Spring Core Framework

1. How does Spring relates or differs from Java EE?

Java EE that stands for Java Enterprise Edition is a specification. It is an effort for standardizing technologies to build Web and enterprise applications using the Java programming language. Vendors follow the Java EE specification to provide implementations, such as Web containers, application servers, message brokers, email systems, and so on. Programmers use Java EE implementations to develop applications.

The power of Java EE comes with its own set of complexities. You have to deal with a large number of XML descriptors, create set of classes for even the most basic enterprise components, and so on. From the developers perspective, it took considerable effort and time to package, ship, deploy or redeploy, and start an application. Spring is one solution to such problems encountered in Java EE.

Spring is a standalone framework for improvements and substitutions to Java EE. The Spring Framework is a Java platform supported by a large number of independent modules to develop enterprise applications. For example, you use the Spring MVC module to create MVC-based Web applications, Spring Security to secure applications, Spring Data for persistence, Spring Cloud for cloud-native distributed microservices, and so on.

With Spring, you only need a simple servlet container like Tomcat. You could pack only what you need and get your application running at considerably less time and effort. In short, you can consider Spring as an integration platform that allows you to use all Java EE technologies.

Both Spring and Java EE are evolving at a fast rate and each complements the other. Java EE 7 is arguably the best release of the EE specification to date and the contribution goes to the Spring Framework and also the other way around.

2. Why do we go for Spring Framework?

The Spring Framework is an open source application framework to make Java EE programming easy. Spring aims to help structure your whole application in a consistent and productive manner. It provides you with the building blocks to easily bind different layers of your application together. There are many cases in which Spring Framework can be used. For example, you can use it when you want to develop a Web application, or you want to expose RESTful Web services, create powerful messaging systems, highly distributed microservices, or big enterprise applications that integrate all of these and more.

3. What is Spring configuration file?

The Spring configuration file is an XML file that defines the beans that are used in the application. In a Spring application, an instance of ApplicationContext, when created, reads the configuration file and initializes all of the Beans. This file also contains information of all the Bean classes and describes how these classes are configured and introduced to each other. In addition, the configuration file tells Spring how to instantiate, configure, and assemble the objects in your application. Spring other than the XML configuration file also supports Annotation-based and Java-based configuration metadata.

4. What is BeanFactory?

BeanFactory is the way to access the Spring container. It contains a collection of beans with their definitions and instantiates them whenever asked for.
The BeanFactory is responsible for instantiating application objects, configuring such objects, and assembling their dependencies.

5. What is Spring IoC container and what are its types?

Inversion of Control (IoC) is a widely used pattern in Java that helps wire lightweight containers or assembles components from different projects into a cohesive application. IoC is a common characteristic of modern frameworks, including Spring.

In Spring, it is the Spring container that instantiates and manages the application objects. With the inversion of control, the flow of the business logic depends on the object graph that is instantiated by the assembler and is made possible by object interactions.

The Spring IoC container is responsible to instantiate, configure and assemble objects that are known as beans. It also manages their lifecycle. The container receives the information on what objects to instantiate, configure, and manage by reading the configuration metadata you define for your application.

Types of Spring IoC containers:

- BeanFactory: Provides the configuration framework and basic functionality
- ApplicationContext: Adds more enterprise-specific functionality. ApplicationContext is a superset ofBeanFactory

6. What are the benefits of Inversion Of Control (IoC)?

The primary benefits of IoC are:

- Makes it easier to test your code. Without IoC, the code you are testing is hard to isolate as it will be highly coupled to the rest of the system
- Enables replacing components without requiring recompilation. This is particularly useful when developing modular systems
- Minimizes the amount of code in your application.
- Enables loose coupling of your code
- Supports lazy loading of services and eager instantiation of beans

7. What are the common implementations of ApplicationContext?

The common implementations of ApplicationContext are:

- ClassPathXmlApplicationContext: Loads bean definitions from XML files present in the classpath
- FileSystemXmlApplicationContext: Loads bean definitions from XML files present in the file system
- XmlWebApplicationContext: Loads bean definitions from an XML file contained in a Web application. By default it loads the configuration file from /WEB-INF/applicationContext.xml.

