1) What are different Spring Bean Scopes?

The beans in spring container can be created in **five scopes**. All the scope names are self-explanatory but lets make them clear so that there will not be any doubt.

1. **singleton**: This bean scope is default and it enforces the container to have only one instance per spring container irrespective of how much time you request for its instance. This singleton behavior is maintained by bean factory itself.
2. **prototype**: This bean scope just reverses the behavior of singleton scope and produces a new instance each and every time a bean is requested.
3. **request**: With this bean scope, a new bean instance will be created for each web request made by client. As soon as request completes, bean will be out of scope and garbage collected.
4. **session**: Just like request scope, this ensures one instance of bean per user session. As soon as user ends its session, bean is out of scope.
5. **global-session**: global-session is something which is connected to Portlet applications. When your application works in Portlet container it is built of some amount of portlets. Each portlet has its own session, but if your want to store variables global for all portlets in your application than you should store them in global-session. This scope doesn’t have any special effect different from session scope in Servlet based applications.

### 2.What is the difference between BeanFactory and ApplicationContext?

BeanFactory is a Lightweight container which loads bean definitions and manages your beans whereas ApplicationContext is the advanced container. ApplicationContext extends the BeanFactory interface. ApplicationContext provides more facilities than BeanFactory such as integration with spring AOP, message resource handling for i18n etc

3.What do you mean by IOC and DI

IOC(Inversion Of Controller):   Giving control to the container to get instance of object is called Inversion of Control., means instead of you are creating object using new operator, let the container do that for you.  
  
DI(Dependency Injection):  Way of injecting properties to an object is called Dependency injection.  
  
We have three types of Dependency injection  
        1)  Constructor Injection  
        2)  Setter/Getter Injection  
        3) InterfaceInjection  
Spring will support only Constructor Injection and Setter/Getter Injection.

### 4.What is autowiring in spring? What are the autowiring modes?

Autowiring is the process to inject the bean automatically. We don't need to write explicit injection logic.

Let's see the code to inject bean using dependency injection.

<bean id="emp" **class**="com.javatpoint.Employee" autowire="byName" />

5) What are inner beans in Spring?

In Spring framework, whenever a bean is used for only one particular property, it’s advise to declare it as an inner bean. And the inner bean is supported both in setter injection ‘**property**‘ and constructor injection ‘**constructor-arg**‘.

For example, let’s say we one Customer class having reference of Person class. In our application, we will be creating only one instance of Person class, and use it inside Customer.

public class Customer

{

private Person person;

//Setters and Getters

}

public class Person

{

private String name;

private String address;

private int age;

//Setters and Getters

}

Now inner bean declaration will look like this:

<bean id="CustomerBean" class="com.howtodoinjava.common.Customer">

<property name="person">

<!-- This is inner bean -->

<bean class="com.howtodoinjava.common.Person">

<property name="name" value="lokesh" />

<property name="address" value="India" />

<property name="age" value="34" />

</bean>

</property>

</bean>

6) Are Singleton beans thread safe in Spring Framework?

Spring framework does not do anything under the hood concerning the multi-threaded behavior of a [**singleton**](http://howtodoinjava.com/design-patterns/singleton-design-pattern-in-java/) bean. It is the developer’s responsibility to deal with concurrency issue and [**thread safety**](http://howtodoinjava.com/2014/06/02/what-is-thread-safety/) of the singleton bean.

While practically, most spring beans have no mutable state (e.g. Service and DAO clases), and as such are trivially thread safe. But if your bean has mutable state (e.g. View Model Objects), so you need to ensure thread safety. The most easy and obvious solution for this problem is to change bean scope of mutable beans from “**singleton**” to “**prototype**“.

7) How can you inject a Java Collection in Spring? Give example?

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

**<list>** : This helps in wiring ie 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.

Let’s see example of each type.

<beans>

<!-- Definition for javaCollection -->

<bean id="javaCollection" class="com.howtodoinjava.JavaCollection">

<!-- java.util.List -->

<property name="customList">

<list>

<value>INDIA</value>

<value>Pakistan</value>

<value>USA</value>

<value>UK</value>

</list>

</property>

<!-- java.util.Set -->

<property name="customSet">

<set>

<value>INDIA</value>

<value>Pakistan</value>

<value>USA</value>

<value>UK</value>

</set>

</property>

<!-- java.util.Map -->

<property name="customMap">

<map>

<entry key="1" value="INDIA"/>

<entry key="2" value="Pakistan"/>

<entry key="3" value="USA"/>

<entry key="4" value="UK"/>

</map>

</property>

<!-- java.util.Properties -->

<property name="customProperies">

<props>

<prop key="admin">admin@nospam.com</prop>

<prop key="support">support@nospam.com</prop>

</props>

</property>

</bean>

</beans>

|  |  |
| --- | --- |
|  |  |

Spring MVC Flow:

## **1.What is the front controller class of Spring MVC?**

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

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

## **2.Can we have multiple Spring configuration files?**

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

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

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

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

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

## **3.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.

## **4.How does Spring MVC provide validation support?**

Spring supports validations primarily into two ways.

