Spring Dependency Injection

APRIL 6, 2018 BY PANKAJ — 46 COMMENTS

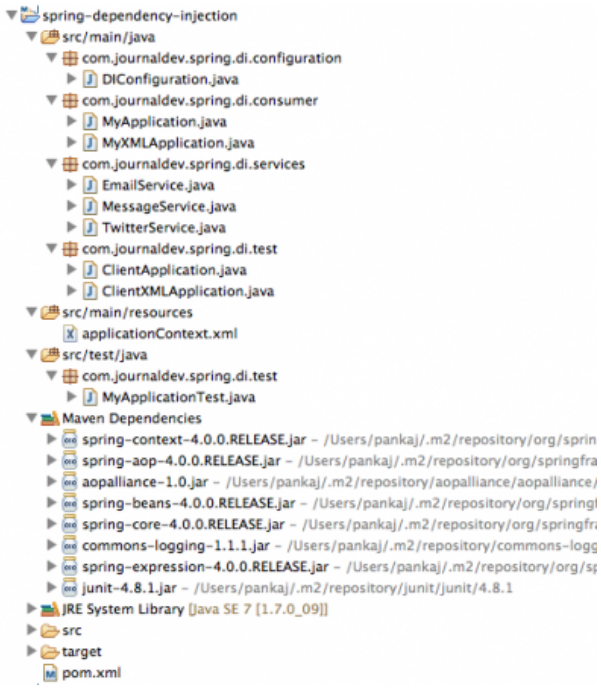
Today we will look into Spring Dependency Injection. **Spring Framework** core concepts are "**Dependency Injection**" and "**Aspect Oriented Programming**". I have written earlier about **Java Dependency Injection** and how we can use **Google Guice** framework to automate this process in our applications.

Table of Contents [hide]

Spring Dependency Injection

This tutorial is aimed to provide details about Spring Dependency Injection example with both annotation based configuration and XML file based configuration. I will also provide JUnit test case example for the application, since easy testability is one of the major benefits of dependency injection.

I have created *spring-dependency-injection* maven project whose structure looks like below image.



Let's look at each of the components one by one.

Spring Dependency Injection – Maven Dependencies

I have added Spring and JUnit maven dependencies in pom.xml file, final pom.xml code is below.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
```

```
	<groupId>com.journaldev.spring</groupId>
	<artifactId>spring-dependency-injection</artifactId>
	<version>0.0.1-SNAPSHOT</version>

	<dependencies>
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-context</artifactId>
			<version>4.0.0.RELEASE</version>
		</dependency>
		<dependency>
			<groupId>junit</groupId>
			<artifactId>junit</artifactId>
			<version>4.8.1</version>
			<scope>test</scope>
		</dependency>
```

Current stable version of Spring Framework is *4.0.0.RELEASE* and JUnit current version is *4.8.1*, if you are using any other versions then there might be a small chance that the project will need some change. If you will build the project, you will notice some other jars are also added to maven dependencies because of transitive dependencies, just like above image.

Spring Dependency Injection – Service Classes

Let's say we want to send email message and twitter message to the users. For dependency injection, we need to have a base class for the services. So I have `MessageService` interface with single method declaration for sending message.

```
package com.journaldev.spring.di.services;

public interface MessageService {

	boolean sendMessage(String msg, String rec);
}
```

Now we will have actual implementation classes to send email and twitter message.

```
package com.journaldev.spring.di.services;

public class EmailService implements MessageService {

	public boolean sendMessage(String msg, String rec) {
		System.out.println("Email Sent to "+rec+ " with Message="+msg);
		return true;
	}

}
```

```
package com.journaldev.spring.di.services;

public class TwitterService implements MessageService {

	public boolean sendMessage(String msg, String rec) {
		System.out.println("Twitter message Sent to "+rec+ " with
Message="+msg);
		return true;
	}

}
```

Now that our services are ready, we can move on to Component classes that will consume the service.

Spring Dependency Injection – Component Classes

Let's write a consumer class for above services. We will have two consumer classes – one with Spring annotations for autowiring and another without annotation and wiring configuration will be provided in the XML configuration file.

