>
```

servlet-context.xml

```
1 <?xmlversion="1.0"encoding="UTF-8"?>
2 <beans:beansxmlns="http://www.springframework.org/schema/mvc"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xmlns:beans="http://www.springframework.org/schema/beans"
5       xmlns:context="http://www.springframework.org/schema/context"
6       xsi:schemaLocation="http://www.springframework.org/schema/mvc http://ww
7           http://www.springframework.org/schema/beans http://www.springframew
8           http://www.springframework.org/schema/context http://www.springfram
9
10      <!-- DispatcherServlet Context: defines this servlet's request-processi
11
```

```
12      <!-- Enables the Spring MVC @Controller programming model -->
13      <annotation-driven/>
14
15      <!-- Handles HTTP GET requests for /resources/** by efficiently serving
  ${webappRoot}/resources directory -->
16      <resourcesmapping="/resources/**"location="/resources/"/>
17
```



```
25      <!-- Resolves views selected for rendering by @Controllers to .jsp reso
26      <beans:beanclass="org.springframework.web.servlet.view.InternalResourceV
27          <beans:propertyname="prefix"value="/WEB-INF/views/"/>
28          <beans:propertyname="suffix"value=".jsp"/>
29      </beans:bean>
30
31      <!-- Configure to plugin JSON as request and response in method handler
32      <beans:beanclass="org.springframework.web.servlet.mvc.method.annotation
33          <beans:propertyname="messageConverters">
34
```

```
35              <beans:list>
36                  <beans:refbean="jsonMessageConverter"/>
37              </beans:list>
38          </beans:property>
39      </beans:bean>
```


```
    <!-- Configure bean to convert JSON to POJO and vice versa -->
    <beans:beanid="jsonMessageConverter"class="org.springframework.http.con
    </beans:bean>

    <context:component-scanbase-package="com.journaldev.spring.controller"/

</beans:beans>
```

Most of the part is auto generated and contains boiler-plate configurations. However important points to note are **annotation-driven** element to support annotations based configuration and plugging in`MappingJackson2HttpMessageConverter` to the `RequestMappingHandlerAdapter` `messageConverters` so that Jackson API kicks in and converts

JSON to Java Beans and vice versa. By having this configuration, we will be using JSON in request body and we will receive JSON data in the response.

Model Class

Let's write a simple POJO class that will serve as input and output to our Restful web service methods.

Employee.java

```
1  packagecom.journaldev.spring.model;
2
3
4
5
6
7
8  importjava.io.Serializable;
9  importjava.util.Date;
10
10 importcom.fasterxml.jackson.databind.annotation.JsonSerialize;
11 importcom.fasterxml.jackson.databind.ser.std.DateSerializer;
11
12 publicclassEmployee implementsSerializable{
13
13     privatestaticfinallongserialVersionUID = -
   7788619177798333712L;
14
```

```
    privateintid;

    privateString name;

    privateDate createdDate;
```



```
    publicintgetId() {
        returnid;
    }
    publicvoidsetId(intid) {
        this.id = id;
    }
    publicString getName() {
        returnname;
    }
    publicvoidsetName(String name) {
        this.name = name;
    }


    @JsonSerialize(using=DateSerializer.class)
```

```
    publicDate getCreatedDate() {
        returncreatedDate;
    }
    publicvoidsetCreatedDate(Date createdDate) {
        this.createdDate = createdDate;
    }

}
```

The only important point to note is the use of `@JsonSerialize` annotation to use `DateSerializer` class for Date conversion from Java type to JSON format and vice versa.

Spring Restful web service End Points

We will have following rest web services end points.

Sl. No	URI	HTTP Method	Details
1	/rest/emp/dummy	GET	Health Check service, to insert a dummy data in the Employees data storage
2	/rest/emp/{id}	GET	To get the Employee object based on the id
3	/rest/emps	GET	To get the list of all the Employees in the data store
4	/rest/emp/create	POST	To create the Employee object and store it
5	/rest/emp/delete/{id}	PUT	To delete the Employee object from the data storage based on the id

We have a class defining all these URI as String constants.

EmpRestURIConstants.java

```
1  packagecom.journaldev.spring.controller;
2
3  publicclassEmpRestURIConstants {
4
5      publicstaticfinalString DUMMY_EMP = "/rest/emp/dummy";
6      publicstaticfinalString GET_EMP = "/rest/emp/{id}";
7      publicstaticfinalString GET_ALL_EMP = "/rest/emps";
8      publicstaticfinalString CREATE_EMP =
"/rest/emp/create";
9      publicstaticfinalString DELETE_EMP =
```

```
10  "/rest/emp/delete/{id}";
    }
```

Spring Restful web service Controller class

Our EmployeeController class will publish all the web service end points mentioned above, let's look at the code of the class and then we will learn about each of the methods in detail.

EmployeeController.java

