**Spring**

**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.  
  
i.e., 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:

* **Constructor Injection** (e.g. Pico container, Spring etc): Dependencies are provided as constructor parameters.
* **Setter Injection** (e.g. Spring): Dependencies are assigned through JavaBeans properties (ex: setter methods).
* **Interface Injection** (e.g. Avalon): Injection is done through an interface.

*Note: Spring supports only Constructor and Setter Injection*

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

Benefits of IOC (Dependency Injection) are as follows:

* 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.
* 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.
* 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.
* 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.

|  |
| --- |
|  |

**4.  What is Spring ?**

Spring is an open source framework created to address the complexity of [enterprise application development](http://www.developersbook.com/spring/interview-questions/spring-interview-questions-faqs.php). 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.

**5. What are the advantages of Spring framework?**

The advantages of Spring are as follows:

* Spring has layered architecture. Use what you need and leave you don't need now.
* Spring Enables POJO Programming. There is no behind the scene magic here. POJO programming enables [continuous integration](http://www.developersbook.com/spring/interview-questions/spring-interview-questions-faqs.php) and testability.
* Dependency Injection and Inversion of Control Simplifies JDBC
* Open source and no vendor lock-in.

**6. What are features of Spring ?**

* **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.

* **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 comes with MVC web [application framework](http://www.developersbook.com/spring/interview-questions/spring-interview-questions-faqs.php), 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.

* **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.

* **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](http://www.developersbook.com/spring/interview-questions/spring-interview-questions-faqs.php) with Hibernate, JDO and iBATIS

**7. How many modules are there in Spring? What are they?**

|  |
| --- |
| (**Roll over to view the Image** )  Spring Framework Modules |

       Spring comprises of seven modules. They are..

* **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.

* **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.

* **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.

* **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 to its generic DAO exception hierarchy.

* **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 to Spring's generic transaction and DAO exception hierarchies.

* **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.

* **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.

**8. What are the types of Dependency Injection Spring supports?>**

* **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.

* **Constructor Injection:**

Constructor-based DI is realized by invoking a constructor with a number of arguments, each representing a collaborator.

|  |
| --- |
|  |

**9. What is Bean Factory ?**

A BeanFactory is like a factory class that contains a collection of beans. The BeanFactory holds Bean Definitions of multiple beans within itself and then instantiates the bean whenever asked for by clients.

* BeanFactory is able to create associations between collaborating objects as they are instantiated. This removes the burden of configuration from bean itself and the beans client.
* BeanFactory also takes part in the life cycle of a bean, making calls to custom initialization and destruction methods.

**10. What is Application Context?**

A bean factory is fine to simple applications, but to take advantage of the full power of the Spring framework, you may want to move up to Springs more advanced container, the application context. On the surface, an application context is same as a bean factory.Both load bean definitions, wire beans together, and dispense beans upon request. But it also provides:

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

**11. What is the difference between Bean Factory and Application Context ?**Important Question

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

* 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.
* ResourceLoader support: Spring’s Resource interface us 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.
* MessageSource support: The application context implements MessageSource, an interface used to obtain localized messages, with the actual implementation being pluggable

**What are the common implementations of the Application Context ?**

   The three commonly used implementation of 'Application Context' are

* **ClassPathXmlApplicationContext :** It Loads context definition from an XML file located in the classpath, treating context definitions as classpath resources. The application context is loaded from the application's classpath by using the code .  
  ApplicationContext context = new ClassPathXmlApplicationContext("bean.[xml](http://www.developersbook.com/spring/interview-questions/spring-interview-questions-faqs-2.php)");
* **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");
* **XmlWebApplicationContext :** It loads context definition from an XML file contained within a [web application](http://www.developersbook.com/spring/interview-questions/spring-interview-questions-faqs-2.php).

**13. How is a typical spring implementation look like ?**

   For a typical Spring Application we need the following files:

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

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

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

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

**15. What do you mean by Bean wiring ?**

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

**16. What do you mean by 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. The autowiring functionality has *five modes*.

* no
* byName
* byType
* constructor
* autodirect

|  |
| --- |
|  |

**17. What is DelegatingVariableResolver?**

       Spring provides a custom JavaServer Faces VariableResolver implementation that extends the standard [Java Server](http://www.developersbook.com/spring/interview-questions/spring-interview-questions-faqs-2.php) Faces managed beans mechanism which lets you use JSF and Spring together. This variable resolver is called as *DelegatingVariableResolver*

**18. How to integrate  Java** [**Server**](http://www.developersbook.com/spring/interview-questions/spring-interview-questions-faqs-2.php) **Faces (JSF) with Spring?**

   JSF and Spring do share some of the same features, most noticeably in the area of IOC services. By declaring JSF managed-beans in the faces-config.xml configuration file, you allow the FacesServlet to instantiate that bean at startup. Your JSF pages have access to these beans and all of their properties.We can integrate JSF and Spring in two ways:

* **DelegatingVariableResolver:** Spring comes with a JSF variable resolver that lets you use JSF and Spring together.
* <?xml version="1.0" encoding="UTF-8"?>
* <!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
* "http://www.springframework.org/dtd/spring-beans.dtd">
* <faces-config>
* <application>
* <variable-resolver>
* org.springframework.web.jsf.DelegatingVariableResolver
* </variable-resolver>
* </application>
* </faces-config>

The DelegatingVariableResolver will first delegate value lookups to the default resolver of the underlying JSF implementation, and then to Spring's 'business context' WebApplicationContext. This allows one to easily inject dependencies into one's JSF-managed beans.

* FacesContextUtils:custom VariableResolver works well when mapping one's properties to beans in faces-config.xml, but at times one may need to grab a bean explicitly. The FacesContextUtils class makes this easy. It is similar to WebApplicationContextUtils, except that it takes a FacesContext parameter rather than a ServletContext parameter.
* ApplicationContext ctx = FacesContextUtils.getWebApplicationContext(FacesContext.getCurrentInstance());

**19. What is  Java Server Faces (JSF) - Spring integration mechanism?**

Spring provides a custom JavaServer Faces VariableResolver implementation that extends the standard JavaServer Faces managed beans mechanism. When asked to resolve a variable name, the following algorithm is performed:

* Does a bean with the specified name already exist in some scope (request, session, application)? If so, return it
* Is there a standard JavaServer Faces managed bean definition for this variable name? If so, invoke it in the usual way, and return the bean that was created.
* Is there configuration information for this variable name in the Spring WebApplicationContext for this application? If so, use it to create and configure an instance, and return that instance to the caller.
* If there is no managed bean or Spring definition for this variable name, return null instead.
* BeanFactory also takes part in the life cycle of a bean, making calls to custom initialization and destruction methods.

As a result of this algorithm, you can transparently use either JavaServer Faces or Spring facilities to create beans on demand.

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

**20. What is Significance of JSF- Spring integration ?**

Spring - JSF integration is useful when an event handler wishes to explicitly invoke the bean factory to create beans on demand, such as a bean that encapsulates the business logic to be performed when a submit button is pressed.

**21. How to integrate your Struts application with Spring?**

To integrate your Struts application with Spring, we have two options:

* Configure Spring to manage your Actions as beans, using the ContextLoaderPlugin, and set their dependencies in a Spring context file.
* Subclass Spring's ActionSupport classes and grab your Spring-managed beans explicitly using a getWebApplicationContext() method.

**What are ORM’s Spring supports ?**

**Spring supports the following ORM’s** :

* Hibernate
* iBatis
* JPA ([Java](http://www.developersbook.com/spring/interview-questions/spring-interview-questions-faqs-3.php) Persistence API)
* TopLink
* JDO (Java Data Objects)
* OJB

**23. What are the ways to access Hibernate using Spring ?**

   There are two approaches to Spring’s Hibernate integration:

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

**24. 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.

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

**25. 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](http://www.developersbook.com/spring/interview-questions/spring-interview-questions-faqs-3.php) of a single HTTP request; that is each and every 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. |

|  |
| --- |
|  |

**26. 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.

**27. 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.

|  |
| --- |
|  |

**28. What do you mean by Aspect ?**

   A modularization of a concern that cuts across [multiple objects](http://www.developersbook.com/spring/interview-questions/spring-interview-questions-faqs-3.php). Transaction management is a good example of a crosscutting concern in [J2EE applications](http://www.developersbook.com/spring/interview-questions/spring-interview-questions-faqs-3.php). In Spring AOP, aspects are implemented using regular classes (the schema-based approach) or regular classes annotated with the @Aspect annotation (@AspectJ style).

**29. 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.

**30. 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.

**31. What are the types of Advice?**

Types of advice:

* *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).
* *After returning advice*: Advice to be executed after a join point completes normally: for example, if a method returns without throwing an exception.
* *After throwing advice*: Advice to be executed if a method exits by throwing an exception.
* *After (finally) advice*: Advice to be executed regardless of the means by which a join point exits (normal or exceptional return).
* *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

**What are the types of the transaction management Spring supports ?**

   Spring Framework supports:

* Programmatic transaction management.
* Declarative transaction management.

**33. 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:

* Provides a consistent programming model across different transaction APIs such as JTA, JDBC, Hibernate, JPA, and JDO.
* Supports declarative transaction management.
* Provides a simpler API for programmatic transaction management than a number of complex transaction APIs such as JTA.
* Integrates very well with Spring's various [data access](http://www.developersbook.com/spring/interview-questions/spring-interview-questions-faqs-4.php) abstractions.

## **) What is Spring?**

***Spring*** is a lightweight inversion of control and aspect-oriented container framework.

## **2) Explain Spring?**

* **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.
* **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.
* **Framework :** ***Spring*** provides most of the intra functionality leaving rest of the coding to the developer.

## **3) What are the different modules in Spring framework?**

* The Core container module
* Application context module
* AOP module (Aspect Oriented Programming)
* JDBC abstraction and DAO module
* O/R mapping integration module (Object/Relational)
* Web module
* ***MVC framework*** module

## **4) What is the structure of Spring framework?**



## **5) What is the Core container module?**

This module is provides the fundamental functionality of the spring framework. In this module **BeanFactory** is the heart of any spring-based application. The entire framework was built on the top of this module. This module makes the ***Spring container***.

## **6) What is Application context module?**

The Application context module makes spring a framework. This module extends the concept of ***BeanFactory***, providing support for internationalization (I18N) messages, application lifecycle events, and validation. This module also supplies many enterprise services such JNDI access, ***EJB integration***, remoting, and scheduling. It also provides support to other framework.

## **7) What is AOP module?**

The ***AOP*** module is used for developing aspects for our Spring-enabled application. Much of the support has been provided by the AOP Alliance in order to ensure the interoperability between ***Spring*** and other ***AOP*** frameworks. This module also introduces metadata programming to ***Spring***. Using Spring’s metadata support, we will be able to add ***annotations*** to our source code that instruct ***Spring*** on where and how to apply aspects.

## **8) What is JDBC abstraction and DAO module?**

Using this module we can keep up the database code clean and simple, and prevent problems that result from a failure to close database resources. A new layer of meaningful exceptions on top of the error messages given by several database servers is bought in this module. In addition, this module uses ***Spring’s AOP module*** to provide transaction management services for objects in a Spring application.

## **9) What are object/relational mapping integration module?**

Spring also supports for using of an object/relational mapping (ORM) tool over straight JDBC by providing the ORM module. Spring provide support to tie into several popular ***ORM frameworks***, including ***Hibernate***, ***JDO***, and ***iBATIS SQL Maps***. Spring’s transaction management supports each of these ***ORM frameworks*** as well as ***JDBC***.

