**Spring Short Note:-**

* Using web framework we can just develop the web app’s.
* Using ORM Framework we can develop the Persistence Logic.
* Using Spring we can develop the all kind of logics business logic, presentation logic, Integration logic and persistence logic etc..
* Using spring we can develop all kind of app’s standalone app’s, two-tier app’s, web based app’s, distributed app’s and enterprise app’s etc.
* Spring 1.x contains 7 module, spring 2.x contains 6 modules
* Spring 3.x contains 20 modules but there are group into 6 major categories etc…
* Spring Core is the base module for all others module is
* Spring DAO/Spring JDBC module provide the abstraction layer plain JDBC to data persistence logic.

Programmer testing on his own peace of code is called as **unit testing…**

|  |
| --- |
| What Is Spring Framework, Spring Introduction Let us see what Spring Framework is, and why we need to work with this Spring…  <http://www.java4s.com/wp-content/uploads/2011/08/rod-johnson-picture.jpg>   * Spring is a Java based open source, light weight, loosely coupled, aspect oriented and dependency based app’s/J2EE Based app’s and to develop the all kind of java app’s. * Opensouce S/W means it is not only free s/w .it source code will exposed/visible to the programmer along with installations. * Spring is a java based light weight and open source framework created by Rod Johnson in 2003.( Most of the Spring app’s can executed without heavyweight webserver/app’s server… ) * Spring is a complete and a modular framework, i mean spring framework can be used for all layer implementations for a real time application or spring can be used for the development of particular layer of a real time application unlike struts [ only for front end related ] and hibernate [ only for database related ], but with spring we can develop all layers. * Spring framework is said to be a non-invasive means it doesn’t force a programmer to extend or implement their class from any predefined class or interface given by Spring API, in struts we used to extend Action Class right that’s why struts is said to be invasive * In case of struts framework, it will forces the programmer that, the programmer class must extend from the base class provided by struts API * Spring is light weight framework because of its POJO model * Spring Framework made J2EE application development little easier, by introducing POJO model * While executing the Spring App’s much memory and CPU resources are not required. * Spring technology is easy to learn and use * If the Degree of Dependency is less b/w two components then we can say the component are loosely coupled component * If the Degree of Dependency is more b/w two components then we can say the component are tightly coupled component   Spring having this much of demand because of the following 3 reasons….   * Simplicity * Testability * Loose Coupling    Simplicity Spring framework is simple because as it is non-invasive, POJO and POJI model   Testability Actually for writing the spring application, server [**Container**] is not mandatory, but for struts applications we need a server, and for EJB too.  If we want to test the application it may need lot of changes in the source and each time we must restart the server to view the changes, which is little tedious and time taking but we can overcome this in Spring, for testing spring application server is not mandatory spring has it own container to run the applications.  Spring can be used to develop any kind of java application, means we can develop starting from console application to enterprise level application   Loose Coupling In spring objects are loosely coupled,  this is the core concept of spring framework we will see in depth about this loose coupling and how its differ from tight coupling |

Spring is loosely coupled s/w reason

* While working with spring we can use either individual module or multiple modules to gather.
* we can integrate spring with Other Java Technologies In app’s deployment like Servlet, JSP, Struts, and Hibernate etc…
* Every spring module provides abstraction layer on providing multiple core technologies and makes programmer free from core technologies based programing while developing the apps.
* It is always recommended to develop large scale apps by enabling the middle ware services like security, transaction management etc…

[**Data Transfer (value) Object (DTO)**](http://javaonlinetutorial.blogspot.in/2011/10/data-transfer-value-object-dto.html)

* While transferring huge amount of data or values between two layers of application instead of sending between two layer multiple number of time its is recommended to combine all these multiple values into single object and sends that object to destination layer
* due to this round trips between layers will be received and sending and receiving values will become easy in struts application, form page view layer use multiple values as form data for action class
* all these values will combined into form bean class object and will be made visible in model layer action class in two forms of single object due to this we can say form bean class is action data transfer object or value object class

**What is the diff b/w dependency lookup and dependency injection?**

* **Dependency lookup**:
  + Resource explicitly searches and gathers dependent value from others
  + In dependency lookup resource , pulls the values from others

or

* + If resource of the application spending time to search and get dependent values from other resources of application  then it is called dependency lookup
  + **Example :**
    - If the student get his dependent value material from instruction by asking for it then is called dependency lookup
    - The way servlet / ejb component gets jdbc data source object form registry through jndi lookup operation is called dependency lookup
  + In dependency lookup resource perform **“pull”** operation on other resource or on underlying s/w to get the dependent values
  + **Disadvantage**: resource has to spend time to search and get dependent values
  + **Advantage**: resource can get only required dependent values.
* **Dependency injection:**
  + if underlying s/w (or) framework server (or) special resource of application assign dependent values to resource, the moment resource is ready then it is called dependency injection or inversion of controller (IOC)
  + **Example :**
    - The way student gets his course material the moment to register for course is called dependency injection
    - If jdbc data source object is assigned to servlet object by servlet container the moment servlet object is created is called dependency injection
  + In dependency injection underlying s/w or container/ framework server or special resource perform **“push”** operation on the resource of application to assign dependent values the moment resource is ready
  + **Advantage**: resource need not spend time to get dependent value and it can use dependent value at the moment resource is ready
  + **Disadvantage**: both necessary and unnecessary values will be injected
  + In struts the way action servlet dynamically assign form data to form bean class properties
  + By creating /locating formbean class object is called dependency injection or ioc

The main logic of the app’s Is called as Business logic

Persistence logic is usual to perform CURD operations on DB s/w’s

Spring Modules, What Are Spring Modules

|  |  |  |  |
| --- | --- | --- | --- |
| [Spring](http://www.java4s.com/spring/) » On Aug 4, 2011 By [Sivateja](https://plus.google.com/118054670710951892925?rel=author) |  |  | [prevButton](http://www.java4s.com/spring/what-is-spring-framework/)[nextButton](http://www.java4s.com/spring/spring-core-module-spring-ioc-tutorial/) |

Actually in spring 1.x, the framework has divided into 7 well defined modules.  But in 2.x framework is divided into 6 modules only..

