Spring practical

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

Hello world program

Client.java

**package** beans;

**import** java.util.ResourceBundle;

**import** org.springframework.beans.factory.BeanFactory;

**import** org.springframework.beans.factory.xml.~~XmlBeanFactory~~;

**import** org.springframework.core.io.ClassPathResource;

**import** org.springframework.core.io.Resource;

**public** **class** Client {

**public** **static** **void** main(String[] args) {

System.*out*.println("excecution started from main");

Resource bundle = **new** ClassPathResource("resource/spring.xml");

System.*out*.println("found the resource");

BeanFactory beanFactory = **new** XmlBeanFactory(bundle);

System.*out*.println("container object created");

Object bean = beanFactory.getBean("t");

System.*out*.println("got the bean object");

Test t = (Test) bean;

System.*out*.println("calling hello method");

t.hello();

System.*out*.println("execution ended");

}

}

Test.java

**package** beans;

**public** **class** Test {

**public** Test(){

System.*out*.println("test .......");

}

**public** **void** hello(){

System.*out*.println("inside the hello method");

System.*out*.println("hello world");

}

}

Spring.xml

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"

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

<!-- copy from spring-beans/factory.xml/spring-beans-2.0.dtd 37 and 38 lines-->

<beans>

<bean id = *"t"* class= *"beans.Test"*>

</bean>

</beans>

Run the Client.java file

Output:

excecution started from main

found the resource

Nov 23, 2017 6:31:30 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions

INFO: Loading XML bean definitions from class path resource [resource/spring.xml]

container object created

test .......

got the bean object

calling hello method

inside the hello method

hello world

execution ended

example 2

Client.java

**package** beans;

**import** java.util.ResourceBundle;

**import** org.springframework.beans.factory.BeanFactory;

**import** org.springframework.beans.factory.xml.~~XmlBeanFactory~~;

**import** org.springframework.core.io.ClassPathResource;

**import** org.springframework.core.io.Resource;

**public** **class** Client {

**public** **static** **void** main(String[] args) {

Resource bundle = **new** ClassPathResource("resource/spring.xml");

BeanFactory beanFactory = **new** XmlBeanFactory(bundle);

Object bean1 = beanFactory.getBean("t");

Object bean2 = beanFactory.getBean("t");

Object bean3 = beanFactory.getBean("t");

System.*out*.println(bean1==bean2);

System.*out*.println(bean2==bean3);

}

}

Test.java

**package** beans;

**public** **class** Test {

**public** Test(){

System.*out*.println("test .......");

}

**public** **void** hello(){

System.*out*.println("inside the hello method");

System.*out*.println("hello world");

}

}

Spring.xml

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"

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

<!-- copy from spring-beans/factory.xml/spring-beans-2.0.dtd 37 and 38 lines-->

<beans>

<bean id = *"t"* class= *"beans.Test"*>

</bean>

</beans>

Output :

Nov 23, 2017 6:32:46 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions

INFO: Loading XML bean definitions from class path resource [resource/spring.xml]

test .......

true

true

core container with scope as singleton

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

Client.java

**package** beans;

**import** java.util.ResourceBundle;

**import** org.springframework.beans.factory.BeanFactory;

**import** org.springframework.beans.factory.xml.~~XmlBeanFactory~~;

**import** org.springframework.core.io.ClassPathResource;

**import** org.springframework.core.io.Resource;

**public** **class** Client {

**public** **static** **void** main(String[] args) {

Resource bundle = **new** ClassPathResource("resource/spring.xml");

BeanFactory beanFactory = **new** XmlBeanFactory(bundle);

beanFactory.getBean("t");

beanFactory.getBean("t");

beanFactory.getBean("t");

}

}

Test.java

**package** beans;

**public** **class** Test {

**public** Test(){

System.*out*.println("test .......");

}

}

spring.xml

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"

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

<!-- copy from spring-beans/factory.xml/spring-beans-2.0.dtd 37 and 38 lines-->

<beans>

<bean id = *"t"* class= *"beans.Test"* scope=*"singleton"* >

</bean>

</beans>

Output

Nov 23, 2017 9:19:46 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions

INFO: Loading XML bean definitions from class path resource [resource/spring.xml]

test .......

core container with scope as prototype

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

Client.java

**package** beans;

**import** java.util.ResourceBundle;

**import** org.springframework.beans.factory.BeanFactory;

**import** org.springframework.beans.factory.xml.~~XmlBeanFactory~~;

**import** org.springframework.core.io.ClassPathResource;

**import** org.springframework.core.io.Resource;

**public** **class** Client {

**public** **static** **void** main(String[] args) {

Resource bundle = **new** ClassPathResource("resource/spring.xml");

BeanFactory beanFactory = **new** XmlBeanFactory(bundle);

beanFactory.getBean("t");

beanFactory.getBean("t");

beanFactory.getBean("t");

}

}

Test.java

**package** beans;

**public** **class** Test {

**public** Test(){

System.*out*.println("test .......");

}

}

Spring.xml

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"

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

<!-- copy from spring-beans/factory.xml/spring-beans-2.0.dtd 37 and 38 lines-->

<beans>

<bean id = *"t"* class= *"beans.Test"* scope=*"prototype"* >

</bean>

</beans>

Output

Nov 23, 2017 9:21:59 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions

INFO: Loading XML bean definitions from class path resource [resource/spring.xml]

test .......

test .......

test .......

j2ee container with scope as singleton

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

Client.java

**package** beans;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public** **class** Client {

**public** **static** **void** main(String[] args) {

ApplicationContext ap = **new** ClassPathXmlApplicationContext("resource/spring.xml");

}

}

Test.java

**package** beans;

**public** **class** Test {

**public** Test(){

System.*out*.println("test .......");

}

}

Spring.xml

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"

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

<!-- copy from spring-beans/factory.xml/spring-beans-2.0.dtd 37 and 38 lines-->

<beans>

<bean id = *"t"* class= *"beans.Test"* scope=*"singleton"* >

</bean>

</beans>

Output

Nov 23, 2017 9:28:01 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh

INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@3b48c8: startup date [Thu Nov 23 21:28:01 IST 2017]; root of context hierarchy

Nov 23, 2017 9:28:01 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions

INFO: Loading XML bean definitions from class path resource [resource/spring.xml]

test .......

