

Industrial strength Java/JEE Career Companion to open more doors

search here ...

Go

[Home](#)[Java FAQs](#)[600+ Java Q&As](#)[Career](#)[Tutorials](#)[Member](#)[Why?](#)[Can u Debug?](#)[Java 8 ready?](#)[Top X](#)[Productivity Tools](#)[Judging Experience?](#)[Home](#) › [Interview](#) › [Core Java Interview Q&A](#) › [Differences Between X & Y](#) ›

Differences between Spring XML and Spring JavaConfig (@Configuration)

Differences between Spring XML and Spring JavaConfig (@Configuration)

Posted on [May 30, 2015](#) by [Arulkumaran Kumaraswamipillai](#)

This is a good candidate to make it to the job interview discussions along the debate of checked Vs unchecked exceptions as the industry is split on this very topic. Some favor XML, whilst others love using the JavaConfig.

Q1. What are the Pros and Cons of Spring JavaConfig Vs. Spring XML?

A1.

Pros for XML:

1) Configuration is centralized in a fewer XML files, and not scattered among all different components @Inject,

600+ Full Stack Java/JEE Interview Q&As ♥Free ♦FAQs

[open all](#) | [close all](#)[+ Ice Breaker Interview](#)[+ Core Java Interview C](#)[+ Java Overview \(4\)](#)[+ Data types \(6\)](#)[+ constructors-methc](#)[+ Reserved Key Wor](#)[+ Classes \(3\)](#)[+ Objects \(8\)](#)[+ OOP \(10\)](#)[+ GC \(2\)](#)[+ Generics \(5\)](#)[+ FP \(8\)](#)[+ IO \(7\)](#)[+ Multithreading \(12\)](#)[+ Algorithms \(5\)](#)[+ Annotations \(2\)](#)[+ Collection and Dat](#)[+ Differences Between](#)[♥ Java Iterable \](#)[♦ Multithreading](#)[♦ Why do Proxy,](#)

@Component, @Configuration, @Repository, etc annotations. So with XML you can have a nice overview of beans and their wirings in a single place.

2) The XML is completely independent to the Java artifacts as there's no coupling between the 2. This enables the class to be used in more than one context with different configurations.

Cons for XML:

1) XML is verbose, so configuration files tend to get big. It's a good thing to split them up.

Pros for JavaConfig:

1) JavaConfig can be seen as the XML file equivalent, written in Java to give the power of compile-time checks, code completion, support for finding references in the workspace and refactoring via the IDE support. The typos generate compile-error, so no more waiting till load time to find XML typos.

Cons for JavaConfig:

1) If not done properly can create annotation mess and confusion.

Q2. For the given code snippet created using the following steps, write how will you go about wiring them up **a)** XML way **b)** JavaConfig way:

Created a MVN project

```
1 mvn archetype:generate -DgroupId=com.mytutorial -
```

pom.xml

```
1 <project xmlns="http://maven.apache.org/POM/4.0.0"
2   xsi:schemaLocation="http://maven.apache.org/
```

- [Core Java Modif](#)
- [Differences betw](#)
- [Java Collection i](#)
- [Event Driven Progr](#)
- [Exceptions \(2\)](#)
- [Java 7 \(2\)](#)
- [Java 8 \(24\)](#)
- [JVM \(6\)](#)
- [Reactive Programn](#)
- [Swing & AWT \(2\)](#)
- [JEE Interview Q&A \(3](#)
- [Pressed for time? Jav](#)
- [SQL, XML, UML, JSC](#)
- [Hadoop & BigData Int](#)
- [Java Architecture Inte](#)
- [Scala Interview Q&As](#)
- [Spring, Hibernate, & I](#)
- [Testing & Profiling/Sa](#)
- [Other Interview Q&A 1](#)
- [Free Java Interview](#)

16 Technical Key Areas

[open all](#) | [close all](#)