* Spring Core Module
* Spring Context [ J2EE ]
* Spring DAO Module [ Spring JDBC ]
* Spring ORM module
* Spring AOP [ Aspect Oriented Programming ]
* Spring WEB-MVC Module

Actually in spring 1.x, web, mvc are given as separate modules…

Spring Core Module is the base for all modules, and very important

Let us see one by one module in depth….. :-)

**Spring Core Module:-**

Spring Core is the base module for the all of other module of Spring F/W .

Gives a light weight spring container called is called “bean factory”.

Using this module we can develop only stand-alone app’s.

By using core module to understand the SpringBean life cycle management and dependency Injection

The concrete java class that is configured in Spring configuration file through annotation is called “**SpringBean**”. Javabean can act as Spring Bean but every springbean need not be a javabean.

Spring gives two built –in containers and light weight containers

* Bean factory container ( it is a part of core module)
* App’s context container ( part of Spring j2EE/ context Module)

**Dependency Injection:-**

Instead of creating the direct object we need to inject the values from dependent class that is called as Dependency injection

**Container:**

Container is a software app’s or java class that can be care of the whole lifecycle (Birth to Death) of given resource.

Spring container is given to perform spring bean lifecycle management and to perform dependency injection on Spring Bean.

To activate the Bean factory container on our app’s to create object for java class that implement **org.springframework.beans.factory.Beanfactory (**Interface **)**

**EX:-**

**XMLBeanfactory bean=new XMLBeanFactroy(“ ”);**

To activate the app’s context container on our app’s to create object for java class that implement **org.springframework.context.applicationcontext**( Interface)

SystemXMLapplicationContext context= new FileSystemApplicationContext(“ ”);

We can’t create the Servlet container or JSP container by creating the object for certain classes.so they are heavy weight container. Because we need to start heavy weight server to activate this container.

But Spring Container are Lightweight container.

App’s context container is enhancement of Bean factory container

Spring core module app’s contains the following resources

* Spring Interface
* Spring Bean class
* Spring configuration
* Client app’s

**SpringBean:-**

* Spring Bean is a concrete class configured in spring configuration file
* Can be a java class
* Implement Spring interface and provide to implement Spring interface method
* Contains methods, constructor supporting dependency Injection
* Contain utility method
* Contains lifecycle method
* It can be POJO class or non-POJO class
* It can be pre-defiend/user defined/third party concrete java class/java bean
* Contain business method implementation having business logic

Spring Support 3 type of Dependency Injections

1. **Setter Injections:-**

If Spring container calls setXXX() method automatically or dynamically to assign values to the properties (member variables) of spring bean class. Then it is called Setter method

1. **Constructor injections**
2. **Interface Injection**

# Difference between Setter Injection and Constructor Injection

|  |  |  |
| --- | --- | --- |
| **Setter Injection** |  | **Constructor Injection** |
| **1.** In Setter Injection, partial injection of dependencies can possible, means if we have 3 dependencies like int, string, long, then its not necessary to inject all values if we use setter injection. If you are not inject it will takes default values for those primitives |  | **1.** In constructor injection, partial injection of dependencies cannot possible, because for calling constructor we must pass all the arguments right, if not so we may get error |
| **2.**Setter Injection will overrides the constructor injection value, provided if we write setter and constructor injection for the same property [i already told regarding this, hope you remember ] |  | **2.** But, constructor injection cannot overrides the setter injected values |
| **3.** If we have more dependencies for example 15 to 20 are there in our bean class then, in this case setter injection is not recommended as we need to write almost 20 setters right, bean length will increase. |  | **3.** In this case, Constructor injection is highly recommended, as we can inject all the dependencies with in 3 to 4 lines [i mean, by calling one constructor] |
| **4.** Setter injection makes bean class object as mutable [We can change ] |  | **4.** Constructor injection makes bean class object as immutable [We cannot change ] |

# Difference between Setter Injection and Constructor Injection

[November 3, 2017](http://www.codenuclear.com/difference-between-setter-injection-and-constructor-injection/) [codeNuclear](http://www.codenuclear.com/author/codenuclear/)

#### Partial dependency

In Setter Injection, partial injection of dependencies can possible, means if we have 3 dependencies like int, string, long, then its not necessary to inject all values if we use setter injection. If you are not inject it will takes default values for those primitives.

In Constructor injection, partial injection of dependencies cannot possible, because for calling constructor we must pass all the arguments right, if not so we may get error.

#### Overriding

Setter Injection will overrides the constructor injection value, provided if we write setter and constructor injection for the same property.

But, constructor injection cannot overrides the setter injected values.

#### Number of dependencies

If we have more dependencies for example 15 to 20 are there in our bean class then, in this case setter injection is not recommended as we need to write almost 20 setters right, bean length will increase.

In this case, Constructor injection is highly recommended, as we can inject all the dependencies with in 3 to 4 lines by calling one constructor.

#### Changes

Setter injection makes bean class object as mutable. (We can easily change the value by setter injection)

Constructor injection makes bean class object as immutable. (We cannot change the value by Constructor injection)

Difference between constructor injection and setter injection as follows:  
  
**Constructor Injection**  :-

1. No Partial Injection
2. Doesn't override the setter property
3. Creates new instance if any modification occurs
4. Better for too many properties

**Setter Injection :-**

1. Partial Injection
2. Overrides the constructor property if both are defined.
3. Doesn't create new instance if you change the property value
4. Better for few properties.

