**Spring Boot**

What is Spring IOC?

* Spring IOC: **Spring IOC** is the mechanism to achieve loose coupling between Objects dependencies. To achieve loose coupling and dynamic binding of the objects at runtime, objects dependencies are injected by other assembler objects.

What is a Framework?

* Framework is a pre-written code which is used by developers to create web apps.

What is a Spring bean?

* A spring bean is simply a Java object. When java objects are created by the spring container, then spring refers to them as “Spring Beans”. Spring Beans are created from normal java classes just like java objects.

What are Bean scopes?

* Scope refers to the lifecycle of a bean.
* How long does the bean live?
* How many instances are created?
* How is the bean shared?

Default Scope is “Singleton”.

What is a Singleton scope?

* Spring container creates only one instance of the bean by default.
* It is cached in memory.
* All requests for the bean will return a shared reference to the same bean.

What are different Spring Bean Scopes?

* Singleton
* Create a single shared instance of the bean, it is default scope.
* Prototype
* Creates a new bean instance for each container request.
* Request
* Scoped to an HTTP web request, only used for web apps.
* Session
* Scoped to an HTTP web session, only used for web apps.
* Global-session
* Scoped to a global HTTP web session, only used for web apps.

What are Java Annotations?

* Special labels/ markers added to java classes.
* Provides the metadata about the class.
* Processed at compile time or runtime for special processing.

What are the goals of Spring boot?

* Enable building production ready applications.
* Provide common non-functional features
* Embedded servers
* Metrices
* Health checks
* Externalized configuration

What Spring boot is NOT?

* ZERO code generation, spring boot don’t generate any code.
* Spring boot is neither an application server nor a web server

What are the features of Spring Boot?

* Quick starter projects with auto configuration like web, JPA
* Spring starter web contains Spring core, Spring MVC, Validations, Logging frameworks as transitive dependencies.
* Embedded servers like tomcat and jetty.
* Before spring-boot we must create a war and we have to deploy it in a Linux Box and again we need to install a server in the box but with spring boot no need to install a server again.
* Well defined project-built structure.

What steps we must do before Spring boot?

* In this step, we'll look at one of the projects which we developed in our Spring MVC. Before we start with any project, we need to decide what frameworks and dependencies to use. This was a web project, so we would want to use Spring MVC and things like that. So, we needed to make decisions on what frameworks, dependencies, and what version of them to use. Sometimes versions mismatch will happen between dependencies then we need to decide which dependencies to use.
* We need to create a complete spring configuration file. In that file we must define the component scan and then we must configure a view resolver to re-direct the views to a JSP. To implement the internationalization, we need to implement a message source and a local resolver.
* We need to configure web.xml. In web.xml we must configure dispatcher servlet in web.xml so that it can handle all the requests and act as a front controller. We needed to configure the context configuration location in here and we also needed to configure the spring security. We needed to configure the filter for it and make sure that it intercepts all the requests.
* So, we configured dependencies, dependency version, Spring configuration, configuration for internationalization, logging. All that stuff we looked at in this specific step is the kind of stuff which you don’t need to do with Spring boot. Spring boot would automatically provide all that stuff for you, so you can concentrate on the business logic.

What is Spring boot Auto configuration?

* We have @SprintBootApplication annotation which indicates
* It is spring context file.
* It enables auto configuration.
* It enables Component scan for the specific package and sub packages.
* Spring boot looks at
* Frameworks available on the CLASSPATH.
* Existing configuration for application.

Based on these, Spring boot provides basic configuration needed to configure the application with these frameworks. This is called AUTO-CONFIGURATION.

* When application started an auto configure jar is present on the classpath which looks all the jars and finds out, there a spring web jar in classpath so it configures dispatcher servlets, internationalization, and logging. This is called Spring boot auto configuration.

**Difference between Spring, SpringMVC, SpringBoot ?**

**Spring Framework**: Spring is an open-source lightweight framework widely used to

develop enterprise applications. Most important feature of spring

framework is dependency injection. At the core of all Spring

modules are dependency injection or IOC.

