1. <#modules>
2. [Diff\_BeanFactory\_ApplicationContext](#Diff_BeanFactory_ApplicationContext)
3. [singleton\_prototype\_bean](#singleton_prototype_bean)
4. [spring\_ctxt\_springcontainer](#spring_ctxt_springcontainer)
5. [XMLBeanFactory](#XMLBeanFactory)
6. [AOP](#AOP)
7. [beanlifecycle](#beanlifecycle)
8. [beanscopes](#beanscopes)
9. [BeanAutowiring](#BeanAutowiring)
10. [autowired\_annotation](#autowired_annotation)
11. [diff\_component\_control\_respo\_serv\_annota](#diff_component_control_respo_serv_annota)
12. [Exception\_SpringMVC](#Exception_SpringMVC)
13. [ContextLoaderListener](#ContextLoaderListener)
14. <#designpatterns>
15. [Difference between applicationContext.xml and spring-servlet.xml in Spring](http://stackoverflow.com/questions/3652090/difference-between-applicationcontext-xml-and-spring-servlet-xml-in-spring)
16. [HOW SPRING WORKS-REFLECTION](#Reflection)

Additional spring intervieq questions index wise:

<http://www.journaldev.com/2696/spring-interview-questions-and-answers>

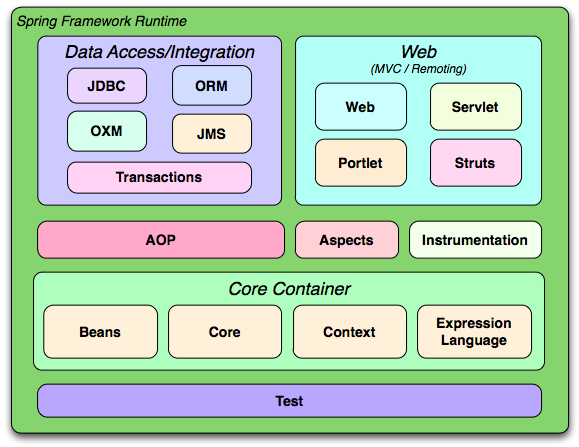
<http://docs.spring.io/spring-framework/docs/3.0.x/reference/overview.html>

The **Spring Framework** is an [open source](http://en.wikipedia.org/wiki/Open_source) [application framework](http://en.wikipedia.org/wiki/Application_framework) and [inversion of control](http://en.wikipedia.org/wiki/Inversion_of_control) container for the [Java platform](http://en.wikipedia.org/wiki/Java_platform). The framework's core features can be used by any Java application, but there are extensions for building web applications on top of the [Java EE](http://en.wikipedia.org/wiki/Java_EE) platform. Although the framework does not impose any specific [programming model](http://en.wikipedia.org/wiki/Programming_model), it has become popular in the Java community as an alternative to, replacement for, or even addition to the [Enterprise JavaBean](http://en.wikipedia.org/wiki/Enterprise_JavaBean) (EJB) model

## 1.2 Modules

The Spring Framework consists of features organized into about 20 modules. These modules are grouped into Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation, and Test, as shown in the following diagram.

* **Spring Core Container**: This is the base module of Spring and provides spring containers (BeanFactory and ApplicationContext).[[6]](http://en.wikipedia.org/wiki/Spring_Framework#cite_note-6)



Overview of the Spring Framework

### 1.2.1 Core Container

The [Core Container](http://docs.spring.io/spring-framework/docs/3.0.x/reference/beans.html#beans-introduction) consists of the Core, Beans, Context, and Expression Language modules.

The [Core and Beans](http://docs.spring.io/spring-framework/docs/3.0.x/reference/beans.html#beans-introduction) modules provide the fundamental parts of the framework, including the IoC and Dependency Injection features. The BeanFactory is a sophisticated implementation of the factory pattern. It removes the need for programmatic singletons and allows you to decouple the configuration and specification of dependencies from your actual program logic.

