What are benefits of using spring?

Following is the list of few of the great benefits of using 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 using the technique Inversion of Control. The objects give their dependencies instead of creating or looking for dependent objects.
* **Aspect oriented (AOP):** Spring supports Aspect oriented programming and enables cohesive development by separating 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, which provides a great alternative to web frameworks such as Struts or other over engineered or less popular web frameworks.
* **Transaction Management:** Spring provides a consistent transaction management interface that can scale down to a local transaction (using a single database, for example) and scale up to global transactions (using JTA, for example).
* **Exception Handling:** Spring provides a convenient API to translate technology-specific exceptions (thrown by JDBC, Hibernate, or JDO, for example) into consistent, unchecked exceptions.

What are the different modules in Spring framework?

Following are the modules of the Spring framework:

* Core module
* Bean module
* Context module
* Expression Language module
* JDBC module
* ORM module
* OXM module
* Java Messaging Service(JMS) module
* Transaction module
* Web module
* Web-Servlet module
* Web-Struts module
* Web-Portlet module

What is 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.

What is Dependency Injection?

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.

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

Types of IoC are:

* **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.

Which DI would you suggest Constructor-based or setter-based DI?

Since you can mix both, Constructor- and Setter-based DI, it is a good rule of thumb to use constructor arguments for mandatory dependencies and setters for optional dependencies. Note that the use of a *@Required* annotation on a setter can be used to make setters required dependencies.

What is Spring IoC container?

The Spring IoC creates the objects, wire them together, configure them, and manage their complete lifecycle from creation till destruction. The Spring container uses dependency injection (DI) to manage the components that make up an application.

What are types of IoC containers? Explain them.

There are two types of IoC containers:

* **Bean Factory container:** This is the simplest container providing basic support for DI .The BeanFactory is usually preferred where the resources are limited like mobile devices or applet based applications
* **Spring ApplicationContext Container:** This container adds more enterprise-specific functionality such as the ability to resolve textual messages from a properties file and the ability to publish application events to interested event listeners.

Give an example of BeanFactory implementation.

The most commonly used BeanFactory implementation is the**XmlBeanFactory** class. This container reads the configuration metadata from an XML file and uses it to create a fully configured system or application.

What are the common implementations of the ApplicationContext?

The three commonly used implementation of 'Application Context' are:

* **FileSystemXmlApplicationContext:** This container loads the definitions of the beans from an XML file. Here you need to provide the full path of the XML bean configuration file to the constructor.
* **ClassPathXmlApplicationContext:** This container loads the definitions of the beans from an XML file. Here you do not need to provide the full path of the XML file but you need to set CLASSPATH properly because this container will look bean configuration XML file in CLASSPATH.
* **WebXmlApplicationContext:** This container loads the XML file with definitions of all beans from within a web application.

What is the difference between Bean Factory and ApplicationContext?

Following are some of the differences:

* Application contexts provide a means for resolving text messages, including support for i18n of those messages.
* Application contexts provide a generic way to load file resources, such as images.
* Application contexts can publish events to beans that are registered as listeners.
* Certain 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.

What does a bean definition contain?

The bean definition contains the information called configuration metadata which is needed for the container to know the followings:

* How to create a bean
* Bean's lifecycle details
* Bean's dependencies

How do you provide configuration metadata to the Spring Container?

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

* XML based configuration file.
* Annotation-based configuration
* Java-based configuration

How do you define a bean scope?

When defining a <bean> in Spring, you have the option of declaring a scope for that bean. For example, to force Spring to produce a new bean instance each time one is needed, you should declare the bean's scope attribute to be**prototype**. Similar way if you want Spring to return the same bean instance each time one is needed, you should declare the bean's scope attribute to be **singleton.**

What bean scopes does Spring support? Explain them.

The Spring Framework supports following five scopes, three of which are available only if you use a web-aware ApplicationContext.

* **singleton:** This scopes the bean definition to a single instance per Spring IoC container.
* **prototype:** This scopes a single bean definition to have any number of object instances.
* **request:** This scopes a bean definition to an HTTP request. Only valid in the context of a web-aware Spring ApplicationContext.
* **session:** This scopes a bean definition to an HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.
* **global-session:** This scopes a bean definition to a global HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

Are Singleton beans thread safe in Spring Framework?

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

Explain Bean lifecycle in Spring framework?