```
package com.journaldev.spring.di.consumer;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.stereotype.Component;

import com.journaldev.spring.di.services.MessageService;

@Component
public class MyApplication {

	//field-based dependency injection
	//@Autowired
	private MessageService service;

//	constructor-based dependency injection
//	@Autowired
//	public MyApplication(MessageService svc){
//		this.service=svc;
//	}

	@Autowired
```

Few important points about MyApplication class:

- `@Component` annotation is added to the class, so that when Spring framework will scan for the components, this class will be treated as component. @Component annotation can be applied only to the class and it's retention policy is Runtime. If you are not not familiar with Annotations retention policy, I would suggest you to read java annotations tutorial.
- `@Autowired` annotation is used to let Spring know that autowiring is required. This can be applied to field, constructor and methods. This annotation allows us to implement constructor-based, field-based or method-based dependency injection in our components.
- For our example, I am using method-based dependency injection. You can uncomment the constructor method to switch to constructor based dependency injection.

Now let's write similar class without annotations.

```
package com.journaldev.spring.di.consumer;

import com.journaldev.spring.di.services.MessageService;

public class MyXMLApplication {

	private MessageService service;

	//constructor-based dependency injection
//	public MyXMLApplication(MessageService svc) {
//		this.service = svc;
//	}

	//setter-based dependency injection
	public void setService(MessageService svc){
		this.service=svc;
	}

	public boolean processMessage(String msg, String rec) {
		// some magic like validation, logging etc
		return this.service.sendMessage(msg, rec);
	}
```

A simple application class consuming the service. For XML based configuration, we can use implement either constructor-based spring dependency injection or method-based spring dependency injection. Note that method-based and setter-based injection approaches are same, it's just that some prefer calling it setter-based and some call it method-based.

Spring Dependency Injection Configuration with Annotations

For annotation based configuration, we need to write a Configurator class that will be used to inject the actual implementation bean to the component property.

```
package com.journaldev.spring.di.configuration;

import org.springframework.context.annotation.Bean;
```

```
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

import com.journaldev.spring.di.services.EmailService;
import com.journaldev.spring.di.services.MessageService;

@Configuration
@ComponentScan(value={"com.journaldev.spring.di.consumer"})
public class DIConfiguration {

	@Bean
	public MessageService getMessageService(){
		return new EmailService();
	}
}
```

Some important points related to above class are:

- `@Configuration` annotation is used to let Spring know that it's a Configuration class.
- `@ComponentScan` annotation is used with `@Configuration` annotation to specify the packages to look for Component classes.
- `@Bean` annotation is used to let Spring framework know that this method should be used to get the bean implementation to inject in Component classes.

Let's write a simple program to test our annotation based Spring Dependency Injection example.

```
package com.journaldev.spring.di.test;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import com.journaldev.spring.di.configuration.DIConfiguration;
import com.journaldev.spring.di.consumer.MyApplication;

public class ClientApplication {

	public static void main(String[] args) {
		AnnotationConfigApplicationContext context = new
AnnotationConfigApplicationContext(DIConfiguration.class);
		MyApplication app = context.getBean(MyApplication.class);

		app.processMessage("Hi Pankaj", "pankaj@abc.com");

		//close the context
		context.close();
	}

}
```

`AnnotationConfigApplicationContext` is the implementation of `AbstractApplicationContext` abstract class and it's used for autowiring the services to components when annotations are used. `AnnotationConfigApplicationContext` constructor takes Class as argument that will be used to get the bean implementation to inject in component classes.

getBean(Class) method returns the Component object and uses the configuration for autowiring the objects. Context objects are resource intensive, so we should close them when we are done with it. When we run above program, we get below output.

