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

There are many advantages of Spring Framework. They are as follows:

#### 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.

#### 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.

#### 5) Fast Development

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

#### 6) Powerful abstraction

It provides powerful abstraction to JavaEE specifications such as [JMS](https://www.javatpoint.com/jms-tutorial), [JDBC](https://www.javatpoint.com/java-jdbc), JPA and JTA.

#### 7) Declarative support

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

# IoC Container

The IoC container is responsible to instantiate, configure and assemble the objects. The IoC container gets informations from the XML file and works accordingly. The main tasks performed by IoC container are:

* to instantiate the application class
* to configure the object
* to assemble the dependencies between the objects

There are two types of IoC containers. They are:

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

### Difference between BeanFactory and the ApplicationContext

The org.springframework.beans.factory.**BeanFactory** and the org.springframework.context.**ApplicationContext** interfaces acts as the IoC container. The ApplicationContext interface is built on top of the BeanFactory interface. It adds some extra functionality than BeanFactory such as simple integration with Spring's AOP, message resource handling (for I18N), event propagation, application layer specific context (e.g. WebApplicationContext) for web application. So it is better to use ApplicationContext than BeanFactory.

1. Resource resource=**new** ClassPathResource("applicationContext.xml");

BeanFactory factory=**new** XmlBeanFactory(resource);

1. ApplicationContext context =

**new** ClassPathXmlApplicationContext("applicationContext.xml");

# Dependency Injection in Spring

Dependency Injection (DI) is a design pattern that removes the dependency from the programming code so that it can be easy to manage and test the application. Dependency Injection makes our programming code loosely coupled.

The Dependency Injection is a design pattern that removes the dependency of the programs. In such case we provide the information from the external source such as XML file. It makes our code loosely coupled and easier for testing.

### Two ways to perform Dependency Injection in Spring framework

Spring framework provides two ways to inject dependency

* By Constructor
* By Setter method

The **ref** element is used to define the reference of another bean. Here, we are using **bean** attribute of **ref** element to specify the reference of another bean.

The **key-ref** and **value-ref** attributes of entry **element** is used to define the reference of bean in the map.

# Dependency Injection by setter method

We can inject the dependency by setter method also. The **<property>** subelement of **<bean>** is used for setter injection.

# Difference between constructor and setter injection

There are many key differences between constructor injection and setter injection.

1. **Partial dependency**: can be injected using setter injection but it is not possible by constructor. Suppose there are 3 properties in a class, having 3 arg constructor and setters methods. In such case, if you want to pass information for only one property, it is possible by setter method only.
2. **Overriding**: Setter injection overrides the constructor injection. If we use both constructor and setter injection, IOC container will use the setter injection.
3. **Changes**: We can easily change the value by setter injection. It doesn't create a new bean instance always like constructor. So setter injection is flexible than constructor injection.

# Autowiring in Spring

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.

It can't be used for primitive and string values.

## Autowiring Modes

There are many autowiring modes:

| **sn** | **Mode** | **Description** |
| --- | --- | --- |
| 1) | no | It is the default autowiring mode. It means no autowiring bydefault. |
| 2) | byName | The byName mode injects the object dependency according to name of the bean. In such case, property name and bean name must be same. It internally calls setter method. |
| 3) | byType | The byType mode injects the object dependency according to type. So property name and bean name can be different. It internally calls setter method. |
| 4) | constructor | The constructor mode injects the dependency by calling the constructor of the class. It calls the constructor having large number of parameters. |
| 5) | autodetect | It is deprecated since Spring 3. |

# 

# 

# Spring AOP Tutorial

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

A **cross-cutting concern** is a concern that can affect the whole application and should be centralized in one location in code as possible, such as transaction management, authentication, logging, security etc.

#### Where use AOP?

AOP is mostly used in following cases:

* to provide declarative enterprise services such as declarative transaction management.
* It allows users to implement custom aspects.

## AOP Concepts and Terminology

AOP concepts and terminologies are as follows:

* Join point
* Advice
* Pointcut
* Introduction
* Target Object
* Aspect
* Interceptor
* AOP Proxy
* Weaving

#### Join point

Join point is any point in your program such as method execution, exception handling, field access etc. Spring supports only method execution join point.