Note : onload it will create the object

j2ee container with scope as prototype without object creation

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

Client.java

**package** beans;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public** **class** Client {

**public** **static** **void** main(String[] args) {

ApplicationContext ap = **new** ClassPathXmlApplicationContext("resource/spring.xml");

}

}

Test.java

**package** beans;

**public** **class** Test {

**public** Test(){

System.*out*.println("test .......");

}

}

Spring.xml

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"

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

<!-- copy from spring-beans/factory.xml/spring-beans-2.0.dtd 37 and 38 lines-->

<beans>

<bean id = *"t"* class= *"beans.Test"* scope=*"prototype"* >

</bean>

</beans>

Output

Nov 23, 2017 9:28:01 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh

INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@3b48c8: startup date [Thu Nov 23 21:28:01 IST 2017]; root of context hierarchy

Nov 23, 2017 9:28:01 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions

INFO: Loading XML bean definitions from class path resource [resource/spring.xml]

Note: just it will load the xml file not creates the object if the scope is prototype

It behaves like core container if the scope is prototype

j2ee container with scope as prototype with object creation

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

Client.java

**package** beans;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public** **class** Client {

**public** **static** **void** main(String[] args) {

ApplicationContext ap = **new** ClassPathXmlApplicationContext("resource/spring.xml");

ap.getBean(“t”);

}

}

Test.java

**package** beans;

**public** **class** Test {

**public** Test(){

System.*out*.println("test .......");

}

}

Spring.xml

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"

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

<!-- copy from spring-beans/factory.xml/spring-beans-2.0.dtd 37 and 38 lines-->

<beans>

<bean id = *"t"* class= *"beans.Test"* scope=*"prototype"* >

</bean>

</beans>

Output

Nov 23, 2017 9:28:01 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh

INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@3b48c8: startup date [Thu Nov 23 21:28:01 IST 2017]; root of context hierarchy

Nov 23, 2017 9:28:01 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions

INFO: Loading XML bean definitions from class path resource [resource/spring.xml]

test .......

ioc container can instantiate private constructor also

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

Client.java

**package** beans;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public** **class** Client {

**public** **static** **void** main(String[] args) {

ApplicationContext ap = **new** ClassPathXmlApplicationContext("resource/spring.xml");

ap.getBean(“t”);

}

}

Test.java

**package** beans;

**public** **class** Test {

**private** Test(){

System.*out*.println("test .......");

}

}

Spring.xml

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"

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

<!-- copy from spring-beans/factory.xml/spring-beans-2.0.dtd 37 and 38 lines-->

<beans>

<bean id = *"t"* class= *"beans.Test"* scope=*"prototype"* >

</bean>

</beans>

Output

Nov 23, 2017 9:28:01 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh

INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@3b48c8: startup date [Thu Nov 23 21:28:01 IST 2017]; root of context hierarchy

Nov 23, 2017 9:28:01 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions

INFO: Loading XML bean definitions from class path resource [resource/spring.xml]

test .......

logic to access the private constructor

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

Dependencies

Setter dependency with one attribute

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

Client.java

package beans;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Client {

public static void main(String[] args) {

ApplicationContext ap = new ClassPathXmlApplicationContext("resource/spring.xml");

Test t = (Test)ap.getBean("t");

t.hello();

}

}

Test.java

**package** beans;

**class** Test {

**private** String name ;

**public** **void** setName(String name){

**this**.name = name ;

}

**public** **void** hello(){

System.*out*.println("hello "+name);

}

}

Spring.xml

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"

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

<!-- copy from spring-beans/factory.xml/spring-beans-2.0.dtd 37 and 38 lines-->

<beans>

<bean id = *"t"* class= *"beans.Test"* scope=*"prototype"* >

<property name=*"name"* value=*"Mounesh"*></property>

</bean>

</beans>

Output

Nov 24, 2017 3:40:18 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh

INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@aa6635: startup date [Fri Nov 24 15:40:18 IST 2017]; root of context hierarchy

Nov 24, 2017 3:40:18 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions

INFO: Loading XML bean definitions from class path resource [resource/spring.xml]

hello Mounesh

setter dependency with two attribute

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

Cleint.java

package beans;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Client {

public static void main(String[] args) {

ApplicationContext ap = new ClassPathXmlApplicationContext("resource/spring.xml");

Test t = (Test)ap.getBean("t");

t.hello();

}

}

Test.java

**package** beans;

**class** Test {

**private** String name ;

**private** **int** age;

**public** **void** setName(String name){

**this**.name = name ;

}

**public** **void** setAge(**int** age){

**this**.age = age;

}

**public** **void** hello(){

System.*out*.println("hello "+name+" age is "+age);

}

}

Spring.xml

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"

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

<!-- copy from spring-beans/factory.xml/spring-beans-2.0.dtd 37 and 38 lines-->

<beans>

<bean id = *"t"* class= *"beans.Test"* scope=*"prototype"* >

<property name=*"name"* value=*"Mounesh"*></property>

<property name=*"age"* value=*"23"*></property>

</bean></beans>

Output

Nov 24, 2017 3:51:36 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh

INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@11c2812: startup date [Fri Nov 24 15:51:36 IST 2017]; root of context hierarchy

Nov 24, 2017 3:51:36 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions

INFO: Loading XML bean definitions from class path resource [resource/spring.xml]

hello Mounesh age is 23

through setter dependency we can nit override the setted value if we tries to

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

override we will get duplicate exception

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

Client.java

package beans;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Client {

public static void main(String[] args) {

ApplicationContext ap = new ClassPathXmlApplicationContext("resource/spring.xml");

Test t = (Test)ap.getBean("t");

t.hello();

}

}

Test.java

**package** beans;

**class** Test {

**private** String name ;

**private** **int** age;

**public** **void** setName(String name){

**this**.name = name ;

}

**public** **void** setAge(**int** age){

**this**.age = age;

}

**public** **void** hello(){

System.*out*.println("hello "+name+" age is "+age);

}

}

Spring.xml

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"

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

<!-- copy from spring-beans/factory.xml/spring-beans-2.0.dtd 37 and 38 lines-->

<beans>

<bean id = *"t"* class= *"beans.Test"* scope=*"prototype"* >

<property name=*"name"* value=*"Mounesh"*></property>

<property name=*"age"* value=*"23"*></property>

<property name=*"name"* value=*"mahesh"*></property>

</bean>

</beans>

Output

Exception in thread "main" org.springframework.beans.factory.parsing.BeanDefinitionParsingException

Using setter dependency we can set only one arg setter method not possible to set the multiple argument setter method

Client.java

package beans;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Client {

public static void main(String[] args) {

ApplicationContext ap = new ClassPathXmlApplicationContext("resource/spring.xml");

Test t = (Test)ap.getBean("t");

t.hello();

}

}

Test.java

**package** beans;

**class** Test {

**private** String name ;

**private** **int** age;

**public** **void** setName(String name,**int** age){

**this**.name = name ;

**this**.age = age;

}

**public** **void** hello(){

System.*out*.println("hello "+name+" age is "+age);

}

}

Spring.xml

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"