```
Dec 16, 2013 11:49:20 PM
org.springframework.context.support.AbstractApplicationContext prepareRefresh
INFO: Refreshing
org.springframework.context.annotation.AnnotationConfigApplicationContext@3067ed13:
startup date [Mon Dec 16 23:49:20 PST 2013]; root of context hierarchy
Email Sent to pankaj@abc.com with Message=Hi Pankaj
Dec 16, 2013 11:49:20 PM
org.springframework.context.support.AbstractApplicationContext doClose
INFO: Closing
org.springframework.context.annotation.AnnotationConfigApplicationContext@3067ed13:
startup date [Mon Dec 16 23:49:20 PST 2013]; root of context hierarchy
```

Spring Dependency Injection XML Based Configuration

We will create Spring configuration file with below data, file name can be anything.

applicationContext.xml code:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.0.xsd">

<!--
        <bean id="MyXMLApp"
class="com.journaldev.spring.di.consumer.MyXMLApplication">
                <constructor-arg>
                        <bean
class="com.journaldev.spring.di.services.TwitterService" />
                </constructor-arg>
        </bean>
-->
        <bean id="twitter" class="com.journaldev.spring.di.services.TwitterService">
</bean>
        <bean id="MyXMLApp"
class="com.journaldev.spring.di.consumer.MyXMLApplication">
```

```
        <property name="service" ref="twitter"></property>
```

Notice that above XML contains configuration for both constructor-based and setter-based spring dependency injection. Since `MyXMLApplication` is using setter method for injection, the bean configuration contains *property* element for injection. For constructor based injection, we have to use *constructor-arg* element.

The configuration XML file is placed in the source directory, so it will be in the classes directory after build.

Let's see how to use XML based configuration with a simple program.

```
package com.journaldev.spring.di.test;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.journaldev.spring.di.consumer.MyXMLApplication;

public class ClientXMLApplication {

	public static void main(String[] args) {
		ClassPathXmlApplicationContext context = new
ClassPathXmlApplicationContext(
				"applicationContext.xml");
		MyXMLApplication app = context.getBean(MyXMLApplication.class);

		app.processMessage("Hi Pankaj", "pankaj@abc.com");

		// close the context
		context.close();
	}

}
```

`ClassPathXmlApplicationContext` is used to get the ApplicationContext object by providing the configuration files location. It has multiple overloaded constructors and we can provide multiple config files also.

Rest of the code is similar to annotation based configuration test program, the only difference is the way we get the ApplicationContext object based on our configuration choice.

When we run above program, we get following output.

```
Dec 17, 2013 12:01:23 AM
org.springframework.context.support.AbstractApplicationContext prepareRefresh
INFO: Refreshing
org.springframework.context.support.ClassPathXmlApplicationContext@4eeaabad: startup
date [Tue Dec 17 00:01:23 PST 2013]; root of context hierarchy
```

```
Dec 17, 2013 12:01:23 AM org.springframework.beans.factory.xml.XmlBeanDefinitionReader
loadBeanDefinitions
INFO: Loading XML bean definitions from class path resource [applicationContext.xml]
Twitter message Sent to pankaj@abc.com with Message=Hi Pankaj
Dec 17, 2013 12:01:23 AM
org.springframework.context.support.AbstractApplicationContext doClose
INFO: Closing
org.springframework.context.support.ClassPathXmlApplicationContext@4eeaabad: startup
date [Tue Dec 17 00:01:23 PST 2013]; root of context hierarchy
```

Notice that some of the output is written by Spring Framework. Since Spring Framework uses log4j for logging purpose and I have not configured it, the output is getting written to console.

Spring Dependency Injection JUnit Test Case

One of the major benefit of dependency injection in spring is the ease of having mock service classes rather than using actual services. So I have combined all of the learning from above and written everything in a single JUnit 4 test class for dependency injection in spring.

```
package com.journaldev.spring.di.test;

import org.junit.Assert;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

import com.journaldev.spring.di.consumer.MyApplication;
import com.journaldev.spring.di.services.MessageService;

@Configuration
@ComponentScan(value="com.journaldev.spring.di.consumer")
public class MyApplicationTest {

        private AnnotationConfigApplicationContext context = null;

        @Bean
        public MessageService getMessageService() {
```

The class is annotated with `@Configuration` and `@ComponentScan` annotation because *getMessageService()* method returns the `MessageService` mock implementation. That's why *getMessageService()* is annotated with `@Bean` annotation.

