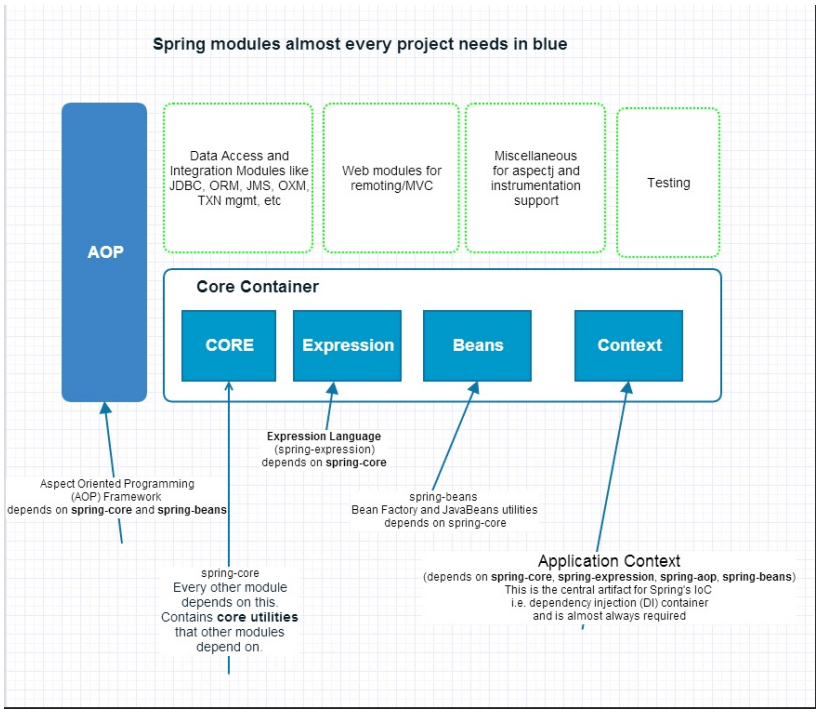
01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

Posted on July 27, 2016 by Arulkumaran Kumaraswamipillai

This extends 13 Spring basics Q1 – Q7 interview questions & answers.

Q8. Can you describe the high level Spring architecture?
A8. A Spring Bean represents a POJO (Plain Old Java Object) performing useful operation(s). All Spring Beans reside within a Spring IoC Container. The Spring Framework hides most of the complex infrastructure and the communication that happens between the Spring Container and the Spring Beans. The Core Container is shown below.



Spring Architecture

Spring framework architecture is modular with layers like core, data access & integration, web/remoting, and other miscellaneous support.

Q9. What are the packages (i.e. jar files) required in your project to get started with a Spring application?
A9. In order to get started with Spring, your maven pom.xml file should at least have the following **core Spring packages**:

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>${spring.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-expression</artifactId>
    <version>${spring.version}</version>
</dependency>
<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-beans</artifactId>
   <version>${spring.version}</version>
</dependency>
<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-aop</artifactId>
   <version>${spring.version}</version>
</dependency>
```

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <version>${spring.version}</version>
</dependency>

```

The following **additional packages** can be added based on the requirements:

spring-context-support package is required for integration of EhCache, JavaMail, Quartz, and Freemarker. So, if you are going to use Java Mail to send emails or Quartz to schedule a job, then you need spring-context-support.

spring-tx package is required for transaction management support, and it depends on spring-core, spring-beans, spring-aop, and spring-context packages.

spring-jdbc and **spring-orm** are required for the database access. spring-jdbc depends on spring-core, spring-beans, spring-context, and spring-tx. if using Object-to-Relation-Mapping (ORM) integration with Hibernate, JPA or iBatis then you need spring-orm, which depends on spring-core, spring-beans, spring-context, and spring-tx.

spring-oxm is required for JAXB, JiBX,XStream, XMLBeans or any other Object-To-Xml(OXM) mapping. You need spring-oxm, which depends on spring-core, spring-beans, and spring-context.

Similarly,

spring-test is required for junit testing.

spring-web is required if you want to use a web framework like Spring MVC, JSF, Struts, etc, and depends on depends on spring-core, spring-beans, and spring-context.

spring-webmvc is required to use Spring as the MVC framework for Web application or RESTFul web service. It depends on spring-core, spring-beans, spring-context, and spring-web.

spring-mock containing mock classes to assist with the testing.

spring-jms for messaging and depends on spring-core and spring-oxm (i.e. for OXM).

Q10. Can you describe the bean life cycle?
A10. A Spring Bean represents a POJO (Plain Old Java Object) performing useful operation(s). All Spring Beans reside within a Spring IoC Container. The Spring Framework hides most of the complex infrastructure and the communication that happens between the Spring Container and the Spring Beans.

Spring Bean life cycle means the construction and destruction of the beans and usually this is in relation to the construction and destruction of the Spring Context. Spring has three ways of calling your code during initialization and shut down.