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

<!-- copy from spring-beans/factory.xml/spring-beans-2.0.dtd 37 and 38 lines-->

<beans>

<bean id = *"t"* class= *"beans.Test"* scope=*"prototype"* >

<property name=*"name"* value=*"Mounesh"*></property>

<property name=*"age"* value=*"23"*></property>

</bean>

</beans>

Output :

Exception in main thread

Constructor dependency

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

Client.java

**package** beans;

**import** org.springframework.beans.factory.BeanFactory;

**import** org.springframework.beans.factory.xml.~~XmlBeanFactory~~;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**import** org.springframework.core.io.ClassPathResource;

**import** org.springframework.core.io.Resource;

**public** **class** Client {

**public** **static** **void** main(String[] args) {

Resource bundle = **new** ClassPathResource("resource/spring.xml");

BeanFactory beanFactory = **new** XmlBeanFactory(bundle);

Test t = (Test)beanFactory.getBean("t");

t.hello();

}

}

Test.java

**package** beans;

**class** Test {

**private** String name,Email;

**private** **int** age;

Test(String name,**int** age,String email){

**this**.name = name ;

**this**.age = age ;

**this**.Email = Email ;

}

**public** **void** hello(){

System.*out*.println("name is "+name);

System.*out*.println("age is "+age);

System.*out*.println("Email is "+Email);

}

}

Spring.xml

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"

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

<!-- copy from spring-beans/factory.xml/spring-beans-2.0.dtd 37 and 38 lines-->

<beans>

<bean id = *"t"* class= *"beans.Test"*>

<constructor-arg value = *"Mounesh"* index=*"0"*/>

<constructor-arg value = *"Mounesh@gmail.com"* index=*"2"*/>

<constructor-arg value = *"23"* index = *"1"*/>

</bean>

</beans>

Output

Nov 24, 2017 5:43:14 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions

INFO: Loading XML bean definitions from class path resource [resource/spring.xml]

name is Mounesh

age is 23

Email is null

Secondary type dependencies with two xml file/passing by reference

Cleint.java

-----------

**package** beans;

**import** org.springframework.beans.factory.BeanFactory;

**import** org.springframework.beans.factory.xml.~~XmlBeanFactory~~;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**import** org.springframework.core.io.ClassPathResource;

**import** org.springframework.core.io.Resource;

**public** **class** Client {

**public** **static** **void** main(String[] args) {

String[] files = **new** String[]{"resource/car.xml","resource/engine.xml"};

ApplicationContext ap = **new** ClassPathXmlApplicationContext(files);

Car t = (Car)ap.getBean("c");

t.carInfo();

}

}

Car.java

**package** beans;

**public** **class** Car {

**private** String carName;

**private** Engine engine ;

**public** **void** setCarName(String carName) {

**this**.carName = carName;

}

**public** **void** setEngine(Engine engine) {

**this**.engine = engine;

}

**public** **void** carInfo()

{

System.*out*.println("carname is "+**this**.carName);

System.*out*.println("model name is "+engine.getModelYear());

}

}

Engine.java

**package** beans;

**public** **class** Engine {

**private** **int** modelYear;

**public** **void** setModelYear(**int** modelYear) {

**this**.modelYear = modelYear;

}

**public** **int** getModelYear() {

**return** modelYear;

}

}

Car.xml

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"

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

<!-- copy from spring-beans/factory.xml/spring-beans-2.0.dtd 37 and 38 lines-->

<beans>

<bean id = *"c"* class= *"beans.Car"*>

<property name=*"carName"* value =*"AUDI"*></property>

<property name=*"engine"* ref=*"e"*></property>

</bean>

</beans>

Engine.xml

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"

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

<!-- copy from spring-beans/factory.xml/spring-beans-2.0.dtd 37 and 38 lines-->

<beans>

<bean id = *"e"* class= *"beans.Engine"*>

<property name=*"modelYear"* value=*"2015"*></property>

</bean>

</beans>

Output

Nov 24, 2017 6:42:43 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh

INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@11c2812: startup date [Fri Nov 24 18:42:43 IST 2017]; root of context hierarchy

Nov 24, 2017 6:42:43 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions

INFO: Loading XML bean definitions from class path resource [resource/car.xml]

Nov 24, 2017 6:42:43 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions

INFO: Loading XML bean definitions from class path resource [resource/engine.xml]

carname is AUDI

model name is 2015

secondary dependencies with one xml file / inner beans/pass by value

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

Client.java

**package** beans;

**import** org.springframework.beans.factory.BeanFactory;

**import** org.springframework.beans.factory.xml.~~XmlBeanFactory~~;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**import** org.springframework.core.io.ClassPathResource;

**import** org.springframework.core.io.Resource;

**public** **class** Client {

**public** **static** **void** main(String[] args) {

ApplicationContext ap = **new** ClassPathXmlApplicationContext("resource/car.xml");

Car t = (Car)ap.getBean("c");

t.carInfo();

}

}

Car.java

**package** beans;

**public** **class** Car {

**private** String carName;

**private** Engine engine ;

**public** **void** setCarName(String carName) {

**this**.carName = carName;

}

**public** **void** setEngine(Engine engine) {

**this**.engine = engine;

}

**public** **void** carInfo()

{

System.*out*.println("carname is "+**this**.carName);

System.*out*.println("model name is "+engine.getModelYear());

}

}

Engine.java

**package** beans;

**public** **class** Engine {

**private** **int** modelYear;

**public** **void** setModelYear(**int** modelYear) {

**this**.modelYear = modelYear;

}

**public** **int** getModelYear() {

**return** modelYear;

}

}

Car.xml

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"

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

<!-- copy from spring-beans/factory.xml/spring-beans-2.0.dtd 37 and 38 lines-->

<beans>

<bean id = *"c"* class= *"beans.Car"*>

<property name=*"carName"* value =*"AUDI"*></property>

<property name=*"engine"* >

<bean id = *"e"* class=*"beans.Engine"*>

<property name=*"modelYear"* value =*"2011"*></property>

</bean>

</property>

</bean>

</beans>

Array dependency injection primitive and secondary types

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

Client.java

package beans;

import org.springframework.beans.factory.BeanFactory;

import org.springframework.beans.factory.xml.XmlBeanFactory;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import org.springframework.core.io.ClassPathResource;

import org.springframework.core.io.Resource;

public class Client {

public static void main(String[] args) {

ApplicationContext ap = new ClassPathXmlApplicationContext("resource/car.xml");

Car t = (Car)ap.getBean("c");

t.carInfo();

}

}

Car.java

**package** beans;

**public** **class** Car {

**private** String[] carName;

**private** Engine[] engine ;

**public** **void** setCarName(String[] carName) {

**this**.carName = carName;

}

**public** **void** setEngine(Engine[] engine) {

**this**.engine = engine;

}

**public** **void** carInfo()

{

**for**(String car:carName){

System.*out*.println("car name is "+car);