**SpringMVC Framework**: Spring MVC provides decoupled way of developing web apps. With

Simple concepts like Dispatcher servlet, ModelAndView and

ViewResolver, it makes easy to develop web applications.

**Spring boot**: It is built on spring framework which makes easier to develop a

Production ready application with auto configuration.

The **spring-boot-starter-parent** dependency is **the parent POM providing dependency and plugin management for Spring Boot-based applications**. It contains the default versions of Java to use, the default versions of dependencies that Spring Boot uses, and the default configuration of the Maven plugins.

Sprint Boot Annotations

* **@SprintBootApplication** annotation which indicates
* It is spring context file.
* It enables auto configuration.
* It enables Component scan for the specific package and sub packages.
* @**ConditionalOnMissingBean**(“DataSource.class”)
* When there is no bean with DataSource then the below method will be executed and creates a bean.
* **@ConditionalOnClass**(“EmbeddedDatabaseType.class”)
* When there is bean with EmbeddedDatabaseType in classpath then the configuration will happen.

**Spring AOP**

**AOP is a programming paradigm that aims to increase modularity by allowing the separation of cross-cutting concerns.** It does this by adding additional behavior to existing code without modifying the code itself.

Instead, we can declare the new code and the new behaviors separately.

Spring's [AOP framework](https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#aop) helps us implement these cross-cutting concerns.

@Before("execution(\* com.vishal.spring.aop.bussiness.\*.\*(..))")

**public** **void** before(JoinPoint joinPoint) {

// advice

logger.info("Intercepted Method calls -{}", joinPoint);

}

------------------------------------------------------------------------------------------------------------------------

@Before("execution(\* com.vishal.spring.aop.bussiness.\*.\*(..))")

\* -> for any return type

com.vishal.spring.aop.bussiness -> in these package

\* -> every class

\* -> every method

(..) -> for n number of arguments

--------------------------------------------------------------------------------------------------------------------------

Execute the below logic(advice) in the method before executing business class method.

"execution(\* com.vishal.spring.aop.bussiness.\*.\*(..))"

------------------------------------------------------------------------------------------------------------------------

**Pointcut**: The expression which defines what kind of methods I want to intercept is called pointcut

Ex: ("execution(\* com.vishal.spring.aop.bussiness.\*.\*(..))")

**Advice**: What should I do when I intercepted. Simply it is the code or logic inside the method of @Before annotation.

Ex: @Before("execution(\* com.vishal.spring.aop.bussiness.\*.\*(..))")

**public** **void** before(JoinPoint joinPoint) {

**// advice**

**logger.info("Intercepted Method calls -{}", joinPoint);**}

**Aspect:** It is the combination of pointcut and advice.

**Joinpoint**: Joinpoint is specific interception of a method call. It is a specific execution instance. These are the points where your aspect’s code can be inserted into the normal flow of your application to add new behavior.

**Weaving and Weaver**: Process around where the whole thing weaved is called weaving. The process of implementing the AOP around you are method calls is called weaving and the framework which implements is called weaver.

**@After, @AfterThrowing, @AfterReturning:**

**@After** is executed after the method(pointcut) is executed.

**@AfterThrowing** is executed when the method throws the error.

**@AfterReturning** is executed when the method returns something.

@AfterReturning(value ="execution(\* com.vishal.spring.aop.bussiness.\*.\*(..))"

,returning = "result")

**public** **void** before(JoinPoint joinPoint,Object result) {

logger.info("{} returned with value {}",joinPoint,result);

}

@AfterThrowing(value ="execution(\* com.vishal.spring.aop.bussiness.\*.\*(..))"

,throwing = "exception")

**public** **void** afterThrowing(JoinPoint joinPoint,Exception exception) {

logger.info("{} returned with value {}",joinPoint,exception);

}

@After(value ="execution(\* com.vishal.spring.aop.bussiness.\*.\*(..))"

)

**public** **void** after(JoinPoint joinPoint) {

logger.info("{} ",joinPoint);

}

**@Around:** It is a combination of @Before and @After. We can use this advice to calculate method execution time.

@Around("execution(\* com.vishal.spring.aop.bussiness.\*.\*(..))")

