04 ♦ 17 Spring bean life cycles & scopes FAQ interview Questions & Answers

Posted on August 26, 2014 by Arulkumaran Kumaraswamipillai

Q1. Does Spring dependency injection happen during compile time or runtime?
A1. Runtime during creating an object.

Q2. What is the difference between prototype scope and singleton scope? Which one is the default?
A2.Singleton means single bean instance per IoC container, and prototype means any number of object instances per IoC container. The default scope is “singleton”.

Q3. When will you use singleton scope? When will you use prototype scope?
A3. **Singleton** scope is used for stateless object use. For

example, injecting a DAO (i.e. Data Access Object) into a business service object. DAOs don't need to maintain conversational states.

Single instance of "myDaoDef" and "myServiceDef" are created

```
<beans xmlns="http://www.springframework.org/sch
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    xsi:schemaLocation="http://www.springframewo

    <bean id="myDaoDef" class="com.mycompany.und

    <bean id="myServiceDef" class="com.mycompany
       <constructor-arg name="myDao" ref="myDaoD
    </bean>
</beans>

```

Q. Is a singleton bean thread-safe?
A. No. Multiple threads access a single instance. It needs to be coded in a thread-safe manner.

Prototype is useful when your objects maintain state in a multi-threaded environment. Each thread needs to use its own object and cannot share the single object. For example, you might hava a RESTFul web service client making multi-threaded calls to Web services. The REST easy client APIs like RESTEasy uses the Apache Connection manager which is not thread safe and each thread should use its own client. Hence, you need to use the prototype scope.

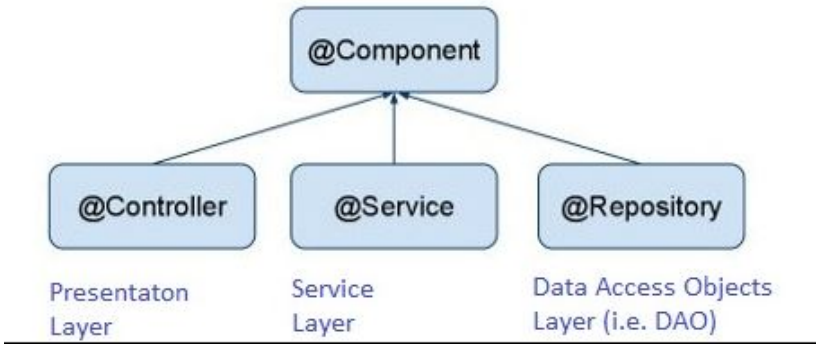
Multiple instances of "myDaoDef" and "myServiceDef" are created

```
<beans xmlns="http://www.springframework.org/sch
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    xsi:schemaLocation="http://www.springframewo

    <bean id="myDaoDef" class="com.mycompany.und

    <bean id="myServiceDef" class="com.mycompany
       <constructor-arg name="myDao" ref="myDaoD
    </bean>

</beans>

```

Q4. How do you provide configuration metadata or wire up dependencies in the Spring Container?
A4. There are three important methods to provide configuration metadata to the Spring Container:

1) XML based configuration file. This was shown in the above XML file with “myDaoDef” and “myServiceDef”
2) Annotation-based configuration.
3) Java-based configuration.

2) Annotation-based configuration



Spring annotations

The *MyDaoImpl*:

```
1 package com.mycompany.understanding.spring;
2
3 @Repository(value = "myDaoDef")
4 @Scope("singleton")
5 public class MyDaoImpl implements MyDao
6 {
7       //...
8 }
9
```

The *MyServiceImpl*:

```
1  package com.mycompany.understanding.spring;
2
3  @Service(value = "myServiceDef")
4  @Scope("singleton")
5  @Transactional(propagation = Propagation.SUPPORT
6  public class MyServiceImpl implements MyService
7  {
8
```

```
    @Resource(name = "myDaoDef")
    private MyAppDao myAppDao;

    //.....

}

```

The annotations shown above allow you to declare beans that are to be picked up by autoscanning with or **@ComponentScan**.

3) Java-based configuration.

@Configuration annotation was designed as the replacement for XML configuration files.

```
@Configuration
@ComponentScan(
        basePackages ={"com.mycompany"},
        useDefaultFilters = false,
        includeFilters =
        {
            @ComponentScan.Filter(type = FilterT
            @ComponentScan.Filter(type = FilterT
        })
public class AppConfig {

    @Bean(name="anotherMyAppBean")
    public HelloWorldService anotherMyAppService
        return new AnotherMyAppServiceImpl();
    }
}

```

@Bean annotation tells that this bean to be managed by the Spring IoC container.

