Why Spring?

* Spring is the most popular application development framework for enterprise Java.
* Spring is a *lightweight* framework with POJO(Plain old Java objects)

Instead of heavy weight EJB’s.

* 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.
* Dependency injection to provide loose coupling
* Declarative programming support through AOP(
* It provides lot of helper classes .makes thing easy
* Earlier EJB part of J2EE was in use(Enterprise Java Bean).

Disadvantage of EJB:1)extremely complex

2)Multiple deployment descriptor

3)Multiple interfaces

4)Poor performance of entity beans

J2EE technologies:

* Java Servlets.
* JavaServer Pages (JSP)
* Enterprise JavaBeans (EJB)
* Java Message Service (JMS)
* Java Naming and Directory Interface (JNDI)
* Java Database Connectivity (JDBC)
* JavaMail.
* Java Transaction Service (JTS)

Spring 5 update:

1)minimum requirement is java 8 or higher

2)Updated spring MVC to use updated servlet API 4.0

Spring Webflux

Features:

1)LightWeight:

Lightweight=which requires less resources. Spring requires less jars.

The basic version of Spring framework is around 2MB.

Version 4.3.9: 65MB

2)IOC(Inversion of Control)

Approach of outsourcing creation and management of objects.

* We don’t create objects.
* Objects are created by Spring container.
* We only tell the container how to create the objects(configuration XML file(SRC folder)).
* So developer can focus on business logic.
* Loose coupling is achieved by IOC .

How?

By using interface reference

3)AOP(Aspect Oriented programming

Everything other than business is known as cross cutting concern or aspect.

E.g of aspects

* Exceptions
* Security
* Database connection
* Frontend
* Testing

Different aspects have different locations

Advantages:

* parallel development
* loose coupling
* Reuse

4)DI(Dependency Injection)

What ?DI is design pattern which removes dependency from programming code .

Why?

So we can easily manage and test the application

To understand the DI better, Let's understand the Dependency Lookup (DL) first:

### Dependency Lookup

The Dependency Lookup is an approach where we get the resource after demand.

There are different ways:

1)calling static factory method

2)JNDI

Problems:

Tight coupling’

DI is implementation of IOC

Objects are provided by framework

Out,err,in provided fy java

E.g of DI: In Servlet to handle request and response we don’t need to create objects of response and request . These objects are provided by servlet container.

5)Container

It create and manage life cycle and configure objects

6)MVC (Model view cpntroller Design Pattern)

2 Types of MVC:

1)Type 1 MVC

M=Pojo

V,C=JSP

2)MVC type 2

M=POJO

V=JSP

C=Servlet

Spring follows front controller design pattern

Model

Controller

Client

DB

C2

C1

From one controller request is send to other controllers.

Advantage: If you want to generate log or implement security.You need to write code for only one controller.

7)POJO based application

POJO: The class which does not depend on any library ,does not extend any other class or implements interface.

8)Declarative support

You only provide metadata i.e annotation

Spring Architecture



Core Container

The Core Container consists of the Core, Beans, Context, and Expression Language modules the details of which are as follows −

* The **Core** module provides the fundamental parts of the framework, including the IoC and Dependency Injection features.manage dependency
* The **Bean** module provides BeanFactory, which is a sophisticated implementation of the factory pattern.=factory for creating beans
* The **Context:**

Context meaning: info about whole application.Beans.xml contains that info

  it is a medium to access any objects defined and configured. The ApplicationContext interface is the focal point of the Context module.

Spring contexts are also called Spring IoC containers, which are responsible for instantiating, configuring, and assembling beans by reading configuration metadata from XML, Java annotations, and/or Java code in the configuration files.

we will create the object of a Spring context. We are using AnnotationConfigApplicationContext as a Spring container. Also, there are other Spring containers like ClassPathXmlApplicationContext, GenericGroovyApplicationContext.

* Spring 3 onwards: AnnotationBased Config
* If XML file is inside: ClassPathXmlApplicationContext
* If XML file outside the appln:FileSystemApplicationContext

ConfigurableApplicationContext context = new AnnotationConfigApplicationContext(EmployyeCongig.class);

* The **SpEL** module provides a powerful expression language for querying and manipulating an object graph at runtime.

## 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 for 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.

### 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.

[**next →**](https://www.javatpoint.com/dependency-injection-in-spring)[**← prev**](https://www.javatpoint.com/example-of-spring-application-in-eclipse)

# IoC Container

1. [IoC Container](https://www.javatpoint.com/ioc-container)
2. [Using BeanFactory](https://www.javatpoint.com/ioc-container)
3. [Using ApplicationContext](https://www.javatpoint.com/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.

### Using BeanFactory

The XmlBeanFactory is the implementation class for the BeanFactory interface. To use the BeanFactory, we need to create the instance of XmlBeanFactory class as given below:

1. Resource resource=**new** ClassPathResource("applicationContext.xml");
2. BeanFactory factory=**new** XmlBeanFactory(resource);

#### Using ApplicationContext

The ClassPathXmlApplicationContext class is the implementation class of ApplicationContext interface. We need to instantiate the ClassPathXmlApplicationContext class to use the ApplicationContext as given below:

1. ApplicationContext context =
2. **new** ClassPathXmlApplicationContext("applicationContext.xml");
3. -----------------------------------------------------------------------------------------

Spring Development Process

1)Configure spring beans

Write configuration in xml file/annotation/java sourcecode

2)create a spring container

ClassPathXmlApplicationContext

3)Retrieve beans from spring container.

getBean method

What is spring bean ?

* Spring bean is simply java object
* When java objects are created by spring container then Spring refers to them as spring bean
* Spring Beans are created from normal Java classes .... just like Java objects.

Scope of bean

It means the how long it will live?

How many instances we can create?

