2016

[CORE SPRING AND SPRING MVC]

SAGAR CHEKE

1. What is IOC (or Dependency Injection)?

The basic concept of the Inversion of Control pattern also known as dependency injection is 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 (in the case of the spring framework, the IOC container) is then responsible for hooking it all up. Applying IOC, objects are given their dependencies at creation time by some external entity that coordinates each object in the system. That is, dependencies are injected into objects. So, IOC means an inversion of responsibility with regard to how an object obtains references to collaborating objects.

2. What are the different types of IOC (dependency injection)?

There are three types of dependency injection:

- **a)** **Constructor Injection:** Dependencies are provided as constructor parameters.
- **b)** **Setter Injection**: Dependencies are assigned through JavaBeans properties (ex: setter methods).
- **c)** **Interface Injection**: Injection is done through an interface.

Note: Spring supports only Constructor and Setter Injection

3. What are Spring beans?

The Spring Beans 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.

4. 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.

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

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

a) XML based configuration file.
b) Annotation-based configuration
c) Java-based configuration

6. What are the benefits of IOC (Dependency Injection)?

Benefits of IOC (Dependency Injection) are as follows:

a) Minimizes the amount of code in your application. With IOC containers you do not care about how services are created and how you get references to the ones you need. You can also easily add additional services by adding a new constructor or a setter method with little or no extra configuration.

b) Make your application more testable by not requiring any singletons or JNDI lookup mechanisms in your unit test cases. IOC containers make unit testing and switching implementations very easy by manually allowing you to inject your own objects into the object under test.

c) Loose coupling is promoted with minimal effort and least intrusive mechanism. The factory design pattern is more intrusive because components or services need to be requested explicitly whereas in IOC the dependency is injected into requesting piece of code. Also some containers promote the design to interfaces not to implementations design concept by encouraging managed objects to implement a well-defined service interface of your own.

d) IOC containers support eager instantiation and lazy loading of services. Containers also provide support for instantiation of managed objects, cyclical dependencies, life cycles management, and dependency resolution between managed objects etc.

7. What is spring?

Spring is an open source framework created to address the complexity of enterprise application development. One of the chief advantages of the spring framework is its layered architecture, which allows you to be selective about which of its components you use while also providing a cohesive framework for J2EE application development.

8. What are the advantages of spring framework?

The advantages of spring are as follows:

- **a)** Spring has layered architecture. Use what you need and leave you don't need now.
- **b)** Spring Enables POJO Programming. POJO programming enables continuous integration and testability.
- **c)** Dependency Injection and Inversion of Control Simplifies JDBC
- **d)** Open source and no vendor lock-in.

9. What are features of spring?

a) **Lightweight:**
Spring is lightweight when it comes to size and transparency. The basic version of spring framework is around 1MB and the processing overhead is also very negligible.

b) **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.

c) **Aspect oriented (AOP):**
Spring supports Aspect oriented programming and enables cohesive development by separating application business logic from system services.

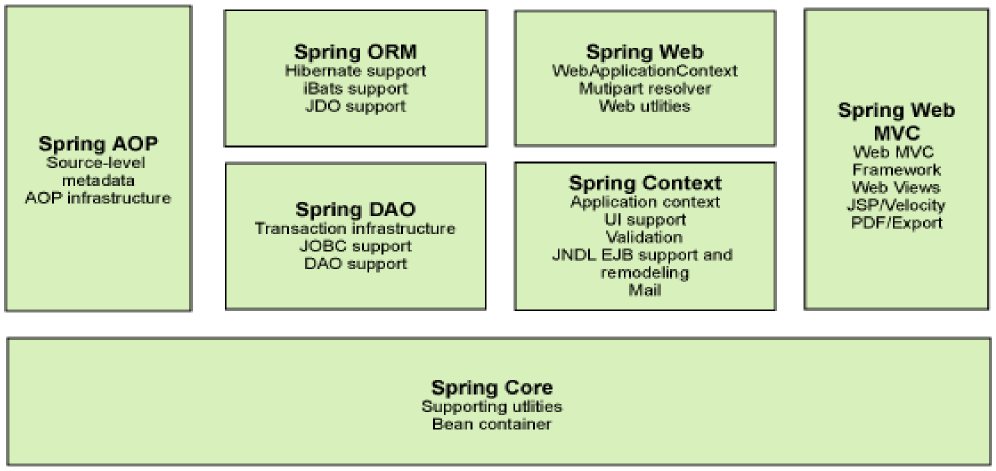
d) **Container:**
Spring contains and manages the life cycle and configuration of application objects.

e) **MVC Framework:**
Spring comes with MVC web application framework, built on core spring functionality. This framework is highly configurable via strategy interfaces, and accommodates multiple view technologies like JSP, Velocity, Tiles, iText, and POI. But other frameworks can be easily used instead of Spring MVC Framework.

f) **Transaction Management:**
Spring framework provides a generic abstraction layer for transaction management. This allowing the developer to add the pluggable transaction managers and making it easy to demarcate transactions without dealing with low-level issues. Spring's transaction support is not tied to J2EE environments and it can be also used in container less environments.

g) **JDBC Exception Handling:**
The JDBC abstraction layer of the spring offers a meaningful exception hierarchy, which simplifies the error handling strategy. Integration with Hibernate, JDO, and iBATIS: Spring provides best Integration services with Hibernate, JDO and iBATIS.

