SPRING FRAMEWORK

Framework

A framework is a large body of predefined code to which developers can add code to solve a problem in a specific domain. There are many popular Java frameworks including Java Server Faces (JSF), Maven, Hibernate, Struts, and Spring.

Spring Framework

The Spring Framework (Spring) is an open-source application framework that provides infrastructure support for developing Java applications.

Spring was developed in the year **2002 by Rod Johnson**. And implemented in the year 2003 By Tomcat.

One of the most popular Java Enterprise Edition (Java EE) frameworks, Spring helps developers create high performing applications using plain old Java objects (POJOs).

We can say the spring is the framework of frameworks.

Spring has several modules which helps us to build java applications in a simple way and removes many complexities.

Spring Framework includes

Plain Old Java Object (POJO)

Aspect-oriented programming (AOP)

Dependency injection (DI)

Even with all these technologies, Spring is a lightweight framework that can be used to create scalable, secure, and robust enterprise web applications.

POJO stands for Plain Old Java Object. It is an ordinary Java object, not bound by any special restriction other than those forced by the Java Language Specification and not requiring any class path. POJOs are used for increasing the readability and re-usability of a program.

Aspect oriented programming (AOP) as the name suggests uses aspects in programming. It can be defined as the breaking of code into different modules, also known as modularization, where the aspect is the key unit of modularity.

Dependency Injection (DI) is a design pattern that removes the dependency from the programming code so that it can be easy to manage and test the application.

Spring – Inversion of Control (IoC)

Spring IoC Container is the core of Spring Framework…

IoC containers creates objects for us, so that we need not to create manually.

IoC (Inversion of control) Container is responsible to instantiate, configure and assemble the objects.

To create objects for us, IoC container needs some information.

We provide that configuration in an xml file.

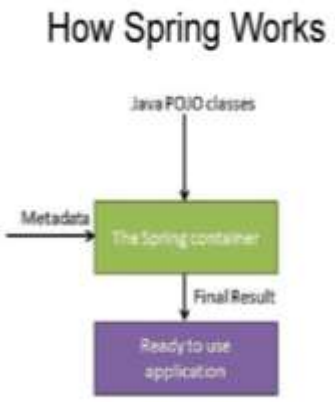
The main task performed by the Ioc Container are:

a. To instantiate the application class

b. To configure the object

c. To assemble the dependencies between the objects...

Spring IOC Architecture

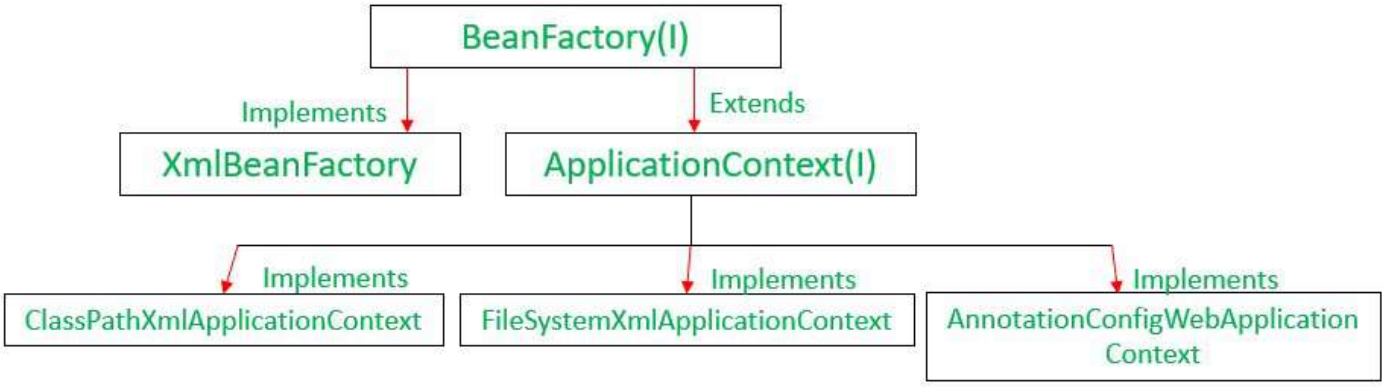


Types of IoC Containers

There are two types of IoC containers

1. Bean Factory-Core Container
2. Application Context-EJB Container

Hierarchy of Bean factory and Application context



Bean Factory / Core Container

The BeanFactory is the actual container which instantiates, configures, and manages a number of beans. These beans typically collaborate with one another, and thus have dependencies between themselves.