**public** **void** around(ProceedingJoinPoint joinPoint) **throws** Throwable {

**long** startTime = System.*currentTimeMillis*();

joinPoint.proceed();

**long** timeTaken = System.*currentTimeMillis*() - startTime;

logger.info("Time Taken by {} is {}",joinPoint,timeTaken);

}

**Spring Data JPA**

**JPA:** The Java Persistence API(JPA) is the standard way of persisting Java objects into relational databases. The JPA consists of two parts.

* A mapping subsystem to map classes onto relational tables.
* An Entity manager API to access the objects, define and execute queries and more.

**Entity Manager:** Entity manager manages the entities. All the operations that we are performing on a specific session are all stored in entity manager. All operations are not actually stored in entity manager but in persistence context. Entity manager is interface for persistence context. All operations should go through entity manager.

Whenever you are an inside of a transaction and entity manager is managing something like updating, deleting, inserting that thing continues to manage by entity manager until the end of the transaction.

**Spring Data JPA, JPA, Hibernate:**

JPA is a specification that defines an API for object relational mappings and for managing persistent objects. Hibernate is a popular implementation of this specification.

Spring data JPA adds a layer on top of JPA. That means it uses all features defined by the JPA specification especially the entity and association mappings, the entity lifecycle management and JPA’s query capabilities. On top of that, Spring data JPA adds its own features like a no-code implementation of the repository pattern and the creation of database queries from method names and reduced boiler plate code.

**Rest API**

**What is API?**

An API is a code that allows to software’s to communicate with each other.

**What is REST?**

Representational state transfer is a software architectural style that defines a set of constraints to be used for creating web services. Web services that conform to the ‘Rest’ architectural style called Restfull web services, provides interoperability between computer systems on the internet.

**What is REST API?**

A Restfull API is an application program interface(API) that uses HTTP requests to GET, PUT, POST, DELETE data. An API for a website is code that allows two software’s to communicate with each other’s.

**What is a Web service?**

Web services are client and sever applications that communicates over the worldwide web’s Hypertext Transfer Protocol(HTTP). Web services provide a standard means of interoperating between software applications running on a variety of platforms and frameworks.

**Docker**

**What is Docker?**

Docker is a container management service and tool designed to make it easier to create, deploy and run applications by using containers. Containers allow a developer to package up an application with all the parts it needs, such as libraries and other dependencies and ship it all out as one package.

**What is Docker Image?**

An image contains all the things that your application needs to run, it contains the software(Java 1.8), libraries, dependencies. When we run these command, an image is downloaded to our machine. So local image was created from registry.

**What is difference between Image and Container in Docker?**

Image is a static version and container is a running version of your image. For the same image multiple containers can be running.

Image is a class and container is an object, so from the image we can create a multiple containers like objects.

**What is DockerClient and DockerDaemon?**

DockerClient is responsible for sending commands to the docker daemon.

DockerDaemon is responsible for managing our local stuff and also pull something from the Image Registry if something is not available on our local. Docker Daemon also helps us to create an image and push that image to registry.

**Kubernetes**

**What is Kubernetes?**

Kubernetes is an open-source container-orchestration system for automating computer application deployment, scaling and management. It was originally designed by Google and is now maintained by the cloud Native computing Foundation.

**What are the features of Kubernetes?**

Auto scaling – Scale containers based on demand.

Service discovery – Help microservices find one another.

Load Balancer – Distribute load among multiple instances of a microservice.

Self-Healing – Do health checks and replace failing instances.

Zero Downtime deployments – Release new versions without downtime.

**Self-introduction**

Hi, my name is Vishal, I am from Visakhapatnam.

I am having 2.1 years of experience in web development.

I worked on backend technologies like java, spring boot and front-end technologies like Reactjs and angular.

My roles are different kinds of rest API development, Junit test case, database stored procedure writing.

I worked on databases like SQL SERVER, MYSQL, COUCHDB.

I developed different types of API and reusable modules in spring boot and developed stored procedures and integrated with code.

**Java Interview question**

What is difference between List and Set?

What is Association, Aggregation, Composition in java?

Can we make any class immutable in java?

Difference between Comparator and Comparable?