Q5. Would both singleton and prototype bean's life cycle be managed by the Spring IoC container?
A5. Yes and no. The singleton bean's complete life cycle will be managed by Spring IoC container, but with regards to prototype scope, IoC container only partially manages the life cycle – instantiates, configures, decorates and otherwise assembles a prototype object, hands it to the client and then has no further knowledge of that prototype instance. As per the spring documentation

"This means that while initialization lifecycle callback methods will be called on all objects regardless of scope, in the case of prototypes, any configured destruction lifecycle callbacks will not be called. It is the responsibility of the client code to clean up prototype scoped objects and release any expensive resources that the prototype bean(s) are holding onto."

Q6. What happens if you inject a prototype scoped bean into a singleton scoped bean?
A6. A new prototype scoped bean will be injected into a singleton scoped bean once at runtime, and the same prototype bean will be used by the singleton bean.

Q7. What if you want the singleton scoped bean to be able to acquire a brand new instance of the prototype-scoped bean again and again at runtime?
A7. In this use-case, there is no use in just dependency injecting a prototype-scoped bean into your singleton bean because as stated above, this only happens once when the Spring container is instantiating the singleton bean and resolving and injecting its dependencies. You can just inject a singleton (e.g. a factory) bean and then use Java class to instantiate (e.g with a newInstance(…) or create(..) method) a new bean again and again at runtime without relying on Spring or alternatively have a look at Spring's "method injection". As per Spring documentation for "Lookup method injection".

"Lookup method injection refers to the ability of the container to override methods on container managed beans, to return the result of looking up another named bean in the container. The lookup will typically be of a prototype bean as in the scenario described above. The Spring Framework implements this method injection by dynamically generating a subclass overriding the method, using bytecode generation via the CGLIB library."

Lookup method to inject new "myDaoDef" prototype bean into singleton "myServiceDef" bean

```
<beans xmlns="http://www.springframework.org/sch
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    xsi:schemaLocation="http://www.springframewo

    <bean id="myDaoDef" class="com.mycompany.und

    <bean id="myServiceDef" class="com.mycompany
       <lookup-method name="createMyDao" bean="m
    </bean>
</beans>

```

Note that the class is abstract as **Spring will decorate this class with cgilib**.

```
package com.mycompany.understanding.spring;

public abstract class MyServiceImpl implements M

 protected abstract MyDao createMyDao();

 @Override
 public void performTask() {
  System.out.println("Performing tasks .........
  createMyDao().printData();
 }
}

```

```
package com.mycompany.understanding.spring;

public class MyDaoImpl implements MyDao {

 @Override
 public void printData() {
  System.out.println("printing data");
  System.out.println(this);
 }
}

```

Q8. What are the scopes defined in HTTP context?
A8. Following scopes are only valid in the context of a web-aware Spring ApplicationContext.

– **request scope** is for a single bean definition to the lifecycle of a single HTTP request.In other words each and every HTTP request will have its own instance of a bean created off the back of a single bean definition.
– **session scope** is for a single bean definition to the lifecycle of a HTTP Session.
– **global session scope** is for a single bean definition to the

lifecycle of a global HTTP Session. Typically only valid when used in a portlet context.

Q9. Does Spring allow you to define your own bean scopes?
A9. Yes, from Spring 2.0 onwards you can define custom scopes. For example,

– You can define a ThreadOrRequest and ThreadOrSession scopes to be able to switch between the environment you run in like JUnit for testing and Servlet container for running as a Web application.
– You can write a custom scope to inject stateful objects into singleton services or factories.
– You can write a custom bean scope that would create new instances per each JMS message consumed
– Oracle Coherence has implemented a datagrid scope for Spring beans. You will find many others like this.

Q10. Spring promotes the “Open/Closed principle (OCP)”. What is your understanding of this principle?
A10. According to GoF design pattern authors “software entities (classes, modules, functions, etc.) should be **open for extension**, but **closed for modification**“. Wherever you have large if/else statements, you need to think if OCP can be applied. Instead of modifying the existing large if/else loops, you need to add new classes to extend behavior.