8. What is the difference between BeanFactory and ApplicationContext?

BeanFactory and ApplicationContext are two ways to access beans in a Spring application.
Some key differences are:

- When you use the BeanFactory, Beans are instantiated when they are requested for the first time, like ingetBean() method, and not when the object of BeanFactory itself gets created. This way of instantiating is known as lazy-instantiation.But singleton beans do not get created lazily when you use theApplicationContext. By default, the ApplicationContext instantiates the singleton beans immediately and set its properties. So ApplicationContext loads singleton beans eagerly which is known as pre-instantiation.
- ApplicationContext creates and manages resources objects on its own whereas for BeanFactory you need to provide a resource object explicitly.
- ApplicationContext has the ability to publish the event to beans that are registered as listeners whileBeanFactory does not support this feature.
- ApplicationContext provides support for internationalization while BeanFactory doesn't.

Dependencies in Spring

9. What is Dependency Injection (DI) in Spring?

Inversion of Control (IoC) in Spring is implemented through Dependency Injection. DI says that you just need to describe how the objects should be created rather than creating your objects yourself. When you use DI, you don't have to directly connect your components and services in the code but you just need to describe which services are required by which components. The IoC container is then responsible for connecting all your components.

Therefore, DI is the primary reason for highly decoupled components that you find in Spring applications.

10. What are the different types of dependency injection?

- **Constructor-based dependency injection**: It injects the dependency via a constructor. This means that the required components are passed into a class at the time of instantiation.
- **Setter-based dependency injection**: It injects the dependency via a setter method. This involves calling setter methods on your beans after invoking a no-argument constructor.
- **Property based**

11. Which type of DI would you suggest – Constructor-based or Setter-based?

It depends. You can use constructor-based DI for mandatory dependencies and setter-based DI for optional dependencies. In setter-based DI, partial injection of dependencies is possible. If you do not inject some, it will take the default values for those primitives. On the other hand, in constructor-based injection, partial injection of dependencies is not possible. This is because for calling a constructor we must pass all the arguments, else an exception will be thrown.

So based on your requirements, use the proper type of DI.

12. Explain how Spring resolves dependencies.

Spring resolves dependencies as follows:

- Spring creates the ApplicationContext and initializes it based on the configuration metadata where the beans are defined.
- The dependencies for each bean is provided at the time of bean creation and is expressed in the form of properties, constructor arguments, or arguments to normal methods.

- Each property or constructor argument is a set of value that is injected and is converted from its specified format to the actual type of that property or constructor argument.

13. How do you declare inter-bean dependencies?

Declaration of inter-bean dependencies is as simple as having one bean method call another. However, this method works only when the @Bean method is declared within a @Configuration class.
Another way of declaring inter-bean dependency is by taking the dependency as a @Bean method parameter. The injection is now by type. The dependency would be first resolved by type and if duplicates are found, by name.

Spring Beans

14. What are Spring Beans?

Spring Bean is an object that is managed by the Spring Framework. A Spring Bean is instantiated, configured, and managed by the Spring Framework container (IoC container). You can define the beans in a configuration file or use annotations. The beans are then instantiated by the Spring IoC container, added to the Spring ApplicationContext, and injected into your application at the points they are asked for.

15. What are the different Bean scopes?

Beans, which are Spring-managed Java classes are created and wired by the Spring Framework. Spring allows you to define how these beans will be created. The scope of the bean is one of those definitions. To specify the bean scope, you can either use Spring annotations or define it in the configuration file.

Spring Framework supports the following scopes:

- **Singleton**: If you use this bean scope, no matter how many times you call the getBean() method, the same bean instance will be returned. This is the default bean scope in Spring.
- **Prototype**: If you use this bean scope, every getBean() call creates a new instance of Spring bean.
- **Request**: If you use this bean scope, it allows each HTTP request to have its own instance of the bean created and supplied by Spring Framework. This bean scope is available on web-aware application context.
- **Session**: If you use this bean scope, it allows the Web application to have bean instance per session basis.
- **Global-session**: If you use this bean scope, it allows bean instance per global session. This bean scope is available on Web-aware application context.

16. What is the default scope of Bean in Spring Framework?

The default scope of a Spring Bean is Singleton.

17. Are Singleton Beans thread safe?

No, Singleton Beans are not thread safe. Singleton Spring Bean has no relation with thread safety. Spring container only manages life-cycle of objects. So, if a non-thread safe object is injected then obviously it is not thread-safe. To make it thread-safe you have to handle it programmatically.

If it is a Web-application, request scope can achieve thread-safety as for each new request it creates a new object. Similarly, prototype scope will also do this as for each invocation it creates a new bean.