Since I am testing `MyApplication` class that is configured with annotation, I am using `AnnotationConfigApplicationContext` and creating it's object in the setUp() method. The context is

getting closed in *tearDown()* method. *test()* method code is just getting the component object from context and testing it.

Do you wonder how Spring Framework does the autowiring and calling the methods that are unknown to Spring Framework. It's done with the heavy use of Java Reflection that we can use to analyze and modify the behaviors of the classes at runtime.

Download Spring Dependency Injection Project

Download the sample Spring Dependency Injection (DI) project from above URL and play around with it to learn more.

NEXT »
Spring MVC Tutorial

About Pankaj

If you have come this far, it means that you liked what you are reading. Why not reach little more and connect with me directly on **Google Plus**, **Facebook** or **Twitter**. I would love to hear your thoughts and opinions on my articles directly.

Recently I started creating video tutorials too, so do check out my videos on **Youtube**.

FILED UNDER: SPRING

Comments

Pranali says
APRIL 16, 2018 AT 10:09 PM
what is significance of @configuration. Even if i remove it from ApplConfig it is executed correctly. and if my properties file has same keys and i write @PropertySource twice then which key will be fetched from both properties file.
Reply

Pankaj says
MAY 23, 2018 AT 12:47 AM
Read this tutorial to understand why it's better to use @Configuration annotation.
https://www.journaldev.com/21033/spring-configuration-annotation
Reply

Ravella Baji says
MARCH 5, 2018 AT 9:51 AM
gracias, amigo.
Reply

Ennahdi El Idrissi, Mohamed says
NOVEMBER 21, 2017 AT 7:04 AM
Hi Pankaj,
Why is DIConfiguration class capable of having only one @Bean method ?
What should be done to invoke TwitterService? Do I have to create a distinct Configuration class?
Reply

Ashok Singh says
AUGUST 27, 2017 AT 12:12 AM
Hi Pankaj,
As this question has been asked by some one . Please can you explain ?
Thanks for nice explanation.
But in above DIConfiguration class, its kind of hardcoding and returning "return new EmailService(), which says always EmailService bean is getting injected over there.
For some scenarios, if I want to instantiate my component with TwitterService() how to do that again?
I meant.. few instances need to be injected with EmailService() and few instances need to be injected with TwitterService().
How to do that? Please explain.

Reply

Pankaj says
AUGUST 27, 2017 AT 1:06 AM
You can do that by injecting message service instance using XML file, create two different beans – one for Twitter and one for Email. Then use them as per your need.
Reply

Ashok Singh says
SEPTEMBER 17, 2017 AT 10:38 AM
Thanks Pankaj
Reply

chandu ujjina says
APRIL 25, 2018 AT 11:35 PM
You can instantiate both beans see the below code

```
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import com.corepractice.Dao.MessageService;
import com.corepractice.Daoimpl.EmailService;
import com.corepractice.Daoimpl.TwitterService;
@Configuration
@ComponentScan("com.corepractice")
public class DIConfiguration {
public DIConfiguration() {
super();
System.out.println("Inside DIConfiguration Constructor::");
}
@Bean
@Qualifier("emailService")
public MessageService getEmailMessageService(){
System.out.println("Inside Bean declaration::");
return new EmailService();
}
@Bean
@Qualifier("twitterService")
public MessageService getTwitterMessageService(){
System.out.println("Inside Bean declaration::");
return new TwitterService();
}
}
```

```
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Service;
import com.corepractice.Dao.MessageService;
@Service
@Qualifier("emailService")
public class EmailService implements MessageService {
@Override
public boolean sendMessage(String msg, String rec) {
System.out.println("Email Sent to "+rec+ " with Message="+msg);
return true;
}
}
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Service;
import com.corepractice.Dao.MessageService;
@Service
@Qualifier("twitterService")
public class TwitterService implements MessageService{
@Override
public boolean sendMessage(String msg, String rec) {
System.out.println("Twitter message Sent to "+rec+ " with Message="+msg);
return true;
}
}
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import com.corepractice.Dao.MessageService;
import com.corepractice.config.DIConfiguration;
public class ClientApplication {
public static void main(String[] args) {
AnnotationConfigApplicationContext context = new
AnnotationConfigApplicationContext(DIConfiguration.class);
MessageService emailservice=(MessageService) context.getBean("emailService");
emailservice.sendMessage("Hi Pankaj","pankaj@abc.com");
MessageService twitterService=(MessageService) context.getBean("twitterService");
twitterService.sendMessage("Hi Pankaj","pankaj@abc.com");
//close the context
context.close();
}
}
```