- [Best Practice \(6\)](#)
- [Coding \(26\)](#)
- [Concurrency \(6\)](#)
- [Design Concepts \(7\)](#)
- [Design Patterns \(11\)](#)
- [Exception Handling \(3\)](#)
- [Java Debugging \(21\)](#)
- [Judging Experience I](#)
- [Low Latency \(7\)](#)
- [Memory Managemen](#)
- [Performance \(13\)](#)
- [QoS \(8\)](#)
- [Scalability \(4\)](#)
- [SDLC \(6\)](#)
- [Security \(13\)](#)

```

3      <modelVersion>4.0.0</modelVersion>
4
5      <groupId>com.mytutorial</groupId>
6      <artifactId>simple-spring</artifactId>
7      <version>1.0-SNAPSHOT</version>
8      <packaging>jar</packaging>
9
10     <name>simple-spring</name>
11     <url>http://maven.apache.org</url>
12
13     <properties>
14         <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
15     </properties>
16
17     <dependencies>
18         <!-- Spring Framework -->
19         <dependency>
20             <groupId>org.springframework</groupId>
21             <artifactId>spring-aspects</artifactId>
22             <version>3.2.13.RELEASE</version>
23         </dependency>
24         <dependency>
25             <groupId>org.springframework</groupId>
26             <artifactId>spring-context-support</artifactId>
27             <version>3.2.13.RELEASE</version>
28         </dependency>
29     </dependencies>
30 </project>

```

[Transaction Manager](#)

80+ step by step Java Tutorials

[open all](#) | [close all](#)

- [Setting up Tutorial \(6\)](#)
- [Tutorial - Diagnosis \(2\)](#)
- [Akka Tutorial \(9\)](#)
- [Core Java Tutorials \(2\)](#)
- [Hadoop & Spark Tuto](#)
- [JEE Tutorials \(19\)](#)
- [Scala Tutorials \(1\)](#)
- [Spring & Hibernate Ti](#)
- [Tools Tutorials \(19\)](#)
- [Other Tutorials \(45\)](#)

Class with main method

```

1 package com.mytutorial;
2
3 import org.springframework.context.ApplicationContext;
4 import org.springframework.context.ConfigurableApplicationContext;
5 import org.springframework.context.support.ClassPathXmlApplicationContext;
6
7 public class App
8 {
9     //injected via XML
10    private SimpleService myService;
11
12    public static void main( String[] args )
13    {
14        ApplicationContext ctx = new ClassPathXmlApplicationContext("springApp.xml");
15        App app = (App)ctx.getBean("springApp");
16        app.executeApp();
17        ((ConfigurableApplicationContext) ctx).close();
18    }
19
20    public void executeApp() {
21        System.out.println("Executing App...");
22        myService.executeService();
23    }
24
25    //invoked by Spring XML setter
26    public void setMyService(SimpleService myService) {
27        this.myService = myService;
28    }
29 }

```

100+ Java pre-interview coding tests

[open all](#) | [close all](#)

- [Can you write code? \(1\)](#)
- [Complete the given code \(1\)](#)
- [Converting from A to B \(1\)](#)
- [Designing your class \(1\)](#)
- [Java Data Structures \(1\)](#)
- [Passing the unit tests \(1\)](#)
- [What is wrong with this code? \(1\)](#)
- [Writing Code Home Assignment \(1\)](#)
- [Written Test Core Java \(1\)](#)
- [Written Test JEE \(1\)](#)

How good are your?

Service layer

[open all](#) | [close all](#)[Career Making Know-](#)[Job Hunting & Resur](#)

```
1 package com.mytutorial;
2
3 public interface SimpleService {
4     void executeService();
5 }
```

```
1 package com.mytutorial;
2
3 public class SimpleServiceImpl implements SimpleService {
4     private SimpleDao myDao;
5
6     public void executeService() {
7         System.out.println("Running simple service");
8         myDao.executeDao();
9     }
10
11     //gets called by the Spring XML to set Dao
12     public void setMyDao(SimpleDao dao) {
13         this.myDao = dao;
14     }
15 }
16 }
```

DAO layer

```
1 package com.mytutorial;
2
3 public interface SimpleDao {
4     void executeDao();
5 }
```

```
1 package com.mytutorial;
2
3 public class SimpleDaoImpl implements SimpleDao {
4
5     public void executeDao() {
6         System.out.println("Running simple dao impl");
7     }
8
9 }
```

A2. The XML config gives you a big picture of what is injected where. dao → service → springApp.

