



Marwadi
University
Marwadi Chandarana Group

AJT - 01CE0411

Unit 6

Spring Framework

Prepared By

Prof. Ravikumar Natarajan

Assistant Professor, CE Dept.

**KNOWLEDGE IS THE CURRENCY
FOR THE 21st CENTURY**

Contents



Marwadi
University
Marwadi Chandarana Group

- Spring: Introduction,
- Architecture,
- Spring MVC Module,
- Life Cycle of Bean Factory,
- Explore: Constructor Injection, Dependency Injection,
- Inner Beans, Aliases in Bean, Bean Scopes,
- Spring Annotations,
- Spring AOP Module, Spring DAO,
- Database Transaction Management,
- CRUD Operation using DAO and Spring API

Video Contents



Marwadi
University
Marwadi Chandarana Group

- Introduction to Spring | Spring Architecture Quick Overview | Advantages Spring Framework
- <https://youtu.be/0RxxiTP1AMw>
- Spring Inversion of Control (IoC), Dependency Injection (DI) Types | DI Constructor & Setter Demo
- <https://youtu.be/xWexKknwadw>
- Spring Annotations | NetBeans Demo
- <https://youtu.be/V4nNQG0m5ew>

Introduction to Spring



Marwadi
University
Marwadi Chandarana Group

- Spring was developed by **Rod Johnson in 2003**. Spring framework makes the easy development of JavaEE application.
- It is helpful for beginners and experienced persons.

Introduction to Spring



Marwadi
University
Marwadi Chandarana Group

Spring enables developers to develop **enterprise-class** applications using **POJOs**. The benefit of using only POJOs is that you do not need an EJB container product such as an **application server** but you have the option of **using only a** robust servlet container such as **Tomcat** or some commercial product .

Spring is organized in a **modular fashion**. Even though the number of packages and classes are substantial, you have to worry only about ones you need and ignore the rest.

Spring's web framework is a **well-designed web MVC framework**, which provides a great alternative to web frameworks such as Struts or other over engineered or less popular web frameworks.

Spring Framework



Marwadi
University
Marwadi Chandarana Group

- 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 framework, in broader sense, can be defined as a **structure where we find solution** of the various technical problems.
- The Spring framework comprises several modules such as IOC, AOP, DAO, Context, ORM, WEB MVC etc. We will learn these modules in next page. Let's understand the IOC and Dependency Injection first.

Spring Architecture



Marwadi
University
Marwadi Chandarana Group

Spring Framework

Data Access / Integration

JDBC

ORM

OXM

JMS

Transactions

Web (MVC / Remoting)

WebSocket

Servlet

Web

Portlet

AOP

Aspects

Instrumentation

Messaging

Core Container

Beans

Core

Context

SpEL

Test

The Spring Framework provides about 20 modules which can be used based on an application requirement.

OXM (Object XML Mapping),
JMS (Java Message Service),
Spring Expression Language (SpEL)

Spring Architecture



Marwadi
University
Marwadi Chandarana Group

Core Container:

The Core Container consists of the Core, Beans, Context, and Expression Language modules whose detail is as follows:

The **Core** module provides the fundamental parts of the framework, including the **IoC (Inversion of Control)** and Dependency Injection features.

The **Bean** module provides BeanFactory.

The **Context** module builds on the solid base provided by the Core and Beans modules and it is a **medium to access any objects** defined and configured.

The **SpEL (Spring Expression Language)** module provides a powerful expression language for querying and manipulating an object graph at runtime.

Spring Architecture

Data Access/Integration:

The Data Access/Integration layer consists of the JDBC, ORM, OXM, JMS and Transaction modules whose detail is as follows:

The **JDBC** module provides a JDBC-abstraction layer that removes the need to do tedious JDBC related coding.

The **ORM** module provides integration layers for popular object-relational mapping APIs, including JPA, JDO, Hibernate, and iBatis.

The OXM module provides an abstraction layer that supports Object/XML mapping implementations for JAXB, Castor, XMLBeans, JiBX and XStream. The Java Messaging Service **JMS** module contains features for producing and consuming messages.

The **Transaction** module supports programmatic and declarative transaction management for classes that implement special interfaces and for all your POJOs.

Spring Architecture



Marwadi
University
Marwadi Chandarana Group

Web:

- The Web layer consists of the Web, Web-MVC, Web-Socket, Portlet modules whose detail is as follows:
- The **Web** module provides basic web-oriented integration features such as multipart file-upload functionality and the initialization of the IoC container using servlet listeners and a web-oriented application context.
- The **Web-MVC** module contains Spring's **model-view-controller (MVC)** implementation for web applications.
- The **Web-Socket** module provides support for WebSocket-based, two-way communication between client and server in web applications.
- The **Web-Portlet** module provides the MVC implementation to be used in a **portlet environment** and **mirrors/render** the functionality of Web-Servlet module.