}

**for**(Engine eng:engine){

System.*out*.println("car model is "+eng.getModelYear());

}

}

}

Engine.java

**package** beans;

**public** **class** Engine {

**private** **int** modelYear;

**public** **void** setModelYear(**int** modelYear) {

**this**.modelYear = modelYear;

}

**public** **int** getModelYear() {

**return** modelYear;

}

}

Car.xml

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"

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

<!-- copy from spring-beans/factory.xml/spring-beans-2.0.dtd 37 and 38 lines-->

<beans>

<bean id = *"e1"* class=*"beans.Engine"*>

<property name=*"modelYear"* value =*"2011"*></property>

</bean>

<bean id = *"e2"* class=*"beans.Engine"*>

<property name=*"modelYear"* value =*"2012"*></property>

</bean>

<bean id = *"e3"* class=*"beans.Engine"*>

<property name=*"modelYear"* value =*"2013"*></property>

</bean>

<bean id = *"c"* class= *"beans.Car"*>

<property name=*"carName"* >

<list>

<value>Audi</value>

<value>Honda</value>

<value>Hyndai</value>

</list>

</property>

<property name=*"engine"* >

<list>

<ref bean=*"e1"*/>

<ref bean=*"e2"*/>

<ref bean=*"e3"*/>

</list>

</property>

</bean>

</beans>

Default collection dependency injection

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

Client.java

package beans;

import org.springframework.beans.factory.BeanFactory;

import org.springframework.beans.factory.xml.XmlBeanFactory;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import org.springframework.core.io.ClassPathResource;

import org.springframework.core.io.Resource;

public class Client {

public static void main(String[] args) {

ApplicationContext ap = new ClassPathXmlApplicationContext("resource/spring.xml");

Test tt = (Test)ap.getBean("t");

tt.printData();

}

}

Test.java

**package** beans;

**import** java.util.List;

**import** java.util.Set;

**import** java.util.Map;;

**class** Test {

**private** List fruits;

**private** Set cricketers;

**private** Map country\_map;

**public** **void** setFruits(List fruits) {

**this**.fruits = fruits;

}

**public** **void** setCricketers(Set cricketers) {

**this**.cricketers = cricketers;

}

**public** **void** setCountry\_map(Map country\_map) {

**this**.country\_map = country\_map;

}

**public** **void** printData(){

System.*out*.println("fruits.........");

**for**(Object frui : fruits)

{

System.*out*.println(frui);

}

System.*out*.println("cricketers.........");

**for**(Object cri : cricketers)

{

System.*out*.println(cri);

}

System.*out*.println("country cap.........");

Set keySet = country\_map.keySet();

**for**(Object key: keySet)

{

System.*out*.println("country "+key + " capital is "+country\_map.get(key));

}

}

}

Spring.xml

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"

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

<beans>

<bean id=*"t"* class=*"beans.Test"*>

<property name=*"fruits"*>

<list>

<value>apple</value>

<value>mango</value>

<value>orange</value>

</list>

</property>

<property name=*"cricketers"*>

<set>

<value>sachin</value>

<value>sehwag</value>

<value>sachin</value>

</set>

</property>

<property name=*"country\_map"*>

<map>

<entry key=*"India"* value=*"delhi"*></entry>

<entry key=*"Pakistan"* value=*"islambad"*></entry>

</map>

</property>

</bean>

</beans>

Output

Nov 25, 2017 11:17:23 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh

INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@18ab963: startup date [Sat Nov 25 23:17:23 IST 2017]; root of context hierarchy

Nov 25, 2017 11:17:23 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions

INFO: Loading XML bean definitions from class path resource [resource/spring.xml]

fruits.........

apple

mango

orange

cricketers.........

sachin

sehwag

country cap.........

country India capital is delhi

country Pakistan capital is islambad

Particular type collection dependency injection

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

Client.java

package beans;

import org.springframework.beans.factory.BeanFactory;

import org.springframework.beans.factory.xml.XmlBeanFactory;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import org.springframework.core.io.ClassPathResource;

import org.springframework.core.io.Resource;

public class Client {

public static void main(String[] args) {

ApplicationContext ap = new ClassPathXmlApplicationContext("resource/spring.xml");

Test tt = (Test)ap.getBean("t");

tt.printData();

}

}

Test.java

**package** beans;

**import** java.util.Hashtable;

**import** java.util.Set;

**import** java.util.TreeSet;

**import** java.util.Vector;

**class** Test {

**private** Vector fruits;

**private** TreeSet cricketers;

**private** Hashtable country\_map;

**public** **void** setFruits(Vector fruits) {

**this**.fruits = fruits;

}

**public** **void** setCricketers(TreeSet cricketers) {

**this**.cricketers = cricketers;

}

**public** **void** setCountry\_map(Hashtable country\_map) {

**this**.country\_map = country\_map;

}

**public** **void** printData(){

System.*out*.println("fruits.........");

**for**(Object frui : fruits)

{

System.*out*.println(frui);

}

System.*out*.println("cricketers.........");

**for**(Object cri : cricketers)

{

System.*out*.println(cri);

}

System.*out*.println("country cap.........");

Set keySet = country\_map.keySet();

**for**(Object key: keySet)

{

System.*out*.println("country "+key + " capital is "+country\_map.get(key));

}

}

}

Spring.xml

<?xml version=*"1.0"* encoding=*"UTF-8"* standalone=*"no"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:util=*"http://www.springframework.org/schema/util"*

xsi:schemaLocation=*"*

*http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.1.xsd*

*http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-3.1.xsd"*>

<bean id=*"t"* class=*"beans.Test"*>

<property name=*"fruits"*>

<util:list list-class=*"java.util.Vector"*>

<value>apple</value>

<value>mango</value>

<value>orange</value>

</util:list>

</property>

<property name=*"cricketers"*>

<util:set set-class=*"java.util.TreeSet"*>

<value>sachin</value>

<value>sehwag</value>

<value>sachin</value>

</util:set>

</property>

<property name=*"country\_map"*>

<util:map map-class=*"java.util.Hashtable"*>

<entry key=*"India"* value=*"delhi"*></entry>

<entry key=*"Pakistan"* value=*"islambad"*></entry>

</util:map>

</property>

</bean></beans>

Output :

Nov 25, 2017 11:11:42 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh

INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@18ab963: startup date [Sat Nov 25 23:11:42 IST 2017]; root of context hierarchy

Nov 25, 2017 11:11:43 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions

INFO: Loading XML bean definitions from class path resource [resource/spring.xml]

fruits.........

