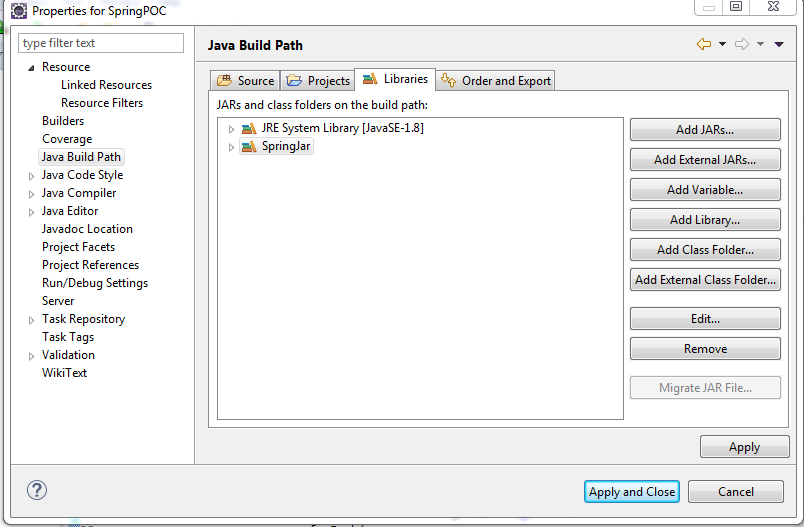
**Download link for Spring framework distribution:**

[**http://repo.spring.io/release/org/springframework/spring/5.1.0.RELEASE/**](http://repo.spring.io/release/org/springframework/spring/5.1.0.RELEASE/)

**Download link for Apache Common logging Jar**:

<https://commons.apache.org/proper/commons-logging/download_logging.cgi>

**Add External Library:**



**Schema:**

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN" "http://www.springframework.org/dtd/spring-beans-2.0.dtd">

<beans>

**Scenario:**

**A class has 2 different Constructor with single parameter where each parameter is different types.**

<constructor-arg type="int" value="30"/>

In that scenario we have to mention **type** to give hints spring Application Context to execute specific constructor.

**How to inject Object in a bean:**

Create bean of that object in spring.xml. Use **ref** keyword along with **property** tag to inject that bean into the owner bean during creation of owner bean.

<bean id="profile" class="org.project.Partha.Profile">

<property name="firstname" value="Partha"/>

<property name="lastname" value="Nandy"/>

</bean>

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

**Inner Bean:**

<property name="company">

<bean class="org.project.Partha.Company">

<property name="companyname" value="TCS"/>

<property name="year" value="4"/>

</bean>

</property>

**Specification of id is not require in bean for inner bean as we are not referring the bean.**

**Create Alias of A bean:**

<alias name="employee" alias="employee-alias"/>

Also can be added **name** keyword within bean tag.

Ref in bean tag can refer name, alias and id of another bean. To restrict reference with id for ref, we have to use like below:

<idref=”<<>>”/> within property tag.

**Initializing Collections:**

List:

Need to add list member in main class like below:

**private List<ProjectDetails> projects;**

In spring.xml, we have to create all bean ids for that object which is used for listing.

<bean id="projectA" class="org.project.Partha.ProjectDetails">

<property name="projectName" value="wyndham"/>

</bean>

<bean id="projectB" class="org.project.Partha.ProjectDetails">

<property name="projectName" value="citi"/>

</bean>

Then to Create list of beans we have to add in spring.xml like below:

<property name="projects">

<list>

<ref bean="projectA"/>

<ref bean="projectB"/>

</list>

</property>

**Autowire Bean:**

**ByName:**

Member variable name should be same with bean id mentioned in spring.xml

private Address address;

<bean id="address" class="org.project.Partha.Address">

<property name="city" value="kolkata"/>

</bean>

<bean id="employee" class="org.project.Partha.Employee" **autowire="byName"**>

**ByType:**

Member variable type should be same with type of bean definition mentioned in spring.xml.

**Note:** Only for single member variable it will works. Otherwise spring cannot decide which bean definition will be wired with which member variable.

<bean id="employee" class="org.project.Partha.Employee" **autowire="byType"**>

**constructor:**

it behaves like **byType** but works with constructor.

**Note** : for a specific type only one bean definition is allowed.

<bean id="employee" class="org.project.Partha.Employee" **autowire="constructor"**>

**public** Employee(Profile profile, Address address) {

**this**.profile = profile;

**this**.address = address;

}

<bean id=*"profile"* class=*"org.project.Partha.Profile"*>

<property name=*"firstname"* value=*"Partha"*/>

<property name=*"lastname"* value=*"Nandy"*/>

</bean>

<bean id=*"address"* class=*"org.project.Partha.Address"*>

<property name=*"city"* value=*"kolkata"*/>

</bean>

Default autowire off.