Bad: Not closed for modification. To support “division” or “subtraction” in the future, this class has to be modified with more “else if”s

```
1  import javax.management.RuntimeErrorException;
2  import org.apache.commons.lang.StringUtils;
3
4  public class MathOperation {
5
6   public int operate(int input1, int input2, Stri
7
8    if(StringUtils.isEmpty(operator)){
9     throw new IllegalArgumentException("Invalid o
10   }
11
12   if(operator.equalsIgnoreCase("+")){
13    return input1 + input2;
14   }
```

```
15    else if(operator.equalsIgnoreCase("*")){
16     return input1 * input2;
17    } else {
18     throw new RuntimeException("unsupported opera
19    }
20   }
21 }
22
```

Good: Closed for modification. To support “division” or “subtraction” add new classes like “SubtractOperation” that implements the interface “Operation”.

```
1 public interface Operation {
2     abstract int operate(int input1, int input2);
3 }
4
```

```
1 public class AddOperation implements Operation {
2   @Override
3   public int operate(int input1, int input2) {
4     return input1 + input2;
5   }
6 }
7
```

```
1 public class MultiplyOperation implements Operati
2
3   @Override
4   public int operate(int input1, int input2) {
5     return input1 * input2;
6   }
7 }
8
```

Q11. What is AOP and how does Spring provide AOP support?

A11. AOP stands for “Aspect Oriented Programming” and it compliments OOP. AOP is used for cross cutting concenrns like logging, auditing, service retry, deadlock retry, performance profiling, transaction management, etc. In OOP unit of modularity is an object, and in AOP unit of modularity is an **Aspect**. AOP framework is pluggable in Spring. AOP provides declarative cross cutting services and allows custom aspects to be implemented.

Spring AOP can be used together with **AOP Alliance** MethodInterceptors. AOP Alliance compliant interceptors

foster interoperability with other AOP frameworks such as Google Guice. Spring can be used with **AspectJ** as well, which has annotation syntax that is concise and expressive. AOP Alliance intends to facilitate and standardize the use of AOP to enhance existing middle ware environments (such as JEE), or development environements (e.g. Eclipse, NetBeans). The AOP Alliance also aims to ensure interoperability between Java/JEE AOP implementations to build a larger AOP community.

Example:

```
<!-- Spring AOP + AspectJ -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-aop</artifactId>
    <version>${spring.version}</version>
</dependency>

<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjrt</artifactId>
    <version>1.6.11</version>
</dependency>

<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>1.6.11</version>
</dependency>

```

Q12. What are the various AOP terminologies? What is the difference between join point and pointcut?
A12. Aspect, Pointcut, Join Points, and Advice.

Aspect is a mechanism where by you can call your method before, after or around some event.

To understand the difference between a **join point** and **pointcut**, think of pointcuts as specifying the weaving rules and join points as situations satisfying those rules. In the example below:

Pointcut defines rules saying, advice should be applied on annotation – “@DeadlockRetry(maxTries = 10, tryIntervalMillis = 5000)” on any method on any class.

Joinpoints will be a list of all methods present in classes with the "@DeadlockRetry" annotation so that advice can be applied on these selected methods.

Advice is the code that runs when program execution reaches a joint point in your pointcut.

Here is a real life **Advice** that retries the method when a database deadlock is encountered. The "DeadlockUtil" determines if the exception is deadlock related by inspecting the exception for presence "encountered a deadlock situation" message or Exception type being "CannotAcquireLockException". Spring aspects can work with five kinds of advice:

