**How to read application.properties variables?**

The **easiest way** to read a value from the property file or yml is to use the spring @value annotation.Spring automatically loads all values from the yml to the spring environment , so we can directly use those values from the environment like :

@Component

public class MySampleBean {

@Value("${name}")

private String sampleName;

// ...

}

# [**Difference between using bean id and name in Spring configuration file**](https://stackoverflow.com/questions/874505/difference-between-using-bean-id-and-name-in-spring-configuration-file)

What will happen if two bean has same id?

Every bean has one or more ids (also called identifiers, or names; these terms refer to the same thing). These ids must be unique within the container the bean is hosted in. A bean will almost always have only one id, but if a bean has more than one id, the extra ones can essentially be considered aliases.

When using XML-based configuration metadata, you use the 'id' or 'name' attributes to specify the bean identifier(s). The 'id' attribute allows you to specify exactly one id, and as it is a real XML element ID attribute, the XML parser is able to do some extra validation when other elements reference the id; as such, it is the preferred way to specify a bean id. However, the XML specification does limit the characters which are legal in XML IDs. This is usually not a constraint, but if you have a need to use one of these special XML characters, or want to introduce other aliases to the bean, you may also or instead specify one or more bean ids, separated by a comma (,), semicolon (;), or whitespace in the 'name' attribute.

Both id and name attributes are giving us a means to provide identifier to particular bean (For this moment, think id means id but not as identifier). you can choose any of the attribute to provide identifier value. In both the cases, you will see same result if you call applicationContext.getBean("bean-identifier"); .

Take @Bean, the java equivalent of <bean> tag, there is no id attribute. you can give your identifier value only through name attribute.

Let me explain it through an example, consider the folowing code to get a bean,  
  
FileSystemXmlApplicationContext context = new FileSystemXmlApplicationContext(...); Foo f = (Foo) context.getBean("foo")// returns Foo object;   
If configuration is provided as   
spring1.xml

<?xml version="1.0" encoding="UTF-8"?>

<beans ...>

<bean id="foo" class="com.intertech.Foo"></bean>

<bean id="bar" class="com.intertech.Bar"></bean>

</beans>

Spring returns Foo object for, Foo f = (Foo) context.getBean("foo");

replace id="foo" with name="foo" in the above spring1.xml

You can still see the same result,   
Spring returns Foo object for, Foo f = (Foo) context.getBean("foo");

Define your xml configuration like,

<?xml version="1.0" encoding="UTF-8"?>

<beans ...>

<bean id="fooIdentifier" class="com.intertech.Foo"></bean>

<bean name="fooIdentifier" class="com.intertech.Foo"></bean>

</beans>

You will get BeanDefinitionParsingException, it will say, Bean name 'fooIdentifier' is already used in this element.  
The same exception will be thrown if   
<bean name="fooIdentifier" class="com.intertech.Foo"></bean>  
<bean name="fooIdentifier" class="com.intertech.Foo"></bean>  
is passed as the configuration   
  
if you keep both id and name to the bean tag, the bean is said to have 2 identifiers, you can get the same bean with any identifier. take config as

<?xml version="1.0" encoding="UTF-8"?><br>

<beans ...>

<bean id="fooById" name="fooByName" class="com.intertech.Foo"></bean>

<bean id="bar" class="com.intertech.Bar"></bean>

</beans>

the following code prints true

FileSystemXmlApplicationContext context = new FileSystemXmlApplicationContext(...);

Foo fooById = (Foo) context.getBean("fooById")// returns Foo object;

Foo fooByName = (Foo) context.getBean("fooByName")// returns Foo object;

System.out.println(fooById == fooByName) //true

# **Spring Dependency Injection Annotation Example, Beans Auto-wiring using @Autowired, @Qualifier & @Resource Annotations Configuration**

**Created on:**August 25, 2014 | **Last updated on:**September 30, 2017  [websystiqueadmin](http://websystique.com/author/blogadminuser/)

This article shows Spring Dependency Injection Annotation example + Spring Auto-wiring Annotation example.  
We will learn Spring Dependency Injection and Beans auto-wiring using Spring @Autowired annotation. **@Autowired** can be applied on a bean’s constructor, field, setter method or a config method to autowire the dependency using Spring’s dependency injection.

@Autowired wires the dependency using bean datatype. If you are looking for bean wiring using bean name (somewhat analogues to byName in XML), you can use standard @Resource annotation with ‘name’ attribute. @Qualifier annotation is often used in conjunction with @Autowired to resolve ambiguity in case more that one bean of injected type exist in application context. Let’s get going.

**Other interesting posts you may like**

#### 1. @Resource Example (autowiring by name)

**Define Beans to work with**

|  |
| --- |
| package com.websystique.spring.domain;    import javax.annotation.Resource;    import org.springframework.stereotype.Component;    @Component("application")  public class Application {        @Resource(name="applicationUser")      private ApplicationUser user;        @Override      public String toString() {          return "Application [user=" + user + "]";      }  } |
| package com.websystique.spring.domain;    import org.springframework.stereotype.Component;    @Component("applicationUser")  public class ApplicationUser {        private String name = "defaultName";        public String getName() {          return name;      }        public void setName(String name) {          this.name = name;      }        @Override      public String toString() {          return "ApplicationUser [name=" + name + "]";      }  } |

Standard @Resource annotation marks a resource that is needed by the application. It is analogous to @Autowired in that both injects beans by type when no attribute provided. But with name attribute, @Resource allows you to inject a bean by it’s name, which @Autowired does not.

In above code, Application’s user property is annotated with @Resource(name=”applicationUser”). In this case, a bean with name ‘applicationUser’ found in applicationContext will be injected here.

**Spring Configuration Class**

|  |
| --- |
| package com.websystique.spring.config;    import org.springframework.context.annotation.ComponentScan;  import org.springframework.context.annotation.Configuration;    @Configuration  @ComponentScan("com.websystique.spring")  public class AppConfig {    } |

Notice @ComponentScan which will make Spring auto detect the annotated beans via scanning the specified package and wire them wherever needed (using @Resource or @Autowired ).

Above configuration is analogues to following in XML

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns="<a class="vglnk" href="<http://www.springframework.org/schema/beans>" rel="nofollow"><span>http</span><span>://</span><span>www</span><span>.</span><span>springframework</span><span>.</span><span>org</span><span>/</span><span>schema</span><span>/</span><span>beans</span></a>"      xmlns:xsi="<a class="vglnk" href="<http://www.w3.org/2001/XMLSchema-instance>" rel="nofollow"><span>http</span><span>://</span><span>www</span><span>.</span><span>w3</span><span>.</span><span>org</span><span>/</span><span>2001</span><span>/</span><span>XMLSchema</span><span>-</span><span>instance</span></a>" xmlns:context="<a class="vglnk" href="<http://www.springframework.org/schema/context>" rel="nofollow"><span>http</span><span>://</span><span>www</span><span>.</span><span>springframework</span><span>.</span><span>org</span><span>/</span><span>schema</span><span>/</span><span>context</span></a>"      xsi:schemaLocation="<a class="vglnk" href="<http://www.springframework.org/schema/beans>" rel="nofollow"><span>http</span><span>://</span><span>www</span><span>.</span><span>springframework</span><span>.</span><span>org</span><span>/</span><span>schema</span><span>/</span><span>beans</span></a> <a class="vglnk" href="<http://www.springframework.org/schema/beans/spring-beans-4.0.xsd>" rel="nofollow"><span>http</span><span>://</span><span>www</span><span>.</span><span>springframework</span><span>.</span><span>org</span><span>/</span><span>schema</span><span>/</span><span>beans</span><span>/</span><span>spring</span><span>-</span><span>beans</span><span>-</span><span>4</span><span>.</span><span>0</span><span>.</span><span>xsd</span></a>      <a class="vglnk" href="<http://www.springframework.org/schema/context>" rel="nofollow"><span>http</span><span>://</span><span>www</span><span>.</span><span>springframework</span><span>.</span><span>org</span><span>/</span><span>schema</span><span>/</span><span>context</span></a> <a class="vglnk" href="<http://www.springframework.org/schema/context/spring-context-4.0.xsd>" rel="nofollow"><span>http</span><span>://</span><span>www</span><span>.</span><span>springframework</span><span>.</span><span>org</span><span>/</span><span>schema</span><span>/</span><span>context</span><span>/</span><span>spring</span><span>-</span><span>context</span><span>-</span><span>4</span><span>.</span><span>0</span><span>.</span><span>xsd</span></a>">        <context:component-scan base-package="com.websystique.spring" />    </beans> |

**Run Application**

Load the context and run it.

|  |
| --- |
| package com.websystique.spring;    import org.springframework.context.annotation.AnnotationConfigApplicationContext;  import org.springframework.context.support.AbstractApplicationContext;    import com.websystique.spring.config.AppConfig;  import com.websystique.spring.domain.Application;    public class AppMain {        public static void main(String args[]){          AbstractApplicationContext  context = new AnnotationConfigApplicationContext(AppConfig.class);            //Byname Autowiring          Application application = (Application)context.getBean("application");          System.out.println("Application Details : "+application);        }    } |

Following will be the output.

|  |
| --- |
| Application Details : Application [user=ApplicationUser [name=defaultName]] |

#### 2. @Autowired Example

**Define Beans to work with**

|  |
| --- |
| package com.websystique.spring.domain;    import org.springframework.stereotype.Component;    @Component  public class License {        private String number="123456ABC";        @Override      public String toString() {          return "License [number=" + number + "]";      }      //setters, getters  } |

**@Autowired on Setter method**

|  |
| --- |
| package com.websystique.spring.domain;    import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Component;    @Component("driver")  public class Driver {        private License license;        @Autowired      public void setLicense(License license) {          this.license = license;      }        @Override      public String toString() {          return "Driver [license=" + license + "]";      }      //getter  } |

**@Autowired on Field**

|  |
| --- |
| package com.websystique.spring.domain;    import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Component;    @Component("driver")  public class Driver {      @Autowired      private License license;        //getter,setter        @Override      public String toString() {          return "Driver [license=" + license + "]";      }  } |

**@Autowired on Constructor**

|  |
| --- |
| package com.websystique.spring.domain;    import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Component;    @Component("driver")  public class Driver {        private License license;        @Autowired      public Driver(License license){          this.license = license;      }        @Override      public String toString() {          return "Driver [license=" + license + "]";      }  } |

**Run Application**  
Load the context and Run it.

|  |
| --- |
| package com.websystique.spring;    import org.springframework.context.annotation.AnnotationConfigApplicationContext;  import org.springframework.context.support.AbstractApplicationContext;    import com.websystique.spring.config.AppConfig;  import com.websystique.spring.domain.Driver;    public class AppMain {        public static void main(String args[]) {          AbstractApplicationContext context = new AnnotationConfigApplicationContext(                  AppConfig.class);            Driver driver = (Driver) context.getBean("driver");          System.out.println("Driver Details : " + driver);      }    } |

Following will be the output

|  |
| --- |
| Driver Details : Driver [license=License [number=123456ABC]] |

#### 3. @Qualifier Example

@Qualifier is useful for the situation where you have more than one bean matching the type of dependency and thus resulting in ambiguity.

**Define Beans to work with**

|  |
| --- |
| package com.websystique.spring.domain;    public interface Car {        public void getCarName();    } |
| package com.websystique.spring.domain;    import org.springframework.stereotype.Component;    @Component("Ferari")  public class Ferari implements Car{        public void getCarName() {          System.out.println("This is Ferari");      }    } |

|  |
| --- |
| package com.websystique.spring.domain;    import org.springframework.stereotype.Component;    @Component("Mustang")  public class Mustang implements Car{        public void getCarName() {          System.out.println("This is Mustang");      }    } |
| package com.websystique.spring.domain;    import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.beans.factory.annotation.Qualifier;  import org.springframework.stereotype.Component;    @Component  public class Bond {        @Autowired      private Car car;        public void showCar(){          car.getCarName();      }  } |

**Run Applicaion**

Load context and Run it.

|  |
| --- |
| package com.websystique.spring;    import org.springframework.context.annotation.AnnotationConfigApplicationContext;  import org.springframework.context.support.AbstractApplicationContext;    import com.websystique.spring.config.AppConfig;  import com.websystique.spring.domain.Bond;    public class AppMain {        public static void main(String args[]) {          AbstractApplicationContext context = new AnnotationConfigApplicationContext(                  AppConfig.class);            Bond bond = (Bond) context.getBean("bond");          bond.showCar();      }    } |

On running, Spring throws following exception:

|  |
| --- |
| Caused by: org.springframework.beans.factory.NoUniqueBeanDefinitionException: No qualifying bean of type [com.websystique.spring.domain.Car] is defined: expected single matching bean but found 2: Ferari,Mustang      at org.springframework.beans.factory.support.DefaultListableBeanFactory.doResolveDependency(DefaultListableBeanFactory.java:970)      at org.springframework.beans.factory.support.DefaultListableBeanFactory.resolveDependency(DefaultListableBeanFactory.java:858)      at org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor$AutowiredFieldElement.inject(AutowiredAnnotationBeanPostProcessor.java:480)      ... 14 more |

What happened is Spring was not able to decide which bean (Ferari or Mustang as both implements Car) to choose for auto-wiring ,it throws this exception.

Happily, @Qualifier saves the day.

Change the Bond class as shown below

|  |
| --- |
| package com.websystique.spring.domain;    import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.beans.factory.annotation.Qualifier;  import org.springframework.stereotype.Component;    @Component  public class Bond {        @Autowired      @Qualifier("Mustang")      private Car car;        public void showCar(){          car.getCarName();      }  } |

**Run Application**

Following will be the output.

|  |
| --- |
| This is Mustang |

#### Mark Autowiring optional with attribute required=”false”

By default, @Autowored annotation makes sure that field is indeed autowired. In case autowiring is not successful, Spring will throw an exception. There are times however when you want to make autowiring optional. Setting @Autowired required attribute to ‘false’ will make this filed optional for autowiring and Spring will skip it(remain null) if dependency not found.

|  |
| --- |
| package com.websystique.spring.domain;    import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Component;    @Component("driver")  public class Driver {      @Autowired(required=false)      private License license;        //getter,setter        @Override      public String toString() {          return "Driver [license=" + license + "]";      }  } |

In above example, if no bean of type License been found, it will remain null and no error will be thrown on context loading.

***caveat :***  
Note that standard @Resource annotation does not have this flexibility. In case the dependency annotated with @Resource not found, Spring will throw an exception. Both @Resource and @Autowired have few differences : No optionality in @Resource and no autowiring by bean name in @Autowired.  
All in all, @Autowired is the most widely used option compare to @Resource and autowire attribute in XML.

**What is Spring Framework?**

Spring is one of the most widely used Java EE framework. Spring framework core concepts are “Dependency Injection” and “Aspect Oriented Programming”.

Spring framework can be used in normal java applications also to achieve loose coupling between different components by implementing dependency injection and we can perform cross cutting tasks such as logging and authentication using spring support for aspect oriented programming.

I like spring because it provides a lot of features and different modules for specific tasks such as Sprintg MVC and Spring JDBC. Since it’s an open source framework with a lot of online resources and active community members, working with Spring framework is easy and fun at same time.

1. **What are some of the important features and advantages of Spring Framework?**

Spring Framework is built on top of two design concepts – Dependency Injection and Aspect Oriented Programming.

Some of the features of spring framework are:

* + Lightweight and very little overhead of using framework for our development.
  + Dependency Injection or Inversion of Control to write components that are independent of each other, spring container takes care of wiring them together to achieve our work.
  + Spring IoC container manages Spring Bean life cycle and project specific configurations such as JNDI lookup.
  + Spring MVC framework can be used to create web applications as well as restful web services capable of returning XML as well as JSON response.
  + Support for transaction management, JDBC operations, File uploading, Exception Handling etc with very little configurations, either by using annotations or by spring bean configuration file.

Some of the advantages of using Spring Framework are:

* + Reducing direct dependencies between different components of the application, usually Spring IoC container is responsible for initializing resources or beans and inject them as dependencies.
  + Writing unit test cases are easy in Spring framework because our business logic doesn’t have direct dependencies with actual resource implementation classes. We can easily write a test configuration and inject our mock beans for testing purposes.
  + Reduces the amount of boiler-plate code, such as initializing objects, open/close resources. I like JdbcTemplate class a lot because it helps us in removing all the boiler-plate code that comes with JDBC programming.
  + Spring framework is divided into several modules, it helps us in keeping our application lightweight. For example, if we don’t need Spring transaction management features, we don’t need to add that dependency in our project.
  + Spring framework support most of the Java EE features and even much more. It’s always on top of the new technologies, for example there is a Spring project for Android to help us write better code for native android applications. This makes spring framework a complete package and we don’t need to look after different framework for different requirements.

1. **What do you understand by Dependency Injection?**

Dependency Injection design pattern allows us to remove the hard-coded dependencies and make our application loosely coupled, extendable and maintainable. We can implement dependency injection pattern to move the dependency resolution from compile-time to runtime.

Some of the benefits of using Dependency Injection are: Separation of Concerns, Boilerplate Code reduction, Configurable components and easy unit testing.

Read more at [Dependency Injection Tutorial](http://www.journaldev.com/2394/java-dependency-injection-design-pattern-example-tutorial). We can also use [Google Guice for Dependency Injection](http://www.journaldev.com/2403/google-guice-dependency-injection-example-tutorial) to automate the process of dependency injection. But in most of the cases we are looking for more than just dependency injection and that’s why Spring is the top choice for this.

1. **How do we implement DI in Spring Framework?**

We can use Spring XML based as well as Annotation based configuration to implement DI in spring applications. For better understanding, please read [Spring Dependency Injection](http://www.journaldev.com/2410/spring-dependency-injection) example where you can learn both the ways with JUnit test case. The post also contains sample project zip file, that you can download and play around to learn more.

1. **What are the benefits of using Spring Tool Suite?**

We can install plugins into Eclipse to get all the features of Spring Tool Suite. However STS comes with Eclipse with some other important stuffs such as Maven support, Templates for creating different types of Spring projects and tc server for better performance with Spring applications.

I like STS because it highlights the Spring components and if you are using AOP pointcuts and advices, then it clearly shows which methods will come under the specific pointcut. So rather than installing everything on our own, I prefer using STS when developing Spring based applications.

1. **Name some of the important Spring Modules?**

Some of the important Spring Framework modules are:

* + **Spring Context** – for dependency injection.
  + **Spring AOP** – for aspect oriented programming.
  + **Spring DAO** – for database operations using DAO pattern
  + **Spring JDBC** – for JDBC and DataSource support.
  + **Spring ORM** – for ORM tools support such as Hibernate
  + **Spring Web Module** – for creating web applications.
  + **Spring MVC** – Model-View-Controller implementation for creating web applications, web services etc.

1. **What do you understand by Aspect Oriented Programming?**

Enterprise applications have some common cross-cutting concerns that is applicable for different types of Objects and application modules, such as logging, transaction management, data validation, authentication etc. In Object Oriented Programming, modularity of application is achieved by Classes whereas in AOP application modularity is achieved by Aspects and they are configured to cut across different classes methods.

AOP takes out the direct dependency of cross-cutting tasks from classes that is not possible in normal object oriented programming. For example, we can have a separate class for logging but again the classes will have to call these methods for logging the data. Read more about Spring AOP support at [Spring AOP Example](http://www.journaldev.com/2583/spring-aop-example-tutorial-aspect-advice-pointcut-joinpoint-annotations).

1. **What is Aspect, Advice, Pointcut, JointPoint and Advice Arguments in AOP?**

**Aspect**: Aspect is a class that implements cross-cutting concerns, such as transaction management. Aspects can be a normal class configured and then configured in Spring Bean configuration file or we can use Spring AspectJ support to declare a class as Aspect using @Aspect annotation.

**Advice**: Advice is the action taken for a particular join point. In terms of programming, they are methods that gets executed when a specific join point with matching pointcut is reached in the application. You can think of Advices as [Spring interceptors](http://www.journaldev.com/2676/spring-mvc-interceptor-example-handlerinterceptor-handlerinterceptoradapter) or [Servlet Filters](http://www.journaldev.com/1933/java-servlet-filter-example-tutorial).

**Pointcut**: Pointcut are regular expressions that is matched with join points to determine whether advice needs to be executed or not. Pointcut uses different kinds of expressions that are matched with the join points. Spring framework uses the AspectJ pointcut expression language for determining the join points where advice methods will be applied.

**Join Point**: A join point is the specific point in the application such as method execution, exception handling, changing object variable values etc. In Spring AOP a join points is always the execution of a method.

**Advice Arguments**: We can pass arguments in the advice methods. We can use args() expression in the pointcut to be applied to any method that matches the argument pattern. If we use this, then we need to use the same name in the advice method from where argument type is determined.

These concepts seems confusing at first, but if you go through [Spring Aspect, Advice Example](http://www.journaldev.com/2583/spring-aop-example-tutorial-aspect-advice-pointcut-joinpoint-annotations) then you can easily relate to them.

1. **What is the difference between Spring AOP and AspectJ AOP?**

AspectJ is the industry-standard implementation for Aspect Oriented Programming whereas Spring implements AOP for some cases. Main differences between Spring AOP and AspectJ are:

* + Spring AOP is simpler to use than AspectJ because we don’t need to worry about the weaving process.
  + Spring AOP supports AspectJ annotations, so if you are familiar with AspectJ then working with Spring AOP is easier.
  + Spring AOP supports only proxy-based AOP, so it can be applied only to method execution join points. AspectJ support all kinds of pointcuts.
  + One of the shortcoming of Spring AOP is that it can be applied only to the beans created through Spring Context.

1. **What is Spring IoC Container?**

**Inversion of Control** (IoC) is the mechanism to achieve loose-coupling between Objects dependencies. To achieve loose coupling and dynamic binding of the objects at runtime, the objects define their dependencies that are being injected by other assembler objects. Spring IoC container is the program that injects dependencies into an object and make it ready for our use.

Spring Framework IoC container classes are part of org.springframework.beans and org.springframework.context packages and provides us different ways to decouple the object dependencies.

Some of the useful ApplicationContext implementations that we use are;

* + AnnotationConfigApplicationContext: For standalone java applications using annotations based configuration.
  + ClassPathXmlApplicationContext: For standalone java applications using XML based configuration.
  + FileSystemXmlApplicationContext: Similar to ClassPathXmlApplicationContext except that the xml configuration file can be loaded from anywhere in the file system.
  + AnnotationConfigWebApplicationContext and XmlWebApplicationContext for web applications.

1. **What is a Spring Bean?**

Any normal java class that is initialized by Spring IoC container is called Spring Bean. We use Spring ApplicationContext to get the Spring Bean instance.

Spring IoC container manages the life cycle of Spring Bean, bean scopes and injecting any required dependencies in the bean.

1. **What is the importance of Spring bean configuration file?**

We use Spring Bean configuration file to define all the beans that will be initialized by Spring Context. When we create the instance of Spring ApplicationContext, it reads the spring bean xml file and initialize all of them. Once the context is initialized, we can use it to get different bean instances.

Apart from Spring Bean configuration, this file also contains spring MVC interceptors, view resolvers and other elements to support annotations based configurations**.**

# Spring Bean Life Cycle Explained

By Lokesh Gupta | Filed Under: [Spring Core](https://howtodoinjava.com/spring-core/)

In this article, learn about **Spring bean life cycle**. We will learn about life cycle stages, initialization and destroy call back methods. We will learn to control the bean life cycle events using XML configuration as well as annotation configuration.

## 1. Types of bean life cycle events

Spring bean factory is responsible for managing the life cycle of beans created through spring container. The life cycle of beans consist of **call back methods** which can be categorized broadly in two groups:

* **Post initialization** call back methods
* **Pre destruction** call back methods

#### 1.1. Spring bean life cycle diagram

Spring Bean Life Cycle

## 2. Spring Bean Life Cycle Callback Methods

Spring framework provides following **4 ways for controlling life cycle events** of a bean:

1. InitializingBean and DisposableBean callback interfaces
2. \*Aware interfaces for specific behavior
3. Custom init() and destroy() methods in bean configuration file
4. @PostConstruct and @PreDestroy annotations

#### 2.1. InitializingBean and DisposableBean callback interfaces

The [org.springframework.beans.factory.InitializingBean](http://static.springsource.org/spring/docs/3.0.x/javadoc-api/org/springframework/beans/factory/InitializingBean.html" \o "InitializingBean) interface allows a bean to perform initialization work after all necessary properties on the bean have been set by the container.

The InitializingBean interface specifies a single method:

|  |
| --- |
| InitializingBean.java |
| void afterPropertiesSet() throws Exception; |

This is not a preferrable way to initialize the bean because it tightly couple your bean class with spring container. A better approach is to use “init-method” attribute in bean definition in applicationContext.xml file.

Similarly, implementing the [org.springframework.beans.factory.DisposableBean](http://static.springsource.org/spring/docs/1.2.9/api/org/springframework/beans/factory/DisposableBean.html" \o "DisposableBean) interface allows a bean to get a callback when the container containing it is destroyed.

The DisposableBean interface specifies a single method:

|  |
| --- |
| DisposableBean.java |
| void destroy() throws Exception;    A sample bean implementing above interfaces would look like this:      package com.howtodoinjava.task;    import org.springframework.beans.factory.DisposableBean;  import org.springframework.beans.factory.InitializingBean;    public class DemoBean implements InitializingBean, DisposableBean  {      //Other bean attributes and methods        @Override      public void afterPropertiesSet() throws Exception      {          //Bean initialization code      }        @Override      public void destroy() throws Exception      {          //Bean destruction code      }  } |

#### 2.2. \*Aware interfaces for specific behavior

Spring offers a range of \*Aware interfaces that allow beans to indicate to the container that they require a certain infrastructure dependency. Each interface will require you to implement a method to inject the dependency in bean.

These interfaces can be summarized as :

|  |  |  |
| --- | --- | --- |
| **AWARE INTERFACE** | **METHOD TO OVERRIDE** | **PURPOSE** |
| ApplicationContextAware | void setApplicationContext(ApplicationContext applicationContext) throws BeansException; | Interface to be implemented by any object that wishes to be notified of the ApplicationContext that it runs in. |
| ApplicationEventPublisherAware | void setApplicationEventPublisher(ApplicationEventPublisher applicationEventPublisher); | Set the ApplicationEventPublisher that this object runs in. |
| BeanClassLoaderAware | void setBeanClassLoader(ClassLoader classLoader); | Callback that supplies the bean class loader to a bean instance. |
| BeanFactoryAware | void setBeanFactory (BeanFactory beanFactory) throws BeansException; | Callback that supplies the owning factory to a bean instance. |
| BeanNameAware | void setBeanName(String name); | Set the name of the bean in the bean factory that created this bean. |
| BootstrapContextAware | void setBootstrapContext(BootstrapContext bootstrapContext); | Set the BootstrapContext that this object runs in. |
| LoadTimeWeaverAware | void setLoadTimeWeaver(LoadTimeWeaver loadTimeWeaver); | Set the LoadTimeWeaver of this object’s containing ApplicationContext. |
| MessageSourceAware | void setMessageSource(MessageSource messageSource); | Set the MessageSource that this object runs in. |
| NotificationPublisherAware | void setNotificationPublisher(NotificationPublisher notificationPublisher); | Set the NotificationPublisher instance for the current managed resource instance. |
| PortletConfigAware | void setPortletConfig (PortletConfig portletConfig); | Set the PortletConfig this object runs in. |
| PortletContextAware | void setPortletContext(PortletContext portletContext); | Set the PortletContext that this object runs in. |
| ResourceLoaderAware | void setResourceLoader(ResourceLoader resourceLoader); | Set the ResourceLoader that this object runs in. |
| ServletConfigAware | void setServletConfig (ServletConfig servletConfig); | Set the ServletConfig that this object runs in. |
| ServletContextAware | void setServletContext(ServletContext servletContext); | Set the ServletContext that this object runs in. |

Java program to show usage of aware interfaces to control string bean life cycle.

|  |
| --- |
| DemoBean.java |
| package com.howtodoinjava.task;    import org.springframework.beans.BeansException;  import org.springframework.beans.factory.BeanClassLoaderAware;  import org.springframework.beans.factory.BeanFactory;  import org.springframework.beans.factory.BeanFactoryAware;  import org.springframework.beans.factory.BeanNameAware;  import org.springframework.context.ApplicationContext;  import org.springframework.context.ApplicationContextAware;  import org.springframework.context.ApplicationEventPublisher;  import org.springframework.context.ApplicationEventPublisherAware;  import org.springframework.context.MessageSource;  import org.springframework.context.MessageSourceAware;  import org.springframework.context.ResourceLoaderAware;  import org.springframework.context.weaving.LoadTimeWeaverAware;  import org.springframework.core.io.ResourceLoader;  import org.springframework.instrument.classloading.LoadTimeWeaver;  import org.springframework.jmx.export.notification.NotificationPublisher;  import org.springframework.jmx.export.notification.NotificationPublisherAware;    public class DemoBean implements ApplicationContextAware,          ApplicationEventPublisherAware, BeanClassLoaderAware, BeanFactoryAware,          BeanNameAware, LoadTimeWeaverAware, MessageSourceAware,          NotificationPublisherAware, ResourceLoaderAware  {      @Override      public void setResourceLoader(ResourceLoader arg0) {          // TODO Auto-generated method stub      }        @Override      public void setNotificationPublisher(NotificationPublisher arg0) {          // TODO Auto-generated method stub        }        @Override      public void setMessageSource(MessageSource arg0) {          // TODO Auto-generated method stub      }        @Override      public void setLoadTimeWeaver(LoadTimeWeaver arg0) {          // TODO Auto-generated method stub      }        @Override      public void setBeanName(String arg0) {          // TODO Auto-generated method stub      }        @Override      public void setBeanFactory(BeanFactory arg0) throws BeansException {          // TODO Auto-generated method stub      }        @Override      public void setBeanClassLoader(ClassLoader arg0) {          // TODO Auto-generated method stub      }        @Override      public void setApplicationEventPublisher(ApplicationEventPublisher arg0) {          // TODO Auto-generated method stub      }        @Override      public void setApplicationContext(ApplicationContext arg0)              throws BeansException {          // TODO Auto-generated method stub      }  } |

#### 2.3. Custom init() and destroy() methods in bean configuration XML file

The default init and destroy methods in bean configuration file can be defined in two ways:

* **Bean local definition** applicable to a single bean
* **Global definition** applicable to all beans defined in beans context

##### 2.3.1. Bean local definition

Local definition is given as below.

|  |
| --- |
| beans.xml |
| <beans>        <bean id="demoBean" class="com.howtodoinjava.task.DemoBean"                      init-method="customInit"                      destroy-method="customDestroy"></bean>    </beans> |

##### 2.3.2. Global definition

Where as global definition is given as below. These methods will be invoked for all bean definitions given under <beans> tag. They are useful when you have a pattern of defining common method names such as init() and destroy() for all your beans consistently. This feature helps you in not mentioning the init and destroy method names for all beans independently.

|  |
| --- |
| <beans default-init-method="customInit" default-destroy-method="customDestroy">            <bean id="demoBean" class="com.howtodoinjava.task.DemoBean"></bean>    </beans> |

Java program to show methods configured in bean XML configuration file.

|  |
| --- |
| DemoBean.java |
| package com.howtodoinjava.task;    public class DemoBean  {      public void customInit()      {          System.out.println("Method customInit() invoked...");      }        public void customDestroy()      {          System.out.println("Method customDestroy() invoked...");      }  } |

#### 2.4. Spring Bean Life Cycle – @PostConstruct and @PreDestroy annotations

Spring 2.5 onwards, you can use annotations also for specifying life cycle methods using @PostConstruct and @PreDestroy annotations.

* @PostConstruct annotated method will be invoked after the bean has been constructed using default constructor and just before it’s instance is returned to requesting object.
* @PreDestroy annotated method is called just before the bean is about be destroyed inside bean container.

Java program to show usage of **annotation configuration** to control using annotations.

|  |
| --- |
| package com.howtodoinjava.task;    import javax.annotation.PostConstruct;  import javax.annotation.PreDestroy;    public class DemoBean  {      @PostConstruct      public void customInit()      {          System.out.println("Method customInit() invoked...");      }        @PreDestroy      public void customDestroy()      {          System.out.println("Method customDestroy() invoked...");      }  } |

**Difference between BeanFactory and ApplicationContext**

1. Instantiate the bean when getBean method is called but in applicationcontext when container is started.
2. By default its support lazy loading but in applicationcontext by default it will support eager loading.
3. Doesnot provide support for internalization but AC will provide the support.
4. BeanFactory will support scope only singleton and prototype but AC will support Singleton,ProtoType,Session,Global-Session and request
5. BeanFactory will support only one configuration file but in AC multiple configuration file can support.
6. BeanFacory doesn’t support annotation based dependency injection but AC will support.
7. **What are different ways to configure a class as Spring Bean?**

There are three different ways to configure Spring Bean.

* + **XML Configuration**: This is the most popular configuration and we can use bean element in context file to configure a Spring Bean. For example:

<bean name="myBean" class="com.journaldev.spring.beans.MyBean"></bean>

* + **Java Based Configuration**: If you are using only annotations, you can configure a Spring bean using @Bean annotation. This annotation is used with @Configuration classes to configure a spring bean. Sample configuration is:
  + @Configuration
  + @ComponentScan(value="com.journaldev.spring.main")
  + public class MyConfiguration {
  + @Bean
  + public MyService getService(){
  + return new MyService();
  + }

}

To get this bean from spring context, we need to use following code snippet:

AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext(

MyConfiguration.class);

MyService service = ctx.getBean(MyService.class);

* + **Annotation Based Configuration**: We can also use @Component, @Service, @Repository and @Controller annotations with classes to configure them to be as spring bean. For these, we would need to provide base package location to scan for these classes. For example:

<context:component-scan base-package="com.journaldev.spring" />

1. **What are different scopes of Spring Bean?**

There are five scopes defined for Spring Beans.