The [Context](http://docs.spring.io/spring-framework/docs/3.0.x/reference/beans.html#context-introduction) module builds on the solid base provided by the [Core and Beans](http://docs.spring.io/spring-framework/docs/3.0.x/reference/beans.html#beans-introduction) modules: it is a means to access objects in a framework-style manner that is similar to a JNDI registry. The Context module inherits its features from the Beans module and adds support for internationalization (using, for example, resource bundles), event-propagation, resource-loading, and the transparent creation of contexts by, for example, a servlet container. The Context module also supports Java EE features such as EJB, JMX ,and basic remoting. The ApplicationContext interface is the focal point of the Context module.

The [Expression Language](http://docs.spring.io/spring-framework/docs/3.0.x/reference/expressions.html) module provides a powerful expression language for querying and manipulating an object graph at runtime. It is an extension of the unified expression language (unified EL) as specified in the JSP 2.1 specification. The language supports setting and getting property values, property assignment, method invocation, accessing the context of arrays, collections and indexers, logical and arithmetic operators, named variables, and retrieval of objects by name from Spring's IoC container. It also supports list projection and selection as well as common list aggregations.

### 1.2.2 Data Access/Integration

The Data Access/Integration layer consists of the JDBC, ORM, OXM, JMS and Transaction modules.

The [JDBC](http://docs.spring.io/spring-framework/docs/3.0.x/reference/jdbc.html#jdbc-introduction) module provides a JDBC-abstraction layer that removes the need to do tedious JDBC coding and parsing of database-vendor specific error codes.

The [ORM](http://docs.spring.io/spring-framework/docs/3.0.x/reference/orm.html#orm-introduction) module provides integration layers for popular object-relational mapping APIs, including [JPA](http://docs.spring.io/spring-framework/docs/3.0.x/reference/orm.html#orm-jpa), [JDO](http://docs.spring.io/spring-framework/docs/3.0.x/reference/orm.html#orm-jdo), [Hibernate](http://docs.spring.io/spring-framework/docs/3.0.x/reference/orm.html#orm-hibernate), and [iBatis](http://docs.spring.io/spring-framework/docs/3.0.x/reference/orm.html" \l "orm-ibatis" \o "13.6 iBATIS SQL Maps). Using the ORM package you can use all of these O/R-mapping frameworks in combination with all of the other features Spring offers, such as the simple declarative transaction management feature mentioned previously.

The [OXM](http://docs.spring.io/spring-framework/docs/3.0.x/reference/oxm.html) module provides an abstraction layer that supports Object/XML mapping implementations for JAXB, Castor, XMLBeans, JiBX and XStream.

The Java Messaging Service ([JMS](http://docs.spring.io/spring-framework/docs/3.0.x/reference/jms.html)) module contains features for producing and consuming messages.

The [Transaction](http://docs.spring.io/spring-framework/docs/3.0.x/reference/transaction.html) module supports programmatic and declarative transaction management for classes that implement special interfaces and for all your POJOs (plain old Java objects).

### 1.2.3 Web

The Web layer consists of the Web, Web-Servlet, Web-Struts, and Web-Portlet modules.

Spring's Web module provides basic web-oriented integration features such as multipart file-upload functionality and the initialization of the IoC container using servlet listeners and a web-oriented application context. It also contains the web-related parts of Spring's remoting support.

The Web-Servlet module contains Spring's model-view-controller ([MVC](http://docs.spring.io/spring-framework/docs/3.0.x/reference/mvc.html#mvc-introduction)) implementation for web applications. Spring's MVC framework provides a clean separation between domain model code and web forms, and integrates with all the other features of the Spring Framework.

The Web-Struts module contains the support classes for integrating a classic Struts web tier within a Spring application. Note that this support is now deprecated as of Spring 3.0. Consider migrating your application to Struts 2.0 and its Spring integration or to a Spring MVC solution.

The Web-Portlet module provides the MVC implementation to be used in a portlet environment and mirrors the functionality of Web-Servlet module.