#### Advice

Advice represents an action taken by an aspect at a particular join point. There are different types of advices:

* **Before Advice**: it executes before a join point.
* **After Returning Advice**: it executes after a joint point completes normally.
* **After Throwing Advice**: it executes if method exits by throwing an exception.
* **After (finally) Advice**: it executes after a join point regardless of join point exit whether normally or exceptional return.
* **Around Advice**: It executes before and after a join point.

#### Pointcut

It is an expression language of AOP that matches join points.

#### Introduction

It means introduction of additional method and fields for a type. It allows you to introduce new interface to any advised object.

#### Target Object

It is the object i.e. being advised by one or more aspects. It is also known as proxied object in spring because Spring AOP is implemented using runtime proxies.

#### Aspect

It is a class that contains advices, joinpoints etc.

#### Interceptor

It is an aspect that contains only one advice.

#### AOP Proxy

It is used to implement aspect contracts, created by AOP framework. It will be a JDK dynamic proxy or CGLIB proxy in spring framework.

#### Weaving

It is the process of linking aspect with other application types or objects to create an advised object. Weaving can be done at compile time, load time or runtime. Spring AOP performs weaving at runtime.

There are 4 types of advices supported in spring1.2 old style aop implementation.

1. **Before Advice** it is executed before the actual method call.
2. **After Advice** it is executed after the actual method call. If method returns a value, it is executed after returning value.
3. **Around Advice** it is executed before and after the actual method call.
4. **Throws Advice** it is executed if actual method throws exception.

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.

# Spring AOP AspectJ Annotation Example

The **Spring Framework** recommends you to use **Spring AspectJ AOP implementation** over the Spring 1.2 old style dtd based AOP implementation because it provides you more control and it is easy to use.

There are two ways to use Spring AOP AspectJ implementation:

1. By annotation: We are going to learn it here.
2. By xml configuration (schema based): We will learn it in next page.

Spring AspectJ AOP implementation provides many annotations:

1. **@Aspect** declares the class as aspect.
2. **@Pointcut** declares the pointcut expression.

The annotations used to create advices are given below:

1. **@Before** declares the before advice. It is applied before calling the actual method.
2. **@After** declares the after advice. It is applied after calling the actual method and before returning result.
3. **@AfterReturning** declares the after returning advice. It is applied after calling the actual method and before returning result. But you can get the result value in the advice.
4. **@Around** declares the around advice. It is applied before and after calling the actual method.
5. **@AfterThrowing** declares the throws advice. It is applied if actual method throws exception.
6. @Aspect
7. **public** **class** TrackOperation{
8. @Pointcut("execution(\* Operation.\*(..))")
9. **public** **void** k(){}//pointcut name
11. @Before("k()")//applying pointcut on before advice
12. **public** **void** myadvice(JoinPoint jp)//it is advice (before advice)
13. {
14. System.out.println("additional concern");
15. //System.out.println("Method Signature: " + jp.getSignature());
16. }
17. }

# Spring JdbcTemplate Tutorial

Spring **JdbcTemplate** is a powerful mechanism to connect to the database and execute SQL queries. It internally uses JDBC api, but eliminates a lot of problems of JDBC API.

## Problems of JDBC API

The problems of JDBC API are as follows:

* We need to write a lot of code before and after executing the query, such as creating connection, statement, closing resultset, connection etc.
* We need to perform exception handling code on the database logic.
* We need to handle transaction.
* Repetition of all these codes from one to another database logic is a time consuming task.

## Advantage of Spring JdbcTemplate

Spring JdbcTemplate eliminates all the above mentioned problems of JDBC API. It provides you methods to write the queries directly, so it saves a lot of work and time.

## Spring Jdbc Approaches

Spring framework provides following approaches for JDBC database access:

* JdbcTemplate
* NamedParameterJdbcTemplate
* SimpleJdbcTemplate
* SimpleJdbcInsert and SimpleJdbcCall

## JdbcTemplate class

It is the central class in the Spring JDBC support classes. It takes care of creation and release of resources such as creating and closing of connection object etc. So it will not lead to any problem if you forget to close the connection.