apple

mango

orange

cricketers.........

sachin

sehwag

country cap.........

country India capital is delhi

country Pakistan capital is islambad

injecting properties type from the particular location

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

Client.java

package beans;

import org.springframework.beans.factory.BeanFactory;

import org.springframework.beans.factory.xml.XmlBeanFactory;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import org.springframework.core.io.ClassPathResource;

import org.springframework.core.io.Resource;

public class Client {

public static void main(String[] args) {

ApplicationContext ap = new ClassPathXmlApplicationContext("resource/spring.xml");

Test tt = (Test)ap.getBean("t");

tt.printData();

}

}

Test.java

**package** beans;

**import** java.util.Properties;

**import** java.util.Set;

**class** Test {

**private** Properties driver;

**public** **void** setDriver(Properties driver) {

**this**.driver = driver;

}

**public** **void** printData() {

System.*out*.println("driver infomation......");

Set keySet = driver.keySet();

**for**(Object obj : keySet){

System.*out*.println(obj +" ---- "+driver.getProperty(obj.toString()));

}

}

}

Spring.xml

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:util=*"http://www.springframework.org/schema/util"*

xsi:schemaLocation=*"*

*http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.1.xsd*

*http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-3.1.xsd"*>

<bean id=*"t"* class=*"beans.Test"*>

<property name=*"driver"*>

<util:properties location=*"classpath:resource/driver.properties"*></util:properties>

</property>

</bean>

</beans>

driver.properties @ location resource/driver.properties

driver=odbc.jdbc.oracledriver

url=oracle url

username=mounesh

password=manager

Output

Nov 26, 2017 4:37:47 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh

INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@11c2812: startup date [Sun Nov 26 16:37:47 IST 2017]; root of context hierarchy

Nov 26, 2017 4:37:47 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions

INFO: Loading XML bean definitions from class path resource [resource/spring.xml]

Nov 26, 2017 4:37:48 PM org.springframework.core.io.support.PropertiesLoaderSupport loadProperties

INFO: Loading properties file from class path resource [resource/driver.properties]

driver infomation......

password ---- manager

url ---- oracle url

driver ---- odbc.jdbc.oracledriver

username ---- mounesh

Required Annotation

=======================

Client.java

**package** beans;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public** **class** Client {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

ApplicationContext ap = **new** ClassPathXmlApplicationContext("resources/spring.xml");

Test t = (Test)ap.getBean("t");

t.printconnection();

}

}

Test.java

**package** beans;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** org.springframework.beans.factory.annotation.Required;

**public** **class** Test {

**private** String name,rollno,college,branch;

@Required

**public** **void** setName(String name) {

**this**.name = name;

}

@Required

**public** **void** setRollno(String rollno) {

**this**.rollno = rollno;

}

**public** **void** setCollege(String college) {

**this**.college = college;

}

**public** **void** setBranch(String branch) {

**this**.branch = branch;

}

**public** **void** printconnection()

{

System.***out***.println("name === "+name);

System.***out***.println("roll no ==="+rollno);

System.***out***.println("college ==="+college);

System.***out***.println("branch ==="+branch);

}}

spring.xml

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"

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

<!-- copy from spring-beans/factory.xml/spring-beans-2.0.dtd 37 and 38 lines-->

<beans>

<bean class=*"org.springframework.beans.factory.annotation.RequiredAnnotationBeanPostProcessor"*></bean>

<bean id = *"t"* class= *"beans.Test"*>

<property name=*"name"* value=*"Mounesh"*></property>

<property name=*"rollno"* value=*"11GAEI5028"*></property>

</bean>

</beans>

Output

Dec 02, 2017 12:25:17 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh

INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@3fa77460: startup date [Sat Dec 02 12:25:17 IST 2017]; root of context hierarchy

Dec 02, 2017 12:25:18 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions

INFO: Loading XML bean definitions from class path resource [resources/spring.xml]

name === Mounesh

roll no ===11GAEI5028

college ===null

branch ===null

**Autowiring**

Main aim is auto dependency injection

Auto wiring enables the spring framework to inject object dependency implicitly

Autowiring can not used to inject the primitive and string values

Limitation are : only we can inject secondary types (not applicable for primitive types)

Suppose we have 100 pages that means 100 forms … and for each 2 or 1 form we need to write the controller atleast we will have 50+ controller , under model we need to have 100 buiness class and we need to have 100 dao one dao for one table

Here controller need to have dependency on business and business dependent on dao

This dependency we need to write 100 times but using autowiring automatically it will inject into D1(dao) inject into B1(buiness) and B1 will injkect into C1(controller)

Bytype -> setter

By name -> setter

Auto detect - > setter/constructor

By constructor -> constructor

**Bytype :**

searches only Type in the xml if finds (matches the type) do the dependency injection

If multiple bean id with same type then ambuity

Ambiguity in autowiring solve by autowire -candidate

autowire -candidate = false then that particular bean will not participate in the autowiring

**Engine.Java**

**public** **class** Engine {

**private** **int** modelYear;

**public** **void** setModelYear(**int** modelYear) {

**this**.modelYear = modelYear;

}

**public** **int** getModelYear() {

**return** modelYear;

}

}

**Car.java**

**public** **class** Car {

**private** Engine engine;

**public** **void** setEngine(Engine engine) {

**this**.engine = engine;

}

**public** Engine getEngine() {

**return** engine;

}

**public** **void** printData(){

System.***out***.println("model of the car is "+engine.getModelYear());

}

}

**Client.Java**

**import org.springframework.context.ApplicationContext;**

**import org.springframework.context.support.ClassPathXmlApplicationContext;**

**public class Client {**

**public static void main(String[] args) {**

**ApplicationContext ap = new ClassPathXmlApplicationContext("resource/spring.xml");**

**Object bean = ap.getBean("c");**

**Car car = (Car)bean;**

**car.printData();**

**}**

**}**

**Spring.xml**

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"

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

<beans>

<bean id=*"e1"* class=*"Engine"*>

<property name=*"modelYear"* value=*"2018"*></property></bean>

<!--

to avoid the ambiguity problem we will use the autowire-candidate="false"

<bean id="e2" class="Engine" autowire-candidate="false">

<property name="modelYear" value="2018"></property></bean> -->

<bean id=*"c"* class=*"Car"* autowire=*"byType"*>

</bean>

</beans>

**Output :**

Sep 02, 2018 9:38:44 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh

INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@4c39a2f1: startup date [Sun Sep 02 21:38:44 IST 2018]; root of context hierarchy

Sep 02, 2018 9:38:45 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions

INFO: Loading XML bean definitions from class path resource [resource/spring.xml]

model of the car is 2018

**ByName :**

Search both name and type if finds(matches the type and name )then does the dependency injection otherwise it will not inject