## **10) What is web module?**

This module is built on the application context module, providing a context that is appropriate for web-based applications. This module also contains support for several web-oriented tasks such as transparently handling multipart requests for file uploads and programmatic binding of request parameters to your business objects. It also contains integration support with ***Jakarta Struts***.

|  |
| --- |
| **11) What is web module?** Spring comes with a full-featured MVC framework for building web applications. Although Spring can easily be integrated with other MVC frameworks, such as Struts, Spring’s MVC framework uses IoC to provide for a clean separation of controller logic from business objects. It also allows you to declaratively bind request parameters to your business objects. It also can take advantage of any of Spring’s other services, such as I18N messaging and validation. **12) What is a BeanFactory?** A BeanFactory is an implementation of the factory pattern that applies Inversion of Control to separate the application’s configuration and dependencies from the actual application code. **13) What is AOP Alliance?** AOP Alliance is an open-source project whose goal is to promote adoption of AOP and interoperability among different AOP implementations by defining a common set of interfaces and components. **14) 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. **15) What does a simple spring application contain?** These applications are like any Java application. They are made up of several classes, each performing a specific purpose within the application. But these classes are configured and introduced to each other through an XML file. This XML file describes how to configure the classes, known as the Spring configuration file. **16) What is XMLBeanFactory?** ***BeanFactory*** has many implementations in Spring. But one of the most useful one is ***org.springframework.beans.factory.xml.XmlBeanFactory***, which loads its beans based on the definitions contained in an XML file. To create an ***XmlBeanFactory***, pass a java.io.InputStream to the constructor. The ***InputStream*** will provide the XML to the factory. For example, the following code snippet uses a java.io.***FileInputStream*** to provide a bean definition XML file to ***XmlBeanFactory***.  **BeanFactory** factory = new **XmlBeanFactory**(new **FileInputStream**("beans.xml"));  To retrieve the bean from a BeanFactory, call the getBean() method by passing the name of the bean you want to retrieve.  MyBean myBean = (MyBean) factory.**getBean**("myBean"); **17) What are important ApplicationContext implementations in spring framework?**  * **ClassPathXmlApplicationContext –** This context loads a context definition from an XML file located in the class path, treating context definition files as class path resources. * **FileSystemXmlApplicationContext –** This context loads a context definition from an XML file in the filesystem. * **XmlWebApplicationContext –** This context loads the context definitions from an XML file contained within a web application.  **18) Explain Bean lifecycle in Spring framework?**  1. The spring container finds the bean’s definition from the XML file and instantiates the bean. 2. Using the dependency injection, spring populates all of the properties as specified in the bean definition. 3. If the bean implements the **BeanNameAware** interface, the factory calls **setBeanName()** passing the bean’s ID. 4. If the bean implements the **BeanFactoryAware** interface, the factory calls **setBeanFactory()**, passing an instance of itself. 5. If there are any **BeanPostProcessors** associated with the bean, their **post- ProcessBeforeInitialization()** methods will be called. 6. If an init-method is specified for the bean, it will be called. 7. Finally, if there are any **BeanPostProcessors** associated with the bean, their **postProcessAfterInitialization()** methods will be called.  **19) What is bean wiring?** Combining together beans within the Spring container is known as bean wiring or wiring. When wiring beans, you should tell the container what beans are needed and how the container should use dependency injection to tie them together. **20) How do add a bean in spring application?** <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/spring-beans.dtd">  <beans>  <bean id="foo" class="com.act.Foo"/>  **<bean id="bar" class="com.act.Bar"/>**  </beans>  In the bean tag the id attribute specifies the bean name and the class attribute specifies the fully qualified class name. |
| **21) What are singleton beans and how can you create prototype beans?** 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.  <beans>  <bean id="bar" class="com.act.Foo" **singleton=”false”/**>  </beans> **22) What are the important beans lifecycle methods?** 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. **23) How can you override beans default lifecycle methods?** The bean tag has two more important attributes with which you can define your own custom initialization and destroy methods. Here I have shown a small demonstration. Two new methods fooSetup and fooTeardown are to be added to your Foo class.  <beans>  <bean id="bar" class="com.act.Foo" **init-method=”fooSetup” destroy=”fooTeardown”/**>  </beans> **24) What are Inner Beans?** When wiring beans, if a bean element is embedded to a property tag directly, then that bean is said to the Inner Bean. The drawback of this bean is that it cannot be reused anywhere else. **25) What are the different types of bean injections?** There are two types of bean injections.   1. **By setter** 2. **By constructor**  **26) What is Auto wiring?** You can wire the beans as you wish. But spring framework also does this work for you. It can auto wire the related beans together. All you have to do is just set the autowire attribute of bean tag to an autowire type.  <beans>  <bean id="bar" class="com.act.Foo" **Autowire=”autowire type”/**>  </beans> **27) What are different types of Autowire types?** There are four different types by which autowiring can be done.   * + **byName**   + **byType**   + **constructor**   + **autodetect**  **28) What are the different types of events related to Listeners?** There are a lot of events related to **ApplicationContext** of spring framework. All the events are subclasses of **org.springframework.context.Application-Event**. They are   * ContextClosedEvent – This is fired when the context is closed. * ContextRefreshedEvent – This is fired when the context is initialized or refreshed. * RequestHandledEvent – This is fired when the web context handles any request.  **29) What is an Aspect?** An aspect is the cross-cutting functionality that you are implementing. It is the aspect of your application you are modularizing. An example of an aspect is logging. Logging is something that is required throughout an application. However, because applications tend to be broken down into layers based on functionality, reusing a logging module through inheritance does not make sense. However, you can create a logging aspect and apply it throughout your application using AOP. **30) What is a Jointpoint?** A joinpoint is a point in the execution of the application where an aspect can be plugged in. This point could be a method being called, an exception being thrown, or even a field being modified. These are the points where your aspect’s code can be inserted into the normal flow of your application to add new behavior.   |  |  | | --- | --- | |  |  | |

## **31) What is an Advice?**

Advice is the implementation of an aspect. It is something like telling your application of a new behavior. Generally, and advice is inserted into an application at join points.

## **32) What is a Pointcut?**

A pointcut is something that defines at what joinpoints an advice should be applied. Advices can be applied at any joinpoint that is supported by the AOP framework. These Pointcuts allow you to specify where the advice can be applied.

## **33) What is an Introduction in AOP?**

An introduction allows the user to add new methods or attributes to an existing class. This can then be introduced to an existing class without having to change the structure of the class, but give them the new behavior and state.

## **34) What is a Target?**

A target is the class that is being advised. The class can be a third party class or your own class to which you want to add your own custom behavior. By using the concepts of AOP, the target class is free to center on its major concern, unaware to any advice that is being applied.

## **35) What is a Proxy?**

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

## **36) What is meant by Weaving?**

The process of applying aspects to a target object to create a new proxy object is called as Weaving. The aspects are woven into the target object at the specified joinpoints.

## **37) What are the different points where weaving can be applied?**

* Compile Time
* Classload Time
* Runtime

## **38) What are the different advice types in spring?**

* **Around :** Intercepts the calls to the target method
* **Before :** This is called before the target method is invoked
* **After :** This is called after the target method is returned
* **Throws :** This is called when the target method throws and exception
* Around : org.aopalliance.intercept.MethodInterceptor
* Before : org.springframework.aop.BeforeAdvice
* After : org.springframework.aop.AfterReturningAdvice
* Throws : org.springframework.aop.ThrowsAdvice

## **39) What are the different types of AutoProxying?**

* BeanNameAutoProxyCreator
* DefaultAdvisorAutoProxyCreator
* Metadata autoproxying

## **40) What is the Exception class related to all the exceptions that are thrown in spring applications?**

**DataAccessException -** org.springframework.dao.DataAccessException

## **41) What kind of exceptions those spring DAO classes throw?**

The springâ€™s DAO class does not throw any technology related exceptions such as SQLException. They throw exceptions which are subclasses of DataAccessException.

## **42) What is DataAccessException?**

DataAccessException is a RuntimeException. This is an Unchecked Exception. The user is not forced to handle these kinds of exceptions.

## **43) How can you configure a bean to get DataSource from JNDI?**

<bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean">

<property name="jndiName">

<value>java:comp/env/jdbc/myDatasource</value>

</property>

</bean>

## **44) How can you create a DataSource connection pool?**

<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">

<property name="driver">

<value>${db.driver}</value>

</property>

<property name="url">

<value>${db.url}</value>

</property>

<property name="username">

<value>${db.username}</value>

</property>

<property name="password">

<value>${db.password}</value>

</property>

</bean>

## **45) How JDBC can be used more efficiently in spring framework?**

JDBC can be used more efficiently with the help of a template class provided by spring framework called as **JdbcTemplate**.

## **46) How JdbcTemplate can be used?**

With use of Spring JDBC framework the burden of resource management and error handling is reduced a lot. So it leaves developers to write the statements and queries to get the data to and from the database.

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

A simple DAO class looks like this.

public class StudentDaoJdbc implements StudentDao {

private **JdbcTemplate** jdbcTemplate;

public void setJdbcTemplate(**JdbcTemplate** jdbcTemplate) {

this.jdbcTemplate = jdbcTemplate;

}

more..

}

The configuration is shown below.

<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">

<property name="dataSource">

<ref bean="dataSource"/>

</property>

</bean>

<bean id="studentDao" class="StudentDaoJdbc">

<property name="jdbcTemplate">

<ref bean="jdbcTemplate"/>

</property>

</bean>

<bean id="courseDao" class="CourseDaoJdbc">

<property name="jdbcTemplate">

<ref bean="jdbcTemplate"/>

</property>

</bean>

## **47) How do you write data to backend in spring using JdbcTemplate?**

The JdbcTemplate uses several of these callbacks when writing data to the database. The usefulness you will find in each of these interfaces will vary. There are two simple interfaces. One is **PreparedStatementCreator** and the other interface is **BatchPreparedStatementSetter**.

## **48) Explain about PreparedStatementCreator?**

PreparedStatementCreator is one of the most common used interfaces for writing data to database. The interface has one method createPreparedStatement().

PreparedStatement **createPreparedStatement**(Connection conn)

throws SQLException;

When this interface is implemented, we should create and return a PreparedStatement from the Connection argument, and the exception handling is automatically taken care off. When this interface is implemented, another interface **SqlProvider** is also implemented which has a method called **getSql()** which is used to provide sql strings to JdbcTemplate.

## **49) Explain about BatchPreparedStatementSetter?**

If the user what to update more than one row at a shot then he can go for **BatchPreparedStatementSetter**. This interface provides two methods

setValues(PreparedStatement ps, int i) throws SQLException;

int getBatchSize();

The getBatchSize() tells the JdbcTemplate class how many statements to create. And this also determines how many times setValues() will be called.

## **50) Explain about RowCallbackHandler and why it is used?**

In order to navigate through the records we generally go for ResultSet. But spring provides an interface that handles this entire burden and leaves the user to decide what to do with each row. The interface provided by spring is **RowCallbackHandler**. There is a method processRow() which needs to be implemented so that it is applicable for each and everyrow.

void **processRow**(java.sql.ResultSet rs);

**34.  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.

**35. Explain the similarities and differences between EJB CMT and the Spring Framework's declarative transaction  
       management ?**

   The basic approach is similar: it is possible to specify transaction behavior (or lack of it) down to individual method level. It is  
    possible to make a setRollbackOnly() call within a transaction context if necessary. The differences are:

|  |
| --- |
|  |

* Unlike EJB CMT, which is tied to JTA, the Spring Framework's declarative transaction management works in any environment. It can work with JDBC, JDO, Hibernate or other transactions under the covers, with configuration changes only.
* The Spring Framework enables declarative transaction management to be applied to any class, not merely special classes such as EJBs.
* The Spring Framework offers declarative rollback rules: this is a feature with no EJB equivalent. Both programmatic and declarative support for rollback rules is provided.
* The Spring Framework gives you an opportunity to customize transactional behavior, using AOP. With EJB CMT, you have no way to influence the container's transaction management other than setRollbackOnly().
* The Spring Framework does not support propagation of transaction contexts across remote calls, as do high-end [application servers](http://www.developersbook.com/spring/interview-questions/spring-interview-questions-faqs-4.php).

**37. 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.

**38. 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.

**39. 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](http://www.developersbook.com/spring/interview-questions/spring-interview-questions-faqs-4.php) 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.

**40. What is SQLExceptionTranslator ?**

SQLExceptionTranslator, is an interface to be implemented by classes that can [translate](http://www.developersbook.com/spring/interview-questions/spring-interview-questions-faqs-4.php) between SQLExceptions and Spring's own data-access-strategy-agnostic org.springframework.dao.DataAccessException.