10. How many modules are there in spring? What are they?



Spring comprises of seven modules. They are.

a) **The core container:**
The core container provides the essential functionality of the spring framework. A primary component of the core container is the BeanFactory, an implementation of the Factory pattern. The BeanFactory applies the Inversion of Control (IOC) pattern to separate an application's configuration and dependency specification from the actual application code.

b) **Spring context:**
The spring context is a configuration file that provides context information to the spring framework. The spring context includes enterprise services such as JNDI, EJB, e-mail, internalization, validation and scheduling functionality.

c) **Spring AOP:**
The Spring AOP module integrates aspect-oriented programming functionality directly into the spring framework, through its configuration management feature. As a result you can easily AOP-enable any object managed by the spring framework. The Spring AOP module provides transaction management services for objects in any Spring-based application. With Spring AOP you can incorporate declarative transaction management into your applications without relying on EJB components.

d) **Spring DAO:**
The Spring JDBC DAO abstraction layer offers a meaningful exception hierarchy for managing the exception handling and error messages thrown by different database vendors. The exception hierarchy simplifies error handling and greatly reduces the amount of exception code you need to write, such as opening and closing connections. Spring DAO's JDBC-oriented exceptions comply with its generic DAO exception hierarchy.

e) **Spring ORM:**
The spring framework plugs into several ORM frameworks to provide its Object Relational tool, including JDO, Hibernate, and iBatis SQL Maps. All of these comply with spring’s generic transaction and DAO exception hierarchies.

f) **Spring Web module:**
The Web context module builds on top of the application context module, providing contexts for Web-based applications. As a result, the spring framework supports integration with Jakarta Struts. The Web module also eases the tasks of handling multi-part requests and binding request parameters to domain objects.

g) **Spring MVC framework:**

The Model-View-Controller (MVC) framework is a full-featured MVC implementation for building Web applications. The MVC framework is highly configurable via strategy interfaces and accommodates numerous view technologies including JSP, Velocity, Tiles, iText, and POI.

11. What are the types of Dependency Injection Spring supports?

a) **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.

b) **Constructor Injection:**
Constructor-based DI is realized by invoking a constructor with a number of arguments, each representing a collaborator.

12. 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.

a) 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.
b) BeanFactory also takes part in the life cycle of a bean, making calls to custom initialization and destruction methods.

13. 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) A means for resolving text messages, including support for internationalization.
b) A generic way to load file resources.
c) Events to beans that are registered as listeners.

14. What is the difference between Bean Factory and Application Context?

On the surface, an application context is same as a bean factory. But application context offers much more.

a) Application contexts provide a means for resolving text messages, including support for i18n of those messages.
b) Application contexts provide a generic way to load file resources, such as images.
c) Application contexts can publish events to beans that are registered as listeners.
d) 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.
e) ResourceLoader support: Spring’s Resource interface use a flexible generic abstraction for handling low-level resources. An application context itself is a ResourceLoader, hence provides an application with access to deployment-specific Resource instances.
f) MessageSource support: The application context implements MessageSource, an interface used to obtain localized messages, with the actual implementation being pluggable.

15. What are the common implementations of the Application Context?

The three commonly used implementation of 'Application Context' are

a) **ClassPathXmlApplicationContext:** It loads context definition from an XML file located in the class path, treating context definitions as class path resources. The application context is loaded from the application's class path by using the code.
ApplicationContext context = new ClassPathXmlApplicationContext("bean.xml");

b) **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");

c) **XmlWebApplicationContext:** It loads context definition from an XML file contained within a web application.

16. How is a typical spring implementation look like?

For a typical Spring Application, we need the following files:

a) An interface that defines the functions.
b) An Implementation that contains properties, its setter and getter methods, functions etc.
c) Spring AOP (Aspect Oriented Programming)
d) A XML file called spring configuration file.
e) Client program that uses the function.

17. What is the typical Bean life cycle in Spring Bean Factory Container?

Bean life cycle in Spring Bean Factory Container is as follows:

a) The spring container finds the bean's definition from the XML file and instantiates the bean.
b) Using the dependency injection, spring populates all of the properties as specified in the bean definition
c) If the bean implements the BeanNameAware interface, the factory calls setBeanName() passing the bean's ID.
d) If the bean implements the BeanFactoryAware interface, the factory calls setBeanFactory(), passing an instance of itself.
e) If there are any BeanPostProcessors associated with the bean, their post- ProcessBeforeInitialization() methods will be called.
f) If an init-method is specified for the bean, it will be called.
g) Finally, if there are any BeanPostProcessors associated with the bean, their postProcessAfterInitialization () methods will be called.

18. What do you mean by Bean wiring?

The act of creating associations between application components (beans) within the spring container is referred to as Bean wiring.

19. What do you mean by Auto Wiring?

The spring container is able to auto wire 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. The auto wiring functionality has five modes.

a) **no:** This is default setting. Explicit bean reference should be used for wiring.
b) **byName:** When autowiring byName, the 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
c) **byType:** When autowiring by datatype, the 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.
d) **constructor:** This mode is like 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.
e) **autodetect:** Spring first tries to wire using autowire by constructor, if it does not work, Spring tries to autowire by byType.

20. *Are there limitations with autowiring?*

Limitations of autowiring are:

a) Overriding: You can still specify dependencies using <constructor-arg> and <property> settings which will always override autowiring.

b) Primitive data types: You cannot autowire simple properties such as primitives, Strings and Classes.

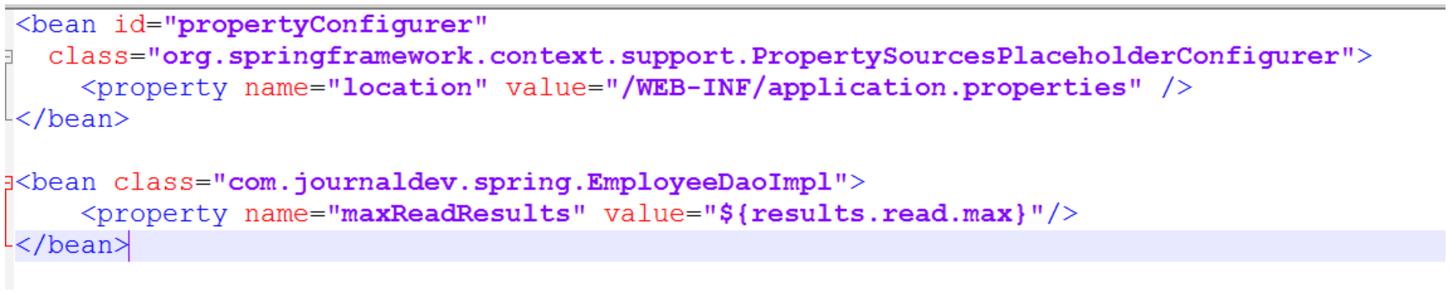
c) Confusing nature: Autowiring is less exact than explicit wiring, so if possible prefer using explicit wiring.

