Spring – I

Spring Basics

Table of Contents

1. Introduction
2. Spring Modules
3. Spring Installation(Maven Dependencies)
4. IOC container
5. BeanFactory
6. ApplicationContext
7. First Application

Introduction

1. The Spring Framework is a Java Platform that is used for developing Java applications.
2. It is developed by Rod Johnson in 2003. It's initial name was interface21(2002).
3. Spring Framework is a product of Pivotal Software, Inc.
4. We can develop standalone application , Web application and Enterprise Applications using Spring Framework.
5. Spring Framework is an open source and light-weight framework.
6. It is a non-invasive framework. Non-invasive means spring framework does not force to implement any interface or extend any class for developing any application.
7. Spring Framework is a alternative to EJB.
8. It uses POJO classes (Plain Old Java Object).

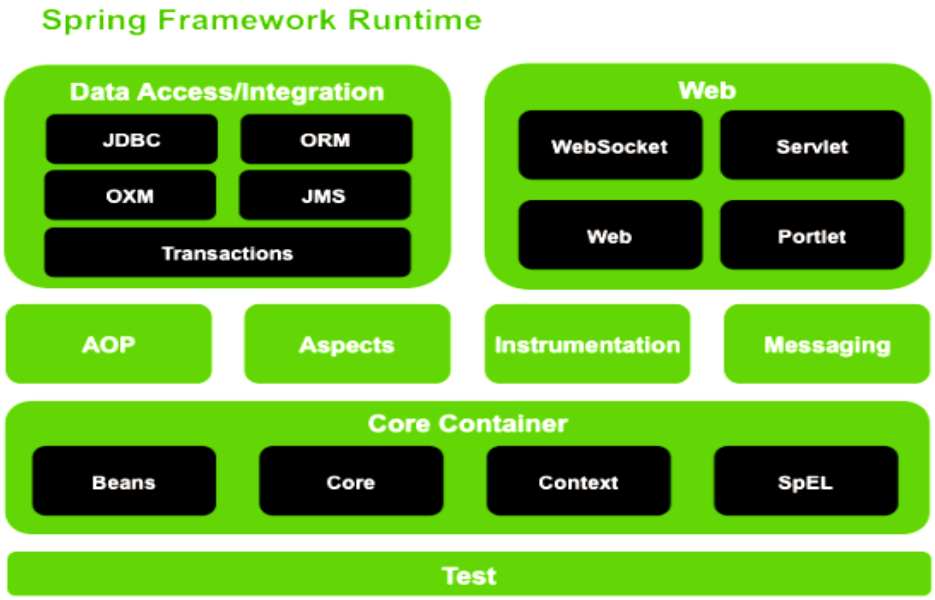
Features of Spring Framework

1. Dependency Injection and loose coupling
2. Aspect-Oriented Programming including Spring's declarative transaction management
3. Spring MVC web application and RESTful web service framework
4. Foundational support for JDBC, JPA, JMS

Spring Modules

The Spring Framework consists about 20 modules. These modules are grouped into six modules. These are -

1. **Core Container**
2. **AOP(Aspect Oriented Programming) And Instrumentation**
3. **Messaging**
4. **Data Access/Integration**
5. **Web**
6. **Test**



1. Core Container

The Core Container consists of **spring-core**, **spring-beans**, **spring-context** and **spring-expression(Spring Expression Language)**.

- ✓ The **spring-core** and **spring-beans** modules provides the fundamental part of the framework, like **IoC Container** and **Dependency Injection** features.
- ✓ The **spring-context** module builds on the top of the **spring-core** and **spring-beans**modules.
- ✓ The **spring-context** module inherits its feature from **spring-beans** module. The **ApplicationContext** Interface is the Central point of the **spring-context** module.
- ✓ The **spring-expression** module provides **Expression Language** for querying and manipulating object graph at Runtime. It is just like **JSP Expression Language**.

2. AOP(Aspect Oriented Programming) And Instrumentation

- ✓ The **spring-aop** module provides an **aspect-oriented programming** implementation that allowing you to define **method interceptors** and **pointcuts** to decouple code that implements functionality that should be separated.
- ✓ The separate **spring-aspects** module provides integration with **AspectJ**.
- ✓ The **spring-instrument** module provide class instrumentation support and **classloader** implementations to be used in certain Application Server.

3. Messaging

- ✓ The Spring Framework 4 include a **spring-messaging** module for developing **messaging-based**applications. It contains a set of **annotations** for mapping messages to methods.