Application Context / EJB Container

The Application Context is **Spring's advanced container**. Similar to BeanFactory, it can load bean definitions, wire beans together, and dispense beans upon request.

The container gets its instructions on what objects to instantiate, configure, and assemble by reading configuration metadata.

Difference between bean factory and application context

BeanFactory: -

Bean factory is an interface. Bean Factory supports lazy Loading. It doesn't support annotation-based dependency injection.

ApplicationContext: -

Application Context is an interface which extends bean factory. Application Context supports aggressive loading. It supports annotation-based dependency injection.

Different bean scopes in Spring

Singleton: - The bean instance will be only one and same instance will be returned by the IoC container. It is the default scope.

Prototype: - The bean instance will be created each time newly whenever we request.

Request: - The bean instance will be created as per HTTP request

Session: - The bean instance will be created as per HTTP session.

Global Session: - The bean instance will be created as per HTTP global session. It can be used in portlet context only.

Different ways of dependency injection using XML file Configuration

1.Constructor based dependency injection :- we can inject the dependency by constructor by using <constructor-arg> (sub element of <bean>) is used for constructor injection…

Here we can inject primitive and String based values, dependent object collection values etc…

For Constructor injection we have to declare a suitable constructor to inject the targeted values..

After the creation of class we have to create gag of constructor in xml file inside specific bean and we can insert values by using attribute and object by using (ref) attribute..

Example:-

<constructor-arg index="0" value = "a"> </constructor-arg>

<constructor-arg index="1" value = "10"> </constructor-arg>

2. By using setter :- we can inject the dependency by using setter method also..

The <property> statement of <bean>is used for setter injection

By using setter we can inject primitive and string-based values, dependent object , collection values etc.

For setter injection , we need to declare setter method for each property of specific class

Then inside the respectice bean tag we have to declare <property> sub element. Where we can pass the value to the properties by providing its variable name to name attribute.

Here "name" points the attribute where value specifies the data which is going to assign to it.

Different ways of dependency injection using Annotation Configuration

We can do dependency injection with the help of annotations…..

There are many annotations which helps us to perform dependency injection. Some of them are below:-

@Component:- we use this annotation on the above of the class to get the object of that class…

@autowired:- we use this annotation on the above of dependent object, so that we can get that object

@Configuration:- we use this annotation on the above of the helper class to configure

@component-scan:-we use this annotation to specify the package, so that we can get object of the classes of that package

Init – method attribute of bean tag in bean

The init-method attribute used to specify a method to be called on bean immediately upon instantiation
We can declare init – attribute inside bean tag of the xml file of spring
We have to pass method name as value to the init – method
Example:-

```
class A {
public void m1() {
                    }

        }
```

In xml file:-

```
<bean id = "a" class ="com.ty.A" init-method = "m1">
</bean>
```

destroy – method attribute in bean tag

The destroy-method is called before the bean is removed from the container
We can declare destroy-method attribute inside a bean tag of xml file and can initialize with a method name.
The method which initialized will get executed before bean instance is removed from container
Example:-

```
class A {
          public void m1() {
        }
  }
```

In xml file:-

```
<bean id = "a" class ="com.ty.A" destroy-method = "m1">
</bean>
```

Spring – Model and View Controller (MVC)

MVC is abbreviated as **Model View Controller** is a design pattern created for developing applications specifically web applications. As the name suggests, it has three major parts.
The traditional software design pattern works in an "Input - Process - Output" pattern whereas MVC works as "Controller -Model - View" approach.
With the emergence of the MVC model, creation of application takes different aspects individually into consideration. These aspects of the application are:

- UI Logic
- Input logic
- Business Logic

But there is a loose coupling between these aspects or elements.
According to this model, each element needs to exist in an application but not tightly connected or interlinked.
The UI logic deals with the view or front end of the application.