XML Configuration

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/sch
```

```

3      xmlns:context="http://www.springframework.or
4      xsi:schemaLocation="http://www.springframewo
5          http://www.springframework.org/schema/be
6          http://www.springframework.org/schema/co
7          http://www.springframework.org/schema/co
8
9      <bean id="springApp" class="com.mytutorial.A
10         <property name="myService" ref="service"
11     </bean>
12
13     <bean id="service" class="com.mytutorial.Sim
14         <property name="myDao" ref="dao" />
15     </bean>
16
17     <bean id="dao" class="com.mytutorial.SimpleD
18
19 </beans>

```

JavaConfig

```

1  package com.mytutorial;
2
3  import org.springframework.context.annotation.Be
4  import org.springframework.context.annotation.Co
5
6  @Configuration
7  public class SpringConfig {
8
9      @Bean
10     public SimpleDao myDao() {
11         return new SimpleDaoImpl();
12     }
13
14     @Bean
15     public SimpleService myService() {
16         SimpleServiceImpl simpleService = new Si
17         return simpleService;
18     }
19
20     @Bean
21     public App app() {
22         App app = new App();
23         return app;
24     }
25 }
26 }
27

```

The App.java will look like with **@Autowired** and **AnnotationConfigApplicationContext**

```

1  package com.mytutorial;
2
3  import org.springframework.beans.factory.annotat
4  import org.springframework.context.ConfigurableA
5  import org.springframework.context.annotation.An
6
7  public class App {

```

```

8
9     @Autowired
10    private SimpleService myService; //myService
11
12    public static void main(String[] args) {
13        AnnotationConfigApplicationContext ctx =
14        App app = (App) ctx.getBean(App.class);
15        app.executeApp();
16        ((ConfigurableApplicationContext) ctx).c
17    }
18
19    public void executeApp() {
20        System.out.println("Executing App...");
21        myService.executeService();
22    }
23
24    public void setMyService(SimpleService mySer
25        this.myService = myService;
26    }
27 }
28
29

```

The **SimpleServiceImpl** requires the **@Autowired**

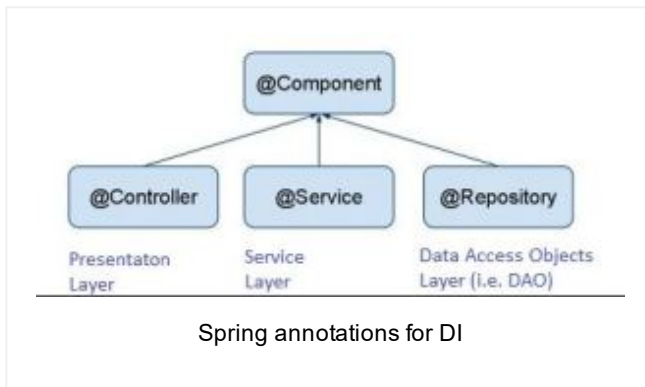
```

1 package com.mytutorial;
2
3 import org.springframework.beans.factory.annotat
4
5 public class SimpleServiceImpl implements Simple
6
7     @Autowired
8     private SimpleDao myDao; // myDao() is invok
9
10    public void executeService() {
11        System.out.println("Running simple servic
12        myDao.executeDao();
13    }
14
15    //gets called by the Spring XML to set Dao
16    public void setMyDao(SimpleDao dao) {
17        this.myDao = dao;
18    }
19 }

```

Use of auto wiring is not a best practice for very large projects. So, lets use the new Java **@Inject** annotation.

**JavaConfig and annotations like
@Inject, @component,
@Repository, @Service, etc**



Add new Java dependency in the pom.xml file to be able to use **@Inject** annotation.

```

1 <dependency>
2     <groupId>javax.inject</groupId>
3     <artifactId>javax.inject</artifactId>
4     <version>1</version>
5 </dependency>
6

```

The **SpringConfig** only having the app(), most importantly “**@ComponentScan**(basePackages=“com.mytutorial”) ” to be able to scan for classes with the spring annotations like **@Component** in the base package “com.mytutorial”.

App.java with **@Inject** and **@Component** annotations

```

1 package com.mytutorial;
2
3 import javax.inject.Inject;
4
5 import org.springframework.context.ConfigurableApplicationContext;
6 import org.springframework.context.annotation.AnnotationConfigApplicationContext;
7 import org.springframework.stereotype.Component;
8
9 @Component
10 public class App {
11
12     @Inject
13     private SimpleService myService;
14
15     public static void main(String[] args) {
16         AnnotationConfigApplicationContext ctx =
17         App app = (App) ctx.getBean(App.class);
18         app.executeApp();
19         ((ConfigurableApplicationContext) ctx).close();
20     }
21
22     public void executeApp() {
23         System.out.println("Executing App...");
24         myService.executeService();
25     }
26 }

```

```
25     }
26
27     // invoked by Spring XML setter
28     public void setMyService(SimpleService mySer
29         this.myService = myService;
30     }
31 }
32
33
```

SimpleServiceImpl.java with @Inject and @Service annotations