**Engine.java**

**package** beans;

**public** **class** Engine {

**private** **int** modelYear;

**public** **void** setModelYear(**int** modelYear) {

**this**.modelYear = modelYear;

}

**public** **int** getModelYear() {

**return** modelYear;

}

}

**Car.java**

**package** beans;

**public** **class** Car {

**private** Engine engine;

**public** **void** setEngine(Engine engine) {

**this**.engine = engine;

}

**public** Engine getEngine() {

**return** engine;

}

**public** **void** printData(){

System.***out***.println("model of the car is "+engine.getModelYear());

}

}

**Client.java**

package beans;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Client {

public static void main(String[] args) {

ApplicationContext ap = new ClassPathXmlApplicationContext("resource/spring.xml");

Object bean = ap.getBean("c");

Car car = (Car) bean;

car.printData();

}

}

**Spring.xml**

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"

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

<beans>

<bean id=*"engine"* class=*"beans.Engine"*>

<property name=*"modelYear"* value=*"2018"*></property></bean>

<!--

to avoid the ambiguity problem we will use the autowire-candidate="false" -->

<bean id=*"e2"* class=*"beans.Engine"* >

<property name=*"modelYear"* value=*"2019"*></property></bean>

<bean id=*"c"* class=*"beans.Car"* autowire=*"byName"*>

</bean>

</beans>

**Output**

Sep 02, 2018 9:41:32 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh

INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@e2585745: startup date [Sun Sep 02 21:41:32 IST 2018]; root of context hierarchy

Sep 02, 2018 9:41:32 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions

INFO: Loading XML bean definitions from class path resource [resource/spring.xml]

model of the car is 2018

**Byconstructor** :

if we have constructor dependency then we should for by constructor

keyword is constructor

internally it will uses byType - searches only the type in the xml xml if finds (matches the type) do the dependency injection \

If multiple bean id with same type then ambuity

Ambiguity in autowiring solve by autowire -candidate

autowire -candidate = false then that particular bean will not participate in the autowiring

**Engine.java**

**package** beans;

**import** java.util.Set;

**public** **class** Engine {

**private** **int** modelYear;

**public** **void** setModelYear(**int** modelYear) {

**this**.modelYear = modelYear;

}

**public** **int** getModelYear() {

**return** modelYear;

}

}

**Car.java**

**package** beans;

**public** **class** Car {

**private** Engine engine;

**public** Car(Engine engine) {

**this**.engine = engine;

}

**public** **void** printData(){

System.***out***.println("modelyear of the engine is "+engine.getModelYear());

}

}

**Client.java**

package beans;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Client {

public static void main(String[] args) {

ApplicationContext ap = new ClassPathXmlApplicationContext("resource/spring.xml");

Car car = (Car)ap.getBean("c");

car.printData();

}

}

**Spring.xml**

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"

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

<beans>

<!-- autowiring by constructor will internally uses byType -->

<bean id=*"engine"* class=*"beans.Engine"* autowire-candidate=*"false"*>

<property name=*"modelYear"* value=*"2015"*></property></bean>

<!--

to avoid the ambiguity problem we will use the autowire-candidate="false" -->

<bean id=*"e2"* class=*"beans.Engine"*>

<property name=*"modelYear"* value=*"2016"*></property></bean>

<bean id=*"c"* class=*"beans.Car"* autowire=*"constructor"*>

</bean>

</beans>

**Output**

Sep 02, 2018 9:44:35 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh

INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@4153bb3c: startup date [Sun Sep 02 21:44:35 IST 2018]; root of context hierarchy

Sep 02, 2018 9:44:35 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions

INFO: Loading XML bean definitions from class path resource [resource/spring.xml]

modelyear of the engine is 2016

**autodetect** :

used for both the constructor and setter dependency injection

if we not sure of the dependency injection will be setter or constructor then we should go for autodetect

internally will use byType searching

searches only the type in the xml xml if finds (matches the type) do the dependency injection \

If multiple bean id with same type then ambuity

Ambiguity in autowiring solve by autowire -candidate

autowire -candidate = false then that particular bean will not participate in the autowiring

**special case** : suppose if class has both the setter and constructor injection then

if a class has parameterized constructor and setter injection then it will do the injection using constructor

**Engine.java**

**package** beans;

**public** **class** Engine {

**private** **int** modelYear;

**public** **void** setModelYear(**int** modelYear) {

**this**.modelYear = modelYear;

}

**public** **int** getModelYear() {

**return** modelYear;

}

}

**Car.java**

**package** beans;

**public** **class** Car {

**private** Engine engine;

**public** Car(Engine engine){

System.***out***.println("construction selected by the autodetect");

**this**.engine = engine;

}

**public** **void** setEngine(Engine engine) {

System.***out***.println("setter selected by the autodetect");

**this**.engine = engine;

}

**public** **void** printdata(){

System.***out***.println("modelYear of the engine is"+engine.getModelYear());

}

}

**Client.java**

package beans;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Client {

public static void main(String[] args) {

ApplicationContext ap = new ClassPathXmlApplicationContext("resource/spring.xml");

Car car = (Car)ap.getBean("c");

car.printdata();

}

}

**Spring.xml**

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"

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

<beans>

<bean id=*"engine"* class=*"beans.Engine"* >

<property name=*"modelYear"* value=*"2015"*></property></bean>

<bean id=*"c"* class=*"beans.Car"* autowire=*"autodetect"* >

</bean>

</beans>

**Output**

Sep 02, 2018 9:48:12 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh

INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@281919f0: startup date [Sun Sep 02 21:48:12 IST 2018]; root of context hierarchy

Sep 02, 2018 9:48:12 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions

INFO: Loading XML bean definitions from class path resource [resource/spring.xml]

construction selected by the autodetect

modelYear of the engine is2015

if class has default constructor, parameterized constructor and setter injection then it will do the injection using setter method

**Engine.java**

**package** beans;

**public** **class** Engine {

**private** **int** modelYear;

**public** **void** setModelYear(**int** modelYear) {

**this**.modelYear = modelYear;

}

**public** **int** getModelYear() {

**return** modelYear;

}

}

**Car.java**

**package** beans;

**public** **class** Car {

**private** Engine engine;

**public** Car(){

System.***out***.println("default constructor");

}

**public** Car(Engine engine){

System.***out***.println("construction selected by the autodetect");

**this**.engine = engine;

}

**public** **void** setEngine(Engine engine) {

System.***out***.println("setter selected by the autodetect");

**this**.engine = engine;

}

**public** **void** printdata(){

System.***out***.println("modelYear of the engine is"+engine.getModelYear());

}

}

**Client.java**

package beans;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Client {

public static void main(String[] args) {

ApplicationContext ap = new ClassPathXmlApplicationContext("resource/spring.xml");