```
1
2 packagecom.journaldev.spring.controller;
3
4 importjava.util.ArrayList;
5 importjava.util.Date;
6 importjava.util.HashMap;
7 importjava.util.List;
8 importjava.util.Map;
9 importjava.util.Set;
10
```


```
11
12
13
14
15
16
17
18 importorg.slf4j.Logger;
19 importorg.slf4j.LoggerFactory;
20 importorg.springframework.stereotype.Controller;
21 importorg.springframework.web.bind.annotation.PathVariable;
22 importorg.springframework.web.bind.annotation.RequestBody;
23 importorg.springframework.web.bind.annotation.RequestMapping;
24 importorg.springframework.web.bind.annotation.RequestMethod;
25 importorg.springframework.web.bind.annotation.ResponseBody;
26
27 importcom.journaldev.spring.model.Employee;
28
29 /**
30  * Handles requests for the Employee service.
31  */
32 @Controller
33 publicclassEmployeeController {
```

```
34
35      privatestaticfinalLogger logger =
  LoggerFactory.getLogger(EmployeeController.class);
36
37
        //Map to store employees, ideally we should use database
38
        Map<Integer, Employee> empData = newHashMap<Integer, Employee>();
39
40
41
42
43
44
45
46
        @RequestMapping(value = EmpRestURIConstants.DUMMY_EMP, method =
47  RequestMethod.GET)
48      public@ResponseBodyEmployee getDummyEmployee() {
49          logger.info("Start getDummyEmployee");
50          Employee emp = newEmployee();
51          emp.setId(9999);
52          emp.setName("Dummy");
53          emp.setCreatedDate(newDate());
54          empData.put(9999, emp);
55          returnemp;
        }
56
```

```
57
58      @RequestMapping(value = EmpRestURIConstants.GET_EMP, method =
  RequestMethod.GET)
59
        public@ResponseBodyEmployee getEmployee(@PathVariable("id")
60  intempId) {
61          logger.info("Start getEmployee. ID="+empId);
62
63          returnempData.get(empId);
64      }
```



```
71
        @RequestMapping(value = EmpRestURIConstants.GET_ALL_EMP, method =
72  RequestMethod.GET)
73      public@ResponseBodyList<Employee> getAllEmployees() {
74          logger.info("Start getAllEmployees.");
75          List<Employee> emps = newArrayList<Employee>();
76          Set<Integer> empIdKeys = empData.keySet();
77          for(Integer i : empIdKeys){
                emps.add(empData.get(i));
            }
```

```
        returnemps;

    }


    @RequestMapping(value = EmpRestURIConstants.CREATE_EMP, method =
RequestMethod.POST)

    public@ResponseBodyEmployee createEmployee(@RequestBodyEmployee
emp) {

        logger.info("Start createEmployee.");

        emp.setCreatedDate(newDate());

        empData.put(emp.getId(), emp);

        returnemp;

    }
```


```
    @RequestMapping(value = EmpRestURIConstants.DELETE_EMP, method =
RequestMethod.PUT)

    public@ResponseBodyEmployee deleteEmployee(@PathVariable("id")
intempId) {
```

```
        logger.info("Start deleteEmployee.");
        Employee emp = empData.get(empId);
        empData.remove(empId);
        returnemp;
    }
}
```

For simplicity, I am storing all the employees data in the HashMap empData. @RequestMapping annotation is used to map the request URI to the handler method. We can also specify the HTTP method that should be used by client application to invoke the rest method.

@ResponseBody annotation is used to map the response object in the response body. Once the response object is returned by the handler method, MappingJackson2HttpMessageConverter kicks in and convert it to JSON response.

@PathVariable annotation is the easy way to extract the data from the rest URI and map it to the method argument.

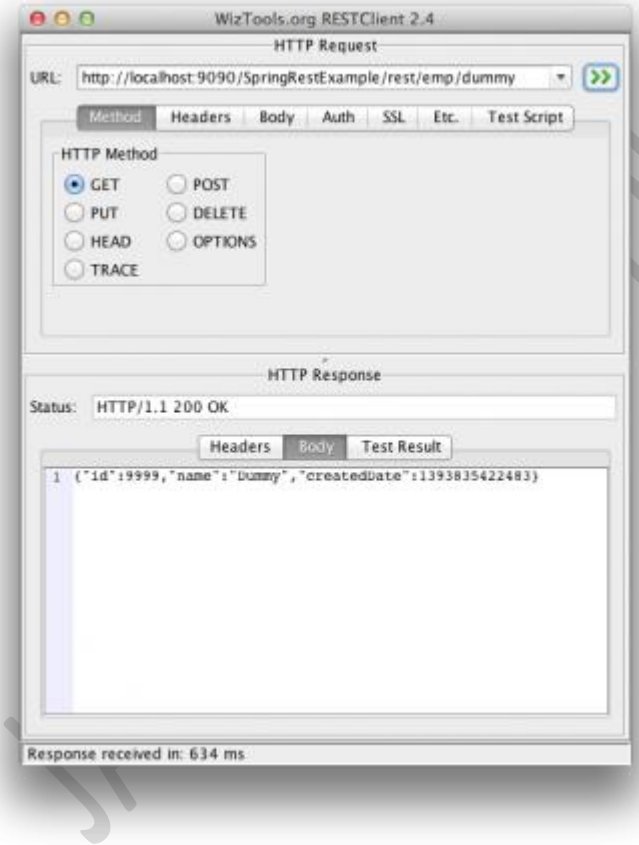
@RequestBody annotation is used to map the request body JSON data into the Employee object, again this is done by the MappingJackson2HttpMessageConverter mapping.

Rest of the code is simple and self understood, our application is ready for deployment and testing. Just export as WAR file and copy it in the servlet container web app directory. If you have server configured in the STS, you can simply run it on the server to get it deployed.

I am using WizTools RestClient to invoke the rest calls but you can also use Chrome extension Postman.

Below screenshots shows the different invocations of the rest apis exposed by our application and it's output.

Health Check – Get Dummy Employee Rest Call



Create Employee POST Rest Call: Make sure request Content-Type is set to "application/json" otherwise you will get HTTP Error Code 415.

Get Employee Rest Call

Delete Employee Rest Call

Get All Employees Rest Call

Spring Rest Client Program

Rest Clients are good to test our rest web service but most of the times, we need to invoke rest services through our program. We can use Spring `RestTemplate` to invoke these methods easily. Below is a simple program invoking our application rest methods using RestTemplate API.

TestSpringRestExample.java