**What is Spring's JdbcTemplate ?**

Spring's *JdbcTemplate* is central class to interact with a [database](http://www.developersbook.com/spring/interview-questions/spring-interview-questions-faqs-5.php) 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);

|  |
| --- |
|  |

**42. What is PreparedStatementCreator ?**

   PreparedStatementCreator:

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

**43. What is SQLProvider ?**

   SQLProvider:

* Has one method – getSql()
* Typically implemented by PreparedStatementCreator implementers.

|  |
| --- |
|  |

* Useful for debugging.

**44. What is RowCallbackHandler ?**

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

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

**45. What are the differences between EJB and Spring ?**

   Spring and EJB feature comparison.

|  |  |  |
| --- | --- | --- |
| **Feature** | **EJB** | **Spring** |
| Transaction management | * Must use a JTA transaction manager. * Supports transactions that span remote method calls. | * Supports multiple transaction environments through its PlatformTransactionManager interface, including JTA, Hibernate, JDO, and JDBC. * Does not natively support distributed transactions—it must be used with a JTA transaction manager. |
| Declarative transaction support | * Can define transactions declaratively through the deployment descriptor. * Can define transaction behavior per method or per class by using the wildcard character \*. * Cannot declaratively define rollback behavior—this must be done programmatically. | * Can define transactions declaratively through the Spring configuration file or through class metadata. * Can define which methods to apply transaction behavior explicitly or by using regular expressions. * Can declaratively define rollback behavior per method and per exception type. |
| Persistence | Supports programmatic bean-managed persistence and declarative container managed persistence. | Provides a framework for integrating with several persistence technologies, including JDBC, Hibernate, JDO, and iBATIS. |
| Declarative [security](http://www.developersbook.com/spring/interview-questions/spring-interview-questions-faqs-5.php) | * Supports declarative security through users and roles. The management and implementation of users and roles is container specific. * Declarative security is configured in the deployment descriptor. | * No security implementation out-of-the box. * Acegi, an [open source](http://www.developersbook.com/spring/interview-questions/spring-interview-questions-faqs-5.php) security framework built on top of Spring, provides declarative security through the Spring configuration file or class metadata. |
| Distributed computing | Provides container-managed remote method calls. | Provides proxying for remote calls via RMI, JAX-RPC, and [web services](http://www.developersbook.com/spring/interview-questions/spring-interview-questions-faqs-5.php). |
|  |  |  |

### **How Spring relate to MVC framework ?**

In Spring's Web MVC framework : a DispatcherServlet that dispatches requests to handlers.The default handler is a very simple Controller interface, just offering a ModelAndView handleRequest(request,response) method.   
Details with code

How to setup MessageSources in Spring ?

Spring currently provides two MessageSource implementations. These are the ResourceBundleMessageSource and the StaticMessageSource.

### **What is ApplicationContext in details ?**

The basis for the context package is the ApplicationContext interface, located in the org.springframework.context package. Deriving from the BeanFactory interface, it provides all the functionality of BeanFactory. To allow working in a more framework-oriented fashion, using layering and hierarchical contexts, the context package also provides the following functionality:   
  
a)MessageSource, providing access to messages in i18n-style  
  
b)Access to resources, such as URLs and files  
  
c)Event propagation to beans implementing the ApplicationListener interface  
  
d)Loading of multiple (hierarchical) contexts, allowing each to be focused on one particular layer, for example the web layer of an application  
  
As the ApplicationContext includes all functionality of the BeanFactory, it is generally recommended that it be used over the BeanFactory, except for a few limited situations such as perhaps in an Applet, where memory consumption might be critical, and a few extra kilobytes might make a difference

Explain Bean Lifecycle in Spring ?

### **The Spring Framework provides *InitializingBean* and *DisposableBean* marker interface. Implementing these interfaces will result in the container calling afterPropertiesSet() for the former and destroy() for the latter to allow the bean to perform certain actions upon initialization and destruction.** a) Initialization callbacks **There are two ways you can achive the initialization work after all necessary properties on the bean are set by the container.** *1) use 'init-method' in XML configuration* **You can do initialization work in init() method. <bean id="testInitBean" class="com.techfaq.TestBean" init-method="init"/> public class TestBean { public void init() { // do some initialization work } }** *2) Implementing the org.springframework.beans.factory.InitializingBean interface* **You can do initialization work in afterPropertiesSet() method. void afterPropertiesSet() throws Exception; <bean id="testInitBean" class="com.techfaq.TestBean" /> public class TestBean { public void afterPropertiesSet() { // do some initialization work } }** b) Destruction callbacks **Implementing the org.springframework.beans.factory.DisposableBean interface allows a bean to get a callback when the container containing it is destroyed. There are two ways you can achive the destroy.** *1) use 'destroy-method' in XML configuration* **You can do cleanup work in cleanup() method. <bean id="testDestBean" class="com.techfaq.TestBean" destroy-method="cleanup"/> public class TestBean { public void cleanup() { // do some destruction work (like releasing pooled connections) } }** *2) Implementing the org.springframework.beans.factory.DisposableBean interface allows a bean to get a callback when the container containing it is destroyed* **void destroy() throws Exception; <bean id="testDestBean" class="com.techfaq.TestBean" /> public class TestBean { public void destroy() { // do some destruction work (like releasing pooled connections) } } What are the Bean Scopes in Spring ?**

When you create a bean definition what you are actually creating is a recipe for creating actual instances of the class defined by that bean definition.   
  
Spring Framework supports exactly five scopes (of which three are available only if you are using a web-aware ApplicationContext).   
**singleton :**   
This scope available for both BeanFactory and ApplicationContext.  
Scopes a single bean definition to a single object instance per Spring IoC container.   
when you define a bean definition and it is scoped as a singleton, then the Spring IoC container will create exactly one instance of the object defined by that bean definition. This single instance will be stored in a cache of such singleton beans, and all subsequent requests and references for that named bean will result in the cached object being returned.  
singleton scope is the default scope.

|  |
| --- |
| <bean id="emailService" class="com.techfaq.EmailService"/>  <!-- the following is equivalent, singleton scope is the default scope. -->  <bean id="emailService" class="com.techfaq.EmailService" scope="singleton"/> <!-- the following is equivalent --> <bean id="emailService" class="com.techfaq.EmailService" singleton="true"/> |

**prototype :**  
This scope available for both BeanFactory and ApplicationContext.   
Scopes a single bean definition to any number of object instances.   
Create a new bean instance every time a request for that specific bean is made ( is injected into another bean or it is requested via a programmatic getBean() method call on the container) .  


|  |
| --- |
| <!-- create new instance every on request. -->  <bean id="emailService" class="com.techfaq.EmailService" scope="prototype"/> <!-- the following is equivalent --> <bean id="emailService" class="com.techfaq.EmailService" singleton="false"/> |

**request :**   
Scopes a single bean definition to the lifecycle of a single HTTP request; that is each and every 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 like XmlWebApplicationContext.   
If you try using these next scopes with regular Spring IoC containers such as the XmlBeanFactory or ClassPathXmlApplicationContext, you will get an IllegalStateException complaining about an unknown bean scope.

|  |
| --- |
| <bean id="emailService" class="com.techfaq.EmailService" scope="request"/> |

**session :**  
Scopes a single bean definition to the lifecycle of a HTTP Session. Only valid in the context of a web-aware Spring ApplicationContext like XmlWebApplicationContext.   
If you try using these next scopes with regular Spring IoC containers such as the XmlBeanFactory or ClassPathXmlApplicationContext, you will get an IllegalStateException complaining about an unknown bean scope.

|  |
| --- |
| <bean id="emailService" class="com.techfaq.EmailService" scope="session"/> |

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 like XmlWebApplicationContext.   
If you try using these next scopes with regular Spring IoC containers such as the XmlBeanFactory or ClassPathXmlApplicationContext, you will get an IllegalStateException complaining about an unknown bean scope.

|  |
| --- |
| <bean id="emailService" class="com.techfaq.EmailService" scope="globalSession"/> |

How do you integrate struts+spring+Hibernate?