21. *Can you inject null and empty string values in spring?*

Yes, you can.

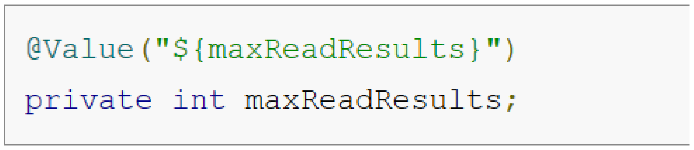
22. *How to inject a java.util.Properties into a Spring Bean?*

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;
```

23. *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:

a) Singleton Pattern: Creating beans with default scope.

b) Factory Pattern: Bean Factory classes

c) Prototype Pattern: Bean scopes

d) Adapter Pattern: Spring Web and Spring MVC

e) Proxy Pattern: Spring Aspect Oriented Programming support

f) Template Method Pattern: JdbcTemplate, HibernateTemplate etc

g) Front Controller: Spring MVC DispatcherServlet

h) Data Access Object: Spring DAO support

i) Dependency Injection and Aspect Oriented Programming

24. *What are ORM's spring supports?*

Spring supports the following ORM's:

a) Hibernate

b) iBatis

c) JPA (Java Persistence API)

d) TopLink

e) JDO (Java Data Objects)

f) OJB

25. What are the ways to access Hibernate using spring?

There are two approaches to Spring's Hibernate integration:

a) Inversion of Control with a HibernateTemplate and Callback
b) Extending HibernateDaoSupport and Applying an AOP Interceptor

26. How to integrate Spring and Hibernate using HibernateDaoSupport?

Spring and Hibernate can integrate using Spring's SessionFactory called LocalSessionFactory. The integration process is of 3 steps.

a) Configure the Hibernate SessionFactory
b) Extend your DAO Implementation from HibernateDaoSupport
c) Wire in Transaction Support with AOP

For Example:

```
<!-- Hibernate 4 SessionFactory Bean definition -->
<beans:bean id="hibernate4AnnotatedSessionFactory" class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
    <beans:property name="dataSource" ref="dataSource"/>
    <beans:property name="annotatedClasses">
        <beans:list>
            <beans:value>com.journaldev.spring.model.Person</beans:value>
        </beans:list>
    </beans:property>
    <beans:property name="hibernateProperties">
        <beans:props>
            <beans:prop key="hibernate.dialect">org.hibernate.dialect.Oracle9Dialect</beans:prop>
            <beans:prop key="hbm2ddl.auto">update</beans:prop>
            <beans:prop key="hibernate.show_sql">true</beans:prop>
            <beans:prop key="hibernate.default_schema">ADMIN</beans:prop>
        </beans:props>
    </beans:property>

</beans:bean>
```

27. What are Bean scopes in Spring Framework?

The Spring Framework supports exactly five scopes (of which three are available only if you are using a web-aware ApplicationContext). The scopes supported are listed below:

Scope	Description
singleton	Scopes a single bean definition to a single object instance per Spring IoC container.
prototype	Scopes a single bean definition to any number of object instances.
request	Scopes a single bean definition to the lifecycle of a single HTTP request; that is each HTTP request will have its own instance of a bean created off the back of a single bean definition. Only valid in the context of a web-aware Spring ApplicationContext.
session	Scopes a single bean definition to the lifecycle of a HTTP Session. Only valid in the context of a web-aware Spring ApplicationContext.
global session	Scopes a single bean definition to the lifecycle of a global HTTP Session. Typically, only valid when used in a portlet context. Only valid in the context of a web-aware Spring ApplicationContext.

28. Is Singleton beans thread safe in Spring Framework?

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

29. What are the methods of bean life cycle?

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 correspondent annotations (@PostConstruct and @PreDestroy).

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

Spring offers the following types of collection configuration elements:

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

31. *What are the different types of events of Listeners?*

Following are the different types of events of listeners:

a) ContextClosedEvent – This event is called when the context is closed.
b) ContextRefreshedEvent – This event is called when context is initialized or refreshed
c) RequestHandledEvent – This event is called when the web context handles request

32. *Annotation Based configuration*

Starting from Spring 2.5 it became possible to configure the dependency injection using annotations. So instead of using XML to describe a bean wiring, you can move the bean configuration into the component class itself by using annotations on the relevant class, method or field declaration.
Annotation injection is performed before XML injection; thus the latter configuration will override the former for properties wired through both approaches. 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. So consider to have following configuration file in case you want to use any annotation in your Spring application.

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

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">

   <context:annotation-config/>
   <!-- bean definitions go here -->

</beans>
```

a) **@Required:** The @Required annotation applies to bean property setter methods and it indicates that the affected bean property must be populated in XML configuration file at configuration time otherwise the container throws a BeanInitializationException.
b) **@Autowired:** The **@Autowired** 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.
    i. **@Autowired on Setter Methods:** You can use @Autowired annotation on setter methods to get rid of the <property> element in XML configuration file. When Spring finds an @Autowired annotation used with setter methods it tries to perform **byType** autowiring on the method.

ii. **@Autowired on Properties:** You can use @Autowired annotation on properties to get rid of the setter methods. When you will pass values of autowired properties using <property> Spring will automatically assign those properties with the passed values or references

iii. **@Autowired on Constructors:** You can apply @Autowired to constructors as well. A constructor @Autowired annotation indicates that the constructor should be autowired when creating the bean even if no <constructor-arg> elements are used while configuring the bean in XML file

iv. **@Autowired with (required=false) option:** By default, the @Autowired annotation implies the dependency is required similar to @Required annotation, however, you can turn off the default behavior by using (required=false) option with @Autowired.

```
@Autowired(required=false)
public void setAge(Integer age) {
   this.age = age;
}
```

c) **@Qualifier:** There may be a situation when you create more than one bean of the same type and want to wire only one of them with a property in such case you can use @Qualifier annotation along with @Autowired to remove the confusion by specifying which exact bean will be wired. Below is an example to show the use of @Qualifier annotation.