# Difference between Bean Factory and Application Context in spring?

**BeanFactory**-Does not support the Annotation based dependency Injection.  
**ApplicationContext**--Support Annotation based dependency Injection.-@Autowired, @PreDestroy  
**BeanFactory**-Does not Support  
**ApplicationContext**- Application contexts can publish events to beans that are registered as listeners  
**BeanFactory**-Does not support way to access Message Bundle(internationalization (I18N)   
**ApplicationContext**-Support internationalization (I18N) messages.  
**BeanFactory**-Doesn’t support.  
**ApplicationContext**-Support  many enterprise services such JNDI access, EJB integration, remoting.  
**BeanFactory-**By default its support Lazy loading  
**ApplicationContext**-- its By default support Aggresive loading.

**BeanFactory**container is defined by the BeanFactory interface.  
Whereas

**ApplicationContext** container is defined by the ApplicationContext interface.  
**BeanFactory** container is used for light weight applications.  
Whereas

**ApplicationContext** container is used for heavy weight applications.

ApplicationContext is derived from BeanFactory to provide added functionality to work in web application.

You can instantiate your spring container by just writing

ApplicationContext context = newClassPathXmlApplicationContext("spring.xml");

or

ApplicationContext context = newClassPathXmlApplicationContext{"spring\_dao.xml","spring\_service.xml};

You can use one or more xml file depending on your project requirement. As I am here using two xml files i.e. one for configuration details for service classes other for dao classes. Here ClassPathXmlApplicationContext is child of ApplicationContext

Both BeanFactory and Application are used to manage life cycle of beans, ApplicationContext can do all things that a BeanFactory does along with AOP,Event etc..

#### BeanFactory v/s ApplicationContext in Spring

|  |  |
| --- | --- |
| **BeanFactory (org.springframework.beans)** | **ApplicationContext (org.springframework.context)** |
| Instantiates bean lazily i.e.;  when ctx.getBean(“beanName”) is invoked from application Eg; BeanFactory bf = new XmlBeanFactory(“SpringXml”); | Instantiates bean eagerly i.e.;  upon ApplicationContext startup/loaded. eg; ApplicationContext app = new ClassPathXmlApplicationContext(“SpringXml”); |
| No Supports for I18N. | Supports for Internalization (I18N) |
| No Supports for Annotation | Supports Annotation based Dependency Injection (DI) |
| Recommended to use in small lightweight application like Mobile, Applet, etc | AppplicationContext best suited for enterprise application |
| Very limited access i.e.; low level resources | It is very convenient to load resources from various resources like ClassPath, FileSystem, etc. |
| No such support | Provides generic way to load resources such as Image file |
| Loading multiple configuration files is not possible | In large enterprise project, you have to load multiple configuration files. In that case, ApplicationContext is very good option Eg.; new ClassPathXmlApplicationContext(new String[]{“confg-1”, “confg-2”, “confg-3”}); |

**Spring Bean Scope:-**

Detailed explanation for each scope can be found here in [Spring bean scopes](http://static.springsource.org/spring/docs/3.1.x/spring-framework-reference/html/beans.html#beans-factory-scopes). Below is the summary

**Singleton** - (Default ) Single bean object instance per spring IoC container

**prototype** - Opposite to singleton, it produces a new instance each and every time a bean is requested.

**request** –

A single instance will be created and available during complete lifecycle of an HTTP request.

Only valid in web-aware Spring ApplicationContext.

or

Scopes a single bean definition to the lifecycle of a single HTTP request; that is, each HTTP request has its own instance of a bean created off the back of a single bean definition. Only valid in the context of a web-aware Spring Application Context.

**session** - Scopes a single bean definition to the lifecycle of an HTTP Session. Only valid in the context of a web-aware Spring ApplicationContext.

or

**global session** –

A single instance will be created and available during complete lifecycle of ServletContext.

Only valid in web-aware Spring ApplicationContext

or

Scopes a single bean definition to the lifecycle of a global HTTP Session. Typically only valid when used in a portlet context. Only valid in the context of a web-aware Spring ApplicationContext

<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">

         <!-- Since scope is not defined, it assigns defaultvalue 'singleton'.

     It creates only one instance per spring IoC. -->

    <bean id="article Bean"class="com.java2novice.bean.Article"/>

         <!-- Since scope is prototype, it creates and returns ColorBean forevery

    call-->

    <bean id="clrBean"class="com.java2novice.bean.ColorBean"scope="prototype"/>

        <bean id="ticketBean"class="com.java2novice.bean.TicketManager"scope="singleton"/>

     </beans>

**Spring DAO:-**

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.

The Java class that separates persistence logic from other logics of the application to provide the flexibility of modification to persistence logic is called as DAO.

**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.

**DAO**

1.SpringDAOmoduleprovidesabstraction layeron plain JDBCprogrammingandsimplifiesJDBCstylepersistencelogicdevelopmentthrough JDBCTemplateclass

2.UsingSpringDAOmodulewecan develop persistencelogicbyimplementingDAOdesignpattern.

**PlainJDBCprogramming:**

1. Load JDBC driver class to register JDBC driver with DriverManager Services.

2.Establishcommunication withDatabase software.

3.Create Statement object

4.Send ande xecute SQLquery in Database software

5.Gather and process the results

6.Takecare ofexception handling

7.CloseJDBCobjects

**SpringJDBCprogramming(SpringDAO):**

1.InjectJDBC Templateclassobject.

2.UseJDBC Templateclass objectto send and executeSQLQueriesin Databasesoftware.