Spring Architecture



Marwadi
University
Marwadi Chandarana Group

Miscellaneous:

There are few other important modules like AOP, Aspects, Instrumentation, Web and Test modules whose detail is as follows:

The **AOP** module provides **aspect-oriented programming** implementation allowing you to define **method-interceptors** and **pointcuts** to cleanly decouple code that implements functionality that should be separated.

The **Aspects** module provides integration with AspectJ which is again a powerful and mature **aspect oriented programming (AOP)** framework.

The **Instrumentation** module provides class instrumentation support and class loader implementations to be used in certain application servers.

The **Messaging** module provides support for STOMP (**Streaming Text Oriented Messaging Protocol**) as the WebSocket sub-protocol to use in applications. It also supports an annotation programming model for routing and processing STOMP messages from WebSocket clients. The **Test** module supports the testing of Spring components with JUnit or TestNG frameworks.

Spring Advantages



Marwadi
University
Marwadi Chandarana Group

1) Predefined Templates

Spring framework provides templates for JDBC, Hibernate, JPA etc. technologies. So there is no need to write too much code. It hides the basic steps of these technologies.

Let's take the example of JdbcTemplate, you don't need to write the code for **exception handling, creating connection, creating statement, committing transaction, closing connection** etc. You need to write the code of executing query only. Thus, it save a lot of JDBC code.

Spring Advantages



Marwadi
University
Marwadi Chandarana Group

2) Loose Coupling

The Spring applications are loosely coupled because of dependency injection.

3) Easy to test

The Dependency Injection makes easier to test the application. The EJB or Struts application require server to run the application but Spring framework doesn't require server.

4) 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. That is why it is said non-invasive.

Spring Advantages



Marwadi
University
Marwadi Chandarana Group

5) Fast Development

The Dependency Injection feature of Spring Framework and its support to various frameworks makes the easy development of JavaEE application.

6) Powerful abstraction

It provides powerful abstraction to JavaEE specifications such as JMS, JDBC, JPA and JTA.

7) Declarative support

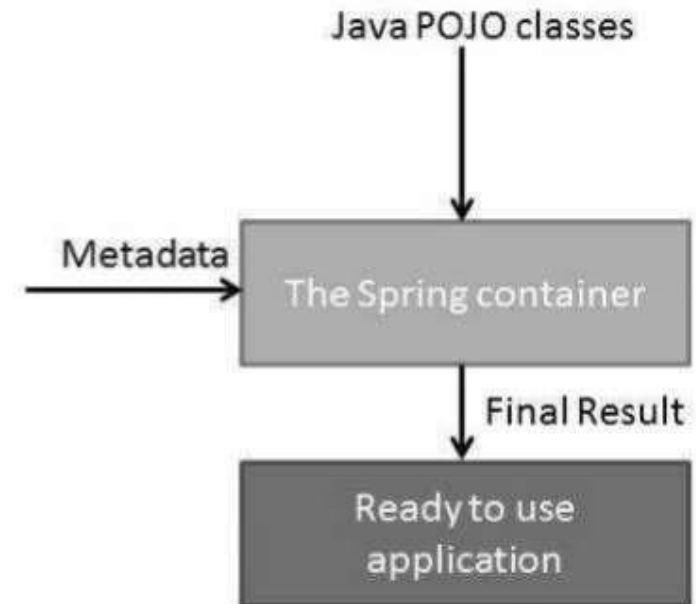
It provides declarative support for caching, validation, transactions and formatting.

Spring - IoC Containers (Inversion of Control)



Marwadi
University
Marwadi Chandarana Group

The Spring 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.





Spring provides the following two distinct types of containers.

1. Spring **BeanFactory Container**

This is the simplest container providing the **basic support for DI** and is defined by the ***org.springframework.beans.factory.BeanFactory* interface**. The BeanFactory and related interfaces, such as BeanFactoryAware, InitializingBean, DisposableBean, are still present in Spring for the purpose of backward compatibility with a large number of third-party frameworks that integrate with Spring.

2.Spring **ApplicationContext Container**

This container adds more **enterprise-specific** functionality such as the ability to resolve textual messages from a properties file and the ability to publish application events to interested **event listeners**. This container is defined by the ***org.springframework.context.ApplicationContext* interface**.

Inversion Of Control (IOC) and Dependency Injection



Marwadi
University
Marwadi Chandarana Group

These are the **design patterns** that are used to **remove dependency** from the programming code. They make the **code easier to test and maintain**. **Dependency Injection is the functionality provided by inversion of control (IoC).**

Let's understand this with the following code:

```
class Employee{  
private Address address;  
Employee(){  
address=new Address();  
}  
}
```

In such case, there is dependency between the Employee and Address (**tight coupling**).