* + **singleton**: Only one instance of the bean will be created for each container. This is the default scope for the spring beans. While using this scope, make sure spring bean doesn’t have shared instance variables otherwise it might lead to data inconsistency issues because it’s not thread-safe.
  + **prototype**: A new instance will be created every time the bean is requested.
  + **request**: This is same as prototype scope; however it’s meant to be used for web applications. A new instance of the bean will be created for each HTTP request.
  + **session**: A new bean will be created for each HTTP session by the container.
  + **global-session**: This is used to create global session beans for Portlet applications.

Spring Framework is extendable and we can create our own scopes too, however most of the times we are good with the scopes provided by the framework.

To set spring bean scopes we can use “scope” attribute in bean element or @Scope annotation for annotation based configurations.

1. **What is Spring Bean life cycle?**

Spring Beans are initialized by Spring Container and all the dependencies are also injected. When context is destroyed, it also destroys all the initialized beans. This works well in most of the cases but sometimes we want to initialize other resources or do some validation before making our beans ready to use. Spring framework provides support for post-initialization and pre-destroy methods in spring beans.

We can do this by two ways – by implementing InitializingBean and DisposableBean interfaces or using **init-method** and **destroy-method** attribute in spring bean configurations. For more details, please read [Spring Bean Life Cycle Methods](http://www.journaldev.com/2637/spring-bean-life-cycle).

1. **How to get ServletContext and ServletConfig object in a Spring Bean?**

There are two ways to get Container specific objects in the spring bean.

* + Implementing Spring \*Aware interfaces, for these ServletContextAware and ServletConfigAware interfaces, for complete example of these aware interfaces, please read [Spring Aware Interfaces](http://www.journaldev.com/2637/spring-bean-life-cycle)
  + Using @Autowired annotation with bean variable of type ServletContext and ServletConfig. They will work only in servlet container specific environment only though.
  + @Autowired

ServletContext servletContext;

1. **What is Bean wiring and @Autowired annotation?**

The process of injection spring bean dependencies while initializing it called Spring Bean Wiring.

Usually it’s best practice to do the explicit wiring of all the bean dependencies, but spring framework also supports autowiring. We can use @Autowired annotation with fields or methods for **autowiring byType**. For this annotation to work, we also need to enable annotation based configuration in spring bean configuration file. This can be done by **context:annotation-config** element.

For more details about @Autowired annotation, please read [Spring Autowire Example](http://www.journaldev.com/2623/spring-autowired-annotation).

1. **What are different types of Spring Bean autowiring?**

There are four types of autowiring in Spring framework.

* + **autowire byName**
  + **autowire byType**
  + **autowire by constructor**
  + autowiring by **@Autowired** and **@Qualifier** annotations

Prior to Spring 3.1, **autowire by autodetect** was also supported that was similar to autowire by constructor or byType. For more details about these options, please read [Spring Bean Autowiring](http://www.journaldev.com/2623/spring-autowired-annotation).

1. **Does Spring Bean provide thread safety?**

The default scope of Spring bean is singleton, so there will be only one instance per context. That means that all the having a class level variable that any thread can update will lead to inconsistent data. Hence in default mode spring beans are not thread-safe.

However, we can change spring bean scope to request, prototype or session to achieve thread-safety at the cost of performance. It’s a design decision and based on the project requirements.

1. **What is a Controller in Spring MVC?**

Just like MVC design pattern, Controller is the class that takes care of all the client requests and send them to the configured resources to handle it. In Spring MVC, org.springframework.web.servlet.DispatcherServlet is the front controller class that initializes the context based on the spring beans configurations.

A Controller class is responsible to handle different kind of client requests based on the request mappings. We can create a controller class by using @Controller annotation. Usually it’s used with @RequestMapping annotation to define handler methods for specific URI mapping.

1. **What’s the difference between @Component, @Controller, @Repository & @Service annotations in Spring?**

**@Component** is used to indicate that a class is a component. These classes are used for auto detection and configured as bean, when annotation based configurations are used.

**@Controller** is a specific type of component, used in MVC applications and mostly used with RequestMapping annotation.

**@Repository** annotation is used to indicate that a component is used as repository and a mechanism to store/retrieve/search data. We can apply this annotation with DAO pattern implementation classes.

**@Service** is used to indicate that a class is a Service. Usually the business facade classes that provide some services are annotated with this.

We can use any of the above annotations for a class for auto-detection but different types are provided so that you can easily distinguish the purpose of the annotated classes.

1. **What is DispatcherServlet and ContextLoaderListener?**

DispatcherServlet is the front controller in the Spring MVC application and it loads the spring bean configuration file and initialize all the beans that are configured. If annotations are enabled, it also scans the packages and configure any bean annotated with @Component, @Controller, @Repository or @Service annotations.

ContextLoaderListener is the listener to start up and shut down Spring’s root WebApplicationContext. It’s important functions are to tie up the lifecycle of ApplicationContext to the lifecycle of the ServletContext and to automate the creation of ApplicationContext. We can use it to define shared beans that can be used across different spring contexts.

1. **What is ViewResolver in Spring?**

ViewResolver implementations are used to resolve the view pages by name. Usually we configure it in the spring bean configuration file. For example:

<!-- Resolves views selected for rendering by @Controllers to .jsp resources in the /WEB-INF/views directory -->

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

<beans:property name="prefix" value="/WEB-INF/views/" />

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

</beans:bean>

InternalResourceViewResolver is one of the implementation of ViewResolver interface and we are providing the view pages directory and suffix location through the bean properties. So if a controller handler method returns “home”, view resolver will use view page located at */WEB-INF/views/home.jsp*.

1. **What is a MultipartResolver and when its used?**

MultipartResolver interface is used for uploading files – CommonsMultipartResolver and StandardServletMultipartResolver are two implementations provided by spring framework for file uploading. By default there are no multipart resolvers configured but to use them for uploading files, all we need to define a bean named “multipartResolver” with type as MultipartResolver in spring bean configurations.

Once configured, any multipart request will be resolved by the configured MultipartResolver and pass on a wrapped HttpServletRequest. Then it’s used in the controller class to get the file and process it. For a complete example, please read [Spring MVC File Upload Example](http://www.journaldev.com/2573/spring-mvc-file-upload-example-single-multiple-files).

1. **How to handle exceptions in Spring MVC Framework?**

Spring MVC Framework provides following ways to help us achieving robust exception handling.

* + **Controller Based** – We can define exception handler methods in our controller classes. All we need is to annotate these methods with @ExceptionHandler annotation.
  + **Global Exception Handler** – Exception Handling is a cross-cutting concern and Spring provides @ControllerAdvice annotation that we can use with any class to define our global exception handler.
  + **HandlerExceptionResolver implementation** – For generic exceptions, most of the times we serve static pages. Spring Framework provides HandlerExceptionResolver interface that we can implement to create global exception handler. The reason behind this additional way to define global exception handler is that Spring framework also provides default implementation classes that we can define in our spring bean configuration file to get spring framework exception handling benefits.

For a complete example, please read [Spring Exception Handling Example](http://www.journaldev.com/2651/spring-mvc-exception-handling-controlleradvice-exceptionhandler-handlerexceptionresolver).

1. **How to create ApplicationContext in a Java Program?**

There are following ways to create spring context in a standalone java program.

* + **AnnotationConfigApplicationContext**: If we are using Spring in standalone java applications and using annotations for Configuration, then we can use this to initialize the container and get the bean objects.
  + **ClassPathXmlApplicationContext**: If we have spring bean configuration xml file in standalone application, then we can use this class to load the file and get the container object.
  + **FileSystemXmlApplicationContext**: This is similar to ClassPathXmlApplicationContext except that the xml configuration file can be loaded from anywhere in the file system.

1. **Can we have multiple Spring configuration files?**

For Spring MVC applications, we can define multiple spring context configuration files through contextConfigLocation. This location string can consist of multiple locations separated by any number of commas and spaces. For example;

<servlet>

<servlet-name>appServlet</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/spring/appServlet/servlet-context.xml,/WEB-INF/spring/appServlet/servlet-jdbc.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

We can also define multiple root level spring configurations and load it through context-param. For example;

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/spring/root-context.xml /WEB-INF/spring/root-security.xml</param-value>

</context-param>

Another option is to use import element in the context configuration file to import other configurations, for example:

<beans:import resource="spring-jdbc.xml"/>

1. **What is ContextLoaderListener?**

ContextLoaderListener is the listener class used to load root context and define spring bean configurations that will be visible to all other contexts. It’s configured in web.xml file as:

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/spring/root-context.xml</param-value>

</context-param>

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

1. **What are the minimum configurations needed to create Spring MVC application?**

For creating a simple Spring MVC application, we would need to do following tasks.

* + Add spring-context and spring-webmvc dependencies in the project.
  + Configure DispatcherServlet in the web.xml file to handle requests through spring container.
  + Spring bean configuration file to define beans, if using annotations then it has to be configured here. Also we need to configure view resolver for view pages.
  + Controller class with request mappings defined to handle the client requests.

Above steps should be enough to create a simple Spring MVC Hello World application.

1. **How would you relate Spring MVC Framework to MVC architecture?**

As the name suggests Spring MVC is built on top of **Model-View-Controller** architecture. DispatcherServlet is the Front Controller in the Spring MVC application that takes care of all the incoming requests and delegate it to different controller handler methods.

Model can be any Java Bean in the Spring Framework, just like any other MVC framework Spring provides automatic binding of form data to java beans. We can set model beans as attributes to be used in the view pages.

View Pages can be JSP, static HTMLs etc. and view resolvers are responsible for finding the correct view page. Once the view page is identified, control is given back to the DispatcherServlet controller. DispatcherServlet is responsible for rendering the view and returning the final response to the client.

1. **How to achieve localization in Spring MVC applications?**

Spring provides excellent support for localization or i18n through resource bundles. Basis steps needed to make our application localized are:

* + Creating message resource bundles for different locales, such as messages\_en.properties, messages\_fr.properties etc.
  + Defining messageSource bean in the spring bean configuration file of type ResourceBundleMessageSource or ReloadableResourceBundleMessageSource.
  + For change of locale support, define localeResolver bean of type CookieLocaleResolver and configure LocaleChangeInterceptor interceptor. Example configuration can be like below:
  + <beans:bean id="messageSource"
  + class="org.springframework.context.support.ReloadableResourceBundleMessageSource">
  + <beans:property name="basename" value="classpath:messages" />
  + <beans:property name="defaultEncoding" value="UTF-8" />
  + </beans:bean>
  + <beans:bean id="localeResolver"
  + class="org.springframework.web.servlet.i18n.CookieLocaleResolver">
  + <beans:property name="defaultLocale" value="en" />
  + <beans:property name="cookieName" value="myAppLocaleCookie"></beans:property>
  + <beans:property name="cookieMaxAge" value="3600"></beans:property>
  + </beans:bean>
  + <interceptors>
  + <beans:bean class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">
  + <beans:property name="paramName" value="locale" />
  + </beans:bean>

</interceptors>

* + Use spring:message element in the view pages with key names, DispatcherServlet picks the corresponding value and renders the page in corresponding locale and return as response.

For a complete example, please read [Spring Localization Example](http://www.journaldev.com/2610/spring-mvc-internationalization-i18n-and-localization-l10n-example).

1. **How can we use Spring to create Restful Web Service returning JSON response?**

We can use Spring Framework to create Restful web services that returns JSON data. Spring provides integration with [Jackson JSON](http://www.journaldev.com/2324/jackson-json-java-parser-api-example-tutorial) API that we can use to send JSON response in restful web service.

We would need to do following steps to configure our Spring MVC application to send JSON response:

* + Adding Jackson JSON dependencies, if you are using Maven it can be done with following code:
  + <!-- Jackson -->
  + <dependency>
  + <groupId>com.fasterxml.jackson.core</groupId>
  + <artifactId>jackson-databind</artifactId>
  + <version>${jackson.databind-version}</version>

</dependency>

* + Configure RequestMappingHandlerAdapter bean in the spring bean configuration file and set the messageConverters property to MappingJackson2HttpMessageConverter bean. Sample configuration will be:
  + <!-- Configure to plugin JSON as request and response in method handler -->
  + <beans:bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter">
  + <beans:property name="messageConverters">
  + <beans:list>
  + <beans:ref bean="jsonMessageConverter"/>
  + </beans:list>
  + </beans:property>
  + </beans:bean>
  + <!-- Configure bean to convert JSON to POJO and vice versa -->
  + <beans:bean id="jsonMessageConverter" class="org.springframework.http.converter.json.MappingJackson2HttpMessageConverter">

</beans:bean>

* + In the controller handler methods, return the Object as response using @ResponseBodyannotation. Sample code:
  + @RequestMapping(value = EmpRestURIConstants.GET\_EMP, method = RequestMethod.GET)
  + public @ResponseBody Employee getEmployee(@PathVariable("id") int empId) {
  + logger.info("Start getEmployee. ID="+empId);
  + return empData.get(empId);

}

* + You can invoke the rest service through any API, but if you want to use Spring then we can easily do it using RestTemplate class.

For a complete example, please read [Spring Restful Webservice Example](http://www.journaldev.com/2552/spring-rest-example-tutorial-spring-restful-web-services).

1. **What are some of the important Spring annotations you have used?**

Some of the Spring annotations that I have used in my project are:

* + **@Controller** – for controller classes in Spring MVC project.
  + **@RequestMapping** – for configuring URI mapping in controller handler methods. This is a very important annotation, so you should go through [Spring MVC RequestMapping Annotation Examples](http://www.journaldev.com/3358/spring-requestmapping-requestparam-pathvariable-example)
  + **@ResponseBody** – for sending Object as response, usually for sending XML or JSON data as response.
  + **@PathVariable** – for mapping dynamic values from the URI to handler method arguments.
  + **@Autowired** – for autowiring dependencies in spring beans.
  + **@Qualifier** – with @Autowired annotation to avoid confusion when multiple instances of bean type is present.
  + **@Service** – for service classes.
  + **@Scope** – for configuring scope of the spring bean.
  + **@Configuration**, **@ComponentScan** and **@Bean** – for java based configurations.
  + AspectJ annotations for configuring aspects and advices, **@Aspect**, **@Before**, **@After**, **@Around**, **@Pointcut** etc.

1. **Can we send an Object as the response of Controller handler method?**

Yes we can, using **@ResponseBody** annotation. This is how we send JSON or XML based response in restful web services.

1. **How to upload file in Spring MVC Application?**

Spring provides built-in support for uploading files through **MultipartResolver** interface implementations. It’s very easy to use and requires only configuration changes to get it working. Obviously we would need to write controller handler method to handle the incoming file and process it. For a complete example, please refer [Spring File Upload Example](http://www.journaldev.com/2573/spring-mvc-file-upload-example-single-multiple-files).

1. **How to validate form data in Spring Web MVC Framework?**

Spring supports JSR-303 annotation based validations as well as provide Validator interface that we can implement to create our own custom validator. For using JSR-303 based validation, we need to annotate bean variables with the required validations.

For custom validator implementation, we need to configure it in the controller class. For a complete example, please read [Spring MVC Form Validation Example](http://www.journaldev.com/2668/spring-validation-example-mvc-validator).

1. **What is Spring MVC Interceptor and how to use it?**

Spring MVC Interceptors are like Servlet Filters and allow us to intercept client request and process it. We can intercept client request at three places – **preHandle**, **postHandle** and **afterCompletion**.

We can create spring interceptor by implementing HandlerInterceptor interface or by extending abstract class **HandlerInterceptorAdapter**.

We need to configure interceptors in the spring bean configuration file. We can define an interceptor to intercept all the client requests or we can configure it for specific URI mapping too. For a detailed example, please refer [Spring MVC Interceptor Example](http://www.journaldev.com/2676/spring-mvc-interceptor-example-handlerinterceptor-handlerinterceptoradapter).

1. **What is Spring JdbcTemplate class and how to use it?**

Spring Framework provides excellent integration with JDBC API and provides JdbcTemplate utility class that we can use to avoid bolier-plate code from our database operations logic such as Opening/Closing Connection, ResultSet, PreparedStatement etc.

For JdbcTemplate example, please refer [Spring JDBC Example](http://www.journaldev.com/2593/spring-jdbc-example).

1. **How to use Tomcat JNDI DataSource in Spring Web Application?**

For using servlet container configured JNDI DataSource, we need to configure it in the spring bean configuration file and then inject it to spring beans as dependencies. Then we can use it with JdbcTemplate to perform database operations.

Sample configuration would be:

<beans:bean id="dbDataSource" class="org.springframework.jndi.JndiObjectFactoryBean">

<beans:property name="jndiName" value="java:comp/env/jdbc/MyLocalDB"/>

</beans:bean>

For complete example, please refer [Spring Tomcat JNDI Example](http://www.journaldev.com/2597/spring-datasource-jndi-with-tomcat-example).

1. **How would you achieve Transaction Management in Spring?**

Spring framework provides transaction management support through Declarative Transaction Management as well as programmatic transaction management. Declarative transaction management is most widely used because it’s easy to use and works in most of the cases.

We use annotate a method with @Transactional annotation for Declarative transaction management. We need to configure transaction manager for the DataSource in the spring bean configuration file.

<bean id="transactionManager"

class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

<property name="dataSource" ref="dataSource" />

</bean>

1. **What is Spring DAO?**

Spring DAO support is provided to work with data access technologies like JDBC, Hibernate in a consistent and easy way. For example we have JdbcDaoSupport, HibernateDaoSupport, JdoDaoSupport and JpaDaoSupport for respective technologies.

Spring DAO also provides consistency in exception hierarchy and we don’t need to catch specific exceptions.

1. **How to integrate Spring and Hibernate Frameworks?**

We can use Spring ORM module to integrate Spring and Hibernate frameworks, if you are using Hibernate 3+ where SessionFactory provides current session, then you should avoid using HibernateTemplate or HibernateDaoSupport classes and better to use DAO pattern with dependency injection for the integration.

Also Spring ORM provides support for using Spring declarative transaction management, so you should utilize that rather than going for hibernate boiler-plate code for transaction management.

For better understanding you should go through following tutorials:

* + [Spring Hibernate Integration Example](http://www.journaldev.com/3524/spring-hibernate-integration-example-tutorial)
  + [Spring MVC Hibernate Integration Example](http://www.journaldev.com/3531/spring-mvc-hibernate-mysql-integration-crud-example-tutorial)

1. **What is Spring Security?**

Spring security framework focuses on providing both authentication and authorization in java applications. It also takes care of most of the common security vulnerabilities such as CSRF attack.

It’s very beneficial and easy to use Spring security in web applications, through the use of annotations such as @EnableWebSecurity. You should go through following posts to learn how to use Spring Security framework.

* + [Spring Security in Servlet Web Application](http://www.journaldev.com/2715/spring-security-example-tutorial)
  + [Spring MVC and Spring Security Integration Example](http://www.journaldev.com/2736/spring-security-example-userdetailsservice)

1. **How to inject a java.util.Properties into a Spring Bean?**

We need to define propertyConfigurer bean that will load the properties from the given property file. Then we can use Spring EL support to inject properties into other bean dependencies. For example;

<bean id="propertyConfigurer"

class="org.springframework.context.support.PropertySourcesPlaceholderConfigurer">

<property name="location" value="/WEB-INF/application.properties" />

</bean>

<bean class="com.journaldev.spring.EmployeeDaoImpl">

<property name="maxReadResults" value="${results.read.max}"/>

</bean>

If you are using annotation to configure the spring bean, then you can inject property like below.

@Value("${maxReadResults}")

private int maxReadResults;

1. **Name some of the design patterns used in Spring Framework?**

Spring Framework is using a lot of design patterns, some of the common ones are:

* + Singleton Pattern: Creating beans with default scope.
  + [Factory Pattern](http://www.journaldev.com/1392/factory-design-pattern-in-java): Bean Factory classes
  + [Prototype Pattern](http://www.journaldev.com/1440/prototype-design-pattern-in-java): Bean scopes
  + [Adapter Pattern](http://www.journaldev.com/1487/adapter-design-pattern-java): Spring Web and Spring MVC
  + [Proxy Pattern](http://www.journaldev.com/1572/proxy-design-pattern): Spring Aspect Oriented Programming support
  + [Template Method Pattern](http://www.journaldev.com/1763/template-method-design-pattern-in-java): JdbcTemplate, HibernateTemplate etc
  + Front Controller: Spring MVC DispatcherServlet
  + Data Access Object: Spring DAO support
  + Dependency Injection and Aspect Oriented Programming

1. **What are some of the best practices for Spring Framework?**

Some of the best practices for Spring Framework are:

* + Avoid version numbers in schema reference, to make sure we have the latest configs.
  + Divide spring bean configurations based on their concerns such as spring-jdbc.xml, spring-security.xml.
  + For spring beans that are used in multiple contexts in Spring MVC, create them in the root context and initialize with listener.
  + Configure bean dependencies as much as possible, try to avoid autowiring as much as possible.
  + For application level properties, best approach is to create a property file and read it in the spring bean configuration file.
  + For smaller applications, annotations are useful but for larger applications annotations can become a pain. If we have all the configuration in xml files, maintaining it will be easier.
  + Use correct annotations for components for understanding the purpose easily. For services use @Service and for DAO beans use @Repository.
  + Spring framework has a lot of modules, use what you need. Remove all the extra dependencies that gets usually added when you create projects through Spring Tool Suite templates.
  + If you are using Aspects, make sure to keep the join pint as narrow as possible to avoid advice on unwanted methods. Consider custom annotations that are easier to use and avoid any issues.
  + Use dependency injection when there is actual benefit, just for the sake of loose-coupling don’t use it because it’s harder to maintain.

That’s all for Spring Framework interview questions. I hope these questions will help you in coming Java EE interview. I will keep on adding more questions to the list as soon as I found them. If you know some more questions that should be part of the list, make sure to add a comment for it and I will include it.

## What is Spring MVC framework?

**The Spring web MVC framework provides**[**model-view-controller**](http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller)**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 model, view and controller parts of application. Spring framework provides lots of advantages over other MVC frameworks e.g.

1. Clear separation of roles – controller, validator, command object, form object, model object, DispatcherServlet, handler mapping, view resolver, etc. Each role can be fulfilled by a specialized object.
2. Powerful and straightforward configuration of both framework and application classes as JavaBeans.
3. Reusable business code – no need for duplication. You can use existing business objects as command or form objects instead of mirroring them in order to extend a particular framework base class.
4. Customizable binding and validation
5. Customizable handler mapping and view resolution
6. Customizable locale and theme resolution
7. A JSP form tag library, introduced in Spring 2.0, that makes writing forms in JSP pages much easier. etc.

## What is DispatcherServlet and ContextLoaderListener?

Spring’s web MVC framework is, like many other web MVC frameworks, request-driven, designed around a central Servlet that handles all the HTTP requests and responses. Spring’s DispatcherServlet however, does more than just that. It is completely integrated with the Spring IoC container so it allows you to use every feature that Spring has.

After receiving an HTTP request, DispatcherServlet consults the HandlerMapping (configuration files) to call the appropriate Controller. The Controller takes the request and calls the appropriate service methods and set model data and then 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.

|  |
| --- |
| <web-app>    <display-name>Archetype Created Web Application</display-name>      <servlet>          <servlet-name>spring</servlet-name>              <servlet-class>                  org.springframework.web.servlet.DispatcherServlet              </servlet-class>          <load-on-startup>1</load-on-startup>      </servlet>        <servlet-mapping>          <servlet-name>spring</servlet-name>          <url-pattern>/</url-pattern>      </servlet-mapping>    </web-app> |

By default, DispatcherServlet loads its configuration file using <servlet\_name>-servlet.xml. E.g. with above web.xml file, DispatcherServlet will try to find spring-servlet.xml file in classpath.

ContextLoaderListener reads the spring configuration file (with value given against “**contextConfigLocation**” in web.xml), parse it and loads the beans defined in that config file. e.g.

|  |
| --- |
| <servlet>      <servlet-name>spring</servlet-name>      <servlet-class>          org.springframework.web.servlet.DispatcherServlet      </servlet-class>        <init-param>          <param-name>contextConfigLocation</param-name>          <param-value>/WEB-INF/applicationContext.xml</param-value>      </init-param>        <load-on-startup>1</load-on-startup>  </servlet> |

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

A front controller is defined as “a controller which handles all requests for a Web Application.” **DispatcherServlet (actually a servlet) is the front controller in Spring MVC that intercepts every request and then dispatches/forwards requests to an appropriate controller.**

When a web request is sent to a Spring MVC application, dispatcher servlet first receives the request. Then it organizes the different components configured in Spring’s web application context (e.g. actual request handler controller and view resolvers) or annotations present in the controller itself, all needed to handle the request.

## How to use Java based configuration?

To configure java based MVC application, first add required dependencies.

|  |
| --- |
| <!-- Spring MVC support -->    <dependency>      <groupId>org.springframework</groupId>      <artifactId>spring-webmvc</artifactId>      <version>4.1.4.RELEASE</version>  </dependency>    <dependency>      <groupId>org.springframework</groupId>      <artifactId>spring-web</artifactId>      <version>4.1.4.RELEASE</version>  </dependency>    <!-- Tag libs support for view layer -->    <dependency>      <groupId>javax.servlet</groupId>      <artifactId>jstl</artifactId>      <version>1.2</version>      <scope>runtime</scope>  </dependency>    <dependency>      <groupId>taglibs</groupId>      <artifactId>standard</artifactId>      <version>1.1.2</version>      <scope>runtime</scope>  </dependency> |

Now add DispatcherServlet entry in web.xml file so that all incoming requests come though DispatcherServlet only.

|  |
| --- |
| <servlet>      <servlet-name>spring</servlet-name>          <servlet-class>              org.springframework.web.servlet.DispatcherServlet          </servlet-class>      <load-on-startup>1</load-on-startup>  </servlet>    <servlet-mapping>      <servlet-name>spring</servlet-name>      <url-pattern>/</url-pattern>  </servlet-mapping> |

Now add below entries in spring configuration file.

|  |
| --- |
| <beans>      <!-- Scan all classes in this path for spring specific annotations -->      <context:component-scan base-package="com.howtodoinjava.demo" />        <bean class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping" />      <bean class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter" />        <!-- Vierw resolver configuration -->      <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">          <property name="prefix" value="/WEB-INF/views/" />          <property name="suffix" value=".jsp" />      </bean>    </beans> |

Add controller code.

|  |
| --- |
| @Controller  @RequestMapping("/employee-module")  public class EmployeeController  {      @Autowired      EmployeeManager manager;        @RequestMapping(value = "/getAllEmployees", method = RequestMethod.GET)      public String getAllEmployees(Model model)      {          model.addAttribute("employees", manager.getAllEmployees());          return "employeesListDisplay";      }  } |

Additionally you should add manager and dao layer classes as well. Finally you add the jsp file to display the view.

I will suggest to read above linked tutorial for complete understanding.

**Read More :**[**Spring MVC Hello World Example**](http://howtodoinjava.com/spring/spring-mvc/spring-mvc-hello-world-example/)

## How can we use Spring to create Restful Web Service returning JSON response?

For adding JSON support to your spring application, you will need to **add Jackson dependency** in first step.

|  |
| --- |
| <!-- Jackson JSON Processor -->  <dependency>      <groupId>com.fasterxml.jackson.core</groupId>      <artifactId>jackson-databind</artifactId>      <version>2.4.1</version>  </dependency> |

Now you are ready to return JSON response from your MVC controller. All you have to do is return JAXB annotated object from method and use @ResponseBody annotation on this return type.

|  |
| --- |
| @Controller  public class EmployeeRESTController  {      @RequestMapping(value = "/employees")      public @ResponseBody EmployeeListVO getAllEmployees()      {          EmployeeListVO employees = new EmployeeListVO();          //Add employees          return employees;      }  } |

Alternatively, you can use @RestController annotation in place of @Controller annotation. This will remove the need to using @ResponseBody.

**@RestController = @Controller + @ResponseBody**

So you can write the above controller as below.

|  |
| --- |
| @RestController  public class EmployeeRESTController  {      @RequestMapping(value = "/employees")      public EmployeeListVO getAllEmployees()      {          EmployeeListVO employees = new EmployeeListVO();          //Add employees          return employees;      }  } |

**Read More :**[**Spring REST Hello World JSON Example**](http://howtodoinjava.com/spring/spring-restful/spring-rest-hello-world-json-example/)

## Can we have multiple Spring configuration files?

YES. **You can have multiple spring context files**. There are two ways to make spring read and configure them.

a) Specify all files in web.xml file using **contextConfigLocation** init parameter.

|  |
| --- |
| <servlet>          <servlet-name>spring</servlet-name>          <servlet-class>              org.springframework.web.servlet.DispatcherServlet          </servlet-class>          <init-param>              <param-name>contextConfigLocation</param-name>              <param-value>                  WEB-INF/spring-dao-hibernate.xml,                  WEB-INF/spring-services.xml,                  WEB-INF/spring-security.xml              </param-value>          </init-param>          <load-on-startup>1</load-on-startup>      </servlet>        <servlet-mapping>          <servlet-name>spring</servlet-name>          <url-pattern>/</url-pattern>      </servlet-mapping> |

b) OR, you can **import them into existing configuration file** you have already configured.

|  |
| --- |
| <beans>      <import resource="spring-dao-hibernate.xml"/>      <import resource="spring-services.xml"/>      <import resource="spring-security.xml"/>        ... //Other configuration stuff    </beans> |

## Difference between <context:annotation-config> vs <context:component-scan>?

1) First big difference between both tags is that <context:annotation-config> is **used to activate applied annotations in already registered beans in application context**. Note that it simply does not matter whether bean was registered by which mechanism e.g. using <context:component-scan> or it was defined in application-context.xml file itself.

2) Second difference is driven from first difference itself. It **registers the beans defined in config file into context + it also scans the annotations inside beans and activate them**. So <context:component-scan> does what <context:annotation-config> does, but additionally it scan the packages and register the beans in application context.

**<context:annotation-config> = Scanning and activating annotations in “already registered beans”.**

**<context:component-scan> = Bean Registration + Scanning and activating annotations**

**Read More :**[**Difference between annotation-config and component-scan**](http://howtodoinjava.com/spring/spring-mvc/spring-mvc-difference-between-contextannotation-config-vs-contextcomponent-scan/)

## Difference between @Component, @Controller, @Repository & @Service annotations?

1) The @Component annotation marks a java class as a bean so the component-scanning mechanism of spring can pick it up and pull it into the application context. To use this annotation, apply it over class as below:

|  |
| --- |
| @Component  public class EmployeeDAOImpl implements EmployeeDAO {      ...  } |

2) The @Repository annotation is a specialization of the @Component annotation with similar use and functionality. In addition to importing the DAOs into the DI container, it also makes the unchecked exceptions (thrown from DAO methods) eligible for translation into Spring DataAccessException.

3) The @Service annotation is also a specialization of the component annotation. It doesn’t currently provide any additional behavior over the @Component annotation, but it’s a good idea to use @Service over @Component in service-layer classes because it specifies intent better.

4) @Controller annotation marks a class as a Spring Web MVC controller. It too is a @Component specialization, so beans marked with it are automatically imported into the DI container. When you add the @Controller annotation to a class, you can use another annotation i.e. @RequestMapping; to map URLs to instance methods of a class.

**Read More :**[**@Component, @Repository, @Service and @Controller Annotations?**](http://howtodoinjava.com/spring/spring-core/how-to-use-spring-component-repository-service-and-controller-annotations/)

## What does the ViewResolver class?

ViewResolver is an interface to be implemented by objects that can resolve views by name. There are plenty of ways using which you can resolve view names. These ways are supported by various in-built implementations of this interface. Most commonly used implementation is InternalResourceViewResolver class. It defines **prefix** and **suffix**properties to resolve the view component.

|  |
| --- |
| <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">      <property name="prefix" value="/WEB-INF/views/" />      <property name="suffix" value=".jsp" />  </bean> |

So with above view resolver configuration, if controller method return “login” string, then the “/WEB-INF/views/login.jsp” file will be searched and rendered.

## What is a MultipartResolver and when its used?

Spring comes with MultipartResolver to handle file upload in web application. There are two concrete implementations included in Spring:

1. **CommonsMultipartResolver** for Jakarta Commons FileUpload
2. **StandardServletMultipartResolver** for Servlet 3.0 Part API

To define an implementation, create a bean with the id “**multipartResolver**” in a DispatcherServlet’s application context. Such a resolver gets applied to all requests handled by that DispatcherServlet.

If a DispatcherServlet detects a multipart request, it will resolve it via the configured MultipartResolver and pass on a wrapped HttpServletRequest. Controllers can then cast their given request to the MultipartHttpServletRequestinterface, which permits access to any MultipartFiles.