18. What are inner beans in Spring?

Inner beans are those beans which are implicitly made anonymous but also scoped as a prototype. If you want Beans to be accessed by any other bean used in your application, you need to define them as inner beans. An inner bean definition does not require a defined id or name; the Spring container ignores these values. The container also ignores the scope of the bean.

19. What is bean autowiring?

The process of injecting the dependencies of a Spring bean during initialization of the bean is called bean wiring. It is basically a process that associates the beans together in the container. When wiring beans, you should tell the container what beans are needed and how the container should use dependency injection to tie them together. Using bean autowiring, you need not define the bean configuration. That is, you need not define the dependency of each bean. To do so, you can just use the autowire attribute of the bean and specify which dependency type it must be, such as byName or byType.
An example of auto-wiring with the byName type is this.
@Bean(autowire=Autowire.BY_NAME)
The preceding annotation is the equivalent of this XML configuration.

20. What are the different autowiring modes?

The different autowiring modes are:

- no: This is the default mode. It specifies no autowiring. Hence you need to set it manually via the ref attribute.

- byname: It specifies auto-wiring by property name. If the name of a bean is same as the name of other bean property, Spring will auto-wire it.
- byType: It specifies auto-wiring by property data type. If the data type of a bean is compatible with the data type of another bean property, Spring will auto wire it.
- constructor: It specifies the autowiring byType mode in constructor argument.

21. What are some limitations of autowiring?

Some limitations of autowiring are:

- Simple properties such as primitives and Strings and arrays of such simple properties cannot be autowired. This is a limitation by design.
- The information of autowiring will not be available to the tools that generate documentation from the Spring container.
- Autowiring is less precise than explicit wiring. In autowiring, the relationships between your Spring-managed objects are no longer documented explicitly. Autowiring is based on the idea that you indicate what properties are to be autowired and Spring deduces its corresponding value. This means, in autowiring, you run the risk of providing more than one match.On the other hand, explicit wiring, as its name implies, requires the target bean to be mapped with an explicit property name/value pair for Spring to use to link the two objects.
- Ambiguity arises when there are dependencies that expect a single value, but multiple beans exist. For example, there are multiple beans within the container that match the type specified in the setter methods or constructor arguments to be autowired. In such situations, when no unique bean definition is present, the container throws an exception of type NoSuchBeanDefinitionException.

22. What are Lazy-initialized beans?

Spring's bean factory, when first created, pre-initiates the Spring beans of the application. This is a good practice as it resolves any dependency error at the time of start-up. However, this might not be applicable for all types of applications as bean instantiation is both resource and time-consuming.

If an application contains a large number of beans, instantiating those at start-up takes considerable time. Also, a significant amount of memory is required for storing all the bean instances. A solution is to instantiate a bean only when required. In Spring such beans are called lazy-initialized bean. A bean that is lazily initialized asks the IoC container to create its instance when it is first requested for and not at start-up.
The @Lazy annotation marks a bean as lazy-initialized.

23. Explain Spring bean lifecycle and callbacks in brief

When a bean is instantiated, it might be required to perform some initialization to get it into a usable state. Similarly, when the bean is not required it must be removed from the container and some clean-up process performed. All these happen during the lifecycle of a Spring bean.

The sequence of a bean lifecycle in Spring is this:

- **Instantiation**: First the spring container finds the bean's definition from the configuration file or annotation, and instantiates the bean.
- **Property population**: Using dependency injection, Spring populates all properties as specified in the bean definition.
- **Setting Bean name**: The bean name is set by passing the bean's id to setBeanName() method when the bean implements BeanNameAware interface. In annotation based beans, the value field indicates the bean name.
- **Pre Initialization**: If there are any objects of BeanPostProcessors that are associated with the BeanFactory, then the method postProcessBeforeInitialization() will be called even before the properties for the Bean are set.
- **Bean initialization**: If the Bean class implements the InitializingBean interface, then the methodafterPropertiesSet() will be called once all the Bean properties defined in the configuration file are set. If the configuration file contains an init-method attribute in the Bean definition, then the value for the attribute will be resolved to a method name in the Bean class and that method will be called.
- **Post initialization**: The postProcessAfterInitialization() method will be called if there are anyBeanPostProcessors attached to the BeanFactory object.
- **Ready to use**: After all the above cycles, the beans defined are ready to be used by the application.
- **Destroy:** If the bean implements a DisposableBean interface, it calls the destroy() method to destroy the bean.

