Spring interview questions

Dispatcher servlet
Front controller handle httprequest and send back

Spring

- Lightweight
- Support integration of other module
- Have variety of framework
    - App
    - Orm
    - Security
    - Mvc

IOC benefits

- Minimize code
- Easy To Test
- Loose Coupling
- Suppoet Eager Instatiation and Lazy loading

Dependency injection Types

- Setter Injection
- Constructor Injection

Bean Scope

- Singleton
- Prototype
- Request
- Session
- Global Session

Container

- Bean Factory vs Application Context
- Support I18
- Support mail
- Support JNDI

AOP

- Logging
- Transaction

What is the difference between concern and cross-cutting concern in Spring AOP?

- Concern is behavior which we want to have in a module of an application. Concern may be defined as a functionality we want to implement to solve a specific business problem.
    - E.g. in any eCommerce application different concerns (or modules) may be inventory management, shipping management, user management etc.

- Cross-cutting concern is a concern which is applicable throughout the application (or more than one module).
    - e.g. logging , security and data transfer are the concerns which are needed in almost every module of an application, hence they are termed as cross-cutting concerns.

- Advice is the implementation of cross-cutting concern which you are interested in applying on other modules of your application.

Pointcut is a predicate or expression that matches join points. Advice is associated with a pointcut expression and runs at any join point matched by the pointcut (for example, expression "execution(* EmployeeManager.getEmployeeById(..))" to match getEmployeeById() the method in EmployeeManager interface). The concept of join points as matched by pointcut expressions is central to AOP, and Spring uses the AspectJ pointcut expression language by defaultAdvices are of mainly 5 types :

Before advice : Advice that executes before a join point, but which does not have the ability to prevent execution flow proceeding to the join point (unless it throws an exception). To use this advice, use

@Before annotation.
After returning advice : Advice to be executed after a join point completes normally. For example, if a method returns without throwing an exception. To use this advice, use

@AfterReturning annotation.
After throwing advice : Advice to be executed if a method exits by throwing an exception. To use this advice, use @AfterThrowing annotation.

After advice : Advice to be executed regardless of the means by which a join point exits (normal or exceptional return). To use this advice, use @After annotation.

Around advice : Advice that surrounds a join point such as a method invocation. This is the most powerful kind of advice. To use this advice, use @Around annotation.

execution of the program, such as the execution of a method or the handling of an exception. In Spring AOP, a join point always represents a method execution. For

example, all the methods defined inside your EmployeeManager interface cab be considered joint points if you apply any cross-cutting concern of them.