# [What is the difference between a Spring application context and a Spring container?](http://stackoverflow.com/questions/10303413/what-is-the-difference-between-a-spring-application-context-and-a-spring-contain)

[**http://stackoverflow.com/questions/10303413/what-is-the-difference-between-a-spring-application-context-and-a-spring-contain**](http://stackoverflow.com/questions/10303413/what-is-the-difference-between-a-spring-application-context-and-a-spring-contain)

There are two types of IoC containers. They are:

1. **BeanFactory interface**
2. **ApplicationContext** interface (

The **XmlBeanFactory** is the implementation class for the BeanFactory interface. To use the BeanFactory, we need to create the instance of XmlBeanFactory class as given below:

Resource resource=**new** ClassPathResource("applicationContext.xml");

BeanFactory factory=**new** XmlBeanFactory(resource);

The **ClassPathXmlApplicationContext** class is the implementation class of **ApplicationContext** interface. We need to instantiate the **ClassPathXmlApplicationContext** class to use the ApplicationContext as given below:

ApplicationContext context =

    new ClassPathXmlApplicationContext("applicationContext.xml");

### Name some of the important Spring Modules?

Some of the important Spring Framework modules are:

* + **Spring Context** – for dependency injection.
  + **Spring AOP** – for aspect oriented programming.
  + **Spring DAO** – for database operations using DAO pattern
  + **Spring JDBC** – for JDBC and DataSource support.
  + **Spring ORM** – for ORM tools support such as Hibernate
  + **Spring Web Module** – for creating web applications.
  + **Spring MVC** – Model-View-Controller implementation for creating web applications, web services etc.

**Spring framework interview questions** is in rise on J2EE and core Java interviews,  As Spring is the best framework available for Java application development and now *Spring IOC container* and Spring MVC framework are used as de-facto framework for all new Java development. With this popularity interview questions from spring framework is top on any list of  [core Java Interview questions](http://javarevisited.blogspot.sg/2011/04/top-20-core-java-interview-questions.html). I thought to put together some spring interview questions and answers which has appeared on many Java and J2EE interviews and useful for practicing before appearing on any Java Job interview. This list of Spring interview questions and answers contains questions from Spring fundamentals e.g. Spring IOC and [dependency Injection](http://javarevisited.blogspot.sg/2012/06/20-design-pattern-and-software-design.html), **Spring MVC framework**, Spring Security, Spring AOP etc, because of length of this post I haven't included Spring interview questions from Spring JDBC and JMS which is also a popular topic in core Java and [J2EE interviews](http://javarevisited.blogspot.sg/2011/09/servlet-interview-questions-answers.html). I suggest to prepare those as well. Any way these Spring questions are not very difficult and based on fundamentals e.g. [What is default scope of Spring bean](http://javarevisited.blogspot.sg/2012/05/what-is-bean-scope-in-spring-mvc.html) and mostly asked during first round or telephonic round of Java interview. Although you can find answers of these **Spring interview questions** by doing Google but I have also included some answers for quick reference. As I said Spring  and Spring MVC is fantastic Java framework and if you are not using it than start using it, these questions will give you some head start as well.

Difference between Spring Context and Spring core.

These are actually 2 of many Spring Framework modules. You can easily find what packages these artifacts contain, for example using this site:

<http://mvnrepository.com/artifact/org.springframework/spring-core/3.1.1.RELEASE>

This can tell you something general about classes contained within an artifact and usually about purpose of the artifact.

In context of Spring Framework, spring-core contains mainly some core utilities and commons stuff (like common enums) and probably all Spring modules (of most of them) depend on it (directly of transitively). In turn spring-context provides Application Context, that is Spring's Dependency Injection Container and it is probably always defined in POMs of artifacts that use Spring (in general). In fact, spring-context depends on spring-core so by defining spring-context as your dependency, you use spring-core anyway.

Top Spring Interview Questions and Answers

[spring interview questions and answers J2EE](http://javarevisited.blogspot.com/2011/09/generics-java-example-tutorial.html)Now let’s start with questions, these **Spring interview Questions** are not very tricky or tough and based upon primary concepts of spring framework. If you are developing application using Spring framework than you may be, already familiar with many of these Java and Spring interview questions and answers. Nevertheless it’s a good recap before appearing to any Spring and Java interview.  
  