## How to upload file in Spring MVC Application?

Let’s say we are going to use CommonsMultipartResolver which uses the Apache commons upload library to handle the file upload in a form. So you will need to add the **commons-fileupload.jar** and **commons-io.jar** dependencies.

|  |
| --- |
| <!-- Apache Commons Upload -->  <dependency>      <groupId>commons-fileupload</groupId>      <artifactId>commons-fileupload</artifactId>      <version>1.2.2</version>  </dependency>    <!-- Apache Commons Upload -->  <dependency>      <groupId>commons-io</groupId>      <artifactId>commons-io</artifactId>      <version>1.3.2</version>  </dependency> |

The following declaration needs to be made in the application context file to enable the MultipartResolver (along with including necessary jar file in the application):

|  |
| --- |
| <bean id="multipartResolver" class="org.springframework.web.multipart.commons.CommonsMultipartResolver">      <!-- one of the properties available; the maximum file size in bytes -->      <property name="maxUploadSize" value="100000"/>  </bean> |

Now create model class FileUploadForm which will hold the multipart data submitted from HTML form.

|  |
| --- |
| import org.springframework.web.multipart.MultipartFile;    public class FileUploadForm  {      private MultipartFile file;        public MultipartFile getFile() {          return file;      }        public void setFile(MultipartFile file) {          this.file = file;      }  } |

Now create FileUploadController class which will actually handle the upload logic.

|  |
| --- |
| import org.springframework.stereotype.Controller;  import org.springframework.ui.Model;  import org.springframework.web.bind.annotation.ModelAttribute;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestMethod;  import org.springframework.web.multipart.MultipartFile;  import com.howtodoinjava.form.FileUploadForm;    @Controller  public class FileUploadController  {      @RequestMapping(value = "/upload", method = RequestMethod.POST)      public String save(@ModelAttribute("uploadForm") FileUploadForm uploadForm, Model map) {            MultipartFile multipartFile = uploadForm.getFile();            String fileName = "default.txt";            if (multipartFile != null) {              fileName = multipartFile.getOriginalFilename();          }            //read and store the file as you like            map.addAttribute("files", fileName);          return "file\_upload\_success";      }  } |

The upload JSP file looks like this:

|  |
| --- |
| <%@ taglib prefix="form" uri="<http://www.springframework.org/tags/form>"%>  <html>  <body>      <h2>Spring MVC file upload example</h2>      <form:form method="post" action="save.html" modelAttribute="uploadForm" enctype="multipart/form-data">          Please select a file to upload : <input type="file" name="file" />          <input type="submit" value="upload" />          <span><form:errors path="file" cssClass="error" /></span>      </form:form>  </body>  </html> |

## How does Spring MVC provide validation support?

Spring supports validations primarily into two ways.

1. Using **JSR-303 Annotations** and any reference implementation e.g. Hibernate Validator
2. Using **custom implementation of org.springframework.validation.Validator** interface

In next question, you see an example about how to use validation support in spring MVC application.

## How to validate form data in Spring Web MVC Framework?

Spring MVC supports validation by means of a validator object that implements the Validator interface. You need to create a class and implement Validator interface. In this custom validator class, you use utility methods such as rejectIfEmptyOrWhitespace() and rejectIfEmpty() in the ValidationUtils class to validate the required form fields.

|  |
| --- |
| @Component  public class EmployeeValidator implements Validator  {      public boolean supports(Class clazz) {          return EmployeeVO.class.isAssignableFrom(clazz);      }        public void validate(Object target, Errors errors)      {          ValidationUtils.rejectIfEmptyOrWhitespace(errors, "firstName", "error.firstName", "First name is required.");          ValidationUtils.rejectIfEmptyOrWhitespace(errors, "lastName", "error.lastName", "Last name is required.");          ValidationUtils.rejectIfEmptyOrWhitespace(errors, "email", "error.email", "Email is required.");      }  } |

If any of form fields is empty, these methods will create a field error and bind it to the field. The second argument of these methods is the property name, while the third and fourth are the error code and default error message.

To activate this custom validator as a spring managed bean, you need to do one of following things:

1) Add @Component annotation to EmployeeValidator class and activate annotation scanning on the package containing such declarations.

|  |
| --- |
| <context:component-scan base-package="com.howtodoinjava.demo" /> |

2) Alternatively, you can register the validator class bean directly in context file.

|  |
| --- |
| <bean id="employeeValidator" class="com.howtodoinjava.demo.validator.EmployeeValidator" /> |

**Read More : Spring MVC**[**Custom Validator**](http://howtodoinjava.com/spring/spring-mvc/spring-mvc-custom-validator-example/)**and**[**JSR-303 Annotations**](http://howtodoinjava.com/spring/spring-mvc/spring-bean-validation-example-with-jsr-303-annotations/)**Examples**

## What is Spring MVC Interceptor and how to use it?

As you know about servlet filters that they can pre-handle and post-handle every web request they serve — before and after it’s handled by that servlet. In the similar way, you can use HandlerInterceptor interface in your spring mvc application **to pre-handle and post-handle web requests** that are handled by Spring MVC controllers. These handlers are mostly used to manipulate the model attributes returned/submitted they are passed to the views/controllers.

A handler interceptor can be registered for particular URL mappings, so it only intercepts requests mapped to certain URLs. Each handler interceptor must implement the HandlerInterceptor interface, which contains three callback methods for you to implement: preHandle(), postHandle() and afterCompletion().

Problem with HandlerInterceptor interface is that your new class will have to implement all three methods irrespective of whether it is needed or not. To avoid overriding, you can use HandlerInterceptorAdapter class. This class implements HandlerInterceptor and provide default blank implementations.

**Read More :**[**Spring MVC Interceptor Example**](http://howtodoinjava.com/spring/spring-mvc/spring-intercepting-requests-using-handlerinterceptor-with-example/)

## How to handle exceptions in Spring MVC Framework?

In a Spring MVC application, you can register one or more exception resolver beans in the web application context to resolve uncaught exceptions. These beans have to implement the HandlerExceptionResolver interface for DispatcherServlet to auto-detect them. Spring MVC comes with a simple exception resolver for you to map each category of exceptions to a view i.e. SimpleMappingExceptionResolver to map each category of exceptions to a view in a configurable way.

Let’s say we have an exception class i.e. AuthException. And we want that everytime this exception is thrown from anywhere into application, we want to show a pre-determined view page /WEB-INF/views/error/authExceptionView.jsp. So the configuration would be.

|  |
| --- |
| <bean class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">      <property name="exceptionMappings">          <props>              <prop key="com.howtodoinjava.demo.exception.AuthException">                  error/authExceptionView              </prop>          </props>      </property>      <property name="defaultErrorView" value="error/genericView"/>  </bean> |

The “**defaultErrorView**” property can be configured to show a generic message for all other exceptions which are not configured inside “**exceptionMappings**” list.

**Read More :**[**Spring MVC SimpleMappingExceptionResolver Example**](http://howtodoinjava.com/spring/spring-mvc/spring-mvc-simplemappingexceptionresolver-example/)

## How to achieve localization in Spring MVC applications?

Spring framework is shipped with LocaleResolver to support the internationalization and thus localization as well. To make Spring MVC application supports the internationalization, you will need to register two beans.

**1) SessionLocaleResolver** : It resolves locales by inspecting a predefined attribute in a user’s session. If the session attribute doesn’t exist, this locale resolver determines the default locale from the accept-language HTTP header.

|  |
| --- |
| <bean id="localeResolver" class="org.springframework.web.servlet.i18n.SessionLocaleResolver">      <property name="defaultLocale" value="en" />  </bean> |

**2) LocaleChangeInterceptor** : This interceptor detects if a special parameter is present in the current HTTP request. The parameter name can be customized with the **paramName** property of this interceptor. If such a parameter is present in the current request, this interceptor changes the user’s locale according to the parameter value.

|  |
| --- |
| <bean id="localeChangeInterceptor" class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">      <property name="paramName" value="lang" />  </bean>    <!-- Enable the interceptor -->  <bean class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping">      <property name="interceptors">          <list>              <ref bean="localeChangeInterceptor" />          </list>      </property>  </bean> |

Next step is to have each locale specific properties file having texts in that locale specific language e.g. messages.properties and messages\_zh\_CN.properties etc.

**Read More :**[**Spring MVC Localization (i10n) Example**](http://howtodoinjava.com/spring/spring-mvc/spring-mvc-internationalization-i18n-and-localization-i10n-example/)

## How to get ServletContext and ServletConfig object in a Spring Bean?

Simply implement ServletContextAware and ServletConfigAware interfaces and override below methods.

|  |
| --- |
| @Controller  @RequestMapping(value = "/magic")  public class SimpleController implements ServletContextAware, ServletConfigAware {        private ServletContext context;      private ServletConfig config;        @Override      public void setServletConfig(final ServletConfig servletConfig) {          this.config = servletConfig;        }        @Override      public void setServletContext(final ServletContext servletContext) {          this.context = servletContext;      }        //other code  } |

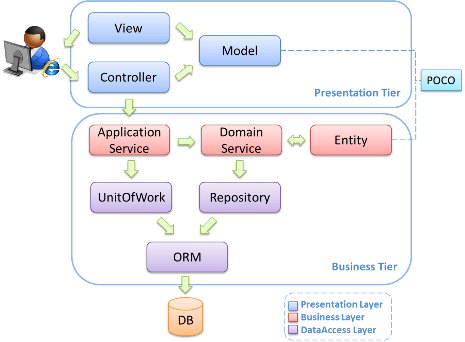
## How to use Tomcat JNDI DataSource in Spring Web Application?

For using servlet container configured JNDI DataSource, we need to configure it in the spring bean configuration file and then inject it to spring beans as dependencies. Then we can use it with JdbcTemplate to perform database operations.

|  |
| --- |
| <bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean">      <property name="jndiName" value="java:comp/env/jdbc/MySQLDB"/>  </bean> |

## How would you relate Spring MVC Framework to 3 tier architecture?

**3-tier is a Architecture Style and MVC is a Design Pattern.**

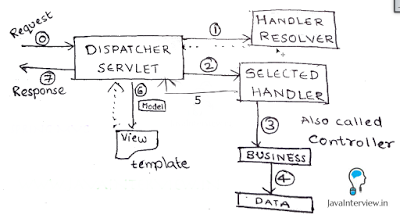
[](http://howtodoinjava.com/wp-content/uploads/2015/02/3-tier-architechture-with-mvc-part-of-it.png)

**In larger applications MVC is the presentation tier only of an 3-tier architecture.** The models views and controllers are only concerned with the presentation, and make use of a middle tier to populate the models with data from the data tier.

**Please share any other good Spring MVC interview questions you may have faced** so that I can include that in this post to benefit others as well.

1. [How does request flow happen in Spring MVC?](http://www.javainterview.in/p/spring-mvc-interview-questions.html#question1)
2. [Can you list a few advantages of using Spring MVC framework?](http://www.javainterview.in/p/spring-mvc-interview-questions.html#question2)
3. [Give examples of important Spring MVC annotations?](http://www.javainterview.in/p/spring-mvc-interview-questions.html#question3)
4. [Can you explain the concept of Interceptors in Spring MVC?](http://www.javainterview.in/p/spring-mvc-interview-questions.html#question4)
5. [Interceptors can be configured using the interceptors property.](http://www.javainterview.in/p/spring-mvc-interview-questions.html#question5)
6. [How do you schedule tasks with Spring?](http://www.javainterview.in/p/spring-mvc-interview-questions.html#question6)
7. [How do you integrate Spring MVC with tiles?](http://www.javainterview.in/p/spring-mvc-interview-questions.html#question7)
8. [How do you configure Spring MVC web application to use UTF-8 encoding for handling forms?](http://www.javainterview.in/p/spring-mvc-interview-questions.html#question8)
9. [How do you enable spring security for a web application?](http://www.javainterview.in/p/spring-mvc-interview-questions.html#question9)

How does request flow happen in Spring MVC?

[](http://4.bp.blogspot.com/-iz_lkHWqkNM/VVDHgFUkLsI/AAAAAAAAAHY/i9l_PNDFe3U/s1600/Spring+MVC+Request+Flow.png)

Shown in the picture below. DispatcherServlet acts as the front controller. Simplified actions taken by DispatcherServlet are listed below.

* All requests arrive at the DispatcherServlet (Front Controller) - STEP 0 in Figure
* DispatcherServlet resolves theme and locale as configured.
* Find’s appropriate Controller (Handler) to handle the request. (pre-processors and post-processors, if configured) (STEP 1)
* Redirect to the Controller (Handler) - STEP 2. Controller executes the request and returns a view name and a view model object. (STEP 3,4,5)
* DispatcherServlet resolves the view name and redirects to the view template. The response html is returned to DispatcherServlet. (STEP 6)
* DispatcherServlet send the response back to the browser. (STEP 7)

Can you list a few advantages of using Spring MVC framework?

* In Spring Web MVC, any POJO can be used as a command or form-backing object.
* Highly flexible databinding – If there is a type mismatch, it is shown as a validation error on the screen. Business POJO’s can directly be used as form-backing objects.
* Flexible view resolution: Controller can either select a view name and prepare model map for it or write directly to response stream.
* Supports JSP, Velocity and Freemarker view technologies.
* Can directly generate XML, JSON, Atom, and many other types of content.
* Highly convenient tag library.

Give examples of important Spring MVC annotations?

Important Spring MVC annotations are listed below.

* @Controller : This class would serve as a controller.
* @RequestMapping : Can be used on a class or a method. Maps an url on the class (or method).
* @PathVariable : Used to map a dynamic value in the url to a method argument.

Example 1 : Maps a url “/players “ for the controller method.

@RequestMapping(value="/players", method=RequestMethod.GET)

public String findAllPlayers(Model model) {

Example 2 : If a url /players/15 is keyed in, playerId is populated with value 15.

@RequestMapping(value="/players/{playerid}", method=RequestMethod.GET)

public String findPlayer(@PathVariable String playerId, Model model) {

Can you explain the concept of Interceptors in Spring MVC?

Handler interceptors are used when you want to apply specific functionality to certain requests. Handler Interceptors should implement the interface HandlerInterceptor.

Three methods are defined:

* preHandle(..) is called before the actual handler is executed;
* postHandle(..) is called after the handler is executed;
* afterCompletion(..) is called after the complete request has finished.

Interceptors can be configured using the interceptors property.

<bean id="handlerMapping" class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping">

<property name="interceptors">

<list>

<ref bean="yourCustomHandlerInterceptor"/>

</list>

</property>

</bean>

How do you schedule tasks with Spring?

Spring 3.0 introduced TaskScheduler abstract to deal with scheduling jobs. Spring has support for Timer (Jdk) and Quartz. Sample methods in the interface TaskScheduler are shown below:

ScheduledFuture scheduleAtFixedRate(Runnable task, long period);

ScheduledFuture scheduleWithFixedDelay(Runnable task, long delay);

Scheduling can also be done using an annotation

@Scheduled(fixedDelay=5000)

public void doSomething() {

// something that should execute periodically

}

Below example shows scheduling with xml configuration

<task:scheduler id="customScheduler" pool-size="30"/>

<task:scheduled-tasks scheduler=" customScheduler ">

<task:scheduled ref="someBean" method="someOtherMethod" fixed-delay="5000" initial-delay="1000"/>

<task:scheduled ref="someOtherBean" method="someMethod" cron="\*/5 \* \* \* \* MON-FRI"/>

</task:scheduled-tasks>

How do you integrate Spring MVC with tiles?

Tiles helps us to define the layout for a web page. We can integrate Spring MVC with tiles by configuring TilesConfigurer and setting up appropriate view resolver.

<bean id="tilesConfigurer"

class="org.springframework.web.servlet.view.tiles2.TilesConfigurer"

p:definitions="/WEB-INF/tiles-defs/templates.xml" />

<bean id="tilesViewResolver"

class="org.springframework.web.servlet.view.UrlBasedViewResolver"

p:viewClass="org.springframework.web.servlet.view.tiles2.TilesView" />

How do you configure Spring MVC web application to use UTF-8 encoding for handling forms?

Using org.springframework.web.filter.CharacterEncodingFilter. Shown below.

<filter>

<filter-name>encoding-filter</filter-name>

<filter-class>

org.springframework.web.filter.CharacterEncodingFilter

</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>UTF-8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>encoding-filter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

How do you enable spring security for a web application?

Spring Security is used to implement Authentication and Authorization for a web application. We can enable spring security by configuring an appropriae security filter. Example shown below. We can create a separate security-context.xml to define the authentication and authorization roles and accesses.

<filter>

<filter-name>springSecurityFilterChain</filter-name>

<filter-class>

org.springframework.web.filter.DelegatingFilterProxy

</filter-class>

</filter>

<filter-mapping>

<filter-name>springSecurityFilterChain</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

1. If you loved these Questions, you will love our PDF Interview Guide with 400+ Questions.   
   [**What is Spring MVC framework?**](https://www.dineshonjava.com/2017/02/spring-mvc-interview-questions-and-answers.html#1)
2. [**MVC is an abbreviation for a design pattern. What does it stand for and what is the idea behind it?**](https://www.dineshonjava.com/2017/02/spring-mvc-interview-questions-and-answers.html#2)
3. [**Do you need spring-mvc.jar in your classpath or is it part of spring-core?**](https://www.dineshonjava.com/2017/02/spring-mvc-interview-questions-and-answers.html#3)
4. [**What is the DispatcherServlet and what is it used for?**](https://www.dineshonjava.com/2017/02/spring-mvc-interview-questions-and-answers.html#4)
5. [**Is the DispatcherServlet instantiated via an application context?**](https://www.dineshonjava.com/2017/02/spring-mvc-interview-questions-and-answers.html#5)
6. [**What is the root application context? How is it loaded?**](https://www.dineshonjava.com/2017/02/spring-mvc-interview-questions-and-answers.html#6)
7. [**What is the @Controller annotation used for? How can you create a controller without an annotation?**](https://www.dineshonjava.com/2017/02/spring-mvc-interview-questions-and-answers.html#7)
8. [**What is the ContextLoaderListener and what does it do?**](https://www.dineshonjava.com/2017/02/spring-mvc-interview-questions-and-answers.html#8)
9. [**What are you going to do in the web.xml. Where do you place it?**](https://www.dineshonjava.com/2017/02/spring-mvc-interview-questions-and-answers.html#9)
10. [**How is an incoming request mapped to a controller and mapped to a method?**](https://www.dineshonjava.com/2017/02/spring-mvc-interview-questions-and-answers.html#10)
11. [**What is the @RequestParam used for?**](https://www.dineshonjava.com/2017/02/spring-mvc-interview-questions-and-answers.html#11)
12. [**What are the differences between @RequestParam and @PathVariable?**](https://www.dineshonjava.com/2017/02/spring-mvc-interview-questions-and-answers.html#12)
13. [**What are some of the valid return types of a controller method?**](https://www.dineshonjava.com/2017/02/spring-mvc-interview-questions-and-answers.html#13)
14. [**What is a View and what’s the idea behind supporting different types of View?**](https://www.dineshonjava.com/2017/02/spring-mvc-interview-questions-and-answers.html#14)
15. [**How is the right View chosen when it comes to the rendering phase?**](https://www.dineshonjava.com/2017/02/spring-mvc-interview-questions-and-answers.html#15)
16. [**What is the Model?**](https://www.dineshonjava.com/2017/02/spring-mvc-interview-questions-and-answers.html#16)
17. [**Why do you have access to the model in your View? Where does it come from?**](https://www.dineshonjava.com/2017/02/spring-mvc-interview-questions-and-answers.html#17)
18. [**What is the purpose of the session scope?**](https://www.dineshonjava.com/2017/02/spring-mvc-interview-questions-and-answers.html#18)
19. [**What is the default scope in the web context?**](https://www.dineshonjava.com/2017/02/spring-mvc-interview-questions-and-answers.html#19)
20. [**Why are controllers testable artifacts?**](https://www.dineshonjava.com/2017/02/spring-mvc-interview-questions-and-answers.html#20)
21. [**What does the InternalResourceViewResolver do?**](https://www.dineshonjava.com/2017/02/spring-mvc-interview-questions-and-answers.html#21)
22. [**What’s internal resource views?**](https://www.dineshonjava.com/2017/02/spring-mvc-interview-questions-and-answers.html#22)

**Popular Tutorials**

* [***Spring Tutorial***](https://www.dineshonjava.com/2012/06/spring-30-baby-step-to-learn.html)
* [***Spring MVC Web Tutorial***](https://www.dineshonjava.com/2012/12/spring-web-mvc-framework-chapter-38.html)
* [***Spring Boot Tutorial***](https://www.dineshonjava.com/2016/06/introduction-to-spring-boot-a-spring-boot-complete-guide.html)
* [***Spring Security Tutorial***](https://www.dineshonjava.com/2013/02/spring-security-take-baby-step-to-secure.html)
* [***Spring AOP Tutorial***](https://www.dineshonjava.com/2012/07/introduction-to-aop-in-spring.html)
* [***Spring JDBC Tutorial***](https://www.dineshonjava.com/2012/12/using-spring-jdbc-framework-chapter-32.html)
* [***Spring HATEOAS***](https://www.dineshonjava.com/2017/01/spring-hateoas-hypermedia-driven-restful-web-service.html)
* [***Microservices with Spring Boot***](https://www.dineshonjava.com/2017/01/microservices-with-spring-boot.html)
* [***REST Webservice***](https://www.dineshonjava.com/2013/06/jax-rs-web-service-tutorial.html)
* [***Core Java***](https://www.dineshonjava.com/2013/01/core-java-baby-step-to-be-best-java-ian.html)
* [***Hibernate Tutorial***](https://www.dineshonjava.com/2012/03/hibernate-3-on-baby-steps.html)
* [***Spring Batch***](https://www.dineshonjava.com/2012/12/spring-batch-process-with-example.html)

**1. What is Spring MVC framework?**  
The Spring web MVC framework provides model-view-controller 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. **Read More about**[**Spring MVC Framework**](https://www.dineshonjava.com/2012/12/spring-web-mvc-framework-chapter-38.html)**.**

**2. MVC is an abbreviation for a design pattern. What does it stand for and what is the idea behind it?**  
MVC is an abbreviation name for a J2EE design pattern and it stands for Model View Controller. According to MVC design pattern in J2EE application we have three layers of architecture as Controller, Model and Service layers. A Model View Controller pattern is made up of the following three parts:

* Model – The lowest level of the pattern which is responsible for maintaining data.
* View – This is responsible for displaying all or a portion of the data to the user.
* Controller – Software Code that controls the interactions between the Model and View.

MVC is popular as it isolates the application logic from the user interface layer and supports separation of concerns. Here the Controller receives all requests for the application and then works with the Model to prepare any data needed by the View. The View then uses the data prepared by the Controller to generate a final presentable response. **Read more about**[**MVC design pattern**](https://www.dineshonjava.com/2013/07/introduction-to-mvc.html)**.**

**3. Do you need spring-mvc.jar in your classpath or is it part of spring-core?**  
In J2EE application with Spring we required spring-mvc jar in application class path due to it provides the mvc pattern support classes. The form tag library comes bundled in spring-webmvc.jar. The spring-webmvc module (also known as the Web-Servlet module) contains Spring’s model-view-controller (MVC) and REST Web Services implementation for web applications. Spring’s MVC framework provides a clean separation between domain model code and web forms and integrates with all of the other features of the Spring Framework.

Spring-mvc.jar is not part of spring core because the Core Container consists of the spring-core, spring-beans, spring-context, spring-context-support, and spring-expression (Spring Expression Language) modules.

**4. What is the DispatcherServlet and what is it used for?**  
In Spring MVC framework Dispatcher Servlet access Front Controller which handles all coming requests and queues for forward to the different controller. Front controller is a typical design pattern in the web applications development. In this case, a single servlet receives all requests and transfers them to to all other components of the application. The task of the DispatcherServlet is send request to the specific Spring MVC controller.

DispatcherServlet is responsible for initialize the WebApplicationContext and it loads all configuration related to the web components like controllers, view resolver, interceptors etc., It will be loaded and initialized by calling init() method init() of DispatcherServlet will try to identify the Spring Configuration Document with naming conventions like “**servlet\_name-servlet.xml**” then all beans can be identify. **Read more about [DispatcherServlet in Spring MVC](https://www.dineshonjava.com/2017/02/what-is-dispatcherservlet-in-spring-and-its-uses.html).**

**5. Is the DispatcherServlet instantiated via an application context?**  
No. ApplicationContext instantiated via DispactcherServlet.

**6. What is the root application context? How is it loaded?**  
In spring mvc for every web application applicationContext.xml file used as the root context configuration. Spring loads this file and creates the ApplicationContext for whole application. This file applicationContext.xml is loaded by ContextLoaderLoaderLinstner which is configured into web.xml file as the context configuration.**Read more about**[**Root Application Context and WebApplicationContext in Spring MVC**](https://www.dineshonjava.com/2017/02/difference-between-applicationcontext-webapplicationcontext-in-spring-mvc.html)**.**

**7. What is the @Controller annotation used for? How can you create a controller without an annotation?**  
In Spring MVC framework @Controller used for creating controllers for the web application. It is meta annotation of @Component annotation to create the bean by spring container.

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. **Read more about**[**@Controller, @Repository and @Service stereotype annotations**](https://www.dineshonjava.com/2012/07/stereotype-annotations-in-spring.html)**.**

**8. What is the ContextLoaderListener and what does it do?**  
This listener is responsible to load the context configuration files. It performs the actual initialization work for the root application context. It reads a “contextConfigLocation” context-param and passes its value to the context instance. We can pass multiple files in the context configuration by commas or space separation. e.g. “WEB-INF/applicationContext.xml, WEB-INF/applicationContext-security.xml”. **Read more about [ContextLoaderListener in Spring MVC](https://www.dineshonjava.com/2017/02/difference-between-applicationcontext-webapplicationcontext-in-spring-mvc.html).**

**9. What are you going to do in the web.xml. Where do you place it?**  
For Spring MVC application we have to do following entry in web.xml file which place under the WEB-INF folder.

**A. Configure spring dispatcher servlet**

<servlet>

<servlet-name>webapp1</servlet-name>

<servlet-class>

org.springframework.web.servlet.DispatcherServlet

</servlet-class>

<!-- We require this configuration when we want to change the default

name / location of the servlet specific configuration files -->

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/app1-servlet.xml</param-value>

</init-param>

<load-on-startup>0</load-on-startup>

</servlet>

**B. Configure Spring ContexLoaderListner but it is optional**

<!-- This is the root application context for whole web application. -->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/rootApplicationContext.xml</param-value>

</context-param>

<listener>

<listener-class>

org.springframework.web.context.ContextLoaderListener

</listener-class>

</listener>

**10. What is @RequestMapping? How is an incoming request mapped to a controller and mapped to a method?**  
@RequestMapping can be applied to the controller class as well as methods. This annotation for mapping web requests onto specific handler classes and/or handler methods. **Read more about**[**@RequestMapping annotation in Spring MVC**](https://www.dineshonjava.com/2016/03/requestmapping-annotation-in-spring-mvc.html)**.**

**11. What is the @RequestParam used for?**  
In spring the @RequestParam annotation is used to bind request parameter values to the handler method arguments in controller.

@Target(value=PARAMETER)

@Retention(value=RUNTIME)

@Documented

public @interface RequestParam

* **defaultValue**-It is String type attribute and the default value to use as a fallback when the request parameter is not provided or has an empty value.
* **name**-It is String type attribute and name of the request parameter to bind to.
* **required**-It is boolean type attribute whether the parameter is required.
* **value**– It is String type attribute and it is alias for name attribute.

**Read more about**[**@RequestParam annotation in Spring MVC**](https://www.dineshonjava.com/2017/02/requestparam-annotation-in-spring-mvc-with-example.html)**.**

**12. What are the differences between @RequestParam and @PathVariable?**  
**@PathVariable:** Is is used to pass parameter along with the url, sometimes we need to pass parameters along with the url to get the data. Spring MVC provides support for customizing the URL in order to get data. To achieving this purpose @PathVariable annotation is used in Spring mvc framework.

**@RequestParam:** It is used to get the request parameters. @RequestParam automatically binds the request parameters to the arguments of your handler method. It also provides auto type conversion for some standard type like int, long, float, string, date etc.**Read more about**[**differences between @RequestParam and @PathVariable annotation in Spring MVC**](https://www.dineshonjava.com/2017/02/requestparam-vs-pathvariable-annotations-in-spring-mvc.html)**.**

**13. What are some of the valid return types of a controller method?**  
There are many return types are available for Handler method which is annotated by @RequestMapping inside controller like :

* ModelAndView (Class)
* Model (Interface)
* Map
* String
* void
* View
* HttpEntity<?> or ResponseEntity<?>
* HttpHeaders

and much more…..[See Docs](http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#mvc-ann-return-types)

**14. What is a View and what’s the idea behind supporting distinct types of View?**

* A View renders web output.
* Many built-in views available for JSPs, XSLT, templating approaches (Velocity, FreeMarker), etc.
* View support classes for creating PDFs, Excel spreadsheets,etc.
* Controllers typically return a ‘logical view name’ String.
* ViewResolvers select View based on view name.

**15. How is the right View chosen when it comes to the rendering phase?**  
**View Resolvers**

* The DispatcherServlet delegates to a ViewResolver to obtain View implementation based on view name.
* The default ViewResolver treats the view name as a Web Application-relative file path – i.e. a JSP: /WEB-INF/reward/list.jsp
* Override this default by registering a ViewResolver bean with the DispatcherServlet
  + We will use InternalResourceViewResolver
  + Several other options available.

**16. What is the Model?**  
Model in Spring MVC has processed data to be rendered in the view by the respective view resolver. Model has two part Service and Persistent.

**17. Why do you have access to the model in your View? Where does it come from?**  
In the Spring MVC model has data to be rendered front to the user so spring provide support to access the model in the view layer. This model is passed by the controller to the dispatcher servlet and dispatcher server send it to the respective view resolver for rendering model’s data to the view and generate html page upfront to user.

**18. What is the purpose of the session scope?**  
This scopes a bean definition to an HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

The idea here is that you have a web application, and you have various objects that exist on a per-session basis, such as maybe a user profile or a shopping cart. You’d like to make those available to some service bean, say, without having to manually pull them off the session and pass them as method arguments every time.

**19. What is the default scope in the web context?**  
singleton

**20. Why are controllers testable artifacts?**  
Spring provides support for mockito framework for testing. Mockito provides MockMVC class is the main entry point of our tests. We can execute requests by calling its perform method.

**21. What does the InternalResourceViewResolver do?**  
InternalResourceViewResolver is one of the implementation of ViewResolver for JSP templates. It is used to resolve “internal resource view” based on a predefined URL pattern. In additional, it allow you to add some predefined prefix or suffix to the view name (prefix + view name + suffix), and generate the final view page URL.

**22. What’s internal resource views?**  
Those views under “WEB-INF” folder are named as internal resource views, as it’s only accessible by the servlet or Spring’s controllers class. In Spring MVC web application, for good practice, it’s always recommended to put the entire views or JSP files under “WEB-INF” folder, to protect it from direct access via manual entered URL.

**Spring MVC Related Posts**

1. [**Spring MVC Web Tutorial**](https://www.dineshonjava.com/2012/12/spring-web-mvc-framework-chapter-38.html)
2. [**Spring MVC InternalResourceViewResolver**](https://www.dineshonjava.com/2017/02/spring-mvc-internalresourceviewresolver-configuration-example.html)
3. [**MVC Design Pattern**](https://www.dineshonjava.com/2013/07/introduction-to-mvc.html)
4. [**Spring MVC DispatcherServlet**](https://www.dineshonjava.com/2017/02/what-is-dispatcherservlet-in-spring-and-its-uses.html)
5. [**Spring MVC WebApplicationContext and Root Application Context**](https://www.dineshonjava.com/2017/02/difference-between-applicationcontext-webapplicationcontext-in-spring-mvc.html)
6. [**Spring MVC @Controller Annotation**](https://www.dineshonjava.com/2012/07/stereotype-annotations-in-spring.html)
7. [**Spring MVC @RequestMapping Annotation**](https://www.dineshonjava.com/2016/03/requestmapping-annotation-in-spring-mvc.html)
8. [**Spring MVC @RequestParam Annotation**](https://www.dineshonjava.com/2017/02/requestparam-annotation-in-spring-mvc-with-example.html)
9. [**Spring MVC ContextLoaderListener**](https://www.dineshonjava.com/2017/02/difference-between-applicationcontext-webapplicationcontext-in-spring-mvc.html)
10. [**Spring MVC @RequestParam and @PathVariable annotations**](https://www.dineshonjava.com/2017/02/requestparam-vs-pathvariable-annotations-in-spring-mvc.html)
11. [**Spring MVC Hello World Example**](https://www.dineshonjava.com/2012/12/spring-30-mvc-hello-world-example.html)
12. [**Spring MVC Exception Handling Example**](https://www.dineshonjava.com/2012/12/spring-exception-handling-example.html)
13. [**Spring MVC with Hibernate CRUD Example**](https://www.dineshonjava.com/2012/12/spring-mvc-with-hibernate-crud-example.html)
14. [**Spring MVC Tiles Plugin with Example**](https://www.dineshonjava.com/2012/12/spring-3-mvc-tiles-plugin-with-example.html)
15. [**Spring MVC Interceptor with example**](https://www.dineshonjava.com/2012/12/spring-3-mvc-and-interceptor-with.html)
16. [**Spring MVC with MongoDB CRUD Example**](https://www.dineshonjava.com/2013/01/integration-spring-mvc3-and-mongodb.html)
17. [**Spring MVC Internationalization & Localization with Example**](https://www.dineshonjava.com/2012/12/spring-3-mvc-internationalization.html)

### 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.

### 3. Which are the Spring framework modules?

The basic modules of the Spring framework are :

* Core module
* Bean module
* Context module
* Expression Language module
* [JDBC module](http://examples.javacodegeeks.com/enterprise-java/spring/jdbc/spring-jdbctemplate-example/)
* [ORM module](http://examples.javacodegeeks.com/enterprise-java/spring/jpaorm/spring-hibernate-mysql-and-maven-showcase/)
* OXM module
* Java Messaging Service(JMS) module
* Transaction module
* Web module
* Web-Servlet module
* Web-Struts module
* Web-Portlet module

### 4. Explain the Core Container (Application context) module

This is the basic Spring module, which provides the fundamental functionality of the Spring framework. BeanFactory is the heart of any spring-based application. Spring framework was built on the top of this module, which makes the Spring container.

