Welcome

Spring Framework

Spring Basics

Overview of Spring Framework

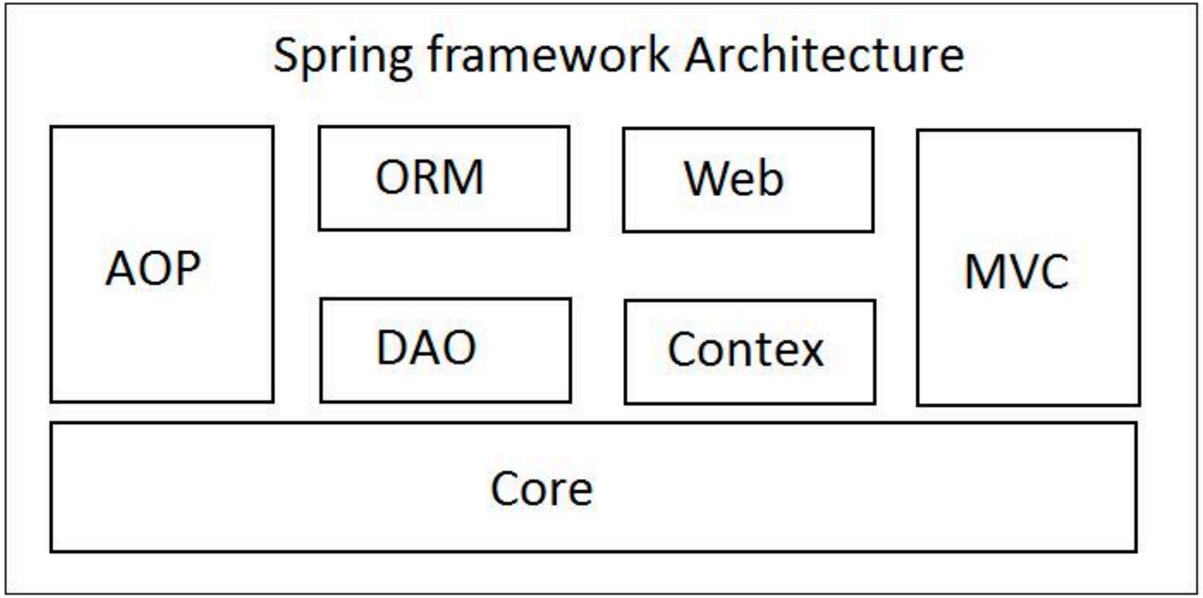
- Spring is the lightweight open source framework
- Simplifies the development of java based enterprise applications
- It was created by Rod Johson and was described in his book "*Expert One-on-One: J2EE Design and Development*"
- It helps in developing loosely coupled and highly cohesive systems
- Loose coupling is achieved by Spring's Inversion of Control (IoC) feature
- High cohesion is achieved by Spring's Aspect oriented programming (AOP) feature

Spring Architecture

- Spring framework provides a number of features which are required for developing an enterprise application
- It does not enforce the developers to integrate their application with the complete framework
- The various features provided by Spring framework are categorized in seven different modules.

1. Core Container
2. Application Context
3. AOP
4. Spring Web
5. Spring MVC
6. Spring DAO
7. Spring ORM

Spring Architecture (Contd..)



Spring Architecture (Contd..)

- Core Container
  - It is the heart of Spring framework and all other modules are built on top of it
  - It provides the dependency injection feature, also is known as inversion of control
  - It contains the BeanFactory which creates and manages the life cycle of the various application objects

- Application Context
  - This module provides various enterprise level services *internationalisation (i18n)*, scheduling, JNDI access, email etc.

Spring Architecture (Contd..)

- AOP
  - This module helps in implementing the various cross cutting concerns in the application like logging, transaction management etc.
  - These concerns are decoupled from the application code and are injected into the various point cuts through configuration file.

- Spring Web
  - Spring framework helps in developing web based application by providing the Spring web module.
  - This module is built on top of application context module and provides web oriented features.

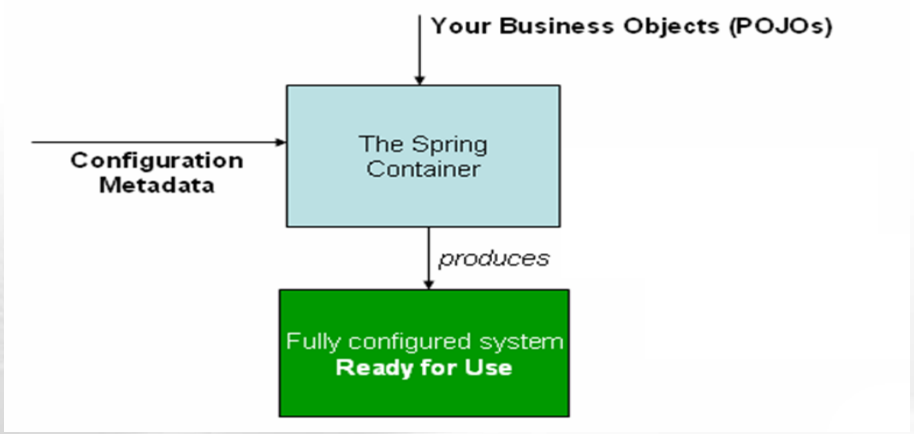
Spring Architecture (Contd..)

- **Spring MVC**
  - Spring MVC module is built on top of Spring web module and helps in developing web application based on MVC design pattern.
- **Spring DAO**
  - Spring DAO module makes it easy to interact with database by providing an abstraction over low level JDBC tasks like creating a database connection, release it etc.
- **Spring ORM**
  - There exist a number of popular object-relational mapping tools like Hibernate,iBatis,JPA etc.
  - Spring ORM module helps in integrating with these tools.