### ResultSetExtractor Interface

**ResultSetExtractor** interface can be used to fetch records from the database. It accepts a ResultSet and returns the list.

#### Method of ResultSetExtractor interface

It defines only one method extractData that accepts ResultSet instance as a parameter. Syntax of the method is given below:

1. **public** T extractData(ResultSet rs)**throws** SQLException,DataAccessException

# RowMapper Example | Fetching records by Spring JdbcTemplate

Like ResultSetExtractor, we can use RowMapper interface to fetch the records from the database using **query()** method of **JdbcTemplate** class. In the execute of we need to pass the instance of RowMapper now.

### RowMapper Interface

**RowMapper** interface allows to map a row of the relations with the instance of user-defined class. It iterates the ResultSet internally and adds it into the collection. So we don't need to write a lot of code to fetch the records as ResultSetExtractor.

#### Advantage of RowMapper over ResultSetExtractor

RowMapper saves a lot of code becuase it internally adds the data of ResultSet into the collection.

### Advantage of ORM Frameworks with Spring

There are a lot of advantage of Spring framework in respect to ORM frameworks. There are as follows:

* **Less coding is required**: By the help of Spring framework, you don't need to write extra codes before and after the actual database logic such as getting the connection, starting transaction, commiting transaction, closing connection etc.
* **Easy to test**: Spring's IoC approach makes it easy to test the application.
* **Better exception handling**: Spring framework provides its own API for exception handling with ORM framework.
* **Integrated transaction management**: By the help of Spring framework, we can wrap our mapping code with an explicit template wrapper class or AOP style method interceptor.

# Spring MVC Tutorial

A Spring MVC is a Java framework which is used to build web applications. It follows the Model-View-Controller design pattern. It implements all the basic features of a core spring framework like Inversion of Control, Dependency Injection.

A Spring MVC provides an elegant solution to use MVC in spring framework by the help of **DispatcherServlet**. Here, **DispatcherServlet** is a class that receives the incoming request and maps it to the right resource such as controllers, models, and views.

## Spring Web Model-View-Controller



* **Model** - A model contains the data of the application. A data can be a single object or a collection of objects.
* **Controller** - A controller contains the business logic of an application. Here, the @Controller annotation is used to mark the class as the controller.
* **View** - A view represents the provided information in a particular format. Generally, JSP+JSTL is used to create a view page. Although spring also supports other view technologies such as Apache Velocity, Thymeleaf and FreeMarker.
* **Front Controller** - In Spring Web MVC, the DispatcherServlet class works as the front controller. It is responsible to manage the flow of the Spring MVC application.

## Advantages of Spring MVC Framework

Let's see some of the advantages of Spring MVC Framework:-

* **Separate roles** - The Spring MVC separates each role, where the model object, controller, command object, view resolver, DispatcherServlet, validator, etc. can be fulfilled by a specialized object.
* **Light-weight** - It uses light-weight servlet container to develop and deploy your application.
* **Powerful Configuration** - It provides a robust configuration for both framework and application classes that includes easy referencing across contexts, such as from web controllers to business objects and validators.
* **Rapid development** - The Spring MVC facilitates fast and parallel development.
* **Reusable business code** - Instead of creating new objects, it allows us to use the existing business objects.
* **Easy to test** - In Spring, generally we create JavaBeans classes that enable you to inject test data using the setter methods.
* **Flexible Mapping** - It provides the specific annotations that easily redirect the page.

### Create the controller class

To create the controller class, we are using two annotations @Controller and @RequestMapping.

The @Controller annotation marks this class as Controller.

The @Requestmapping annotation is used to map the class with the specified URL name.

# Spring MVC RequestParam Annotation

In Spring MVC, the **@RequestParam** annotation is used to read the form data and bind it automatically to the parameter present in the provided method. So, it ignores the requirement of **HttpServletRequest** object to read the provided data.

# Spring MVC Form Tag Library

The Spring MVC form tags are the configurable and reusable building blocks for a web page. These tags provide JSP, an easy way to develop, read and maintain.