Sharing of bean

Bydefault the scope is singleton.What is Singleton?

1)Spring container creates only one instance of bean by default

2)It is cashed in memory.

3)All request for the bean will return shared reference to the bean.

2)Prototype= Always new instance is created,

Destroy method will not be called. In prototype scope Spring container instantiate objects, configure them and hand it over to client .So client is responsible for clean up code.

3)Request scope: Http request

4)Sessions Scope: Http session

5)Global session scope

Setup Resources like db,files

Bean life cycle

Custom init methods

Internal spring processing

Dependency injected

Bean instanciated

Container started

Bean is ready to use

Container is shutdown

Custom destroy methods

Init destroy methods signature:

* Can have any access specifier
* Can have any method name
* Can have any return type. But note that you can not capture the return value
* Can not have any argument.it should be no-arg

IOC

Need?

CricketCoach.class

getWorkOutDetails()

KhokhoCoach.class

BasketBallCoach.class

MyApp

getWorkOutDetails()

getWorkOutDetails()

If multiple classes have same method. With the same object reference we should be able to access the method getWorkoutDetails from different classes.

How to achieve?

* Create interface .
* Configure objects
* Create Container .
* Retrieve object.
* Call the method.

--------------------------------------------------------------------------------------

IOC using Annotations:

Annotation: metadata

* XML congiguration is verbose
* Annotation reduces xml configuration work

Steps:

1)add component scan in xml file

2)Add @Component(“id”) annotation in class. It registers spring bean automatically

3) Retrieve beans from spring container.

Dependency Injection using annotation

@Autowire: By type(class or interface)

* Constructor injection
* Setter injection
* Field injection: It is done by java technology : Reflection

NouniquebeanDefinationexception: when there are many implementation of same service.

Spring Configuration:

1)Create a class & annotate it with @Configure annotation

Add @Bean annotation to it.

@Bean

**public** Coach getCoach(){

System.***out***.println("inside coach bean ");

**return** **new** TennisCoach();

}

Since we didn't specify a scope, the bean scope is singleton. As a result, it will give the same instance of the bean for any requests.

It is important to note that this method has the @Bean annotation. The annotation will intercept ALL calls to the method " getCoach ()". Since no scope is specified the @Bean annotation uses singleton scope. Behind the scenes, during the @Bean interception, it will check in memory of the Spring container (applicationContext) and see if this given bean has already been created.

If this is the first time the bean has been created then it will execute the method as normal. It will also register the bean in the application context. So that is knows that the bean has already been created before. Effectively setting a flag.

The next time this method is called, the @Bean annotation will check in memory of the Spring container (applicationContext) and see if this given bean has already been created. Since the bean has already been created (previous paragraph) then it will immediately return the instance from memory. It will not execute the code inside of the method. Hence this is a singleton bean.

2)Add @ComponentScanAnnotation to it

3)Read Spring Configuration class from Test Class using

AnnotationConfigApplicationContext context = **new** AnnotationConfigApplicationContext(SportConfig.**class**);

4)Retrieve bean from spring container

Using getBean() method

Spring MVC

The Spring Web MVC framework provides Model-View-Controller (MVC) architecture and ready components that can be used to develop flexible and loosely coupled web applications. The MVC pattern results in separating the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between these elements.

* The **Model** encapsulates the application data and in general they will consist of POJO.
* The **View** is responsible for rendering the model data and in general it generates HTML output that the client's browser can interpret.
* The **Controller** is responsible for processing user requests and building an appropriate model and passes it to the view for rendering.

The DispatcherServlet

The Spring Web model-view-controller (MVC) framework is designed around a *DispatcherServlet* that handles all the HTTP requests and responses. The request processing workflow of the Spring Web MVC *DispatcherServlet* is illustrated in the following diagram −



Following is the sequence of events corresponding to an incoming HTTP request to *DispatcherServlet* −

* After receiving an HTTP request, *DispatcherServlet* consults the *HandlerMapping* to call the appropriate *Controller*.
* The *Controller* takes the request and calls the appropriate service methods based on used GET or POST method. The service method will set model data based on defined business logic and returns view name to the *DispatcherServlet*.
* The *DispatcherServlet* will take help from *ViewResolver* to pickup the defined view for the request.
* Once view is finalized, The *DispatcherServlet* passes the model data to the view which is finally rendered on the browser.

All the above-mentioned components, i.e. HandlerMapping, Controller, and ViewResolver are parts of *WebApplicationContext* w which is an extension of the plain*ApplicationContext* with some extra features necessary for web applications.

Required Configuration

@Configuration

@EnableWebMvc

@ComponentScan(basePackages="com.app")

**public** **class** HelloWorldConfig {

@Bean

**public** ViewResolver viewResolver(){

InternalResourceViewResolver viewResolver= **new** InternalResourceViewResolver();

viewResolver.setViewClass(JstlView.**class**);

viewResolver.setPrefix("WEB-INF/views/");

viewResolver.setSuffix(".jsp");

**return** viewResolver;

}

}

**public** **class** HelloWorldInitializer **extends** AbstractAnnotationConfigDispatcherServletInitializer{

@Override

**protected** Class<?>[] getRootConfigClasses() {

// **TODO** Auto-generated method stub

**return** **new** Class[]{HelloWorldConfig.**class**};

}

@Override

**protected** Class<?>[] getServletConfigClasses() {

// **TODO** Auto-generated method stub

**return** **null**;

}

@Override

**protected** String[] getServletMappings() {

**return** **new** String[]{"/"};

}

}

Steps

1)Add spring jars in WEB-INF-> lib folder. It will get add in buid path so no need to add separately.Add Jsp and JSTL jars as well.

2)Configuration File

* SpringConfiguration.xml
* Web.xml

Add both the file in Web.xml