```
1 package com.mytutorial;
2
3 import javax.inject.Inject;
4
5 import org.springframework.stereotype.Service;
6
7 @Service
8 public class SimpleServiceImpl implements Simple
9
10     @Inject
11     private SimpleDao myDao;
12
13     public void executeService() {
14         System.out.println("Running simple servic
15         myDao.executeDao();
16     }
17
18     //gets called by the Spring XML to set Dao
19     public void setMyDao(SimpleDao dao) {
20         this.myDao = dao;
21     }
22 }
```

SimpleDaoImpl.java with the @Repository annotation

```
1 package com.mytutorial;
2
3 import org.springframework.stereotype.Repository
4
5 @Repository
6 public class SimpleDaoImpl implements SimpleDao
7
8     public void executeDao() {
9         System.out.println("Running simple dao im
10     }
11
12 }
```

XML and annotations like @Inject, @component, @Repository, @Service, etc


```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/sch
3     xmlns:context="http://www.springframework.or
4     xsi:schemaLocation="http://www.springframewo
5         http://www.springframework.org/schema/be
6         http://www.springframework.org/schema/co
7         http://www.springframework.org/schema/co
8
9     <context:component-scan base-package="com.my
10
11 </beans>
```

App.java

```
1 package com.mytutorial;
2
3 import javax.inject.Inject;
4
5 import org.springframework.context.ApplicationCo
6 import org.springframework.context.ConfigurableA
7 import org.springframework.context.support.Class
8 import org.springframework.stereotype.Component;
9
10 @Component
11 public class App {
12
13     @Inject
14     private SimpleService myService;
15
16     public static void main(String[] args) {
17         ApplicationContext ctx = new ClassPathXm
18         App app = (App) ctx.getBean(App.class);
19         app.executeApp();
20         ((ConfigurableApplicationContext) ctx).c
21     }
22
23     public void executeApp() {
24         System.out.println("Executing App...");
25         myService.executeService();
26     }
27
28     // invoked by Spring XML setter
29     public void setMyService(SimpleService mySer
30         this.myService = myService;
31     }
32 }
```

all the other implementation classes same as before.

Popular Posts

♦ [11 Spring boot interview questions & answers](#)

824 views

♦ [Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers](#)

765 views

18 Java scenarios based interview Questions and Answers

399 views

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

388 views

01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

295 views

♦ 7 Java debugging interview questions & answers

293 views

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

285 views

♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

279 views

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

239 views

001B: ♦ Java architecture & design concepts interview questions & answers

201 views

Bio

Latest Posts



Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via [Amazon.com](https://www.amazon.com) in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site. **945+** paid members. [join my LinkedIn Group](#). [Reviews](#)



**About Arulkumaran Kumaraswamipillai**

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via [Amazon.com](https://www.amazon.com) in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site. **945+** paid members. [join my LinkedIn Group](#). **Reviews**

◀ Part #3: JPA Hibernate Spring Maven Eclipse Tutorial – testing data access with HSQLDB

♥ Top 6 Spring wiring via JavaConfig [i.e. @Configuration] examples ▶

Posted in Differences Between X & Y, member-paid

Empowers you to open more doors, and fast-track

Technical Know Hows

☀ [Java generics in no time](#) ☀ [Top 6 tips to transforming your thinking from OOP to FP](#) ☀ [How does a HashMap internally work? What is a hashing function?](#)
☀ [10+ Java String class interview Q&As](#) ☀ [Java auto un/boxing benefits & caveats](#) ☀ [Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect](#)

Non-Technical Know Hows

☀ [6 Aspects that can motivate you to fast-track your career & go places](#) ☀ [Are you reinventing yourself as a Java developer?](#) ☀ [8 tips to safeguard your Java career against offshoring](#) ☀ [My top 5 career mistakes](#)

Prepare to succeed

☀ [Turn readers of your Java CV go from “Blah blah” to “Wow”?](#) ☀ [How to prepare for Java job interviews?](#) ☀ [16 Technical Key Areas](#) ☀ [How to choose from multiple Java job offers?](#)

Select Category ▼

© Disclaimer

The contents in this Java-Success are copy righted. The author has the right to correct or enhance the current content without any prior notice.

These are general advice only, and one needs to take his/her own circumstances into consideration. The author will not be held liable for any damages caused or alleged to be caused either directly or indirectly by these materials and resources. Any trademarked names or labels used in this blog remain the property of their respective trademark owners. No guarantees are made regarding the accuracy or usefulness of content, though I do make an effort to be accurate. Links to external sites do not imply endorsement of the linked-to sites.