Inversion Of Control (IOC) and Dependency Injection

In the Inversion of Control scenario, we do this something like this:

```
class Employee{  
    private 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**.

Advantages of Dependency Injection



Marwadi
University
Marwadi Chandarana Group

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

Dependency Injection



Marwadi
University
Marwadi Chandarana Group

DI exists in two major variants:

1. **Constructor-based dependency injection** Constructor-based DI is accomplished when the container invokes a class constructor with a number of arguments, each representing a dependency on the other class.
2. **Setter-based dependency injection** Setter-based DI is accomplished by the container calling setter methods on your beans after invoking a no- argument constructor or no-argument static factory method to instantiate your bean.

You can mix both, Constructor-based and Setter-based DI but it is a good rule of thumb to use constructor arguments for mandatory dependencies and setters for optional dependencies. The code is cleaner with the DI principle and decoupling is more effective when objects are provided with their dependencies.

Dependency Injection

Consider you have an application which has a text editor component and you want to provide a spell check. Your standard code would look something like this –

```
public class TextEditor
{
    private SpellChecker spellChecker;
    public TextEditor()
    { spellChecker = new SpellChecker(); }
}
```

What we've done here is, create a dependency between the TextEditor and the SpellChecker.

```
public class TextEditor
{
    private SpellChecker spellChecker;
    public TextEditor(SpellChecker spellChecker)
    { this.spellChecker = spellChecker; }
}
```

In an inversion of control scenario, the TextEditor should not worry about SpellChecker implementation. The SpellChecker will be implemented independently and will be provided to the TextEditor at the time of TextEditor instantiation. This entire procedure is controlled by the Spring Framework.

Spring MVC



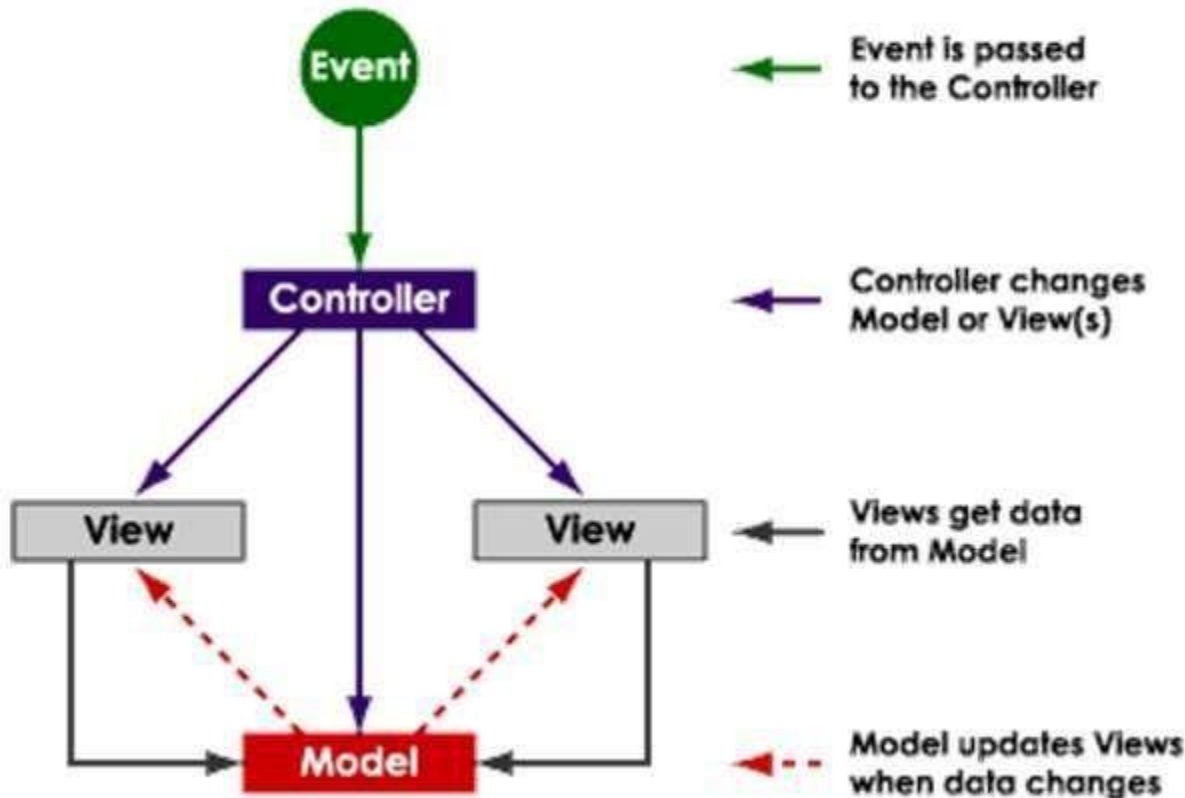
Marwadi
University
Marwadi Chandarana Group

MVC: instances of the following:

M – **Model:** Eg. Java Bean, POJO etc

V – **View:** org.springframework.web.servlet.**View**

C – **Controller:** org.springframework.web.servlet.mvc.**Controller**



Spring MVC



Marwadi
University
Marwadi Chandarana Group

MVC for Web

Event – HTTP request from client

Controller – Custom Spring Controller Class

View – Eg. JSP

Model –Eg Java Beans

Model is passed to the server-side View which is returned to the client

Spring MVC



Marwadi
University
Marwadi Chandarana Group

Spring MVC Lifecycle

Step 1: Incoming HTTP request is mapped to the Spring DispatcherServlet.

Step 2: The DispatcherServlet creates a container using the bean definitions found in the Servlet configuration file.

Step 3: The container finds an appropriate Controller to process the request.

Step 4: The Controller performs some custom logic and returns a ModelAndView to the container.

Step 5: The container renders the ModelAndView using the appropriate ViewResolver bean.

Step 6: An HTTP response is sent to the browser

Spring MVC Example..



Marwadi
University
Marwadi Chandarana Group

link.html