Car car = (Car)ap.getBean("c");

car.printdata();

}

}

**Spring.xml**

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"

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

<beans>

<bean id=*"engine"* class=*"beans.Engine"* >

<property name=*"modelYear"* value=*"2015"*></property></bean>

<bean id=*"c"* class=*"beans.Car"* autowire=*"autodetect"* >

</bean>

</beans>

**Output :**

Sep 02, 2018 9:49:57 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh

INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@72f733b6: startup date [Sun Sep 02 21:49:57 IST 2018]; root of context hierarchy

Sep 02, 2018 9:49:57 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions

INFO: Loading XML bean definitions from class path resource [resource/spring.xml]

default constructor

setter selected by the autodetect

modelYear of the engine is2015

**Note :** Best approach is among these is “bytype”

**Annotation Autowired**

Autowired is the property level annotation it will used in the property

By default it will use byType for injecting, if we are using the byType then there will be chance of ambiguity problem to resolve that we will use **qualifier annotation**

To activate these to annotation we need to instantiate **AutowireAnnotaionBeanPostProcessor** in the xml file

No need to write the setter or constructor in @Autowired annotation

**Example**

**Engine.java**

**package** beans;

**public** **class** Engine {

**private** **int** modelYear;

**public** **void** setModelYear(**int** modelYear) {

**this**.modelYear = modelYear;

}

**public** **int** getModelYear() {

**return** modelYear;

}

}

**Car.java**

**package** beans;

**import** org.springframework.beans.factory.annotation.Autowired;

**public** **class** Car {

@Autowired

**private** Engine engine;

**public** **void** printData(){

System.***out***.println("model year is "+engine.getModelYear());

}

}

**Client.java**

**package** beans;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public** **class** Client {

**public** **static** **void** main(String[] args) {

ApplicationContext ap = **new** ClassPathXmlApplicationContext("resource/spring.xml");

Car car = (Car)ap.getBean("c");

car.printData();

}

}

**Spring.xml**

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"

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

<beans>

<bean class=*"org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor"*/>

<bean id=*"engine"* class=*"beans.Engine"* >

<property name=*"modelYear"* value=*"2015"*>

</property>

</bean>

<bean id=*"c"* class=*"beans.Car"*>

</bean>

</beans>

Output :

Sep 02, 2018 11:47:32 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh

INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@1d2a9de8: startup date [Sun Sep 02 23:47:32 IST 2018]; root of context hierarchy

Sep 02, 2018 11:47:33 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions

INFO: Loading XML bean definitions from class path resource [resource/spring.xml]

model year is 2015

**@qualifier annotation**

**Engine.java**

**package** beans;

**public** **class** Engine {

**private** **int** modelYear;

**public** **void** setModelYear(**int** modelYear) {

**this**.modelYear = modelYear;

}

**public** **int** getModelYear() {

**return** modelYear;

}

}

**Car.java**

**package** beans;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.beans.factory.annotation.Qualifier;

**public** **class** Car {

@Autowired

@Qualifier("e2")

**private** Engine engine;

**public** **void** printData(){

System.***out***.println("model year is "+engine.getModelYear());

}

}

**Client.java**

**package** beans;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public** **class** Client {

**public** **static** **void** main(String[] args) {

ApplicationContext ap = **new** ClassPathXmlApplicationContext("resource/spring.xml");

Car car = (Car)ap.getBean("c");

car.printData();

}

}

**Spring.xml**

<beans xmlns = *"http://www.springframework.org/schema/beans"*

xmlns:xsi = *"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:context = *"http://www.springframework.org/schema/context"*

xsi:schemaLocation = *"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-3.0.xsd*

*http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context-3.0.xsd"*>

<!-- <bean class="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor"/> -->

<context:annotation-config/>

<bean id=*"e1"* class=*"beans.Engine"* >

<property name=*"modelYear"* value=*"2015"*>

</property>

</bean>

<bean id=*"e2"* class=*"beans.Engine"* >

<property name=*"modelYear"* value=*"2018"*>

</property>

</bean>

<bean id=*"c"* class=*"beans.Car"*>

</bean>

</beans>

**Output**

Sep 02, 2018 11:54:32 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh

INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@25458d33: startup date [Sun Sep 02 23:54:32 IST 2018]; root of context hierarchy

Sep 02, 2018 11:54:33 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions

INFO: Loading XML bean definitions from class path resource [resource/spring.xml]

model year is 2018

Autowiring using xml

We can reduce manual dependency in the xml file

Note:<Context: annotation-config/> this will activate all the annotaions

**Stero type annotations**

@Controller – used for the controller classes

@Repository – used for the DAO classes

@Component – non MVC classes used

@Service – for the business classes

If a class contains @Component then it will create the object automatically

**Static variable injecting in spring**

Static injection we will achieve using the “MethodInvokingFactoryBean”

**Car.java**

**package** beans;

**public** **class** Car {

**private** **static** String *carName*;

**public** **static** **void** setCarName(String carName) {

Car.*carName* = carName;

}

**public** **void** prinCar(){

System.***out***.println("car name is "+Car.*carName*);

}

}

**Client.java**

**package** beans;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public** **class** Client {

**public** **static** **void** main(String[] args) {

ApplicationContext ap = **new** ClassPathXmlApplicationContext("resource/spring.xml");

Car car = (Car) ap.getBean("c");

car.prinCar();

}

}

**Spring.xml**

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"

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

<beans>

<bean id=*"c"* class=*"beans.Car"* />

<bean class=*"org.springframework.beans.factory.config.MethodInvokingFactoryBean"*>

<property name=*"staticMethod"* value=*"beans.Car.setCarName"*></property>

<property name=*"arguments"*>

<list>

<value>audi</value>

</list>

</property>

</bean>

</beans>

Output

Sep 04, 2018 8:00:07 AM org.springframework.context.support.AbstractApplicationContext prepareRefresh

INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@5dbf0c97: startup date [Tue Sep 04 08:00:07 IST 2018]; root of context hierarchy

Sep 04, 2018 8:00:07 AM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions

INFO: Loading XML bean definitions from class path resource [resource/spring.xml]

car name is audi

**Bean Life Cycles**

===============

For bean object we can implement life cycles in three approaches

**Programmatic** = need to implement from intialisingBean and disposableBean

intialisingBean for intialiaze purpose using afterPropertySet() method

disposableBean for destroy purpose using destroy() method

first it will inject the values and then init() method and at last destroy method

**Declarative** = without implementing the bean classes we can achieve the life cycle method to the bean class in xml

<bean id=”myId” class=”beans.Test” init-method=”methodName” destroy-method=”methodname”/>

Annotation = we need to used @PostConstruct in the init() method and @preConstruct in the destroy() method.