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

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">

   <context:annotation-config/>

   <!-- Definition for profile bean -->
   <bean id="profile" class="com.tutorialspoint.Profile">
   </bean>

   <!-- Definition for student1 bean -->
   <bean id="student1" class="com.tutorialspoint.Student">
      <property name="name"  value="Zara" />
      <property name="age"  value="11"/>
   </bean>

   <!-- Definition for student2 bean -->
   <bean id="student2" class="com.tutorialspoint.Student">
      <property name="name"  value="Nuha" />
      <property name="age"  value="2"/>
   </bean>

</beans>
```

```
public class Profile {
   @Autowired
   @Qualifier("student1")
   private Student student;
```

33. Java based configuration

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

@Configuration & @Bean Annotations: Annotating a class with the @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.

```
package com.tutorialspoint;
import org.springframework.context.annotation.*;

@Configuration
public class HelloWorldConfig {

   @Bean
   public HelloWorld helloWorld(){
      return new HelloWorld();
   }
}
```

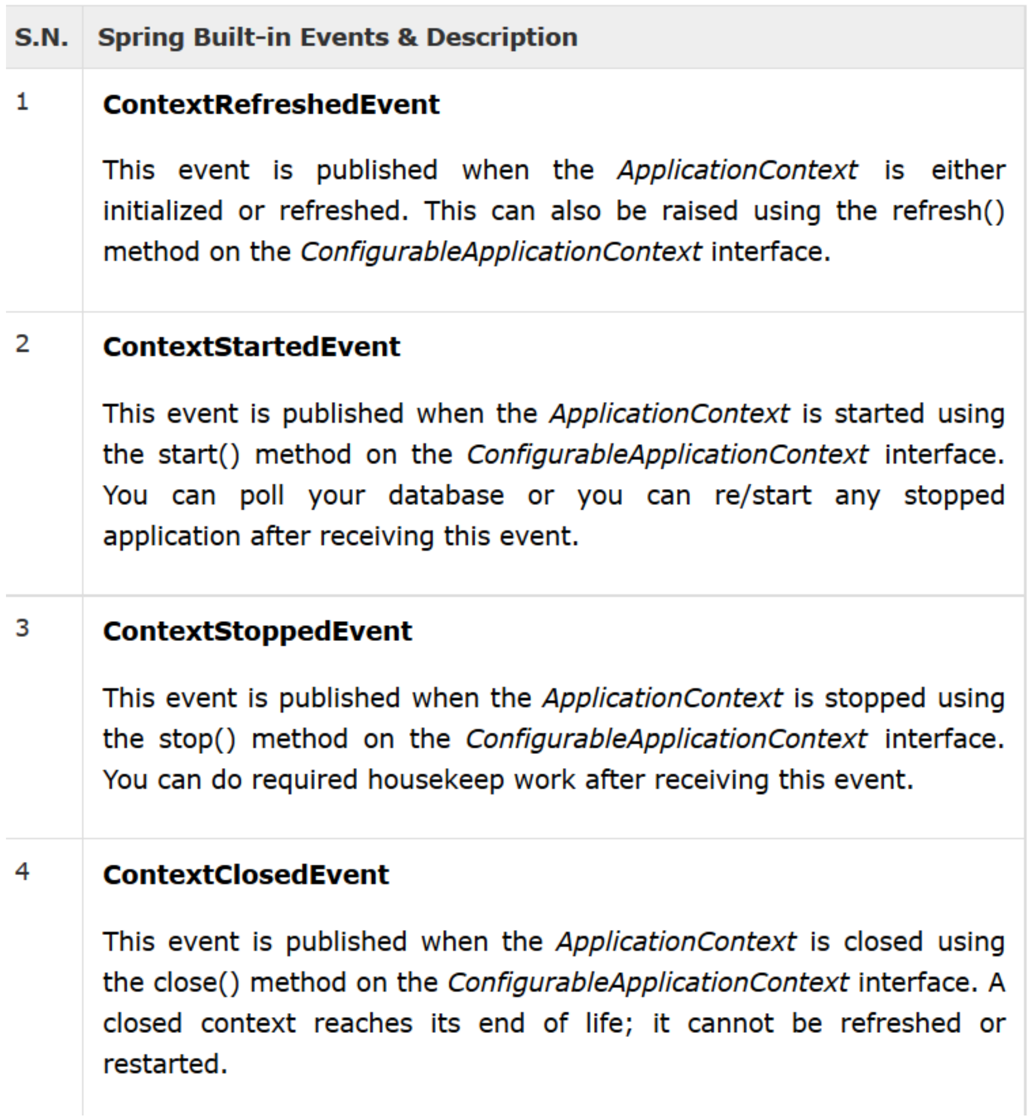
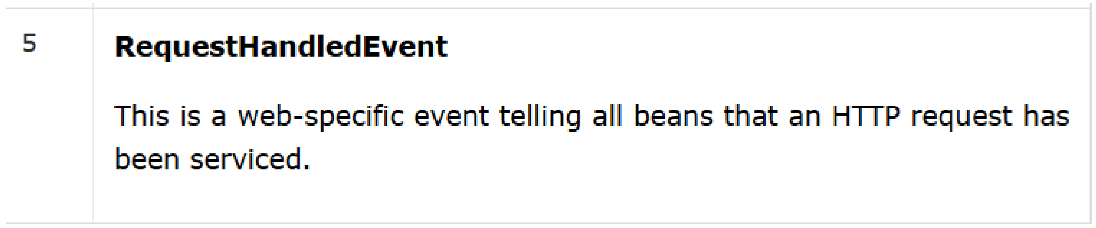
Here the method name annotated with @Bean works as bean ID and it creates and returns actual bean. Your configuration class can have declaration for more than one @Bean. Once your configuration classes are defined you can load and provide them to Spring container using AnnotationConfigApplicationContext

