**What is Spring Framework? \*\*\***

Spring Framework is a Java platform that provides comprehensive infrastructure support for developing Java applications. Spring handles the infrastructure so you can focus on your application.

Spring enables you to build applications from “plain old Java objects” (POJOs) and to apply enterprise services non-invasively to POJOs. This capability applies to the Java SE programming model and to full and partial Java EE.

**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:

1. **Lightweight** and very little overhead of using framework for our development.
2. **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.
3. Spring IoC container manages Spring Bean life cycle and project specific configurations such as JNDI lookup.
4. 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.
5. 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:

1. Reducing direct dependencies between different components of the application, usually Springs IoC container is responsible for initializing resources or beans and injects them as dependencies.
2. 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.
3. 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.
4. 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 on our project.
5. 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 the different framework for different requirements.

## What is Inversion of Control? \*\*\*

Inversion of Control is a principle in software engineering by which the control of objects or portions of a program is transferred to a container or framework. It’s most often used in the context of object-oriented programming.

By contrast with traditional programming, in which our custom code makes calls to a library, IoC enables a framework to take control of the flow of a program and make calls to our custom code. To enable this, frameworks use abstractions with additional behavior built in. **If we want to add our own behavior, we need to extend the classes of the framework or plugin our own classes.**

The advantages of this architecture are:

* decoupling the execution of a task from its implementation
* making it easier to switch between different implementations
* greater modularity of a program
* greater ease in testing a program by isolating a component or mocking its dependencies and allowing components to communicate through contracts

Inversion of Control can be achieved through various mechanisms such as: Strategy design pattern, Service Locator pattern, Factory pattern, and Dependency Injection (DI).

IoC is also known as dependency injection (DI). It is a process whereby objects define their dependencies, that is, the other objects they work with, only through constructor arguments, arguments to a factory method, or properties that are set on the object instance after it is constructed or returned from a factory method. The container then injects those dependencies when it creates the bean. This process is fundamentally the inverse, hence the name Inversion of Control (IoC), of the bean itself controlling the instantiation or location of its dependencies by using direct construction of classes, or a mechanism such as the Service Locator pattern.

**What do you understand by Dependency Injection? \*\*\***

Dependency Injection, an aspect of Inversion of Control (IoC), is a general concept stating that you do not create your objects manually but instead describe how they should be created. An IoC container will instantiate required classes if needed.

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.

**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.

**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.

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

Enterprise applications have some common cross-cutting concerns that are applicable to 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 are 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.

**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 or Servlet Filters.

**Pointcut**: Pointcut are regular expressions that are 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 a specific point in the application such as method execution, exception handling, changing object variable values etc. In Spring AOP a join point 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 the argument type is determined.

**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 shortcomings of Spring AOP is that it can be applied only to the beans created through Spring Context.

**What is Spring IoC Container? \*\*\***

The **Spring IoC container** is at the core of the Spring Framework. The container will create the objects, wire them together, configure them, and manage their complete life cycle from creation till destruction. The Spring container uses dependency injection (DI) to manage the components that make up an application.

**Spring provides following two distinct types of containers.**

1. BeanFactory container
2. ApplicationContext container

**Some of the useful ApplicationContext implementations that we use are:**

1. **AnnotationConfigApplicationContext**—Loads a Spring application context

from one or more Java-based configuration classes

1. **AnnotationConfigWebApplicationContext**—Loads a Spring web application

context from one or more Java-based configuration classes

1. **ClassPathXmlApplicationContext**—Loads a context definition from one or

more XML files located in the classpath, treating context-definition files as classpath

resources

1. **FileSystemXmlApplicationContext**—Loads a context definition from one or

more XML files in the filesystem

1. **XmlWebApplicationContext**—Loads context definitions from one or more

XML files contained in a web application

**What is a Spring Bean?**

The instance of any normal java class which form backbone of any application that is **initialized** **assembled**, and otherwise **managed** by a 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.

