**Spring** is a *lightweight* framework. It can be thought of as a *framework of frameworks* because it provides **support to various frameworks** such as **Struts**, **Hibernate**, Tapestry, EJB, JSF etc.

The Spring framework comprises of many **modules** such as core, beans, context, expression language, AOP, Aspects, Instrumentation, JDBC, ORM, OXM, JMS, Transaction, Web, Servlet, Struts etc. These modules are grouped into

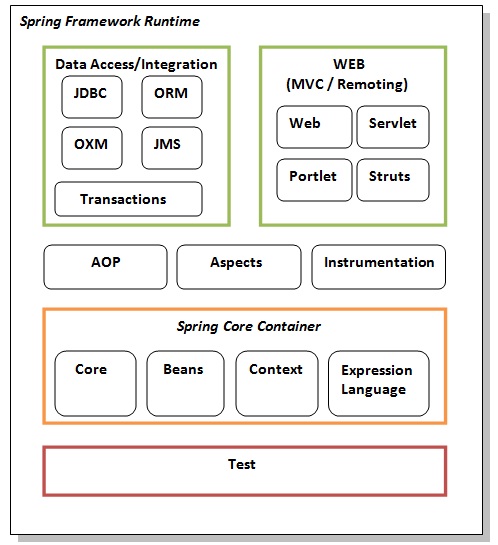
1. Test

2. Core Container

3. a. AOP, 3.bAspects, 3.c.Instrumentation

4. Data Access / Integration

5. Web (MVC / Remoting) as displayed in the following diagram.



**Test**

This layer provides support of testing with **JUnit and TestNG**.

### Spring Core Container

#### Core and Beans

These modules provide **IOC and Dependency Injection** features.

IOC (Inversion of Control) and DI (Dependency Injection) is a design pattern to provide loose coupling.

#### Context

This module supports internationalization (I18N), EJB, JMS, Basic Remoting.

#### Expression Language

It is an extension to the EL defined in JSP. It provides support to setting and getting property values, method invocation, accessing collections and indexers, named variables, logical and arithmetic operators, retrieval of objects by name etc.

To deal with the databases, spring framework provides **JdbcTemplate**

**weaving** :The process of applying aspects to a target object to **create a new proxied object is called weaving**

# **IoC Container(Spring container):manage the life cycle of Objects**

The main tasks performed by IoC container are:

* to **instantiate** the application class
* to **configure** the object
* to **assemble** the dependencies between the objects

**RegisterShutdownHook handles shutdown of IOC**

**Spring container is responsible to create the objects, wire the objects together, manage the objects life cycle.**

In Spring based applications, the objects lives in spring container.

There are two types of IoC containers. (container implementations in spring by using )

1. **BeanFactory**
2. **ApplicationContext:** ApplicationContext extends BeanFactory and It adds some extra functionality than BeanFactory.

**1.Using BeanFactory**

Resource resource=**new** ClassPathResource("applicationContext.xml");

BeanFactory factory=**new** XmlBeanFactory(resource);

**2.Using** **ApplicationContext**

ApplicationContext context=**new** ClassPathXmlApplicationContext("-||-")

**ClassPathXmlApplicationContext** searches .xml file in: classpath

**Beans:**

 A bean is an object that is **instantiated, assembled, and managed by a Spring IoC container**. These beans are created with the configuration metadata that you supply to the container. For example, in the form of XML <bean/> definitions which you have already seen in the previous chapters.

**configuration metadata**, which is needed for the container to know the following −

* How to create a bean
* Bean's lifecycle details
* Bean's dependencies

Example:

**<bean** id="dao" class="com.javatpoint.dao.EmpDao"**>**

**<property** name="template" ref="jt"**></property>**

**</bean>**

**Dependency Injection (DI):**

Dependency Injection (DI) is a design pattern that **removes** the dependency from the programming code.

**Need for Dependency Injection:**

Suppose class A needs the object of class B to instantiate or operate a method, then class A is said to be **dependent** on class B. [Spring](https://www.geeksforgeeks.org/introduction-to-spring-framework/) IOC resolves such dependencies with Dependency Injection, which makes the code **easier to test and reuse**.

Spring framework provides two ways to inject dependency(**DI** is acheived by using)

* By Constructor
* By Setter method(Partial dependency achieved )

**Difference between constructor and setter injection**

 If we use both constructor and setter injection, IOC container will use the setter injection.

**So setter injection is flexible than constructor injection.**

# **Coupling in Java**

Coupling: means do changes in class A force related changes in class B.

**There are two types of coupling: 1.Tight 2.Loose**

**For detail:** <https://www.geeksforgeeks.org/coupling-in-java/>

**Autowiring**

When beans are combined together within the Spring container, it’s called **wiring** .Autowiring enables the programmer to inject the bean automatically. We don't need to write explicit injection logic.

<bean **id**="emp" **class**="com.javatpoint.Employee" **autowire**="byName" />

The autowiring modes are given below:

|  |  |  |
| --- | --- | --- |
| **No.** | **Mode** | **Description** |
| 1) | No | this is the default mode, it means autowiring is not enabled. |
| 2) | Byname | injects the bean based on the **property name**. It uses setter method. |
| 3) | byType | injects the bean based on the **property type**. It uses setter method. |
| 4) | Constructor | It injects the bean using **constructor** |

There are 5 **bean** scopes in spring framework.

**Singleton** (default scope)

**Prototype**

Request

Session

**globalsession**

**Configuration metadata:**

**Configuration metadata** can be provided to Spring container in following ways:

way of providing metadata in spring framework**.**

**XML-Based configuration:**

**Annotation-Based configuration**

**Java-based configuration**

**Transaction management**

Spring framework provides two type of transaction management supports:

1. **Programmatic Transaction Management**: should be used for few transaction operations.
2. **Declarative Transaction Management**: should be used for many transaction operations.

### ****AOP****

### ****Basic of AOP:****

### 1. One of the key components of Spring Framework is the Aspect oriented programming (AOP) framework.

### 2. Aspect-Oriented Programming entails breaking down program logic into distinct parts called so-called concerns.

### 3. AOP is like triggers in programming languages such as .NET, Java.

### 4. When a method is executed, you can add extra functionality(Advice) before or after the method execution.

**Advice**

This is the **actual action** to be taken either before or after the method execution. This is an actual piece of code that is invoked during the program execution by Spring AOP framework.

AOP enables you to dynamically add or remove concern before or after the business logic. Aspect-oriented programming. The core of AOP is an aspect

**Aspect**: is a class in spring AOP that contains **advices** and **joinpoints**.

**Advice**: represents action taken by aspect.

There are 5 types of advices in spring AOP.

1. Before Advice
2. After Advice
3. After Returning Advice
4. **Throws Advice**
5. **Around Advice**

### ****JoinPoint****

A point during the execution of a program is called JoinPoint, such as the execution of a method or the handling of an exception. **In Spring AOP, a joinpoint always represents a method execution.**

# **Spring MVC Tutorial:**

A Spring MVC provides an elegant solution to use MVC in spring framework by the help of **DispatcherServlet**.



* **Model** - A model **class** contains the data of the application. A data can be a single object or a collection of objects.
* **Controller** – is a **interface** ,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 **page** 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.

The **DispatcherServlet** is the core of Spring Web MVC framework. It handles all the HTTP requests and responses.