In order to activate the these two annotations we need to instatiate commonAnnotationBeanPostProcessor

**Note** for more than one class we need to configure same init and destroy method that time we need to add default-init-method=” ” and default-destroy-method in the <beans> tag (globally)

**Look up method DI**

**Method** which has no implementation is called loop up methods.

In the concrete class if we don’t want the implementation then that method is the loop up method

Using the spring we can provide the implementation to the method

Using the spring we can replace the method implementation at runtime using method-Replacer tag

**Example:**

Bank.java

**package** beans;

**public** **class** Bank {

**public** **void** withdraw(){

System.***out***.println(":withdraw method:");

}

**public** **void** calcInterest(){

System.***out***.println("calc interest method");

}

**public** **void** checkBal(){

System.***out***.println("check bal method");

}

}

Client.java

**package** beans;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public** **class** Client {

**public** **static** **void** main(String[] args) {

ApplicationContext ap = **new** ClassPathXmlApplicationContext("resource/spring.xml");

Bank b = (Bank)ap.getBean("bank");

b.calcInterest();

b.checkBal();

b.withdraw();

}

}

NewCalcInterest.java

**package** beans;

**import** java.lang.reflect.Method;

**import** org.springframework.beans.factory.support.MethodReplacer;

**public** **class** NewCalcInterest **implements** MethodReplacer{

@Override

**public** Object reimplement(Object o, Method m, Object[] param)

**throws** Throwable {

System.***out***.println("new calc interest method impl ");

**return** o;

}

}

Spring.xml

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"

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

<beans>

<bean id=*"bank"* class=*"beans.Bank"*>

<replaced-method name=*"calcInterest"* replacer=*"nci"*></replaced-method>

</bean>

<bean id=*"nci"* class=*"beans.NewCalcInterest"*></bean>

</beans>

Output :

Sep 09, 2018 11:56:46 AM org.springframework.context.support.AbstractApplicationContext prepareRefresh

INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@ad3b47c6: startup date [Sun Sep 09 11:56:46 IST 2018]; root of context hierarchy

Sep 09, 2018 11:56:46 AM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions

INFO: Loading XML bean definitions from class path resource [resource/spring.xml]

new calc interest method impl

check bal method

:withdraw method:

Note : When we load the bean it will create the Bank proxy object then it will return the bankproxy object.then on that object it will call the methods.

**@resource and @inject**

**-------------------------------**

@resource and @inject are similar to @autowired but

@autowire is provided by the spring

@resource and @ inject is provided by the JDK

@resource uses by name checking and @inject uses by type checking to resolve the ambiguity we will use @qualifier

Whenever we are using the @ resource and @ inject we need to activate the annotations By <context:annotation-config/>

**Property file DI using the expression**

To load the property into the IOC

PropertyPlaceHolderConfigure

Method - setLocation(string str)

From the property we can pass only string types secondary types we can not pass.

CP.java

**package** beans;

**public** **class** CP {

**private** String username;

**private** String password;

**public** **void** setUsername(String username) {

**this**.username = username;

}

**public** String getUsername() {

**return** username;

}

**public** **void** setPassword(String password) {

**this**.password = password;

}

**public** String getPassword() {

**return** password;

}

**public** **void** printCredential(){

System.***out***.println("username is "+username);

System.***out***.println("password is "+password);

}

}

Client.java

**package** beans;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public** **class** Cleint {

**public** **static** **void** main(String[] args) {

ApplicationContext ap = **new** ClassPathXmlApplicationContext("resource/spring.xml");

CP cp = (CP)ap.getBean("cp");

cp.printCredential();

}

}

details.properties

username = mounesh;

password = password;

spring.xml

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"

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

<beans>

<!-- load the properties into the IOC context -->

<bean class=*"org.springframework.beans.factory.config.PropertyPlaceholderConfigurer"*>

<property name=*"location"* value=*"resource/details.properties"*></property>

</bean>

<!-- inject the context scope in to the required class -->

<bean id=*"cp"* class=*"beans.CP"*>

<property name=*"username"* value=*"${username}"*></property>

<property name=*"password"* value=*"${password}"*></property>

</bean>

</beans>

Output

Sep 09, 2018 5:16:54 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh

INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@6464f885: startup date [Sun Sep 09 17:16:54 IST 2018]; root of context hierarchy

Sep 09, 2018 5:16:54 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions

INFO: Loading XML bean definitions from class path resource [resource/spring.xml]

Sep 09, 2018 5:16:54 PM org.springframework.core.io.support.PropertiesLoaderSupport loadProperties

INFO: Loading properties file from class path resource [resource/details.properties]

username is mounesh;

password is password;

I18N

Using this we can provide the language support

I10N

Using this we can provide the business support or validation support

**SPRING MVC**

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

Spring features

Form backup support

It holds the data into pojo classes ,

Controller supports both multi action and multi form

It holds the previous form data

Validation support

Internationalization support

Interceptor support

ViewResolvers support

Exception handling

Spring mvc is based on the jsp-model architecture

Jsp model architecture there are four architecture

1. Jsp model 1 arch – front controller as jsp ()
2. Jsp model 2 arch – front controller as servlet -> struct 1.x ,spring,JSF
3. Jsp model 3 arch – front controller as filter -> struct 2.x
4. Jsp model 4 arch – front controller as tag lib

In spring framework front controller is Dispatcher Servlet

**Data Access Operations (DAO)**

Using dao layer we can connect to rdbms,xml,loap and nosql

If we are using dao then we should follow interface implementation pattern follow

Because implementation will be provided by the third party in future if the implementation changed that should not affect our business that’s why we should follow interface implementation pattern.

It is recommended to use the connection pool concept,

Connection if direct database interaction requires 1sec if we use connection pool concept it will take 0.1 sec

Limitation is there in connecting the database directly if one more person tries to open the direct connection it will throw the exception if in case of the connection pool concept we can use the waiting conception then it gives the connection to the person.

Sun provided interface for connection pool is DataSource all the server vendors gives the implementation classes

Connection map uses two map object one used map and unused map

Spring JdbcTemplate class :

If we use jdbc template method it will only create the connection and close the connection

And we will not get any compile time exception we will get runtime exception

All jdbc template methods are type safe . using Types we can send the specified type

DataAccessException – spring-dao.jar

setDatasource – for connection pool

update – for update the database operation

queryForInt

queryForFloat

queryForDouble

queryForObject

queryForList

queryForMap

AOP in spring

AOP => aspect orient Program

Aspect means service

Advice means service provider

PointCut point/condition to execute aspects for business methods

Advisor is point cut with advice combination

Proxy combination of service and business produces proxy object