**Which is the best way of injecting beans and why? \*\*\***

The recommended approach is to use constructor **arguments for mandatory dependencies and setters for optional ones**. Constructor injection allows injecting values to immutable fields and makes testing easier.

### Differentiate between constructor injection and setter injection.

#### Constructor Injection vs Setter Injection

|  |  |  |
| --- | --- | --- |
| Properties | Setter Injection | Constructor Injection |
| Injecting Mechanism | Setter injection in Spring uses setter methods like setDependency() to inject dependency on any bean managed by Spring's IOC container | [Constructor injection uses constructor to inject dependency on any Spring-managed bean.](http://javarevisited.blogspot.sg/2012/01/what-is-constructor-overloading-in-java.html) |
| Readability | User POJO's setter method thus it is more readable in ApplicationContecxt file | Not much readable as it uses index positions to inject dependencies. One needs to check the java class to verify if the dependencies are injected properly |
| Ensuring DI | There is no gaurantee that all the dependenies are set using setter injection as while injecting if a bean is not injected we would not know | Ensures DI as constructor Injection does not allow you to construct object, until your dependencies are ready. |
| Security | Setter Injection can override the previous dependency. | Constructor Injection is secure in these matter because every time you call the constructor, a new object is gets created. |
| When to use | When there are lot of dependencies | When there are less dependencies |

**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 initializes 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.

### Differentiate between BeanFactory and ApplicationContext.

#### BeanFactory vs ApplicationContext

|  |  |
| --- | --- |
| BeanFactory | ApplicationContext |
| It is an interface defined in org.springframework.beans.factory.BeanFactory | It is an interface defined in org.springframework.context.ApplicationContext |
| It uses Lazy initialization | It uses Eager/ Aggressive initialization |
| It explicitly provides a resource object using the syntax | It creates and manages resource objects on its own |
| It doesn’t supports internationalization | It supports internationalization |
| It doesn’t supports annotation based dependency | It supports annotation based dependency |

**What are different ways to configure a class as Spring Bean? \*\*\***

There are three different ways to configure Spring Bean.

1. **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.gaurav.spring.beans.MyBean"></bean>

1. **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.gaurav.spring.main")

public class MyConfiguration {

@Bean

public MyService getService(){

return new MyService();

}

}

AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext(MyConfiguration.class);

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

1. **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.gaurav.spring" />

**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.

**What is Spring Bean life cycle?**

Spring Beans are initialized by Spring Container and all the dependencies are also injected. When the 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.

**What does the Spring bean lifecycle look like? \*\*\***

1 Spring instantiates the bean.

2 Spring injects values and bean references into the bean’s properties.

3 If the bean implements **BeanNameAware**, Spring passes the bean’s ID to the set-

BeanName() method.

4 If the bean implements **BeanFactoryAware**, Spring calls the setBeanFactory()

method, passing in the bean factory itself.

5 If the bean implements **ApplicationContextAware**, Spring calls the set-

ApplicationContext() method, passing in a reference to the enclosing application

context.

6 If the bean implements the **BeanPostProcessor** interface, Spring calls its post-

ProcessBeforeInitialization() method.

7 If the bean implements the **InitializingBean** interface, Spring calls its after-

PropertiesSet() method. Similarly, if the bean was declared with an initmethod,

then the specified initialization method is called.

8 If the bean implements **BeanPostProcessor**, Spring calls its postProcess-

AfterInitialization() method.

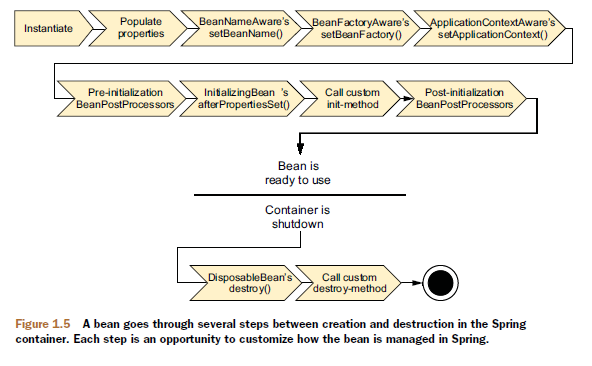
9 At this point, the bean is ready to be used by the application and remains in the

application context until the application context is destroyed.