Reply

Jaydeep says

MAY 19, 2018 AT 11:38 AM

Thanks Man

Reply

Anoop says
OCTOBER 26, 2016 AT 7:24 PM
Awesome demo and explanation.
Reply

RafalQA says
OCTOBER 16, 2016 AT 7:35 AM
Hello, I have some question.
There is any another way to use MyApplication class with autowired without the:
“AnnotationConfigApplicationContext context = new
AnnotationConfigApplicationContext(DIConfiguration.class);
MyApplication app = context.getBean(MyApplication.class);” ???
Reply

Kapil Verm says
FEBRUARY 20, 2016 AT 5:19 PM
Nice explanation. Thanks Pankaj!
Reply

Prasanth Varada says
FEBRUARY 17, 2016 AT 8:52 PM
Hi Pankaj,
Thanks for nice explanation.
But in above DIConfiguration class, its kind of hardcoding and returning “return new EmailService(), which says always EmailService bean is getting injected over there.
For some scenarios, if I want to instantiate my component with TwitterService() how to do that again?
I meant.. few instances need to be injected with EmailService() and few instances need to be injected with TwitterService().
How to do that? Please explain.
Reply

Márton P says
DECEMBER 21, 2016 AT 6:45 AM
I am also curious about this. To me the whole point of this setup seems to be about being able to swap beans on the fly, yet I haven’t got setService to work.
Reply

anonoumus says
FEBRUARY 4, 2016 AT 12:05 AM
Nice explanation
Reply

John K Njue says
JANUARY 27, 2016 AT 3:04 AM
Nice and well articulated tutorial. will catch up with you on social media.
Reply

Venkatraman E K says
JANUARY 24, 2016 AT 6:26 AM
While I was trying to implement the DI on annotation based. I couldn't find the componentscan. Where i can get the Jar.
If I try to download your source code and try to compile the code. I'm coming across the following issue as:Exception in thread "main" org.springframework.beans.factory.NoSuchBeanDefinitionException: No unique bean of type [com.journaldev.spring.di.consumer.MyApplication] is defined: expected single bean but found 0:
Could you help us resolve on the above issues.
Reply

bala says
NOVEMBER 1, 2015 AT 1:46 AM
very good article sir.nice demo thanks for sharing this article
Reply

karablaxos says
OCTOBER 5, 2015 AT 9:09 AM
Hi pankaj why if we remove the @ComponentScan anottation and try to configure @Component("MyApplication") by xml component-scan is says that No bean named 'MyApplication' is defined?
Reply

Hernán says
SEPTEMBER 15, 2015 AT 8:16 AM
Hi, i am sorry for my simple question, but i have done the tutorial and it works fine, except for the JUnit file at the end. Since it doesnt have a Main method, how am i run the tests ?

Reply

BinhLe says
OCTOBER 6, 2015 AT 4:40 AM
Hi Hernán, you can run it by JUnit.
I'm sorry for my english. :))
Reply

Pankaj says
OCTOBER 6, 2015 AT 4:42 AM
Run it as JUnit test in Eclipse.
Reply

Shashank says
SEPTEMBER 12, 2015 AT 8:05 PM
Hi Pankaj,
I am using mybatis with spring. I have an interface which is implemented in xml. There is no implementing class of the interface. The interface has methods like select,insert,delete, etc. I am getting the name of the mapper or interface from the database and then I need to execute the methods at runtime. It needs to be dynamic so I cant know the name of the interface or mapper beforehand. Any help would be great.
Thanks.
Reply

vamsi says
AUGUST 19, 2015 AT 8:19 AM
Hiiii mr. pankaj,
I have done console application with annotations am getting this exception can please let me know what is this
Exception in thread "main" org.springframework.beans.factory.NoSuchBeanDefinitionException: No qualifying bean of type [com.sapmle.MyCommunication] is defined
at
org.springframework.beans.factory.support.DefaultListableBeanFactory.getBean(DefaultListableBeanFactory.java:318)
at
org.springframework.context.support.AbstractApplicationContext.getBean(AbstractApplicationContext.java:985)
at Test.main(Test.java:11)
Reply