3.Gather and process theresultsinSpringJDBCprogramming programmer just need to developApplicationsSpecificlogicsbecause theJDBCTemplate classinternallyuses usesplainJDBCgeneratescommonlogics.

**AdvantagesofSpringDAO:**

1.Theprocessofconverting one form of execption to another form of exception iscalledExceptionRethrowing.

2.InSpring, HibernateFrameworkstheunderlying technologiesgeneratedCheckedexceptions willbeconverted into unChecked exceptionsbyusingExceptionRethrowingconceptasshown in below

**MVC Flow:-**

1. First request will be received by DispatcherServlet.
2. DispatcherServlet will take the help of HandlerMapping and get to know the @Controllerclass name associated with the given request.
3. So request transfer to the @Controller, and then @Controller will process the request by executing appropriate methods and returns ModeAndView object (contains Model data and View name) back to the DispatcherServlet
4. Now DispatcherServlet send the model object to the ViewResolver to get the actual view page.
5. Finally DispatcherServlet will pass the Model object to the View page to display the result

Spring MVC Execution Flow Diagram, Spring MVC 3.2 Flow

|  |  |  |  |
| --- | --- | --- | --- |
| [Spring-MVC](http://www.java4s.com/spring-mvc/) » On Jul 13, 2013 By [Sivateja](https://plus.google.com/118054670710951892925?rel=author) |  |  | [nextButton](http://www.java4s.com/spring-mvc/spring-mvc-hello-world-spring-mvc-3-2-hello-world-example-in-eclipse/) |

Let us see the flow of spring MVC (3.2). I am not going to describe what is M,V,C :-) hope you already know that mess right ? so lets start with the flow…

[](http://www.java4s.com/wp-content/uploads/2013/07/Spring-MVC-execution-flow.png)

**Spring MVC 3.2 Execution Flow**

Step **1**: First request will be received by DispatcherServlet  
Step **2**: DispatcherServlet will take the help of HandlerMapping and get to know the Controller class name associated with the given request  
Step **3**: So request transfer to the Controller, and then controller will process the request by executing appropriate methods and returns ModeAndView object (contains *Model* data and *View* name) back to the DispatcherServlet  
Step 4: Now DispatcherServlet send the model object to the ViewResolver to get the actual view page  
Step **5**: Finally DispatcherServlet will pass the *Model* object to the *View* page to display the result

That’s it :-)

Just remember this diagram for the interview purpose, i will explain you the practical flow in the first example.

## Spring overview

### **1. What is Spring?**

Spring is an open source development framework for [Enterprise Java](http://www.javacodegeeks.com/tutorials/java-tutorials/enterprise-java-tutorials/). The core features of the Spring Framework can be used in developing any Java application, but there are extensions for building web applications on top of the Java EE platform. Spring framework targets to make Java EE development easier to use and promote good programming practice by enabling a [POJO-based programming model](http://www.javacodegeeks.com/2012/09/how-to-write-better-pojo-services.html).

### **2. What are benefits of Spring Framework?**

* **Lightweight:**Spring is lightweight when it comes to size and transparency. The basic version of spring framework is around 2MB.
* **Inversion of control (IOC):** Loose coupling is achieved in Spring, with the [Inversion of Control technique](http://www.javacodegeeks.com/2011/08/what-is-dependency-inversion-is-it-ioc.html). The objects give their dependencies instead of creating or looking for dependent objects.
* **Aspect oriented (AOP):** [Spring supports Aspect oriented programming](http://www.javacodegeeks.com/2011/01/aspect-oriented-programming-spring-aop.html) and separates application business logic from system services.
* **Container:** Spring contains and manages the life cycle and configuration of application objects.
* **MVC Framework:** Spring’s web framework is a well-designed [web MVC framework](http://www.javacodegeeks.com/2011/02/spring-mvc-development-tutorial.html), which provides a great alternative to web frameworks.
* **Transaction Management:** Spring provides a consistent transaction management interface that can scale down to a local transaction and scale up to global transactions (JTA).
* **Exception Handling:** Spring provides a convenient API to translate technology-specific exceptions (thrown by JDBC, Hibernate, or JDO) into consistent, unchecked exceptions.

### **13. What is Spring IoC container?**

The Spring IoC is responsible for creating the objects,managing them (with dependency injection (DI)), wiring them together, configuring them, as also managing their complete lifecycle.

### **14. What are the benefits of IOC?**

IOC or dependency injection minimizes the amount of code in an application. It makes easy to test applications, since no singletons or JNDI lookup mechanisms are required in unit tests. Loose coupling is promoted with minimal effort and least intrusive mechanism. IOC containers support eager instantiation and lazy loading of services.

### **15. What are the common implementations of the ApplicationContext?**

The **FileSystemXmlApplicationContext** container loads the definitions of the beans from an XML file. The full path of the XML bean configuration file must be provided to the constructor.  
The **ClassPathXmlApplicationContext** container also loads the definitions of the beans from an XML file. Here, you need to set CLASSPATH properly because this container will look bean configuration XML file in CLASSPATH.  
The **WebXmlApplicationContext:** container loads the XML file with definitions of all beans from within a web application.

### **27. Explain Bean lifecycle in Spring framework**

* The spring container finds the bean’s definition from the XML file and instantiates the bean.
* Spring populates all of the properties as specified in the bean definition (DI).
* If the bean implements BeanNameAware interface, spring passes the bean’s id to setBeanName() method.
* If Bean implements BeanFactoryAware interface, spring passes the beanfactory to setBeanFactory() method.
* If there are any bean BeanPostProcessors associated with the bean, Spring calls postProcesserBeforeInitialization()method.
* If the bean implements IntializingBean, its afterPropertySet() method is called. If the bean has init method declaration, the specified initialization method is called.
* If there are any BeanPostProcessors associated with the bean, their postProcessAfterInitialization() methods will be called.
* If the bean implements DisposableBean, it will call the destroy() method.

