
ACADGILD
LEARN. DO. EARN
J2EE

ACADGILD

Introduction to Spring

Session - 22

LEARN. DO. EARN

Agenda

ACADGILD

Sl No	Agenda Title
1	What is Spring?
2	Why Spring?
3	Spring - J2EE Application Server?
4	Features of Spring
5	Architecture of Spring
6	Core Spring
7	BeanFactory Vs ApplicationContext
8	Spring Core API

Sl No	Agenda Title
9	Resource
10	Resource – Sample Code
11	Bean Factory
12	BeanFactory and Resource Example
13	Sample Application
14	Bean Definition Configuration File
15	Inversion of Control (IoC)
16	Dependency Injection

What is Spring?

ACADGILD

- **Spring** is the most popular application development framework for enterprise Java.

- The current version is *Spring Framework 4.0*, which was released in December 2013.

- Spring is a popular open source **application framework** that can make J2EE development easier by enabling a **POJO**-based programming model.

- Consists of:
  - a container
  - a framework for managing components
  - and a set of snap-in services for:
    - web user interfaces
    - Transactions
    - and persistence

Why Spring?

ACADGILD

- The essence of Spring is in providing enterprise services to **Plain Old Java Objects (POJO's)**.
- Applications built using Spring are very easy to unit test.
- **Spring** can eliminate the need to use a variety of custom properties file formats by handling configuration in a consistent way throughout applications and projects.
- **Spring** can eliminate the need to use a variety of custom properties file formats, by handling configuration in a consistent way throughout applications and projects.
- **Spring** can provide an alternative to EJB that's appropriate for many applications.
- **Spring** provides a consistent framework for data access, whether using JDBC or an O/R mapping product such as TopLink, Hibernate or a JDO implementation.

Spring - J2EE Application Server?

ACADGILD

- **Spring** is NOT a J2EE Application Server.
- **Spring** can nicely integrate with J2EE Application Servers (or any Java Environment).
- **Spring** can elegantly replace the services traditionally provided by J2EE Application Server.
- **Spring** provides elegant integration points with :
  - JDO
  - EJB
  - RMI
  - Web Services
  - JMS
  - Hibernate

Features of Spring

ACADGILD

- It is a lightweight framework.
- There is no dependency on the framework.
- Spring does not reinvent the wheel. Instead it makes all the existing solutions easier to use.
- Spring is based on Dependency Injection flavor of Inversion of Control.
- Spring includes a proxy based AOP (Aspect Oriented Programming) framework.
- It integrates with a variety of web frameworks like Struts, WebWork, Spring MVC, Tapestry, JSP, etc.