```
public static void main(String[] args) {
   ApplicationContext ctx =
   new AnnotationConfigApplicationContext(HelloWorldConfig.class);

   HelloWorld helloWorld = ctx.getBean(HelloWorld.class);

   helloWorld.setMessage("Hello World!");
   helloWorld.getMessage();
}
```

34. *Event Handling in Spring*

The ApplicationContext publishes certain types of events when loading the beans. For example, a ContextStartedEvent is published when the context is started and ContextStoppedEvent is published when the context is stopped.

Event handling in the ApplicationContext is provided through the ApplicationEvent class and ApplicationListener interface. So if a bean implements the ApplicationListener, then every time an ApplicationEvent gets published to the ApplicationContext, that bean is notified.

S.N.	Spring Built-in Events & Description
1	**ContextRefreshedEvent** This event is published when the *ApplicationContext* is either initialized or refreshed. This can also be raised using the refresh() method on the *ConfigurableApplicationContext* interface.
2	**ContextStartedEvent** This event is published when the *ApplicationContext* is started using the start() method on the *ConfigurableApplicationContext* interface. You can poll your database or you can re/start any stopped application after receiving this event.
3	**ContextStoppedEvent** This event is published when the *ApplicationContext* is stopped using the stop() method on the *ConfigurableApplicationContext* interface. You can do required housekeep work after receiving this event.
4	**ContextClosedEvent** This event is published when the *ApplicationContext* is closed using the close() method on the *ConfigurableApplicationContext* interface. A closed context reaches its end of life; it cannot be refreshed or restarted.
5	**RequestHandledEvent** This is a web-specific event telling all beans that an HTTP request has been serviced.

35. Custom Events in Spring

a) Create an event class, CustomEvent by extending ApplicationEvent. This class must define a default constructor which should inherit constructor from ApplicationEvent class.
b) Once your event class is defined, you can publish it from any class, let us say EventClassPublisher which implements ApplicationEventPublisherAware. You will also need to declare this class in XML configuration file as a bean so that the container can identify the bean as an event publisher because it implements the ApplicationEventPublisherAware interface.
c) A published event can be handled in a class, let us say EventClassHandler which implements ApplicationListener interface and implements onApplicationEvent method for the custom event.
d) Create beans configuration file Beans.xml under the src folder and a MainApp class which will work as spring application.

36. *Spring - JDBC Framework*

JdbcTemplate Class: The JdbcTemplate class executes SQL queries, update statements and stored procedure calls, performs iteration over ResultSets and extraction of returned parameter values. It also catches JDBC exceptions and translates them to the generic, more informative, exception hierarchy defined in the org.springframework.dao package. Instances of the JdbcTemplate class are thread safe once configured. So you can configure a single instance of a JdbcTemplate and then safely inject this shared reference into multiple DAOs.
A common practice when using the JdbcTemplate class is to configure a DataSource in your Spring configuration file, and then dependency-inject that shared DataSource bean into your DAO classes, and the JdbcTemplate is created in the setter for the DataSource.

```
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
    <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe" />
    <property name="username" value="sys as sysdba" />
    <property name="password" value="sgr" />
</bean>

<bean id="studentJDBCTemplate" class="com.tutorialspoint.StudentJDBCTemplate">
    <property name="dataSource" ref="dataSource"></property>

</bean>
```

37. *What is AOP?*

Aspect-oriented programming, 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. The core construct of AOP is the aspect, which encapsulates behaviors affecting multiple classes into reusable modules.

38. *How the AOP used in Spring?*

AOP is used in the Spring Framework: To provide declarative enterprise services, especially as a replacement for EJB declarative services. The most important such service is declarative transaction management, which builds on the Spring Framework's transaction abstraction. To allow users to implement custom aspects, complementing their use of OOP with AOP.

39. *What do you mean by Aspect?*

A modularization of a concern that cuts across multiple objects. Transaction management is a good example of a crosscutting concern in J2EE applications. In Spring AOP, aspects are implemented using regular classes (the schema-based approach) or regular classes annotated with the @Aspect annotation (@AspectJ style).

40. *What do you mean by JointPoint?*

A point during the execution of a program, such as the execution of a method or the handling of an exception. In Spring AOP, a join point always represents a method execution.

41. *What do you mean by Advice?*

Action taken by an aspect at a particular join point. Different types of advice include "around," "before" and "after" advice. Many AOP frameworks, including Spring, model an advice as an interceptor, maintaining a chain of interceptors "around" the join point.

42. *What are the types of Advice?*

Types of advice:

a) **Before advice:** Advice that executes before a join point, but which does not have the ability to prevent execution flow proceeding to the join point (unless it throws an exception).

b) **After returning advice*:*** Advice to be executed after a join point completes normally: for example, if a method returns without throwing an exception.
c) **After throwing advice*:*** Advice to be executed if a method exits by throwing an exception.
d) **After (finally) advice*:*** Advice to be executed regardless of the means by which a join point exits (normal or exceptional return).
e) **Around advice*:*** Advice that surrounds a join point such as a method invocation. This is the most powerful kind of advice. Around advice can perform custom behavior before and after the method invocation. It is also responsible for choosing whether to proceed to the join point or to shortcut the advised method execution by returning its own return value or throwing an exception

43. What are the types of the transaction management Spring supports?

Spring Framework supports:
a) Programmatic transaction management.
b) Declarative transaction management.

44. What are the benefits of the Spring Framework transaction management?

The Spring Framework provides a consistent abstraction for transaction management that delivers the following benefits:
a) Provides a consistent programming model across different transaction APIs such as JTA, JDBC, Hibernate, JPA, and JDO.
b) Supports declarative transaction management.
c) Provides a simpler API for programmatic transaction management than a number of complex transaction APIs such as JTA.
d) Integrates very well with Spring's various data access abstractions.

45. Why most users of the Spring Framework choose declarative transaction management?

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.

46. When to use programmatic and declarative transaction management?

Programmatic transaction management is usually a good idea only if you have a small number of transactional operations. On the other hand, if your application has numerous transactional operations, declarative transaction management is usually worthwhile. It keeps transaction management out of business logic, and is not difficult to configure.

47. Explain about the Spring DAO support?

The Data Access Object (DAO) support in Spring is aimed at making it easy to work with data access technologies like JDBC, Hibernate or JDO in a consistent way. This allows one to switch between the persistence technologies fairly easily and it also allows one to code without worrying about catching exceptions that are specific to each technology.

48. What are the exceptions thrown by the Spring DAO classes?

Spring DAO classes throw exceptions which are subclasses of DataAccessException (org.springframework.dao.DataAccessException). Spring provides a convenient translation from technology-specific exceptions like SQLException to its own exception class hierarchy with the DataAccessException as the root exception. These exceptions wrap the original exception.

49. *What is SQLExceptionTranslator?*

SQLExceptionTranslator, is an interface to be implemented by classes that can translate between SQLExceptions and Spring's own data-access-strategy-agnostic org.springframework.dao.DataAccessException.

50. *What is Spring's JdbcTemplate?*

Spring's JdbcTemplate is central class to interact with a database through JDBC. JdbcTemplate 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.