```
<html>
<body>
<a href="Link.spring">GO TO SPRING HOME PAGE</a>
</body>
</html>
```

Spring MVC



Marwadi
University
Marwadi Chandarana Group

Spring MVC Step 1

Incoming HTTP request is mapped to the Spring DispatcherServlet.

This is configured like any other Servlet mapping. Pick a URI-Pattern and associate it with the DispatcherServlet

web.xml

```
<servlet>
```

```
  <servlet-name>Dispatcher</servlet-name>
```

```
  <servlet-class>
```

```
org.springframework.web.servlet.DispatcherServlet
```

```
</servlet-class>
```

```
</servlet>
```

```
<servlet-mapping>
```

```
  <servlet-name>Dispatcher</servlet-name>
```

```
  <url-pattern>*.spring</url-pattern>
```

```
</servlet-mapping>
```

Spring MVC



Marwadi
University
Marwadi Chandarana Group

Spring MVC Step 2

- The DispatcherServlet creates a container using the bean definitions found in the Servlet configuration file.
- The DispatcherServlet locates the configuration file using the naming convention servletname-servlet.xml.

Dispatcher-servlet.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:p="<http://www.springframework.org/schema/p>"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

<bean name="/Link.spring" **class**="LinkController"/>

</beans>

Spring MVC



Marwadi
University
Marwadi Chandarana Group

Spring MVC Step 3

The container finds an appropriate Controller to process the request.

For example, a request for /Link.spring is mapped to the Controller defined by the **mycontroller.LinkController** class

Spring MVC Step 4

The Controller performs some custom logic and returns a ModelAndView to the container. The Controller is a Java class that extends the `org.springframework.web.servlet.mvc.Controller` class.

The request is handled by the `handleRequest` method.

Spring MVC



Marwadi
University
Marwadi Chandarana Group

Spring MVC Step 4

- LinkController.java

```
import javax.servlet.http.HttpServletRequest;
```

```
import javax.servlet.http.HttpServletResponse;
```

```
import org.springframework.web.servlet.ModelAndView;
```

```
import org.springframework.web.servlet.mvc.Controller;
```

```
public class LinkController implements Controller {
```

```
    public ModelAndView handleRequest(HttpServletRequest arg0,  
        HttpServletResponse arg1) throws Exception {
```

```
        return new ModelAndView("Welcome.jsp");
```

```
    }
```

```
}
```

Spring MVC



Marwadi
University
Marwadi Chandarana Group

Spring MVC Step 5

The container renders the ModelAndView using the appropriate View.

Welcome.jsp

```
<h2>
```

```
<font color="Green">
```

```
Welcome to Spring Home Page.
```

```
</font>
```

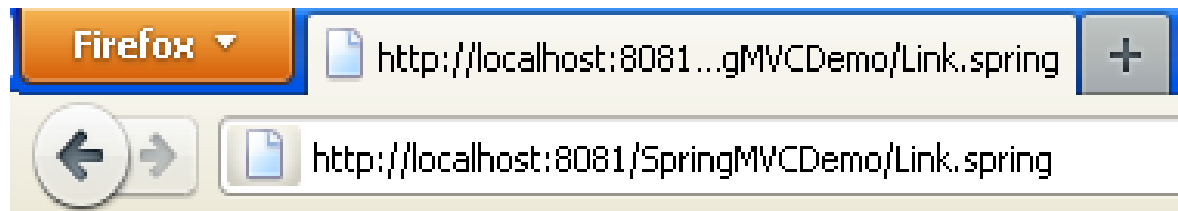
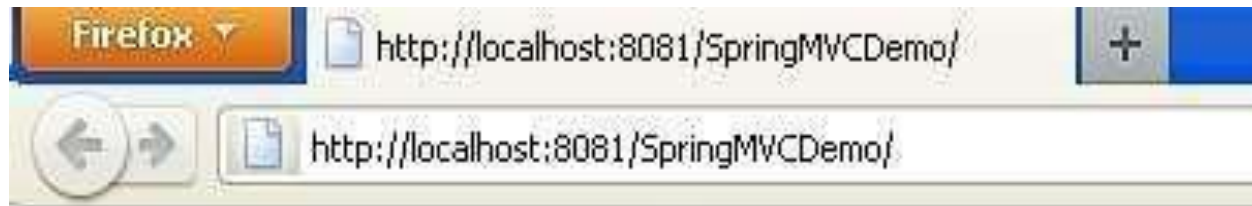
```
</h2>
```

Spring MVC



Marwadi
University
Marwadi Chandarana Group

Output:-



Welcome to Spring Home Page.

Spring – Bean Definition



Marwadi
University
Marwadi Chandarana Group

The objects that form the backbone of your application and that are managed by the Spring IoC container are called **beans**.

A bean is an object that is instantiated, assembled, and otherwise 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.

Bean definition contains the information called **configuration metadata**, which is needed for the container to know the following –

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

Spring – Bean Life Cycle



Marwadi
University
Marwadi Chandarana Group

The life cycle of a Spring bean is easy to understand. When a bean is instantiated, it may be required to perform some initialization to get it into a usable state. Similarly, when the bean is no longer required and is removed from the container, some cleanup may be required.

To define setup and teardown for a bean, we simply declare the `<bean>` with **init method** and/or **destroy-method** parameters.