1. Programmatically, usually called 'interface callbacks': Spring calls your bean during the setup and tear down of the Spring Context, and your bean needs to implement **InitializingBean** or **DisposableBean**. Spring 3.0 has the Lifecycle interface with start/stop lifecycle control methods. The typical use case for this is to control asynchronous processing.

2. Declarative 'method callbacks' on a per bean basis: You use a method callback by adding a method to your bean, which you then reference in your XML config. When Spring reads the config it figures out that there's a bean of type "A" with a method that it needs to call on startup and another it needs to call on shutdown.

3. Declarative 'method callbacks' to all beans.

Initialization:

Step 1: The spring container finds the bean's definition from the **XML file** or annotations (like **@Configuration**) and instantiates the bean.

Step 2: Using the dependency injection, spring populates all of the bean properties as specified in the bean definition.

Step 3: If the bean implements the **BeanNameAware** interface, the factory calls “setBeanName()” passing the bean’s ID.

Step 4: If the bean implements the **BeanFactoryAware** interface, the factory calls “setBeanFactory()”, passing an instance of itself.

Step 5: If the bean implements the **ApplicationContextFactoryAware** interface, the container calls “bean.setApplicationContext(container)”.

Step 6: If there are any **BeanPostProcessors** associated with the bean, their “postProcessBeforeInitialization()” method will be called.

Step 7a: If the bean implements **InitializingBean** interface, “bean.afterPropertiesSet()” method will be invoked.

Step 7b: If the bean declares custom init method, the container calls custom init method of that bean

```
1 <bean id="myAppBean" class="com.myapp.MyAppBeanIm
```

Step 8: If there are any **BeanPostProcessors** associated with the bean, their “postProcessAfterInitialization()” method will be called.

Step 9: Bean is now ready for use.

shutdown:

Step 1: If the bean implements **DisposableBean** interface, container calls the “bean.destroy()”.

Step 2: If the bean declares custom destroy method, container calls custom destroy method of bean.

Bean Life Cycle Example

Step 1: myApp-applicationContext.xml file

```
1 <bean id="myAppBean" class="com.myapp.MyAppBeanIm
```

Step 2: MyAppBeanImpl bean with business logic interface MyAppBean, and life cycle interfaces BeanNameAware and BeanFactoryAware

```
1  package com.myapp;
2
3  public class MyAppBeanImpl implements MyAppBean,
4
5      @Override
6      public String sayHello() {
7          return "Hello, I am initialized";
8      }
9
10     @Override
11     public void setBeanFactory(BeanFactory beanF
12         System.out.println("received the beanFac
13
14     }
15
16     @Override
17     public void setBeanName(String name) {
18         System.out.println("the name of the bean
19
20     }
21 }
22
```

Step 3: To test the bean initialization, the cogif file “myApp-applicationContext.xml” is read via XmlBeanFactory class.

```
1 public static void main(String[] args) {
2     final XmlBeanFactory beanFactory = new XmlBea
3             new ClassPathResource("myApp-applicat
4     MyAppBean myApp = (MyAppBean) beanFactory.get
5     System.out.println(myApp.sayHello());
6 }
7
```

Step 4:: The output of the code is:

the name of the bean is myAppBean
received the beanFactory
org.springframework.beans.factory.xml.XmlBeanFactory@f6f

542:
defining beans [myAppBean]; root of factory hierarchy
Hello, I am initialized

Q11. What is a BeanFactory?
A11. The BeanFactory is the actual container which instantiates, configures, and manages a number of beans. These beans typically collaborate with one another, and thus have dependencies between themselves.

Q12. How do you bootstrap the initial bean?
A12. Beans are wired up inside Spring XML file or via annotations like @Component, @Resource, etc. The initial bean needs to be bootstrapped, and there are a number of approaches as shown below.

1. Using the "**ClassPathXmlApplicationContext**" class in Spring

```
1  import org.springframework.beans.factory.BeanFac
2  import org.springframework.context.support.Class
3
4  public class TestSpring {
5
6      public static void main(String[] args) {
7          ClassPathXmlApplicationContext appContex
8                  new String[] {"myApp-application
9          BeanFactory factory = (BeanFactory) appCo
10         MyAppBean myApp = (MyAppBean) factory.get
11         System.out.println(myApp.sayHello());
12     }
13 }
14
```

2. Using the "**FileSystemResource**" class in Spring

```
1  import org.springframework.beans.factory.xml.Xml
2  import org.springframework.core.io.FileSystemRes
3  import org.springframework.core.io.Resource;
4
5  public class TestSpring {
6
7      public static void main(String[] args) {
8          Resource res = new FileSystemResource("bi
9          XmlBeanFactory factory = new XmlBeanFacto
10         MyAppBean myApp = (MyAppBean) factory.get
11         System.out.println(myApp.sayHello());
12     }
13 }
```

```
14
```