```
1 packagecom.journaldev.spring;
2
3 importjava.util.LinkedHashMap;
4 importjava.util.List;
5
6 importorg.springframework.web.client.RestTemplate;
```

```
7
8 importcom.journaldev.spring.controller.EmpRestURIConstants;
9 importcom.journaldev.spring.model.Employee;
10
11publicclassTestSpringRestExample {
12
13    publicstaticfinalString SERVER_URI = "http://localhost:9090/SpringRestE
14
15    publicstaticvoidmain(String args[]){
16
17
18
19
20
21        testGetDummyEmployee();
22        System.out.println("*****");
23        testCreateEmployee();
24        System.out.println("*****");
25        testGetEmployee();
26        System.out.println("*****");
27        testGetAllEmployee();
28    }
29
```

```
30    privatestaticvoidtestGetAllEmployee() {
31        RestTemplate restTemplate = newRestTemplate();
32        //we can't get List<Employee> because JSON convertor doesn't know t
33        //object in the list and hence convert it to default JSON object ty
34        List<LinkedHashMap> emps = restTemplate.getForObject(SERVER_URI+Empl
35        System.out.println(emps.size());
36        for(LinkedHashMap map : emps){
37            System.out.println("ID="+map.get("id")+",Name="+map.get("name")
38        }
39    }
```



```
46    privatestaticvoidtestCreateEmployee() {
47        RestTemplate restTemplate = newRestTemplate();
48        Employee emp = newEmployee();
49        emp.setId(1);emp.setName("Pankaj Kumar");
50        Employee response = restTemplate.postForObject(SERVER_URI+EmpRestUR
  Employee.class);
51        printEmpData(response);
52
```

```
53      }
54
55      privatestaticvoidtestGetEmployee() {
56          RestTemplate restTemplate = newRestTemplate();
57          Employee emp = restTemplate.getForObject(SERVER_URI+"/rest/emp/1", 
58          printEmpData(emp);
59      }
60
        privatestaticvoidtestGetDummyEmployee() {
            RestTemplate restTemplate = newRestTemplate();
            Employee emp = restTemplate.getForObject(SERVER_URI+EmpRestURIConst
            printEmpData(emp);
        }

        publicstaticvoidprintEmpData(Employee emp){
            System.out.println("ID="+emp.getId()+",Name="+emp.getName()+",Creat
```

```
    }
}
```

Most of the program is simple to understand, however when invoking rest method returning a Collection, we need to use `LinkedHashMap` because JSON to object conversion doesn't know about the Employee object and converts it to the collection of `LinkedHashMap`. We can write a utility method to convert from`LinkedHashMap` to our Java Bean object.

When we run above program, we get following output in the console.

```
ID=9999,Name=Dummy,CreatedDate=Tue Mar 04 21:02:41 PST 2014
*****
ID=1,Name=Pankaj Kumar,CreatedDate=Tue Mar 04 21:02:41 PST 2014
*****
ID=1,Name=Pankaj Kumar,CreatedDate=Tue Mar 04 21:02:41 PST 2014
*****
2
ID=1,Name=Pankaj Kumar,CreatedDate=1393995761654
ID=9999,Name=Dummy,CreatedDate=1393995761381
```

workspace for practice :