```
public void init()  
{  
    // do some destruction work  
}
```

```
public void destroy()  
{  
    // do some destruction work  
}
```

Spring Example



Marwadi
University
Marwadi Chandarana Group

- 1) create POJO class
- 2) create application class with main method
- 3) Bean Configuration

1) Create POJO Class HelloWorld.java



Marwadi
University
Marwadi Chandarana Group

```
public class HelloWorld {  
  
    private String message;  
  
    public void setMessage(String message)  
    { this.message = message; }  
  
    public void getMessage()  
    { System.out.println("Your Message : " + message); }  
  
    public void init()  
    { System.out.println("Bean is going through init."); }  
  
    public void destroy()  
    { System.out.println("Bean will destroy now."); }  
  
}
```

2) Create application main method



Marwadi
University
Marwadi Chandarana Group

```
import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp
{
    public static void main(String[] args)
    {
        AbstractApplicationContext context = new
        ClassPathXmlApplicationContext("Beans.xml");

        HelloWorld obj = (HelloWorld) context.getBean("helloWorld");

        obj.getMessage();

        context.registerShutdownHook();
    }
}
```

3) Create configuration file



Marwadi
University
Marwadi Chandarana Group

Beans.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-3.0.xsd">

<bean id = "helloWorld" class = "HelloWorld" init-method = "init" destroy-
method = "destroy">

<property name = "message" value = "Hello World!"/> </bean>

</beans>
```

Output



Marwadi
University
Marwadi Chandarana Group

Bean is going through init.
Your Message : Hello World!
Bean will destroy now.

Inner Bean



Marwadi
University
Marwadi Chandarana Group

Inner beans are the beans that are defined within the scope of another bean. Thus, a `<bean/>` element inside the `<property/>` or `<constructor-arg/>` elements is called inner bean. This page gives an example to inject inner bean in spring.

Example uses constructor based injection, the approach is similar for setter based injection. (Next Slide)

Here is a class called PaymentGateway, which requires Order object to be injected.

Inner Bean Example

1) PaymentGateway.java



Marwadi
University
Marwadi Chandarana Group

```
package com.java2novice.beans;

public class PaymentGateway {
    private Order order;

    public PaymentGateway(Order ord){
        this.order = ord;
    }
    public void setOrder(Order ord){
        this.order = ord;
    }
    public void processOrder(){
        System.out.println("Processing order: "+order.getItem());
    }
}
```


Inner Bean Example

2) Order.java



Marwadi
University
Marwadi Chandarana Group

```
public class Order {  
  
    private String item;  
    private String price;  
    private String address;  
  
    public String getItem() {  
        return item;  
    }  
    public void setItem(String item)  
{  
        this.item = item;  
    }  
}
```

```
    public String getPrice() {  
        return price;  
    }  
    public void setPrice(String price) {  
        this.price = price;  
    }  
    public String getAddress() {  
        return address;  
    }  
    public void setAddress(String  
address) {  
        this.address = address;  
    }  
}
```

Inner Bean Example

3) Bean Configuration applicationContext.xml



Marwadi
University
Marwadi Chandarana Group

Note: When you are using inner bean configuration, bean id is not required. The approach is same for setter based injection

```
<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-3.0.xsd">
```

```
  <bean id="paymentGwBean" class="PaymentGateway">  
    <constructor-arg>  
      <bean class="Order">  
        <property name="item" value="JavaTextBook" />  
        <property name="price" value="RS2 22.50" />  
        <property name="address" value="Bangalore" />  
      </bean>  
    </constructor-arg>  
  </bean>  
</beans>
```

Inner Bean Example

4) Application Class with main() SpringDemo.java



Marwadi
University
Marwadi Chandarana Group

```
package com.java2novice.test;

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

public class SpringDemo {

    public static void main(String a[]){
        String confFile = "applicationContext.xml";
        ApplicationContext context = new ClassPathXmlApplicationContext(confFile);
        PaymentGateway gateway = (PaymentGateway)
context.getBean("paymentGwBean");
        gateway.processOrder();
    }
}
```

Inner Bean Example Output



Marwadi
University
Marwadi Chandarana Group

Processing order: JavaTextBook



Aliases in Beans

In Spring, Bean is accessed using id or name usually. Spring also provides an alias (**second name**) to bean. Hence we can also access a bean by bean alias name. To define a bean alias name, spring provides a <alias> tag as:

Syntax: <alias name = “bean-name” alias = “alias-name-for-defined-bean”/>

In the syntax, there is a <alias> tag that defines a “name” attribute, which represents the bean that is being aliased and attribute “alias” is used to specify the alias name for the bean.