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| This tutorial guide you to Integrate Struts Spring Hibernate in a web application.  Struts : View (User Interface)  Spring : Service Layer  Hibernate : DAO layer    Here is the steps for Integration Struts Spring Hibernate Web Application  **You can download the code with zip..** [**strutsspringhibernate.zip**](http://www.techfaq360.com/tutorial/spring/strutsspringhibernate.zip)**Code is ready..just download and run in tomcat.. Click donate button**  Steps to run the downloded code :  step 1. unzip the downloaded zip file to D:\tomcat\webapps\ directory. step 2. D:\tomcat\bin > stratup.bat step 3. Go to browser and type the URL : http://localhost:8080/strutsspringhibernate/userForm.do    step 4. Enter User Name and Department and submit the "save" button.   **Here is the code Explanation** **Struts Part**  **Step 1. Add ActionServlet in web.xml** **ActionServlet**  ActionServlet is the controller in Struts. ActionServlet loads the struts-config.xml. On load-on-startup the servlet container Instantiate the ActionServlet .  **First Task by ActionServlet :** The ActionServlet takes the Struts Config file name as an init-param.  At startup, in the init() method, the ActionServlet reads the Struts Config file and load into memory.  **Second Task by ActionServlet :** If the user types http://localhost:8081/strutsspringhibernate/userForm.do in the browser URL bar, the URL will be intercepted and processed by the ActionServlet since the URL has a pattern \*.do, with a suffix of "do". Because servlet-mapping is  <servlet-mapping>  <servlet-name>action</servlet-name> <url-pattern>\*.do</url-pattern> </servlet-mapping> **Third Task by ActionServlet :** Then ActionServlet delegates the request handling to another class called **RequestProcessor** by invoking its process() method.   |  | | --- | | <servlet> <servlet-name>action</servlet-name> <servlet-class>org.apache.struts.action.ActionServlet</servlet-class> <init-param> <param-name>config</param-name> <param-value>/WEB-INF/struts-config.xml</param-value> </init-param> <load-on-startup>1</load-on-startup> </servlet> <servlet-mapping> <servlet-name>action</servlet-name> <url-pattern>\*.do</url-pattern> </servlet-mapping> |   **Step 2. add ActionMapping in struts-config.xml**  http://localhost:8081/strutsspringhibernate/userForm.do call to /jsp/userForm.jsp jsp.   |  | | --- | | <?xml version="1.0" encoding="ISO-8859-1" ?>  <!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN" "http://jakarta.apache.org/struts/dtds/struts-config\_1\_2.dtd">  <struts-config> <form-beans> <form-bean name="UserForm"  type="com.UserForm"> </form-bean> </form-beans>    <action-mappings>  <action path="/userForm" type="org.apache.struts.actions.ForwardAction" parameter="/jsp/userForm.jsp"/>  <action path="/saveUser" type="com.UserSaveAction" name="UserForm" scope="request" validate="false" input="/pages/Input.jsp"> <forward name="success" path="/jsp/success.jsp"/> </action>  </action-mappings>   </struts-config> |   Enter username and deptname and click the Save button.  Save button calls /saveUser.do , saveUser.do calls com.UserSaveAction based on above configuration in struts-config.xml.  **Step 3. userForm.jsp and success.jsp**  userForm.jsp is the form where you enter the User Name and Department   |  | | --- | | <%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html"%> <%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean"%>  <html:form action="/saveUser.do" method="post"> User Name : <html:text property="userName" size="30" maxlength="120"/> <br> Department : <html:text property="deptName" size="30" maxlength="120"/> <br> <html:submit>Save</html:submit> </html:form> |   success.jsp   |  | | --- | | <h2>User Data saved </h2> |   **com.UserSaveAction.java** UserSaveAction get the user entered information (RequestProcessor set the user entered data into the UserForm). Get the service from applicationContext.xml using UserService userservice = (UserService) ServiceFinder.getContext(request) .getBean("userservice");  In the Action class just get the data from UserForm and call to userservice.saveUser(userbean);   |  | | --- | | package com;  import javax.servlet.http.HttpServletRequest; import javax.servlet.http.HttpServletResponse;  import org.apache.struts.action.Action; import org.apache.struts.action.ActionForm; import org.apache.struts.action.ActionForward; import org.apache.struts.action.ActionMapping;   public class UserSaveAction extends Action{  public ActionForward execute( ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response) throws Exception{     UserService userservice = (UserService) ServiceFinder.getContext(request) .getBean("userservice");  UserForm userForm = (UserForm) form; UserBean userbean = new UserBean(); userbean.setUserName(userForm.getUserName()); userbean.setDeptName(userForm.getDeptName()); userservice.saveUser(userbean);  return mapping.findForward("success"); }  } |   **com.UserForm.java** UserForm is the Form which contain the entered user information. (RequestProcessor set the user entered data into the UserForm).   |  | | --- | | package com;  import org.apache.struts.action.ActionForm;   public class UserForm extends ActionForm{  String userName; String deptName;  /\*\* \* @return Returns the deptName. \*/ public String getDeptName() { return deptName; } /\*\* \* @param deptName The deptName to set. \*/ public void setDeptName(String deptName) { this.deptName = deptName; } /\*\* \* @return Returns the userName. \*/ public String getUserName() { return userName; } /\*\* \* @param userName The userName to set. \*/ public void setUserName(String userName) { this.userName = userName; } } |  **Spring Part**  **Step 1. Add ContextLoaderServlet in web.xml** **ContextLoaderServlet** ContextLoaderServlet register an ApplicationContext using the the below configuration in web.xml file.  You can get any service (Defind in applicationContext.xml) using request.getSession().getServletContext().getBean("userservice");   |  | | --- | | <param-name>contextConfigLocation</param-name> <param-value>/WEB-INF/applicationContext.xml</param-value> </context-param>  <servlet> <servlet-name>context</servlet-name> <servlet-class>org.springframework.web.context.ContextLoaderServlet</servlet-class> <load-on-startup>1</load-on-startup> </servlet> |   **Step 2. applicationContext.xml**  applicationContext.xml contains all the Dependency Injections . And it contains hibernate SessionFactory information also. Here Hibernate integatred with Spring.  userservice object has userdao object as setter Injection  userdao object has sessionFactory object as setter Injection.   |  | | --- | | <?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/spring-beans.dtd">  <beans>  <bean id="myDataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close"> <property name="driverClassName"><value>com.mysql.jdbc.Driver</value></property> <property name="url"><value>jdbc:mysql://localhost:3306/techfaq</value></property> <property name="username"><value>techfaq</value></property> <property name="password"><value>techfaq</value></property>  </bean> <bean id="mySessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean"> <property name="dataSource"><ref bean="myDataSource"/></property> <property name="mappingResources"> <list> <value>/com/user.hbm.xml</value> </list> </property> <property name="hibernateProperties"> <value> hibernate.dialect=org.hibernate.dialect.MySQLDialect </value> </property> </bean>   <bean id="userdao" class="com.UserDAO"> <property name="sessionFactory"><ref bean="mySessionFactory"/></property>  </bean>  <bean id="userservice" class="com.UserService"> <property name="userdao"><ref bean="userdao"/></property>  </bean>  </beans> |   **com.UserService.java**   |  | | --- | | package com;  public class UserService { private UserDAO userdao;  public void saveUser(UserBean user){ getUserdao().saveUser(user); }  /\*\* \* @return Returns the userdao. \*/ public UserDAO getUserdao() { return userdao; } /\*\* \* @param userdao The userdao to set. \*/ public void setUserdao(UserDAO userdao) { this.userdao = userdao; } } |   **com.ServiceFinder.java**   |  | | --- | | package com;   import org.springframework.context.ApplicationContext;   import org.springframework.web.context.support.WebApplicationContextUtils;    import javax.servlet.ServletRequest;  import javax.servlet.http.HttpServletRequest;     public class ServiceFinder {     public static ApplicationContext getContext(HttpServletRequest httpRequest) { return WebApplicationContextUtils.getRequiredWebApplicationContext( httpRequest.getSession().getServletContext());  }  } |  **Hibernate Part**  **Step 1. Create Table in Data Base**   |  | | --- | | CREATE TABLE `techfaq`.`user\_test`  ( `user\_id` int(10) unsigned NOT NULL auto\_increment, `user\_name` varchar(45) default NULL, `dept\_name` varchar(45) default NULL,  PRIMARY KEY (`user\_id`)); |   **Step 2. user.hbm.xml which maps to user\_test TABLE**   |  | | --- | | <?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">  <hibernate-mapping> <class name="com.UserBean" table="USER\_TEST"> <id name="userId" column="USER\_ID" type="int"> <generator class="native"/> </id>  <property name="userName" column="user\_name"/> <property name="deptName" column="dept\_name"/>     </class>  </hibernate-mapping> |   **Step 3. UserBean.java which maps to user.hbm.xml and user\_test TABLE**   |  | | --- | | package com;   public class UserBean { int userId; String userName; String deptName;    /\*\* \* @return Returns the userName. \*/ public String getUserName() { return userName; } /\*\* \* @param userName The userName to set. \*/ public void setUserName(String userName) { this.userName = userName; }  /\*\* \* @return Returns the deptName. \*/ public String getDeptName() { return deptName; } /\*\* \* @param deptName The deptName to set. \*/ public void setDeptName(String deptName) { this.deptName = deptName; } /\*\* \* @return Returns the userId. \*/ public int getUserId() { return userId; } /\*\* \* @param userId The userId to set. \*/ public void setUserId(int userId) { this.userId = userId; } } |   **Step 4. UserDAO.java which is used to save user details into Data Base**   |  | | --- | | package com;   import org.hibernate.Session; import org.hibernate.SessionFactory; import org.hibernate.Transaction;  import org.springframework.orm.hibernate.support.HibernateDaoSupport;   public class UserDAO {  private SessionFactory sessionFactory; public void saveUser(UserBean user){ Session session = getSessionFactory().openSession(); try { Transaction tx = session.beginTransaction(); session.save(user); tx.commit(); }catch(Exception e){ e.printStackTrace(); }finally{ session.close(); }  }   /\*\* \* @return Returns the sessionFactory. \*/ public SessionFactory getSessionFactory() { return sessionFactory; } /\*\* \* @param sessionFactory The sessionFactory to set. \*/ public void setSessionFactory(SessionFactory sessionFactory) { this.sessionFactory = sessionFactory; } } | | |  | | --- | | [SCJP 5.0 Simulator Exam Kit](http://www.techfaq360.com/scjp_mock_test.jsp)  [SCJP 6.0 Simulator Exam Kit](http://www.techfaq360.com/scjp_mock_test.jsp)  [SCWCD5.0 Simulator Exam Kit](http://www.techfaq360.com/scwcd.jsp)  [SCWCD4.0 Simulator Exam Kit](http://www.techfaq360.com/scwcd.jsp)  [OCA 10g Simulator Exam Kit](http://www.techfaq360.com/ocp_mock_test.jsp)  [**Your Ad Here**](http://www.adbrite.com/mb/commerce/purchase_form.php?opid=753533&afsid=1)  [SCJP 5.0 Simulator Free Trial](http://www.techfaq360.com/scjp_mock_test.jsp)  [SCJP 6.0 Simulator Free Trial](http://www.techfaq360.com/scjp_mock_test.jsp)  [SCWCD5.0 Simulator Free Trial](http://www.techfaq360.com/scwcd.jsp)  [SCWCD4.0 Simulator Free Trial](http://www.techfaq360.com/scwcd.jsp)  [OCA 10g Simulator Free Trial](http://www.techfaq360.com/ocp_mock_test.jsp) | |

How do you Spring integration with Hibernate ?

Steps to Integrate Spring with Hibernate

**Step 1. Create SessionFactory**

Spring allows you to define resources like a JDBC DataSource or a Hibernate SessionFactory as beans in an application context. Set up a JDBC DataSource and a Hibernate SessionFactory on top of it.

|  |
| --- |
| <beans> <bean id="myDataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close"> <property name="driverClassName" value="org.hsqldb.jdbcDriver"/> <property name="url" value="jdbc:hsqldb:hsql://localhost:9001"/> <property name="username" value="sa"/> <property name="password" value="sa"/> </bean> <bean id="mySessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean"> <property name="dataSource" ref="myDataSource"/> <property name="mappingResources"> <list> <value>emp.hbm.xml</value> </list> </property> <property name="hibernateProperties"> <value> hibernate.dialect=org.hibernate.dialect.HSQLDialect </value> </property> </bean> </beans> |

**Step 2. Define HibernateTemplate and Create HibernateTemplate**

The HibernateTemplate class provides many methods that mirror the methods exposed on the Hibernate Session interface.  
Define DAO object and inject Session Factory.

|  |
| --- |
| <beans> <bean id="empDao" class="com.techfaq.EmpDAO"> <property name="sessionFactory" ref="mySessionFactory"/> </bean> </beans> |

EmpDAO.java class

|  |
| --- |
| public class EmpDAO {  private HibernateTemplate hibernateTemplate;  public void setSessionFactory(SessionFactory sessionFactory) { this.hibernateTemplate = new HibernateTemplate(sessionFactory); }  public Collection getEmpByDept(String dept) throws DataAccessException { return this.hibernateTemplate.find("from com.bean.Emp e where e.dept=?", dept); } } |

OR You can use

**Plain Hibernate to access data base.**

Define DAO object and inject Session Factory.

|  |
| --- |
| <beans> <bean id="empDao" class="com.techfaq.EmpDAO"> <property name="sessionFactory" ref="mySessionFactory"/> </bean> </beans> |

EmpDAO.java class

|  |
| --- |
| public class EmpDAO{  private SessionFactory sessionFactory;  public void setSessionFactory(SessionFactory sessionFactory) { this.sessionFactory = sessionFactory; }  public Collection getEmpByDept(String dept) { return this.sessionFactory.getCurrentSession() .createQuery("from com.bean.Emp e where e.dept=?") .setParameter(0, dept) .list(); } } |

OR You can use

**HibernateDaoSupport.**

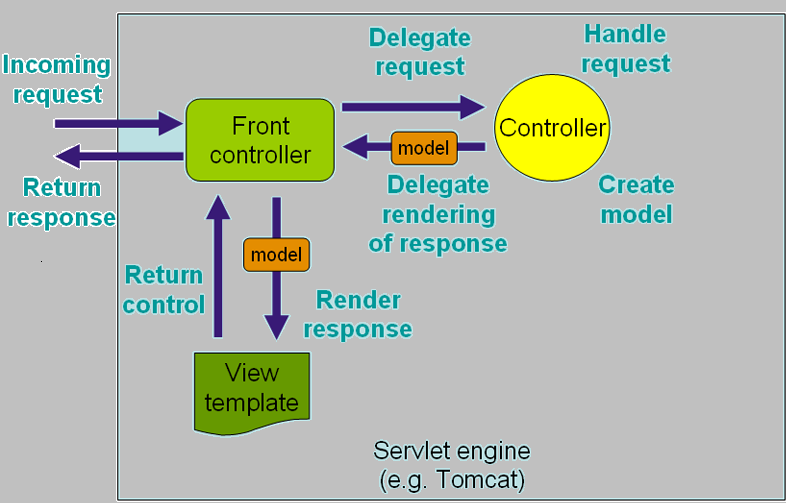
HibernateDaoSupport base class offers methods to access the current transactional Session. EmpDAO.java class

|  |
| --- |
| public class EmpDAO extends HibernateDaoSupport {  public Collection getEmpByDept(String dept) throws DataAccessException, MyException { Session session = getSession(false); try { Query query = session.createQuery("from com.bean.Emp e where e.dept=?"); query.setString(0, dept); List result = query.list(); if (result == null) { throw new MyException("No results."); } return result; } catch (HibernateException ex) { throw convertHibernateAccessException(ex); } } } |

How Spring relate to MVC framework?

## **Web MVC framework**

In Spring's Web MVC framework : a DispatcherServlet that dispatches requests to handlers.The default handler is a very simple Controller interface, just offering a ModelAndView handleRequest(request,response) method.