4. Data Access/Integration

- ✓ The Data Access/Integration layer consists JDBC, ORM, OXM, JMS and Transaction modules.
- ✓ The **spring-jdbc** module provides a **JDBC-abstraction** layer to remove slow JDBC coding and parsing of database-vendors specific error codes.
- ✓ The **spring-tx** module supports programmatic and declarative transaction management.
- ✓ The **spring-orm** module provides a integration layer with different-different **ORM Tools**like- **Hibernate, JPA, JDO** etc.
- ✓ The **spring-oxm** module provides an abstraction layer that supports **Object/XML** Mapping implementations with **JAXB, XMLBeans, JiBX, XStream** and **Castor**.
- ✓ The **spring-jms** (Java Messaging Services) module supports for producing and consuming messages. It provides integration with **spring-messaging** module.

5. Web

- ✓ The Spring Web layer consists of **spring-web, spring-webmvc, spring-websocket** and **spring-webmvc-portlet**.
- ✓ The **spring-web** module provides basic web-oriented features like- **multipart** file upload, initialization of IoC container using servlet listeners and web-oriented application context.
- ✓ The **spring-webmvc**(also known as Web-Servlet module) module provides Spring's Web MVC(Model-View-Controller) implementation for web application.
- ✓ The **spring-webmvc-portlet**(also known as Web-Portlet module) provides MVC implementation to be used in **Portlet** environment.

6. Test

- ✓ The **spring-test** module supports **Unit Testing** and **Integration Testing** of the spring components with **JUnit** and **TestNG**.

Current Version and requirements

- ✓ Current Stable **version 4.3.8.RELEASE**
- ✓ Preview release **version 5.0.0.RELEASE**
- ✓ Here we are using spring **version 4.3.5.RELEASE**

Minimum Requirements of spring framework 4.X

- ✓ **Java 6+**
- ✓ **Servlet 2.5+**

Maven

Maven is a project management and comprehension tool that can manage whole project in a single file i.e. pom.xml(project object model).

It defines the source directory, target directory, properties and project dependencies.

Spring Framework Maven Dependencies

Shared Version number properties

```
<! – Shared version number properties -->
<properties>
        <org.springframework.version>4.3.5.RELESE</org.springframework.version>
</properties>
```

Spring Core **Description:** Core utilities used by all modules. **Depends on**: None **API's**: org.springframework.core.*, org.springframework.util.*; **Dependencies** <dependency> <groupId> **org.springframework** </groupId> <artifactId> **spring-core** </artifactId> <version> **${spring.version}** </version> </dependency>	**Spring Bean** **Description:** Bean Factory and JavaBeans utilities **Depends on**: spring-core **API's**: org.springframework.beans.* **Dependencies** <dependency> <groupId> **org.springframework** </groupId> <artifactId> **spring-beans** </artifactId> <version> **${spring.version}** </version> </dependency>
Spring Expression Language **Description:** Expression Language **Depends on**: spring-core **API's**: org.springframework.expression.*; **Dependencies** <dependency> <groupId> **org.springframework** </groupId> <artifactId> **spring-expression** </artifactId> <version> **${spring.version}** </version> </dependency>	**Spring AOP Framework** **Description:** Aspect Oriented Programming **Depends on**: Spring-core, spring-beans **API's**: org.springframework.aop.* **Dependencies** <dependency> <groupId> **org.springframework** </groupId> <artifactId> **spring-aop** </artifactId> <version> **${spring.version}** </version> </dependency>
Spring Application Context **Description:** This is the central artifact for spring's Dependency Injection Container and is generally always defined. **Depends on**: spring-core, spring-expression, spring-	**Spring Application Context utilities** **Description:** Various Application Context utilities EhCache, JavaMail, Quartz, and Freemarker Integration.

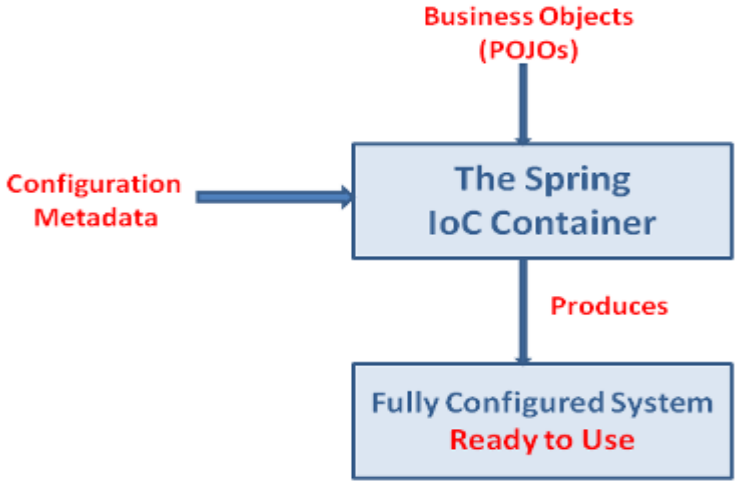
aop, and spring-beans **API's**: ApplicationContext (depends on spring-core, spring-expression, spring-aop, spring-beans) **Dependencies** <dependency> <groupId> **org.springframework** </groupId> <artifactId> **spring-context** </artifactId> <version> **${spring.version}** </version> </dependency>	**Dependencies** <dependency> <groupId> **org.springframework** </groupId> <artifactId> **spring-context-support** </artifactId> <version> **${spring.version}** </version> </dependency>
Spring Transaction **Description:** Transaction Management Abstraction **Depends on**: spring-core, spring-beans, spring-aop spring-context **API's**: org.springframework.transaction.*, org.springframework.dao.*; **Dependencies** <dependency> <groupId> **org.springframework** </groupId> <artifactId> **spring-tx** </artifactId> <version> **${spring.version}** </version> </dependency>	**Spring JDBC Template** **Description:** JDBC Data Access Library **Depends on**: spring-core, spring-beans, spring-context, spring-tx **API's**: org.springframework.jdbc.* **Dependencies** <dependency> <groupId> **org.springframework** </groupId> <artifactId> **spring-jdbc** </artifactId> <version> **${spring.version}** </version> </dependency>