### **28. Which are the important beans lifecycle methods? Can you override them?**

There are two important bean lifecycle methods. The first one is setup which is called when the bean is loaded in to the container. The second method is the teardown method which is called when the bean is unloaded from the container.  
The bean tag has two important attributes (init-method and destroy-method) with which you can define your own custom initialization and destroy methods. There are also the correspondive annotations(@PostConstruct and @PreDestroy).

### **29. What are inner beans in Spring?**

When a bean is only used as a property of another bean it can be declared as an inner bean. Spring’s XML-based configuration metadata provides the use of <bean/> element inside the <property/> or <constructor-arg/> elements of a bean definition, in order to define the so-called inner bean. Inner beans are always anonymous and they are always scoped as prototypes.

### **30. How can you inject a Java Collection in Spring?**

Spring offers the following types of [collection configuration elements](http://examples.javacodegeeks.com/enterprise-java/spring/beans-spring/spring-collections-list-set-map-and-properties-example/):

* The <list> type is used for injecting a list of values, in the case that duplicates are allowed.
* The <set> type is used for wiring a set of values but without any duplicates.
* The <map> type is used to inject a collection of name-value pairs where name and value can be of any type.
* The <props> type can be used to inject a collection of name-value pairs where the name and value are both Strings.

### **31. What is bean wiring?**

Wiring, or else bean wiring is the case when beans are combined together within the Spring container. When wiring beans, the Spring container needs to know what beans are needed and how the container should use dependency injection to tie them together.

### **32. What is bean auto wiring?**

The Spring container is able to [autowire relationships](http://examples.javacodegeeks.com/enterprise-java/spring/beans-spring/spring-autowire-example/) between collaborating beans. This means that it is possible to automatically let Spring resolve collaborators (other beans) for a bean by inspecting the contents of the BeanFactorywithout using <constructor-arg> and <property> elements.

### **33. Explain different modes of auto wiring?**

The autowiring functionality has five modes which can be used to instruct Spring container to use autowiring for dependency injection:

* **no:** This is default setting. Explicit bean reference should be used for wiring.
* **byName:** When autowiring byName, the Spring container looks at the properties of the beans on which autowireattribute is set to byName in the XML configuration file. It then tries to match and wire its properties with the beans defined by the same names in the configuration file.
* **byType:** When autowiring by datatype, the Spring container looks at the properties of the beans on which autowireattribute is set to byType in the XML configuration file. It then tries to match and wire a property if its type matches with exactly one of the beans name in configuration file. If more than one such beans exist, a fatal exception is thrown.
* **constructor:** This mode is similar to byType, but type applies to constructor arguments. If there is not exactly one bean of the constructor argument type in the container, a fatal error is raised.
* **autodetect:**Spring first tries to wire using autowire by constructor, if it does not work, Spring tries to autowirebybyType.

### **48. Types of the transaction management Spring support**

Spring supports two types of transaction management:

* **Programmatic transaction management:** This means that you have managed the transaction with the help of programming. That gives you extreme flexibility, but it is difficult to maintain.
* **Declarative transaction management:** This means you separate [transaction management from the business code](http://www.javacodegeeks.com/2011/09/spring-declarative-transactions-example.html). You only use annotations or XML based configuration to manage the transactions.

### **49. What are the benefits of the Spring Framework’s transaction management?**

* It provides a consistent programming model across different transaction APIs such as JTA, JDBC, Hibernate, JPA, and JDO.
* It provides a simpler API for programmatic transaction management than a number of complex transaction APIs such as JTA.
* It supports declarative transaction management.
* It integrates very well with Spring’s various data access abstractions.

### **50. Which Transaction management type is more preferable?**

Most users of the Spring Framework choose declarative transaction management because it is the option with the least impact on application code, and hence is most consistent with the ideals of a non-invasive lightweight container. Declarative transaction management is preferable over programmatic transaction management though it is less flexible than programmatic transaction management, which allows you to control transactions through your code.

## Spring Model View Controller (MVC)

### **64. What is Spring MVC framework?**

Spring comes with a [full-featured MVC framework for building web applications](http://examples.javacodegeeks.com/enterprise-java/spring/mvc/spring-mvc-hello-world-example/). Although Spring can easily be integrated with other MVC frameworks, such as Struts, Spring’s MVC framework uses IoC to provide a clean separation of controller logic from business objects. It also allows to declaratively bind request parameters to business objects.

### **65. DispatcherServlet**

The Spring Web MVC framework is designed around a DispatcherServlet that handles all the HTTP requests and responses.

### **66. WebApplicationContext**

The WebApplicationContext is an extension of the plain ApplicationContext that has some extra features necessary for web applications. It differs from a normal ApplicationContext in that it is capable of resolving themes, and that it knows which servlet it is associated with.

### **67. What is Controller in Spring MVC framework?**

Controllers provide access to the application behavior that you typically define through a service interface. Controllers interpret user input and transform it into a model that is represented to the user by the view. Spring implements a controller in a very abstract way, which enables you to create a wide variety of controllers.

### **68. @Controller annotation**

The @Controller annotation indicates that a particular class serves the role of a controller. Spring does not require you to extend any controller base class or reference the Servlet API.

### **69. @RequestMapping annotation**

@RequestMapping annotation is used to map a URL to either an entire class or a particular handler method.

**Diff b/w ClassNotFoundException and NoClassDefFoundError**

**1. java.lang.ClassNotFoundException :** This exception indicates that the class was not found on the classpath. This indicates that we were trying to load the class definition, and the class did not exist on the classpath.