Overview of the Spring Framework

### **DispatcherServlet**

DispatcherServlet a central servlet that dispatches requests to controllers and offers other functionality facilitating the development of web applications. DispatcherServlet completely integrated with the Spring IoC container and allows you to use every other feature that Spring has.   
The DispatcherServlet is an Servlet declared in the web.xml.  
Consider the following DispatcherServlet servlet configuration (in the web.xml file.)

|  |
| --- |
| <web-app> ... <servlet> <servlet-name>test</servlet-name> <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class> <load-on-startup>1</load-on-startup> </servlet> <servlet-mapping> <servlet-name>test</servlet-name> <url-pattern>\*.do</url-pattern> </servlet-mapping> </web-app> |

With the above servlet configuration , you will need to have a file called '/WEB-INF/**test**-servlet.xml' in your application; this file will contain all of your Spring Web MVC-specific components (beans).   
The WebApplicationContext is an extension of the plain ApplicationContext that has some extra features necessary for web applications  
The Spring DispatcherServlet has a couple of special beans it uses in order to be able to process requests and render the appropriate views. These beans are included in the Spring framework and can be configured in the WebApplicationContext

### **Controllers**

controller is the org.springframework.web.servlet.mvc.Controller interface, the source code for which is listed below. 

|  |
| --- |
| public interface Controller { /\*\* \* Process the request and return a ModelAndView object which the DispatcherServlet \* will render. \*/ ModelAndView handleRequest( HttpServletRequest request, HttpServletResponse response) throws Exception; } |

As you can see, the Controller interface defines a single method that is responsible for handling a request and returning an appropriate model and view.  
Command controllers provide a way to interact with data objects and dynamically bind parameters from the HttpServletRequest to the data object specified.  
**AbstractCommandController** : This class does not offer form functionality;  
**AbstractFormController** : an abstract controller offering form submission support.After a user has filled the form, the AbstractFormController binds the fields, validates the command object, and hands the object back to the controller to take the appropriate action.  
**SimpleFormController** : a form controller that provides even more support when creating a form with a corresponding command object. The SimpleFormController let's you specify a command object, a viewname for the form, a viewname for page you want to show the user when form submission has succeeded.  
**AbstractWizardFormController** : as the class name suggests, this is an abstract class - your wizard controller should extend it. This means you have to implement the validatePage(), processFinish() and processCancel() methods.

### **Handler mappings**

Using a handler mapping you can map incoming web requests to appropriate handlers. the *SimpleUrlHandlerMapping* or the *BeanNameUrlHandlerMapping* you can use. **BeanNameUrlHandlerMapping**: A very simple, but very powerful handler mapping is the BeanNameUrlHandlerMapping, which maps incoming HTTP requests to names of beans, defined in the web application context.  
<bean id="defaultHandlerMapping" class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"/>   
**SimpleUrlHandlerMapping**:powerful handler mapping - is the SimpleUrlHandlerMapping. This mapping is configurable in the application context  
<bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">  
<property name="mappings">  
<value>  
/\*/account.form=editAccountFormController  
/\*/editaccount.form=editAccountFormController  
/ex/view\*.html=helpController  
/\*\*/help.html=helpController  
</value>  
</property>  
</bean>

### **Views**

**Resolving views - the ViewResolver interface**: All controllers in the Spring Web MVC framework return a ModelAndView instance. Views in Spring are addressed by a view name and are resolved by a view resolver. Example :  
<bean id="viewResolver"  
class="org.springframework.web.servlet.view.UrlBasedViewResolver">  
<property name="prefix" value="/WEB-INF/jsp/"/>  
<property name="suffix" value=".jsp"/>  
</bean>  
When returning "successaccount" as a viewname, this view resolver will hand the request over to the RequestDispatcher that will send the request to /WEB-INF/jsp/successaccount.jsp.

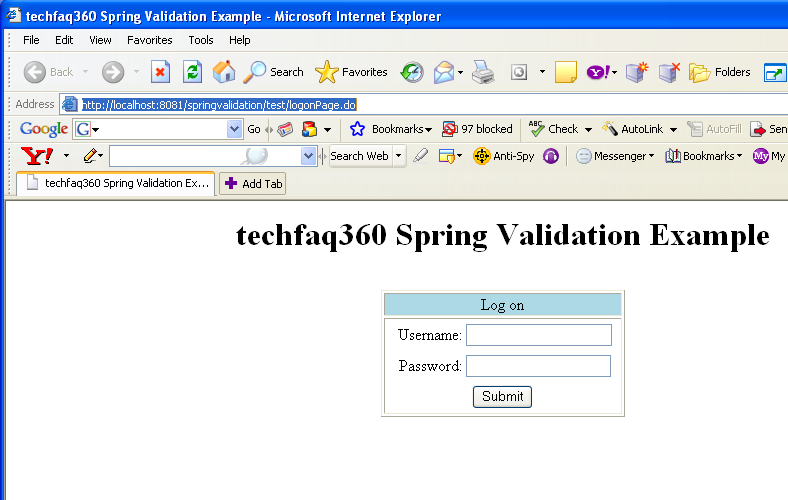
### **How Validation Framework work in Spring with code Example ?**

## **Developing Your First Spring Web Application with Spring Validation Framework with Code Example**

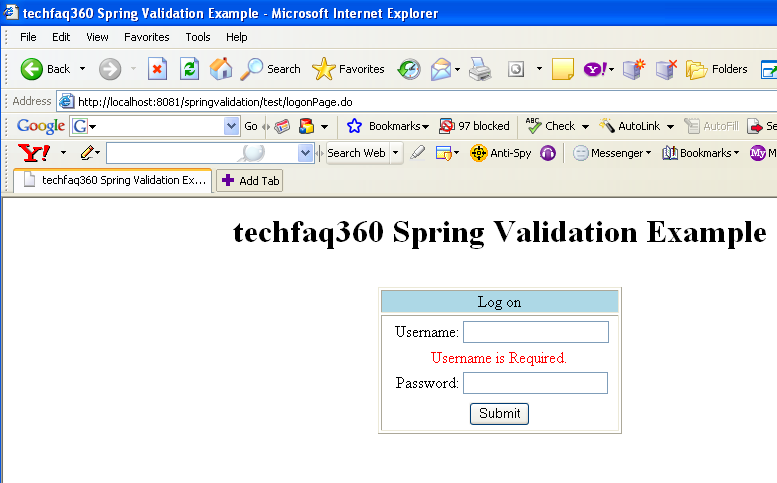
Here is the steps how to setup Your First Spring Web Application with Spring Validation Framework with Code Example

**You can download the code with zip** [**springvalidation.zip**](http://www.techfaq360.com/tutorial/spring/springvalidation.zip) **..Code is ready..just download and run in tomcat.. Click donate button**

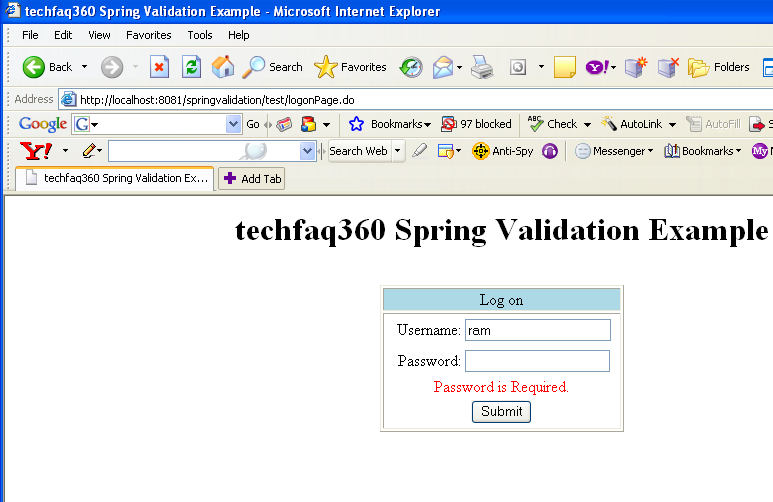
Steps to run the downloded code :   
step 1. unzip the downloaded zip file to D:\tomcat\webapps\ directory.  
step 2. D:\tomcat\bin > stratup.bat  
step 3. Go to browser and type the URL : http://localhost:8081/springvalidation/test/logonPage.do



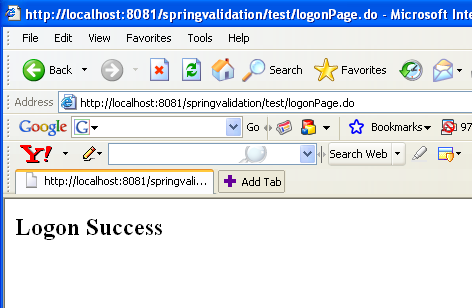
step 4. Don't Enter username and password and submit the "submit" button.



step 5. Now Enter username and don't enter password and submit the "submit" button.



step 6. Now Enter username and password both and submit the "submit" button. You will redirect to success page



### **Validator Flow In Spring**

**Step a. if your URL is http://localhost:8081/springvalidation/test/logonPage.do**   
The "/test/logonPage.do" call the *formBackingObject(HttpServletRequest request)* of the mentioned controller.  
In our case *LogonController* (This is mapped in **test**-servlet.xml of this application).   
*formBackingObject(HttpServletRequest request)* method set the FormBean with the default value. You can sent any value.   
and get the **formView** property from the **test**-servlet.xml and forward to the mentioned JSP.   
In our case (<property name="formView"><value>logonForm</value></property>);   
So it forwared to **logonForm.jsp** . Name should be same JSP and formView property.   
  
  
**Step b. The JSP page fills in the form fields with the information in the "command" object.**   
The JSP page fills in the form fields with the information in the "command" object.   
In our case "command" object is **userBean**   
<property name="commandName"><value>userBean</value></property>   
<property name="commandClass"><value>com.UserBean</value></property>   
When the form is submitted, the details are passed to the Validator servlet bean for validation.   
In our case **"logonFormValidator"**   
<property name="validator"><ref bean="logonFormValidator"/></property>   
  
  
**Step c. This checks the form data for errors. If there are no errors, it returns control directly to the Controller servlet bean. onSubmit() method of LogonController**  
  
  
**Step d. If there were errors in the form, then the Validator servlet bean calls the JSP page again. This time, the "command" object contains the data previously entered by the user, so the JSP page fills in the form fields exactly as before.**

### **Here is the code Explanation**

**Step 1. Set the DispatcherServlet in web.xml**

#### **DispatcherServlet**

DispatcherServlet a central servlet that dispatches requests to controllers and offers other functionality facilitating the development of web applications. DispatcherServlet completely integrated with the Spring IoC container and allows you to use every other feature that Spring has.   
The DispatcherServlet is an Servlet declared in the web.xml.  
Consider the following DispatcherServlet servlet configuration (in the web.xml file.)

|  |
| --- |
| <web-app> <servlet> <servlet-name>test</servlet-name> <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class> <load-on-startup>1</load-on-startup> </servlet> <servlet-mapping> <servlet-name>test</servlet-name> <url-pattern>\*.do</url-pattern> </servlet-mapping> </web-app> |

With the above servlet configuration , you will need to have a file called '/WEB-INF/**test**-servlet.xml' in your application.

**Step 2. test-servlet.xml file**

test-servlet.xml file contains viewResolver , Handler mappings and Controllers.  
**viewResolver :**   
All controllers in the Spring Web MVC framework return a ModelAndView instance. Views in Spring are addressed by a view name and are resolved by a view resolver. For Example : if your controller return new ModelAndView("success"); means control forwared to "/WEB-INF/jsp/success.jsp" based on below configuration.   
When returning "success" as a viewname, this view resolver will hand the request over to the RequestDispatcher that will send the request to /WEB-INF/jsp/success.jsp.