  
**Spring Security Interview questions Answer**   
Some of the reader requested to provide Spring Security interview questions and answer, So i though to update this article with few of Spring security question I came across. Here are those:  
  
**How do you setup LDAP Authentication using Spring Security?**  
This is a very popular Spring Security interview question as Spring provides out of the box support to connect Windows Active directory for LDAP authentication and with few configuration in Spring config file you can have this feature enable. See [How to perform LDAP authentication in Java using Spring Security](http://javarevisited.blogspot.sg/2011/11/ldap-authentication-active-directory.html) for detailed code explanation and sample.

Steps:

In Summary now LDAP login will be done on these step

1) Your Service or application bind itself with LDAP using manager-dn and manager-password.

2) LDAP search for user to find UserDn

3) LDAP bind using UserDn

spring-ldap-1.3.1.RELEASE-all.jar and spring-security-ldap-3.1.0.RC3.jar

Read more: <http://javarevisited.blogspot.com/2011/11/ldap-authentication-active-directory.html#ixzz3BLyFYgYo>

**How do you control concurrent Active session using Spring Security?**  
Another Spring interview question which is based upon Out of box feature provided by Spring framework. **You can easily control How many active session a user can have with a Java application by using Spring Security.** See this spring security example of [how to control concurrent session in Java and Spring](http://javarevisited.blogspot.sg/2012/03/spring-security-example-tutorial-how-to.html) for exact details.

As I said it’s simple and easy when you use spring security framework or library. In fact is all declarative and no code is require to enable **concurrent session disable functionality**. You will need to include following xml snippet in your *Spring Security Configuration file* mostly named as applicaContext-security.xml. Here is sample **spring security Example** of limiting user session in Java web application:

**<session-management** invalid-session-url="/logout.html"**>**  
    **<concurrency-control** max-sessions="1" error-if-maximum-exceeded="true" **/>**  
**</session-management>**

Read more: <http://javarevisited.blogspot.com/2012/03/spring-security-example-tutorial-how-to.html#ixzz3BLyxeMza>

**Question1: What is IOC or inversion of control?**

Answer: This *Spring interview question* is first step towards Spring framework and many interviewer starts Spring interview from this question. As the name implies **Inversion of control** means now we have inverted the control of creating the object from our own using new operator to container or framework. Now it’s the responsibility of container to create object as required. We maintain one xml file where we configure our components, services, all the classes and their property. We just need to mention which service is needed by which component and container will create the object for us. This concept is known as **dependency injection** because all object dependency (resources) is injected into it by framework.

Example:

  <bean id="createNewStock" class="springexample.stockMarket.CreateNewStockAccont">   
        <property name="newBid"/>

  </bean>

In this example CreateNewStockAccont class contain getter and setter for newBid and container will instantiate newBid and set the value automatically when it is used. This whole process is also called wiring in Spring and by using annotation it can be done automatically by Spring, refereed as auto-wiring of bean in Spring.

**Question 2: Explain Bean-LifeCycle.**

Ans: Spring framework is based on IOC so we call it as IOC container also So Spring beans reside inside the IOC container. Spring beans are nothing but Plain old java object (POJO).

Following steps explain their life cycle inside container.

1. Container will look the bean definition inside configuration file (e.g. bean.xml).

2 using reflection container will create the object and if any property is defined inside the bean definition then it will also be set.

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 before the properties for the Bean are set.

6. If an init() method is specified for the bean, it will be called.  
7. If the Bean class implements the DisposableBean interface, then the method destroy() will be called when the Application no longer needs the bean reference.

8. If the Bean definition in the Configuration file contains a 'destroy-method' attribute, then the corresponding method definition in the Bean class will be called.

**Question 3: what is Bean Factory, have you used XMLBeanFactory?**

Ans:

BeanFactory is factory Pattern which is based on IOC [design principles](http://javarevisited.blogspot.sg/2012/03/10-object-oriented-design-principles.html). It is used to make a clear separation between application configuration and dependency from actual code.

XmlBeanFactory is one of the implementation of bean Factory which we have used in our project.