**2. java.lang.NoClassDefFoundError :** This exception indicates that the JVM looked in its internal class definition data structure for the definition of a class and did not find it. This is different than saying that it could not be loaded from the classpath. Usually this indicates that we previously attempted to load a class from the classpath, but it failed for some reason – now we’re trying to use the class again (and thus need to load it, since it failed last time), but we’re not even going to try to load it, because we failed loading it earlier (and reasonably suspect that we would fail again). The earlier failure could be a ClassNotFoundException or an ExceptionInInitializerError (indicating a failure in the static initialization block) or any number of other problems. The point is, a NoClassDefFoundError is not necessarily a classpath problem.

**Spring Core:-**

**How to inject null value or Blank value in spring Bean**

To inject an NULL value which is equivalent to bean.setMessage(null):

<bean id=”beanId” class=”bean”>  
<property name=”message”><null/></property>  
</bean>

To inject an empty Stringe which is equivalent to bean.setMessage(“”)

<bean id=”beanId” class=”bean”>  
<property name=”message” value=””></property>  
</bean>

**Spring Bean Scopes – Examples**

There are 5 spring bean scopes as below:

1. Singleton:

If scope is set to singleton, the Spring IoC container creates exactly one instance of the object defined by that bean definition. This single instance is stored in a cache of such singleton beans, and all subsequent requests and references for that named bean return the cached object.

If we not set any scope to bean then spring container will set to the default scope and default scope of the bean is always singleton. use ‘singleton’ to set the bean scope to Singleton.

**2. Prototype**

If scope is set to prototype, the Spring IoC container creates new bean instance of the object every time a request for that specific bean is made.

Use ‘prototype’ word during spring configuration to set the bean scope to Proto Type.

As a rule, use the prototype scope for all state-full beans and the singleton scope for stateless beans.

**3. Request:**

This scopes a bean definition to an HTTP Request and its only valid in the context of a web-aware Spring ApplicationContext.

Use the ‘request’ keyword to set the bean scope to HttpRequest during spring bean configuration.

**4. Session:**

This scopes a bean definition to an HTTP Session and its only valid in the context of a web-aware Spring ApplicationContext.

Use the ‘session’ keyword to set the bean scope to Http Session during spring bean configuration.

**5. Global Session**

This scopes a bean definition to a Global HTTP Session and its only valid in the context of a web-aware Spring ApplicationContext.

Use the ‘global-session’ keyword to set the bean scope to Global Http Session during spring bean configuration.

Configuration of these scopes can be done as below:

Define the above mentioned scopes in beans.xml file along with bean declaration.

<!– A bean definition with singleton scope –>

<bean id=”…” class=”…” scope=”singleton”>

<!– collaborators and configuration for this bean go here –>

</bean>

Can also be mention the bean scope using annotation:

@Configuration

public class BeanJavaConfiguration {

@Bean

@Scope(“prototype”)

public Appleapp() {

return new Apple();

}

}

**Diff b/w HttpSession’s getSession(), getSession(true) and getSession(false) methods ?**

**getSession() :** Returns the current session associated with this request, or if the request does not have a session, creates one.

**getSession(true) :** Returns the current HttpSession associated with this request, if there is no current session, returns a new session

**getSession(false) :** Returns the current HttpSession associated with this request, if there is no current session, returns null

**Spring MVC**

Spring MVC framework is a robust Model view controller framework which helps us to develop a loosely coupled web application. It separates different aspects of web applications with the help of MVC architecture.

**Model:** Model carries application data. It generally includes POJO in the form of business objects

**View:** View is used to render User interface (UI). It will render application data on UI. For example JSP

**Controller:** Controller takes care of processing user request and calling back end services.

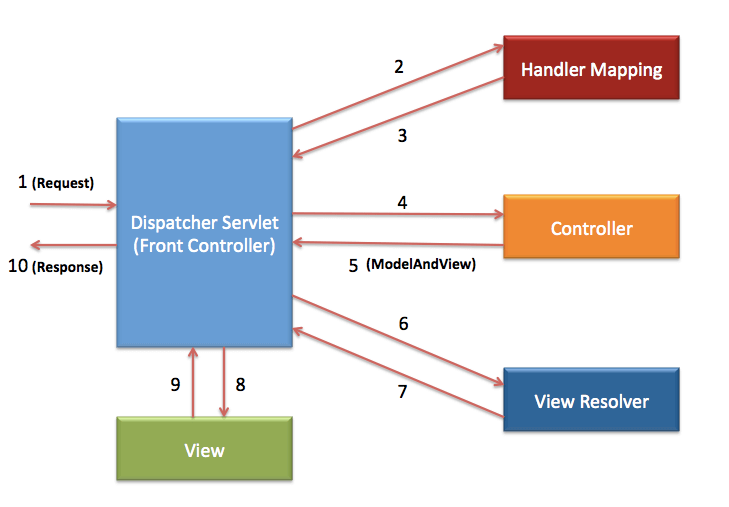
This Spring MVC tutorial is made for beginners as well as for experienced programmer.

**Spring MVC workflow**

Following steps are involved in Spring MVC workflow.

1. The request will be received by Front Controller i.e. **DispatcherServlet**.
2. DispatcherServlet will pass this request to HandlerMapping. **HandlerMapping** will find suitable Controller for the request
3. **HandlerMapping** will send the details of the controller to DispatcherServlet.
4. DispatcherServlet will call the **Controller** identified by HandlerMapping. The **Controller** will process the request by calling appropriate method and prepare the data. It may call some business logic or directly retrieve data from the database.
5. The **Controller** will send **ModelAndView**(Model data and view name) to **DispatcherServlet**.
6. Once DispatcherServlet receives ModelAndView object, it will pass it to **ViewResolver** to find appropriate View.
7. **ViewResolver** will identify the view and send it back to **DispatcherServlet**.
8. **DispatcherServlet** will call appropriate **View** identified by ViewResolver.
9. The **View** will create Response in form of **HTML** and send it to **DispatcherServlet**.
10. **DispatcherServlet** will send the response to the **browser**. The browser will render the html code and display it to **end user**.

