Spring Highlights

EJBs are heavyweight and tightly coupled and Springs are lightweight and loosely coupled.

Heavyweight uses OS libraries and lightweight doesn’t use.

Tightly coupled dependent on other layers.

Loosely coupled doesn’t dependent on other layers.

ORM tools: - JDBC, Hibernate, JPA, ibatis

Spring developed using Runtime Polymorphism and association (has-A relation).

IOC container reads XML documents and pass the reference to pojo.

**IOC containers: Core and J2EE Container.**

**Spring MVC Container: Web Container.**

Core Container = Bean Factory(Interface) 🡪 XMLBeanFactory

J2EE Container = ApplicationContext(Interface) 🡪 ConfigurableApplicationContext(Interface) 🡪 ClassPathXmlApplicationContext(Class)

MVC Conatiner = WebApplicationContext(Interface)🡪 WebApplicationContextUtil (Factory Class)

XMLBeanFactory Container creates instance of pojo in user request (Lazy container).

ApplicationContext Container creates instance of pojo on loading xml file in case of singleton scope (Eager container).

What containers do:-

1. Create instance of pojo classes.
2. Mange life cycle of pojo.
3. Do dependency injection in pojo.

In xml file configure dtd or xsd in main beans tag.

Scope of beans: - Singleton, Prototype

MVC Scope of beans: - Request, Session, Global Session (Context)

Spring can access private constructors.

Two types of Dependency Injection:- Constructor Parameters and Setter Method

Bean references.

Required annotation. To activate annotation use “RequiredAnnotationBeanPostProcessor” class as bean.

P- Namespace and C- Namespaces.

Auto wiring: Auto dependency injection. Can inject secondary types only.

Auto wiring using Annotations. Activate it using bean class.

Stereo Type : Component, Controller, Repository, Services.

To create object by using auto scanning.

Use context-annotation to activate all annotations.

Inject static variables. “MethodInvokingFactoryBean”

How to create/get instance of java Singleton class, use factory-method tag in bean if static method.

If instance method use factory-bean tag also.

Create factory class by implementing factorybean interface.

Maintain pojo lifecycle 🡪 pojo implementing interfaces, configure in beans.xml, use annotations.

Classes to activate annotations.

DI by lookup method.3rd type.

What are lookup method? Interface methods, abstract method, and concreate method which we are overriding.

Method replacer.

DI using @Resource and @Inject. These are given by j2ee.

J2EE Stereo Types: @Named

Load properties from property file.

I18N – Language support

L10N – Business and Validation support.

IOC ApplicationContextListners.

Spring MVC using JSP model architecture

DispatcherServlet talk with hanlderMapping, Controller and View resolver.

Dispatcher Servlet Maping:-

1. Same name of spring xml file (dispatcher-servlet.xml) as dispatcher servlet name.
2. Read any xml file using init-param tag in servlet tag.
3. Read xml file using context-param and listener tags.

Handler Mapping types:

1. Bean Name handler mapping (Using name of pattern and controller mapping) (default).
2. Controller Class Name Handler Mapping (Same name controller automatic will find it)
3. Simple Url Handler Mapping (Map pattern to Id)

Controllers:-

1. Implementing Controller interface.
2. Extending AbstractController Class.
3. Extending ParameterizableViewController Class. (can set viewName DI).
4. UrlFileNameController (Just to forward to other page).
5. Throwaway Controller (like Struts 2 Action Class ,populate form data) (removed in 3.0)
6. MultiActionController (No override method, method name should be same as action name, if want to change method name as action name use method name resolver)
7. Basecommand controller
8. Abstract command controller.
9. AbstractForm controller.
10. Simpleform controller.
11. AbstractWizardFormController.

DAOs.

Spring DAO -> Use Interface –Implementation model.

1. Before implementing DAO classes must need to provide DAO interface.
2. In DAO interface, create signature with model objects.
3. In DAO implementations, get connection from connection pool.

Sun Datasources DS Implementations for connection pool:-

Apache 🡪 BasicDataSource

Mchange (c3p0) 🡪 CombopoolDataSource (Small Enterprise apps use this).

Spring 🡪 DriverManagerDataSource

Oracle Weblogic 🡪 WeblogicDataSource

Use JDBC Template for Spring JDBC implementation

**AOP Aspect Orient Programing**

1. Before - Method before advice
2. After :- After returning advice
3. Around :- Method Interceptor
4. While exceptions :- Throws Advice

Proxy Class: - ProxyFactoryBean: - to combine business code with services in controller.

Aspect – A Service

Advice – Service Provider

Point Cut – A point/condition to execute aspect for business methods

Advisor – This is a point cut with advice combination

Proxy – weaver – It combine services code with business

Target – business object

Approaches to implement AOP

1. Programmatic
2. Declarative (Xmls)
3. Annotations

Conditions execute/not execute service for a method (Staticmethodpointcut and Namemethodpointcut classes)

Advisors:-

DefaultPointCutAdvisor

RegExMethodPointCutAdvisor

**Spring in Action Book**

AOP: - Cross Cutting Concerns: - any functionality that affects multiple points of an application. Security, for example, is a cross-cutting concern, in that many methods in an application can have security rules applied to them.

Crosscutting concerns can now be modularized into special classes called **aspects**.

**Inversion of Control (IoC)** refers to a programming style where a framework or runtime, controls the program flow. Inversion of control means we are changing the control from normal way. It works on Dependency Inversion Principle. Inversion of Control (IoC) means that objects do not construct other objects on which they rely on. Instead, the application will get these objects from an external framework (an IoC container). **IoC** is a generic term meaning rather than having the application call the methods in a framework, the framework calls implementations provided by the application.

**DI is a software design pattern** that allow us to develop loosely coupled code. DI is a great way to reduce tight coupling between software components. DI also enables us to better manage future changes and other complexity in our software. The purpose of DI is to make code maintainable.