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

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

## **5.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 asrejectIfEmptyOrWhitespace() and rejectIfEmpty() in the ValidationUtils class to validate the required form fields.

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

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

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

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

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

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

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

### [How is validation done using Spring Framework?](http://www.javainterview.in/p/spring-interview-questions.html)

Spring validator can be used both in web and business layers to validate objects. It is based on the org.springframework.validation.Validator interface having two important methods

* supports(Class) – does this validator support a particular class
* validate(Object, org.springframework.validation.Errors) – validates and sets errors into Errors object

An example is provided below: MessageCodesResolver can be used to convert the message code into proper internationalized text.

public class CarValidator implements Validator {

public boolean supports(Class clazz) {

return Car.class.equals(clazz);

}

public void validate(Object obj, Errors e) {

ValidationUtils.rejectIfEmpty(e, "name", "name.is.empty");

Car c = (Car) obj;

if (c.getUsedYears() < 0) {

e.rejectValue("usedYears", "not.yet.bought");

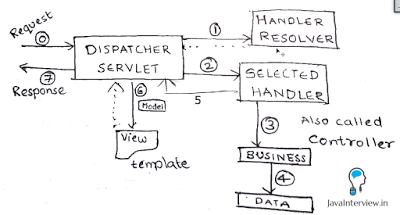
}

}

}

### 6. How does request flow happen in Spring MVC?

When we are sending the request it first goes to the DispatcherServlet. The Controller takes the request and calls the appropriate service methods and set model data and then returns view name to the DispatcherServlet. The DispatcherServlet will take help from ViewResolver to pickup the defined view for the request. Once view is finalized, The DispatcherServlet passes the model data to the view which is finally rendered on the browser.

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

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

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

### Spring JDBC:

### 1)What are the advantages of JdbcTemplate in spring?

**Less code**: By using the JdbcTemplate class, you don't need to create connection,statement,start transaction,commit transaction and close connection to execute different queries. You can execute the query directly.

### 2) What are classes for spring JDBC API?

1. JdbcTemplate
2. SimpleJdbcTemplate
3. NamedParameterJdbcTemplate
4. SimpleJdbcInsert
5. SimpleJdbcCall

### 3) How can you fetch records by spring JdbcTemplate?

You can fetch records from the database by the **query method of JdbcTemplate**. There are two interfaces to do this:

1. [ResultSetExtractor](http://www.javatpoint.com/ResultSetExtractor-example)
2. [RowMapper](http://www.javatpoint.com/RowMapper-example)

### 4) What is the advantage of NamedParameterJdbcTemplate?

NamedParameterJdbcTemplate class is used to pass value to the named parameter. A named parameter is better than ? (question mark of PreparedStatement).

### 5)What are Class loaders? Explain the types of class loader

Class loaders are the part of the Java Runtime Environment that dynamically loads Java classes into the Java virtual machine. It is responsible for locating libraries, reading there content and loading the classes contained within the libraries. When JVM is started three class loaders are used  
  
1. Bootstrap class loader  
  
2. Extensions class loader  
  
3. System class loader  
  
Bootstrap class loader loads the core java libraries. It is written in native code. The bootstrap class loader is responsible for loading key java classes like java.lang.Object and other runtime code into memory. The runtime classes are packaged inside jre/lib/rt.jar file.   
  
Extensions class loader loads the code in the extension directories. It is implemented by ExtClassLoader class.  
  
System class loader the code found on the java.class.path which map to the system class path variables. It is implemented by AppClassLoader class. All user classes by default are load by the system class loader.

### 6)What is the difference between creating String as new() and literal?

When we create string with new() Operator, it’s created in heap and not added into string pool while String created using literal are created in String pool itself which exists in PermGen area of heap.

String s = new String("Test");  
does not  put the object in String pool , we need to call String.intern() method which is used to put  them into String pool explicitly. its only when you create String object as String literal e.g. String s = "Test" Java automatically put that into String pool.