### 5. BeanFactory – BeanFactory implementation example

A BeanFactory is an implementation of the factory pattern that applies Inversion of Control to separate the application’s configuration and dependencies from the actual application code.

The most commonly used BeanFactory implementation is the XmlBeanFactory class.

### 6. XMLBeanFactory

The most useful one is org.springframework.beans.factory.xml.XmlBeanFactory, which loads its beans based on the definitions contained in an XML file. This container reads the configuration metadata from an XML file and uses it to create a fully configured system or application.

### 7. Explain the AOP module

The AOP module is used for developing aspects for our Spring-enabled application. Much of the support has been provided by the AOP Alliance in order to ensure the interoperability between [Spring and other AOP frameworks](http://www.javacodegeeks.com/2014/02/applying-aspect-oriented-programming.html). This module also introduces metadata programming to Spring.

### 8. Explain the JDBC abstraction and DAO module

With the [JDBC abstraction and DAO module](http://examples.javacodegeeks.com/enterprise-java/spring/jdbc/spring-jdbctemplate-example/) we can be sure that we keep up the database code clean and simple, and prevent problems that result from a failure to close database resources. It provides a layer of meaningful exceptions on top of the error messages given by several database servers. It also makes use of Spring’s AOP module to provide transaction management services for objects in a Spring application.

### 9. Explain the object/relational mapping integration module

Spring also supports for using of an [object/relational mapping (ORM) too](http://www.javacodegeeks.com/2011/12/persistence-layer-with-spring-31-and_14.html)l over straight JDBC by providing the ORM module. Spring provides support to tie into several popular ORM frameworks, including [Hibernate](http://www.javacodegeeks.com/2010/05/jboss-42x-spring-3-jpa-hibernate.html), JDO, and [iBATIS SQL Maps](http://www.javacodegeeks.com/2012/02/mybatis-3-spring-integration-tutorial.html). Spring’s transaction management supports each of these ORM frameworks as well as JDBC.

### 10. Explain the web module

The [Spring web module](http://examples.javacodegeeks.com/enterprise-java/spring/mvc/spring-mvc-hello-world-example/) is built on the application context module, providing a context that is appropriate for web-based applications. This module also contains support for several web-oriented tasks such as transparently handling multipart requests for file uploads and programmatic binding of request parameters to your business objects. It also contains integration support with Jakarta Struts.

### 11. Explain the Spring MVC module

MVC framework is provided by Spring for building web applications. Spring can easily be integrated with other MVC frameworks, but [Spring’s MVC framework](http://www.javacodegeeks.com/2012/09/spring-adding-spring-mvc-part-1.html) is a better choice, since it uses IoC to provide for a clean separation of controller logic from business objects. With Spring MVC you can declaratively bind request parameters to your business objects.

### 12. Spring configuration file

Spring configuration file is an XML file. This file contains the classes information and describes how these classes are configured and introduced to each other.

### 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.

### 16. What is the difference between Bean Factory and ApplicationContext?

Application contexts provide a means for resolving text messages, a generic way to load file resources (such as images), they can publish events to beans that are registered as listeners. In addition, operations on the container or beans in the container, which have to be handled in a programmatic fashion with a bean factory, can be handled declaratively in an application context. The application context implements MessageSource, an interface used to obtain localized messages, with the actual implementation being pluggable.

### 17. What does a Spring application look like?

* An interface that defines the functions.
* The implementation that contains properties, its setter and getter methods, functions etc.,
* [Spring AOP](http://examples.javacodegeeks.com/enterprise-java/spring/aop/spring-aop-example/)
* The Spring configuration XML file.
* Client program that uses the function

## **Dependency Injection**

### 18. What is Dependency Injection in Spring?

[Dependency Injection](http://www.javacodegeeks.com/2014/02/dependency-injection-options-for-java.html), an aspect of Inversion of Control (IoC), is a general concept, and it can be expressed in many different ways.This concept says that you do not create your objects but describe how they should be created. You don’t directly connect your components and services together in code but describe which services are needed by which components in a configuration file. A container (the IOC container) is then responsible for hooking it all up.

### 19. What are the different types of IoC (dependency injection)?

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

### 20. Which DI would you suggest Constructor-based or setter-based DI?

You can use both Constructor-based and Setter-based Dependency Injection. The best solution is using constructor arguments for mandatory dependencies and setters for optional dependencies.

# How to Use Profiles in Spring Boot

### Want to learn more about the features available in Spring Boot? Check out this post to learn more about how to use profiles in your Spring Boot application.

Spring Boot is gaining popularity like never before, and I know it will be a persistent player in the coming months. There are some features that every technology has and it is very useful in enterprise applications. In this post, we will talk about one feature in particular: profiles.

## **What Are Profiles?**

Every enterprise application has many environments, like:

Dev | Test | Stage | Prod | UAT / Pre-Prod

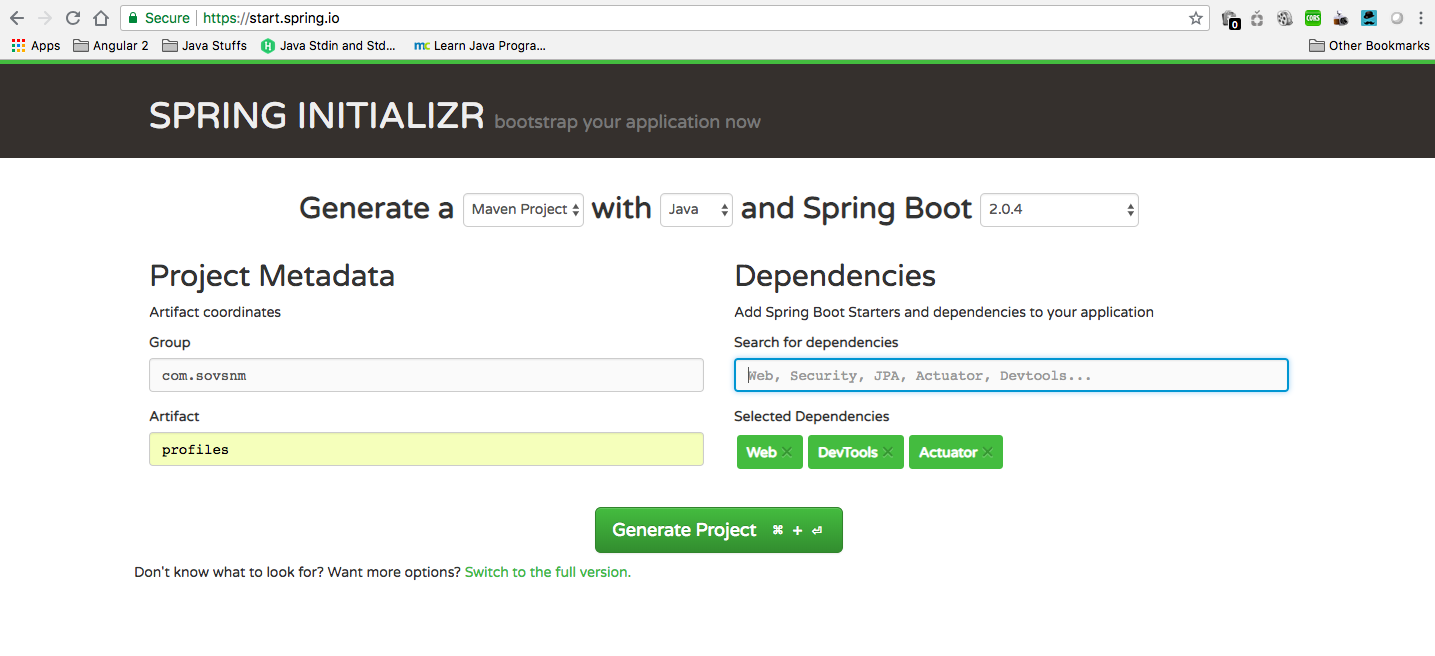
Each environment requires a setting that is specific to them. For example, in DEV, we do not need to constantly check database consistency. Whereas in TEST and STAGE, we need to. These environments host specific configurations called Profiles.

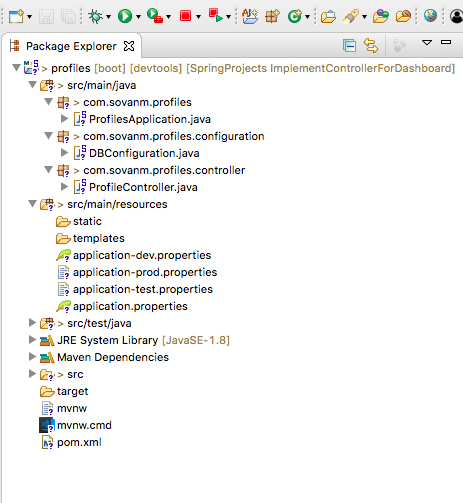
## **How Do we Maintain Profiles?**

This is simple — properties files!  
We make properties files for each environment and set the profile in the application accordingly, so it will pick the respective properties file. Don't worry, we will see how to set it up.

This article will demonstrate how to setup Profiles for your Spring Boot application.

Let's start with setting up a Spring Boot application from the [Spring Starter](https://start.spring.io/).



Next, we need to import the project into STS as a Maven Project. Below is the project structure:  


In this demo application, we will see how to configure different databases at runtime based on the specific environment by their respective profiles.

As the DB connection is better to be kept in a property file, it remains external to an application and can be changed. We will do so here. But, Spring Boot — by default — provides just one property file ( application.properties). So, how will we segregate the properties based on the environment?

The solution would be to create more property files and add the "profile" name as the suffix and configure Spring Boot to pick the appropriate properties based on the profile.

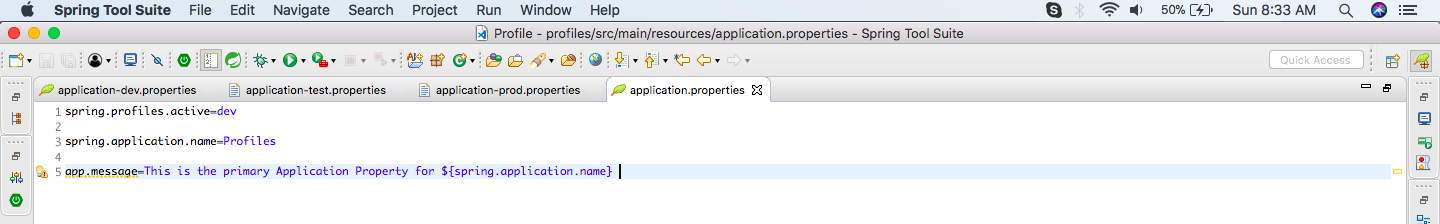
Then, we need to create three  application.properties:

1. application-dev.properties
2. application-test.properties
3. application-prod.properties

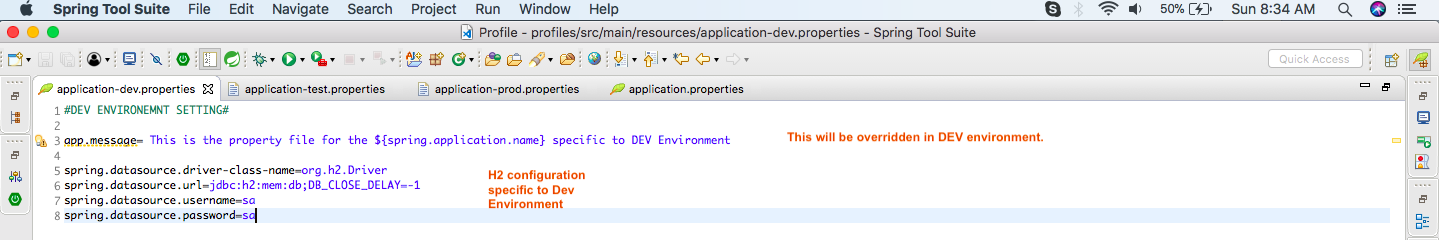
Of course, the application.properties will remain as a master properties file, but if we override any key in the profile-specific file, the latter will gain precedence.

I will now define DB configuration properties for in respective properties file and add code in DBConfiguration.class to pick the appropriate settings.

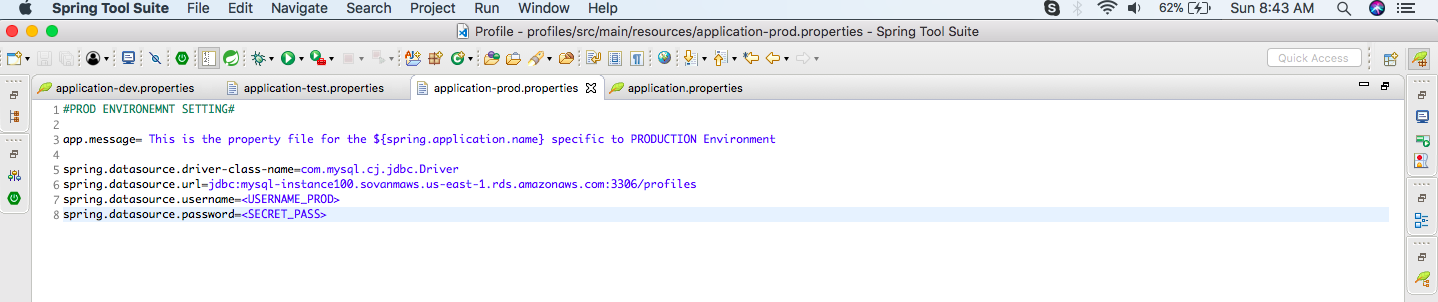
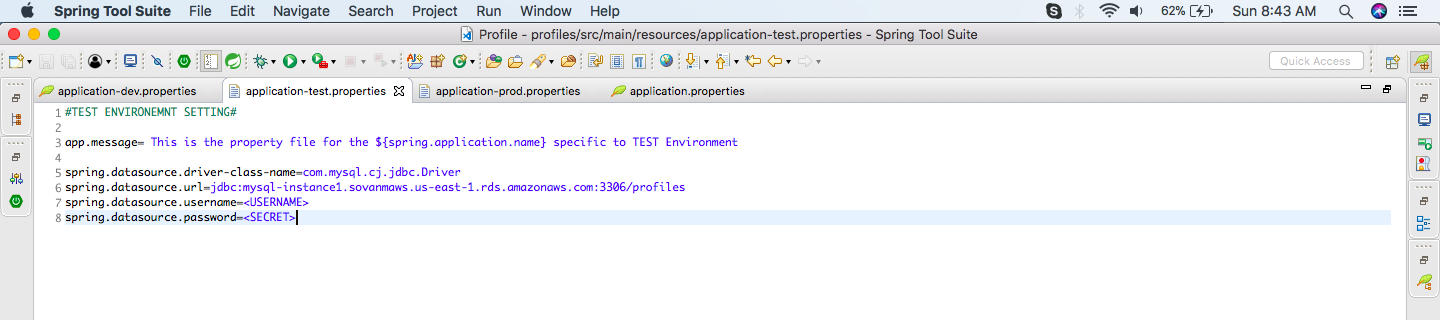
Here is the base  application.properties:



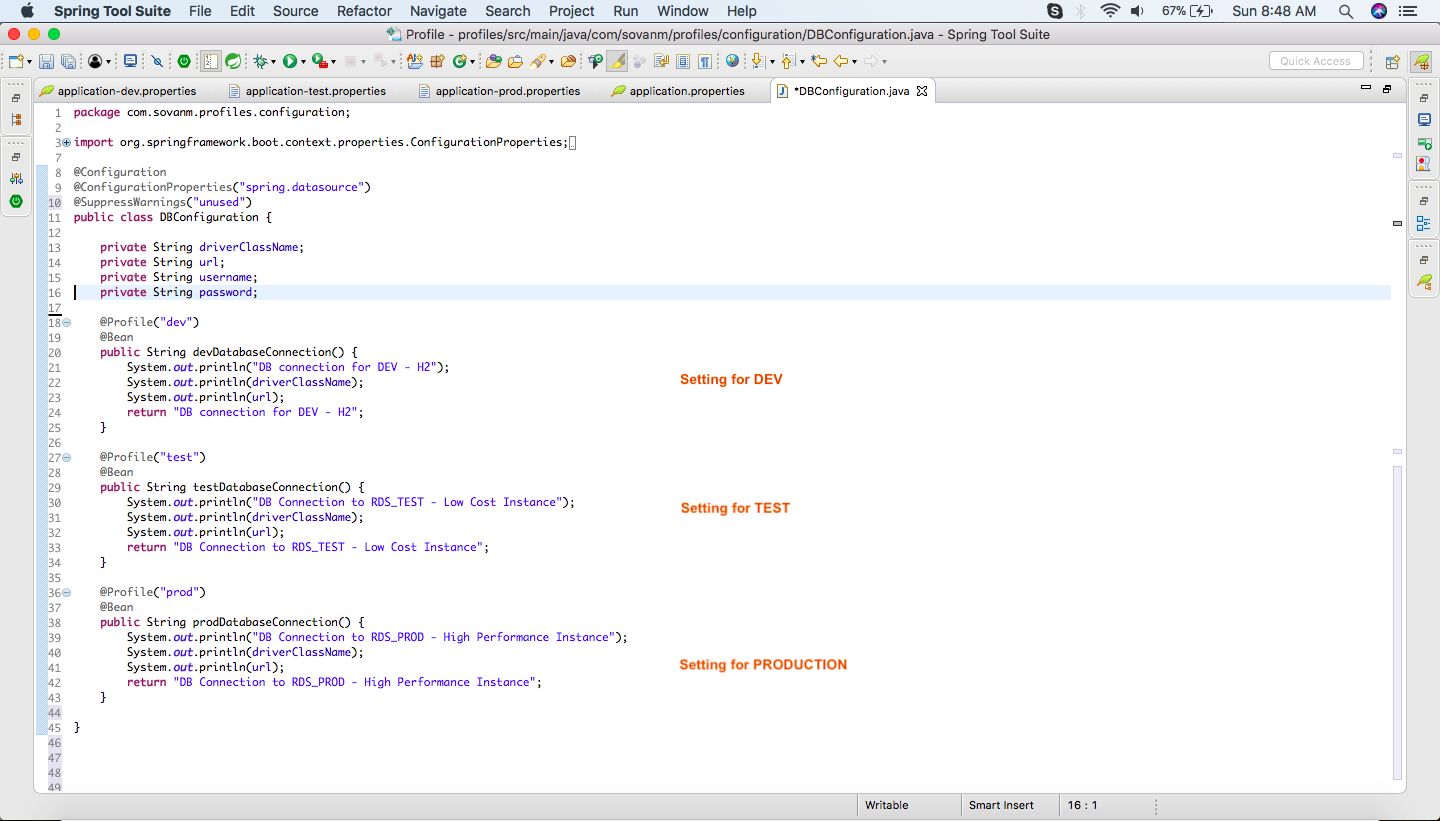
In DEV, we will use an in-memory database:



In TEST, we will be using a lower instance of RDS MySQL database, and in PROD, we will use a higher instance of the MySQL database. (It's the price that matters...)



Now, we are done with properties files. Let's configure in the DBConfiguration.class to pick the correct one.



We have used the @Profile("Dev")   to let the system know that this is the BEAN  that should be picked up when we set the application profile to DEV. The other two beans will not be created at all.

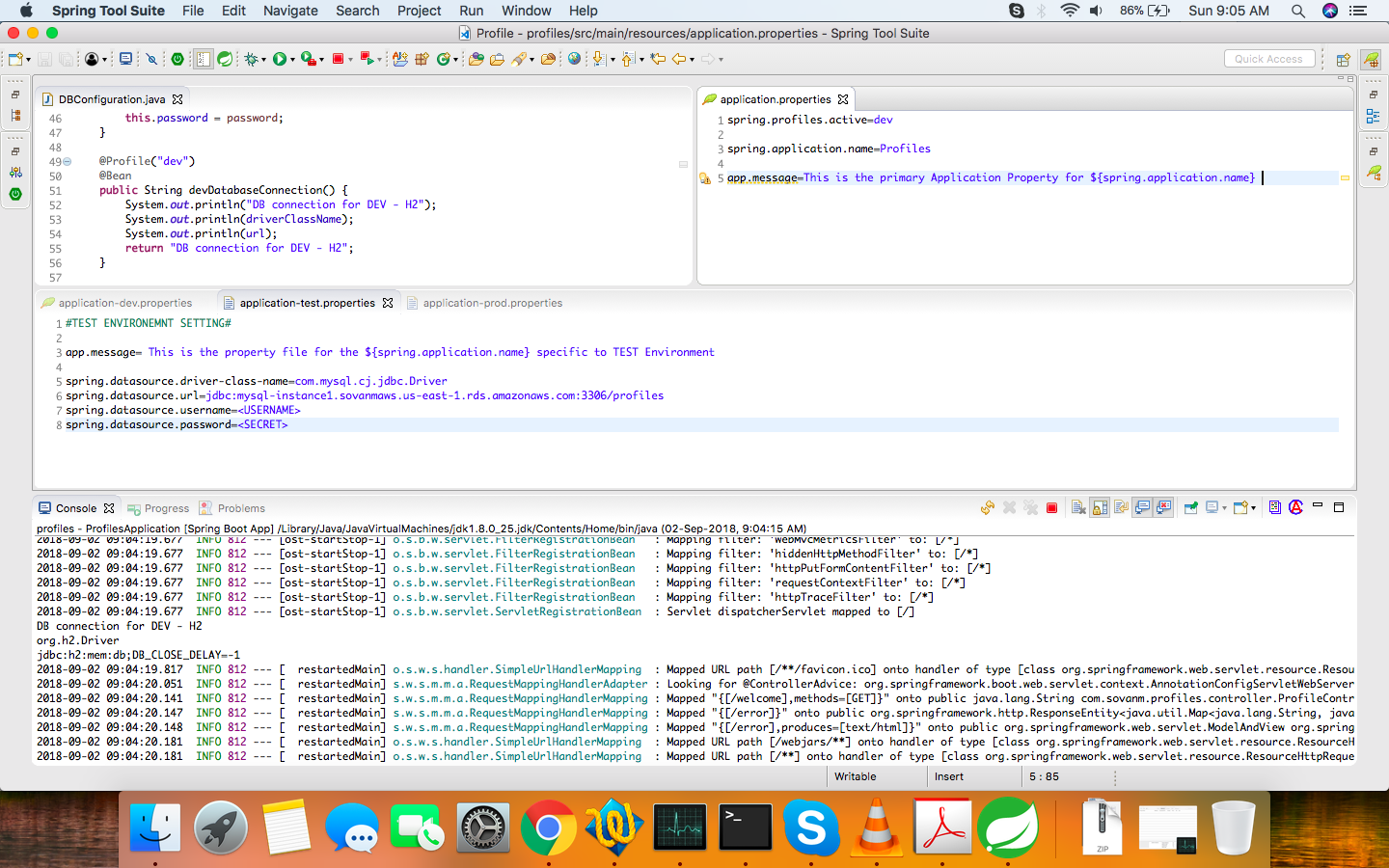
One last setting is how to let the system know that this is DEV, TEST, or PROD. But, how do we do this?

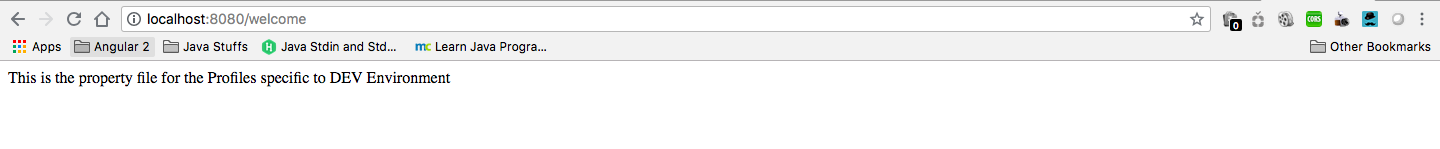
We will use the application.properties to use the key below:

spring.profiles.active=dev

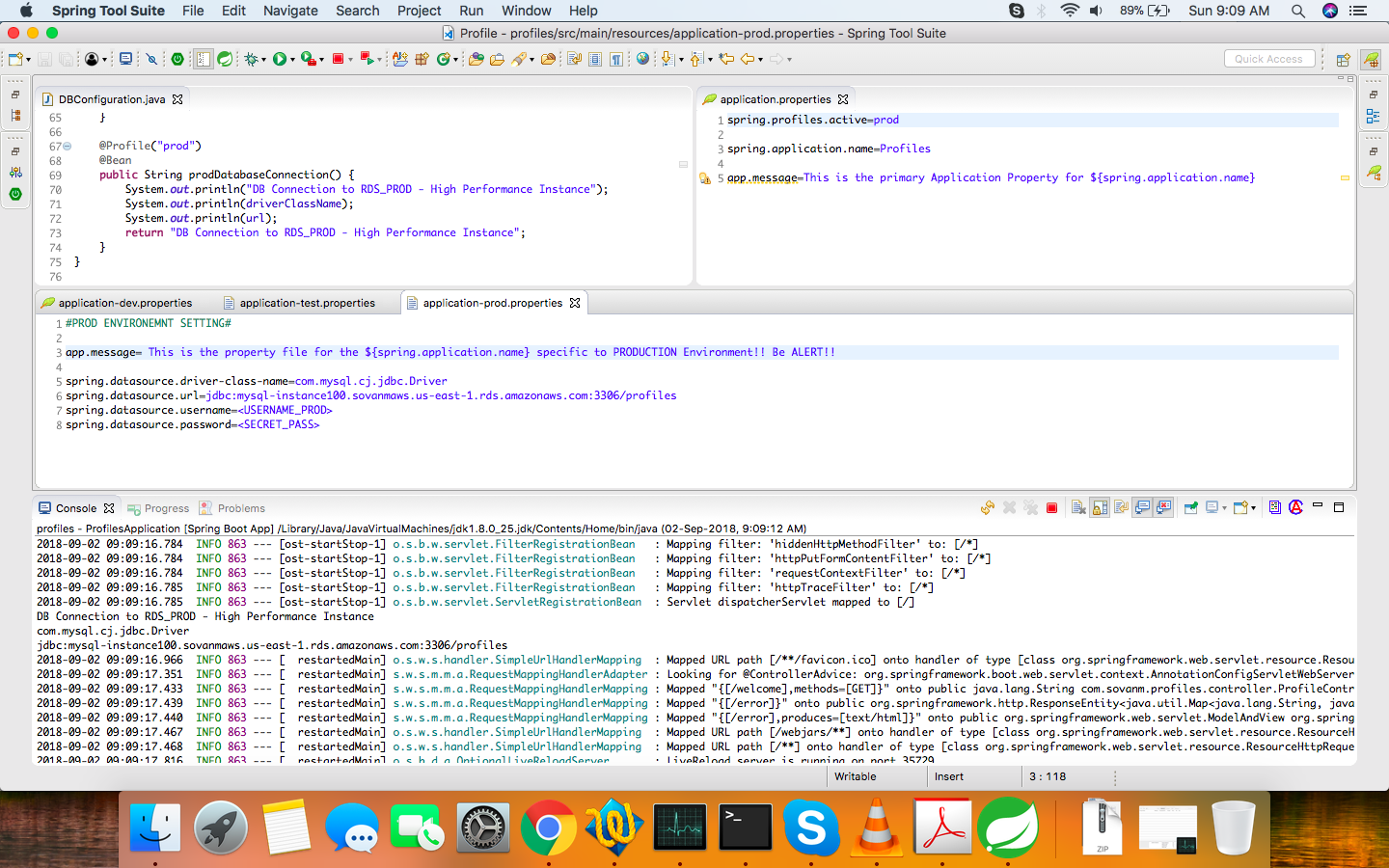
From here, Spring Boot will know which profile to pick. Let's run the application now!

With the profile in DEV mode, and it should pick H2 DB.





Now, change the profile to PROD. We will see MySQL with High Config for DB. This should be picked, and the message will be overridden with the PROD message.



That's it! We just have to change it once at the application.properties to let Spring Boot know which environment the code is deployed in, and it will do the magic with the setting.

# Spring vs. Spring Boot: A Comparison of These Java Frameworks

### Want to learn more about these two popular Java frameworks? Check out this look at Spring and Spring Boot and how they each solve a different type of problem.

What is Spring Boot? And, what is a Spring Framework? What are their goals? How can we compare them? There must be a lot of questions running through your mind. At the end of this blog, you will have the answers to all of these questions. In learning more about the Spring and Spring Boot frameworks, you will come to understand that each solve a different type of problem.

## **What Is Spring? What Are the Core Problems Spring Solves?**

The Spring Framework is one of the most popular application development frameworks for Java. One of the best features in Spring is that it has the **Dependency Injection (DI)**or **Inversion Of Control (IOC),**which allows us to develop loosely coupled applications. And, loosely coupled applications can be easily unit-tested.

### Example Without Dependency Injection

Consider the example below — MyController  depends on MyService  to perform a certain task. So, to get the instance of MyService, we will use:

MyService service = new MyService();

Now, we have created the instance for MyService , and we see both are tightly coupled. If I create a mock for MyService in a unit test for MyController , how do I make MyController use the mock? It's bit difficult — isn't it?

@RestController

public class MyController {

private MyService service = new MyService();

@RequestMapping("/welcome")

public String welcome() {

return service.retrieveWelcomeMessage();

}

}

### ****Example With a Dependency Injection****

With the help of only two annotations, we can get the instance of MyService easily, which is not tightly coupled. The Spring Framework does all the hard work to make things simpler.

* @Component is simply used in the Spring Framework as a bean that you need to manage within your own BeanFactory (an implementation of the Factory pattern).
* @Autowired is simply used to in the Spring Framework to find the correct match for this specific type and autowire it.

So, Spring framework will create a bean for MyService and autowire it into MyController.

In a unit test, I can ask the Spring Framework to auto-wire the mock of MyService into MyController .

@Component

public class MyService {

public String retrieveWelcomeMessage(){

return "Welcome to InnovationM";

}

}

@RestController

public class MyController {

@Autowired

private MyService service;

@RequestMapping("/welcome")

public String welcome() {

return service.retrieveWelcomeMessage();

}

}

The Spring Framework has many other features, which are divided into twenty modules to solve many common problems. Here are some of the more popular modules:

* Spring JDBC
* Spring MVC
* Spring AOP
* Spring ORM
* Spring JMS
* Spring Test
* Spring Expression Language (SpEL)

Aspect Oriented Programming(AOP) is another strong side of the Spring Framework. The key unit in object-oriented programming is the **class**, whereas, in AOP, the key unit is the **aspect**. For example, if you want to add the security in your project, logging, etc., you can just use the AOP and keep these as a cross-cutting concern away from your main business logic. You can perform any action after a method call, before a method call, after a method returns, or after the exception arises.

The Spring Framework does not have its own ORM, but it provides a very good integration with ORM, like Hibernate, Apache iBATIS, etc.

In short, we can say that the Spring Framework provides a decoupled way of developing web applications. Web application development becomes easy with the help of these concepts in Spring, like Dispatcher Servlet, ModelAndView, and View Resolver.

## **If Spring Can Solve so Many Problems, Why Do We Need Spring Boot?**

Now, if you have already worked on Spring, think about the problem that you faced while developing a full-fledged Spring application with all functionalities. Not able to come up with one? Let me tell you — there was lot of difficulty to setup Hibernate Datasource, Entity Manager, Session Factory, and Transaction Management. It takes a lot of time for a developer to set up a basic project using Spring MVC with minimum functionality.

<bean

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

<property name="prefix">

<value>/WEB-INF/views/</value>

</property>

<property name="suffix">

<value>.jsp</value>

</property>

</bean>

<mvc:resources mapping="/webjars/\*\*" location="/webjars/"/>

<servlet>

<servlet-name>dispatcher</servlet-name>

<servlet-class>

org.springframework.web.servlet.DispatcherServlet

</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/my-servlet.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>dispatcher</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

When we use Hibernate, we have to configure these things like a datasource, EntityManager, etc.