Ex.: <alias name = “triangle” alias = “triangle_alias” />

In the example, we defined an alias for “triangle” bean. The “name” attribute refers to “triangle” bean defined and “alias” attribute defines an alias to “triangle” bean.

Bean Scopes



Marwadi
University
Marwadi Chandarana Group

In the spring bean configurations, bean attribute called 'scope' defines what kind of object has to be created and returned. There are 5 types of bean scopes available, they are:

- 1) **singleton:** Returns a single bean instance per Spring IoC container.
- 2) **prototype:** Returns a new bean instance each time when requested.
- 3) **request:** Returns a single instance for every HTTP request call.
- 4) **session:** Returns a single instance for every HTTP session.
- 5) **global session:** global session scope is equal as session scope on portlet-based web applications.

If no bean scope is specified in bean configuration file, then it will be by default 'singleton'.

Example:

```
<bean id="ticketBean" class="TicketManager" scope="singleton"/>
```

Spring Annotations



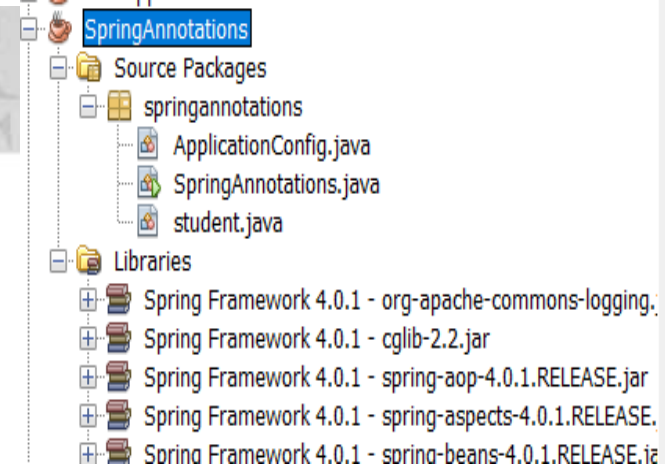
Marwadi
University
Marwadi Chandarana Group

- Spring annotations are a form of metadata that provide additional information about a program.
- Annotations such as:
 - **@Bean** – used in method level with @configuration.
 - **@Configuration** – used in classes for instantiation.
 - **@ComponentScan** – to scan attributes
 - **@Value** – default value/parameter
 - **@Override** etc.

Spring Annotations

Student.java

```
package springannotations;
import org.springframework.beans.factory.annotation.Value;
public class student {
    @Value("101")
    private int RollNo;
    @Value("Abc")
    private String StudentName;
    @Value("91.50")
    private Double Percentage;
    @Override
    public String toString(){
        return "Student [roll="+RollNo+ ", "
            + "StudentName=" +StudentName +","
            + "Percentage=" + Percentage +"]";
    }
}
```



Spring Annotations

ApplicationConfig.java

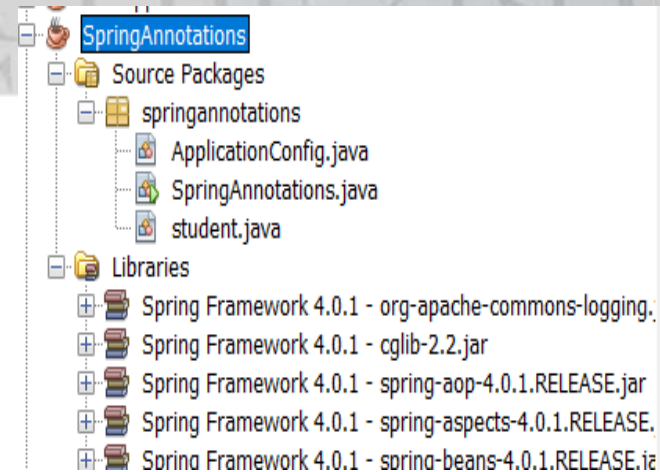
```
package springannotations;
```

```
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.ComponentScan;  
import org.springframework.context.annotation.Configuration;
```

```
@Configuration
```

```
@ComponentScan("springannotations")
```

```
public class ApplicationConfig {  
    @Bean  
    public student studentBean(){  
        return new student();  
    }  
}
```