Following is sequence of a bean lifecycle in Spring:

* **Instantiate** - First the spring container finds the bean's definition from the XML file and instantiates the bean..
* **Populate properties** - Using the dependency injection, spring populates all of the properties as specified in the bean definition..
* **Set Bean Name** - If the bean implements BeanNameAware interface, spring passes the bean's id to setBeanName() method.
* **Set Bean factory** - If Bean implements BeanFactoryAware interface, spring passes the beanfactory to setBeanFactory() method.
* **Pre Initialization** - Also called postprocess of bean. If there are any bean BeanPostProcessors associated with the bean, Spring calls postProcesserBeforeInitialization() method.
* **Initialize beans** - If the bean implements IntializingBean,its afterPropertySet() method is called. If the bean has init method declaration, the specified initialization method is called.
* **Post Initialization** - If there are any BeanPostProcessors associated with the bean, their postProcessAfterInitialization() methods will be called.
* **Ready to use** - Now the bean is ready to use by the application.
* **Destroy** - If the bean implements DisposableBean , it will call the destroy() method .

What are inner beans in Spring?

A <bean/> element inside the <property/> or <constructor-arg/> elements defines a so-called inner bean. An inner bean definition does not require a defined id or name; the container ignores these values. It also ignores the scope flag. Inner beans are always anonymous and they are always scoped as prototypes.

How can you inject Java Collection in Spring?

Spring offers four types of collection configuration elements which are as follows:

* **<list>**: This helps in wiring i.e. injecting a list of values, allowing duplicates.
* **<set>**: This helps in wiring a set of values but without any duplicates.
* **<map>**: This can be used to inject a collection of name-value pairs where name and value can be of any type.
* **<props>**: This can be used to inject a collection of name-value pairs where the name and value are both Strings.

What is bean auto wiring?

The Spring container is able to autowire relationships between collaborating beans. This means that it is possible to automatically let Spring resolve collaborators (other beans) for your bean by inspecting the contents of the BeanFactory without using <constructor-arg> and <property> elements.

What are 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 which means no autowiring and you should use explicit bean reference for wiring. You have nothing to do special for this wiring. This is what you already have seen in Dependency Injection chapter.
* **byName**: Autowiring by property name. Spring container looks at the properties of the beans on which autowire attribute 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**: Autowiring by property datatype. Spring container looks at the properties of the beans on which autowire attribute 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**: 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.

What are the limitations with autowiring?

Limitations of autowiring are:

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

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.

How do you turn on annotation wiring?

Annotation wiring is not turned on in the Spring container by default. So, before we can use annotation-based wiring, we will need to enable it in our Spring configuration file by configuring <context:annotation-config/>.

What does @Required annotation mean?

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.

What does @Autowired annotation mean?

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

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

Java based configuration option enables you to write most of your Spring configuration without XML but with the help of few Java-based annotations.

For example: Annotation **@Configuration** indicates that the class can be used by the Spring IoC container as a source of bean definitions. The **@Bean**annotation tells Spring that a method annotated with @Bean will return an object that should be registered as a bean in the Spring application context.

What are ORM's Spring supports ?

Spring supports the following ORM's :

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

Brief introduction on work expression and technical experience

Sewnior tech lead,

2 onsore ,8 offshore

Grainger B2B sites

Version 5.2

Work experience

All modules

CS cockpit,

Flexisearch,

Punchouts,

Pricing,

Prior to hybris

Migration and datamodels?

Change dataModel of an attribute?

New attrinute to be created.

Items.xml

What migration recquired in that process?

Backup data first,

Write a flexi search query

FlexiSearch Query can we insert data?

E commerce-technical front

B2B –create a cart---(Cart controller)

After checkout

Order fulfillment

What steps are there in order process in hybris front?

**What web service we use?**

**Rest or soap**

**How hybris works with Rest webservices?**

**Omni channel commerce**

Hybris automatically generate classes?

What classes are generated?

Pricing systems?

Out of the box features for pricing?

whats your role in project?

Immplemntation partner

In which environment it is deployment—Cluster environment

How cronjob is run on cluster environment?

How can you control your cronjob on a specific node?

How are you managing cache in cluster environment?

How do we clear cache? hAC

Why do we clear cache if hybris is taking care of cache clearing?

Have you worked with workflow?

Have you ever created one?

How to create events?----where are they used?

How to create events-using spring