<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"

destroy-method="close">

<property name="driverClass" value="${db.driver}" />

<property name="jdbcUrl" value="${db.url}" />

<property name="user" value="${db.username}" />

<property name="password" value="${db.password}" />

</bean>

<jdbc:initialize-database data-source="dataSource">

<jdbc:script location="classpath:config/schema.sql" />

<jdbc:script location="classpath:config/data.sql" />

</jdbc:initialize-database>

<bean

class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean"

id="entityManagerFactory">

<property name="persistenceUnitName" value="hsql\_pu" />

<property name="dataSource" ref="dataSource" />

</bean>

<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">

<property name="entityManagerFactory" ref="entityManagerFactory" />

<property name="dataSource" ref="dataSource" />

</bean>

<tx:annotation-driven transaction-manager="transactionManager"/>

## **How Does Spring Boot Solve This Problem?**

1. Spring Boot does all of those using AutoConfiguration and will take care of all the internal dependencies that your application needs — all you need to do is run your application. Spring Boot will auto-configure with the Dispatcher Servlet, if Spring jar  is in the class path. It will auto-configue to the datasource, if Hibernate jar  is in the class path. Spring Boot gives us a pre-configured set of Starter Projects to be added as a dependency in our project.
2. During web-application development, we would need the jars that we want to use, which versions of the jars to use, and how to connect them together. All web applications have similar needs, for example, Spring MVC, Jackson Databind, Hibernate core, and Log4j (for logging). So, we had to choose the compatible versions of all these jars. In order to decrease the complexity, Spring Boot has introduced what we call **Spring Boot Starters.**

### Dependency for Spring Web Project

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>4.2.2.RELEASE</version>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

<version>2.5.3</version>

</dependency>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-validator</artifactId>

<version>5.0.2.Final</version>

</dependency>

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.17</version>

</dependency>

**Starters** are a set of convenient dependencies that you can include in your Spring Boot application.  For using Spring and Hibernate, we just have to include the spring-boot-starter-data-jpa dependency in the project.

### Dependency for Spring Boot Starter Web

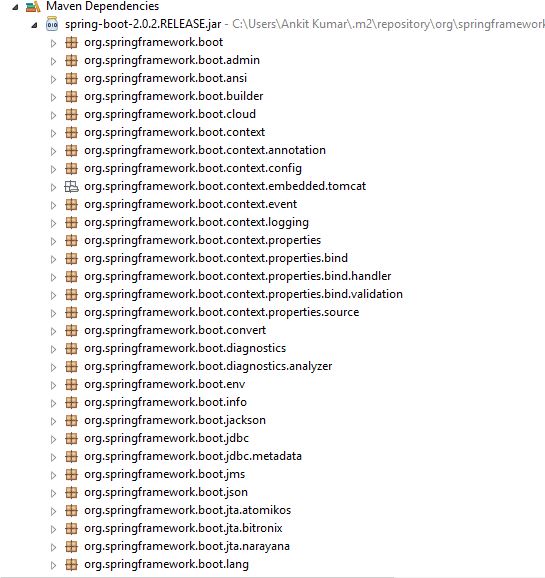
<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

The following screenshot shows the different packages under a single dependency that are added into our application:



There are other packages that you will see. Once you add that starter dependency, the Spring Boot Starter Web comes pre-packaged with all of these. As a developer, we would not need to worry about these dependencies and their compatible versions.

## Spring Boot Starter Project Options

These are few starter projects to help us get started quickly with developing specific types of applications.

* spring-boot-starter-web-services: SOAP Web Services
* spring-boot-starter-web: Web and RESTful applications
* spring-boot-starter-test: Unit testing and Integration Testing
* spring-boot-starter-jdbc: Traditional JDBC
* spring-boot-starter-hateoas: Add HATEOAS features to your services
* spring-boot-starter-security: Authentication and Authorization using Spring Security
* spring-boot-starter-data-jpa: Spring Data JPA with Hibernate
* spring-boot-starter-cache: Enabling Spring Framework’s caching support
* spring-boot-starter-data-rest: Expose Simple REST Services using Spring Data REST

## **Spring Beans**

### 21. What are Spring beans?

The [Spring Beans](http://examples.javacodegeeks.com/enterprise-java/spring/beans-spring/spring-3-bean-reference-example/) are Java Objects that form the backbone of a Spring application. They are instantiated, assembled, and managed by the Spring IoC container. These beans are created with the configuration metadata that is supplied to the container, for example, in the form of XML <bean/> definitions.

Beans defined in spring framework are singleton beans. There is an attribute in bean tag named "singleton" if specified true then bean becomes singleton and if set to false then the bean becomes a prototype bean. By default it is set to true. So, all the beans in spring framework are by default singleton beans.

### 22. What does a Spring Bean definition contain?

A Spring Bean definition contains all configuration metadata which is needed for the container to know how to create a bean, its lifecycle details and its dependencies.

### 23. How do you provide configuration metadata to the Spring Container?

There are three important methods to provide configuration metadata to the Spring Container:

* XML based configuration file.
* Annotation-based configuration
* [Java-based configuration](http://examples.javacodegeeks.com/enterprise-java/spring/beans-spring/spring-3-java-config-example/)

### 24. How do you define the scope of a bean?

When defining a <bean> in Spring, we can also declare a scope for the bean. It can be defined through the scope attribute in the bean definition. For example, when Spring has to produce a new bean instance each time one is needed, the bean’s scope attribute to be prototype. On the other hand, when the same instance of a bean must be returned by Spring every time it is needed, the the bean scope attribute must be set to singleton.

### 25. Explain the bean scopes supported by Spring

There are five scoped provided by the Spring Framework supports following five scopes:

* In **singleton** scope, Spring scopes the bean definition to a single instance per Spring IoC container.
* In **prototype** scope, a single bean definition has any number of object instances.
* In **request** scope, a bean is defined to an HTTP request. This scope is valid only in a web-aware Spring ApplicationContext.
* In **session**scope, a bean definition is scoped to an HTTP session. This scope is also valid only in a web-aware Spring ApplicationContext.
* In **global-session** scope, a bean definition is scoped to a global HTTP session. This is also a case used in a web-aware Spring ApplicationContext.

The default scope of a Spring Bean is Singleton.

### 26. Are Singleton beans thread safe in Spring Framework?

No, singleton beans are not thread-safe in Spring framework.

### 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 autowire by byType.

### 34. Are there limitations with autowiring?

Limitations of autowiring are:

* **Overriding:**You can still specify dependencies using <constructor-arg> and <property> settings which will always override autowiring.
* **Primitive data types:** You cannot autowire simple properties such as primitives, Strings, and Classes.
* **Confusing nature:** Autowiring is less exact than explicit wiring, so if possible prefer using explicit wiring.

### 35. Can you inject null and empty string values in Spring?

Yes, you can.

## **Spring Annotations**

### 36. What is Spring Java-Based Configuration? Give some annotation example.

[Java based configuration](http://www.javacodegeeks.com/2013/04/spring-java-configuration.html) option enables you to write most of your Spring configuration without XML but with the help of few Java-based annotations.  
An example is the @Configuration annotation, that indicates that the class can be used by the Spring IoC container as a source of bean definitions. Another example is the@Bean annotated method that will return an object that should be registered as a bean in the Spring application context.

### 37. What is Annotation-based container configuration?

An alternative to XML setups is provided by annotation-based configuration which relies on the bytecode metadata for wiring up components instead of angle-bracket declarations. Instead of using XML to describe a bean wiring, the developer moves the configuration into the component class itself by using annotations on the relevant class, method, or field declaration.

### 38. How do you turn on annotation wiring?

Annotation wiring is not turned on in the Spring container by default. In order to use annotation based wiring we must enable it in our Spring configuration file by configuring <context:annotation-config/> element.

### 39. @Required annotation

This annotation simply indicates that the affected bean property must be populated at configuration time, through an explicit property value in a bean definition or through autowiring. The container throws BeanInitializationException if the affected bean property has not been populated.

### 40. @Autowired annotation

The @Autowired annotation provides more fine-grained control over where and how autowiring should be accomplished. It can be used to autowire bean on the setter method just like @Required annotation, on the constructor, on a property or pn methods with arbitrary names and/or multiple arguments.

### 41. @Qualifier annotation

When there are more than one beans of the same type and only one is needed to be wired with a property, the @Qualifierannotation is used along with @Autowired annotation to remove the confusion by specifying which exact bean will be wired.

## **Spring Data Access**

### 42. How can JDBC be used more efficiently in the Spring framework?

When using the Spring JDBC framework the burden of resource management and error handling is reduced. So developers only need to write the statements and queries to get the data to and from the database. JDBC can be used more efficiently with the help of a template class provided by Spring framework, which is the JdbcTemplate (example [here](http://examples.javacodegeeks.com/enterprise-java/spring/jdbc/spring-jdbctemplate-example/)).

### 43. JdbcTemplate

JdbcTemplate class provides many convenience methods for doing things such as converting database data into primitives or objects, executing prepared and callable statements, and providing custom database error handling.

### 44. Spring DAO support

The [Data Access Object (DAO) support in Spring](http://www.javacodegeeks.com/2012/09/spring-dao-and-service-layer.html) is aimed at making it easy to work with data access technologies like JDBC, Hibernate or JDO in a consistent way. This allows us to switch between the persistence technologies fairly easily and to code without worrying about catching exceptions that are specific to each technology.

### 45. What are the ways to access Hibernate by using Spring?

There are two ways to access Hibernate with Spring:

* Inversion of Control with a Hibernate Template and Callback.
* Extending HibernateDAOSupport and Applying an AOP Interceptor node.

### 46. ORM’s Spring support

Spring supports the following ORM’s:

* Hibernate
* iBatis
* JPA (Java Persistence API)
* TopLink
* JDO (Java Data Objects)
* OJB

### 47. How can we integrate Spring and Hibernate using HibernateDaoSupport?

Use Spring’s SessionFactory called LocalSessionFactory. The integration process is of 3 steps:

* Configure the Hibernate SessionFactory
* Extend a DAO Implementation from HibernateDaoSupport
* Wire in Transaction Support with AOP

### 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 Aspect Oriented Programming (AOP)**

### 51. Explain AOP

[Aspect-oriented programming](http://www.javacodegeeks.com/2014/02/applying-aspect-oriented-programming.html), or AOP, is a programming technique that allows programmers to modularize crosscutting concerns, or behavior that cuts across the typical divisions of responsibility, such as logging and transaction management.

### 52. Aspect

The core construct of AOP is the aspect, which encapsulates behaviors affecting multiple classes into reusable modules. It ia a module which has a set of APIs providing cross-cutting requirements. For example, a logging module would be called AOP aspect for logging. An application can have any number of aspects depending on the requirement. In Spring AOP, aspects are implemented using regular classes annotated with the @Aspect annotation (@AspectJ style).

### 53. What is the difference between concern and cross-cutting concern in Spring AOP

The Concern is behavior we want to have in a module of an application. A Concern may be defined as a functionality we want to implement.  
The cross-cutting concern is a concern which is applicable throughout the application and it affects the entire application. For example, logging, [security](http://www.javacodegeeks.com/2013/04/spring-aop-in-security-controlling-creation-of-ui-components-via-aspects.html) and data transfer are the concerns which are needed in almost every module of an application, hence they are cross-cutting concerns.

### 54. Join point

The join point represents a point in an application where we can plug-in an AOP aspect. It is the actual place in the application where an action will be taken using Spring AOP framework.

### 55. Advice

The advice is the actual action that will be taken either before or after the method execution. This is actual piece of code that is invoked during the program execution by the Spring AOP framework.

Spring aspects can work with five kinds of advice:

* **before:** Run advice before the a method execution.
* **after:** Run advice after the a method execution regardless of its outcome.
* **after-returning:** Run advice after the a method execution only if method completes successfully.
* **after-throwing:** Run advice after the a method execution only if method exits by throwing an exception.
* **around:** Run advice before and after the advised method is invoked.

### 56. Pointcut

The pointcut is a set of one or more joinpoints where an advice should be executed. You can specify pointcuts using expressions or patterns.

### 57. What is Introduction?

An Introduction allows us to add new methods or attributes to existing classes.

### 58. What is Target object?

The target object is an object being advised by one or more aspects. It will always be a proxy object. It is also referred to as the advised object.

### 59. What is a Proxy?

A proxy is an object that is created after applying advice to a target object. When you think of client objects the target object and the proxy object are the same.

### 60. What are the different types of AutoProxying?

* BeanNameAutoProxyCreator
* DefaultAdvisorAutoProxyCreator
* Metadata autoproxying

### 61. What is Weaving? What are the different points where weaving can be applied?

Weaving is the process of linking aspects with other application types or objects to create an advised object.  
Weaving can be done at compile time, at load time, or at runtime.

### 62. Explain XML Schema-based aspect implementation?

In this implementation case, aspects are implemented using regular classes along with XML based configuration.

### 63. Explain annotation-based (@AspectJ based) aspect implementation

This implementation case (@AspectJ based implementation) refers to a style of declaring aspects as regular Java classes annotated with Java 5 annotations.

## **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.

Ok, so now you are ready for your interview! Don’t forget to check our dedicated page full of [Spring Tutorials](http://www.javacodegeeks.com/tutorials/java-tutorials/enterprise-java-tutorials/spring-tutorials/), and our [Examples dedicated subsection](https://examples.javacodegeeks.com/category/enterprise-java/spring/)!

If you enjoyed this, then [**subscribe to our newsletter**](http://eepurl.com/xRlkj) to enjoy weekly updates and complimentary whitepapers! Also, check out our [**courses**](https://www.javacodegeeks.com/courses/) for more advanced training!

**Question1: What is IOC or inversion of control?**([answer](http://javarevisited.blogspot.com/2012/12/inversion-of-control-dependency-injection-design-pattern-spring-example-tutorial.html))

Answer: This *Spring interview question* is the first step towards Spring framework and many interviewers start Spring interview from this question. As the name implies **Inversion of control** means now we have inverted the control of creating the object from our own using new operator to container or framework. Now it’s the responsibility of container to create an object as required. We maintain one XML file where we configure our components, services, all the classes and their property. We just need to mention which service is needed by which component and container will create the object for us. This concept is known as [dependency injection](http://javarevisited.blogspot.com/2013/06/spring-helloworld-example-in-java-using-3.0-dependency-injection.html) because all object dependency (resources) is injected into it by the framework.

Example:

  <bean id="createNewStock" class="springexample.stockMarket.CreateNewStockAccont">   
        <property name="newBid"/>

  </bean>

In this example, CreateNewStockAccont class contain getter and setter for newBid and container will instantiate newBid and set the value automatically when it is used. This whole process is also called wiring in Spring and by using annotation it can be done automatically by Spring, referred as auto-wiring of bean in Spring.

**Question 2: Explain the Spring Bean-LifeCycle.**

Ans: Spring framework is based on IOC so we call it as IOC container also So Spring beans reside inside the IOC container. Spring beans are nothing but Plain old java object (POJO).

Following steps explain their life cycle inside the container.

1. The container will look the bean definition inside configuration file (e.g. bean.xml).

2 using reflection container will create the object and if any property is defined inside the bean definition then it will also be set.

3. If the bean implements the BeanNameAware interface, the factory calls setBeanName() passing the bean’s ID.  
4. If the bean implements the BeanFactoryAware interface, the factory calls setBeanFactory(), passing an instance of itself.  
5. If there are any BeanPostProcessors associated with the bean, their post- ProcessBeforeInitialization() methods will be called before the properties for the Bean are set.

6. If an init() method is specified for the bean, it will be called.  
7. If the Bean class implements the DisposableBean interface, then the method destroy() will be called when the Application no longer needs the bean reference.

8. If the Bean definition in the Configuration file contains a 'destroy-method' attribute, then the corresponding method definition in the Bean class will be called.  
To learn more about spring beans and their life cycle I also suggest you take a look at these [Spring Framework books](http://javarevisited.blogspot.com/2013/03/5-good-books-to-learn-spring-framework-mvc-java-programmer.html).

**Question 3: what is Bean Factory, have you used XMLBeanFactory?**

Ans: BeanFactory is factory Pattern which is based on IOC [design principles](http://javarevisited.blogspot.sg/2012/03/10-object-oriented-design-principles.html).it is used to make a clear separation between application configuration and dependency from actual code. The XmlBeanFactory is one of the implementations of Bean Factory which we have used in our project. The**org.springframework.beans.factory.xml.XmlBeanFactory is used to create bean instance defined in our XML file.**

BeanFactory factory = new XmlBeanFactory(new FileInputStream("beans.xml"));

Or

ClassPathResource resorce = new ClassPathResource("beans.xml");   
XmlBeanFactory factory = new XmlBeanFactory(resorce);

**Question 4: What are the difference between BeanFactory and ApplicationContext in Spring?**([answer](http://javarevisited.blogspot.com/2012/11/difference-between-beanfactory-vs-applicationcontext-spring-framework.html))

Answer: This one is very popular Spring interview question and often asks in entry level interview. ApplicationContext is the preferred way of using spring because of functionality provided by it and interviewer wanted to check whether you are familiar with it or not.

|  |  |
| --- | --- |
| **ApplicationContext.** | **BeanFactory** |
| Here we can have more than one config files possible | In this only one config file or .xml file |
| Application contexts can publish events to beans that are registered as listeners | Don't support. |
| Support internationalization (I18N) messages | It’s not |
| Support application life-cycle events, and validation. | Doesn’t support. |
| Supports many enterprise services such JNDI access, EJB integration, remoting | Doesn’t support. |

**Question 5: What are different modules in spring?**

Answer: spring has seven core modules

1.      The Core container module

2.      Application context module

3.      AOP module (Aspect Oriented Programming)

4.      JDBC abstraction and DAO module

5.      O/R mapping integration module (Object/Relational)

6.      Web module

7.      MVC framework module

**Question 6: What is the difference between singleton and prototype bean?**

[](http://3.bp.blogspot.com/-f39tbNy0pO0/To2-Q8flynI/AAAAAAAAAQY/il_T_LzytMA/s1600/Spring_Interview_Questions_252x150.png)

Ans: This is another popular *spring interview questions* and an important concept to understand. Basically, a bean has scopes which define their existence on the application

**Singleton:** means single bean definition to a single object instance per Spring IOC container.  
**Prototype**: means a single bean definition to any number of object instances.

Whatever beans we defined in spring framework are singleton beans. There is an attribute in bean tag named ‘singleton’ if specified true then bean becomes singleton and if set to false then the bean becomes a prototype bean. By default, it is set to true. So, all the beans in spring framework are by default singleton beans.

  <bean id="createNewStock"     class="springexample.stockMarket.CreateNewStockAccont" **singleton=”false”**>   
        <property name="newBid"/>   
  </bean>

**Question 7: What type of transaction Management Spring support?**

Ans: This spring interview question is little difficult as compared to previous questions just because **transaction management** is a complex concept and not every developer familiar with it. Transaction management is critical in any applications that will interact with the database. The application has to ensure that the data is consistent and the integrity of the data is maintained.  Following two type of transaction management is supported by spring:

1. Programmatic transaction management

2. Declarative transaction management.

**Question 8: What is AOP?**

Answer: The core construct of AOP is the aspect, which encapsulates behaviours affecting multiple classes into reusable modules. AOP is a programming technique that allows a developer to modularize crosscutting concerns, that cuts across the typical divisions of responsibility, such as **logging and transaction management.**Spring AOP, aspects are implemented using regular classes or regular classes annotated with the @Aspect annotation. You can also check out these [Spring MVC interview questions](http://java67.blogspot.com/2012/08/spring-interview-questions-answers.html) for more focus on Java web development using Spring framework.

**Question 9: Explain Advice?**

Answer: It’s an implementation of aspect; advice is inserted into an application at join points. Different types of advice include “around,” “before” and “after” advice

**Question 10: What are joint Point and point cut?**

Ans: This is not really a spring interview questions I would say an AOP one.  Similar to [Object oriented programming](http://javarevisited.blogspot.sg/2012/03/10-object-oriented-design-principles.html), AOP is another popular programming concept which complements OOPS. A join point is an opportunity within the code for which we can apply an aspect. In Spring AOP, a join point always represents a method execution.

**Pointcut**: a predicate that matches join points. A point cut is something that defines at what join-points an advice should be applied.  
  
Here are few more Spring fundamental interview questions for practice  
  
Question 11: Difference between the setter and constructor injection in Spring? ([answer](http://javarevisited.blogspot.com/2012/11/difference-between-setter-injection-vs-constructor-injection-spring-framework.html))  
Question 12: How to implement Role Base Access Control (RBAC) using Spring Security? ([answer](http://javarevisited.blogspot.com/2013/07/role-based-access-control-using-spring-security-ldap-authorities-mapping-mvc.html))  
Question 13: How to call the stored procedure from Java using Spring Framework? ([answer](http://javarevisited.blogspot.com/2013/04/spring-framework-tutorial-call-stored-procedures-from-java.html))  
Question 14: How to Setup JDBC Database connection pool in Spring Web application? ([answer](http://javarevisited.blogspot.com/2012/06/jdbc-database-connection-pool-in-spring.html))  
Question 15: Difference between Factory Pattern and Dependency Injection in Java? ([answer](http://javarevisited.blogspot.com/2015/06/difference-between-dependency-injection.html))

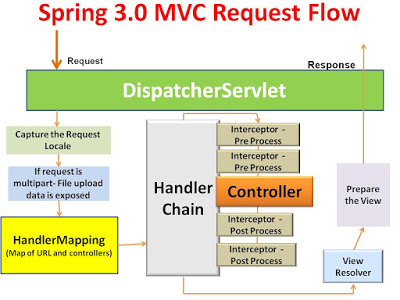
These **spring interview Questions and answers**are not very difficult and focused on spring fundamentals rather than focusing on an advanced feature of session management, spring security, authentication etc. we will cover of those question on some other interview article. I would also suggest that share some spring questions asked to you guys during the interview and then I can put together those with their answers for quick reference of everybody.

**Further Reading**  
Expert Spring MVC and Web Flow ([read here](http://aax-us-east.amazon-adsystem.com/x/c/QkMXgQWl_bjzxPorY45rqW4AAAFd78Q9jgEAAAFKAUWNNoo/https:/assoc-redirect.amazon.com/g/r/http:/www.amazon.com/Expert-Spring-Flow-Experts-Voice/dp/159059584X/ref=as_at?creativeASIN=159059584X&linkCode=w61&imprToken=tB5T6CTV-48piVLrLaqYXg&slotNum=2&tag=javamysqlanta-20))  
Spring Fundamentals By Bryan Hansen ([see](http://www.shareasale.com/m-pr.cfm?merchantID=53701&userID=880419&productID=546412741))  
Introduction to Spring MVC 4 By Bryan Hansen ([see](http://www.shareasale.com/m-pr.cfm?merchantID=53701&userID=880419&productID=557072989))  
Hibernate Interview Questions and answers ([see here](http://java67.blogspot.com/2016/02/top-20-hibernate-interview-questions.html))

P.S. - If you are an experienced Java/JEE Program and want to learn Spring Security end-to-end, I recommend [Learn Spring Security](http://www.baeldung.com/learn-spring-security-course?utm_source=javarevisited&utm_medium=web&utm_campaign=lss&affcode=22136_bkwjs9xa) course by Eugen Paraschiv, The definitive guide to secure your Java application. It's useful for both junior and experienced Java Web developers.  
  
He is also the author of [REST with Spring](http://www.baeldung.com/rest-with-spring-course?utm_source=javarevisited&utm_medium=web&utm_campaign=rws&affcode=22136_bkwjs9xa) course, one of the best online course to learn RESTful WebServices using Spring framework.

**1) What is spring framework? Why Java programmer should use Spring framework**  
Very common Spring interview question, Spring is a framework which helps Java programmer in development. Spring provides dependency Injection and IOC container, Spring MVC flow and several useful API for Java programmer.

**2) What is default scope of bean in Spring framework ?**  
The default scope of a Spring bean is [Singleton](http://javarevisited.blogspot.sg/2011/03/10-interview-questions-on-singleton.html) scope, you can read this article which explains about all possible scope of a spring bean : [What is bean scope in Spring](http://javarevisited.blogspot.sg/2012/05/what-is-bean-scope-in-spring-mvc.html)  
  
  
**3) Does Spring singleton beans are thread-safe ?**  
No, Spring singleton beans are not thread-safe. Singleton doesn't mean bean would be[thread-safe](http://javarevisited.blogspot.sg/2012/01/how-to-write-thread-safe-code-in-java.html).  
  
  
**4) What is dependency Injection?**  
Dependency Injection is one of the design pattern, which allows injecting dependency on Object, instead of object resolving the dependency.  
  
5) What is Inversion of Control concept, how does Spring support IOC?  
  
6) What is Spring MVC ? Can you explain How one request is processed ?  
  
7) How to you create a controller in Spring ?  
  
  
**8) What is view Resolver pattern ? how it work in Spring MVC**  
View Resolver pattern is a J2EE pattern which allows a web application to dynamically choose it's view technology e.g. HTML, JSP, Tapestry, JSF, XSLT or any other view technology. In this pattern, View resolver holds mapping of different views, controller return name of the view, which is then passed to View Resolver for selecting an appropriate view.Spring MVC framework supplies inbuilt view resolver for selecting views.  
  
  
**9) What is Spring Security ?**  
Spring security is a project under spring framework umbrella, which provides support for security requirements of enterprise Java projects. Spring Security formerly known as aegis security provides out of box support for creating login screen, remember me cookie support, securing URL, authentication provider to authenticate the user from the database, LDAP and in memory, concurrent active session management support and much more. In order to use Spring security in a Spring MVC based project, you need to include spring-security.jar and configure it in application-Context-security.XML file, you can name it whatever you want, but make sure to supply this to ContextLoaderListener, which is responsible for creating Spring context and initializing dispatcher servlet. You can see [Pro Spring Security](http://aax-us-east.amazon-adsystem.com/x/c/QtzWM5Ktm7_KkIXxbfbbJqYAAAFd78WP4AEAAAFKAcMYsfc/https:/www.amazon.com/Pro-Spring-Security-Carlo-Scarioni/dp/1430248181/ref=as_at?creativeASIN=1430248181&linkCode=w61&imprToken=26Cp9Ih4jNmbrnLMhzoC2A&slotNum=0&tag=javamysqlanta-20) by Carlo Scarioni to learn more about Spring Security.  
**10) How do you control concurrent Session on Java web application using Spring Security?**  
You can use Spring Security to control a number of active session in Java web application. Spring security framework provides this feature out of the box and when enabled , a user can only have one active session at a time. See this Spring Security example to learn more about [How to control concurrent user session using Spring security](http://javarevisited.blogspot.sg/2012/03/spring-security-example-tutorial-how-to.html)  
  
 **11) What types of dependency injection is supported by Spring Framework? When do you use Setter and Constructor Injection, pros and cons?**  
There are 2 types of dependency injection supported by Spring, constructor based injection, and setter-based injection. Both types have their own advantages and disadvantages, you should use Constructor injection when object's dependencies are not optional and they must be initialized with their dependencies. Also use constructor injection if the order of initialization or dependency matters because in Setter based injection you cannot impose any order. Use setter injection when dependencies are optional. See the [difference between setter and constructor injection in Spring](http://javarevisited.blogspot.sg/2012/11/difference-between-setter-injection-vs-constructor-injection-spring-framework.html) for more detailed answer.  
  
  
12) What is the difference between ApplicationContext and BeanFactory in Spring framework?  
  
13) How do you call stored procedure by using Spring framework?  
  
14) What does JdbcTemplate and JmsTemplate class offer in Spring?  
  
15) Can we use more than one configuration file for our Spring project?  
  
16) Explain Spring MVC flow with a simple example e.g. starting from Container receives a request and forward to your Java application ?  
  
17) What is the difference in Spring MVC and Spring core?  
  
18) Can you use Spring MVC framework along with Struts ? I have an existing Java MVC application which is based in Struts, Can I migrate that to use Spring MVC ? How ?  
  
19) What is the advantage of Spring MVC framework over Struts 1.0 or Struts 2.0 ? is it worth to convert an existing Struts application to Spring MVC ?  
  
20) How do Spring resolves view returned by ModelAndView class ?  
  
Some Spring MVC questions are tricky e.g. Struts and Spring integration and can be only answered by experienced Java program with 2 to 4-year experience in Spring MVC framework.  
  
  
**21) If a user checked in CheckBox and got a validation error in other fields and then he unchecked the CheckBox, what would be selection status in command object in Spring MVC ? How do you fix this issue?**  
Since during HTTP post, if the checkbox is unchecked than HTTP does include a request parameter for checkbox, which means updated selection won't be picked up. you can use hidden form field, starting with \_ to fix this in Spring MVC. quite a tricky question to answer if you are not aware of HTTP POST behavior and Spring MVC.

[](https://4.bp.blogspot.com/-UmPMU39vWFM/V6MyIuNHgiI/AAAAAAAAGw4/hoNHnOFOwZkLIBB19RVW_JvNS9KeUIRlQCLcB/s1600/Spring-3-MVC-Request-Flow.jpg)

**22) What are different implementations of View interface you have used in Spring MVC?**  
ULBased View e.g. JSP , JSTLView,  
  
  
**23) How to escape HTML special characters using Spring MVC?**  
There are some methods in Spring tag library, can't remember now.  
  
  
These were some of the **Core Spring framework and MVC Interview questions** from my collection, I have given short answers for most of these Spring interview question. I suggest to research more or read along those Spring question to prepare for follow-up Spring interview questions.  
  