```
JdbcTemplate template = new JdbcTemplate(myDataSource);
```

51. *What is PreparedStatementCreator ?*

PreparedStatementCreator:

a) Is one of the most common used interfaces for writing data to database
b) Has one method – createPreparedStatement(Connection)
c) Responsible for creating a PreparedStatement.
d) Does not need to handle SQLExceptions.

52. *What is SQLProvider?*

SQLProvider:

a) Has one method – getSql()
b) Typically implemented by PreparedStatementCreator implementers.
c) Useful for debugging.

53. *What is RowCallbackHandler?*

The RowCallbackHandler interface extracts values from each row of a ResultSet.

a) Has one method – processRow(ResultSet)
b) Called for each row in ResultSet.
c) Typically stateful.

❖ *Spring MVC*

54. *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 model, view and controller parts of application. Spring framework provides lots of advantages over other MVC frameworks e.g.

a) 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.
b) Powerful and straightforward configuration of both framework and application classes as JavaBeans.
c) 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.
d) Customizable binding and validation
e) Customizable handler mapping and view resolution

f) Customizable locale and theme resolution

g) A JSP form tag library, introduced in Spring 2.0, which makes writing forms in JSP pages much easier. etc.

55. What are 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 pick up 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.

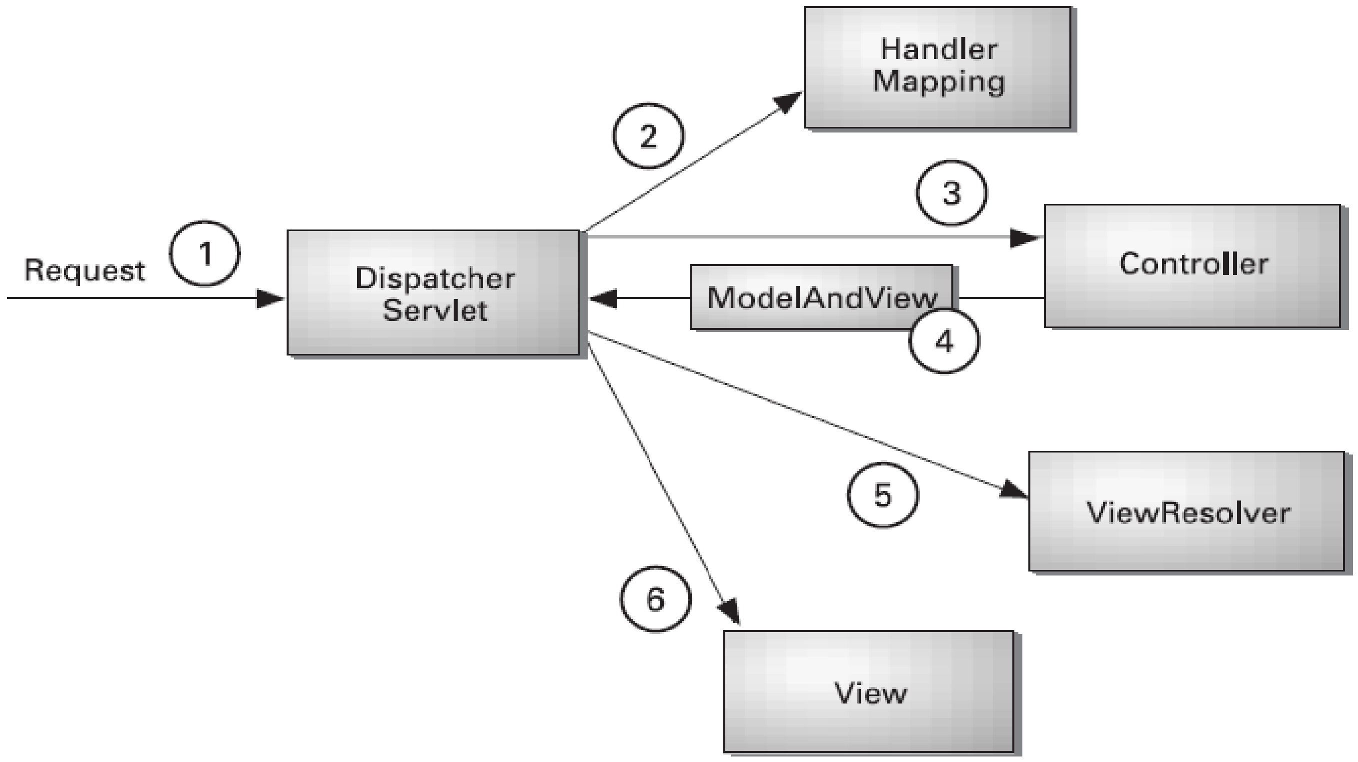


Figure 8.1 The life cycle of a request in Spring MVC

```
<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 class path.