Spring IoC Containers

- The Spring IoC container is at the core of the Spring Framework
- The container will create the objects, wire them together, configure them, and manage their complete life cycle from creation till destruction
- The Spring container uses dependency injection (DI) to manage the components that make up an application

Spring IoC Containers (Contd..)



Spring IoC Containers (Contd..)

- **Types of Containers**

  1. **BeanFactory Container**
  2. **ApplicationContext Container**

BeanFactory Container

- BeanFactory is essentially nothing more than the interface for an advanced factory capable of maintaining a registry of different beans and their dependencies.
- The BeanFactory enables us to read bean definitions and access them using the bean factory.

BeanFactory Container (Contd..)

- **Create BeanFactory**

```
InputStream is = new FileInputStream(“beans.xml");
BeanFactory factory = new XmlBeanFactory(is);

//Get bean
Employee obj = (Employee) factory.getBean(“employee”);
```

- **Other way to create bean factory**

```
Resource resource = new FileSystemResource(“beans.xml");
BeanFactory factory = new XmlBeanFactory(resource);

ClassPathResource resource = new ClassPathResource(“beans.xml");
BeanFactory factory = new XmlBeanFactory(resource);

//Get bean
Employee obj = factory.getBean(“employee”, Employee.class);
```

BeanFactory Methods

- **The BeanFactory interface has only six methods for client code to call**

1. **boolean containsBean(String):** returns true if the BeanFactory contains a bean definition or bean instance that matches the given name
2. **Object getBean(String):** returns an instance of the bean registered under the given name
3. **Object getBean(String, Class):** returns a bean, registered under the given name. The bean returned will be cast to the given Class.
4. **Class getType(String name):** returns the Class of the bean with the given name
5. **boolean isSingleton(String):** determines whether or not the bean definition or bean instance registered under the given name is a singleton
6. **String[] getAliases(String):** Return the aliases for the given bean name, if any were defined in the bean definition

ApplicationContext Container

- ApplicationContext container adds more enterprise-specific functionality
  - The ability to resolve textual messages from a properties file
  - The ability to publish application events to interested event listeners.
- This container is defined by the org.springframework.context.ApplicationContext interface.
- The *ApplicationContext* container includes all functionality of the *BeanFactory* container, so it is generally recommended over the *BeanFactory*.
- BeanFactory can still be used for lightweight applications like mobile devices or applet based applications where data volume and speed is significant.

Types of ApplicationContext Container

- **The most commonly used ApplicationContext implementations are:**

  1. **FileSystemXmlApplicationContext -** This container loads the definitions of the beans from an XML file. Here you need to provide the full path of the XML bean configuration file to the constructor.
  2. **ClassPathXmlApplicationContext -** This container loads the definitions of the beans from an XML file. Here you do not need to provide the full path of the XML file but you need to set CLASSPATH properly because this container will look bean configuration XML file in CLASSPATH.
  3. **WebXmlApplicationContext -** This container loads the XML file with definitions of all beans from within a web application.

ApplicationContext (Contd..)

- **Create FileSystemXmlApplicationContext**

```
ApplicationContext context = new FileSystemXmlApplicationContext(“D:\\beans.xml”);

Organization org = context.getBean("organization", Organization.class);
```

- **Create ClassPathXmlApplicationContext**

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

Organization org = context.getBean("organization", Organization.class);
```

ApplicationContext Example

Address.java

```
package com.emexo.spring.ioc.applicationcontext.classpath;

public class Address {
    private String street = "Neeladri Nager";
    private String city = "Bangalore";
    private String state = "Karnataka";

    public void getAddress(){
        System.out.println(street);
        System.out.println(city);
        System.out.println(state);
    }
}
```

ApplicationContext Example (contd..)

beans.xml

```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"
     xmlns:context="http://www.springframework.org/schema/context"
     xsi:schemaLocation="http://www.springframework.org/schema/beans
     http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
     http://www.springframework.org/schema/context
     http://www.springframework.org/schema/context/spring-context-3.0.xsd">

     <bean id="address"
class="com.emexo.spring.ioc.applicationcontext.classpath.Address" />

</beans>
```

ApplicationContext Example (contd..)

Main.java

```
package com.emexo.spring.ioc.applicationcontext.classpath;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class TestMain {
    public static void main(String[] args) {
        ApplicationContext context = new
ClassPathXmlApplicationContext("beans.xml");

        Address address = context.getBean("address", Address.class);
        address.getAddress();

    }
}
```

ApplicationContext Example (contd..)

O/P:

```
15:23:11.277 [main] DEBUG o.s.b.f.s.DefaultListableBeanFactory - Creating shared
instance of singleton bean 'address'
15:23:11.278 [main] DEBUG o.s.b.f.s.DefaultListableBeanFactory - Creating instance
of bean 'address'
15:23:11.288 [main] DEBUG o.s.b.f.s.DefaultListableBeanFactory - Eagerly caching
bean 'address' to allow for resolving potential circular references
15:23:11.290 [main] DEBUG o.s.b.f.s.DefaultListableBeanFactory - Finished creating
instance of bean 'address'
15:23:11.292 [main] DEBUG o.s.b.f.s.DefaultListableBeanFactory - Returning cached
instance of singleton bean 'lifecycleProcessor'
15:23:11.307 [main] DEBUG o.s.b.f.s.DefaultListableBeanFactory - Returning cached
instance of singleton bean 'address'
Neeladri Nager
Bangalore
Karnataka

Process finished with exit code 0
```