Ram says
AUGUST 8, 2015 AT 11:10 AM
Hi This is nice demo.
I think if you look at it then its a kind of Strategy design pattern with Spring DI and with annotations.
Reply

Umesh Hemant Annegirikar says
MAY 30, 2015 AT 5:12 AM
I have got the following exception while using xml configuration
Exception in thread "main" java.lang.VerifyError: class org.springframework.expression.spel.SpelException overrides final method getPosition.()I
at java.lang.ClassLoader.defineClass1(Native Method)
at java.lang.ClassLoader.defineClass(Unknown Source)
at java.security.SecureClassLoader.defineClass(Unknown Source)
at java.net.URLClassLoader.defineClass(Unknown Source)
at java.net.URLClassLoader.access$100(Unknown Source)
at java.net.URLClassLoader$1.run(Unknown Source)
at java.net.URLClassLoader$1.run(Unknown Source)
at java.security.AccessController.doPrivileged(Native Method)
at java.net.URLClassLoader.findClass(Unknown Source)
at java.lang.ClassLoader.loadClass(Unknown Source)
at sun.misc.Launcher$AppClassLoader.loadClass(Unknown Source)
at java.lang.ClassLoader.loadClass(Unknown Source)
at org.springframework.context.expression.StandardBeanExpressionResolver.
(StandardBeanExpressionResolver.java:57)
at
org.springframework.context.support.AbstractApplicationContext.prepareBeanFactory(AbstractApplicationContext.java:441)
at
org.springframework.context.support.AbstractApplicationContext.refresh(AbstractApplicationContext.java:352)
at org.springframework.context.support.ClassPathXmlApplicationContext.
(ClassPathXmlApplicationContext.java:139)
at org.springframework.context.support.ClassPathXmlApplicationContext.
(ClassPathXmlApplicationContext.java:83)
at com.SpringDI.test.ClientXMLApplication.main(ClientXMLApplication.java:10)
I have tried to debug it but i am not able to resolve it
Reply

Nena007 says
MAY 1, 2015 AT 1:25 PM
In your annotated example, you haven't mentioned how type collisions would be resolved. Since you have two types of MessageService implementations, how would the Spring IoC container resolve which MessageService type (email or twitter) to inject?
Reply

Cuong Huy To says
AUGUST 8, 2016 AT 7:47 AM
It is through the file DIConfiguration
@Configuration
@ComponentScan(value={"com.journaldev.spring.di.consumer"})
public class DIConfiguration {
@Bean
public MessageService getMessageService(){
return new EmailService();
}
}
Reply

harikishore says
MARCH 16, 2015 AT 3:37 AM
nice articel…simply superb!!!
Reply

Frank says
FEBRUARY 2, 2015 AT 8:53 PM
What way do you recommend me for use spring 4? Anotattions or Xml?
Reply

jery says
MAY 19, 2015 AT 10:12 PM
Xml is best way
Reply

trinsit.w says
DECEMBER 24, 2014 AT 2:23 AM
Good article. Love it! Thank you very much.
Reply

Bikash Mohapatra says
DECEMBER 18, 2014 AT 11:43 PM

Very nice explanation and working fine.
Thanks..
Reply

Andreas Papandreas says
DECEMBER 17, 2014 AT 11:51 AM
Pankaj, you said that 'Context objects are resource intensive, so we should close them when we are done with it'. What about WebApplicationContext, it is from org.springframework.web.context.support.WebApplicationContextUtils and do we neeed to close it since it does not have any close method?
Reply

Ty Davis says
DECEMBER 4, 2014 AT 7:29 PM
This doesnt work at all. I wont be coming here anymore
Reply