The Spring Framework provides several callback methods to create a bean and some methods to destroy the bean in the Spring IoC container. Beans can be notified after creation and before they are destroyed and removed from the bean container. This involves specifying the callback method to be invoked by the container. Attribute init-method="initMethodName" is specified for the initialization callback and destroy-method="destroyMethodName" is specified for the destroy callback in the XML configuration file. Here,initMethodName and destroyMethodName are names of instance methods in the bean class.
The Spring Framework also provides marker interfaces that can change the behaviour of your bean.InitializingBean and DisposableBean are two such marker Interfaces. When a class implements these interfaces, the container calls afterPropertiesSet() for the former and destroy() for the latter to allow the bean to perform certain actions upon initialization and destruction.

24. What is component scanning and how it works?

In a traditional Spring application, the beans or components are declared in a configuration file in order to allow the Spring container to detect and register them. However, you should know that Spring is capable of auto scanning, detecting, and further instantiating beans from the already defined project package. This can save you from the tedious task of having to declare beans in a separate XML file. Additionally, it helps in avoiding bean definitions.

As you know, Spring also supports annotation based configuration. You can annotate the class with @Component, or one of the stereotype @Controller, @Service, and @Repository to indicate that this class is a candidate component. A candidate component is a component that matches against filter criteria and has a corresponding bean definition registered with the container.
As @Repository, @Service, and @Controller are annotated with @Component. So, you can just use @Component for all the components and Spring will auto-scan all of them.

25. What is the NoSuchBeanDefinitionException and how to fix it?

NoSuchBeanDefinitionException is an exception that the container throws when BeanFactory cannot find a definition for a bean instance when asked for. This may happen because of a non-existing bean, a non-unique bean, or a manually registered singleton instance without an associated bean definition.
Trying to inject an undefined bean is a common cause of this exception. Often this happens when the bean resides in a package that is not scanned by Spring.

Another reason for NoSuchBeanDefinitionException being thrown is because of a non-unique bean. That is when two bean definitions exist in the context instead of one. For example, consider two beans: BeanA and BeanB that implements an IBean interface. If another Bean, sayBeanX auto wires IBean, Spring will throw an exception of type NoSuchBeanDefinitionException. This happens because Spring cannot figure out which implementation (BeanA or BeanB) to inject. To fix this, use the@Qualifier annotation to explicitly name the implementation to inject.

Spring Annotations

26. Are annotations better than XML for configuring Spring?

Spring supports both XML and Annotation based configuration, and both complement each other. It all depends on the requirements and the developer's personal opinion on which of the configurations better suits him.

Annotations provide a lot of context in their declaration, leading to shorter and more concise configuration. All the information is in a single source file. When the class changes, you don't have to worry about the XML file.

However, XML excels at wiring up components without touching their source code or recompiling them. XML clearly separates the Plain Old Java Object (POJO) and its behavior.

No matter the choice, Spring can accommodate both styles and even mix them together.

27. What is annotation-based container configuration?

Starting with Spring 2.5, annotation support has been added to the Spring Framework. It is now possible to configure Beans using annotations.

XML configurations are injected after annotations. Therefore, when you use both annotations and XML based configuration, annotations configuration gets overridden by the XML ones. In the annotation-based configuration, the configuration is moved into the component class itself by using annotations on the relevant class, method, or field declaration.

28. Explain @Component and the stereotype annotations in brief.

@Component is a generic stereotype for any Spring-managed component. @Repository, @Service, and @Contollerare specializations of @Component for more specific use cases.
The @Component annotation is used on classes to indicate a Spring component. The @Component annotation marks the Java class as a bean or says component so that the component-scanning mechanism of Spring can add into the application context.
The stereotype annotations are:

- @Service: This annotation is used on classes. The @Service marks a Java class that performs some service, such as execute business logic, perform calculations and call external APIs. This annotation is a specialized form of the @Component annotation intended to be used in the service layer.
- @Repository: This annotation is used on Java classes which directly access the database. The @Repositoryannotation works as a marker for any class that fulfills the role of repository or Data Access Object. This annotation has an automatic translation feature. For example, when an exception occurs in the @Repository, there is a handler for that exception and there is no need to add a try-catch block.
- @Controller: This annotation is used on Java classes that play the role of controller in your application. The@Controller annotation allows auto detection

of component classes in the classpath and auto-registering bean definitions for them. To enable auto detection of such annotated controllers, you can add component scanning to your configuration. The Java class annotated with @Controller is capable of handling multiple request mappings.