Read more: <http://www.java67.com/2012/08/spring-interview-questions-answers.html#ixzz4q0XxGN18>

|  |
| --- |
| **What is Aspect Oriented Programming (AOP)?** - Basically Aspect oriented programming complements object oriented programming by providing another way of programming model structure. - In addition to classes, AOP gives you aspect, which enables modularization of concerns such as Transaction management or logging and can be separated out from the application business logic of the code (these kinds of concerns are termed as crosscutting concerns). AOP supports separating application business logic from System services. **What is IOC or Dependency Injection?** - The basic concept of IOC (Dependency of Injection) is that you do not create your objects but describe how they should be created. - You don’t directly connect your component and services together in code but describe which services are needed by which component in configuration file. - You just need to describe the dependency, the Spring container is then responsible for hooking it all up. |
| **When to use Dependency Injections?** There are different scenarios where you Dependency Injections are useful.  - You need to inject configuration data into one or more component. - You need to inject the same dependency into multiple components. - You need to inject different implementation of the same dependency. - You need to inject the same implementation in different configuration. - You need some of the services provided by container. **When you should not use Dependency Injection?** There were scenarios where you don’t need dependency injections e.g. - You will never need a different implementation. - You will never need different configurations. |
| **What is Bean Factory in Spring?** - A Bean Factory is like a factory class that contains collections of beans. The Bean Factory holds bean definition of multiple beans within itself and then instantiates the bean when asked by client. - Bean Factory is actual representation of the Spring IOC container that is responsible for containing and managing the configured beans. **Different Spring Bean Scope.** **1. singleton :** Return a single bean instance per Spring IOC container. **2. prototype :** Return a new bean instance each time when requested. **3. request :** Return a single bean instance per HTTP request. **4. session :** Return a single bean instance per HTTP session. **5. global session :** Return a single bean instance per global HTTP session and only valid when used in portlet context. **How you will decide when to use prototype scope and when singleton scope bean?** - You should use the prototype scope for all beans that are stateful and the singleton scope should be used for stateless beans. |
| **What are the different types of IOC?** There are three types of dependency Injection:  **1. Constructor Injection :**   - Dependencies are provided as a constructor parameter.  - **Example :** You want to inject some object say 'Foo' through constructor in class 'HellowWorld' like below.  public class HelloWorld { public HelloWorld(Foo foo)    {        this.foo = foo;    }  }  - In configuration file you need to do following entry.  <bean id="helloWorldBean" class="com.xyz.services.HelloWorld"> <constructor-arg ref="fooBean" /> </bean> <bean id="fooBean" class="com.xyz.service.Foo"> </bean>  **2. Setter Injection :**   - Dependencies are assigned through setter method.  - Example : Same example as above.  public class HelloWorld { private Foo fooBean; public HelloWorld(){ }  public void setFooBean(Foo fooBean) {    this.fooBean=fooBean; }  - And in configuration file you need to do following entry.  <bean id="helloWorldBean" class=" com.xyz.services.HelloWorld"> <property name=fooBean ref="fooBean" /> </bean> <bean id="fooBean" class="com.xyz.service.Foo"> </bean>  **3. Interface Injection :**   Injection is done through an interface and not supported in spring framework. |
| **How to Call Stored procedure in Spring Framework?** - To call a Stored procedure in Spring framework you need to create Class which will should extends StoredProcedure class.  - Take the example of getting Employee Details by Employee Id. package com.mytest.spring.storeproc  import java.sql.ResultSet; import java.sql.Types; import java.util.HashMap; import java.util.Map; import javax.sql.DataSource; import org.springframework.jdbc.core.SqlOutParameter; import org.springframework.jdbc.core.SqlParameter; import org.springframework.jdbc.object.StoredProcedure; public class EmployeeInfo extends StoredProcedure  {    private static final String EMP\_ID = "EMP\_ID";     private static final String EMP\_NAME = "EMP\_NAME";    private static final String JOIN\_DATE = "JOIN\_DATE";    public SnapshotSearchStoredProcedure(DataSource dataSource, String procedureName)     {        super(dataSource, procedureName);         declareParameter(new SqlParameter(EMP\_ID, Types.NUMERIC));        declareParameter(new SqlOutParameter(EMP\_NAME, Types.VARCHAR));        declareParameter(new SqlOutParameter(JOIN\_DATE, Types.VARCHAR));        compile ();    }    public Map execute(Integer empId)     {        Map<String, Object> inputs = new HashMap<String, Object>();        inputs.put(P\_CLD\_IDR, empId);        Map<String, Object> result = execute (inputs);        return result;    } }  - You just need to call the execute method from the DAO layer. |
| **Differentiate between BeanFactory and ApplicationContext in spring.** - With ApplicationContext more than one config files are possible while only one config file or .xml file is possible with BeanFactory.  - ApplicationContext publishes events to beans that are registered as listeners while BeanFactory doesn't support this - ApplicationContext support internationalization messages, application life-cycle events, validation and many enterprise services like JNDI access, EJB integration, remoting etc. while BeanFactory doesn't support any of these. **What is the difference between singleton and prototype bean?** Mainly it is the scope of a beans which defines their existence on the application **Singleton :** It means single bean definition to a single object instance per Spring IOC container. **Prototype :** It means a single bean definition to any number of object instances. **How do beans become 'singleton' or prototype?** - There exists an attribute in bean tag, called 'singleton’.  - If it is marked 'true', the bean becomes 'singleton'.  - If it is marked 'false', the bean becomes 'prototype'. |
| **What type of transaction Management Spring support?** Spring supports two types of transaction management:   1. Programmatic transaction management 2. Declarative transaction management. **When do you use programmatic and declarative transaction management?** - Programmatic transaction management is used preferably when you have a small number of transactional operations.  - Incase of large number of transactional operations it is better to use declarative transaction management. **What is IOC?** - IOC stands for Inversion of Control pattern.  - It is also called as dependency injection.  - This concept says that you do not create your objects but describe how they should be created.  - Similarly, you do not directly connect your components and services together in code but describe which services are needed by which components in a configuration file.  - A container then hooks them all up. |
| **What are the different types of IoC (dependency injection)?** - There are three types of dependency injection: 1. Constructor Injection : Here dependencies are provided as constructor parameters.  2. Setter Injection : Dependencies are assigned through JavaBeans properties.  3. Interface Injection : Injection is performed through an interface. - Spring supports only first two categories of Injection. **What are the benefits of IOC?** The main benefits of IOC or dependency injections are : 1. It minimizes the amount of code in your application.  2. It makes your application easy to test as it doesn't require any singletons or JNDI lookup mechanisms in your unit test cases.  3. Loose coupling is promoted with minimal effort and least intrusive mechanism.  4. IOC containers support eager instantiation and lazy loading of services. **What is Bean Wiring?** - Bean wiring means creating associations between application components i.e. beans within the spring container. |
| **How do you access Hibernate using Spring ?** There are two ways to Spring’s Hibernate integration: 1. By Inversion of Control with a HibernateTemplate and Callback.  2. By extending HibernateDaoSupport and Applying an AOP Interceptor. **How would you integrate Spring and Hibernate using HibernateDaoSupport?** This can be done through Spring’s SessionFactory called LocalSessionFactory. The steps in integration process are: 1. Configure the Hibernate SessionFactory.  2. Extend your DAO Implementation from HibernateDaoSupport. 3. Wire in Transaction Support with AOP. **What are the various transaction manager implementations in Spring?** 1. DataSourceTransactionManager : PlatformTransactionManager implementation for single JDBC data sources.  2. HibernateTransactionManager: PlatformTransactionManager implementation for single Hibernate session factories.  3. JdoTransactionManager : PlatformTransactionManager implementation for single JDO persistence manager factories.  4. JtaTransactionManager : PlatformTransactionManager implementation for JTA, i.e. J2EE container transactions. |
| **What are the different modules in spring framework?** The Spring features or organized into about 20 modules. These modules are grouped into Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation and Test, as depicted below.  http://www.careerride.com/Images/spring-interview.png |
| **What is Auto Wiring in Spring?** - The Auto-wiring in spring framework can be performed by configuring in xml and Spring Auto-Wiring with Annotation @Autowired.  - Auto-wiring beans with xml configuration: In Spring framework, you can wire beans automatically with auto-wiring feature. To enable auto-wiring just define the “autowire” attribute in <bean> tag.  <bean id="customer" class="com.test.autowire.Customer" autowire="byName" />  - There are five modes of Auto-wiring supported.  **1. no – Default, no auto wiring, set it manually via “ref” attribute.**  <bean id="customer" class="com.test.autowire.Customer"> <property name="person" ref="person" /> </bean> <bean id="person" class="com.test.autowire.Person" />  **2. byName – Auto wiring by property name. If the name of a bean is same as the name of other bean property, auto wire it.**  - In below example the name of the person bean is same as name of “customer” bean’s property Person object. So spring will auto-wire it via setter method.  <bean id="customer" class="com.test.autowire.Customer" autowire="byName"/> <bean id="person" class="com.test.autowire.Person" />  **3. byType – Auto wiring by property data type. If data type of a bean is compatible with the data type of other bean property, auto wire it.**  - In below example the data type of the person bean is same as name of “customer” bean’s property Person object. So spring will auto-wire it via setter method.  <bean id="customer" class="com.test.autowire.Customer" autowire="byType"/> <bean id="person" class="com.test.autowire.Person" />  **4. constructor – byType mode in constructor argument.**  - Here the data type of “person” bean is same as the constructor argument data type in “customer” bean’s property (Person object), so, Spring auto wired it via constructor method – “public Customer(Person person)”  <bean id="customer" class="com.test.autowire.Customer" autowire="constructor"/> <bean id="person" class="com.test.autowire.Person" />  **5. autodetect – If a default constructor is found, use “autowired by constructor”; Otherwise, use “autowire by type”.**  - If a default constructor is found, uses “constructor”; Otherwise, uses “byType”. In this case, since there is a default constructor in “Customer” class, so, Spring auto wired it via constructor method – “public Customer(Person person)”.  <bean id="customer" class="com.test.autowire.Customer" autowire="autodetect"/> <bean id="person" class="com.test.autowire.Person" /> |
| **What is JdbcTemplate in Spring? And how to use it?** The JdbcTemplate class is the main class of the JDBC Core package. The JdbcTemplate (The class internally use JDBC API) helps to eliminate lot of code you write with simple JDBC API (Creating connection, closing connection, releasing resources, handling JDB Exceptions, handle transaction etc.). The JdbcTemplate handles the creation and release of resources, which helps you to avoid common error like forgetting to close connection.  - **Examples :**  **1. Getting row count from database.**  int rowCount = this.jdbcTemplate.queryForObject("select count(\*) from t\_employee", int.class);  **2. Querying for a String.**  String lastName = this.jdbcTemplate.queryForObject( "select last\_name from t\_employee where Emp\_Id = ?", new Object[]{377604L}, String.class);  **3. Querying for Object**  Employee employee = this.jdbcTemplate.queryForObject( "select first\_name, last\_name from t\_employee where Emp\_Id = ?", new Object[]{3778604L}, new RowMapper() { public Employee mapRow(ResultSet rs, int rowNum) throws SQLException { Employee employee = new Employee(); employee.setFirstName(rs.getString("first\_name")); employee.setLastName(rs.getString("last\_name")); return employee; } });  **4. Querying for N number of objects.**  List<Employee> employeeList = this.jdbcTemplate.query("select first\_name, last\_name from t\_employee", new RowMapper<Employee>() { public Employee mapRow(ResultSet rs, int rowNum) throws SQLException  {    Employee employee = new Employee();    employee.setFirstName(rs.getString("first\_name"));    employee.setLastName(rs.getString("last\_name"));    return employee; } }); |
| **What NamedParameterJdbcTemplate in Spring?** The NamedParameterJdbcTemplate allow basic set of JDBC operations, it allows named parameter in the query instead of traditional (?) placeholder, the functionality is similar to JdbcTemplate class.  - **Example :**  NamedParameterJdbcTemplate namedParameterJdbcTemplate;  String empInsrtQuery = "INSERT INTO Employee (name, age, salary) VALUES (:name, :age, :salary)";  Map namedParameters = new HashMap();  namedParameters.put("name", name);  namedParameters.put("age", age);  namedParameters.put("salary", salary);  namedParameterJdbcTemplate.update(empInsrtQuery, namedParameters); |
| **What are Advice, Aspect, Join-point and point cut in spring?** **Advice :**  An advice is an action taken by the aspect at particular join-point is called Advice.  **Aspect :**  An aspect is a subprogram which is associated with specific property of a program (Example separating logging code from the main program). An aspect is functionality or feature that cross cuts over object. AOP increase modularity of a program.  **Join-Point :**  A join point is a point used in spring AOP framework to represent a method execution. It always point during execution of program, method or exception. A join point is basically an opportunity within the code to apply aspect.  **Point Cut :**  In AOP a point cut is a set of many join points where an advice can execute. A chunk of code (known as Advice) associated with join point get executed. |
| **What are the different types of Advice?** There are different types of Advice.  **Before Advice :**  The advice which executed before a join point called before advice. The before advice does not have the ability to prevent the execution flow proceeding at the join point (unless it throws an exception).  **After Return Advice :**  The advice which executed after a join point completed normally without any exception.  **Around Advice :**  It is responsible for choosing whether to proceeds to the join point or shortcut the advised method execution by returning its own return value or throwing an exception. This is most powerful kind of advice. With Around advice you can perform custom behavior before and after method execution.  **After Throwing Advice :**  The advice executed when a method throws an exception.  **After (finally) Advice :** The advice is executed when program exits the join points either normally or by throwing an exception. |
| **What is Weaving in Spring?** Weaving is the process of linking aspect with other application types or object to create an advised object. This can be performed at compile time, runtime and load time. In spring framework weaving is performed at runtime. **What is AOP Proxy?** AOP proxy is an object to implement the aspect contracts (advice method executions and so on). The AOP proxy is object is created by the AOP framework. In spring framework AOP proxy is JDK dynamic proxy or CGLIB proxy. |
| **What is front controller in Spring MVC?** The Front Controller is basically a type of Design pattern which are being implemented in different framework (e.g. Struts and Spring MVC etc.).  - In Spring MVC DispatcherServlet act as a Front Controller for the framework and responsible for intercepting every request and then dispatches/forwards request to appropriate controller. Configure the DispatcherServlet in the web.xml file of web application and request which we want to be handled by DispatcherServlet should be mapped using URL mapping.  - For example all the requests ending with \*.do will be handled by the DispatcherServlet.  <web-app> <servlet> <servlet-name>example</servlet-name> <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class> <load-on-startup>1</load-on-startup> </servlet> <servlet-mapping> <servlet-name>example</servlet-name> <url-pattern>\*.do</url-pattern> </servlet-mapping> </web-app> |
| **Difference between FileSystemResource and ClassPathResource.** - In FileSystemResource you need to give the configuration file (i.e. spring-config.xml) relative to your project or the absolute location of the file.  - In ClassPathResource spring looks for the file in the ClassPath so configuration (i.e. spring-config.xml) file should be included in the classpath. If spring-config.xml is in classpath, you can simply give the name of the file.  - For Example: If your configuration file is at src/main/java/com/test/loadresource then your FileSystemResource would be:  FileSystemResource resource = new FileSystemResource("src/main/java/com/test/loadresource/spring-config.xml");  And ClassPathResource would be :  ClassPathResource resource = new ClassPathResource("com/test/loadresource /spring-config.xml"); |
| **What is inner Bean Definition?** - A bean definition added inside the property or constructor-arg elements are called inner bean.  - **Example :**  <bean id="outerbean" class="..."> <!-- instead of using a reference to a target bean, simply define the target bean inline --> <property name="targetbean"> <bean class="com.example.Person"> <!-- this is the inner bean --> <property name="name" value="XYZ"/> <property name="age" value="35"/> </bean> </property> </bean> |
| **Give examples of how spring platform can be used by an application developer.** The spring platform can be used by an application developer in the following way : - Java method can be made to execute in a database transaction without having to deal with transaction APIs. - Local Java method can be made a remote procedure without having to deal with remote APIs. - Local Java method can be made a management operation without having to deal with JMX APIs. - Local Java method can be made a message handler without having to deal with JMS APIs. |
| **What are the various ways of using spring?** - There are various ways and forms in which springs can be used. - They are listed as follows: 1. Full-fledged Spring web app. 2. Spring middle-tier provides help of a third-party web framework. 3. Remote usage scenario: allow the system to be used to remotely use the resources from the server. The remote usage can be done to grab the data from the server or for troubleshooting the environment.  4. EJB -- Wrapping existing POJOs. |
| **Specify the locations where spring can publish its artifacts.** Spring generally publishes its artifacts to four different places: 1. Community download site http://www.springsource.org/downloads/community. 2. Maven Central, which is considered the default repository that Maven queries, and does not require any special configuration to use. 3. The Enterprise Bundle Repository (EBR), which is considered to be run by SpringSource and also hosts all the libraries that integrate with Spring. 4. Public Maven repository hosted on Amazon S3 for the development of snapshots and milestone releases. The jar file names are given in the same form as Maven Central, which makes it a useful place to get development versions of Spring to use with other libraries Spring Framework deployed in Maven Central. |
| **What are the features of the new spring build system in use now days?** Now the new Spring build system is used which comes with the following features : 1. "Spring Built" system which is based on Ivy. 2. consistent procedure for deployment. 3. dependency management which is made consistent. 4. consistent generation for OSGi. |
| **List in brief the new features Spring 3.0 has to offer.** Spring 3.0 offers the following new features: 1. Spring Expression Language. 2. IoC enhancements or Java based bean metadata. 3. field formatting and General-purpose type conversion system. 4. Object to XML mapping functionality (OXM) moved from Spring Web Services project. 5. Comprehensive REST support. 6. @MVC additions. 7. Declarative model validation. 8. Early support for Java EE 6. 9. Embedded database support. |
| **What are the annotations supported by the spring framework?** Due to addition of some core features from the JavaConfig project to the Spring Framework,the following annotations are now directly supported : - @Configuration - @Bean - @DependsOn - @Primary - @Lazy - @Import - @ImportResource - @Value |
| **Explain in brief the metadata contained in the BeanDefinition objects.** Bean definitions contain the following metadata:  1. A package-qualified class name: which is the actual implementation class of the bean being defined. 2. Bean behavioral configuration elements, which state the procedure for how the bean should behave in the container 3. References to other beans that are required for the bean to do its work; these references are also called collaborators or dependencies. 4. Other configuration settings to set when a new object is created, for example, the number of connections to use in a bean that manages a connection pool or the size limit of the pool. |
| **What are the different properties the metadata translates into to make up the bean definition?** Metadata translates into the following properties to make up: 1. Property defined in class the section called “Instantiating beans”. 2. Property defined in name the section called “Naming beans” scope constructor arguments the section known as “Dependency injection”. 3. Properties the section known as “Dependency injection” auto wiring mode the section known as “Autowiring collaborators”. 4. lazy-initialization mode the section called “Lazy-initialized beans”. 5. Initialization method the section called “Initialization callbacks”. 6. Destruction method the section called “Destruction callbacks”. |
| **Briefly describe the 2 ways in which the class property can be used.** The class property can be put to use in the following ways:  1. It is used to specify the java bean for class to be constructed when container directly creates java bean by calling constructor reflectively. This procedure is also used for java code which uses a new operator. 2. It is also used to specify the class which contains static factory method which is invoked for creating object. Another case in this which doesn’t occur much is when static factory method is invoked by container on a class for creating the bean. |
| **Explain the dependency resolution process.** The dependency resolution process is carried out as follows: 1. Creation an initialization of application text inside configuration metadata( can be specified in XML, java code,annotations) is done. This describes all the beans. 2. For each bean, its dependency is expressed in the form of properties, constructor arguments, or arguments to the static-factory method if that is used instead of a normal constructor. 3. Each property or constructor argument is actual definition of the value which belong to set, or a reference to another bean which is in the container. 4. Each property or constructor argument which is a value is gone through conversion from its specified format to the actual type of that property or constructor argument. |
| **How can you use spring to resolve collaborators for your bean? Also state the advantages of auto wiring?** - The spring can be used to resolve collaborators automatically by inspecting contents of the ApplicationContext. - The Autowiring has the following advantages in spring:  1. Reduces requirement for specifying properties or construction arguments. 2. Updates configuration while evolving of objects. To understand this lets consider an example: modifying would not be required in configuration of a dependency while adding it to a class. 3. Switching to explicit wiring when code base is stable can be neglected. |
| **List the limitations of auto wiring.** **Autowiring has the following limitations :**  1. Overriding and auto wiring are caused due to dependency in property and constructor argument setting. 2. Less precise as compared to explicit wiring. 3. Tools that generate documentation using a spring might not have access to wiring information. 4. There is a possibility of clash between bean definitions and argument or method to be wired. 5. The problem does not occur much in case of maps, arrays and collection and cannot be resolved randomly for dependencies which expect one value. |
| **In scenarios where you have to avoid using auto wiring, what are the other options that can be explored to achieve the same?** In scenarios where using autowiring is prohibited, the following replacements can achive the same : 1. Abandon autowiring for favor of explicit wiring. 2. Avoid autowiring for a bean definition by setting the autowire-candidate for attributes to false as described in the next section. 3. Set a single bean definition as the primary candidate by setting the primary attribute of its <bean/> element to true. 4. When Java 5 or later is used, implementation needs more fine-grained control available with annotation-based configuration. |
| **How do you call the multiple lifecycle mechanisms configured for the same bean?** - Multiple lifecycle mechanisms which are configured for the same bean, with different initialization methods, are called by the following ways : 1. Methods annotated with use of @PostConstruct. 2. afterPropertiesSet() as stated by the InitializingBean callback interface. 3. Custom configuration init() method - Destroy methods are called in the same order as follows : 1. Methods annotated with the use of @PreDestroy. 2. destroy() as stated by the DisposableBean callback interface. 3. destroy() method with custom configuration. |
| **What are the methods associated with the FactoryBean interface?** Following are the methods associated with factory bean interface:  1. Object getObject() : This returns an instance of the object which the factory creates. This instance can possibly be shared, depending on the factory returns singletons or prototypes. 2. Boolean isSingleton() : This returns true if this FactoryBean returns singletons, otherwise false. 3. Class getObjectType() : This returns the object type returned by the getObject() method or null if the type is unknown in advance. |
| **What are the functionalities provided by the context package to enhance the BeanFactory functionality?** The functionalities provided by context package are as follows:  1. Message access in i18n-style, through the MessageSource interface. 2. Resource access such as URLs and files, through which, the ResourceLoader interface, can be accessed and used. 3. Publication of events, through the beans that allow, implementing the ApplicationListener interface. It is done through the use, of the ApplicationEventPublisher interface. 4. Multiple loading contexts, allowing each to be focused on one particular layer, such as the web layer of an application, through the HierarchicalBeanFactory interface. |
| **How can you achieve Internationalization using MessageSource?** Internationnalization using message source can be achieved by the following :  **1. String get Mesage(String code, Objct[] args, String default, Local loc) :** The basic method used for retrieving a message from the MessageSource. When no message is to be found for the specified locale, the default message is used.  **2. String getMessage(String code, Objct[] args, Locale loc) :** Essentially the same as the last method, but with a difference, no default message is specified; if the message cannot be found, a NoSuchMessageException is shown.  **3. String getMessage(MessageSourceResolvable resolvable, Local locale) :** Properties used for the preceding methods are also wrapped in a class named MessageSourceResolvable, which you can use with this method. |
| **Enlist and explain the main methods associated with the Resource interface.** Methods associated with resource interface are as follows:  **1. getInputStream() :** It locates and opens the resource, returning an InputStream to read from the resource. **2. exists() :** Returns a boolean which indicates whether this resource actually exists in physical form. **3. isOpen() :** Returns a boolean which indicates whether this resource represents a handle with an open stream. **4. getDescription() :** Returns a description for this resource, which is used for error output when working with the resource. |
| **Give some examples where property editing is used in spring.** Following are the examples for property editing in spring : 1. Setting of properties on beans is done by using PropertyEditors. To mention java.lang.string as the value for a property of some bean you're declaring in XML file, Spring will use the ClassEditor to try resolving the parameter to a Class object. 2. Passing HTTP request parameters in Spring's MVC framework can be done using all kinds of PropertyEditors that can be manually bind in all subclasses of the CommandController. |
| **Explain the main AOP concepts and terminology.** The main AOP concepts are as follows:   1. Aspect : A modularization of a concern that cuts through multiple classes. 2. Join point : It is a point during which the execution of a programming, such as the execution of method or handling of an exception. 3. Advice : An action taken by an aspect for a particular join point. 4. Point cut : A predicate that is matched to join points. 5. Introduction : Declaration of additional methods or fields on behalf of a type. 6. Target object : Object which is advised by one or more aspects. 7. AOP proxy : An object which is created by AOP framework for implementing the aspect contracts. 8. Weaving : Linking of aspects with other application types or objects to create an advised object. |

Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run". We take an opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss. Most Spring Boot applications need very little Spring configuration.

## **Features**

* Create stand-alone Spring applications
* Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files)
* Provide opinionated 'starter' POMs to simplify your Maven configuration
* Automatically configure Spring whenever possible
* Provide production-ready features such as metrics, health checks and externalized configuration
* Absolutely **no code generation** and **no requirement for XML** configuration

The [reference guide](https://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle) includes detailed descriptions of all the features, plus an extensive [howto](https://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/" \l "howto) for common use cases.

## **Quick Start**

If you are Java developer you can use [start.spring.io](https://start.spring.io/) to generate a basic project, follow the "Quick Start" example below, or read the reference documentation [getting started](https://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#getting-started) guide.

GRADLE

The recommended way to get started using spring-boot in your project is with a dependency management system – the snippet below can be copied and pasted into your build. Need help? See our getting started guides on building with [Maven](https://spring.io/guides/gs/maven/) and [Gradle](https://spring.io/guides/gs/gradle/).

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.0.2.RELEASE</version>

</parent>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

</dependencies>

hello/SampleController.java

**package** hello**;**

**import** org.springframework.boot.\***;**

**import** org.springframework.boot.autoconfigure.\***;**

**import** org.springframework.stereotype.\***;**

**import** org.springframework.web.bind.annotation.\***;**

**@Controller**

**@EnableAutoConfiguration**

**public** **class** **SampleController** **{**

**@RequestMapping(**"/"**)**

**@ResponseBody**

String **home()** **{**

**return** "Hello World!"**;**

**}**

**public** **static** **void** **main(**String**[]** args**)** **throws** Exception **{**

SpringApplication**.**run**(**SampleController**.**class**,** args**);**

**}**

**}**

## **Spring Boot CLI**

Spring Boot ships with a command line tool that can be used if you want to quickly prototype with Spring. It allows you to run Groovy scripts, which means that you have a familiar Java-like syntax, without so much boilerplate code. Follow the instructions in our main documentation if you want to [install the Spring Boot CLI](https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#getting-started-installing-the-cli).

# Building an Application with Spring Boot

This guide provides a sampling of how [Spring Boot](https://github.com/spring-projects/spring-boot) helps you accelerate and facilitate application development. As you read more Spring Getting Started guides, you will see more use cases for Spring Boot. It is meant to give you a quick taste of Spring Boot. If you want to create your own Spring Boot-based project, visit [Spring Initializr](https://start.spring.io/), fill in your project details, pick your options, and you can download either a Maven build file, or a bundled up project as a zip file.

## What you’ll build

You’ll build a simple web application with Spring Boot and add some useful services to it.

## What you’ll need

* About 15 minutes
* A favorite text editor or IDE
* [JDK 1.8](http://www.oracle.com/technetwork/java/javase/downloads/index.html) or later
* [Gradle 4+](http://www.gradle.org/downloads) or [Maven 3.2+](https://maven.apache.org/download.cgi)
* You can also import the code straight into your IDE:
  + [Spring Tool Suite (STS)](https://spring.io/guides/gs/sts)
  + [IntelliJ IDEA](https://spring.io/guides/gs/intellij-idea/)

## How to complete this guide

Like most Spring [Getting Started guides](https://spring.io/guides), you can start from scratch and complete each step, or you can bypass basic setup steps that are already familiar to you. Either way, you end up with working code.

To **start from scratch**, move on to [Build with Gradle](https://spring.io/guides/gs/spring-boot/#scratch).

To **skip the basics**, do the following:

* [Download](https://github.com/spring-guides/gs-spring-boot/archive/master.zip) and unzip the source repository for this guide, or clone it using [Git](https://spring.io/understanding/Git): git clone <https://github.com/spring-guides/gs-spring-boot.git>
* cd into gs-spring-boot/initial
* Jump ahead to [[initial]](https://spring.io/guides/gs/spring-boot/#initial).

**When you’re finished**, you can check your results against the code in gs-spring-boot/complete.

## Build with Gradle

## Build with Maven

## Build with your IDE

## Learn what you can do with Spring Boot

Spring Boot offers a fast way to build applications. It looks at your classpath and at beans you have configured, makes reasonable assumptions about what you’re missing, and adds it. With Spring Boot you can focus more on business features and less on infrastructure.

For example:

* Got Spring MVC? There are several specific beans you almost always need, and Spring Boot adds them automatically. A Spring MVC app also needs a servlet container, so Spring Boot automatically configures embedded Tomcat.
* Got Jetty? If so, you probably do NOT want Tomcat, but instead embedded Jetty. Spring Boot handles that for you.
* Got Thymeleaf? There are a few beans that must always be added to your application context; Spring Boot adds them for you.

These are just a few examples of the automatic configuration Spring Boot provides. At the same time, Spring Boot doesn’t get in your way. For example, if Thymeleaf is on your path, Spring Boot adds a SpringTemplateEngine to your application context automatically. But if you define your own SpringTemplateEngine with your own settings, then Spring Boot won’t add one. This leaves you in control with little effort on your part.

|  |  |
| --- | --- |
|  | Spring Boot doesn’t generate code or make edits to your files. Instead, when you start up your application, Spring Boot dynamically wires up beans and settings and applies them to your application context. |

## Create a simple web application

Now you can create a web controller for a simple web application.

src/main/java/hello/HelloController.java

package hello;

import org.springframework.web.bind.annotation.RestController;

import org.springframework.web.bind.annotation.RequestMapping;

@RestController

public class HelloController {

@RequestMapping("/")

public String index() {

return "Greetings from Spring Boot!";

}

}

The class is flagged as a @RestController, meaning it’s ready for use by Spring MVC to handle web requests. @RequestMapping maps / to the index() method. When invoked from a browser or using curl on the command line, the method returns pure text. That’s because @RestController combines @Controller and @ResponseBody, two annotations that results in web requests returning data rather than a view.

## Create an Application class

Here you create an Application class with the components:

src/main/java/hello/Application.java

package hello;

import java.util.Arrays;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

import org.springframework.context.annotation.Bean;

@SpringBootApplication

public class Application {

public static void main(String[] args) {

SpringApplication.run(Application.class, args);

}

@Bean

public CommandLineRunner commandLineRunner(ApplicationContext ctx) {

return args -> {

System.out.println("Let's inspect the beans provided by Spring Boot:");

String[] beanNames = ctx.getBeanDefinitionNames();

Arrays.sort(beanNames);

for (String beanName : beanNames) {

System.out.println(beanName);

}

};

}

}

@SpringBootApplication is a convenience annotation that adds all of the following:

* @Configuration tags the class as a source of bean definitions for the application context.
* @EnableAutoConfiguration tells Spring Boot to start adding beans based on classpath settings, other beans, and various property settings.
* Normally you would add @EnableWebMvc for a Spring MVC app, but Spring Boot adds it automatically when it sees **spring-webmvc** on the classpath. This flags the application as a web application and activates key behaviors such as setting up a DispatcherServlet.
* @ComponentScan tells Spring to look for other components, configurations, and services in the hello package, allowing it to find the controllers.

The main() method uses Spring Boot’s SpringApplication.run() method to launch an application. Did you notice that there wasn’t a single line of XML? No **web.xml** file either. This web application is 100% pure Java and you didn’t have to deal with configuring any plumbing or infrastructure.

There is also a CommandLineRunner method marked as a @Bean and this runs on start up. It retrieves all the beans that were created either by your app or were automatically added thanks to Spring Boot. It sorts them and prints them out.

## Run the application

To run the application, execute:

./gradlew build && java -jar build/libs/gs-spring-boot-0.1.0.jar

If you are using Maven, execute:

mvn package && java -jar target/gs-spring-boot-0.1.0.jar

You should see some output like this:

Let's inspect the beans provided by Spring Boot:

application

beanNameHandlerMapping

defaultServletHandlerMapping

dispatcherServlet

embeddedServletContainerCustomizerBeanPostProcessor

handlerExceptionResolver

helloController

httpRequestHandlerAdapter

messageSource

mvcContentNegotiationManager

mvcConversionService

mvcValidator

org.springframework.boot.autoconfigure.MessageSourceAutoConfiguration

org.springframework.boot.autoconfigure.PropertyPlaceholderAutoConfiguration

org.springframework.boot.autoconfigure.web.EmbeddedServletContainerAutoConfiguration

org.springframework.boot.autoconfigure.web.EmbeddedServletContainerAutoConfiguration$DispatcherServletConfiguration

org.springframework.boot.autoconfigure.web.EmbeddedServletContainerAutoConfiguration$EmbeddedTomcat

org.springframework.boot.autoconfigure.web.ServerPropertiesAutoConfiguration

org.springframework.boot.context.embedded.properties.ServerProperties

org.springframework.context.annotation.ConfigurationClassPostProcessor.enhancedConfigurationProcessor

org.springframework.context.annotation.ConfigurationClassPostProcessor.importAwareProcessor

org.springframework.context.annotation.internalAutowiredAnnotationProcessor

org.springframework.context.annotation.internalCommonAnnotationProcessor

org.springframework.context.annotation.internalConfigurationAnnotationProcessor

org.springframework.context.annotation.internalRequiredAnnotationProcessor

org.springframework.web.servlet.config.annotation.DelegatingWebMvcConfiguration

propertySourcesBinder

propertySourcesPlaceholderConfigurer

requestMappingHandlerAdapter

requestMappingHandlerMapping

resourceHandlerMapping

simpleControllerHandlerAdapter

tomcatEmbeddedServletContainerFactory

viewControllerHandlerMapping

You can clearly see **org.springframework.boot.autoconfigure** beans. There is also a tomcatEmbeddedServletContainerFactory.

Check out the service.

$ curl localhost:8080

Greetings from Spring Boot!

## Add Unit Tests

You will want to add a test for the endpoint you added, and Spring Test already provides some machinery for that, and it’s easy to include in your project.

Add this to your build file’s list of dependencies:

testCompile("org.springframework.boot:spring-boot-starter-test")

If you are using Maven, add this to your list of dependencies:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

Now write a simple unit test that mocks the servlet request and response through your endpoint:

src/test/java/hello/HelloControllerTest.java

package hello;

import static org.hamcrest.Matchers.equalTo;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.content;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

import org.junit.Test;

import org.junit.runner.RunWith;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.http.MediaType;

import org.springframework.test.context.junit4.SpringRunner;

import org.springframework.test.web.servlet.MockMvc;

import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;

@RunWith(SpringRunner.class)

@SpringBootTest

@AutoConfigureMockMvc

public class HelloControllerTest {

@Autowired

private MockMvc mvc;

@Test

public void getHello() throws Exception {

mvc.perform(MockMvcRequestBuilders.get("/").accept(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

.andExpect(content().string(equalTo("Greetings from Spring Boot!")));

}

}

The MockMvc comes from Spring Test and allows you, via a set of convenient builder classes, to send HTTP requests into the DispatcherServlet and make assertions about the result. Note the use of the @AutoConfigureMockMvc together with @SpringBootTest to inject a MockMvc instance. Having used @SpringBootTest we are asking for the whole application context to be created. An alternative would be to ask Spring Boot to create only the web layers of the context using the @WebMvcTest. Spring Boot automatically tries to locate the main application class of your application in either case, but you can override it, or narrow it down, if you want to build something different.

As well as mocking the HTTP request cycle we can also use Spring Boot to write a very simple full-stack integration test. For example, instead of (or as well as) the mock test above we could do this:

src/test/java/hello/HelloControllerIT.java

package hello;

import static org.hamcrest.Matchers.\*;

import static org.junit.Assert.\*;

import java.net.URL;

import org.junit.Before;

import org.junit.Test;

import org.junit.runner.RunWith;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.boot.test.web.client.TestRestTemplate;

import org.springframework.boot.web.server.LocalServerPort;

import org.springframework.http.ResponseEntity;

import org.springframework.test.context.junit4.SpringRunner;

@RunWith(SpringRunner.class)

@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM\_PORT)

public class HelloControllerIT {

@LocalServerPort

private int port;

private URL base;

@Autowired

private TestRestTemplate template;

@Before

public void setUp() throws Exception {

this.base = new URL("http://localhost:" + port + "/");

}

@Test

public void getHello() throws Exception {

ResponseEntity<String> response = template.getForEntity(base.toString(),

String.class);

assertThat(response.getBody(), equalTo("Greetings from Spring Boot!"));

}

}

The embedded server is started up on a random port by virtue of the webEnvironment = SpringBootTest.WebEnvironment.RANDOM\_PORT and the actual port is discovered at runtime with the @LocalServerPort.

## Add production-grade services

If you are building a web site for your business, you probably need to add some management services. Spring Boot provides several out of the box with its [actuator module](https://docs.spring.io/spring-boot/docs/2.0.2.RELEASE/reference/htmlsingle/#production-ready), such as health, audits, beans, and more.

Add this to your build file’s list of dependencies:

compile("org.springframework.boot:spring-boot-starter-actuator")

If you are using Maven, add this to your list of dependencies:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

Then restart the app:

./gradlew build && java -jar build/libs/gs-spring-boot-0.1.0.jar

If you are using Maven, execute:

mvn package && java -jar target/gs-spring-boot-0.1.0.jar

You will see a new set of RESTful end points added to the application. These are management services provided by Spring Boot.

2018-03-17 15:42:20.088 ... : Mapped "{[/error],produces=[text/html]}" onto public org.s...

2018-03-17 15:42:20.089 ... : Mapped "{[/error]}" onto public org.springframework.http.R...

