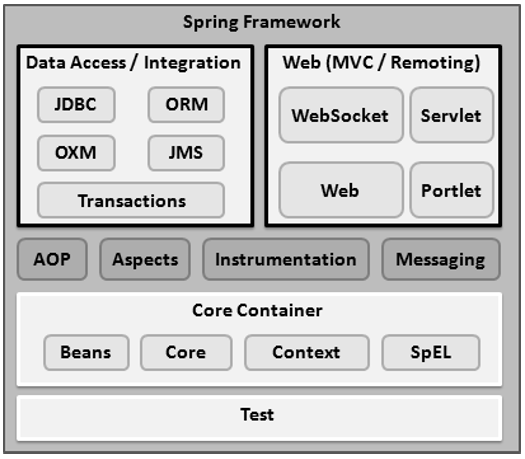
**Spring Framework**

Spring framework is an open source Java platform. It was initially written by Rod Johnson. Spring is lightweight when it comes to size and transparency. The core features of the Spring Framework can be used in developing any Java application, but there are extensions for building web applications on top of the Java EE platform. Spring framework targets to make J2EE development easier to use and promotes good programming practices by enabling a POJO-based programming model.

Spring Architecture-

****

Core Container-

The Core Container consists of the Core, Beans, Context, and Expression Language modules the details of which are as follows −

* The Core module provides the fundamental parts of the framework, including the IoC and Dependency Injection features.
* The Bean module provides BeanFactory, which is a sophisticated implementation of the factory pattern.
* The Context module builds on the solid base provided by the Core and Beans modules and it is a medium to access any objects defined and configured. The ApplicationContext interface is the focal point of the Context module.
* The SpEL module provides a powerful expression language for querying and manipulating an object graph at runtime.

## Data Access/Integration-

The Data Access/Integration layer consists of the JDBC, ORM, OXM, JMS and Transaction modules whose detail is as follows -

* The JDBC module provides a JDBC-abstraction layer that removes the need for tedious JDBC related coding.
* The ORM module provides integration layers for popular object-relational mapping APIs, including JPA, JDO, Hibernate, and iBatis.
* The OXM module provides an abstraction layer that supports Object/XML mapping implementations for JAXB, Castor, XMLBeans, JiBX and XStream.
* The Java Messaging Service JMS module contains features for producing and consuming messages.
* The Transaction module supports programmatic and declarative transaction management for classes that implement special interfaces and for all your POJOs.

## Web-

The Web layer consists of the Web, Web-MVC, Web-Socket, and Web-Portlet modules the details of which are as follows −

* The Web module provides basic web-oriented integration features such as multipart file-upload functionality and the initialization of the IoC container using servlet listeners and a web-oriented application context.
* The Web-MVC module contains Spring's Model-View-Controller (MVC) implementation for web applications.
* The Web-Socket module provides support for WebSocket-based, two-way communication between the client and the server in web applications.
* The Web-Portlet module provides the MVC implementation to be used in a portlet environment and mirrors the functionality of Web-Servlet module.

There are few other important modules like AOP, Aspects, Instrumentation, Web and Test modules the details of which are as follows −

* The AOP module provides an aspect-oriented programming implementation allowing you to define method-interceptors and pointcuts to cleanly decouple code that implements functionality that should be separated.
* The Aspects module provides integration with AspectJ, which is again a powerful and mature AOP framework.
* The Instrumentation module provides class instrumentation support and class loader implementations to be used in certain application servers.
* The Messaging module provides support for STOMP as the WebSocket sub-protocol to use in applications. It also supports an annotation programming model for routing and processing STOMP messages from WebSocket clients.
* The Test module supports the testing of Spring components with JUnit or TestNG frameworks.

Dependency Injection-

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. Dependency Injection makes our programming code loosely coupled.

The Dependency Injection is a design pattern that removes the dependency of the programs. In such case we provide the information from the external source such as XML file.

In the spring boot, @Autowired annotation is used for dependency injection. In spring boot application, all loaded beans are eligible for auto wiring to another bean. The annotation @Autowired in spring boot is used to auto-wire a bean into another bean.

The autowiring functionality has four modes. These are ' no ', ' byName ', ' byType ' and ' constructor '. Another autowire mode autodetect has been deprecated

Scope of Bean-

When defining a <bean> you have the option of declaring a scope for that bean. For example, to force Spring to produce a new bean instance each time one is needed, you should declare the bean's scope attribute to be prototype. Similarly, if you want Spring to return the same bean instance each time one is needed, you should declare the bean's scope attribute to be singleton.

The Spring Framework supports the following five scopes, three of which are available only if you use a web-aware ApplicationContext.

**Singleton-**

This scopes the bean definition to a single instance per Spring IoC container (default).

**Prototype-**

This scopes a single bean definition to have any number of object instances.

**Request-**

This scopes a bean definition to an HTTP request. Only valid in the context of a web-aware Spring ApplicationContext.

**Session-**

This scopes a bean definition to an HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

**global-session-**

This scopes a bean definition to a global HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

Singleton Scope-

If a scope is set to singleton, the Spring IoC container creates exactly one instance of the object defined by that bean definition. This single instance is stored in a cache of such singleton beans, and all subsequent requests and references for that named bean return the cached object.

The default scope is always singleton. However, when you need one and only one instance of a bean, you can set the scope property to singleton in the bean configuration file.

@ComponentScan-

The @ComponentScan annotation is used with the @Configuration annotation to tell Spring the packages to scan for annotated components. @ComponentScan is also used to specify base packages and base package classes using thebasePackageClasses or basePackages attributes of @ComponentScan.The basePackageClasses attribute is a type-safe alternative to basePackages. When you specify basePackageClasses, Spring will scan the package (and subpackages) of the classes you specify.

@PostConstruct-

@PostConstruct is an annotation used on a method that needs to be executed after dependency injection is done to perform any initialization.

@preDestroy-

The @PreDestroy annotation is used on methods as a callback notification to signal that the instance is in the process of being removed by the container.

@Inject is used for autorwiring, it gives you a chance to use standard annotations rather than Spring specific annotation like @Autowired.

@Named annotation is used for conflict resolution if there are multiple candidates of the same type.

Spring AOP-

Aspect Oriented Programming (AOP) compliments OOPs in the sense that it also provides modularity. But the key unit of modularity is aspect than class. AOP breaks the program logic into distinct parts (called concerns). It is used to increase modularity by cross-cutting concerns. A cross-cutting concern is a concern that can affect the whole application and should be centralized in one location in code as possible, such as transaction management, authentication, logging, security etc.

Advice represents an action taken by an aspect at a particular join point. There are different types of advices:

* Before Advice: it executes before a join point.
* After Returning Advice: it executes after a joint point completes normally.
* After Throwing Advice: it executes if method exits by throwing an exception.
* After (finally) Advice: it executes after a join point regardless of join point exit whether normally or exceptional return.
* Around Advice: It executes before and after a join point.

Web Application with Spring MVC-

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.

* **Model** - A model contains the data of the application. A data can be a single object or a collection of objects.
* **Controller** - A controller contains the business logic of an application. Here, the @Controller annotation is used to mark the class as the controller.
* **View** - A view represents the provided information in a particular format. Generally, JSP+JSTL is used to create a view page. Although spring also supports other view technologies such as Apache Velocity, Thymeleaf and FreeMarker.
* **Front Controller** - In Spring Web MVC, the DispatcherServlet class works as the front controller. It is responsible to manage the flow of the Spring MVC application.