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.

56. 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 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.

57. 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) 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>
```

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

a) 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.

b) 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 activates them.** So <context:component-scan> does what <context:annotation-config> does, but additionally it scan the packages and register the beans in application context.

59. Difference between @Component, @Controller, @Repository & @Service annotations?

a) 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 {
    ...
}
```

b) 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 eligible for translation into Spring DataAccessException.

c) 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.

d) @Controller annotation marks a class as a Spring Web MVC controller. It 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.

60. *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.

61. *What is a MultipartResolver and when it's used?*

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

a) **CommonsMultipartResolver** for Jakarta Commons FileUpload
b) **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 MultipartHttpServletRequest interface, which permits access to any MultipartFiles.

62. *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

63. *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 r
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "lastName", "error.lastName", "Last name is requ
        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" />
```

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

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. These classes implements HandlerInterceptor and provide default blank implementations.

```
<bean id="handlerMapping" class="org.springframework.web.servlet.mvc.method.annotation.RequestMa
ppingHandlerMapping">
      <property name="interceptors">
            <list>
                  <ref bean="yourCustomHandlerInterceptor"/>
            </list>
        </property>
</bean>
```

65. 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.

66. 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.

67. 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
}
```

68. 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>
```

69. How do you integrate Spring MVC with tiles?

Tiles help 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" />
```

70. *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>
```

71. *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 appropriate 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>
```

🗆 *Spring Web services*

A. SOAP Web services

72. What is a Web Service?

Web Services work on client-server model where client applications can access web services over the network. Web services provide endpoint URLs and expose methods that can be accessed over network through client programs written in java, shell script or any other different technologies. Web services are stateless and don't maintain user session like web applications.

73. What are the advantages of Web Services?

Some of the advantages of web services are:

a) Interoperability: Web services are accessible over network and runs on HTTP/SOAP protocol and uses XML/JSON to transport data, hence it can be developed in any programming language. Web service can be written in java programming and client can be PHP and vice versa.
b) Reusability: One web service can be used by many client applications at the same time.
c) Loose Coupling: Web services client code is totally independent with server code, so we have achieved loose coupling in our application.
d) Easy to deploy and integrate, just like web applications.
e) Multiple service versions can be running at same time.

74. What are different types of Web Services?

There are two types of web services:

a) SOAP Web Services: Runs on SOAP protocol and uses XML technology for sending data.
b) Restful Web Services: It's an architectural style and runs on HTTP/HTTPS protocol almost all the time. REST is a stateless client-server architecture where web services are resources and can be identified by their URIs. Client applications can use HTTP GET/POST methods to invoke Restful web services.

75. What is SOAP?

SOAP stands for Simple Object Access Protocol. SOAP is an XML based industry standard protocol for designing and developing web services. Since it's XML based, it's platform and language independent. So our server can be based on JAVA and client can be on .NET, PHP etc. and vice versa.

76. What are advantages of SOAP Web Services?

SOAP web services have all the advantages that web services has, some of the additional advantages are:

a) WSDL document provides contract and technical details of the web services for client applications without exposing the underlying implementation technologies.
b) SOAP uses XML data for payload as well as contract, so it can be easily read by any technology.
c) SOAP protocol is universally accepted, so it's an industry standard approach with many easily available open source implementations.

77. What are disadvantages of SOAP Web Services?

Some of the disadvantages of SOAP protocol are:

a) Only XML can be used, JSON and other lightweight formats are not supported.
b) SOAP is based on the contract, so there is a tight coupling between client and server applications.
c) SOAP is slow because payload is large for a simple string message, since it uses XML format.
d) Anytime there is change in the server side contract, client stub classes need to be generated again.
e) Can't be tested easily in browser

78. What is WSDL?

WSDL stands for Web Service Description Language. WSDL is an XML based document that provides technical details about the web service. Some of the useful information in WSDL document are: method name, port types, service end point, binding, method parameters etc.

79. What are different components of WSDL?

Some of the different tags in WSDL xml are:

a) xsd:import namespace and schemaLocation: provides WSDL URL and unique namespace for web service.
b) message: for method arguments
c) part: for method argument name and type
d) portType: service name, there can be multiple services in a wsdl document.
e) operation: contains method name
f) soap:address for endpoint URL.

80. What is UDDI?

UDDI is acronym for Universal Description, Discovery and Integration. UDDI is a directory of web services where client applications can lookup for web services. Web Services can register to the UDDI server and make them available to client applications.

81. What is difference between Top Down and Bottom Up approach in SOAP Web Services?

In Top Down approach first WSDL document is created to establish the contract between web service and client and then code is written, it's also termed as contract first approach. This is hard to implement because classes need to be written to confirm the contract established in WSDL. Benefit of this approach is that both client and server code can be written in parallel.
In Bottom Up approach, first web service code is written and then WSDL is generated. It's also termed as contract last approach. This approach is easy to implement because WSDL is generated based on code. In this approach client code has to wait for WSDL from server side to start their work.

82. Compare SOAP and REST web services?

SOAP	REST
SOAP is a standard protocol for creating web services.	REST is an architectural style to create web services.
SOAP is acronym for Simple Object Access Protocol.	REST is acronym for REpresentational State Transfer.
SOAP uses WSDL to expose supported methods and technical details.	REST exposes methods through URIs, there are no technical details.
SOAP web services and client programs are bind with WSDL contract	REST doesn't have any contract defined between server and client
SOAP web services and client are tightly coupled with contract.	REST web services are loosely coupled.
SOAP learning curve is hard, requires us to learn about WSDL generation, client stubs creation etc.	REST learning curve is simple, POJO classes can be generated easily and works on simple HTTP methods.
SOAP supports XML data format only	REST supports any data type such as XML, JSON, image etc.
SOAP web services are hard to maintain, any change in WSDL contract requires us to create client stubs again and then make changes to client code.	REST web services are easy to maintain when compared to SOAP, a new method can be added without any change at client side for existing resources.
SOAP web services can be tested through programs or software such as Soap UI.	REST can be easily tested through CURL command, Browsers and extensions such as Chrome Postman.