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

**SimpleUrlHandlerMapping**:   
The "/test/logonPage.do" call the logonController.  
<bean id="urlMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">  
<property name="urlMap">  
<map>  
<entry key="/test/logonPage.do"><ref bean="logonController"/></entry>  
</map>  
</property>  
</bean>  
**Controllers** :   
<bean id="logonController" class="com.LogonController">  
<property name="sessionForm"><value>true</value></property>   
<property name="commandName"><value>userBean</value></property>  
<property name="commandClass"><value>com.UserBean</value></property>   
<property name="validator"><ref bean="logonFormValidator"/></property>  
<property name="formView"><value>logonForm</value></property>  
<property name="successView"><value>success</value></property>  
</bean>  
In the above configuation   
  
<property name="sessionForm"><value>true</value></property>   
Keep command object throughout session   
**validator**:   
<bean id="logonFormValidator" class="com.LogonFormValidator"/>

**Step 3. Controller classe**

"/test/logonPage.do" call the *formBackingObject(HttpServletRequest request)* of the controller.  
*formBackingObject(HttpServletRequest request)* method set the FormBean with the default value. This method is used to display the form.   
And onSubmit() method will be called after click on submit button of the form.   
LogonController.java

|  |
| --- |
| package com;  import javax.servlet.ServletException; import javax.servlet.http.HttpServletRequest;  import org.springframework.web.servlet.ModelAndView; import org.springframework.web.servlet.mvc.SimpleFormController;   public class LogonController extends SimpleFormController{  public Object formBackingObject(HttpServletRequest request) throws ServletException { UserBean backingObject = new UserBean(); System.out.println("formBackingObject");  /\* The backing object should be set up here, with data for the initial values \* of the form’s fields. This could either be hard-coded, or retrieved from a \* database. \*/   return backingObject; }  public ModelAndView onSubmit(Object command) throws ServletException { UserBean user = (UserBean)command; System.out.println("username :"+user.getUserName()); System.out.println("password :"+user.getPassword()); //Now you can validate to database return new ModelAndView("succes"); }    } |

**Step 4. Validator classe**

Validator class need to implement Validator interface .It only has two methods. The supports() method is used to check whether the validator supports a given class, and the validate() method is used to actually validate an object of the supported class.  
LogonFormValidator.java

|  |
| --- |
| package com;  import org.springframework.validation.Errors; import org.springframework.validation.Validator;   public class LogonFormValidator implements Validator{  public boolean supports(Class clazz) { return clazz.equals(UserBean.class); }  public void validate(Object obj, Errors errors) { UserBean user = (UserBean) obj; if (user == null) { errors.rejectValue("username", "error.login.not-specified", null,"Value required."); } else {   if (user.getUserName()== null || user.getUserName().trim().length() <= 0) { System.out.println("user name null value"); errors.rejectValue("userName", "error.login.invalid-user", null, "Username is Required."); } else { if (user.getPassword()== null || user.getPassword().trim().length() <= 0) { errors.rejectValue("password", "error.login.invalid-pass", null, "Password is Required."); } }  } } } |

**Step 5. Bean classe**

UserBean.java

|  |
| --- |
| package com;   public class UserBean { String userName; String password;  /\*\* \* @return Returns the password. \*/ public String getPassword() { return password; } /\*\* \* @param password The password to set. \*/ public void setPassword(String password) { this.password = password; } /\*\* \* @return Returns the userName. \*/ public String getUserName() { return userName; } /\*\* \* @param userName The userName to set. \*/ public void setUserName(String userName) { this.userName = userName; } } |

**Step 6. JSP code logonForm.jsp and success.jsp**

In the JSP page you have to bind <spring:bind path="userBean.userName">  
where userBean = commandName in the test-servlet.xml   
userName = field name in the com.UserBean class.   
logonForm.jsp

|  |
| --- |
| <%@ taglib prefix="core" uri="http://java.sun.com/jstl/core" %> <%@ taglib prefix="fmt" uri="http://java.sun.com/jstl/fmt" %> <%@ taglib prefix="str" uri="http://jakarta.apache.org/taglibs/string-1.1" %> <%@ taglib prefix="spring" uri="/spring" %>   <html> <head><title>techfaq360 Spring Validation Example</title></head> <body>  <center>  <h1>techfaq360 Spring Validation Example</h1> <br/>  <form method="post" action="/springvalidation/test/logonPage.do"> <table width="25%" border="1"> <tr> <td align="center" bgcolor="lightblue">Log on</td> </tr> <tr> <td> <table border="0" width="100%"> <tr> <td width="33%" align="right">Username: </td> <td width="66%" align="left"> <spring:bind path="userBean.userName"> <input type="text"  name="userName"  value="<core:out value="${status.value}"/>"/>  </spring:bind> </td>  </tr> <tr> <td colspan="2" align="center"> <spring:hasBindErrors name="userBean"> <font color="red"><core:out value="${status.errorMessage}"/></font> </spring:hasBindErrors> </td> </tr> <tr> <td width="33%" align="right">Password: </td> <td width="66%" align="left"> <spring:bind path="userBean.password"> <input type="password" name="password" /> </spring:bind> </td> </tr> <tr> <td colspan="2" align="center"> <spring:hasBindErrors name="userBean"> <font color="red"><core:out value="${status.errorMessage}"/></font> </spring:hasBindErrors> </td> </tr> <tr> <td align="center" colspan="2"> <input type="submit" alignment="center" value="Submit"> </td> </tr> </table>  </td> </tr> </table>  </form>  </center>  </body> </html> |

succes.jsp

|  |
| --- |
| <h2>Logon Success </h2> |

How spring Setter Injection work with List, Set, Map and Properties ?

## **Collections Setter Injection**

This is the example how Java Collection type List, Set, Map and Properties used in Setter Injection.

|  |
| --- |
| <bean id="empBean" class="com.techfaq.EmpBean"> <property name="adminEmails"> <props> <prop key="administrator">administrator@somecompany.org</prop> <prop key="support">support@somecompany.org</prop> <prop key="development">development@somecompany.org</prop> </props> </property> <property name="addressList"> <list> <ref bean="address" /> </list> </property> <property name="parameters"> <map> <entry key="TRANS\_ID"> <value>65</value> </entry> <entry key="SERVER\_NAME"> <value>tsserver1</value> </entry> <entry key="PROCESS\_CODE"> <value>PCXYZ</value> </entry> </map> </property> <property name="roleSet"> <set> <ref bean="roleBean" /> </set> </property> </bean> |

  
The Java Code is (EmpBean.java)

|  |
| --- |
| package com.techfaq; public class EmpBean { private Map parameters; private Set roleSet; private List addressList; private Properties adminEmails; public Map getParameters() { return parameters; } public void setParameters(Map parameters) { this.parameters = parameters; } /\*\* \* @return Returns the addressList. \*/ public List getAddressList() { return addressList; } /\*\* \* @param addressList The addressList to set. \*/ public void setAddressList(List addressList) { this.addressList = addressList; } /\*\* \* @return Returns the adminEmails. \*/ public Properties getAdminEmails() { return adminEmails; } /\*\* \* @param adminEmails The adminEmails to set. \*/ public void setAdminEmails(Properties adminEmails) { this.adminEmails = adminEmails; } /\*\* \* @return Returns the roleSet. \*/ public Set getRoleSet() { return roleSet; } /\*\* \* @param roleSet The roleSet to set. \*/ public void setRoleSet(Set roleSet) { this.roleSet = roleSet; } } |

### **Explain Dependency Injection with code Example ?**

The basic principle behind **Dependency Injection (DI)** is that objects define their dependencies . Then, it is the job of the container to actually inject those dependencies when it creates the bean.   
The basic concept of the Inversion of Control pattern (also known as **Dependency Injection (DI)** ) 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. In a typical IOC scenario, the container creates all the objects, wires them together by setting the necessary properties, and determines when methods will be invoked.  
  
The two major flavors of Dependency Injection are **Setter Injection (injection via JavaBean setters)** and **Constructor Injection (injection via constructor arguments)**.

### **Setter Injection:**

Find below an example of a class that can only be dependency injected using pure setter injection.  
XML configuration for setter Dependency Injection is below

|  |
| --- |
| <bean id="empBean" class="com.techfaq.EmpBean"> <!-- setter injection using the nested <ref/> element --> <property name="addressBean"><ref bean="addressBean"/></property> </bean> <bean id="addressBean" class="com.techfaq.AddressBean"/> |

The Java Code is (EmpBean.java)

|  |
| --- |
| package com.techfaq; public class EmpBean { private AddressBean addressBean; public void setAddressBean(AddressBean addressBean) { this.addressBean = addressBean; } public AddressBean getAddressBean() { return addressBean; } } |

The Java Code is (AddressBean.java)

|  |
| --- |
| package com.techfaq; public class AddressBean { private int id; public void setId(int id) { this.id = id; } public int getId() { return id; } } |

### **Constructor Injection:**

Find below an example of a class that can only be dependency injected using pure Constructor injection.  
XML configuration for Constructor Dependency Injection is below

|  |
| --- |
| <bean id="empBean" class="com.techfaq.EmpBean"> <!-- setter injection using the nested <ref/> element --> <constructor-arg><ref bean="addressBean"/></constructor-arg> <!--OR you can use <constructor-arg ref="yetAnotherBean"/>--> </bean> <bean id="addressBean" class="com.techfaq.AddressBean"/> |

The Java Code is (EmpBean.java)

|  |
| --- |
| package com.techfaq; public class EmpBean { private AddressBean addressBean; public EmpBean(AddressBean addressBean) { this.addressBean = addressBean; } public AddressBean getAddressBean() { return addressBean; } } |

The Java Code is (AddressBean.java)

|  |
| --- |
| package com.techfaq; public class AddressBean { private int id; public void setId(int id) { this.id = id; } public int getId() { return id; } } |

### **What is IOC-Inversion of Control ?**

**Core** package is the most fundamental part of the framework and provides the IoC and Dependency Injection features. The basic concept here is the BeanFactory, which provides a sophisticated implementation of the factory pattern which removes the need for programmatic singletons and allows you to decouple the configuration and specification of dependencies from your actual program logic.BeanFactory create object for you based on XML configuration.

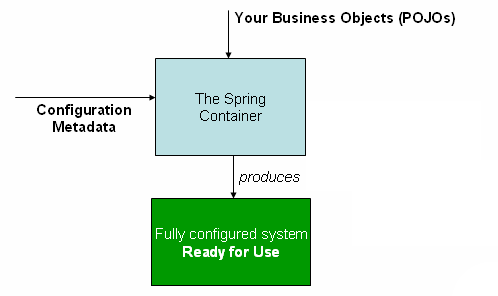
The **org.springframework.beans** and **org.springframework.context** packages provide the basis for the Spring Framework's IoC container. The **BeanFactory** interface provides an advanced configuration mechanism capable of managing objects of any nature. The **ApplicationContext** interface builds on top of the BeanFactory (it is a sub-interface) and adds other functionality such as easier integration with Spring's AOP features, message resource handling (for use in internationalization), event propagation, and application-layer specific contexts such as the WebApplicationContext for use in web applications.  
  
**BeanFactory** provides the configuration framework and basic functionality, while the **ApplicationContext** adds more enterprise-centric functionality to it. The ApplicationContext is a complete superset of the BeanFactory, and any description of BeanFactory capabilities and behavior is to be considered to apply to the ApplicationContext as well.

### **Beans :**

Objects that form the backbone of your application and that are managed by the Spring IoC container are referred to as beans. A bean is simply an object that is instantiated, assembled and otherwise managed by a Spring IoC container.   


### **The container**

:   
The *org.springframework.beans.factory.BeanFactory* is the actual representation of the Spring IoC container that is responsible for containing and managing beans.   
The **BeanFactory** interface is the central IoC container interface in Spring. Its responsibilities include instantiating or sourcing application objects, configuring such objects, and assembling the dependencies between these objects.   
The most commonly used **BeanFactory** implementation is the **XmlBeanFactory** class. This implementation allows you to express the objects that compose your application, and interdependencies between such objects, in terms of XML. The **XmlBeanFactory** takes this XML configuration metadata and uses it to create a fully configured system or application.



The Spring IoC container

### **What do you mean by Advice? and what are the Advices?**

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.   
  
Types of advice:   
  
  
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).   
  
  
After returning advice: Advice to be executed after a join point completes normally: for example, if a method returns without throwing an exception.   
  
  
After throwing advice: Advice to be executed if a method exits by throwing an exception.   
  