before: Run advice before the a method execution.
after: Run advice after the a method execution regardless of its outcome.
after-returning: Run advice after the a method execution only if method completes successfully.
after-throwing: Run advice after the a method execution only if method exits by throwing an exception.
around: Run advice before and after the advised method is invoked.

```
1  import java.lang.reflect.Method;
2  import org.aspectj.lang.ProceedingJoinPoint;
3  import org.aspectj.lang.annotation.Around;
4  import org.aspectj.lang.annotation.Aspect;
5  import org.aspectj.lang.reflect.MethodSignature;
6  import org.springframework.core.Ordered;
7
8  @Aspect
9  public class DeadlockRetryAspect implements Orde
10 {
11     //In AspectJ, a joint point is really the sp
12     //This runs in any method in any class with
13     @Around(value = "@annotation(deadlockRetry)"
14     public Object invoke(final ProceedingJoinPoi
15     {
16         final Integer maxTries = deadlockRetry.m
17         long tryIntervalMillis = deadlockRetry.t
18
19         Object target = pjp.getTarget();
20         MethodSignature signature = (MethodSigna
21         Method method = signature.getMethod();
22         int count = 0;
23
24         do
```

```
25          {
26              try
27              {
28                  count++;
29                  Object result = pjp.proceed();
30                  return result;
31              }
32              catch (Throwable e)
33              {
34                  if (!DeadlockUtil.isDeadLock(e))
35                  {
36                      throw new RuntimeException(e
37                  }
38
39                  if (tryIntervalMillis > 0)
40                  {
41                      try
42                      {
43                          Thread.sleep(tryInterval
44                      }
45                      catch (InterruptedException
46                      {
47                          ie.printStacktrace();
48                      }
49                  }
50              }
51          }
52          while (count <= maxTries);
53
54          //gets here only when all attempts have
55          throw new RuntimeException("DeadlockRetr
56                  + " due to deadlock in all retry
57                  new DeadlockDataAccessException(
58
59      }
60
61      @Override
62      public int getOrder()
63      {
64          return 99;
65      }
66  }
67
```

The custom annotation that is used as the point cut weaving rule.

```
1  import java.lang.annotation.ElementType;
2  import java.lang.annotation.Inherited;
3  import java.lang.annotation.Retention;
4  import java.lang.annotation.RetentionPolicy;
5  import java.lang.annotation.Target;
6
7  @Retention(RetentionPolicy.RUNTIME)
8  @Target(ElementType.METHOD)
9  @Inherited
10 public @interface DeadlockRetry
11 {
12     int maxTries() default 10;
13
14     int tryIntervalMillis() default 1000;
15 }
```

```
16
```

Bootstrap AOP into Spring context xml file.

```
1 <aop:aspectj-autoproxy />
2 <context:component-scan base-package="com.myapp"/
3
```

Q13. What does a typical Spring application contain? What is an @Autowired annotation?
A13. Spring applications predominantly use POJOs.

1) It will have Java interfaces with method declarations for functions related to business service, data access, etc.
2) Corresponding implementation classes for the interfaces defined.
3) Wiring up of the Spring managed beans and dependencies via XML based application context XML files, Anootations, and Java config file with @Configuration annotation.
4) Wiring up of JdbcTemplate, Hibernate, Transaction Manager, etc
5) wiring up of Spring AOP with aspectj for cross cutting concerns like deadlock retry, service retry, performance profiling, etc.
6) If using SpringMVC, then it needs to be bootstrapped via the web.xml or new JEE annotations.
7) If using some other Web framework, it needs to be bootstrapped via web.xml file. For example

```
1 web.xml --> myAppServletContext.xml --> myapp-app
```

Pay attention to how the different artifacts are wired up using both Spring xml files and annotations. The Spring beans can be wired either by **name** or **type**. **@Autowired** by default is a type driven injection. **@Autowired** is a Spring annotation, while **@Inject** is a JSR-330 annotation. **@Inject** is equivalent to @Autowired or @Autowired(required=true). **@Qualifier** spring annotation can be used to further fine-tune auto-wiring. There may be a situation when you create more than one bean of the same type and want to wire only one of them with a property, in such case you can use **@Qualifier** annotation

along with @Autowired to remove the confusion by specifying which exact bean will be wired. It is not a best practice to use @Autowired in large Spring applications as it can cause confusions.

Q14. Which DI would you favor – Constructor-based or setter-based DI?
A14. You can use both constructor-based and setter-based Dependency Injection. Favor constructor arguments for mandatory dependencies and setters for optional dependencies.

Q15. How can you inject a Java Collection in Spring?
A15. You can use list, set, map, or prop elements to configure collections.

Q16. What does the @Required annotation mean?
A16. This annotation indicates that the annotated bean property must be injected at configuration time, and if not set, the container throws **BeanInitializationException** if the affected bean property has not been populated.

Q17. What does the @Qualifier annotation mean?
A17. When there are **more than one beans of the same type** and only one is needed to be wired with a property, the @Qualifier annotation is used along with @Autowired annotation to remove the confusion by specifying which exact bean will be wired.

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

296 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**

‹ 03: ♥♦ Spring DIP, DI & IoC in detail interview Q&As

♦ 15 Java old I/O and NIO (i.e. New I/O) interview Q&As ›

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

Tags: Java/JEE FAQs, JEE FAQs, Popular frameworks

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

 ↑