**org.springframework.beans.factory.xml.XmlBeanFactory is used to create bean instance defined in our xml file.**

BeanFactory factory = new XmlBeanFactory(new FileInputStream("beans.xml"));

Or

ClassPathResource resorce = new ClassPathResource("beans.xml");   
XmlBeanFactory factory = new XmlBeanFactory(resorce);

**Question 4: What are the difference between BeanFactory and ApplicationContext in spring?**

Answer : Spring container has two interfaces : 1) Bean Factory 2) ApplicationContext. This one is very popular spring interview question and often asks in entry level interview. ApplicationContext is preferred way of using spring because of functionality provided by it and interviewer wanted to check whether you are familiar with it or not.

|  |  |
| --- | --- |
| **ApplicationContext.** | **BeanFactory** |
| Here we can have more than one config files possible | In this only one config file or .xml file |
| Application contexts can publish events to beans that are registered as listeners | Doesn’t support. |
| Support internationalization (I18N) messages | It’s not |
| Support application life-cycle events, and validation. | Doesn’t support. |
| Support  many enterprise services such JNDI access, EJB integration, remoting | Doesn’t support. |

The *ApplicationContext* includes all functionality of the *BeanFactory*, it is generally recommended over the *BeanFactory*. BeanFactory can still be used for light weight applications like mobile devices or applet based applications.

The most commonly used ApplicationContext implementations are:

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

**Question 5: What are different modules in spring?**

Answer : spring have seven core modules

1.      The Core container module

2.      Application context module

3.      AOP module (Aspect Oriented Programming)

4.      JDBC abstraction and DAO module

5.      O/R mapping integration module (Object/Relational)

6.      Web module

7.      MVC framework module

a

**Question 6: What is difference between singleton and prototype bean?**

Ans: This is another popular *spring interview questions* and an important concept to understand. Basically a bean has scopes which defines their existence on the application

**Singleton:** means single bean definition to a single object instance per Spring IOC container.  
**Prototype**: means a single bean definition to any number of object instances.

Whatever beans we 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.**

  <bean id="createNewStock"     class="springexample.stockMarket.CreateNewStockAccont" **singleton=”false”**>   
        <property name="newBid"/>   
  </bean>

**Question 7: What type of transaction Management Spring support?**

Ans: This spring interview questions is little difficult as compared to previous questions just because **transaction management** is a complex concept and not every developer familiar with it. Transaction management is critical in any applications that will interact with the database. The application has to ensure that the data is consistent and the integrity of the data is maintained.  Two type of transaction management is supported by spring

1. Programmatic transaction management

2. Declarative transaction management.

**Question 8:** **What is AOP?**

Answer : The core construct of AOP is the aspect, which encapsulates behaviors affecting multiple classes into reusable modules. AOP is a programming technique that allows developer to modularize crosscutting concerns,  that cuts across the typical divisions of responsibility, such as **logging and transaction management.** Spring AOP, aspects are implemented using regular classes or regular classes annotated with the @Aspect annotation

**Question 9: Explain Advice?**

Answer: It’s an implementation of aspect; advice is inserted into an application at join points. Different types of advice include “around,” “before” and “after” advice

**Question 10: What is joint Point and point cut?**

Ans: This is not really a spring interview questions I would say an AOP one.  Similar to [Object oriented programming](http://javarevisited.blogspot.sg/2012/03/10-object-oriented-design-principles.html), AOP is another popular programming concept which complements OOPS. Join point is an opportunity within code for which we can apply an aspect. In Spring AOP, a join point always represents a method execution.

**Pointcut**: a predicate that matches join points. A point cut is something that defines at what join-points an advice should be applied

These spring interview Questions and answers are not very difficult and focused on spring fundamentals rather than focusing on advanced feature of session management, spring security, authentication etc. we will cover of those question on some other interview article. I would also suggest that share some spring questions asked to you guys during interview and than I can put together those with there answers for quick reference of everybody

### What are different scopes of Spring Bean?

There are five scopes defined for Spring Beans.

1. **singleton**: Only one instance of the bean will be created for each container. This is the default scope for the spring beans. While using this scope, make sure spring bean doesn’t have shared instance variables otherwise it might lead to data inconsistency issues because it’s not thread-safe.
2. **prototype**: A new instance will be created every time the bean is requested.
3. **request**: This is same as prototype scope, however it’s meant to be used for web applications. A new instance of the bean will be created for each HTTP request.
4. **session**: A new bean will be created for each HTTP session by the container.
5. **global-session**: This is used to create global session beans for Portlet applications.