  
After (finally) advice: Advice to be executed regardless of the means by which a join point exits (normal or exceptional return).   
  
  
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

### **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.

### **How to integrate Struts Spring Hibernate ?**

Step 1.   
In the struts-config.xml add plugin   
<plug-in className="org.springframework.web.struts.ContextLoaderPlugIn">   
<set-property property="contextConfigLocation"   
value="/WEB-INF/applicationContext.xml"/>   
</plug-in>   
  
  
Step 2.   
  
In the applicationContext.xml file   
Configure datasourse   
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">   
<property name="driverClassName"><value>oracle.jdbc.driver.OracleDriver</value>   
</property>   
<property name="url"><value>jdbc:oracle:thin:@10.10.01.24:1541:ebizd</value>   
</property>   
<property name="username"><value>sa</value></property>   
     <property name="password"><value></value></property>   
</bean>   
  
  
Step 3.   
  
Configure SessionFactory   
<!-- Hibernate SessionFactory -->   
<bean id="sessionFactory" class="org.springframework.orm.hibernate.LocalSessionFactoryBean">   
<property name="dataSource"><ref local="dataSource"/></property>   
<property name="mappingResources">   
<list>   
<value>com/test/dbxml/User.hbm.xml</value>   
</list>   
</property>   
<property name="hibernateProperties">   
<props>   
<prop key="hibernate.dialect">net.sf.hibernate.dialect.OracleDialect </prop>   
</props>   
</property>   
</bean>   
  
  
Step 4.   
Configure User.hbm.xml   
<hibernate-mapping>   
<class name="org.test.model.User" table="app\_user">   
  
<id name="id" column="id" >   
<generator class="increment"/>   
</id>   
<property name="firstName" column="first\_name" not-null="true"/>   
<property name="lastName" column="last\_name" not-null="true"/>   
  
</class>   
</hibernate-mapping>   
  
  
Step 5.   
  
In the applicationContext.xml ? configure for DAO   
<bean id="userDAO" class="org.test.dao.hibernate.UserDAOHibernate">   
<property name="sessionFactory"><ref local="sessionFactory"/></property>   
</bean>   
  
  
Step 6.   
  
DAO Class   
public class UserDAOHibernate extends HibernateDaoSupport implements UserDAO {   
private static Log log = LogFactory.getLog(UserDAOHibernate.class);   
  
public List getUsers() {   
return getHibernateTemplate().find("from User");   
}   
  
public User getUser(Long id) {   
return (User) getHibernateTemplate().get(User.class, id);   
}   
  
public void saveUser(User user) {   
getHibernateTemplate().saveOrUpdate(user);   
  
if (log.isDebugEnabled()) {   
log.debug("userId set to: " + user.getId());   
}   
}   
  
public void removeUser(Long id) {   
Object user = getHibernateTemplate().load(User.class, id);   
getHibernateTemplate().delete(user);   
}   
}

### **How to integrate Spring with Hibernate ?**

There are two steps   
step 1.   
In the bean.xml which is realted to bean factory spring.   
<beans>   
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">   
<property name="driverClassName">   
<value>org.hsqldb.jdbcDriver</value>   
</property>   
<property name="url">   
<value>jdbc:hsqldb:data/tutorial</value>   
</property>   
<property name="username">   
<value>sa</value>   
</property>   
<property name="password">   
<value></value>   
</property>   
</bean>   
<bean id="factory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">   
<property name="mappingResources">   
<list>   
<value>Event.hbm.xml</value>   
</list>   
</property>   
<property name="hibernateProperties">   
<props>   
<prop key="hibernate.dialect">org.hibernate.dialect.HSQLDialect</prop>   
<prop key="hibernate.show\_sql">false</prop>   
</props>   
</property>   
<property name="dataSource">   
<ref bean="dataSource"/>   
</property>   
</bean>   
<bean id="eventDao" class="EventSpringDao">   
<property name="sessionFactory">   
<ref bean="factory" />   
</property>   
</bean>   
</beans>   
  
EventSpringDao class has setter and getter method of sessionFactory.   
  
Step 2.   
  
In the EventSpringDao class you can do getSessionFactory().openSession() to access data base.   
  
or   
your class you can extend HibernateDaoSupport.   
Exmaple:   
public EventSpringDao extends HibernateDaoSupport{   
public void saveOrUpdate(Event event){   
getHibernateTemplate().saveOrUpdate(event);   
}   
  
}

### **What is JdbcTemplate in Spring ?**

  
The JdbcTemplate class is the central class in the JDBC core package. It simplifies the use of JDBC since it handles the creation and release of resources. This helps to avoid common errors such as forgetting to always close the connection. It executes the core JDBC workflow like statement creation and execution, leaving application code to provide SQL and extract results. This class executes SQL queries, update statements or stored procedure calls, imitating 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.   
  
In the bean.xml file   
  
<?xml version="1.0" encoding="UTF-8"?>   
<beans xmlns="http://www.springframework.org/schema/beans"   
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"   
xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">   
  
<bean id="eventDao" class="com.example.JdbcEventDao">   
<property name="dataSource" ref="dataSource"/>   
</bean>   
  
<!-- the DataSource (parameterized for configuration via a PropertyPlaceHolderConfigurer) -->   
<bean id="dataSource" destroy-method="close" class="org.apache.commons.dbcp.BasicDataSource">   
<property name="driverClassName" value="${jdbc.driverClassName}"/>   
<property name="url" value="${jdbc.url}"/>   
<property name="username" value="${jdbc.username}"/>   
<property name="password" value="${jdbc.password}"/>   
</bean>   
  
</beans>   
  
And In Java class   
  
public class JdbcEventDao implements EventDao {   
  
private JdbcTemplate jdbcTemplate;   
  
public void setDataSource(DataSource dataSource) {   
this.jdbcTemplate = new JdbcTemplate(dataSource);   
}   
  
/// DO METHODS   
}

### **What is AOP - Aspect-oriented programming ?**

  
the system service like logging, transaction management, security etc., must be included in the program in general case.   
AOP makes it possible to modularize and separate these services and then apply them declaratively to the components and we can focus on our own specific concerns.   
Transaction parameters are like EJB   
PROPAGATION\_REQUIRED   
PROPAGATION\_REQUIRED\_NEW etc

What is RowCallbackHandler ?

The RowCallbackHandler interface extracts values from each row of a ResultSet.   
  
Has one method ? processRow(ResultSet)   
Called for each row in ResultSet.

### **What is PreparedStatementCreator ?**

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

### **What are Bean scopes in Spring Framework ?**

### **If you are using ApplicationContext then five scopes (singleton,prototype,request,session,globalsession) and if you are using BeanFactory then two(singleton,prototype); singleton : Scopes a single bean definition to a single object instance per Spring IoC container. Only one singleton object will be created. Default scope. prototype: Scopes a single bean definition to any number of object instances. Every call create new object like Emp emp = new Emp(); request: Scopes a single bean definition to the lifecycle of a single HTTP request; that is each and every 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. Lifecycle interfaces in spring ?**

1) InitializingBean   
2) DisposableBean   
  
1) InitializingBean   
<bean id="exampleInitBean" class="examples.ExampleBean" init-method="init"/>   
public class ExampleBean {   
public void init() { // do some initialization work }   
}   
OR   
<bean id="exampleInitBean" class="examples.AnotherExampleBean"/>   
public class AnotherExampleBean implements InitializingBean {   
public void afterPropertiesSet() { // do some initialization work }   
}   
  
2) DisposableBean   
<bean id="exampleInitBean" class="examples.ExampleBean" destroy-method="cleanup"/>   
public class ExampleBean {   
public void cleanup() {   
// do some destruction work (like releasing pooled connections)   
} }   
OR   
<bean id="exampleInitBean" class="examples.AnotherExampleBean"/>   
public class AnotherExampleBean implements DisposableBean {   
public void destroy() { // do some destruction work (like releasing pooled connections) }   
}

### **What are the common implementations of the Application Context ? \* ClassPathXmlApplicationContext : It Loads context definition from an XML file located in the classpath ApplicationContext context = new ClassPathXmlApplicationContext("bean\_test.xml"); \*FileSystemXmlApplicationContext : It loads context definition from an XML file in the filesystem. ApplicationContext context = new FileSystemXmlApplicationContext("bean\_test.xml"); XmlWebApplicationContext : It loads context definition from an XML file contained within a web application. Q.What is the difference between Bean Factory and Application Context ?**

### **\*The ApplicationContext builds on top of the BeanFactory (it's a subclass) and adds other functionality such as easier integration with Springs AOP features, message resource handling (for use in internationalization), event propagation, declarative mechanisms to create the ApplicationContext and optional parent contexts, and application-layer specific contexts such as the WebApplicationContext, among other enhancements. \*Application Context is suitabe for J2EE Applications. \*The main usage scenario when you might prefer to use the BeanFactory is when memory usage is the greatest concern (such as in an applet where every last kilobyte counts), and you don't need all the features of the ApplicationContext.**

### **Q.What is Application Context in Spring?**

The ApplicationContext builds on top of the BeanFactory (it's a subclass) and adds other functionality such as easier integration with Springs AOP features, message resource handling (for use in internationalization), event propagation, declarative mechanisms to create the ApplicationContext and optional parent contexts, and application-layer specific contexts such as the WebApplicationContext, among other enhancements.   
  
the BeanFactory provides the configuration framework and basic functionality, while the ApplicationContext adds enhanced capabilities to it, some of them perhaps more J2EE and enterprise-centric. In general, an ApplicationContext is a complete superset of a BeanFactory, and any description of BeanFactory capabilities and behavior should be considered to apply to ApplicationContexts as well.   
  
in a J2EE-environment, the best option is to use the ApplicationContext, since it offers all the features of the BeanFactory and adds on to it in terms of features, while also allowing a more declarative approach to use of some functionality, which is generally desirable.

### **What is BeanFactory and how can configure BeanFactory ?**

The BeanFactory is the actual container which instantiates, configures, and manages a number of beans. These beans typically collaborate with one another, and thus have dependencies between themselves. These dependencies are reflected in the configuration data used by the BeanFactory.   
  
Three ways you can configure BeanFactory.   
Resource res = new FileSystemResource("beans.xml");   
XmlBeanFactory factory = new XmlBeanFactory(res);   
or   
  
ClassPathResource res = new ClassPathResource("beans.xml");   
XmlBeanFactory factory = new XmlBeanFactory(res);   
or   
  
ClassPathXmlApplicationContext appContext = new ClassPathXmlApplicationContext(   
new String[] {"applicationContext.xml", "applicationContext-part2.xml"});   
// of course, an ApplicationContext is just a BeanFactory   
BeanFactory factory = (BeanFactory) appContext;

### **What are the types of Dependency Injection Spring supports with Example?**

  
Setter Injection and Constructor Injection .   
  
The two major flavors of Dependency Injection are Setter Injection (injection via JavaBean setters); and Constructor Injection (injection via constructor arguments). Spring provides sophisticated support for both, and even allows you to mix the two when configuring the one object.   
  