Architecture of Spring

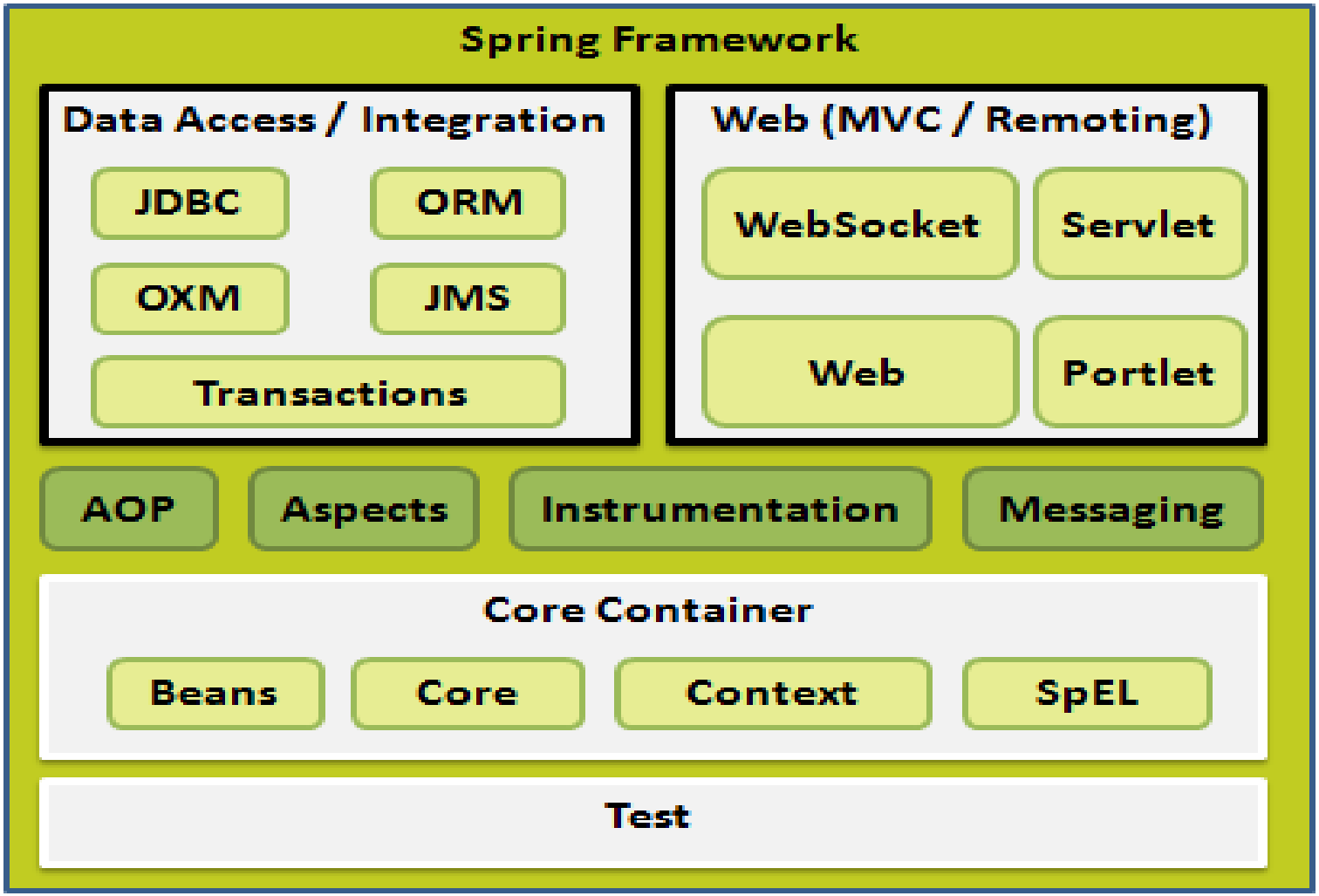
ACADGILD

- Spring is well organized architecture consisting of seven modules.

- Modules in the Spring Framework are:
  - Spring AOP
  - Spring ORM
  - Spring Web
  - Spring DAO
  - Spring Context
  - Spring Web MVC
  - Spring Core

Architecture of Spring (Contd.)

ACADGILD



Core Spring

ACADGILD

- The Core Spring can be thought of as a Framework and a Container for managing Business Objects and their relationship.

- The Beauty of the Framework is that, in most of the times we don't need to depend on Spring specific Classes and Interfaces.
  - This is unlike other Frameworks, where they will force the Client Applications to depend on their propriety implementations.

- Business Components in Spring and POJO (Plain Old Java Object) or POJI (Plain Old Java Interface).

BeanFactory Vs ApplicationContext

ACADGILD

- There are two ways in which clients can use the functionality of the Spring Framework:
  - **BeanFactory**
  - **ApplicationContext**

- Two of the most fundamental and important packages in Spring are:
  - **org.springframework.beans**
  - **org.springframework.context**

- Code in these packages provides the basis for Spring's *Inversion of Control* – **IOC** (alternately called **Dependency Injection**) features.

- The **BeanFactory** provides an advanced configuration mechanism - capable of managing beans (objects) of any nature, using potentially any kind of storage facility.

BeanFactory Vs ApplicationContext (Contd.)

ACADGILD

- The **ApplicationContext** builds on top of the **BeanFactory** and adds other functionality such as :
  - easier integration with Springs AOP features
  - message resource handling (for use in internationalization)
  - event propagation
  - declarative mechanisms to create the ApplicationContext
  - application-layer specific contexts such as the WebApplicationContext

- In short,
  - the **BeanFactory** provides the configuration framework and basic functionality
  - the **ApplicationContext** adds enhanced capabilities to it, some of the m perhaps more J2EE and enterprise-centric.

Spring Core API

ACADGILD

- The **Core API** in Spring is very limited and it generally involves in:
  - **Configuring**
  - **Creating**
  - and **Making Associations**

  between various **Business Components**.

- **Spring** refers to these Business Components as **Beans**.

- The following are the Core Classes or the Interfaces that are available in Spring for achieving the goal.
  - **Resource**
  - **BeanFactory**

Resource

ACADGILD

- Interface for a resource is descriptor that abstracts from the actual type of underlying resource, such as a file or class path resource.

- **Package:** org.springframework.core.io

- Various classes which provides concrete implementation of 'Resources' are:
  - FileSystemResource
  - ClassPathResource
  - UrlResource
  - ByteArrayResource
  - InputStreamResource
  - ServletContextResource

Resource – Sample Code

ACADGILD

Example

```
String xmlFile = "./resources/myXml.xml");

Resource xmlResource = new FileSystemResource(xmlFile);
```

Bean Factory

ACADGILD

- The **BeanFactory** provides an advanced configuration mechanism capable of managing *beans* (objects) of any nature.

- The **BeanFactory** is the actual **container** which instantiates, configures, and manages a number of beans.
  - These beans typically collaborate with one another, and thus have dependencies between themselves.
  - These dependencies are reflected in the configuration data used by t he BeanFactory

- A BeanFactory is represented by the interface **org.springframework.beans.factory.BeanFactory**, for which there are multiple implementations.
  - The most commonly used simple BeanFactory implementation is **org.springframework.beans.factory.xml.XmlBeanFactory**

BeanFactory and Resource Example

ACADGILD

Example:

```
Resource xmlResource = new FileSystemResource ("beans.xml");

BeanFactory factory = new XmlBeanFactory(xmlResource);
```

Sample Application

ACADGILD

- In this simple application, which is based on "Spring", we create the following:
  - **Business Object** (Namer.java)
    - Business Component 'Namer', which is used to store the given name
  - **XML Configuration file** (namer.xml)
    - to define and configure the Bean class along with its properties
  - **Client program** (SimpleSpringApp.java)
    - which makes reference to the Xml File using the Resourceobject
    - and then the contents of the Xml File are read using the XmlBeanFactory class
    - An instance of the object of type Namer is then retrieved by calling the BeanFactory.getBean(id) method

Sample Application (Contd.)