Spring Annotations

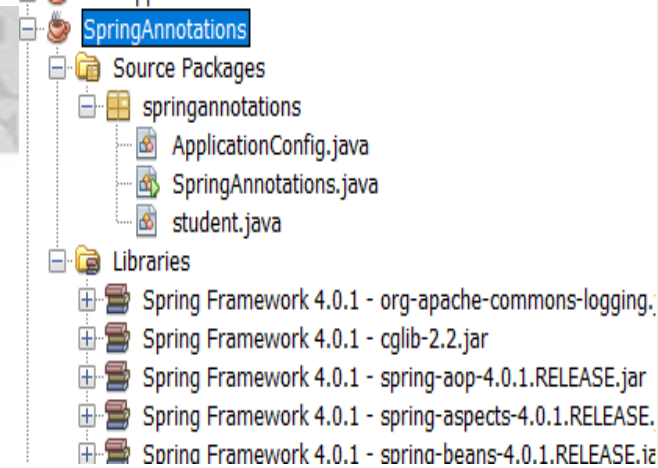
ApplicationConfig.java

package springannotations;

```
import org.springframework.context.annotation.AnnotationConfigApplicationContext;  
import org.springframework.context.support.AbstractApplicationContext;
```

```
public class SpringAnnotations {
```

```
    public static void main(String[] args) {  
        AbstractApplicationContext context = new  
AnnotationConfigApplicationContext(ApplicationConfig.class);  
        student s = (student) context.getBean("studentBean");  
        System.out.println(s);  
        context.close();  
    }  
}
```





Introduction to Spring AOP Module

Aspect-Oriented Programming (AOP) complements Object-Oriented Programming (**OOP**) by providing **another way of thinking** about program structure. The key unit of modularity in **OOP** is the **class**, whereas in **AOP** the unit of modularity is the ***aspect***.

AOP breaks the program logic into distinct parts (called concerns). It is used to increase modularity by cross-cutting concerns.

Aspects enable the **modularization** of concerns such as **transaction management** that **cut across multiple types** and **objects**. (Such concerns are often termed ***crosscutting concerns*** in AOP literature.)

One of the **key components of Spring** is the ***AOP framework***. While the Spring **IoC container does not depend** on AOP, meaning you do not need to use AOP if you don't want to, AOP complements Spring IoC to provide a very capable **middleware** solution.



Introduction to Spring AOP Module

AOP is used in the Spring Framework to...

... provide **declarative enterprise services**, especially as a replacement for EJB declarative services. The most important such service is *declarative transaction management*.

... allow users to implement **custom aspects, complementing** their use of **OOP** with AOP.

AOP concepts



Marwadi
University
Marwadi Chandarana Group

Aspect: a modularization of a concern that cuts across multiple classes. Transaction management is a good example of a crosscutting concern in J2EE applications. In Spring AOP, aspects are implemented **using regular classes** (the schema-based approach) or regular classes annotated with the **@Aspect annotation** (the @AspectJ style).

Join point: a point during the execution of a program, such as the execution of a method or the handling of an exception. **In Spring AOP, a join point always represents a method execution.**

Advice: action taken by an aspect at a particular join point. Different types of advice include "around," "before" and "after" advice. Many AOP frameworks, including Spring, model an advice as an *interceptor*, maintaining a chain of interceptors *around* the join point.

AOP concepts



Marwadi
University
Marwadi Chandarana Group

Pointcut: a predicate that matches **join points**. Advice is associated with a pointcut expression and runs at any join point matched by the pointcut (for example, the execution of a method with a certain name). The concept of join points as matched by pointcut expressions is central to AOP, and Spring uses the **AspectJ pointcut expression language by default**.

Introduction: **declaring additional methods** or **fields** on behalf of a type. Spring AOP allows you to introduce new interfaces (and a corresponding implementation) to any advised object. For example, you could use an introduction to make a bean implement an `IsModified` interface, to simplify caching. (An introduction is known as an inter-type declaration in the AspectJ community.)

Target object: **object being advised** by one or **more aspects**. Also referred to as the *advised* object. Since Spring AOP is implemented using runtime proxies, this object will always be a *proxied* object.

AOP Concepts



Marwadi
University
Marwadi Chandarana Group

AOP proxy: an **object** created by the AOP framework in order to **implement** the **aspect contracts** (advise method executions and so on).

Weaving: **linking aspects** with **other application types** or objects to create an advised object. This can be done at compile time (using the AspectJ compiler, for example), load time, or at runtime. Spring AOP, like other pure Java AOP frameworks, performs weaving at runtime.