Setter Injection Example :   
XML ----   
<bean id="createCreditCard" class="springexample.creditcardaccount.CreateCreditCardAccount">   
<property name="smsInterface">   
            <ref bean="sms" />   
        </property>   
        <property name="daoInterface">   
            <ref bean="dao" />   
        </property>   
</bean>   
Java Class : CreateCreditCardAccount : Setter and Getter   
public SMSInterface getSmsInterface() {   
        return smsInterface;   
    }   
    /\*\*   
     \* @param smsInterface The smsInterface to set.   
     \*/   
    public void setSmsInterface(SMSInterface smsInterface) {   
        this.smsInterface = smsInterface;   
    }   
    /\*\*   
     \* @return Returns the daoInterface.   
     \*/   
    public DAOInterface getDaoInterface() {   
        return daoInterface;   
    }   
    /\*\*   
     \* @param daoInterface The daoInterface to set.   
     \*/   
    public void setDaoInterface(DAOInterface daoInterface) {   
        this.daoInterface = daoInterface;   
    }   
  
  
  
Constructor Injection Example:   
<bean id="orderService"   
class="spring.OrderService">   
<constructor-arg>   
<ref bean="orderDAO">   
</constructor-arg>   
</bean>   
  
Java Class : OrderService   
  
public class OrderService{   
private OrderDAO orderDAO;   
public OrderService(OrderDAO orderDAO){   
this.orderDAO=orderDAO;

}   
}

### **What is Spring ? and What are the Benefits/Advantages of Using Spring Framework ?**

Spring is an open source framework created to address the complexity of enterprise application development.   
\*Lightweight container   
\*No App Server Dependent ? like EJB JNDI Calls   
\*Objects are created Lazily , Singleton - configuration   
\*Components can added Declaratively   
Initialization of properties is easy ? no need to read from properties file   
\*Declarative transaction, security and logging service - AOP   
application code is much easier to unit test   
\*With a Dependency Injection approach, dependencies are explicit, and evident in constructor or JavaBean properties   
\*Spring's configuration management services can be used in any architectural layer, in whatever runtime environment.   
\*Spring can effectively organize your middle tier objects   
\*not requires special deployment steps

### **What are the benefits of IOC - Inversion of control (Dependency Injection)?**

  
\*Loose coupling: Components can added Declaratively so we can add and remove the components with out code change.   
For Example :   
<bean id="createCreditCard" class="springexample.creditcardaccount.CreateCreditCardAccount">   
        <property name="emailInterface">   
            <ref bean="email" />   
        </property>   
        <property name="smsInterface">   
            <ref bean="sms" />   
        </property>   
        <property name="daoInterface">   
            <ref bean="dao" />   
        </property>   
                   
    </bean>   
  
There are three components associated with createCreditCard.   
Tommorow you don't want smsInterface so just remove the xml configuration.   
        <property name="smsInterface">   
            <ref bean="sms" />   
        </property>   
  
\*Not Required any singletons : Don't need to code for singleton class. Every class is by default singleton. you can make not singleton   
by making singleton="false"   
<bean id="sms" class="springexample.sms.SMS" singleton="false">   
</bean>   
  
\*Objects are created Lazily .   
  
\*Initialization of properties is easy ? no need to read from properties file.   
For Example:   
<bean id="email" class="springexample.email.Email" >   
        <property name="smtpHost">   
                <value>smtp.sa.com</value>   
        </property>   
</bean>   
IOC Contained set the setter method of Email class. setSmtpHost(). You don't nned to read from properties file with extra coding.   
  
\*No App Server Dependent ? like EJB JNDI Calls

### **What are the different types of IOC (dependency injection) ?**

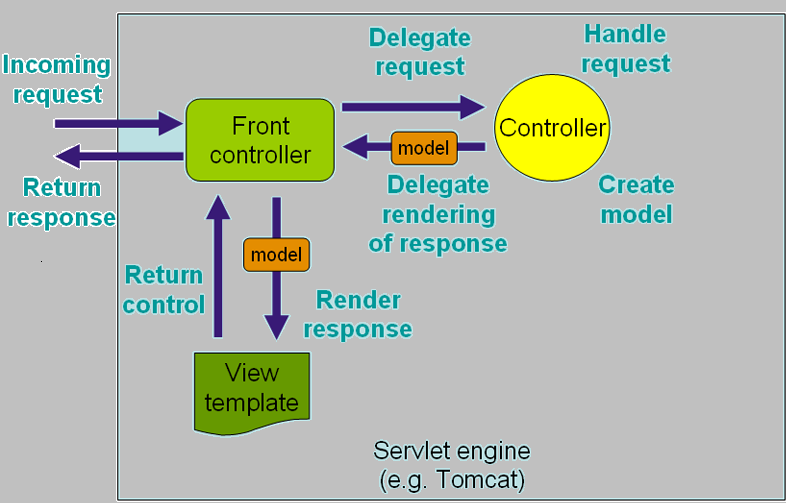
  
Spring container support   
Setter Injection and Constructor Injection .   
  
The two major flavors of Dependency Injection are Setter Injection (injection via JavaBean setters); and Constructor Injection (injection via constructor arguments). Spring provides sophisticated support for both, and even allows you to mix the two when configuring the one object.   
  
Interface Injection (e.g. Avalon): Injection is done through an interface. Spring does not support this.

### **;What is IOC - Inversion of Control ?**

  
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.   
For Example :   
In the XML file you declare   
<bean id="creditRating" class="springexample.creditrating.CreditRating">   
</bean>   
You don't need to create object for CreditRating.   
No nned to do new CreditRating(). Spring container will create the object for you.   
  
  
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.   
In a typical IOC scenario, the container creates all the objects, wires them together by setting the necessary properties, and determines when methods will be invoked.   
  
For Example :   
In the XML File   
<bean id="createCreditCard" class="springexample.creditcardaccount.CreateCreditCardAccount">   
        <property name="daoInterface">   
            <ref bean="dao" />   
        </property>   
                   
    </bean>   
  
Then Spring container will create the dependency relation. CreateCreditCardAccount class should have setter and getter for daoInterface.   
  
You don't need to create daoInterface object and set the setter method. Spring container will do for you.   
  
The two major flavors of Dependency Injection are Setter Injection (injection via JavaBean setters); and Constructor Injection (injection via constructor arguments). Spring provides sophisticated support for both, and even allows you to mix the two when configuring the one object

Spring MVC

In Spring's Web MVC framework : a DispatcherServlet that dispatches requests to handlers.The default handler is a very simple Controller interface, just offering a ModelAndView handleRequest(request,response) method.



Overview of the Spring Framework

### **DispatcherServlet**

DispatcherServlet a central servlet that dispatches requests to controllers and offers other functionality facilitating the development of web applications. DispatcherServlet completely integrated with the Spring IoC container and allows you to use every other feature that Spring has.   
The DispatcherServlet is an Servlet declared in the web.xml.  
Consider the following DispatcherServlet servlet configuration (in the web.xml file.)

|  |
| --- |
| <web-app> ... <servlet> <servlet-name>test</servlet-name> <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class> <load-on-startup>1</load-on-startup> </servlet> <servlet-mapping> <servlet-name>test</servlet-name> <url-pattern>\*.do</url-pattern> </servlet-mapping> </web-app> |

With the above servlet configuration , you will need to have a file called '/WEB-INF/**test**-servlet.xml' in your application; this file will contain all of your Spring Web MVC-specific components (beans).   
The WebApplicationContext is an extension of the plain ApplicationContext that has some extra features necessary for web applications  
The Spring DispatcherServlet has a couple of special beans it uses in order to be able to process requests and render the appropriate views. These beans are included in the Spring framework and can be configured in the WebApplicationContext

### **Controllers**

controller is the org.springframework.web.servlet.mvc.Controller interface, the source code for which is listed below. 

|  |
| --- |
| public interface Controller { /\*\* \* Process the request and return a ModelAndView object which the DispatcherServlet \* will render. \*/ ModelAndView handleRequest( HttpServletRequest request, HttpServletResponse response) throws Exception; } |

As you can see, the Controller interface defines a single method that is responsible for handling a request and returning an appropriate model and view.  
Command controllers provide a way to interact with data objects and dynamically bind parameters from the HttpServletRequest to the data object specified.  
**AbstractCommandController** : This class does not offer form functionality;  
**AbstractFormController** : an abstract controller offering form submission support.After a user has filled the form, the AbstractFormController binds the fields, validates the command object, and hands the object back to the controller to take the appropriate action.  
**SimpleFormController** : a form controller that provides even more support when creating a form with a corresponding command object. The SimpleFormController let's you specify a command object, a viewname for the form, a viewname for page you want to show the user when form submission has succeeded.  
**AbstractWizardFormController** : as the class name suggests, this is an abstract class - your wizard controller should extend it. This means you have to implement the validatePage(), processFinish() and processCancel() methods.

### **Handler mappings**

Using a handler mapping you can map incoming web requests to appropriate handlers. the *SimpleUrlHandlerMapping* or the *BeanNameUrlHandlerMapping* you can use. **BeanNameUrlHandlerMapping**: A very simple, but very powerful handler mapping is the BeanNameUrlHandlerMapping, which maps incoming HTTP requests to names of beans, defined in the web application context.  
<bean id="defaultHandlerMapping" class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"/>   
**SimpleUrlHandlerMapping**:powerful handler mapping - is the SimpleUrlHandlerMapping. This mapping is configurable in the application context  
<bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">  
<property name="mappings">  
<value>  
/\*/account.form=editAccountFormController  
/\*/editaccount.form=editAccountFormController  
/ex/view\*.html=helpController  
/\*\*/help.html=helpController  
</value>  
</property>  
</bean>

### **Views**

**Resolving views - the ViewResolver interface**: All controllers in the Spring Web MVC framework return a ModelAndView instance. Views in Spring are addressed by a view name and are resolved by a view resolver. Example :  
<bean id="viewResolver"  
class="org.springframework.web.servlet.view.UrlBasedViewResolver">  
<property name="prefix" value="/WEB-INF/jsp/"/>  
<property name="suffix" value=".jsp"/>  
</bean>  
When returning "successaccount" as a viewname, this view resolver will hand the request over to the RequestDispatcher that will send the request to /WEB-INF/jsp/successaccount.jsp.

Spring Web Flow:

Flow: a reusable sequence of steps that can execute in different contexts.

a flow consists of a series of steps called "states". Entering a state typically

results in a view being displayed to the user. On that view, user events occur that are handled by the state. These events can trigger transitions to other states which result in view navigations

Every flow begins with the following root element:

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

<flow xmlns="http://www.springframework.org/schema/webflow"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/webflow

http://www.springframework.org/schema/webflow/spring-webflow-2.0.xsd">

</flow>

All states of the flow are defined within this element. The first state defined becomes the flow's starting point.

**view-state**

Use the view-state element to define a step of the flow that renders a view:

<view-state id="enterBookingDetails" />

By convention, a view-state maps its id to a view template in the directory where the flow islocated. For example, the state above might render

/WEB-INF/hotels/booking/enterBookingDetails.xhtml if the flow itself was located in the /WEB-INF/hotels/booking dir

**transition**

Use the transition element to handle events that occur within a state:

<view-state id="enterBookingDetails">

<transition on="submit" to="reviewBooking" />

</view-state>

These transitions drive view navigations.

Use the end-state element to define a flow outcome:

<end-state id="bookingCancelled" />

When a flow transitions to a end-state it terminates and the outcome is returned.

Within a flow, there are several points where you can execute actions. These points are:

• On flow start

• On state entry

• On view render

• On transition execution

• On state exit

• On flow end

Actions are defined using a concise expression language. Spring Web Flow uses the Unified EL by default. The next few sections will cover the essential language elements for defining actions.

The action element you will use most often is the evaluate element. Use the evaluate

element to evaluate an expression at a point within your flow. With this single tag you can

invoke methods on Spring beans or any other flow variable. For example:

<evaluate expression="entityManager.persist(booking)" />

**Assigning an evaluate result**

If the expression returns a value, that value can be saved in the flow's data model called

flowScope:

<evaluate expression="bookingService.findHotels(searchCriteria)" result="flowScope.hotels" />

**Converting an evaluate result**

If the expression returns a value that may need to be converted, specify the desired type using the

result-type attribute:

<evaluate expression="bookingService.findHotels(searchCriteria)" result="flowScope.hotels"

result-type="dataModel"/>

**Checkpoint: flow actions**

Now review the sample booking flow with actions added:

<flow xmlns="http://www.springframework.org/schema/webflow"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/webflow

http://www.springframework.org/schema/webflow/spring-webflow-2.0.xsd">

<input name="hotelId" />

<on-start>

<evaluate expression="bookingService.createBooking(hotelId, currentUser.name)"

result="flowScope.booking" />

</on-start>

<view-state id="enterBookingDetails">

<transition on="submit" to="reviewBooking" />

</view-state>

<view-state id="reviewBooking">

<transition on="confirm" to="bookingConfirmed" />

<transition on="revise" to="enterBookingDetails" />

<transition on="cancel" to="bookingCancelled" />

</view-state>

<end-state id="bookingConfirmed" />

<end-state id="bookingCancelled" />

</flow>

This flow now creates a Booking object in flow scope when it starts