Request Scope :-say a web request comes from client, In the same request we are creating the bean with statement factory.getBean("MyBean1") will it return two different instances. My understanding is it will return the same instance as it is done under the same http request. Right? While it will return the two different instances in case of prototype request. Correct?

Correct.

Prototype:- It is results in the creation of a new bean instance every time a request for that specific bean is made. As a rule of thumb, you should use the prototype scope for all beans that are stateful, while the singleton scope should be used for stateless beans. How come its different from request scope and how its support for stateful beans.

### What is Bean wiring and @Autowired annotation?

The process of injection spring bean dependencies while initializing it called Spring Bean Wiring.

Usually it’s best practice to do the explicit wiring of all the bean dependencies, but spring framework also supports autowiring. We can use @Autowired annotation with fields or methods for **autowiring byType**. For this annotation to work, we also need to enable annotation based configuration in spring bean configuration file. This can be done by **context:annotation-config** element.

For more details about **@Autowired** annotation, please read [Spring Autowire Example](http://www.journaldev.com/2623/spring-bean-autowire-by-name-type-constructor-autowired-and-qualifier-annotations-example).

### What are different types of Spring Bean autowiring?

There are four types of autowiring in Spring framework.

* 1. **autowire byName**
  2. **autowire byType**
  3. **autowire by constructor**
  4. autowiring by **@Autowired** and **@Qualifier** annotations

Prior to Spring 3.1, **autowire by autodetect** was also supported that was similar to autowire by constructor or byType. For more details about these options, please read [Spring Bean Autowiring](http://www.journaldev.com/2623/spring-bean-autowire-by-name-type-constructor-autowired-and-qualifier-annotations-example)

### What is Bean wiring and @Autowired annotation?

The process of injection spring bean dependencies while initializing it called Spring Bean Wiring.

Usually it’s best practice to do the explicit wiring of all the bean dependencies, but spring framework also supports autowiring. We can use @Autowired annotation with fields or methods for **autowiring byType**. For this annotation to work, we also need to enable annotation based configuration in spring bean configuration file. This can be done by **context:annotation-config** element.

For more details about @Autowired annotation, please read [Spring Autowire Example](http://www.journaldev.com/2623/spring-bean-autowire-by-name-type-constructor-autowired-and-qualifier-annotations-example).

### What’s the difference between @Component, @Controller, @Repository & @Service annotations in Spring?

**@Component** is used to indicate that a class is a component. These classes are used for auto detection and configured as bean, when annotation based configurations are used.

**@Controller** is a specific type of component, used in MVC applications and mostly used with RequestMapping annotation.

**@Repository** retrieve/search data. We can apply this annotation with DAO pattern implementation classes.

**@Service** is used to indicate that a class is a Service. Usually the business facade classes that provide some services are annotated with this.

annotation is used to indicate that a component is used as repository and a mechanism to store/

**Spring MVC Framework provides following ways to help us achieving robust exception handling.**

1. **Controller Based** – We can define exception handler methods in our controller classes. All we need is to annotate these methods with **@ExceptionHandler** annotation.
2. **Global Exception Handler** – Exception Handling is a cross-cutting concern and Spring provides @**ControllerAdvice** annotation that we can use with any class to define our global exception handler.
3. **HandlerExceptionResolver implementation** – For generic exceptions, most of the times we serve static pages. Spring Framework provides HandlerExceptionResolver interface that we can implement to create global exception handler. The reason behind this additional way to define global exception handler is that Spring framework also provides default implementation classes that we can define in our spring bean configuration file to get spring framework exception handling benefits.

For a complete example, please read [Spring Exception Handling Example](http://www.journaldev.com/2651/spring-mvc-exception-handling-exceptionhandler-controlleradvice-handlerexceptionresolver-json-response-example).