Types of Advice



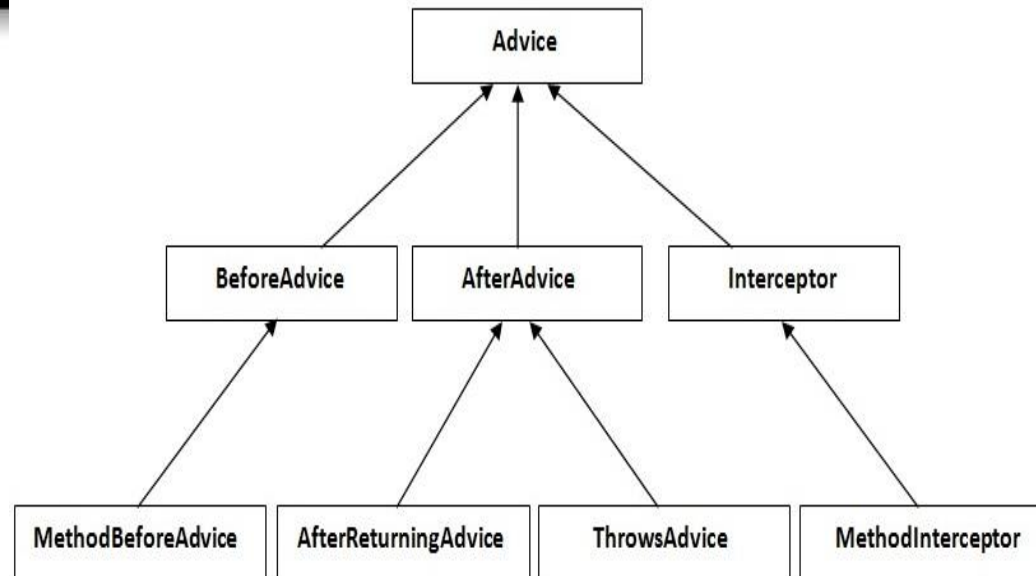
Marwadi
University
Marwadi Chandarana Group

1. **Before advice:** Advice that executes before a **join point**, but which does not have the **ability to prevent execution flow** proceeding to the join point (unless it throws an exception).
2. **After returning advice:** Advice to be executed **after a join point** completes **normally**. For example, if a method returns without throwing an exception.
3. **After throwing advice:** Advice to be executed **if a method exits** by throwing an **exception**.
4. **After (finally) advice:** Advice to be executed regardless of the means by which a **join point exits** (normal or exceptional return).
5. **Around advice:** Advice that **surrounds a join point** such as a method invocation. This is the most **powerful** kind of advice. Around advice can **perform custom** behavior before and after the method invocation.

Advice Interface



Marwadi
University
Marwadi Chandarana Group



All are interfaces in aop.

MethodBeforeAdvice interface extends the BeforeAdvice interface.

AfterReturningAdvice interface extends the AfterAdvice interface.

ThrowsAdvice interface extends the AfterAdvice interface.

MethodInterceptor interface extends the Interceptor interface. It is used in around advice.

Database Transaction Management



Marwadi
University
Marwadi Chandarana Group

Comprehensive transaction support is among the most compelling reasons to use the Spring Framework. The Spring Framework provides a **consistent abstraction for transaction management** that delivers the following benefits:

- Consistent programming model across **different transaction APIs** such as Java Transaction API (JTA), JDBC, Hibernate, Java Persistence API (JPA), and Java Data Objects (JDO).
- Support for **declarative transaction** management.
- Simpler API for **programmatic transaction management** than complex transaction APIs such as JTA.
- Excellent **integration** with Spring's data access abstractions.

Spring DAO



Marwadi
University
Marwadi Chandarana Group

The **Data Access Object (DAO)** support in Spring is aimed at making it easy to work with data access technologies like JDBC, Hibernate, JPA or JDO in a consistent way.

This allows one to switch between the aforementioned persistence technologies fairly easily and it also allows one to code without worrying about catching exceptions that are specific to each technology.

Annotations used for configuring DAO or Repository classes

@Repository

```
public class JdbcMovieFinder implements MovieFinder {  
    private JdbcTemplate jdbcTemplate;
```

@Autowired

```
    public void init(DataSource dataSource) {  
        this.jdbcTemplate = new JdbcTemplate(dataSource);  
    }  
    // ...  
}
```

Spring DAO



Marwadi
University
Marwadi Chandarana Group

Autowiring feature of spring framework enables you to inject the object dependency implicitly. It internally uses setter or constructor injection. Autowiring can't be used to inject primitive and string values. It works with reference only.

Advantage of Autowiring

It requires the less code because we don't need to write the code to inject the dependency explicitly.

Disadvantage of Autowiring

No control of programmer.

CRUD Operation Using DAO and Spring API



Marwadi
University
Marwadi Chandarana Group

<https://www.dineshonjava.com/spring-mvc-with-hibernate-crud-example/>

Learning Resources



Marwadi
University
Marwadi Chandarana Group

<http://www.java2novice.com/spring/bean-scope-annotation/>

<https://www.journaldev.com/2583/spring-aop-example-tutorial-aspect-advice-pointcut-joinpoint-annotations>

Summary



Marwadi
University
Marwadi Chandarana Group

<http://www.java2novice.com/spring/bean-scope-annotation/>

<https://www.journaldev.com/2583/spring-aop-example-tutorial-aspect-advice-pointcut-joinpoint-annotations>

Summary



Marwadi
University
Marwadi Chandarana Group

- Spring: Introduction,
- Architecture,
- Spring MVC Module,
- Life Cycle of Bean Factory,
- Explore: Constructor Injection, Dependency Injection,
- Inner Beans, Aliases in Bean, Bean Scopes,
- Spring Annotations,
- Spring AOP Module, Spring DAO,
- Database Transaction Management,
- CRUD Operation using DAO and Spring API



Marwadi
University
Marwadi Chandarana Group

END OF UNIT - 6