2018-03-17 15:42:20.121 ... : Mapped URL path [/webjars/\*\*] onto handler of type [class ...

2018-03-17 15:42:20.121 ... : Mapped URL path [/\*\*] onto handler of type [class org.spri...

2018-03-17 15:42:20.157 ... : Mapped URL path [/\*\*/favicon.ico] onto handler of type [cl...

2018-03-17 15:42:20.488 ... : Mapped "{[/actuator/health],methods=[GET],produces=[application/vnd...

2018-03-17 15:42:20.490 ... : Mapped "{[/actuator/info],methods=[GET],produces=[application/vnd.s...

2018-03-17 15:42:20.491 ... : Mapped "{[/actuator],methods=[GET],produces=[application/vnd.spring...

They include: errors, [actuator/health](http://localhost:8080/actuator/health), [actuator/info](http://localhost:8080/actuator/info), [actuator](http://localhost:8080/actuator).

|  |  |
| --- | --- |
|  | There is also a /actuator/shutdown endpoint, but it’s only visible by default via JMX. To [enable it as an HTTP endpoint](https://docs.spring.io/spring-boot/docs/2.0.2.RELEASE/reference/htmlsingle/#production-ready-endpoints-enabling-endpoints), add management.endpoints.shutdown.enabled=true to your application.properties file. |

It’s easy to check the health of the app.

$ curl localhost:8080/actuator/health

{"status":"UP"}

You can try to invoke shutdown through curl.

$ curl -X POST localhost:8080/actuator/shutdown

{"timestamp":1401820343710,"error":"Method Not Allowed","status":405,"message":"Request method 'POST' not supported"}

Because we didn’t enable it, the request is blocked by the virtue of not existing.

For more details about each of these REST points and how you can tune their settings with an application.properties file, you can read detailed [docs about the endpoints](https://docs.spring.io/spring-boot/docs/2.0.2.RELEASE/reference/htmlsingle/#production-ready-endpoints).

## View Spring Boot’s starters

You have seen some of [Spring Boot’s "starters"](https://docs.spring.io/spring-boot/docs/2.0.2.RELEASE/reference/htmlsingle/#using-boot-starter). You can see them all [here in source code](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-project/spring-boot-starters).

## JAR support and Groovy support

The last example showed how Spring Boot makes it easy to wire beans you may not be aware that you need. And it showed how to turn on convenient management services.

But Spring Boot does yet more. It supports not only traditional WAR file deployments, but also makes it easy to put together executable JARs thanks to Spring Boot’s loader module. The various guides demonstrate this dual support through the spring-boot-gradle-plugin and spring-boot-maven-plugin.

On top of that, Spring Boot also has Groovy support, allowing you to build Spring MVC web apps with as little as a single file.

Create a new file called **app.groovy** and put the following code in it:

@RestController

class ThisWillActuallyRun {

@RequestMapping("/")

String home() {

return "Hello World!"

}

}

## Spring Boot Tutorial

Spring Boot uses completely new development model to make Java Development very easy by avoiding some tedious development steps and boilerplate code and configuration.

### What is Spring Boot?

Spring Boot is a Framework from “The Spring Team” to ease the bootstrapping and development of new Spring Applications.  
It provides defaults for code and annotation configuration to quick start new Spring projects within no time. It follows “Opinionated Defaults Configuration” Approach to avoid lot of boilerplate code and configuration to improve Development, Unit Test and Integration Test Process.

### What is NOT Spring Boot?

Spring Boot Framework is not implemented from the scratch by The Spring Team, rather than implemented on top of existing Spring Framework (Spring IO Platform).  
It is not used for solving any new problems. It is used to solve same problems like Spring Framework.

### Why Spring Boot?

* To ease the Java-based applications Development, Unit Test and Integration Test Process.
* To reduce Development, Unit Test and Integration Test time by providing some defaults.
* To increase Productivity.

Don’t worry about what is “Opinionated Defaults Configuration” Approach at this stage. We will explain this in detail with some examples in coming posts.

### Advantages of Spring Boot:

* It is very easy to develop Spring Based applications with Java or Groovy.
* It reduces lots of development time and increases productivity.
* It avoids writing lots of boilerplate Code, Annotations and XML Configuration.
* It is very easy to integrate Spring Boot Application with its Spring Ecosystem like Spring JDBC, Spring ORM, Spring Data, Spring Security etc.
* It follows “Opinionated Defaults Configuration” Approach to reduce Developer effort
* It provides Embedded HTTP servers like Tomcat, Jetty etc. to develop and test our web applications very easily.
* It provides CLI (Command Line Interface) tool to develop and test Spring Boot(Java or Groovy) Applications from command prompt very easily and quickly.
* It provides lots of plugins to develop and test Spring Boot Applications very easily using Build Tools like Maven and Gradle
* It provides lots of plugins to work with embedded and in-memory Databases very easily.

In Simple Terminology, What Spring Boot means

[What Is Spring Boot, Spring Boot Tutorial](https://cdn.journaldev.com/wp-content/uploads/2015/05/WhatIsSpringBoot1.png)

That means Spring Boot is nothing but existing Spring Framework + Some Embedded HTTP Servers (Tomcat/Jetty etc.) – XML or Annotations Configurations.  
Here minus means we don’t need to write any XML Configuration and few Annotations only.

### Main Goal of Spring Boot:

The main goal of Spring Boot Framework is to reduce Development, Unit Test and Integration Test time and to ease the development of Production ready web applications very easily compared to existing Spring Framework, which really takes more time.

* To avoid XML Configuration completely
* To avoid defining more Annotation Configuration(It combined some existing Spring Framework Annotations to a simple and single Annotation)
* To avoid writing lots of import statements
* To provide some defaults to quick start new projects within no time.
* To provide Opinionated Development approach.

By providing or avoiding these things, Spring Boot Framework reduces Development time, Developer Effort and increases productivity.

### Limitation/Drawback of Spring Boot:

Spring Boot Framework has one limitation.  
It is some what bit time consuming process to convert existing or legacy Spring Framework projects into Spring Boot Applications but we can convert all kinds of projects into Spring Boot Applications. It is very easy to create brand new/Greenfield Projects using Spring Boot.

To Start Opinionated Approach to create Spring Boot Applications, The Spring Team (The Pivotal Team) has provided the following three approaches.

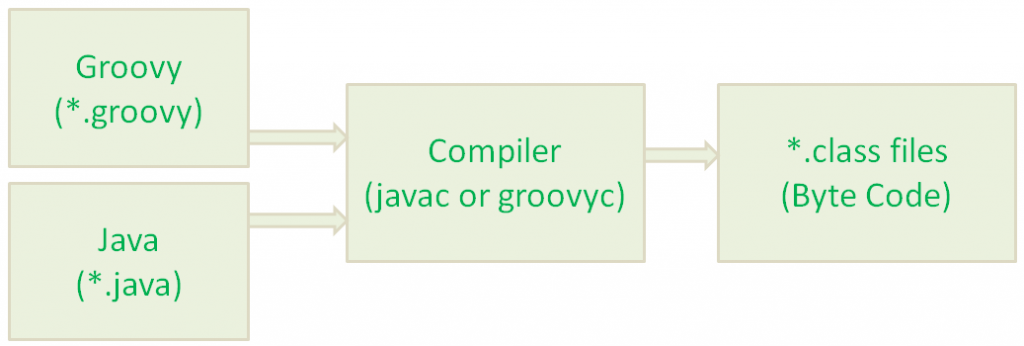
* Using Spring Boot CLI Tool
* Using Spring STS IDE
* Using Spring Initializr Website

We will discuss one by one in detail with some good examples in coming posts. We can find Spring Initializr Website at: http://start.spring.io/  
We can develop two flavors of Spring-Based Applications using Spring Boot

* Java-Based Applications
* Groovy Applications

We can use Spring Boot CLI or Spring STS IDE or Spring Initializr Website to develop Spring Boot Groovy Applications. However, we can use Spring STS IDE or Spring Initializr Website to develop Spring Boot Java Applications.

Anyhow, Groovy is also JVM language almost similar to Java Language. We can combine both Groovy and Java into one Project. Because like Java files, Groovy files are finally compiled into \*.class files only. Both \*.groovy and \*.java files are converted to \*.class file (Same byte code format).

[](https://cdn.journaldev.com/wp-content/uploads/2015/05/java_groovy_compiler2.png)  
Spring Boot Framework Programming model is inspired by Groovy Programming model. Spring Boot internally uses some Groovy based techniques and tools to provide default imports and configuration.

Spring Boot Framework also combined existing Spring Framework annotations into some simple or single annotations. We will explore those annotations one by one in coming posts with some real-time examples.

Spring Boot Framework drastically changes Spring-Java Based Applications Programming model into new Programming model. As of now, Spring Boot is at initial stage only but future is all about Spring Boot only.

# Creating a Spring Boot Project With Eclipse and Maven

### In this article, we look at three options for creating Spring Boot projects with Maven and Eclipse: using Spring Initializr, using the STS plugin, and doing it manually.

[**[](https://dzone.com/users/1041459/ranga_pec.html)**](https://dzone.com/users/1041459/ranga_pec.html)**by**

[**Ranga Karanam**](https://dzone.com/users/1041459/ranga_pec.html)

**·**

**Dec. 12, 17 ·**[**Java Zone**](https://dzone.com/java-jdk-development-tutorials-tools-news)**· Tutorial**

**Like** [**(7)**](https://dzone.com/articles/creating-a-spring-boot-project-with-eclipse-and-ma)

**Comment (3)**

**Save**

[**Tweet**](https://dzone.com/articles/creating-a-spring-boot-project-with-eclipse-and-ma)

**45.47k Views**

Join the DZone community and get the full member experience.

[**JOIN FOR FREE**](https://dzone.com/articles/creating-a-spring-boot-project-with-eclipse-and-ma)

[Akka from A to Z, An Architects Guide](https://dzone.com/go?i=290426&u=https%3A%2F%2Finfo.lightbend.com%2Fcollateral-akka-for-architects-a-z-guide-design-build-run-reactive-apps-register.html%3Futm_source%3Ddzone%26utm_medium%3Dpre-roll-text%26utm_campaign%3DCOLL-2018-APR-AKKA-for-Architects-A-Z-Guide-Design-Build-Run-Reactive-Apps%26utm_term%3Dnone%26utm_content%3Djava-zone) starting off with Actors and Akka Streams, then on to clustering, sharding, event sourcing & CQRS, and more.

Setting up a basic project with Spring Boot is a cake walk. In this article, we will explore the different options of creating Spring Boot projects with Maven and Eclipse. You will learn:

* How to bootstrap a simple project with Spring Initializr.
* How to use the Spring Starter Eclipse Plugin to create a simple project with Spring Boot, Maven, and Eclipse.
* How to create a Spring Boot project manually, step-by-step.

## Video References

* [Spring](https://in28minutes.teachable.com/p/spring-framework-tutorial-for-beginners)
* [Spring MVC](https://www.youtube.com/watch?v=BjNhGaZDr0Y)
* [Spring Boot](https://www.youtube.com/watch?v=PSP1-2cN7vM)
* [Eclipse](https://www.youtube.com/watch?v=s4ShbtOHMCA)
* [Maven](https://www.youtube.com/watch?v=0CFWeVgzsqY)
* [JUnit](https://www.youtube.com/watch?v=o5k9NOR9lrI)
* [Mockito](https://www.youtube.com/watch?v=d2KwvXQgQx4)

## Tools You Will Need

* Maven 3.0+ is your build tool
* Your favorite IDE (we use Eclipse in this article)
* JDK 1.8+

## Introduction to Maven

Defining what Maven does is very difficult. To help explain, let's consider some of the things that a developer does every day...

* Manages dependencies
  + Web layer (Spring MVC)
  + Data layer (JPA, Hibernate)
* Build a JAR or a WAR or an EAR
* Run the application locally
* Deploy to a T environment
* Add new dependencies to a project
* Run unit tests
* Generate projects
* Create Eclipse Workspace

Maven helps us do all of this and more!

### Naming a Project

How can other projects use our project? By using our project groupId and artifactId. We give a name to the project in the pom.xml, as shown below:

<groupId>com.in28minutes.learning.maven</groupId>

<artifactId>maven-in-few-steps</artifactId>

### Declaring Dependencies

Dependencies are frameworks that you need in order to develop your project. In the example below, we add two dependencies:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

## Maven Build Lifecycle

When we run mvn clean install, we are executing the complete Maven build lifecycle. This lifecycle is a sequence of the following steps:

* Validate
* Compile
* Test
* Package
* Integration test
* Verify
* Install
* Deploy

Maven follows **convention** over **configuration**.

The predefined folder structure looks like this:

* Source code
  + ${basedir}/src/main/java
  + ${basedir}/src/main/resources
* Test code

## How Does Maven Work?

A Maven repository contains all the JARs indexed by artifact ID and group ID. Once we add a dependency to our pom.xml, Maven asks the Maven repository for the JAR dependencies, giving group ID and the artifact ID as the input.

The JAR dependencies are stored on your machine in a folder called maven local repository. All our projects refer to the JARs from here.

**Note**: A local repository is a temp folder on your machine where Maven stores the JAR and dependency files that are downloaded from the Maven repository.

### Important Maven Commands

* **mvn -version**: Finds the Maven version
* **mvn compile**: Compiles source files
* **mvn test-compile**: Compiles test files as well as source files
* **mvn clean**: Deletes target directory
* **mvn test**:  Runs unit tests
* **mvn package**: Creates a JAR for the project
* **help:effective-settings**: Debugs Maven settings
* **help:effective-pom**: Look at the complete pom after all inheritances from parent poms are resolved
* **dependency:tree**: Look sat all the dependencies and transitive dependencies
* **dependency:sources**: Downloads source code for all dependencies
* **-debug**: Debug flag; can be used with all the above commands

## Creating Spring Boot Projects With Eclipse and Maven

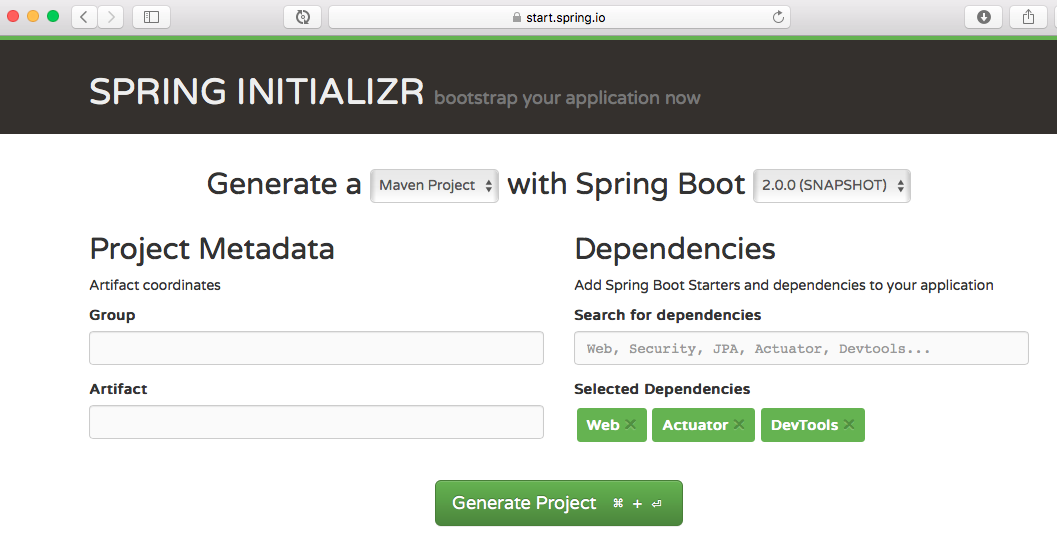
There are three options to create Spring Boot Projects with Eclipse and Maven

1. [Spring Initializr](https://start.spring.io/)
2. Use STS or STS Eclipse Plugin and create a Spring Boot Maven project directly from Eclipse
3. Manually create a Maven project and add Spring Boot starter dependencies

We will use a Spring Boot Starter Web as an example.

### Option 1: Bootstrap Spring Boot Project With Spring Initializr

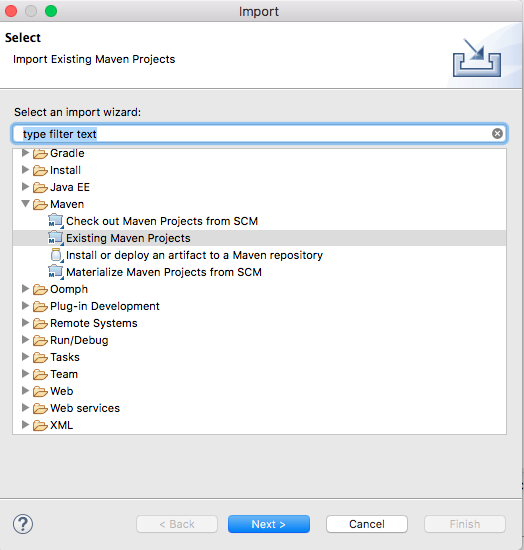
Creating a web application with Spring Initializr is a cake walk. We will use Spring Web MVC as our web framework. [Spring Initializr](http://start.spring.io/) is a great tool to bootstrap your Spring Boot projects.



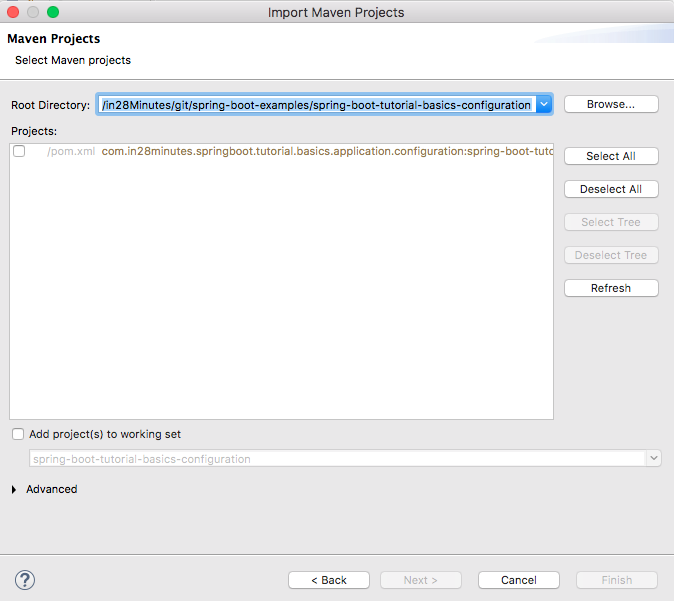
As shown in the image above, following steps have to be done:

* Launch Spring Initializr and choose the following
  + Choose com.in28minutes.springboot as **Group**
  + Choose student-services as **Artifact**
  + Choose from the following dependencies:
    - Web
    - Actuator
    - DevTools
* Click **Generate Project**.

This would download a ZIP file to your local machine. Unzip the zip file and extract to a folder. In Eclipse, Click **File** > **Import** > **Existing Maven Project** as shown below.



Navigate or type in the path of the folder where you extracted the ZIP file on the next screen.



Once you click **Finish**, Maven will take some time to download all the dependencies and initialize the project.

That's it. Your first Spring project is ready!

Follow these links to understand more about the project that we created:

* [Spring Boot vs. Spring vs. Spring MVC](http://www.springboottutorial.com/spring-boot-vs-spring-mvc-vs-spring)
* [Auto Configuration](http://www.springboottutorial.com/spring-boot-auto-configuration)
* [Spring Boot Starter Projects](http://www.springboottutorial.com/spring-boot-starter-projects)
* [Spring Boot Starter Parent](http://www.springboottutorial.com/spring-boot-starter-parent)
* [Spring Boot Initializr](http://www.springboottutorial.com/spring-initialzr-bootstrap-spring-boot-applications)

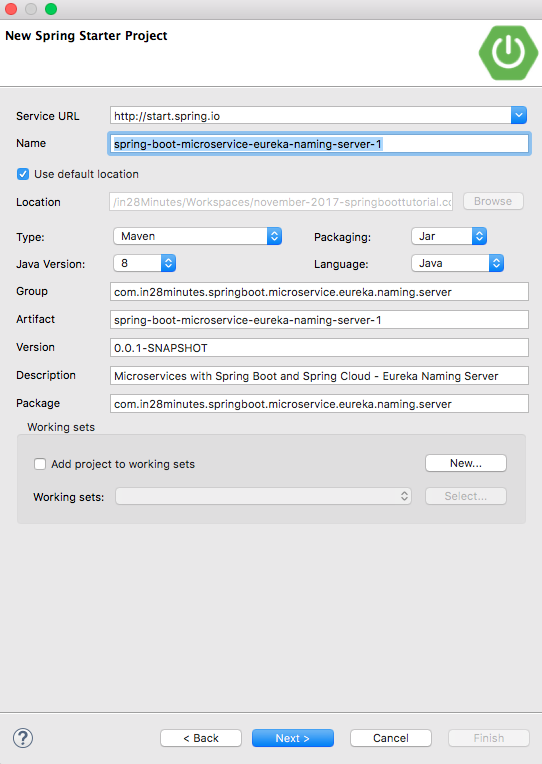
### Option 2: Use STS or STS Eclipse Plugin to Create Spring Boot Maven Project

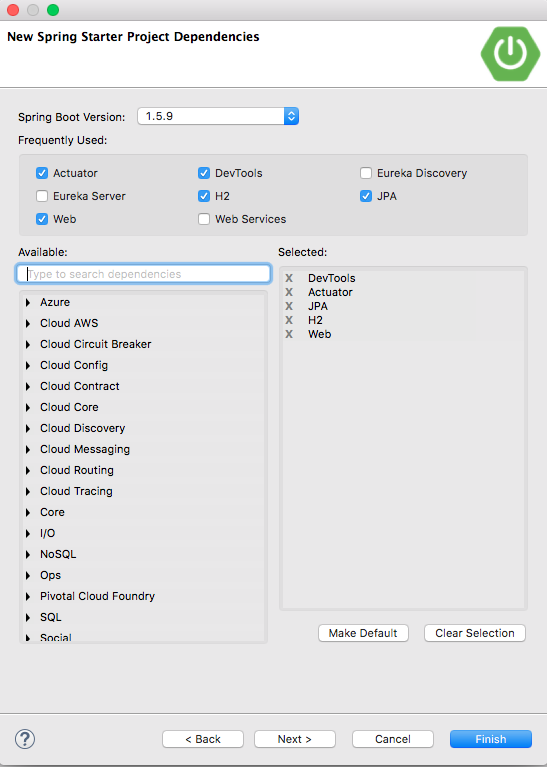
With the Spring tool suite, you can directly create a Spring Boot project from Eclipse. You should either download the complete installation of STS or you can install the STS Eclipse plugin. [This link](https://spring.io/tools/sts/all)provides the complete download of STS as well as the update sites for the STS Eclipse Plugin.

In Eclipse/STS, start with **File** > **New** > **Spring Starter Project**. In the next screen, you can choose the following for your project:

* Group ID
* Artifact ID
* Root package
* Version
* Description
* Java version
* Language
* Packaging

Make sure you choose **Maven** as **Type**.



In the next screen, you can choose the dependencies that you want to add to your Spring Boot project.

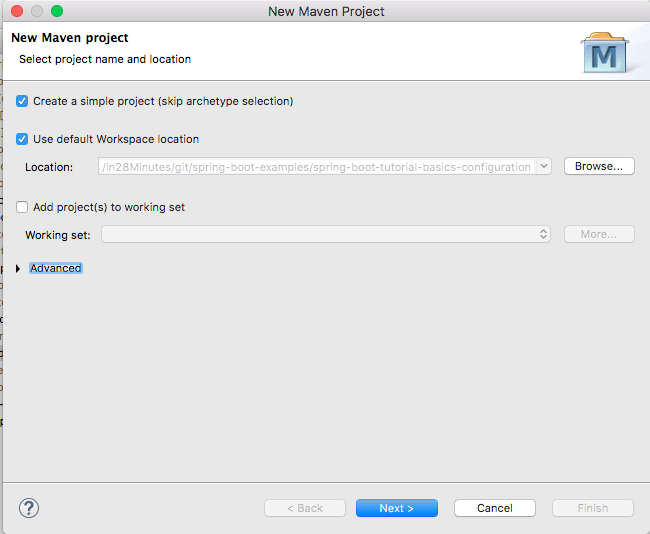
Once you click **Finish**, Maven will take some time to download all the dependencies and initialize the project.

Now your Spring project is ready!

### Option 3: Manually Create a Maven Spring Boot Project

The last option is to create the project manually.

In Eclipse, start with **File** > **New** > **MavenProject**.

Choose **Create a simple project**, as shown in the screenshot below:

In the next screen, provide these details for your project and click **Finish**.

* Group ID
* Artifact ID
* Version

This would create a basic Maven project with zero dependencies. Next, add in the appropriate Spring Boot starters into the pom.xml:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

Starter Web is used for developing Spring Boot Web applications or RESTful services. Starter Test provides unit testing and integration test capabilities with Spring Test, Mockito, and JUnit.

One this we are missing is the version for these dependencies. We will add a Spring Boot starter parent as the parent pom in the pom.xml:

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.0.0.M6</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

Let's configure the Java version to use 1.8:

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

</properties>

The next step is to create a Spring Boot application class that will be the launching point of the web application.

/src/main/java/com/in28minutes/springboot/tutorial/SpringBootWebApplication.java:

package com.in28minutes.springboot.tutorial;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringBootWebApplication {

public static void main(String[] args) {

SpringApplication.run(SpringBootWebApplication.class, args);

}

}

All you need to do is add @SpringBootApplication and use the SpringApplication.run() static method to launch the Spring application context.

When you launch this class as a Java application, you will see that an embedded Tomcat server launches and you are ready to add features to this application.

## Summary

In this article, we looked at the different options for creating Spring Boot projects with Maven and Eclipse. I love Option 2: creating the project directly from Eclipse using the STS plugin. But, you might have your own preference.

# What is Spring Boot Auto Configuration?

*Jan 28, 2017*  
*6 minute read*

This guide will help you understand Spring Boot Auto Configuration with examples. We will use a simple code example creating couple of simple rest services.

## You will learn

* Why do we need Auto Configuration?
* What is Auto Configuration?
* A few examples of Spring Boot Auto Configuration
* How is Auto Configuration implemented in Spring Boot?
* How to debug Auto Configuration?

## 10 Step Reference Courses

* [Spring Framework for Beginners in 10 Steps](https://courses.in28minutes.com/p/spring-framework-for-beginners)
* [Spring Boot for Beginners in 10 Steps](https://courses.in28minutes.com/p/spring-boot-for-beginners-in-10-steps)
* [Spring MVC in 10 Steps](https://www.youtube.com/watch?v=BjNhGaZDr0Y)
* [JPA and Hibernate in 10 Steps](https://courses.in28minutes.com/p/jpa-and-hibernate-tutorial-for-beginners-with-spring-boot)
* [Eclipse Tutorial for Beginners in 5 Steps](https://courses.in28minutes.com/p/eclipse-tutorial-for-beginners)
* [Maven Tutorial for Beginners in 5 Steps](https://courses.in28minutes.com/p/maven-tutorial-for-beginners-in-5-steps)
* [JUnit Tutorial for Beginners in 5 Steps](https://courses.in28minutes.com/p/junit-tutorial-for-beginners)
* [Mockito Tutorial for Beginners in 5 Steps](https://courses.in28minutes.com/p/mockito-for-beginner-in-5-steps)
* [Complete in28Minutes Course Guide](https://courses.in28minutes.com/p/in28minutes-course-guide)

## Tools you will need

* Maven 3.0+ is your build tool
* Your favorite IDE. We use Eclipse.
* JDK 1.8+

## Complete Maven Project With Code Examples

*Our Github repository has all the code examples -*[*https://github.com/in28minutes/in28minutes.github.io/tree/master/code-zip-files*](https://github.com/in28minutes/in28minutes.github.io/tree/master/code-zip-files)

* All other examples related to Restful Web Services
  + Website-springbootrestservices-all-examples.zip

## Why do we need Spring Boot Auto Configuration?

Spring based applications have a lot of configuration.

When we use Spring MVC, we need to configure component scan, dispatcher servlet, a view resolver, web jars(for delivering static content) among other things.

<bean

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

<property name="prefix">

<value>/WEB-INF/views/</value>

</property>

<property name="suffix">

<value>.jsp</value>

</property>

</bean>

<mvc:resources mapping="/webjars/\*\*" location="/webjars/"/>

Below code snippet shows typical configuration of a dispatcher servlet in a web application.

<servlet>

<servlet-name>dispatcher</servlet-name>

<servlet-class>

org.springframework.web.servlet.DispatcherServlet

</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/todo-servlet.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>dispatcher</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

When we use Hibernate/JPA, we would need to configure a datasource, an entity manager factory, a transaction manager among a host of other things.

<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"

destroy-method="close">

<property name="driverClass" value="${db.driver}" />

<property name="jdbcUrl" value="${db.url}" />

<property name="user" value="${db.username}" />

<property name="password" value="${db.password}" />

</bean>

<jdbc:initialize-database data-source="dataSource">

<jdbc:script location="classpath:config/schema.sql" />

<jdbc:script location="classpath:config/data.sql" />

</jdbc:initialize-database>

<bean

class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean"

id="entityManagerFactory">

<property name="persistenceUnitName" value="hsql\_pu" />

<property name="dataSource" ref="dataSource" />

</bean>

<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">

<property name="entityManagerFactory" ref="entityManagerFactory" />

<property name="dataSource" ref="dataSource" />

</bean>

<tx:annotation-driven transaction-manager="transactionManager"/>

Above examples are typical with any Spring framework implementation or integration with other frameworks.

## Spring Boot : Can we think different?

Spring Boot brings in new thought process around this.

*Can we bring more intelligence into this? When a spring mvc jar is added into an application, can we auto configure some beans automatically?*

* How about auto configuring a Data Source if Hibernate jar is on the classpath?
* How about auto configuring a Dispatcher Servlet if Spring MVC jar is on the classpath?

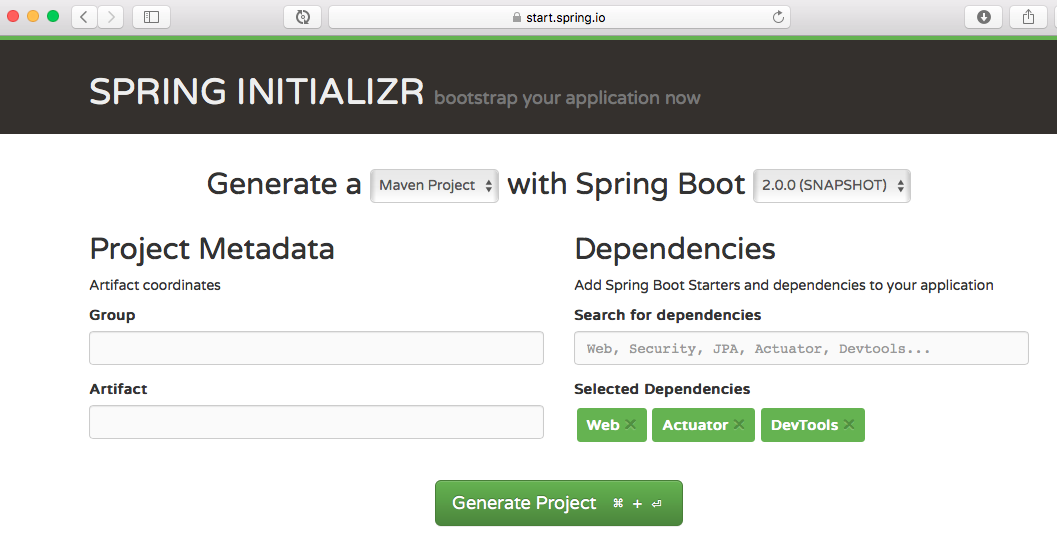
There would be provisions to override the default auto configuration.

*Spring Boot looks at a) Frameworks available on the CLASSPATH b) Existing configuration for the application. Based on these, Spring Boot provides basic configuration needed to configure the application with these frameworks. This is called Auto Configuration.*

To understand Auto Configuration further, lets bootstrap a simple Spring Boot Application using Spring Initializr.

## Creating REST Services Application with Spring Initializr

*Spring Initializr*[*http://start.spring.io/*](http://start.spring.io/)*is great tool to bootstrap your Spring Boot projects.*



As shown in the image above, following steps have to be done.

* Launch Spring Initializr and choose the following
  + Choose com.in28minutes.springboot as Group
  + Choose student-services as Artifact
  + Choose following dependencies
    - Web
    - Actuator
    - DevTools
* Click Generate Project.
* Import the project into Eclipse.
* If you want to understand all the files that are part of this project, you can go here.

### Spring Boot Auto Configuration in action.

When we run StudentServicesApplication.java as a Java Application, you will see a few important things in the log.