The Spring MVC form tags can be seen as data binding-aware tags that can automatically set data to Java object/bean and also retrieve from it. Here, each tag provides support for the set of attributes of its corresponding HTML tag counterpart, making the tags familiar and easy to use.

## Configuration of Spring MVC Form Tag

The form tag library comes under the spring-webmvc.jar. To enable the support for form tag library, it is required to reference some configuration. So, add the following directive at the beginning of the JSP page:

1. **<**%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %**>**

## List of Spring MVC Form Tags

| **Form Tag** | **Description** |
| --- | --- |
| form:form | It is a container tag that contains all other form tags. |
| form:input | This tag is used to generate the text field. |
| form:radiobutton | This tag is used to generate the radio buttons. |
| form:checkbox | This tag is used to generate the checkboxes. |
| form:password | This tag is used to generate the password input field. |
| form:select | This tag is used to generate the drop-down list. |
| form:textarea | This tag is used to generate the multi-line text field. |
| form:hidden | This tag is used to generate the hidden input field. |

In Controller class, the **@PathVariable** annotation bounds the method parameter with a temporary URL. For example:

1. @RequestMapping(value="/viewemp/{pageid}")

# Spring MVC Tiles Example

Spring provides integration support with apache tiles framework. So we can simply manage the layout of the Spring MVC application with the help of spring tiles support.

## Advantage of Tiles support in Spring MVC

**Reusability:** We can reuse a single component in multiple pages like header and footer components.

**Centralized control:** We can control the layout of the page by a single template page only.

**Easy to change the layout:** By the help of single template page, we can change the layout of the page anytime. So your website can easily adopt new technologies such as bootstrap, jQuery, etc.

### 1) What is Spring?

It is a lightweight, loosely coupled and integrated framework for developing enterprise applications in java.

### 2) What are the advantages of spring framework?

1. Predefined Templates
2. Loose Coupling
3. Easy to test
4. Lightweight
5. Fast Development
6. Powerful Abstraction
7. Declarative support

### 3) What are the modules of spring framework?

1. Test
2. Spring Core Container
3. AOP, Aspects and Instrumentation
4. Data Access/Integration
5. Web

### 4) What is IOC and DI?

IOC (Inversion of Control) and DI (Dependency Injection) is a design pattern to provide loose coupling. It removes the dependency from the program.

### 5) What is the role of IOC container in spring?

IOC container is responsible to:

* create the instance
* configure the instance, and
* assemble the dependencies

### 6) What are the types of IOC container in spring?

There are two types of IOC containers in spring framework.

1. BeanFactory
2. ApplicationContext

### 7) What is the difference between BeanFactory and ApplicationContext?

BeanFactory is the **basic container** whereas ApplicationContext is the **advanced container**. ApplicationContext extends the BeanFactory interface. ApplicationContext provides more facilities than BeanFactory such as integration with spring AOP, message resource handling for i18n etc.

### 8) What is the difference between constructor injection and setter injection?

| **No.** | **Constructor Injection** | **Setter Injection** |
| --- | --- | --- |
| ) | No Partial Injection | Partial Injection |
| 2) | Desn't override the setter property | Overrides the constructor property if both are defined. |
| 3) | Creates new instance if any modification occurs | Doesn't create new instance if you change the property value |
| 4) | Better for too many properties | Better for few properties. |

### 9) What is autowiring in spring? What are the autowiring modes?

Autowiring enables the programmer to inject the bean automatically. We don't need to write explicit injection logic.

The autowiring modes are given below:

| **No** | **Mode** | **Description** |
| --- | --- | --- |
| 1) | no | this is the default mode, it means autowiring is not enabled. |
| 2) | byName | injects the bean based on the property name. It uses setter method. |
| 3) | byType | injects the bean based on the property type. It uses setter method. |
| 4) | constructor | It injects the bean using constructor |

### 10) What are the different bean scopes in spring?

There are 5 bean scopes in spring framework.

| **N** | **Scope** | **Description** |
| --- | --- | --- |
| 1 | singleton | The bean instance will be only once and same instance will be returned by the IOC container. It is the default scope. |
| 2 | prototype | The bean instance will be created each time when requested. |
| 3 | request | The bean instance will be created per HTTP request. |
| 4 | session | The bean instance will be created per HTTP session. |
| 5 | globalsession | The bean instance will be created per HTTP global session. It can be used in portlet context only. |

### 11) In which scenario, you will use singleton and prototype scope?

Singleton scope should be used with EJB **stateless session bean** and prototype scope with EJB **stateful session bean**.