Spring Aspect Oriented Programming (AOP)

29. What is AOP?

AOP that stands for Aspect Oriented Programming is a way to modify existing classes in a code base to change their behavior based on rules defined separately. This modification can be done before the classes are packaged into a JAR or WAR, or can happen dynamically while the code is being loaded.

Rather than finding all the points of code that you want to modify in the source code and hard coding them, you just define the rules for how to find points of interest in the code and what changes you would like to do to them. These rules are called aspects – the A of AOP.

30. What is Aspect?

Consider a use case where an end user enters login and password, goes to the home page, performs some action, and returns back.

The implementation code would typically consist of an HTML client that in turn would send the message to a servlet, which would perform the action. Now as you implement additional features, such as logging and transacting, your implementation code would just get bloated more. And assuming that features like logging or transaction management would have to be implemented across all modules, this would just make the code redundant, and maintenance much more difficult. This is where you can use aspect.

When you have common concerns, such as logging, and exception handling that needs to be used across multiple objects, you can put it into one single module, known as an aspect that can be used across all other objects.

Aspects are Java classes configured in a configuration file. Spring AspectJ provides the @Aspect annotation to declare a class as an aspect.

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

Concern means logic or functionality. Concern is a behaviour you want to have in a module of your application. A Concern may also be defined as a functionality that you want to implement to solve a specific business problem.

On the other hand, a cross-cutting concern is a concern or a common functionality that spans across tiers and layers. For example, logging, security, and data transfer are the concerns which are needed in almost every module of an application and therefore may be considered as cross-cutting concerns.

32. What is advice, joinpoint, pointcut and advice arguments in AOP?

- **Advice**: Advice is the actual action that will be taken either before or after the method execution. This is the actual piece of code that is invoked during the program execution by the Spring AOP Framework. It is the implementation of cross-cutting concern which you are interested in applying to other modules of your application.
- **Joinpoint**: A joinpoint is a candidate point in the program execution of the application where an aspect can be plugged in. This joinpoint could be a point where a method is called, an exception thrown, or even a field being modified. At these points, you can insert your aspect's code between the normal flow of your application to add a new feature or say behavior.
- **Pointcut**: A pointcut tells at what join points an advice should be applied. Advice can be applied at any joinpoint supported by the AOP framework. You can specify these pointcuts using Class and method names or through regular expressions that define the matching class and method name patterns.
- **Advice Arguments**: Advice arguments are the parameters that are passed to advice methods. You can use joinpoint as a parameter and get the method signature or the target object. You can use args() expression in the pointcut to be applied to any method that matches the argument pattern. If you use this, then you need to use the same name in the advice method from where argument type is determined.

Spring Model View Controller (MVC)

33. What is Spring MVC framework?

Spring Model View Controller (MVC) is the Web module of the Spring Framework. It provides a rich functionality for building robust Web Applications. The Spring MVC Framework is architected and designed in such a way that every piece of logic and functionality is highly configurable.

The MVC framework of Spring is request-driven and is designed around a central servlet, calledDispatcherServlet that dispatches requests to controllers.

Spring's DispatcherServlet is completely integrated with Spring IoC container and allows you to use every other feature of Spring.

34. What is DispatcherServlet?

In Spring MVC, the client sends a request to the Web container in the form of HTTP request. This incoming request is intercepted by the Front controller which is the DispatcherServlet and it then tries to find out appropriate handler mappings.
The DispatcherServlet dispatches the request to the appropriate controller with the help of handler mappings. Once controller returns the model along with the logical view, DispatcherServlet takes the help of ViewResolver to resolve the view and will pass the model data to the view, which is finally rendered on the browser.
You can, therefore, consider DispatcherServlet as the entry point of Spring MVC is the. It is a normal servlet class which implements HttpServlet base class.

35. How to upload a file in Spring MVC application?

The consider MultipartResolver interface is used for uploading files in Spring MVC. The Spring Framework provides two types of considering MultipartResolver implementations. One is to use with Apache Commons FileUpload. The other for use with Servlet 3.0 multipart request parsing.

36. What is the use of @RestController annotation?

This annotation is used at the class level. The consider @RestController annotation marks the class as a REST controller where every method returns a domain object instead of a view. By annotating a class with this annotation you no longer need to add a response body to all the consider RequestMapping methods. It means that you no more use view-resolvers or send HTML in response. You just send the domain object as HTTP response in the standard format like JSON that is understood by the REST consumers.