Below diagram will make it clearer.



I hope you will have a good understanding of How Spring MVC handles the request and send the response back.

I have written some good examples on Spring MVC. Let me list them down here

**Annotations:-**

**@Controller** :

The @Controller annotation is used to mark the class as the controller in Spring 3.

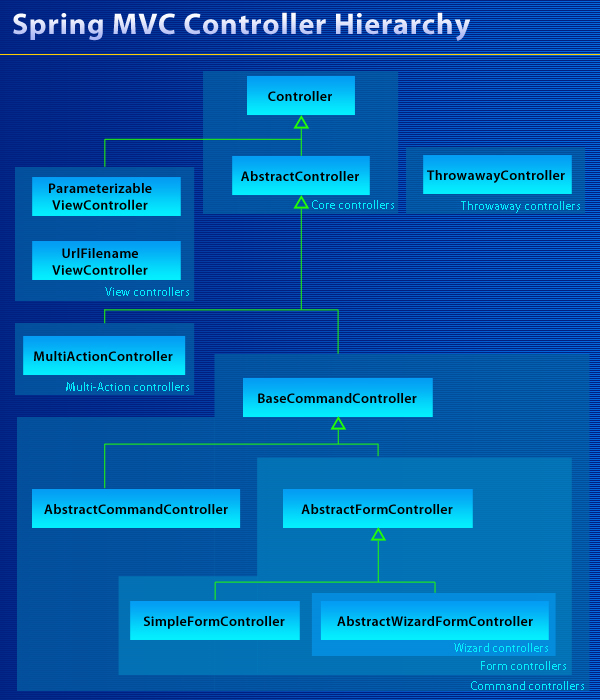
**@RequestMapping**  :

The **@RequestMapping** annotation is used to map the request url. It is applied on the method.

**Spring MVC controller types:-**

Spring MVC provides many abstract controllers, which is designed for specific tasks. Here is the list of anstract controllers that comes with the Spring MVC module:

* SimpleFormController
* AbstractController
* AbstractCommandController
* CancellableFormController
* AbstractCommandController
* MultiActionController
* ParameterizableViewController
* ServletForwardingController
* ServletWrappingController
* UrlFilenameViewController
* AbstractController
* AbstractCommandController
* SimpleFormController
* CancellableFormController.



**Types of ViewReslovers in Spring MVC:-**

1. **AbstractCachingViewResolver :** Abstract view resolver that caches views. Often views need preparation before they can be used; extending this view resolver provides caching.
2. **XmlViewResolver :**Implementation of ViewResolver that accepts a configuration file written in XML with the same DTD as Spring’s XML bean factories. The default configuration file is /WEB-INF/views.xml.
3. **ResourceBundleViewResolver :**Implementation of ViewResolver that uses bean definitions in a ResourceBundle, specified by the bundle base name. Typically you define the bundle in a properties file, located in the classpath. The default file name is views.properties.
4. **UrlBasedViewResolver :**Simple implementation of the ViewResolver interface that effects the direct resolution of logical view names to URLs, without an explicit mapping definition. This is appropriate if your logical names match the names of your view resources in a straightforward manner, without the need for arbitrary mappings.
5. **InternalResourceViewResolver :**Convenient subclass of UrlBasedViewResolver that supports InternalResourceView (in effect, Servlets and JSPs) and subclasses such as JstlView and TilesView. You can specify the view class for all views generated by this resolver by using setViewClass(..).
6. **VelocityViewResolver/FreeMarkerViewResolver :**Convenient subclass of UrlBasedViewResolver that supports VelocityView (in effect, Velocity templates) or FreeMarkerView ,respectively, and custom subclasses of them.
7. **ContentNegotiatingViewResolver :**Implementation of the ViewResolver interface that resolves a view based on the request file name or Accept header.

InternalResourceViewResolver is a subclass of UrlBasedViewResolver.

UrlBasedViewResolver and InternalResourceViewResolver are often used in MVC application where the controller return the name of the view that should been rendered. The controller return an logical name of the view, and the resolver made it a file name (of the jsp), by adding some pre - and postfix. For example: logical view name return by the controller: main/example, prefix: /WEB-INF/pages/, postfix: .jsp -> /WEB-INF/pages/main/example.jsp gets rendered with the model-data provided by the controller

* The UrlBasedViewResolver needs a [View](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/servlet/View.html) class (like the most other ViewResolvers too). *(very brif: The view class is responsible for rendering, while the resolver is responsible to pick the right template/...)* The view used in UrlBasedViewResolver has to be an subclass of [AbstractUrlBasedView](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/servlet/view/AbstractUrlBasedView.html).
* The InternalResourceViewResolver is convenient subclass of UrlBasedViewResolver that has be default already a configured view: [InternalResourceView](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/servlet/view/InternalResourceView.html) (or [JstlView](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/servlet/view/JstlView.html) when Jstl is present). So it is the right resolver when JSPs are used as template engine.

*There are other AbstractUrlBasedView implementations, for example for JasperReports, Freemaker, Velocity, Tiles, .... Most of them has a convenient subclass of UrlBasedViewResolvertoo.*

BeanNameViewResolver very very old resolvers, from the old Spring 2.0 time. At this time each controller was for handling one URL. At this time there was no @RequestMapping annotation, and one has to tell Spring which controller was for which url. One could list them all, or have this Resolver, that was able to map url->controller by the controller name. Since Spring 3.0 (more exact since 2.5) and Springs annotation support, this resolver is used very rarely.