The Input logic deals with the controller.
Lastly, business logic deals with the model of an application.
This loosely coupled element helps developers handle development complication when building any application. It helps users to focus on one specific element of the implementation at a time. Take a scenario where you are working with business; you can specifically deal with that and focus on the business logic building without depending on the view logic.

MODEL: The Model encloses the clean application related data. But the model does not deal with any logic about how to present the data.

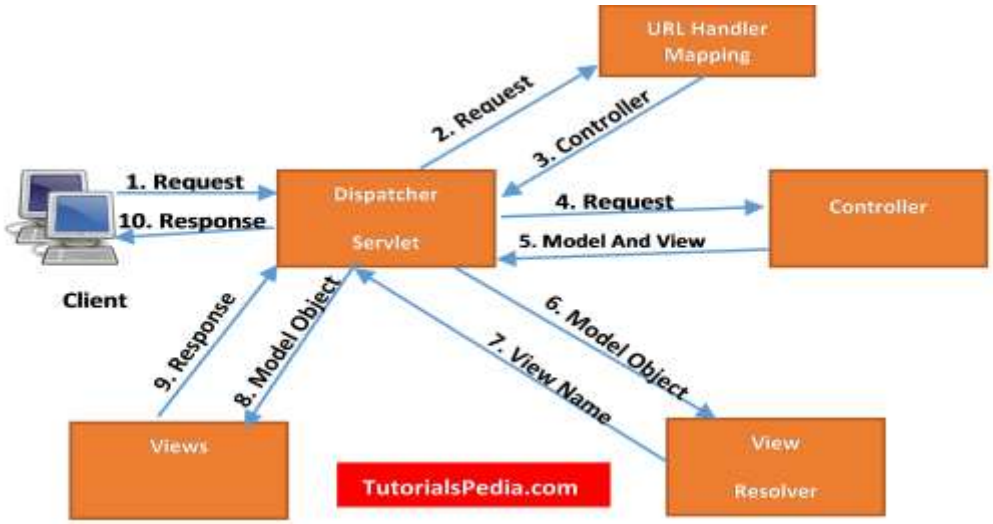
VIEW: The View element is used for presenting the data of the model to the user. This element deals with how to link up with the model's data but doesn't provide any logic regarding what this data all about or how users can use these data.

CONTROLLER: The Controller is in between the model and the view element. It listens to all the incident and actions triggered in the view and performs an appropriate response back to the events.

A Spring MVC is a Java framework which is used to build web applications. It follows the Model-View-Controller design pattern. It implements all the basic features of a core spring framework like Inversion of Control, Dependency Injection.
A Spring MVC provides an elegant solution to use MVC in spring framework by the help of **DispatcherServlet**.
Here, **DispatcherServlet** is a class that receives the incoming request and maps it to the right resource such as controllers, models, and views.

Dispatcher Servlet / Front Controller

- In Spring Web MVC, the Dispatcher Servlet class works as the front controller. It is responsible to manage the flow of the Spring MVC application.
- Every request from the clint is sent to front controller and the front controller will decide which controller should be called for that request.
- We no need to create front controller spring mvc will provide us.
- All the requests from web.xml are sent to front controller.



Handler Mapping

- Handler Mapping works as helper for Dispatcher servlet. It helps to identify appropriate controller bean to Dispatcher servlet.
- It identified controller bean with URL. It matches the name in URL with controller bean.
- If it matches it will return back it to dispatcher servlet. finally the servlet executes the business method of controller and returns ModelAndView object back to dispatcher servlet.
- By default, the dispatcher servlet uses the BeanNameUrlHandlerMapping to map the incoming request.

Controller

- Any class specified with @Controller annotation serves the role of controller.
- Controller acts as an interface between view and model.
- Dispatcher servlet sends the request to the controller and controller returns the model and view.

View Resolver

- After getting the model and view from controller the dispatcher servlet takes the help of view resolver for which view, we have to send as response to the request.