@ExceptionHandler(EmployeeNotFoundException.class)

    public ModelAndView handleEmployeeNotFoundException(HttpServletRequest request, Exception ex){

        logger.error("Requested URL="+request.getRequestURL());

        logger.error("Exception Raised="+ex);

        ModelAndView modelAndView = new ModelAndView();

        modelAndView.addObject("exception", ex);

        modelAndView.addObject("url", request.getRequestURL());

        modelAndView.setViewName("error");

        return modelAndView;

    }

}

### What is ContextLoaderListener?

**ContextLoaderListener** is the listener class used to load root context and define spring bean configurations that will be visible to all other contexts. It’s configured in **web.xml** file as:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <context-param>      <param-name>contextConfigLocation</param-name>      <param-value>/WEB-INF/spring/root-context.xml</param-value>  </context-param>    <listener>      <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener> |

### Name some of the design patterns used in Spring Framework?(NTTDAta,2014)

Spring Framework is using a lot of design patterns, some of the common ones are:

* 1. [Singleton Pattern](http://www.journaldev.com/1377/java-singleton-design-pattern-best-practices-with-examples): Creating beans with default scope.
  2. [Factory Pattern](http://www.journaldev.com/1392/factory-design-pattern-in-java): Bean Factory classes
  3. [Prototype Pattern](http://www.journaldev.com/1440/prototype-pattern-in-java): Bean scopes
  4. [Adapter Pattern](http://www.journaldev.com/1487/adapter-design-pattern-in-java-example-tutorial): Spring Web and Spring MVC
  5. [Proxy Pattern](http://www.journaldev.com/1572/proxy-design-pattern-in-java-example-tutorial): Spring Aspect Oriented Programming support
  6. [Template Method Pattern](http://www.journaldev.com/1763/template-method-design-pattern-in-java): JdbcTemplate, HibernateTemplate etc
  7. Front Controller: Spring MVC DispatcherServlet
  8. Data Access Object: Spring DAO support
  9. Dependency Injection and Aspect Oriented Programming

J:Observer pattern: used for Spring MVC –internalViewResolver.

# [Difference between applicationContext.xml and spring-servlet.xml in Spring](http://stackoverflow.com/questions/3652090/difference-between-applicationcontext-xml-and-spring-servlet-xml-in-spring)

Spring lets you define multiple contexts in a parent-child hierarchy.

The applicationContext.xml defines the beans for the "root webapp context", i.e. the context associated with the webapp.

The spring-servlet.xml (or whatever else you call it) defines the beans for one servlet's app context. There can be many of these in a webapp, one per Spring servlet (e.g. spring1-servlet.xml for servlet spring1, spring2-servlet.xml for servlet spring2).

Beans in spring-servlet.xml can reference beans in applicationContext.xml, but not vice versa.

All Spring MVC controllers must go in the spring-servlet.xml context.

In most simple cases, the applicationContext.xml context is unnecessary. It is generally used to contain beans that are shared between all servlets in a webapp. If you only have one servlet, then there's not really much point, unless you have a specific use for it.

# [How reflection is used in framework? [closed]](http://stackoverflow.com/questions/20817425/how-reflection-is-used-in-framework)

For example, Spring uses <bean> definitions as such

<bean id="someId" class="com.foopack.Foo">

<property name="someField" value="someValue" />

</bean>

When the Spring context processes this <bean> element, it will use Class.forName(String) with the argument as com.foopack.Foo to instantiate that Class (Class#newInstance() or get a Constructor, depending). It will then again use reflection to get the appropriate setter for the <property> element and set its value to the specified value.

Junit also uses reflection to get a set of @Test annotated methods to invoke. To do this, it needs to get a Class instance.

Servlet based web applications also use reflection to instantiate Servlet, Filter, and the different types of Listeners classes. For example, you would have

<servlet>

<servlet-name>YourServlet</servlet-name>

<servlet-class>com.servlets.YourServlet</servlet-class>

<servlet>

and the container would take that fully qualified class name, com.servlets.YourServlet, and instantiate and register it.

JSON parser/generator libraries also use reflection. For example, with Gson, given a class like

class Foo {

private String name = "FOOOO";

}

and an instance like this

Foo foo = new Foo();

you would serialize it like so

Gson gson = new Gson();

String json = gson.toJson(foo);

Gson would then call getClass() on the instance foo, get a set of the Field instances of that Class, iterate over the set, and serialize the values of the fields to a JSON format.