**ApplicationContext** :

**Default scenario:**

During initialization of application context, all possible bean definitions of spring.xml gets initialized.

When **getBean** happens only those created beans will be hand over as per requirement.

**Basic Bean Scope:**

**Singleton:**

Only once per Spring Container. Means one bean per bean definition is created during Application Context For all requests, same bean for a specific bean definition are provided.

**Prototype:**

New bean created with every request or reference.

<bean id="employee" class="org.project.Partha.Employee" scope="**prototype**">

**Request**-New bean per servlet request

**Session**- New Bean per session

**Global Session**- New bean per global HTTP session.

**Get Access ApplicationContext from another class(bean):**

public class Employee implements ApplicationContextAware

Means the class which implements **ApplicationContextAware** that class will able to aware of Application Context.

Implement BeanNameAware by a class to get bean name.

@Override

**public** **void** setBeanName(String beanName) {

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

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

}

**Bean Definition Inheritance:**

<bean id=*"parentemp"* class=*"org.project.Partha.Employee"*>

<property name=*"id"* value=*"20"*/>

</bean>

<bean id="employee" class="org.project.Partha.Employee" autowire="byName" parent="parentemp">

**Inheritance of list in bean definition:**

If one list tag for a particular property is created with some bean in parent bean definition and if same list tag for that same property is created in child bean definition.

In normal scenario, Child will override parent bean reference in list.

<bean id=*"parentemp"* class=*"org.project.Partha.Employee"*>

<property name=*"id"* value=*"20"*/>

<property name=*"projects"*>

<list>

<ref bean=*"projectA"*/>

</list>

</property>

</bean>

<bean id=*"employee"* class=*"org.project.Partha.Employee"* autowire=*"byName"* parent=*"parentemp"*><

<property name=*"projects"*>

<list>

<!-- <ref bean="projectA"/>-->

<ref bean=*"projectB"*/>

</list>

</property>

To merge parent and child, we need to add attribute **merge=true** with list tag within Child bean definition for that particular property.

<property name=*"projects"*>

<list merge=*"true"*>

<!-- <ref bean="projectA"/>-->

<ref bean=*"projectB"*/>

</list>

</property>

**InitializingBean interface:**

If any class implements above mentioned interface, that means that method of class will be called after initial bean initialization.

@Override

**public** **void** setApplicationContext(ApplicationContext arg0) **throws** BeansException {

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

System.***out***.println("Bean initialization Done");

}

**DisposableBean:**

It is used to perform some execution during destroy of bean.

AbstractApplicationContext context= **new** ClassPathXmlApplicationContext("spring.xml");

context.registerShutdownHook();

**public** **class** Employee **implements** ApplicationContextAware, BeanNameAware,InitializingBean,DisposableBean {

**private** **int** id;

@Override

**public** **void** destroy() **throws** Exception {

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

System.***out***.println("Disposal Bean");

}

**Drawback: We are using spring interface. We are binding with spring itself.**

**Alternative way initialize action and destroy action of bean:**

Own methods has been created like below:

**public** **void** myinit()

{

System.***out***.println("My init method is created");

}

**public** **void** destroy()

{

System.***out***.println("My destroy method is created");

}

**Configuration in spring.xml:**

<bean id=*"parentemp"* class=*"org.project.Partha.Employee"* init-method=*"myinit"* destroy-method=*"destroy"*>

<property name=*"id"* value=*"20"*/>

<property name=*"projects"*>

**To configure globally for all beans:**

<beans default-init-method=*"myinit"* default-destroy-method=*"destroy"*>

<bean id=*"parentemp"* class=*"org.project.Partha.Employee"* >

<property name=*"id"* value=*"20"*/>

**BeanPostProcessor:**

First we have to create new class which implements **BeanPostProcessor** interface. In that case all methods of that class will be executed before and after initialization of each bean.

@Override

**public** Object postProcessAfterInitialization(Object Bean, String BeanName) **throws** BeansException {

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

System.***out***.println("After Initialization Bean");

**return** Bean;

@Override

**public** Object postProcessBeforeInitialization(Object Bean, String BeanName) **throws** BeansException {

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

System.***out***.println("Before Initialization Bean");

**return** Bean;

}

**In spring.xml, below needs to add:**

<bean class=*"org.project.Partha.DisplayNameBeanPostProcessor"*/>

</beans>

**BeanFactoryPostProcessor:**

A new class needs to create which implements **BeanFactoryPostProcessor** and needs to implement method as well.

@Override

**public** **void** postProcessBeanFactory(ConfigurableListableBeanFactory arg0) **throws** BeansException {

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

System.***out***.println("My Bean Factory Post Processor is called");

}

Above method will be called after initialization of application context/bean factory object.