10 If the bean implements the **DisposableBean** interface, Spring calls its

destroy() method. Likewise, if the bean was declared with a destroy-method,

the specified method is called.



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

**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

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;

**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 the spring framework also supports auto-wiring. 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.

**What are different types of Spring Bean autowiring? \*\*\***

There are four types of autowiring in Spring framework.

1. **Byname** - Attempts to match all properties of the autowired bean with beans that have the same name (or ID) as the properties.
2. **Bytype** - Attempts to match all properties of the autowired bean with beans whose types are assignable to the properties.
3. **Constructor** - Tries to match up a constructor of the autowired bean with beans whose types are assignable to the constructor arguments.
4. **Autodetect** - If you want to autowire your beans, but you can’t decide which type of autowiring to use, have no fear. You can set the autowire attribute to autodetect to let Spring make the decision for you.Spring make the decision for you.
5. **@Autowired** annotation – We can use Spring @Autowired annotation for spring bean autowiring. @Autowired annotation can be applied on variables and methods for autowiring byType. We can also use @Autowired annotation on constructor for constructor based spring autowiring.

For @Autowired 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 or by defining a bean of type oor.

1. **@Qualifier** annotation – This annotation is used to avoid conflicts in bean mapping and we need to provide the bean name that will be used for autowiring. This way we can avoid issues where multiple beans are defined for same type. This annotation usually works with the @Autowired annotation. For constructors with multiple arguments, we can use this annotation with the argument names in the method.

**What do you understand by auto wiring and name the different modes of it?**

The Spring container is able to autowire relationships between the collaborating beans. That is, it is possible to let Spring resolve collaborators for your bean automatically by inspecting the contents of the BeanFactory.  
Different modes of bean auto-wiring are:

1. no: This is default setting which means no autowiring. Explicit bean reference should be used for wiring.
2. byName: It injects the object dependency according to name of the bean. It matches and wires its properties with the beans defined by the same names in the XML file.
3. byType: It injects the object dependency according to type. It matches and wires a property if its type matches with exactly one of the beans name in XML file.
4. constructor: It injects the dependency by calling the constructor of the class. It has a large number of parameters.
5. autodetect: First the container tries to wire using autowire by constructor, if it can’t then it tries to autowire by byType.

**What are the limitations with auto wiring? \*\*\***

Following are some of the limitations you might face with auto wiring:

* Overriding possibility: You can always specify dependencies using <constructor-arg> and <property> settings which will override autowiring.
* Primitive data type: Simple properties such as primitives, Strings and Classes can’t be autowired.
* Confusing nature: Always prefer using explicit wiring because autowiring is less precise.

**Does Spring Bean provide thread safety?**

No, singleton beans are not thread-safe, as thread safety is about execution, whereas the singleton is a design pattern focusing on creation. Thread safety depends only on the bean implementation itself.

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.

**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 a 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.

**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.

**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.

**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.

**What is a MultipartResolver and when it’s 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.

**How to handle exceptions in Spring MVC Framework? \*\*\***

Spring MVC Framework provides the 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.

**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"/>

**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>

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

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

**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.

The 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.

**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.

**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 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 @ResponseBody annotation. 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.

**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

@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.

**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.

**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..

**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

**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.

**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.

**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>

**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 the 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>

**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.

**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.

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.

**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.

**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.gaurav.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;

**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: Bean Factory classes

Prototype Pattern: Bean scopes

Adapter Pattern: Spring Web and Spring MVC

Proxy Pattern: Spring Aspect Oriented Programming support

Template Method Pattern: JdbcTemplate, HibernateTemplate etc

Front Controller: Spring MVC DispatcherServlet

Data Access Object: Spring DAO support

Dependency Injection and Aspect Oriented Programming

**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, the 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 get 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 an actual benefit, just for the sake of loose-coupling don’t use it because it’s harder to maintain.