3. Using the "**ClassPathResource**"

```
1  import org.springframework.beans.factory.xml.Xml
2  import org.springframework.core.io.ClassPathReso
3
4  public class TestSpring {
5
6      public static void main(String[] args) {
7         ClassPathResource res = new ClassPathReso
8         XmlBeanFactory factory = new XmlBeanFacto
9         MyAppBean myApp = (MyAppBean) factory.get
10        System.out.println(myApp.sayHello());
11     }
12 }
13
```

4. For @Configuration annotation driven configurations use **AnnotationConfigApplicationContext**

```
1 @Configuration
2 public class AppConfig {
3      @Bean
4      public MyAppBean myBean() {
5          // instantiate, configure and return bea
6      }
7 }
8
```

```
1  public class TestSpring {
2
3      public static void main(String[] args) {
4         AnnotationConfigApplicationContext ctx =
5         ctx.register(AppConfig.class);
6         ctx.refresh();
7         MyAppBean myApp = ctx.getBean(MyAppBean.c
8         System.out.println(myApp.sayHello());
9      }
10 }
11
```

5. From a Web application. As opposed to the BeanFactory, which will often be created programmatically, ApplicationContexts can be created declaratively using a ContextLoader. You can register an **ApplicationContext** using the **ContextLoaderListener** as shown below in the **web.xml** file. The Spring context listener provides more flexibility in terms of how an application is wired together. It uses the application's Spring configuration to determine what

object to instantiate and loads the objects into the application context used by the servlet container.

```
<web-app>
  .....
  <listener>
     <listener-class>org.springframework.web.cont
  </listener>
  ....
</web-app>
```

By default, it looks for a file named **applicationContext.xml** file in WEB-INF folder. But, you can configure the org.springframework.web.context.ContextLoaderListener class to use a context parameter called contextConfigLocation to determine the location of the Spring configuration file. The context parameter is configured using the context-parameter element. The context-param element has two children that specify parameters and their values. The param-name element specifies the parameter's name. The param-value element specifies the parameter's value.

```
<web-app>
  ...
  <context-param>
    <param-name>contextConfigLocation</param-nam
    <param-value>WEB-INF/myApp-applicationContex
  </context-param>

  <listener>
    <listener-class>org.springframework.web.cont
  </listener>
  ...
</web-app>
```

Q13. What would you do, if it's not practical (or impossible) to wire up your entire application into the Spring framework, but you still need a Spring loaded bean in order to perform a task?

A13. For example,

— an auto generated web service client class! But you do want to use the dependency injection feature of Spring to get some of the other beans injected in to this class.

— A legacy code that needs to make use of a Spring bean.

The ApplicationContextAware interface provided by Spring allows you to wire some Java classes which are unable (or you don't want it) to be wired to the Spring application context.

STEP 1: The **ApplicationContextAware** interface makes sense when an object requires access to a set of collaborating beans.

```
1  import org.springframework.beans.BeansException;
2  import org.springframework.context.ApplicationCo
3  import org.springframework.context.ApplicationCo
4
5  public class MyServiceFactory implements Applica
6
7      private ApplicationContext context;
8
9      public void testMethod2(){
10         System.out.println("Test method2 invoked
11     }
12
13     @Override
14     public void setApplicationContext(Applicatio
15             throws BeansException {
16        System.out.println("setting application c
17        this.context = ctx;
18     }
19
20
21     public MyBeanService getInstance(String acce
22         //.....some logic
23         MyBeanService beanService = (MyBeanServi
24         return beanService;
25     }
26 }
27
```

STEP 2: The **beans.xml** file. The **MyServiceFactory** is not wired up.

```
1  <bean id="myBeanDao" class="test.MyBeanDaoImpl"/
2
3  <bean id="myBeanService" class="test.MyBeanServi
4      <!-- setter injection of dao into service --
5      <property name="beanDao" ref="myBeanDao" />
6  </bean>
7
8  <!-- No DI wiring -->
9  <bean id="myServiceFactory" class="test.MyServic
10
11
```

Step 3: Finally, the **bootstrapping** code that makes use of the MyServiceFactory class.

```
1  import org.springframework.beans.factory.BeanFac
2  import org.springframework.context.support.Class
3
4  public class TestSpring2 {
5
6      public static void main(String[] args) {
7          ClassPathXmlApplicationContext appContex
8                  new String[] {"beans.xml"});
9          BeanFactory factory = (BeanFactory) appC
10         MyServiceFactory servicefactory = (MySer
11         MyBeanService service = servicefactory.g
12     }
13 }
14
```

Top 20+ Spring Interview Questions & Answers:

1. 9 Spring Bean Scopes Interview Q&As

2. 17 Spring FAQ interview Questions & Answers

3. 30+ Hibernate interview questions & answers

Popular Posts

♦ 11 Spring boot interview questions & answers

826 views

♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers

767 views

18 Java scenarios based interview Questions and Answers

400 views

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

389 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

240 views

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

202 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 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 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**

‹ 02: Q6 – Q12 Scala FP interview questions & answers

01b: ♦ 15+ Hibernate basics Q8 – Q15 interview questions & answers

›

Posted in member-paid, Spring, Spring Job Interview Essentials

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