IOC Container (Inversion of Control)

- In spring, an object is created, instantiated, assembled, and managed by a Spring IoC container.
- IoC gets the entry from the spring configuration file and works accordingly.
- This Configuration file may be XML **or** Java file **or** Annotations.
- IoC Container is also known as **Dependency Injection** (DI).
- We can say DI is a process where objects define their dependencies.
- The org.springframework.beans and **org.springframework.context** packages are the basis for Spring IoC container.
- **BeanFactory** and **ApplicationContext** interfaces provide the functionality of IoC container.

Spring Framework has mainly three IoC containers-

1. BeanFactory
2. ApplicationContext
3. WebApplicationContext



BeanFactory

- BeanFactory is a root interface of IoC container.
- The implementation is provided by**org.springframework.beans.factory.xml.XmlBeanFactory**.
- Normally a BeanFactory will load bean definitions defined in a configuration file (such as an XML file), and use the org.springframework.beans package to configure the beans.
- However, an implementation could simply return Java objects it creates as necessary directly in Java code.
- There are no constraints on how the definitions could be stored: LDAP, RDBMS, XML, properties file, etc. Implementations are encouraged to support references amongst beans **(Dependency Injection).**

Employee.java

```
package com.Biditvats.domain;

public class Employee {

   private int empId;
   private String empName;
   private String email;
   private float salary;

   public Employee(){}

   public Employee(int empId, String empName, String email, float salary) {
      this.empId = empId;
      this.empName = empName;
      this.email = email;
      this.salary = salary;
   }

   public int getEmpId() {
      return empId;
   }
   public void setEmpId(int empId) {
      this.empId = empId;
   }
   public String getEmpName() {
      return empName;
   }
   public void setEmpName(String empName) {
      this.empName = empName;
   }
   public String getEmail() {
      return email;
   }
   public void setEmail(String email) {
      this.email = email;
   }
   public float getSalary() {
      return salary;
   }
   public void setSalary(float salary) {
      this.salary = salary;
```

```
    }
}
```

Application.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="employee" class="com.Biditvats.domain.Employee">
        <property name="empId" value="1001" />
        <property name="empName" value="CP Verma" />
        <property name="email" value="CPVerma@gmail.com" />
        <property name="salary" value="95000.00" />
    </bean>

</beans>
```

TestEmployee.java

```
package com.Biditvats.test;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;

import com.Biditvats.domain.Employee;

public class TestEmployee {

    public static void main(String[] args) {

        Resource resource = new ClassPathResource("Application.xml");
        BeanFactory factory = new XmlBeanFactory(resource);

        Employee emp = (Employee)factory.getBean("employee");

        System.out.println("Employe ID: "+emp.getEmpId());
        System.out.println("Employe Name: "+emp.getEmpName());
        System.out.println("Employe Email: "+emp.getEmail());
        System.out.println("Employe Salary: "+emp.getSalary());
    }
}
```

Using ApplicatioContext implemented Class ClassPathXmlApplicationContext

We have to change only inside the **TestEmployee.java** file :

TestEmployee.java

```
package com.Biditvats.test;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClasspathxmlApplicationContext;
```

```
import com.Biditvats.domain.Employee;

public class TestEmployee {

   public static void main(String[] args) {

      ApplicationContext context = new ClassPathXmlApplicationContext(“Application.xml”);

      Employee emp = (Employee)context.getBean("employee");

      System.out.println("Employe ID: "+emp.getEmpId());
      System.out.println("Employe Name: "+emp.getEmpName());
      System.out.println("Employe Email: "+emp.getEmail());
      System.out.println("Employe Salary: "+emp.getSalary());
   }
}
```