<https://malliktalksjava.com/2009/07/09/log4j/>

<http://www.javainterviewpoint.com/spring-mvc-beannameurlhandlermapping-example/>

**Types of Handler Mapping in Spring MVC:-**

**HandlerMapping** is an Interface to be implemented by objects that define a mapping between requests and handler objects.

Bydefault **DispatcherServlet**uses **BeanNameUrlHandlerMapping** and **DefaultAnnotationHandlerMapping**.

In spring we majorly use the below handler mappings

1. BeanNameUrlHandlerMapping
2. ControllerClassNameHandlerMapping
3. SimpleUrlHandlerMapping

Let’s take **BeanNameUrlHandlerMapping** in this article. Here we will be mapping each request to a Bean directly like below

<bean class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"/>

<bean name="/helloWorld.htm"

class="com.javainterviewpoint.HelloWorldController" />

<bean name="/hello\*.htm"

class="com.javainterviewpoint.HelloWorldController" />

Here we can see we have mentioned the [**Spring container**](https://www.javainterviewpoint.com/category/spring-core/) to use **BeanNameUrlHandlerMapping** and we have mapped each possible request to a controller.

### **SpringConfig-servlet.xml**

* The **SpringConfig-servlet.xml** is also placed under the WEB-INF directory.
* Here we have configured **BeanNameUrlHandlerMapping** as the HandlerMapping
* Each request is mapped to a Controller as well

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:mvc="http://www.springframework.org/schema/mvc"

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

http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.0.xsd

http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="prefix" value="/WEB-INF/Jsp/"/>

<property name="suffix" value=".jsp"/>

</bean>

<bean class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"/>

<bean name="/helloWorld.htm"

class="com.javainterviewpoint.HelloWorldController" />

<bean name="/hello\*"

class="com.javainterviewpoint.HelloWorldController" />

<bean name="/welcome.htm"

class="com.javainterviewpoint.WelcomeController"/>

</beans>

2. **ControllerClassNameHandlerMapping in Spring MVC:-**

**ControllerClassHandlerMapping,**this type of **HandlerMapping** uses a convention to map the requested URL to the Controller. It will take the **Controller name** and **converts them to** **lower case**with a **leading “/”**

**Using BeanNameUrlHandlerMapping**

<bean class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"/>

<bean name="/helloWorld.htm"

class="com.javainterviewpoint.HelloWorldController" />

<bean name="/hello\*.htm"

class="com.javainterviewpoint.HelloWorldController" />

**Using ControllerClassNameHandlerMapping**

 <bean class="org.springframework.web.servlet.mvc.support.ControllerClassNameHandlerMapping" />

<bean class="com.javainterviewpoint.HelloWorldController"></bean>

<bean class="com.javainterviewpoint.WelcomeController"></bean>

When using **ControllerClassNameHandlerMapping**, there is no need for  **bean name**

**Spring MVC SimpleUrlHandlerMapping Example**

**SimpleUrlHandlerMapping,**this type of HandlerMapping is the simplest of all handler mappings which allows you specify URL pattern and handler explicitly  
There are two ways of defining **SimpleUrlHandlerMapping,**using **<value>**tag and **<props>** tag. **SimpleUrlHandlerMapping**has a property called **mappings** we will be passing the URL pattern to it.

**Using <value> tag**

**Left Side** of  **“=”**is **URL Pattern** and **right side** is the **id or name**of the bean

<bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">

<property name="mappings">

<value>

/welcome.htm=welcomeController

/welcome\*=welcomeController

/hell\*=helloWorldController

/helloWorld.htm=helloWorldController

</value>

</property>

</bean>

**Using <props> tag**

The **property key** is the **URL Pattern** and **property value** is the **id or name**of the bean

 <bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">

<property name="mappings">

<props>

<prop key="/welcome.htm">welcomeController</prop>

<prop key="/welcome\*">welcomeController</prop>

<prop key="/helloworld">helloWorldController</prop>

<prop key="/hello\*">helloWorldController</prop>

<prop key="/HELLOworld">helloWorldController</prop>

</props>

</property>

</bean>

**SpringConfig-servlet.xml**

* The **SpringConfig-servlet.xml** is also placed under the WEB-INF directory.
* Here we have configured**SimpleUrlHandlerMapping** as the HandlerMapping
* Each request is mapped to a Controller as well

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:mvc="http://www.springframework.org/schema/mvc"

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

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.0.xsd

http://www.springframework.org/schema/mvc

http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="prefix" value="/WEB-INF/Jsp/"/>

<property name="suffix" value=".jsp"/>

</bean>

<bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">

<property name="mappings">

<props>

<prop key="/welcome.htm">welcomeController</prop>

<prop key="/welcome\*">welcomeController</prop>

<prop key="/helloworld">helloWorldController</prop>

<prop key="/hello\*">helloWorldController</prop>

<prop key="/HELLOworld">helloWorldController</prop>

</props>

</property>

</bean>

<bean id="helloWorldController" class="com.javainterviewpoint.HelloWorldController"></bean>

<bean id="welcomeController" class="com.javainterviewpoint.WelcomeController"></bean>

</beans>

In the above example where ever

* **helloworld** is requested, the DispatcherServlet redirects it to the **HelloWorldController**.
* **hello123**is requested, the DispatcherServlet redirects it to the **HelloWorldController**.
* **HELLOworld**is requested, the DispatcherServlet redirects it to the **HelloWorldController**.
* **welcome.htm** is requested, the DispatcherServlet redirects it to the **WelcomeController**.
* **welcome123** is requested, the DispatcherServlet redirects it to the **WelcomeController**.
* **hELLOWorld**is requested, you will get **404** error as we have add mapping for it.