Different Annotations Used in Spring MVC

Annotation	Description
@Controller	The @Controller annotation **indicates that a particular class serves the role of a controller**
@RequestMapping	One of the most important annotations in spring is the **@RequestMapping Annotation** which is used to map HTTP requests to handler methods of MVC and REST controllers. In Spring MVC applications, the DispatcherServlet (Front Controller) is responsible for routing incoming HTTP requests to handler methods of controllers.
@ModelAttribute	The @ModelAttribute annotation **binds a method parameter or method return value to a named model attribute and then exposes it to a web view**.
@PathVariable	@PathVariable annotation is **used to extract the value from the URI**. It is most suitable for the RESTful web service where the URL contains some value.

Annotation	Description
@Configuration	@Configuration annotation which **indicates that the class has @Bean definition methods**. So Spring container can process the class and generate Spring Beans to be used in the application. This annotation is part of the spring core framework.
@ComponentScan	**@ComponentScan annotation along with the @Configuration annotation to specify the packages that we want to be scanned**. @ComponentScan without arguments tells Spring to scan the current package and all of its sub-packages.

Spring BOOT

Spring boot is a module of spring framework which is used to create stand-alone, production-grade Spring based Applications with minimum programmer's efforts.

Spring Boot Starters

Spring boot Starters make the development of Spring Boot based Java Applications much faster and easier.
It automatically downloads the maven dependencies as well as pre-defined setups.
Thus, it makes the developer's job easy by preventing the manual selection of the right dependencies / versions.

Some of the most used Spring Boot Starters

- spring-boot-starter-data-jpa
  Starter for using Spring Data JPA with Hibernate.
- spring-boot-starter-web
  Starter for building web, including RESTful, applications using Spring MVC. Uses Tomcat as the default embedded container.
- spring-boot-starter-test
  Starter for testing Spring Boot applications with libraries including JUnit Jupiter, Ham crest and Mockito.
- spring-boot-starter-data-jdbc
  Starter for using Spring Data JDBC.

Advantages of Spring Boot

- No Requirement for Complex XML Configuration.
- Embedded Tomcat Server to run Boot applications.
- An auto-Configuration feature by Spring boot that configures your application automatically for certain dependencies.
- Decreased boilerplate code.
- It provides opinionated '**starter**' POMs to simplify our Maven configuration.
- It creates **stand-alone** Spring applications that can be started using Java **-jar**.

Exception handling in Spring Boot

- Handling exceptions and errors in APIs and sending the proper response to the client is good for enterprise applications.
- The @ControllerAdvice is an annotation, to handle the exceptions globally.
- The @ExceptionHandler is an annotation used to handle the specific exceptions and sending the custom responses to the client.
- For Examples if I want to throw an custom exception there are three steps.
- 1.To create exception by extending runtime exception.

```
package com.tutorialspoint.demo.exception;
public class ProductNotfoundException extends RuntimeException {
   private static final long serialVersionUID = 1L;
}
```

Difference between Spring MVC and Spring Boot

S.No.	SPRING MVC	SPRING BOOT
1.	Spring MVC is a Model View, and Controller based web framework widely used to develop web applications.	Spring Boot is built on top of the conventional spring framework, widely used to develop REST APIs.
2.	If we are using Spring MVC, we need to build the configuration manually.	If we are using Spring Boot, there is no need to build the configuration manually.
3.	In the Spring MVC, a deployment descriptor is required.	In the Spring Boot, there is no need for a deployment descriptor.
4.	Spring MVC specifies each dependency separately.	It wraps the dependencies together in a single unit.
5.	Spring MVC framework consists of four components : Model, View, Controller, and Front Controller.	There are four main layers in Spring Boot: Presentation Layer, Data Access Layer, Service Layer, and Integration Layer.
6.	It takes more time in development.	It reduces development time and increases productivity.
7.	Spring MVC do not provide powerful batch processing.	Powerful batch processing is provided by Spring Boot.
8.	Ready to use feature are provided by it for building web applications.	Default configurations are provided by it for building a Spring powered framework.