83. What are different ways to test web services?

SOAP web services can be tested programmatically by generating client stubs from WSDL or through software such as Soap UI.
REST web services can be tested easily with program, curl commands and through browser extensions. Resources supporting GET method can be tested with browser itself, without any program.

84. Can we maintain user session in web services?

Web services are stateless so we can't maintain user sessions in web services.

85. What is difference between SOA and Web Services?

Service Oriented Architecture (SOA) is an architectural pattern where applications are designed in terms of services that can be accessed through communication protocol over network. SOA is a design pattern and doesn't go into implementation. Web Services can be thought of as Services in SOAP architecture and providing means to implement SOA pattern.

86. How would you choose between SOAP and REST web services?

Web Services work on client-server model and when it comes to choose between SOAP and REST, it all depends on project requirements. Let's look at some of the conditions affecting our choice:

a. Do you know your web service clients beforehand? If Yes, then you can define a contract before implementation and SOAP seems better choice. But if you don't then REST seems better choice because you can provide sample request/response and test cases easily for client applications to use later on.
b. How much time you have? For quick implementation REST is the best choice. You can create web service easily, test it through browser/curl and get ready for your clients.
c. What kind of data format are supported? If only XML then you can go with SOAP but if you think about supporting JSON also in future then go with REST.

87. What is JAX-WS API?

JAX-WS stands for Java API for XML Web Services. JAX-WS is XML based Java API to build web services server and client application. It's part of standard Java API, so we don't need to include anything else which working with it.

88. Name some frameworks in Java to implement SOAP web services?

We can create SOAP web services using JAX-WS API, however some of the other frameworks that can be used are Apache Axis and Apache CXF. Note that they are not implementations of JAX-WS API, they are totally different framework that work on Servlet model to expose your business logic classes as SOAP web services.

89. Name important annotations used in JAX-WS API?

Some of the important annotations used in JAX-WS API are:

a) @WebService
b) @SOAPBinding
c) @WebMethod

90. What is use of javax.xml.ws.Endpoint class?

Endpoint class provides useful methods to create endpoint and publish existing implementation as web service. This comes handy in testing web services before making further changes to deploy it on actual server.

91. How to get WSDL file of a SOAP web service?

WSDL document can be accessed by appending ?wsdl to the SOAP endoint URL. In above example, we can access it at http://localhost:8888/testWS?wsdl location.

92. What is wsimport utility?

We can use wsimport utility to generate the client stubs. This utility comes with standard installation of JDK

93. What is the use of @XmlRootElement annotation?

XmlRootElement annotation is used by JAXB to transform java object to XML and vice versa. So we have to annotate model classes with this annotation.

B. REST Web services:-

94. What is REST Web Services?

REST is the acronym for REpresentational State Transfer. REST is an architectural style for developing applications that can be accessed over the network. REST architectural style was brought in light by Roy Fielding in his doctoral thesis in 2000. REST is a stateless client-server architecture where web services are resources and can be identified by their URIs. Client applications can use HTTP GET/POST methods to invoke Restful web services. REST doesn't specify any specific protocol to use, but in almost all cases it's used over HTTP/HTTPS. When compared to SOAP web services, these are lightweight and don't follow any standard. We can use XML, JSON, text or any other type of data for request and response.

95. What are advantages of REST web services?

Some of the advantages of REST web services are:

a) Learning curve is easy since it works on HTTP protocol
b) Supports multiple technologies for data transfer such as text, xml, json, image etc.
c) No contract defined between server and client, so loosely coupled implementation.
d) REST is a lightweight protocol
e) REST methods can be tested easily over browser.

96. What are disadvantages of REST web services?

Some of the disadvantages of REST are:

a) Since there is no contract defined between service and client, it has to be communicated through other means such as documentation or emails.
b) Since it works on HTTP, there can't be asynchronous calls.
c) Sessions can't be maintained.

97. What is a Resource in Restful web services

Resource is the fundamental concept of Restful architecture. A resource is an object with a type, relationship with other resources and methods that operate on it. Resources are identified with their URI, HTTP methods they support and request/response data type and format of data.

98. What are different HTTP Methods supported in Restful Web Services?

Restful web services supported HTTP methods are – GET, POST, PUT, DELETE and HEAD.

99. What is the use of Accept and Content-Type Headers in HTTP Request?

These are important headers in Restful web services. Accept headers tells web service what kind of response client is accepting, so if a web service is capable of sending response in XML and JSON format and client sends Accept header as

"application/xml" then XML response will be sent. For Accept header "application/json", server will send the JSON response.
Content-Type header is used to tell server what is the format of data being sent in the request. If Content-Type header is "application/xml" then server will try to parse it as XML data. This header is useful in HTTP Post and Put requests.

100. What is JAX-RS API?

Java API for RESTful Web Services (JAX-RS) is the Java API for creating REST web services. JAX-RS uses annotations to simplify the development and deployment of web services. JAX-RS is part of JDK, so you don't need to include anything to use it's annotations.

101. Name some implementations of JAX-RS API?

There are two major implementations of JAX-RS API.

a) Jersey: Jersey is the reference implementation provided by Sun. For using Jersey as our JAX-RS implementation, all we need to configure its servlet in web.xml and add required dependencies. Note that JAX-RS API is part of JDK not Jersey, so we have to add its dependency jars in our application.
b) RESTEasy: RESTEasy is the JBoss project that provides JAX-RS implementation.

102. How to set different status code in HTTP response?

For setting HTTP status code other than 200, we have to use javax.ws.rs.core.Response class for response. Below are some of the sample return statements showing it's usage.

```
return Response.status(422).entity(exception).build();
return Response.ok(response).build(); //200
```