05: ♦ 9 Spring Bean scopes interview Q&A

Posted on June 24, 2015 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, injectiong a DAO (i.e. Data Access Object) into a

service object. DAOs don't need to maintain conversation state. For example,

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/sch
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-
4      xsi:schemaLocation="http://www.springframewo
5
6      <bean id="myDaoDef" class="com.mycompany.und
7
8      <bean id="myServiceDef" class="com.mycompany
9         <constructor-arg name="myDao" ref="myDaoD
10     </bean>
11 </beans>
```

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 have 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.

Q4. Would both singleton and prototype bean's life cycle be managed by the Spring IoC container?
A4. 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 or 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."

Q5. What happens if you inject a prototype scoped bean into a singleton scoped bean?
A5. 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.

Q6. 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?
A6. 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 whilst 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". Spring lookup-method example to inject prototype scoped bean into a singleton scoped bean

Q7. What are the scopes defined in HTTP context?
A7. 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.

Q8. Does Spring allow you to define your own bean scopes?
A8. 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.

Q9. When will you use a spring lookup method?
A9. to inject a new prototype scoped been into a singleton scoped bean. Spring lookup-method example to inject prototype scoped bean into a singleton scoped bean.

Popular Posts

♦ 11 Spring boot interview questions & answers

827 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**

‹ ♥ javap for debugging and better understanding some Java concepts with 3 practical examples

Remote debugging in Java with Java Debug Wire Protocol (JDWP) ›

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

 ↑