ACADGILD

Namer.java

```
public class Namer
 {
private String name;
public Namer()
   { }
public String getName()
   { return name;
   }
public void setName(String name)
   { this.name = name;
   }
}
```

Sample Application (Contd.)

ACADGILD

Namer.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-
  2.0.xsd">
<bean id="namerId" class="Namer">
    <property name = "name">
      <value>Steve</value>
    </property>
</bean>
</beans>
```

Sample Application (Contd.)

ACADGILD

SimpleSpringApp.java (Client code)

```
import org.springframework.beans.factory.*;
import org.springframework.beans.factory.xml.*;
import org.springframework.core.io.*;

public class SimpleSpringApp {
public static void main(String args[]){
Resource namerXmlFile = new FileSystemResource("src/
    namer.xml");
BeanFactory factory = new XmlBeanFactory(namerXmlFile);
Namer namer = (Namer)factory.getBean("namerId");
 System.out.println(namer.getName());
}
}
```

Bean Definition Configuration File

ACADGILD

- All the basic definition of the Bean classes along with the Configuration Information, their relationships with other Bean objects can be defined in the **Xml Configuration File**

- The major configuration features :
  - Making Associations between Bean Objects
    - using the 'ref' element through the property bean
  - Mapping Collection Properties
    - By including various Collection properties
  - Importing Configuration Files into a master xml file

    E.g.:
```
<beans>
        <import resource = "emp.xml"/>
        <import resource = "address.xml"/>
</beans>
```

Bean Definition Configuration File (Contd.)

ACADGILD

- **Bean Life Cycle**
  - The entire Bean objects defined in the Xml Configuration File undergoes a Standard Lifecycle Mechanism
  - Lifecycle interfaces like **InitializingBean** and **DisposableBean** are available to enhance/modify the lifecycle.
  - The **InitializingBean** interface has a single method called **afterPropertiesSet()** which will be called immediately after all the property values that have been defined in the Xml Configuration file is set.
  - The **DisposableBean** has a single method called **destroy()** which will be called during the shut down of the Bean Container

- **Controlling the Order of Creation of Beans**
  - in a situation where Component A must be created before Component B, we can use the depends-on attribute which takes a list of previously defined Bean Definition identifiers

Source: http://www.javabeat.net/articles

Bean Definition Configuration File (Contd.)

ACADGILD

- **Creating Bean Instances through Factory classes**
  - If method in Factory class is a static method, then use ‘factory-method’ attribute
  - If method is a non-static method, then 'factory-method' and 'factory-bean' attributes must be used
- **Bean Inheritance**
  - Minimal support of Inheritance is given between the Bean Components in the form of the attribute 'parent' within the 'bean‘ tag

Inversion of Control (IoC)

ACADGILD

- The Framework insists that Associations between Business objects should be externalized
  - and the Client Applications should never be involved in doing these kinds of activities
- Component Wiring is a fancy term given to make associations between various Components
- Instead of Clients having the control to establish relationship between Components, now the Framework carries this job
  - which means that the Control is reversed from the Clients to the Framework
  - that's why this principle is rightly termed as **Inversion of Control**

Inversion of Control (Contd.)

ACADGILD

- Spring is most closely identified with a flavor of Inversion of Control known as **Dependency Injection**

- Dependency Injection is a form of IOC that removes explicit dependence on container APIs
  - ordinary Java methods are used to inject dependencies such as collaborating objects or configuration values into application object instances.

- The two major flavors of Dependency Injection are
  - **Setter Injection** (injection via JavaBean setters)
  - **Constructor Injection** (injection via constructor arguments)

Dependency Injection

ACADGILD

- Dependency Injection has several important benefits:
  - Because components don't need to look up collaborators at runtime, they're much simpler to write and maintain.
  - With a Dependency Injection approach, dependencies are explicit, and evident in constructor or JavaBean properties.
  - Easy to use objects either inside or outside the IoC container.
  - Spring also provides unique support for instantiating objects from static factory methods or even methods on other objects managed by the IoC container.
  - Your business objects can potentially be run in different Dependency Injection frameworks - or outside any framework - without code changes.

Dependency Injection (Contd.)

ACADGILD

- There are 2 types of Dependency Injection(DI) techniques that we can use:
  - **Setter Injection**
    - using setter methods in a bean class, the Spring IOC container will inject the dependencies
  - **Constructor Injection**
    - The constructor will take arguments based on number of dependencies required
    - You don't have option to reconfigure the dependencies at later point of time, since all the dependencies are resolved only at the time of invoking the constructor
    - E.g. -

```
<constructor-arg index="0" type="java.lang.String" value="MyName" />
```