Spring Overview

Spring Architecture

Dependency Injection

Spring Container

Configuration file

Application Context

POJO Classes

Dependency Injection

Setter Injection

Constructor Injection

Autowire

Annotation Configuration

Component Scan

Stereo type annotation - @Component, @Service, @Repository, @Controller

**Spring & Spring Boot**

Spring Overview

* Application development framework for Enterprise Java
* Spring core- Java applications, Web applications…
* POJO(Plain Old Java Object) Based programming model - easy development of J2EE applications
* Spring Web framework - MVC(Model View Controller) framework

**Inversion Of Control (IOC) and Dependency Injection**

IoC refers to the programming style where a framework or runtime, controls the program flow. Inversion of control means we are changing the control from normal way.

These are the design patterns that are used to remove dependency from the programming code. They make the code easier to test and maintain. Let's understand this with the following code:

class Employee{

Address address;

Employee(){

address=new Address();

}

}

In such case, there is dependency between the Employee and Address (tight coupling). In the Inversion of Control scenario, we do this something like this:

class Employee{

Address address;

Employee(Address address){

this.address=address;

}

}

Thus, IOC makes the code loosely coupled. In such case, there is no need to modify the code if our logic is moved to new environment.

In Spring framework, IOC container is responsible to inject the dependency. We provide metadata to the IOC container either by XML file or annotation.

#### Advantage of Dependency Injection

* makes the code loosely coupled so easy to maintain
* makes the code easy to test

### Advantages of Spring Framework

#### 1) Predefined Templates - Spring framework provides templates for JDBC, Hibernate(ORM), JPA etc. technologies. So there is no need to write too much code. It hides the basic steps of these technologies.

#### 2) Loose Coupling - The Spring applications are loosely coupled because of dependency injection.

#### 3) Lightweight - Spring framework is lightweight because of its POJO implementation. The Spring Framework doesn't force the programmer to inherit any class or implement any interface.

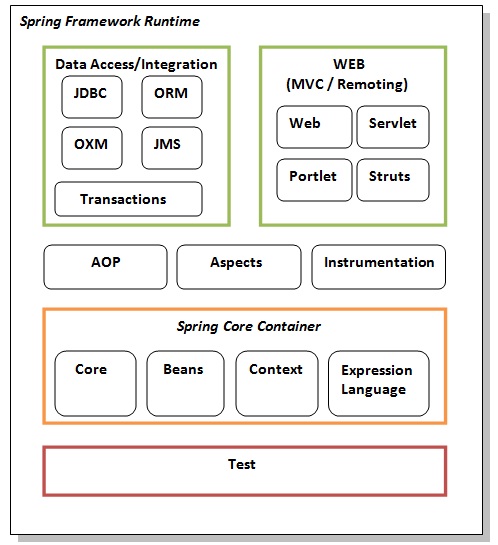
@spingboot

@GetRequest

#### 4) Fast Development - The Dependency Injection feature of Spring Framework and it support to various frameworks makes the easy development of JavaEE application.

#### 5) Declarative support - It provides declarative support for caching, validation, transactions and formatting.

Spring Architecture



pom.xml - spring-core spring-context

### Test - This layer provides support of testing with JUnit, Mokito….

### Spring Core Container

The Spring Core container contains core, beans, context and expression language (EL) modules.

#### Core, Beans and context - These modules provide IOC and Dependency Injection features.

#### Expression Language- 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.

### AOP, Aspects and Instrumentation ( logging - bank example - wthdraw, deposit log -time)

### An aspect is a common feature that's typically scattered across methods, classes, object hierarchies, or even entire object models.

### Bank example --- withdraw amount from ATM -Business logic

### Log details, security,Performance monitoring, transaction management

The concern is the behavior we want to have in a particular module of an application. It can be defined as a **functionality** we want to implement. ... For example, logging, security and data transfer are the concerns needed in almost every module of an application, thus they are the **cross**-**cutting** concerns.

### Data Access / Integration

This group comprises of JDBC, ORM, OXM, JMS and Transaction modules. These modules basically provide support to interact with the database

### Web- (MVC)

This group comprises of Web, Web-Servlet, Web-Struts. These modules provide support to create web application

**Spring Bean**

**Bean: is an object, which is created, managed and destroyed in Spring Container. We can inject an object into the Spring Container through the metadata(either xml or annotation), which is called inversion of control.**

Simple Hello World using Spring and Maven

1. Create a Maven Project

Group id - dura.com

artifact id - helloworld

2. In pom.xml - add spring dependency

<project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<groupId>dura.com</groupId>

<artifactId>HelloWorld</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>HelloWorld</name>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>${spring.version}</version>

</dependency>

</dependencies>

<properties>

<spring.version>3.2.3.RELEASE</spring.version>

</properties>

</project>

**3. Create the Spring Bean Configuration File**

Create a bean configuation file named "applicationContext.xml" under src/main/resources directory. This xml file specifies the bean you will define in the project. Note the class is the fully qualified class name with the package.

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=[*http://www.w3.org/2001/XMLSchema-instance*](http://www.w3.org/2001/XMLSchema-instance)>

<context:component-scan base-package=*"dura.com"* />

<bean id=*"helloWorldService"*

class=*"dura.com.HelloWorldService"*>

<property name=*"name"* value=*"This is CGI Readers"* />

</bean>

</beans>

**4. Create a Spring Bean**

Create HelloWorldService Bean under /src/main/java directory. The package should be the same with how you declared in the bean configuration file.

package dura.com;

import org.springframework.stereotype.Service;

@Service("helloWorldService")

public class HelloWorldService {

private String name;

public void setName(String name) {

this.name = name;

}

public String sayHello() {

return "Hello! " + name;

}

}

**5.**

Create a Hello class under /src/main/java directory to test the project.

package dura.com;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import dura.com.HelloWorldService;

public class Hello {

public static void main(String[] args) {

// loading the definitions from the given XML file - Loosely Coupled

ApplicationContext context = new ClassPathXmlApplicationContext(

"applicationContext.xml");

HelloWorldService service = (HelloWorldService) context

.getBean("helloWorldService");

String message = service.sayHello();

System.*out*.println(message);

//set a new name - Tightly coupled

service.setName("Welcome"); Hello Spring …… Hello Welcome……

message = service.sayHello();

System.*out*.println(message);

}

}

Right click Hello.java and run it as Java Application.