Annotations used in Spring Boot

- **@SpringBootApplication:**
  This annotation is used to mark a configuration class that declares one or more @Bean methods and also triggers auto-configuration and component scanning.
- **@Bean:**
  It is applied on a method to specify that it returns a bean to be managed by Spring context.
- **@ RestController:**
  It is used to build REST API in a declarative way. @RestController annotation is applied to a class to mark it as a request handler, and Spring will do the building and provide the RESTful web service at runtime.
- **@ Service:**
  It is used at class level. It tells the Spring that class contains the business logic.
- **@ Repository:**
  It is a class-level annotation. The repository is a **DAOs** (Data Access Object) that access the database directly. The repository does all the operations related to the database.
- **@Configuration**
  It indicates that a class is a configuration class that may contain bean definitions.

- **@Autowired**
  Marks a constructor, field, or setter method to be auto wired by Spring dependency injection.

- **@ PostMapping**
  It maps the HTTP POST requests on the specific handler method. It is used to create a web service endpoint that creates.
  It is used instead of using: @RequestMapping(method = RequestMethod.POST)

- **@ GetMapping**
  It maps the HTTP GET requests on the specific handler method. It is used to create a web service endpoint that fetches.
  It is used instead of using: @RequestMapping(method = RequestMethod.GET)

- **@ DeleteMapping**
  It maps the HTTP DELETE requests on the specific handler method. It is used to create a web service endpoint that deletes a resource.
  It is used instead of using: @RequestMapping(method = RequestMethod.DELETE)

- **@ PutMapping**
  It maps the HTTP PUT requests on the specific handler method. It is used to create a web service endpoint that creates or updates.
  It is used instead of using: @RequestMapping(method = RequestMethod.PUT)

- ***@*RequestBody**
  It is used to bind HTTP request with an object in a method parameter. Internally it uses HTTP MessageConverters to convert the body of the request.
  When we annotate a method parameter with @RequestBody, the Spring framework binds the incoming HTTP request body to that parameter.

- **@PathVariable**
  It is used to extract the values from the URI. It is most suitable for the RESTful web service, where the URL contains a path variable. We can define multiple @PathVariable in a method.

- **@RequestParam:**
  It is used to extract the query parameters form the URL. It is also known as a query parameter. It is most suitable for web applications. It can specify default values if the query parameter is not present in the URL.

- **@EnableSwagger2**
  It is used to enable the Swagger2 for your Spring Boot application.

- **@Controller**
  It indicates that a particular class serves the role of a controller. Spring Controller annotation is typically used in combination with annotated handler methods based on the @RequestMapping annotation. It can be applied to classes only. It's used to mark a class as a web request handler.

Status Code

An HTTP Status Code refers to a 3-digit code that is part of a server's HTTP Response.
The first digit of the code describes the category in which the response falls. This already gives a hint to determine whether the request was successful or not.

The Internet Assigned Numbers Authority (IANA) maintains the official registry of HTTP Status Codes. Below are the different categories:

1. **Informational (1xx)**: Indicates that the request was received and the process is continuing. It alerts the sender to wait for a final response.
2. **Successful (2xx)**: Indicates that the request was successfully received, understood, and accepted.
3. **Redirection (3xx)**: Indicates that further action must be taken to complete the request.
4. **Client Errors (4xx)**: Indicates that an error occurred during the request processing and it is the client who caused the error.
5. **Server Errors (5xx)**: Indicates that an error occurred during request processing but that it was by the server.

Some of the most common HTTP codes:

Code	Status	Description
200	OK	The request was successfully completed.
201	Created	A new resource was successfully created.
400	Bad Request	The request was invalid.
401	Unauthorized	The request did not include an authentication token or the authentication token was expired.
403	Forbidden	The client did not have permission to access the requested resource.
404	Not Found	The requested resource was not found.
405	Method Not Allowed	The HTTP method in the request was not supported by the resource. For example, the DELETE method cannot be used with the Agent API.
500	Internal Server Error	The request was not completed due to an internal error on the server side.

503	Service Unavailable	The server was unavailable.

Response Entity

- ResponseEntity is meant to represent the entire HTTP response.
  We can control anything that goes into it such as HTTP status code, headers, and body.
- *ResponseEntity* is a generic type.
- While *@ResponseBody* puts the return value into the body of the response, ResponseEntity also allows us to add headers and status code.