Mapping servlet: 'dispatcherServlet' to [/]

Mapped "{[/error]}" onto public org.springframework.http.ResponseEntity<java.util.Map<java.lang.String, java.lang.Object>> org.springframework.boot.autoconfigure.web.BasicErrorController.error(javax.servlet.http.HttpServletRequest)

Mapped URL path [/webjars/\*\*] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]

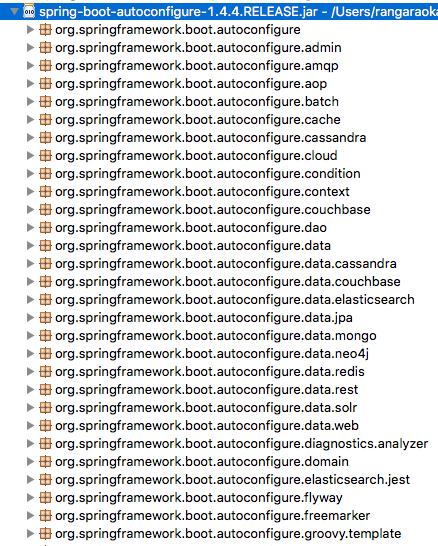
Above log statements are good examples of Spring Boot Auto Configuration in action.

As soon as we added in Spring Boot Starter Web as a dependency in our project, Spring Boot Autoconfiguration sees that Spring MVC is on the classpath. It autoconfigures dispatcherServlet, a default error page and webjars.

If you add Spring Boot Data JPA Starter, you will see that Spring Boot Auto Configuration auto configures a datasource and an Entity Manager.

## Where is Spring Boot Auto Configuration implemented?

All auto configuration logic is implemented in spring-boot-autoconfigure.jar. All auto configuration logic for mvc, data, jms and other frameworks is present in a single jar.



Other important file inside spring-boot-autoconfigure.jar is /META-INF/spring.factories. This file lists all the auto configuration classes that should be enabled under the EnableAutoConfiguration key. A few of the important auto configurations are listed below.

org.springframework.boot.autoconfigure.EnableAutoConfiguration=\

org.springframework.boot.autoconfigure.aop.AopAutoConfiguration,\

org.springframework.boot.autoconfigure.MessageSourceAutoConfiguration,\

org.springframework.boot.autoconfigure.PropertyPlaceholderAutoConfiguration,\

org.springframework.boot.autoconfigure.jackson.JacksonAutoConfiguration,\

org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration,\

org.springframework.boot.autoconfigure.jdbc.JdbcTemplateAutoConfiguration,\

org.springframework.boot.autoconfigure.jdbc.JndiDataSourceAutoConfiguration,\

org.springframework.boot.autoconfigure.jdbc.XADataSourceAutoConfiguration,\

org.springframework.boot.autoconfigure.jdbc.DataSourceTransactionManagerAutoConfiguration,\

org.springframework.boot.autoconfigure.security.SecurityAutoConfiguration,\

org.springframework.boot.autoconfigure.security.SecurityFilterAutoConfiguration,\

org.springframework.boot.autoconfigure.web.DispatcherServletAutoConfiguration,\

org.springframework.boot.autoconfigure.web.EmbeddedServletContainerAutoConfiguration,\

org.springframework.boot.autoconfigure.web.ErrorMvcAutoConfiguration,\

### Example Auto Configuration

We will take a look at DataSourceAutoConfiguration.

Typically all Auto Configuration classes look at other classes available in the classpath. If specific classes are available in the classpath, then configuration for that functionality is enabled through auto configuration. Annotations like @ConditionalOnClass, @ConditionalOnMissingBean help in providing these features!

@ConditionalOnClass({ DataSource.class, EmbeddedDatabaseType.class }) : This configuration is enabled only when these classes are available in the classpath.

@Configuration

@ConditionalOnClass({ DataSource.class, EmbeddedDatabaseType.class })

@EnableConfigurationProperties(DataSourceProperties.class)

@Import({ Registrar.class, DataSourcePoolMetadataProvidersConfiguration.class })

public class DataSourceAutoConfiguration {

@ConditionalOnMissingBean : This bean is configured only if there is no other bean configured with the same name.

@Bean

@ConditionalOnMissingBean

public DataSourceInitializer dataSourceInitializer() {

return new DataSourceInitializer();

}

Embedded Database is configured only if there are no beans of type DataSource.class or XADataSource.class already configured.

@Conditional(EmbeddedDatabaseCondition.class)

@ConditionalOnMissingBean({ DataSource.class, XADataSource.class })

@Import(EmbeddedDataSourceConfiguration.class)

protected static class EmbeddedDatabaseConfiguration {

}

## Debugging Auto Configuration

There are two ways you can debug and find more information about auto configuration.

* Turning on debug logging
* Using Spring Boot Actuator

### Debug Logging

You can turn debug logging by adding a simple property value to application.properties. In the example below, we are turning on Debug level for all logging from org.springframework package (and sub packages).

logging.level.org.springframework: DEBUG

When you restart the application, you would see an auto configuration report printed in the log. Similar to what you see below, a report is produced including all the auto configuration classes. The report separates the positive matches from negative matches. It will show why a specific bean is auto configured and also why something is not auto configured.

=========================

AUTO-CONFIGURATION REPORT

=========================

Positive matches:

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

DispatcherServletAutoConfiguration matched

- @ConditionalOnClass classes found: org.springframework.web.servlet.DispatcherServlet (OnClassCondition)

- found web application StandardServletEnvironment (OnWebApplicationCondition)

Negative matches:

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

ActiveMQAutoConfiguration did not match

- required @ConditionalOnClass classes not found: javax.jms.ConnectionFactory,org.apache.activemq.ActiveMQConnectionFactory (OnClassCondition)

AopAutoConfiguration.CglibAutoProxyConfiguration did not match

- @ConditionalOnProperty missing required properties spring.aop.proxy-target-class (OnPropertyCondition)

### Spring Boot Actuator

Other way to debug auto configuration is to add spring boot actuator to your project. We will also add in HAL Browser to make things easy.

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.data</groupId>

<artifactId>spring-data-rest-hal-browser</artifactId>

</dependency>

HAL Browser auto configuration <http://localhost:8080/actuator/#http://localhost:8080/autoconfig> would show the details of all the beans which are auto configured and those which are not.





Important Questions and Anseres

## Spring Interview Questions and Answers

Anyway let's see frequently asked questions on Spring framework on Java interviews :  
  
**1) What is spring framework? Why Java programmer should use Spring framework**  
Very common Spring interview question, Spring is a framework which helps Java programmer in development. Spring provides Dependency Injection and IOC container, Spring MVC flow and several useful API for Java programmer.  
  
 **2) What is default scope of bean in Spring framework? (**[**answer**](http://javarevisited.blogspot.sg/2012/05/what-is-bean-scope-in-spring-mvc.html)**)**  
The default scope of a Spring bean is the Singleton scope and in the web application default scope of a spring bean is request scope. Singleton bean means the same instance of a bean is shared with all other beans, while request scope means a bean is alive only for a request.  
  
  
  
**3) Does Spring singleton beans are thread-safe?**  
No, Spring singleton beans are not thread-safe. Singleton doesn't mean bean would be[thread-safe](http://javarevisited.blogspot.sg/2012/01/how-to-write-thread-safe-code-in-java.html).  
  
  
**4) What is Dependency Injection?**  
Dependency Injection is one of the design pattern, which allows injecting dependency on Object, instead of object resolving the dependency.  
  
  
**5) What is Inversion of Control concept, how does Spring support IOC?** ([answer](http://javarevisited.blogspot.sg/2012/12/inversion-of-control-dependency-injection-design-pattern-spring-example-tutorial.html))  
The simple meaning of inversion of the control means that now the framework, Spring is responsible for creating objects, wiring dependencies and managing their life-cycle instead of a developer, which was the case before. That's where control is inverted from developer to framework.  
  
  
**6) What is Spring MVC? Can you explain How one request is processed?** ([answer](http://javarevisited.blogspot.sg/2017/06/how-spring-mvc-framework-works-web-flow.html))  
Spring MVC is a framework to develop Java web application. It provides an implementation of MVC or Model View Controller architecture which is built on separation of concerns and makes the development of Java web application easy. In order to use this in your project, you need to learn Spring and include its JAR file.  
  
  
**7) How do you create a controller in Spring? @Controller vs @RestController?**([answer](http://javarevisited.blogspot.sg/2017/08/difference-between-restcontroller-and-controller-annotations-spring-mvc-rest.html))  
A controller is nothing but a class, also known as a bean in Spring terminology. If you are using annotation then you can create controller by using @Controller annotation. If you are developing RESTful web service then you can also create REST controllers by using @RestController annotation. See the answer for more detailed discussion.

# **Difference between @RestController and @Controller Annotation in Spring MVC and REST**

Posted by: [Javin Paul](https://www.javacodegeeks.com/author/javin-paul) in [Enterprise Java](https://www.javacodegeeks.com/category/java/enterprise-java) August 29th, 2017 [2 Comments](https://www.javacodegeeks.com/2017/08/difference-restcontroller-controller-annotation-spring-mvc-rest.html#comments) 3095 Views

The @RestController annotation in Spring MVC is nothing but a combination of the @Controller and the @ResponseBody annotation. It was added into Spring 4.0 to make the development of RESTful Web Services in Spring framework easier. If you are familiar with the [REST web services](http://www.java67.com/2017/04/3-great-books-to-learn-java-web-services-soap-and-restful.html)you know that the fundamental difference between a web application and a REST API is that the response from a web application is a generally view of HTML + CSS + JavaScript while REST API just return data in form of JSON or XML. This difference is also obvious in the @Controller and the @RestController annotation. The job of the @Controller is to create a Map of model object and find a view but the @RestController simply returns the object and object data is directly written into HTTP response as JSON or XML.

This can also be done with the traditional @Controller and the use of the @ResponseBody annotation but since this is the default behavior of RESTful Web services, Spring introduced @RestController which combined the behavior of @Controllerand @ResponseBody together.

In short, the following two code snippets are equal in Spring MVC:

|  |  |
| --- | --- |
| 1  2  3  4  5 | @Controller  @ResponseBody  public class MVCController {     .. your logic  } |
| 1  2  3  4 | @RestController  public class RestFulController {    .... your logic  } |

Obviously, everybody would like to declare just one annotation instead of two. Also, the @RestController is more obvious than the previous two.

## **What are @Controller and @RestController in Spring?**

In Spring framework, a Controller is a class which is responsible for preparing a model Map with data to be displayed by the view as well as choosing the right view itself. It can also directly write into response stream by using @ResponseBodyannotation and complete the request.

The behavior of writing directly into response stream is very useful for responding calls to RESTful web services because there we just return data instead of returning a view as explained in my earlier post about [how Spring MVC works internally](http://javarevisited.blogspot.sg/2017/06/how-spring-mvc-framework-works-web-flow.html).

If you have developed RESTful Web services before Spring 4 e.g. in Spring 3 or Spring 3.1, you would have been familiar by using a combination of the @Controller and the @ResponseBody to create a RESTful response. Spring guys take cognizant of this issues and created the @RestController.

Now, you don’t need to use the @Controller and the @RestponseBody annotation. Instead you can use the @RestControllerto provide the same functionality. In short, it is a convenience controller which combines the behavior of the @Controlerand the @Response body into one.

You can further join Eugen Paraschiv’s [REST with Spring Master class](http://courses.baeldung.com/p/rest-with-spring-the-master-class?affcode=22136_bkwjs9xa) if you are more curious about learning the advanced techniques to develop RESTFul Web Service in Spring.

## **Difference between @RestController and @Controller in Spring**

Now that, you are familiar with both of these annotations, it’s a good time to analyze some factual difference between the @RestController and the @Controler. This is a very important concept, not just from Interview point of view but also from Spring Core and Spring Web Application developer Certification. If you are preparing for [Spring certifications](http://javarevisited.blogspot.sg/2017/06/how-to-prepare-for-spring-framework-certifications.html), you should be familiar with such subtle differences. Additionally, you can also take a look at free Spring tests to get an idea about exam format and level of questions.

Anyway, let’s get back to the point, here are some important differences between these two annotations.

1. The @Controller is a common annotation which is used to mark a class as Spring MVC Controller while the @RestController is a special controller used in [RESTFul web services](http://javarevisited.blogspot.sg/2015/08/difference-between-soap-and-restfull-webservice-java.html" \t "_blank) and the equivalent of @Controller + @ResponseBody.
2. The @RestController is relatively new, added only on Spring 4.0 but @Controller is an old annotation, exists since Spring started supporting annotation, and officially it was added on Spring 2.5 version.
3. The @Controller annotation indicates that the class is a “Controller” e.g. a web controller while the @RestControllerannotation indicates that the class is a controller where @RequestMapping methods assume @ResponseBody semantics by default i.e. servicing REST API.
4. The @Controller is a specialization of @Component annotation while @RestController is a specialization of @Controllerannotation. It is actually a convenience controller annotated with @Controller and @ResponseBody as shown below.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | @Target(value=TYPE)  @Retention(value=RUNTIME)  @Documented  @Controller  @ResponseBody  public @interface RestController |

1. and here is how the declaration of @Controller looks like:

|  |  |
| --- | --- |
| 1  2  3  4  5 | @Target(value=TYPE)  @Retention(value=RUNTIME)  @Documented  @Component  public @interface Controller |

1. One of the key difference between @Controler and @RestCotroller in Spring MVC is that once you mark a class as @RestController then every method is written a domain object instead of a view. You can see Bryan Hassen’s [Introduction to Spring MVC 4](http://www.shareasale.com/m-pr.cfm?merchantID=53701&userID=880419&productID=557072989) to learn more about how to use the @RestController annotation in your Spring based application.
2. Another key difference between @RestController and @Controller is that you don’t need to use @ResponseBody on every handler method once you annotate the class with @RestController as shown below:

**with @RestControler:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | @RestController  public class Book{    @RequestMapping(value={"/book"})  public Book getBook(){  //...  return book;  }  } |

**without @RestController:**

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10 | @Controller  public class Book{    @RequestMapping(value={"/book"})  @ResponseBody  public Book getBook(){  //...  return book;  }  } |

You can see that if you use Spring MVC @Controller annotation to create a [RESTful response](http://javarevisited.blogspot.sg/2017/02/how-to-consume-json-from-restful-web-services-Spring-RESTTemplate-Example.html) you need to annotate each method with the @ResponseBody annotation, which is not required when you use @RestController. It not only makes your code more readable but also saves a couple of key strokes for you.

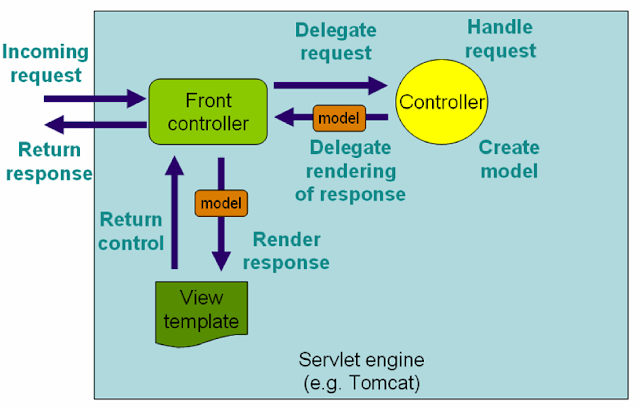
Here is a simple HelloWorld example using @RestController and **SpringBoot** framework:

That’s all about the **difference between @Controller and @RestController annotation in Spring MVC** and REST. @RestController is nothing but the shortcut to use both @Controller and @ResponseBody annotation together.

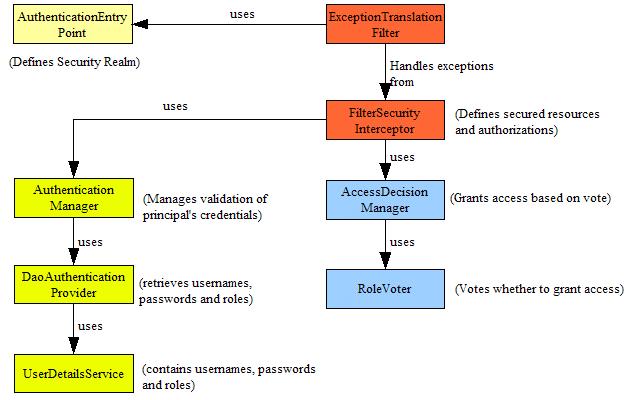
Spring purposefully added this annotation in Spring 4 to make the development of RESTful web services easier using Spring framework. It can directly convert the response to JSON or XML depending upon MIME type of request.

So, if you are creating a RESTful Web Services it’s better to use @RestController than combining the @Controller to @ResponseBody.

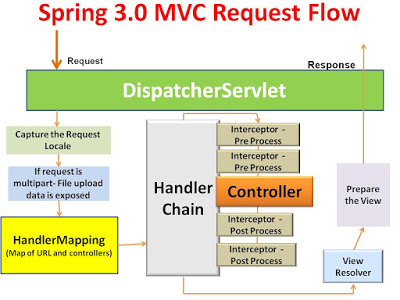
**8) What is view Resolver pattern? how it works in Spring MVC**  
View Resolver pattern is a J2EE pattern which allows a web application to dynamically choose it's view technology e.g. HTML, JSP, Tapestry, JSF, XSLT or any other view technology.  
  
In this pattern, View resolver holds mapping of different views, controller return name of the view, which is then passed to View Resolver for selecting an appropriate view.  
  
Spring MVC framework also supplies inbuilt view resolver for selecting views. See [Spring Master Class - Beginner to Expert](https://click.linksynergy.com/fs-bin/click?id=JVFxdTr9V80&subid=0&offerid=323058.1&type=10&tmpid=14538&RD_PARM1=https%3A%2F%2Fwww.udemy.com%2Fspring-tutorial-for-beginners%2F) to learn more about how Spring MVC internally works.

[](https://click.linksynergy.com/fs-bin/click?id=JVFxdTr9V80&subid=0&offerid=323058.1&type=10&tmpid=14538&RD_PARM1=https://www.udemy.com/spring-tutorial-for-beginners/)

**9) What is Spring Security?**  
Spring security is a project under spring framework umbrella, which provides support for security requirements of enterprise Java projects.  
  
Spring Security formerly known as aegis security provides out of box support for creating login screen, remember me cookie support, securing URL, authentication provider to authenticate the user from the database, LDAP and in memory, concurrent active session management support and much more.  
  
In order to use Spring security in a Spring MVC based project, you need to include spring-security.jar and configure it in application-Context-security.XML file, you can name it whatever you want, but make sure to supply this to ContextLoaderListener, which is responsible for creating Spring context and initializing dispatcher servlet.  
  
You can further see [Learn Spring Security Certification Class](https://courses.baeldung.com/p/learn-spring-security-the-certification-class?utm_source=javarevisited&utm_medium=web&utm_campaign=lss&affcode=22136_bkwjs9xa) learn more about authentication, authorization, and session control using Spring Security.

[](https://courses.baeldung.com/p/learn-spring-security-the-certification-class?utm_source=javarevisited&utm_medium=web&utm_campaign=lss&affcode=22136_bkwjs9xa)

**10) How do you control concurrent Session on Java web application using Spring Security?**  
You can use Spring Security to control a number of active session in Java web application. Spring security framework provides this feature out of the box and when enabled, a user can only have one active session at a time. See this Spring Security example to learn more about [How to control concurrent user session using Spring security](http://javarevisited.blogspot.sg/2012/03/spring-security-example-tutorial-how-to.html)  
  
 **11) What types of dependency injection is supported by Spring Framework? When do you use Setter and Constructor Injection, pros and cons?**  
There are 2 types of dependency injection supported by Spring, constructor based injection, and setter-based injection.  
  
Both types have their own advantages and disadvantages, you should use Constructor injection when object's dependencies are not optional and they must be initialized with their dependencies.  
  
Also, use constructor injection if the order of initialization or dependency matters because in Setter based injection you cannot impose any order. Use setter injection when dependencies are optional. See the [difference between setter and constructor injection in Spring](http://javarevisited.blogspot.sg/2012/11/difference-between-setter-injection-vs-constructor-injection-spring-framework.html) for a more detailed answer.  
  
  
  
12) What is the difference between ApplicationContext and BeanFactory in Spring framework? ([answer](http://javarevisited.blogspot.sg/2012/11/difference-between-beanfactory-vs-applicationcontext-spring-framework.html))  
  
  
13) How do you call a stored procedure by using Spring framework? ([answer](http://javarevisited.blogspot.sg/2013/04/spring-framework-tutorial-call-stored-procedures-from-java.html))  
  
  
14) What do JdbcTemplate and JmsTemplate class offer in Spring?  
  
  
15) Can we use more than one configuration file for our Spring project?  
  
  
16) Explain Spring MVC flow with a simple example e.g. starting from Container receives a request and forward to your Java application? ([answer](http://javarevisited.blogspot.sg/2017/06/how-spring-mvc-framework-works-web-flow.html))  
  
  
  
**17) What is the difference in Spring MVC and Spring core?**  
The Spring MVC is part of Spring framework which helps you to develop Java web application using model web controller pattern, while Spring Core provides the Dependency injection and Inversion of Control. The Spring Container is part of Spring core.  
  
Both functionalities come in different JAR files. If you are developing just a core Java application using Spring then you just need Spring Core but if you are developing Web application then you need spring-mvc.jar as well. See [Introduction to Spring MVC](https://pluralsight.pxf.io/c/1193463/424552/7490?u=https%3A%2F%2Fwww.pluralsight.com%2Fcourses%2Fspringmvc-intro) to learn more about Spring framework architecture and components.  
  
  
18) Can you use Spring MVC framework along with Struts? I have an existing Java MVC application which is based in Struts, Can I migrate that to use Spring MVC? How?  
  
  
  
19) What is the advantage of Spring MVC framework over Struts 1.0 or Struts 2.0 ? is it worth to convert an existing Struts application to Spring MVC?  
  
  
  
20) How does Spring resolve view returned by ModelAndView class?  
  
  
  
Some Spring MVC questions are tricky e.g. Struts and Spring integration and can be only answered by experienced Java program with 2 to 4-year experience in Spring MVC framework.  
  
  
**21) If a user checked in CheckBox and got a validation error in other fields and then he unchecked the CheckBox, what would be selection status in command object in Spring MVC? How do you fix this issue?**  
Since during HTTP post, if the checkbox is unchecked than HTTP does include a request parameter for checkbox, which means updated selection won't be picked up. you can use hidden form field, starting with \_ to fix this in Spring MVC. quite a tricky question to answer if you are not aware of HTTP POST behavior and Spring MVC.

[](https://pluralsight.pxf.io/c/1193463/424552/7490?u=https://www.pluralsight.com/courses/springmvc-intro)

**22) What are different implementations of View interface you have used in Spring MVC?**  
ULBased View e.g. JSP, JSTLView,  
  
  
**23) How to escape HTML special characters using Spring MVC?**  
There are some methods in Spring tag library, can't remember now.  
  
  
These were some of the **Core Spring framework and MVC Interview questions** from my collection, I have given short answers for most of these Spring interview question. I suggest to research more or read along those Spring question to prepare for follow-up Spring interview questions.  
  
Read more: <http://www.java67.com/2012/08/spring-interview-questions-answers.html#ixzz5WErqHsEV>

## 1. What is IoC – Inversion of Control

In traditional programming, the flow of the business logic is determined by objects that are statically assigned to one another. With inversion of control, the flow depends on the object graph that is instantiated by the assembler and is made possible by object interactions being defined through abstractions. The binding process is achieved through dependency injection, although some argue that the use of a service locator also provides inversion of control.

**Inversion of control as a design guideline** serves the following purposes:

1. There is a decoupling of the execution of a certain task from implementation.
2. Every module can focus on what it is designed for.
3. Modules make no assumptions about what other systems do but rely on their contracts.
4. Replacing modules has no side effect on other modules.

## 2. What is DI – Dependency Injection

IoC is a design paradigm with the goal of giving more control to the targeted components of your application, the ones getting the work done. While Dependency injection is a pattern used to create instances of objects that other objects rely on without knowing at compile time which class will be used to provide that functionality. IoC relies on dependency injection because a mechanism is needed in order to activate the components providing the specific functionality.

The two concepts work together in this way to allow for much more flexible, reusable, and encapsulated code to be written. As such, they are important concepts in designing object-oriented solutions.

## 3. How to implement IoC

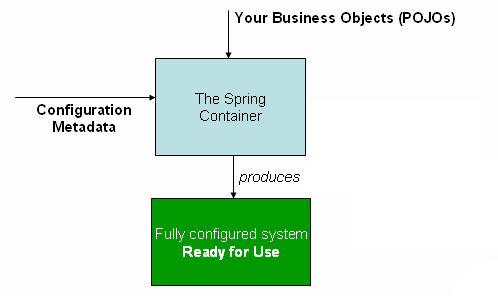
In object-oriented programming, there are several basic techniques to implement inversion of control. These are:

1. using a factory pattern
2. using a service locator pattern
3. using a **dependency injection** of any given below type:
   * a constructor injection
   * a setter injection
   * an interface injection

## 4. IoC in Spring Framework

The org.springframework.beans and org.springframework.context packages provide the basis for the Spring Framework’s IoC container. The [BeanFactory](https://docs.spring.io/spring/docs/1.2.x/javadoc-api/org/springframework/beans/factory/BeanFactory.html" \o "BeanFactory) interface provides an advanced configuration mechanism capable of managing objects of any nature. The [ApplicationContext](https://docs.spring.io/spring/docs/2.5.5/javadoc-api/org/springframework/context/ApplicationContext.html" \o "ApplicationContext) interface builds on top of the BeanFactory (it is a sub-interface) and adds other functionality such as easier integration with Spring’s AOP features, message resource handling (for use in internationalization), event propagation, and application-layer specific contexts such as the WebApplicationContext for use in web applications.

The **org.springframework.beans.factory.BeanFactory is the actual representation of the Spring IoC container** that is responsible for containing and otherwise managing the aforementioned beans. The BeanFactory interface is the central IoC container interface in Spring.

[](https://howtodoinjava.files.wordpress.com/2013/03/container-magic.png)

There are a number of implementations of the BeanFactory interface. The most commonly used BeanFactory implementation is the XmlBeanFactory class. Other commonly used class is XmlWebApplicationContext. Depending on the bean definition, the factory will return either an independent instance of a contained object (the Prototype design pattern), or a single shared instance (a superior alternative to the Singleton design pattern, in which the instance is a singleton in the scope of the factory). Which type of instance will be returned depends on the bean factory configuration: the API is the same.

Before we dive into **dependency injection types**, let first identify the ways of creating a bean in spring framework as it will help in understanding the things in next section.

## 5. How to create beans in Spring

A bean definition can be seen as a recipe for creating one or more actual objects. The container looks at the recipe for a named bean when asked, and uses the configuration metadata encapsulated by that bean definition to create (or acquire) an actual object.

#### 5.1. Using constructor

When creating a bean using the constructor approach, all normal classes are usable by and compatible with Spring. That is, the class being created does not need to implement any specific interfaces or be coded in a specific fashion. Just specifying the bean class should be enough. When using XML-based configuration metadata you can specify your bean class like so:

|  |
| --- |
| beans.xml |
| <bean id="exampleBean"/> |

#### 5.2. Using static factory method

When defining a bean which is to be created using a static factory method, along with the class attribute which specifies the class containing the static factory method, another attribute named factory-method is needed to specify the name of the factory method itself.

|  |
| --- |
| beans.xml |
| <bean id="exampleBean" factory-method="createInstance"/> |

Spring expects to be able to call this method and get back a live object, which from that point on is treated as if it had been created normally via a constructor.

#### 5.3. Using instance factory method

In a fashion similar to instantiation via a static factory method, instantiation using an instance factory method is where the factory method of an existing bean from the container is invoked to create the new bean.

|  |
| --- |
| beans.xml |
| <bean id="myFactoryBean"  class="...">    <bean id="exampleBean"  factory-bean="myFactoryBean" factory-method="createInstance"></bean> |

## 6. Dependency Injection in Spring Framework

The basic principle behind Dependency Injection (DI) is that objects define their dependencies only through constructor arguments, arguments to a factory method, or properties which are set on the object instance after it has been constructed or returned from a factory method. Then, it is the job of the container to actually inject those dependencies when it creates the bean. This is fundamentally the inverse, hence the name Inversion of Control (IoC).

#### 6.1. Setter Injection

Setter-based DI is realized by calling setter methods on your beans after invoking a no-argument constructor or no-argument static factory method to instantiate your bean.

|  |
| --- |
| TestSetterDI.java |
| public class TestSetterDI {    DemoBean demoBean = null;    public void setDemoBean(DemoBean demoBean) {      this.demoBean = demoBean;  }  } |

#### 6.2. Constructor Injection

Constructor-based DI is realized by invoking a constructor with a number of arguments, each representing a collaborator. Additionally, calling a static factory method with specific arguments to construct the bean, can be considered almost equivalent, and the rest of this text will consider arguments to a constructor and arguments to a static factory method similarly.

|  |
| --- |
| ConstructorDI.java |
| public class ConstructorDI {    DemoBean demoBean = null;    public TestSetterDI (DemoBean demoBean) {      this.demoBean = demoBean;  }  } |

#### 6.3. Interface Injection

In this methodology we implement an interface from the IOC framework. IOC framework will use the interface method to inject the object in the main class. It is much more appropriate to use this approach when you need to have some logic that is not applicable to place in a property. Such as logging support.

|  |
| --- |
| public void SetLogger(ILogger logger)  {    \_notificationService.SetLogger(logger);    \_productService.SetLogger(logger);  } |

## 7. IoC vs DI Interview Questions

#### 7.1. What is difference between component and service?

A component is a glob of software that’s intended to be used, without change, by an application that is out of the control of the writers of the component. By ‘without change’ means that the using application doesn’t change the source code of the components, although they may alter the component’s behavior by extending it in ways allowed by the component writers.

A service is similar to a component in that it’s used by foreign applications. The main difference is that a component to be used locally (think jar file, assembly, dll, or a source import). A service will be used remotely through some remote interface, either synchronous or asynchronous (eg web service, messaging system, RPC, or socket.)

#### 7.2. How DI is different from Service locator pattern?

The key benefit of a Dependency Injector is that it allows to plug-in a suitable implementation of a service according to environment and usage. Injection isn’t the only way to break this dependency, another is to use a service locator. The basic idea behind a service locator is to have an object that knows how to get hold of all of the services that an application might need. It then scans all such services and store them as a singleton Registry. When asked for a service implementation, a requester can query the registry with a token and get appropriate implementation.

Mostly these registries are populated via some configuration files. The key difference is that with a Service Locator every user of a service has a dependency to the locator. The locator can hide dependencies to other implementations, but you do need to see the locator.

#### 7.3. Which one should be better to use i.e. service locator or dependency injection?

Well, it as I already said that key difference is that with a Service Locator every user of a service has a dependency to the locator. It means you must know the details of service locator in terms of input and output. So, it actually becomes the deciding factor which pattern to choose from.

If it is easy and necessary to maintain registry information then go for service locator, or else simply use dependency injection as it does not bother the users of service with any per-requisites.

#### 7.4. Which is better constructor injection or setter injection?

The choice between setter and constructor injection is interesting as it mirrors a more general issue with object-oriented programming – should you fill fields in a constructor or with setters.  
Constructors with parameters give you a clear statement of what it means to create a valid object in an obvious place. If there’s more than one way to do it, create multiple constructors that show the different combinations. Another advantage with constructor initialization is that it allows you to clearly hide any fields that are immutable by simply not providing a setter. I think this is important – if something shouldn’t change then the lack of a setter communicates this very well. If you use setters for initialization, then this can become a pain.

But If you have a lot of constructor parameters things can look messy, particularly in languages without keyword parameters. If you have multiple ways to construct a valid object, it can be hard to show this through constructors, since constructors can only vary on the number and type of parameters. Constructors also suffer if you have simple parameters such as strings. With setter injection you can give each setter a name to indicate what the string is supposed to do. With constructors you are just relying on the position, which is harder to follow.

My preference is to start with constructor injection, but be ready to switch to setter injection as soon as the problems I’ve outlined above start to become a problem.

#### 7.5. What is Bean Factory ?

A BeanFactory is like a factory class that contains a collection of beans. The BeanFactory holds Bean Definitions of multiple beans within itself and then instantiates the bean whenever asked for by clients.

BeanFactory is able to create associations between collaborating objects as they are instantiated. This removes the burden of configuration from bean itself and the beans client. BeanFactory also takes part in the life cycle of a bean, making calls to custom initialization and destruction methods.

#### 7.6. What is Application Context?

A bean factory is fine to simple applications, but to take advantage of the full power of the Spring framework, you may want to move up to Springs more advanced container, the application context. On the surface, an application context is same as a bean factory.Both load bean definitions, wire beans together, and dispense beans upon request. But it also provides:

* A means for resolving text messages, including support for internationalization.
* A generic way to load file resources.
* Events to beans that are registered as listeners.

#### 7.7. What are the common implementations of the Application Context?

The three commonly used implementation of ApplicationContext are:

1. ClassPathXmlApplicationContext : It Loads context definition from an XML file located in the classpath, treating context definitions as classpath resources. The application context is loaded from the application’s classpath by using the code .

|  |
| --- |
| ApplicationContext context = new ClassPathXmlApplicationContext("bean.xml"); |

1. FileSystemXmlApplicationContext : It loads context definition from an XML file in the filesystem. The application context is loaded from the file system by using the code .

|  |
| --- |
| ApplicationContext context = new FileSystemXmlApplicationContext("bean.xml"); |

1. XmlWebApplicationContext : It loads context definition from an XML file contained within a web application.

#### 7.8. What should be used preferably BeanFactory or ApplicationContext?

A BeanFactory pretty much just instantiates and configures beans. An ApplicationContext also does that, and it provides the supporting infrastructure to enable lots of enterprise-specific features such as transactions and AOP.

In short, favor the use of an ApplicationContext.

In this tutorial, we learned the **difference between ioc and di in spring**.