**spring.xml:**

<bean class=*"org.project.Partha.MyBeanFactoryPP"*/>

</beans>

BeanPostProcessor is called🡪BeanFactory is initialized🡪All singleton bean is initialized

**To Use placeholder in spring.xml need to use spring provided BeanPostProcessor like below:**

Need to create .properties file.

**Need to add below details like below in spring.xml:**

Profile.firstname=Partha

Profile.lastname=Nandy

<bean id=*"profile"* class=*"org.project.Partha.Profile"*>

<property name=*"firstname"* value=*"${Profile.firstname}"*/>

<property name=*"lastname"* value=*"${Profile.lastname}"*/>

</bean>

<bean class=*"org.project.Partha.MyBeanFactoryPP"*/>

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

<property name=*"locations"* value=*"config.properties"*/>

</bean>

**Coding to Interfaces:**

We need to create one interface and need to implement by class.

**Then we need to mention below line like:**

Human emp=(Human)context.getBean("employee");

//Employee emp=(Employee)context.getBean("employee-alias");

emp.totalPackage();

Human is an interface.

In that case we may hide implementation class details.

**@Required Annotation**

During initialization of a bean, if any dependency does not available, spring throws **NullPointerException.**

To get early error after application context/Bean Factory initialization and just before bean initialization we have to add **@Required above all setter method which is mandatory to execute during bean initialization.**

@Required

**public** **void** setId(**int** id) {

**this**.id = id;

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

}

**Need to add below in spring.xml as spring defined Bean Post Processor:**

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

</beans>

**In Console:**

Caused by: org.springframework.beans.factory.BeanInitializationException: Property 'id' is required for bean 'parentemp'

at org.springframework.beans.factory.annotation.RequiredAnnotationBeanPostProcessor.postProcessPropertyValues(RequiredAnnotationBeanPostProcessor.java:151)

at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.populateBean(AbstractAutowireCapableBeanFactory.java:1148)

at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.doCreateBean(AbstractAutowireCapableBeanFactory.java:519)

... 11 more

**@Autowire Annotation:**

First it will try to auto wire dependency bean using **byType** if in spring.xml, multiple bean definitions with same type is present, and then spring will try to auto wire dependency bean using **byName.**

In some scenario, if bean id and member variable name is different (means not possible using **byName**) in case of availability of multiple bean definitions in spring.xml, we have to introduce **qualifier** tag.

**Path for annotation-config.xml:**

<https://github.com/spring-projects/spring-framework/blob/master/spring-context/src/test/resources/org/springframework/context/annotation/configuration/annotation-config.xml>

**Below Details need to add in spring.xml:**

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

</beans>

<bean id=*"addressA"* class=*"org.project.Partha.Address"*>

<qualifier value=*"addressfirst"*/>

<property name=*"city"* value=*"kolkata"*/>

</bean>

<bean id=*"addressB"* class=*"org.project.Partha.Address"*>

<qualifier value=*"addressSecond"*/>

<property name=*"city"* value=*"Behala"*/>

</bean>

@Autowired

@Qualifier("addressfirst")

**public** **void** setAddress1(Address address1) {

**this**.address1 = address1;

}

@Autowired

@Qualifier("addressSecond")

**public** **void** setAddress2(Address address2) {

**this**.address2 = address2;

}

**To mention all annotation related Bean Post Processor:**

<context:annotation-config/>

**JSR 250 annotation:**

**@Resource:**

To tag specific bean id with a setter method of a specific class we have to write code like below:

@Resource(name="addressA")

**public** **void** setAddress1(Address address1) {

**this**.address1 = address1;

}

Injection by name

If we don’t specify name, if member variable name is same with bean id, then that will be injected using setter.

**@PostConstruct:**

This annotation is used to define **Init** method without any configuration in .**xml**. Method is called after dependency injection (bean initialization is done).

@PostConstruct

**public** **void** myinitemployee()

{

System.***out***.println("My init employee method is created");

}

**@PreDestroy:**

This annotation is used to define destroy method without any configuration in .xml. This method is called when the bean is about to destroy.

@PreDestroy

**public** **void** mydestroyemployee()

{

System.***out***.println("My destroy employee method is created");

}

AbstractApplicationContext context= **new** ClassPathXmlApplicationContext("spring.xml");

context.registerShutdownHook();

Above snippet is used for register hook.

**@Component:**

This above mentioned annotation is used to mention a class as bean.

But that class will be restricted with single type of bean instance.

**In spring.xml:**

<context:component-scan base-package=*"org.project.Partha"*></context:component-scan>

</beans>

**In class:**

@Component

**public** **class** TCS **implements** Human {

**private** String id;

Stereotype Annotation:

@**Repository**

**@Service**

**@Controller**

**@Component**