Pankaj says
DECEMBER 5, 2014 AT 10:26 AM
Well it works for me and many others, see others comments too. If you are facing any issues, then check your settings and try to solve it.
Reply

Ismael says
JULY 2, 2014 AT 12:33 PM
I think that the DIConfiguration.java is wrong on the "service" configuration. It says
@Bean
public MessageService getMessageService(){
return new EmailService();
}
As java config uses the name of the method as id, it should be instead
@Bean
public MessageService service(){
return new EmailService();
}
Then the MyApplication.java would have access to the service.
On DIConfiguration.java you should decide if you need the TwitterService or EmailService.
Unless I am missing something □
Reply

Pankaj says
JULY 2, 2014 AT 8:20 PM
You can choose any method name, that is the beauty of Spring framework and how it process @Bean and @Autowired annotations.
Reply

mani says
JUNE 9, 2014 AT 11:31 PM
could use please explain @value annotation with an example using annotation based dependency injection
Reply

Atmrakash Sharma says
JUNE 4, 2014 AT 1:33 AM
nice article ..earlier i was usnig xml base configuration..but after reading this article i moved my p[roject to pannotaion based configuration file…thanks (Y)
Reply

suvendu says
MAY 15, 2014 AT 6:21 AM
great explanation .. thank you. □
Reply

K KKumar says
APRIL 16, 2014 AT 5:49 PM
Nice tutorial.
Reply

Nandu says
MARCH 25, 2014 AT 8:41 AM
Very nice explanation…I appreciate your efforts.
Before that I read all above theoretically but u explain it with an example which is easily understandable to any one.
thanks
Reply

Adarsha says
MARCH 20, 2014 AT 1:59 PM
Hi Pankaj,
I am new for Spring , I followed your above code with some changes, But i am getting Error :
IOException parsing XML document from class path resource [CoreSpring/src/main/resources/Test.xml]; nested exception is java.io.FileNotFoundException: class path resource [CoreSpring/src/main/resources/Test.xml] cannot be opened because it does not exist
at org.springframework.beans.factory.xml.XmlBeanDefinitionReader.loadBeanDefinitions(XmlBeanDefinitionReader.java:341).
I placed Bean XML into /CoreSpring/src/main/resources/Test.xml path But still getting same error.But when i tried without Maven it working good.Please help me.
Class Code :
public class InjectSimple {
private String name;
private int age;
private int height;
private float ageInSeconds;
private boolean programmer;
public static void main(String a[])
{
ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext(
"/CoreSpring/src/main/resources/Test.xml");
InjectSimple simple=(InjectSimple) context.getBean("sample");
System.out.println(simple);
}
//Setter and Getter Methods.
Reply

Pankaj says
MARCH 20, 2014 AT 6:27 PM
src/main/resources are already defined as source path, so you need to specify only the file name "Test.xml". For confirmation, export as WAR and then unzip it to see the file location.
Reply

Leave a Reply

Your email address will not be published. Required fields are marked *
Comment

Name *

Email *

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

Search for tutorials...

DOWNLOAD ANDROID APP

SPRING FRAMEWORK

Spring Tutorial

Spring Core

- Spring Framework
- Spring Dependency Injection
- Spring IoC and Bean
- Spring Bean Life Cycle
- Spring REST
- Spring REST XML
- Spring RestTemplate
- Spring AOP
- Spring AOP Method Profiling
- Spring Annotations
- Spring @Autowired
- Spring @RequestMapping

Spring MVC

Miscellaneous

- Spring Data MongoDB
- Spring Interview Questions

RECOMMENDED TUTORIALS

Java Tutorials

- Java IO
- Java Regular Expressions
- Multithreading in Java
- Java Logging
- Java Annotations
- Java XML
- Collections in Java
- Java Generics
- Exception Handling in Java
- Java Reflection
- Java Design Patterns
- JDBC Tutorial

Java EE Tutorials

- Servlet JSP Tutorial
- Struts2 Tutorial
- Spring Tutorial
- Hibernate Tutorial
- Primefaces Tutorial
- Apache Axis 2
- JAX-RS
- Memcached Tutorial