### 12) What are the transaction management supports provided by spring?

Spring framework provides two type of transaction management supports:

1. **Programmatic Transaction Management**: should be used for few transaction operations.
2. **Declarative Transaction Management**: should be used for many transaction operations.

### 13) What are the advantages of JdbcTemplate in spring?

**Less code**: By using the JdbcTemplate class, you don't need to create connection,statement,start transaction,commit transaction and close connection to execute different queries. You can execute the query directly.

14) What are classes for spring JDBC API?

1. JdbcTemplate
2. SimpleJdbcTemplate
3. NamedParameterJdbcTemplate
4. SimpleJdbcInsert
5. SimpleJdbcCall

### 15) How can you fetch records by spring JdbcTemplate?

You can fetch records from the database by the **query method of JdbcTemplate**. There are two interfaces to do this:

1. [ResultSetExtractor](https://www.javatpoint.com/ResultSetExtractor-example)
2. [RowMapper](https://www.javatpoint.com/RowMapper-example)

### 16) What is the advantage of NamedParameterJdbcTemplate?

NamedParameterJdbcTemplate class is used to pass value to the named parameter. A named parameter is better than ? (question mark of PreparedStatement).

It is **better to remember**.

### 17) What is the advantage of SimpleJdbcTemplate?

The **SimpleJdbcTemplate** supports the feature of var-args and autoboxing.

### 18) What is AOP?

AOP is an acronym for Aspect Oriented Programming. It is a methodology that divides the program logic into pieces or parts or concerns.

It increases the modularity and the key unit is Aspect.

### 19) What are the advantages of spring AOP?

AOP enables you to dynamically add or remove concern before or after the business logic. It is **pluggable** and **easy to maintain**.

### 20) What are the AOP terminology?

AOP terminologies or concepts are as follows:

* JoinPoint
* Advice
* Pointcut
* Aspect
* Introduction
* Target Object
* Interceptor
* AOP Proxy
* Weaving

### 21) What is JoinPoint?

JoinPoint is any point in your program such as field access, method execution, exception handling etc.

### 22) Does spring framework support all JoinPoints?

No, spring framework supports method execution joinpoint only.

### 23) What is Advice?

Advice represents action taken by aspect.

### 24) What are the types of advice in AOP?

There are 5 types of advices in spring AOP.

1. Before Advice
2. After Advice
3. After Returning Advice
4. Throws Advice
5. Around Advice

### 25) What is Pointcut?

Pointcut is expression language of Spring AOP.

### 26) What is Aspect?

Aspect is a class in spring AOP that contains advices and joinpoints.

### 27) What is Introduction?

Introduction represents introduction of new fields and methods for a type.

### 28) What is target object?

Target Object is a proxy object that is advised by one or more aspects.

### 29) What is interceptor?

Interceptor is a class like aspect that contains one advice only.

### 30) What is weaving?

Weaving is a process of linking aspect with other application.

### 31) Does spring perform weaving at compile time?

No, spring framework performs weaving at runtime.

### 32) What are the AOP implementation?

There are 3 AOP implementation.

1. Spring AOP
2. Apache AspectJ
3. JBoss AOP

### 33) What is the front controller class of Spring MVC?

The **DispatcherServlet** class works as the front controller in Spring MVC.

34) What does @Controller annotation?

The **@Controller** annotation marks the class as controller class. It is applied on the class.

### 35) What does @RequestMapping annotation?

The **@RequestMapping** annotation maps the request with the method. It is applied on the method.

### 36) What does the ViewResolver class?

The **View Resolver** class resolves the view component to be invoked for the request. It defines prefix and suffix properties to resolve the view component.

### 37) Which ViewResolver class is widely used?

The **org.springframework.web.servlet.view.InternalResourceViewResolver** class is widely used.