37. What are Spring profiles?

Spring allows developers to register beans by condition. Using the @Profile annotation you can map the bean to a particular profile. The annotation simply takes the names of one (or multiple) profiles.
You can now map your beans to different profiles – for example, dev, test, and prod. You can then activate different profiles in different environments to bootstrap just the beans you need.
The @Profile annotation can be applied at the class level or method level.
Any @Component or @Configurationcan be marked with @Profile to control when it is loaded.

38. What are some of the important Spring MVC annotations you have used?

Some important annotations in Spring MVC are:

- @Controller: Used on Java classes that play the role of controller in your application. The @Controllerannotation allows auto detection of component classes in the classpath and auto-registering bean definitions for them. To enable auto detection of such annotated controllers, you can add component scanning to your configuration. The Java class annotated with @Controller is capable of handling multiple request mappings.
- @RequestMapping: Used in both class and method level.
  The @RequestMapping annotation is used to map web requests onto specific handler classes and handler methods. When @RequestMapping is used on a class level, it creates a base URI for which the controller will be used. When this annotation is used on methods, it will give you the URI on which the handler methods will be executed. From this, you can infer that the class level request mapping will remain the same whereas each handler method will have their own request mapping.
  Sometimes you may want to perform different operations based on the HTTP method used, even though the request URI may remain the same. In such situations, you can use @RequestMapping method attribute to narrow down the HTTP methods in order to invoke the methods of your class.
- @PathVariable: Used to annotate request handler method arguments.
  The @RequestMapping annotation can be used to handle dynamic changes in the URI where certain URI value acts as a parameter. The@PathVariable annotation can be used to declare this parameter.
- @RequestAttribute: Used to bind the request attribute to a handler method parameter. Spring retrieves the value of the named attribute to populate the parameter annotated with @RequestAttribute.
- @ResponseBody: Used to annotate request handler methods.
  The @ResponseBodyannotation is similar to the@RequestBody annotation.
  The @ResponseBodyannotation indicates that the result type should be written straight in the response body in a format you specify, such as JSON. Spring converts the returned object into a response body by using
  the HttpMessageConveter.

39. What is ContextLoaderListener?

Usually, when you build multi-tier applications you don't want to clutter all the beans in one config file[appname]-servlet.xml. For example, consider you configure Spring Security and you want to include all those beans in a separate security-context.xml. Similarly, you want to configure all the beans belonging to service layer in applicationContext.xml, and beans belonging to DAO layer in dao-context.xml. When you configure all these beans in different context files, you need to somehow let know Spring that these files exist, because Spring only knows about [appname]-servlet.xml. It is the ContextLoaderListener that helps Spring recognize all the other context files.

40. What is Spring Expression Language (SpEL)?

The Spring Expression Language (SpEL for short) is a powerful expression language that supports querying and manipulating an object graph at runtime. Basically, SpEL helps you in querying an object at runtime, instead of having to depend only on configuration files. You can use it for both XML and annotation based configurations. It supports a wide range of relational expressions to accessing properties of a class or method.

41. Name some view technologies that Spring supports.

Some view technologies that Spring supports are:

- Thymeleaf
- Groovy Markup Template Engine
- Velocity & FreeMarker
- JSP & JSTL
- Tiles

Spring Data Access

42. What is JdbcTemplate?

Spring simplifies handling database access with the Spring JDBC Template exposed through the JdbcTemplateclass. Spring JDBC Template simplifies the use of JDBC and helps to avoid common errors. Spring JDBC Template is responsible for executing the core JDBC workflow. Application code only needs to provide SQL and retrieve results. Spring JDBC Template executes SQL queries or updates and initiates iteration overResultSet objects. It also catches JDBC exceptions and translates them to the generic, more informative exception hierarchy.

43. Why use an embedded database? Name few embedded databases that Spring supports natively.

Spring 3 introduced the support for embedded Java database engines. The embedded databases are very useful during the development phase of the project. They provide almost the same features as their standalone version, and at the same time are lightweight, have a quick start time, and extremely fast. Being fast they are a big boon to developers who have to continuously run their integration tests directly or as part of Continuous Integration (CI).

An embedded database also improve testability and provides ease of configuration. All these features allow developers to focus more on the development instead of how to configure a data source. It eliminates the burden of bigger, bulkier databases like Oracle.

Some embedded databases that Spring natively supports are HSQL, H2, and Derby.

Spring Boot

44. What is Spring Boot?

Spring Boot makes creating production-grade applications and services powered by the Spring Framework absolutely easy. Spring Boot with its opinionated view of the Spring platform enables new and existing users to quickly get up and running Spring applications quickly.

Spring Boot provides a range of non-functional features, such as embedded servers, security, metrics, health checks, and externalized configuration that are common to large classes of projects.
In addition, Spring Boot favours convention over configuration. Therefore, you can develop the application without worrying about any XML configuration.

45. Why will you use Spring Boot in your application?

Spring-based applications have a lot of configurations. For example, if you use Spring MVC, you need to add configurations, such as component scan, dispatcher servlet, view resolver, and Web JARs.

Spring Boot looks at only two things. One, the JARs available on the classpath. Second, the existing configuration of the application.

Based on these, Spring Boot provides basic configuration needed to configure the application. This is called Auto Configuration. Spring Boot makes it easier for you to create Spring application.

For example, consider that you want to create a Spring Boot application with ActiveMQ as the messaging service. You add the artifactId as spring−Boot−starter−activemq in your Maven POM, and Spring Boot will take all the defaults and create an application with ActiveMQ configured. If you don't want to use the inbuilt ActiveMQ, you can simply override spring.activemq.broker-url in your application.properties to use an external one.

46. List few advantages and disadvantages of Spring Boot.

Some advantages of Spring Boot are:

- Provides a lot of default configurations to get up and running Spring applications faster
- Comes with embedded Tomcat, Jetty, and Undertow servlet containers, which avoids JAR deployments.
- Reduces boilerplate code
- Increases productivity as you can create Spring application quickly
- Provides a lot of maven integrated starter projects. You don't have the problem of version mismatch
- Enables quickly creating a sample project using spring Boot initializer

Some disadvantages of Spring Boot are:

- If you are new to Spring and want to learn how the dependency injection, AOP programming, and proxies work, starting with Spring Boot is not a good choice. Spring Boot hides most of these details from you.
- Converting your old Spring application to Spring Boot application may not be straightforward and can be time-consuming.
- Spring Boot, with the unused dependencies, may increase the deployment binary size unnecessarily.

47. What is a Spring Boot starter POM?

Spring Boot starters reduce the number of dependencies that need to be manually added to an application. The starter POM contains a lot of dependencies to get an application up and running quickly. They are convenient dependency descriptors that can be added to your application's Maven POM. For example, if you are developing an application that uses Spring Batch for batch processing, you just have to include spring-boot-starter-batch that will import all the required dependencies.

48. What is Actuator in Spring Boot?

Spring Boot actuator is a subproject of Spring Boot used to access the current state of running the application in the production environment. Actuators are mainly used to expose different types of information about the running application, such as health, metrics, info, dump, and env. You can simply use the restful Web services endpoints provided by Spring Boot actuator and check various metrics.

49. What is DevTools in Spring Boot?

Spring Boot comes with DevTools to increase the productivity of developer. With Spring Boot DevTools, you as a programmer, don't need to restart your application for each small changes made to the application. You just need to reload the changes. Spring DevTools thus avoids the pain of redeploying application every time you make any change. This module is disabled in the production environment.

50. Explain CommandLineRunner in Spring Boot. How does it differ from ApplicationRunner?

In Spring Boot application you can execute any required task just before Spring Boot finishes its start-up. To do so, you need to create Spring bean using CommandLineRunner or ApplicationRunner interface and Spring Boot will automatically detect them.

Both the interfaces have a run() method that needs to be overridden in implementing classes. You also need to make the class as a bean by using a Spring stereotype such as @Component. The run() method of bothCommandLineRunner and ApplicationRunner are executed just before Spring application finishes its start-up.

CommandLineRunner and ApplicationRunner serve the same purpose. The difference is that the run() method ofCommandLineRunner accepts an array of String as an argument while the run() method of ApplicationRunneraccepts Spring ApplicationArguments as an argument.

The arguments which you pass to main() method while starting Spring Boot, can be accessed in the run()method of both CommandLineRunner and ApplicationRunner implementation classes. You can also create more than one CommandLineRunner and ApplicationRunner beans. To execute them in an order, you can use the @Orderannotation or Ordered interface.